MAXQ7665/MAXQ7666 USER'S GUIDE

48-TQFN
7mm x 7mm
-40°C to +125°C

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.
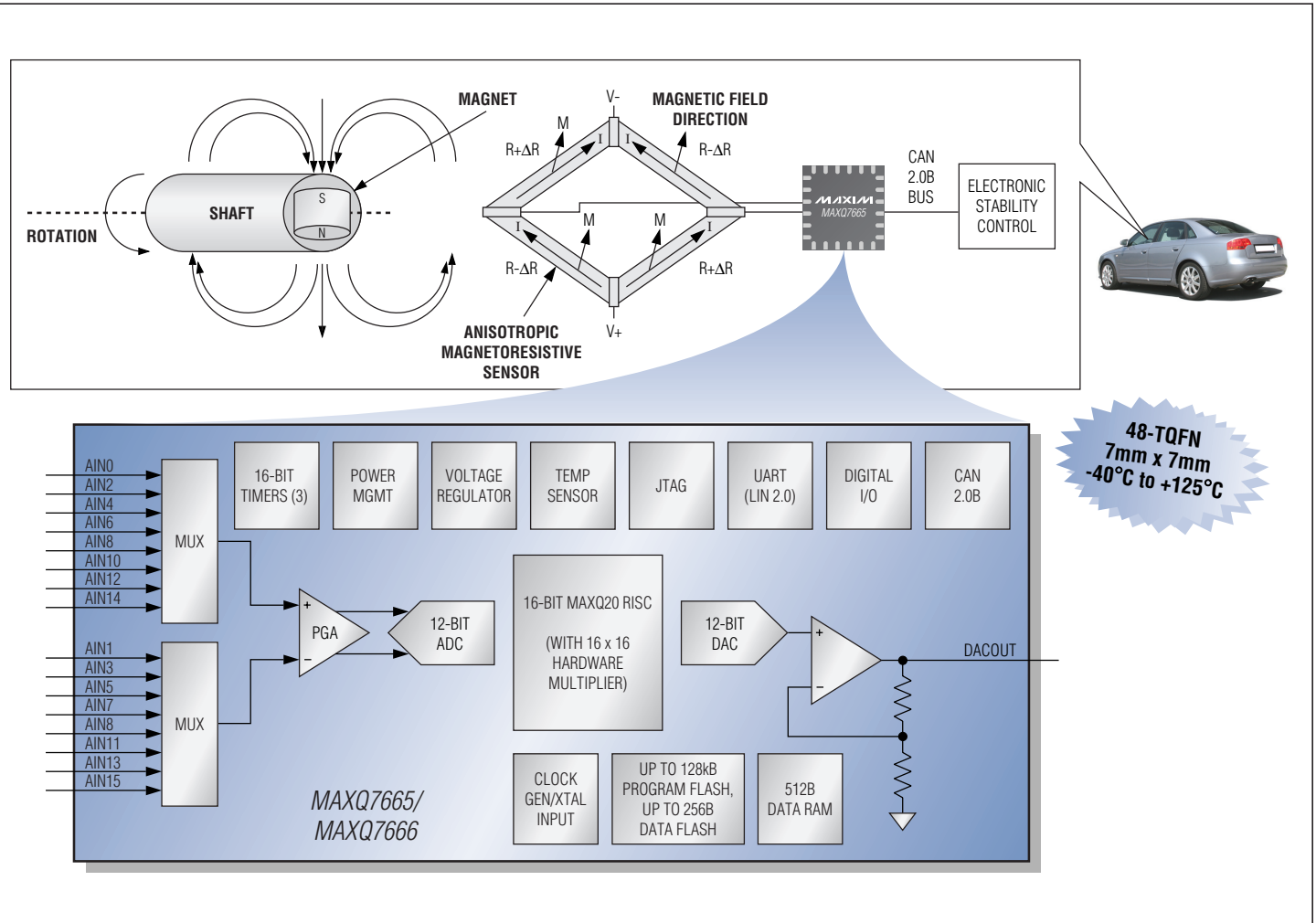
Rev 0; 12/07

For the modulo increment or decrement operation, the selected range of bits in AP are incremented or decremented. However, if these bits roll over or under, they simply wrap around without affecting the remaining bits in the accumulator pointer. So, the operations can be defined as follows:

- Increment modulo 2: AP = AP[3:1] + ((AP[0] + 1) mod 2)
- Decrement modulo 2: AP = AP[3:1] + ((AP[0] - 1) mod 2)
- Increment modulo 4: AP = AP[3:2] + ((AP[1:0] + 1) mod 4)
- Decrement modulo 4: AP = AP[3:2] + ((AP[1:0] - 1) mod 4)
- Increment modulo 8: AP = AP[3] + ((AP[2:0] + 1) mod 8)
- Decrement modulo 8: AP = AP[3] + ((AP[2:0] - 1) mod 8)
- Increment modulo 16: AP = (AP + 1) mod 16
- Decrement modulo 16:   AP = (AP - 1) mod 16

For this example, assume that all 16 accumulator registers are initially set to zero.

```
        move  AP, #02h           ; select A[ 2] as active accumulator
        move  APC, #02h          ; auto-increment AP[ 1:0] modulo 4
                                 ; AP   A[ 0]   A[ 1]   A[ 2]   A[ 3]
                                 ; 02   0000    0000    0000    0000
        add   #01h               ; 03   0000    0000    0001    0000
        add   #02h               ; 00   0000    0000    0001    0002
        add   #03h               ; 01   0003    0000    0001    0002
        add   #04h               ; 02   0003    0004    0001    0002
        add   #05h               ; 03   0003    0004    0006    0002
```

1.3.5.3 ALU Operations Using the Active Accumulator and a Source

The following arithmetic and logical operations can use any register or immediate value as a source. The active accumulator Acc is always used as the second operand and the implicit destination. Also, Acc may not be used as the source for any of these operations.

```
        add   A[ 4]              ; Acc = Acc + A[ 4]
        addc  #32h               ; Acc = Acc + 0032h + Carry

        sub   A[ 15]             ; Acc = Acc − A[ 15]
        subb  A[ 1]              ; Acc = Acc − A[ 1] - Carry
        cmp   #00h               ; If (Acc == 0000h), set Equals flag

        and   A[ 0]              ; Acc = Acc AND A[ 0]
        or    #55h               ; Acc = Acc OR #0055h

        xor   A[ 1]              ; Acc = Acc XOR A[ 1]
```

SECTION 1: MAXQ7665/MAXQ7666 CORE ARCHITECTURE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

MAXQ7665/MAXQ7666 User's Guide

SECTION 1: MAXQ7665/MAXQ7666 CORE ARCHITECTURE

1.1 Overview

The MAXQ7665/MAXQ7666 are low-power, high-performance, 16-bit RISC microcontrollers based on the MAXQ® architecture. They include support for integrated, in-system-programmable flash memory and a wide range of peripherals including a 12-bit 500ksps SAR ADC with a programmable gain amplifier (PGA) and a full CAN 2.0B controller supporting transfer rates up to 1Mbps. The MAXQ7665/MAXQ7666 are ideally suited for low-cost, low-power embedded applications such as automotive, industrial controls, and building automation. Except where explicitly noted, the MAXQ7665 and MAXQ7666 features are identical.

The MAXQ7665/MAXQ7666 key features include:

- 8MHz, 16-bit, single-cycle RISC CPU with Harvard Memory Architecture

- Up to 64k x 16 (128kB) on-chip program flash (16kB program flash and dedicated 256B data flash in MAXQ7666) and 512 bytes internal RAM

- High-precision, low-power analog input/output module including a 12-bit, 500ksps SAR ADC, 1x–32x PGA, 12-bit DAC and local/remote temperature sensor

- Full CAN 2.0B controller supporting transfer rates up to 1Mbps

- High-performance timer/digital I/O peripherals

- Flexible crystal/clock module

- Advanced power monitoring/management module
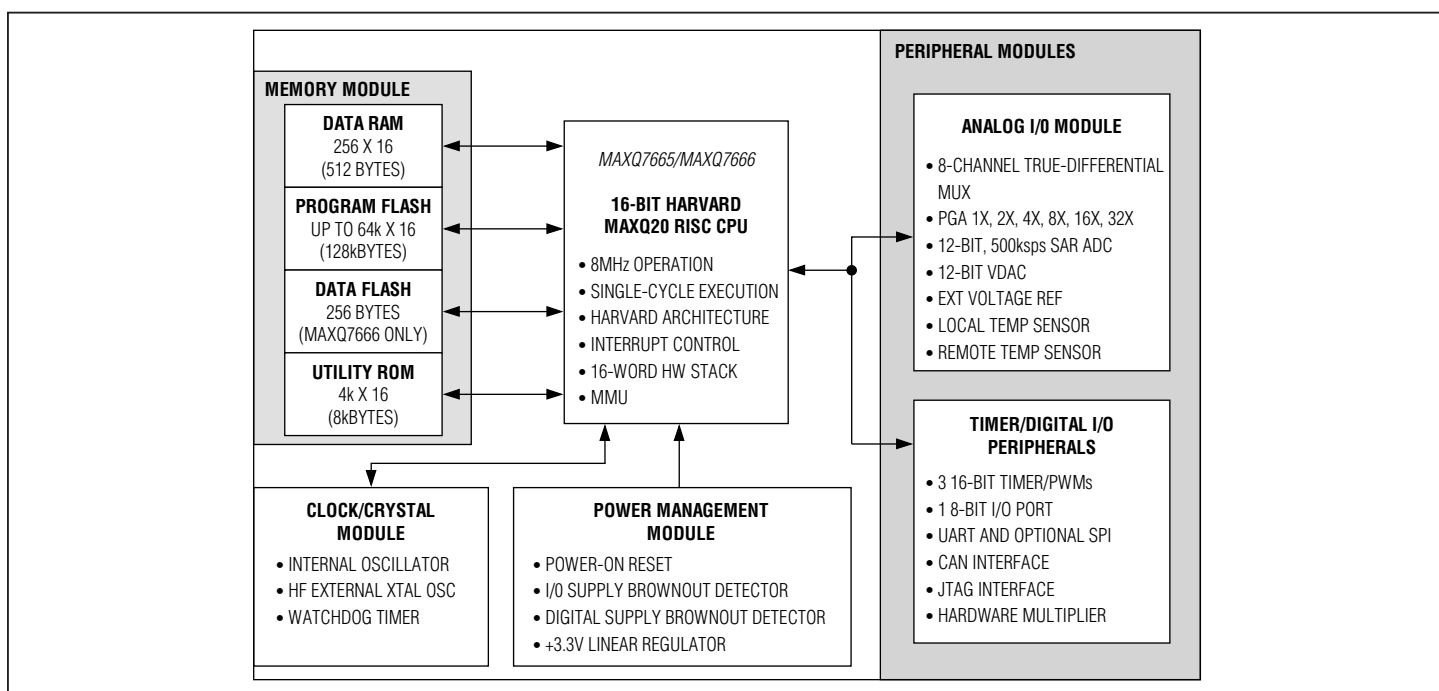
Figure 1-1. MAXQ7665/MAXQ7666 Block Diagram

1.1.1 References

The online MAXQ7665 and MAXQ7666 QuickView pages contain additional information and links to the data sheet. Errata sheets for the MAXQ products are available at **www.maxim-ic.com/errata**. For more information on other MAXQ microcontrollers, development hardware and software, frequently asked questions and software examples, visit the MAXQ home page at **www.maxim-ic.com/MAXQ**. For general questions and discussion of the MAXQ platform, visit our discussion board at **http://discuss.dalsemi.com**.

MAXQ is a registered trademark of Maxim Integrated Products, Inc.

1.1.2 Instruction Set

As part of the MAXQ family, the MAXQ7665/MAXQ7666 use the standard 16-bit MAXQ20 instruction set, with all instructions a fixed 16 bits in length. A register-based, transport-triggered architecture allows all instructions to be coded as simple transfer operations. All instructions reduce to either writing an immediate value to a destination register or memory location or moving data between registers and/or memory locations.

This simple top-level instruction decoding allows all instructions to be executed in a single cycle. Since all CPU operations are performed on registers only, any new functionality can be added by simply adding new register modules. The simple instruction set also provides maximum flexibility for code optimization by a compiler.

1.1.3 Harvard Memory Architecture

As part of the MAXQ family, the MAXQ7665/MAXQ7666 core architecture is based on the MAXQ20 design, which implements a 16-bit internal databus and ALU. Program memory, data memory, and register space on the MAXQ7665/MAXQ7666 follow the Harvard architecture model. Each type of memory is kept separate and is accessed by a separate bus, allowing different word lengths for different types of memory. Registers may be either 8 or 16 bits in width. Program memory is 16 bits in width to accommodate the standard MAXQ 16-bit instruction set. Data memory is also 16 bits in width but can be accessed in 8-bit or 16-bit modes for maximum flexibility.

The MAXQ7665/MAXQ7666 include a flexible memory management unit (MMU), which allows code to be executed from either the program flash, the utility ROM, or the internal data SRAM. Any of these three memory spaces may also be accessed in data space at any time, with the single restriction that whichever physical memory area is currently being used as program space cannot be read from in data space.

1.1.4 Register Space

Since all functions in the MAXQ family are accessed through registers, common functionality is provided through a common register set. Many of these registers provide the equivalent of higher level op codes by directly accessing the arithmetic logic unit (ALU), the loop counter registers, and the data pointer registers. Others, such as the interrupt registers, provide common control and configuration functions that are equivalent across all MAXQ microcontrollers.

The common register set, also known as the System Registers, includes the following:

- ALU access and control registers, including working accumulator registers and the processor status flags
- Two Data Pointers and a Frame Pointer for data memory access
- Auto-decrementing Loop Counters for fast, compact looping
- Instruction Pointer and other branching control access points
- Stack Pointer and an access point to the 16-bit-wide dedicated hardware stack
- Interrupt vector, identification, and masking registers

The MAXQ7665/MAXQ7666 peripheral register space (modules 0 to 5) contains registers that access the following peripherals:

- General-purpose, 8-bit, I/O port (P0)
- Serial UART interface
- Serial peripheral interface (SPI)
- Hardware multiplier
- JTAG debug engine
- Three programmable Type 2 timer/counters
- Controller area network (CAN) interface
- Analog input/output module

1.2 Architecture

The MAXQ7665/MAXQ7666 architecture is designed to be modular and expandable. Top-level instruction decoding is extremely simple and based on transfers to and from registers. The registers are organized into functional modules, which are in turn divided into the system register and peripheral register groups. Figure 1-2 illustrates the modular architecture and the basic transport possibilities.
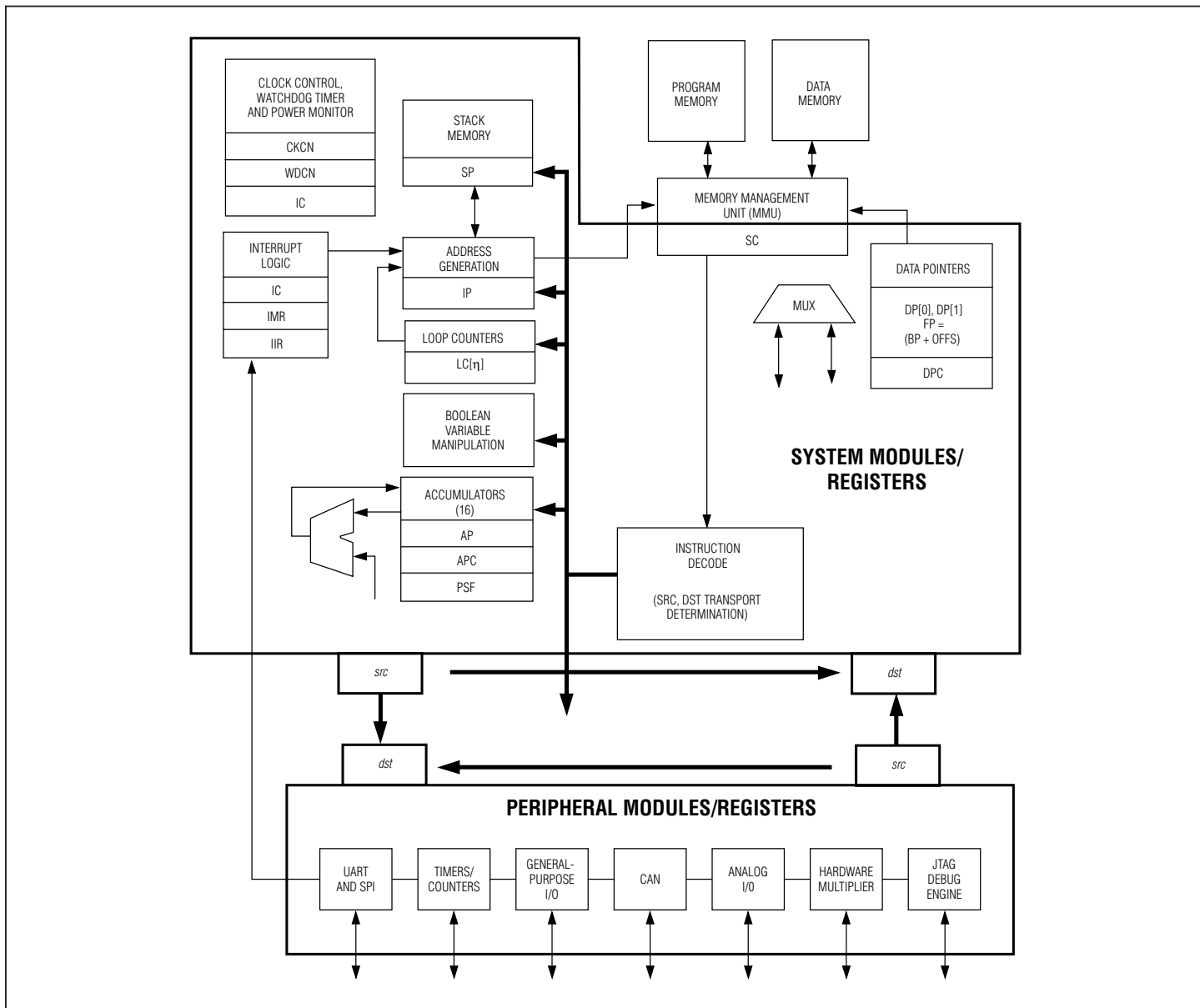
Figure 1-2. MAXQ7665/MAXQ7666 Transport-Triggered Architecture

Memory access from the MAXQ7665/MAXQ7666 is based on a Harvard architecture with separate address spaces for program and data memory. The simple instruction set and transport-triggered architecture allow the MAXQ7665/MAXQ7666 to run in a nonpipelined execution mode where each instruction can be fetched from memory, decoded, and executed in a single clock cycle. Data memory is accessed through one of three data pointer registers. Two of these data pointers, DP[0] and DP[1], are stand-alone 16-bit pointers. The third data pointer, FP, is composed of a 16-bit base pointer (BP) and an 8-bit offset register (OFFS). All three pointers support post-increment/decrement functionality for read operations and pre-increment/decrement for write operations. For the Frame Pointer (FP=BP[OFFS]), the increment/decrement operation is executed on the OFFS register and does not affect the base pointer (BP). Stack functionality is provided by dedicated memory with a 16-bit width and depth of 16. An on-chip memory management unit (MMU) is accessible through system registers to allow logical remapping of physical program and data spaces, and thus facilitates in-system programming and fast access to data tables, arrays, and constants physically located in program memory.

1.2.1 Instruction Decoding

Every MAXQ7665/MAXQ7666 instruction is encoded as a single 16-bit word according to the format in Figure 1-3.

FORMAT	DESTINATION							SOURCE							
f	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s

Figure 1-3. Instruction Word Format

Bit 15 (f) indicates the format for the source field of the instruction as follows:

- If f equals 0, the instruction is an immediate source instruction, and the source field represents an immediate 8-bit value.

- If f equals 1, the instruction is a register source instruction, and the source field represents the register that the source value will be read from.

Bits 0 to 7 (sssssss) represent the source for the transfer. Depending on the value of the format field, this can either be an immediate value or a source register. If this field represents a register, the lower four bits contain the module specifier and the upper four bits contain the register index in that module.

Bits 8 to 14 (ddddddd) represent the destination for the transfer. This value always represents a destination register, with the lower four bits containing the module specifier and the upper three bits containing the register subindex within that module.

Since the source field is 8 bits wide and 4 bits are required to specify the module, any one of 16 registers in that module may be specified as a source. However, the destination field has one less bit, which means that only eight registers in a module can be specified as a destination in a single-cycle instruction.

While the asymmetry between source and destination fields of the op code may initially be considered a limitation, this space can be used effectively. Firstly, since read-only registers will never be specified as destinations, they can be placed in the second eight locations in a module to give single-cycle read access. Secondly, there are often critical control or configuration bits associated with system and certain peripheral modules where limited write access is beneficial (e.g., watchdog-timer enable and reset bits). By placing such bits in one of the upper 24 registers of a module, this write protection is added in a way that is virtually transparent to the assembly source code. Anytime that it is necessary to directly select one of the upper 24 registers as a destination, the prefix register PFX is used to supply the extra destination bits. This prefix register write is inserted automatically by the assembler and requires one additional execution cycle.

The MAXQ7665/MAXQ7666 architecture is transport-triggered. This means that writing to or reading from certain register locations will also cause side effects to occur. These side effects form the basis for the higher level op codes defined by the assembler, such as ADDC, OR, JUMP, and so on. While these op codes are actually implemented as MOVE instructions between certain register locations, the encoding is handled by the assembler and need not be a concern to the programmer. The registers defined in the System Register and Peripheral Register maps operate as described in the documentation; the unused "empty" locations are the ones used for these special cases.

The MAXQ7665/MAXQ7666 instruction set is designed to be highly orthogonal. All arithmetic and logical operations that use two registers can use any register along with the accumulator. Data can be transferred between any two registers in a single instruction.

MAXQ7665/MAXQ7666 User's Guide

1.2.2 Register Space

The MAXQ7665/MAXQ7666 architecture provides a total of 16 register modules. Each of these modules contains 32 registers. Of these possible 16 register modules, only 13 are used on the MAXQ7665/MAXQ7666—seven for system registers and six for peripheral registers. The first eight registers in each module may be read from or written to in a single cycle; the second eight registers may be read from in a single cycle and written to in two cycles (by using the prefix register PFX); the last 16 registers may be read or written in two cycles (always requiring use of the prefix register PFX).

Registers may be either 8 or 16 bits in length. Within a register, any number of bits can be implemented; bits not implemented are fixed at zero. Data transfers between registers of different sizes are handled as shown in Table 1-1.

- If the source and destination registers are both 8 bits wide, data is transferred bit to bit accordingly.

- If the source register is 8 bits wide and the destination register is 16 bits wide, the data from the source register is transferred into the lower 8 bits of the destination register. The upper 8 bits of the destination register are set to the current value of the prefix register; this value is normally zero, but it can be set to a different value by the previous instruction if needed. The prefix register reverts back to zero after one cycle, so this must be done by the instruction immediately before the one that will be using the value.

- If the source register is 16 bits wide and the destination register is 8 bits wide, the lower 8 bits of the source are transferred to the destination register.

- If both registers are 16 bits wide, data is copied bit to bit.

Table 1-1. Register-to-Register Transfer Operations

SOURCE REGISTER SIZE (BITS)	DESTINATION REGISTER SIZE (BITS)	PREFIX SET?	DESTINATION SET TO VALUE	
			HIGH 8 BITS	LOW 8 BITS
8	8	—		Source [7:0]
8	16	No	00h	Source [7:0]
8	16	Yes	Prefix [7:0]	Source [7:0]
16	8	—		Source [7:0]
16	16	No	Source [15:8]	Source [7:0]

The above rules apply to all data movements between defined registers. Data transfer to/from undefined register locations has the following behavior:

- If the destination is an undefined register, the MOVE is a dummy operation but may trigger an underlying operation according to the source register (e.g., @DP[n]--).

- If the destination is a defined register and the source is undefined, the source data for the transfer will depend upon the source module width. If the source is from a module containing 8-bit or 8-bit and 16-bit source registers, the source data will be equal to the prefix data concatenated with 00h. If the source is from a module containing only 16-bit source registers, 0000h source data is used for the transfer.

The 16 available register modules are broken up into two different groups. The low six modules (specifiers 0h through 5h) are known as the Peripheral Register modules, while the high 10 modules (specifiers 6h to Fh) are known as the System Register modules. These groupings are descriptive only, as there is no difference between accessing the two register groups from a programming perspective.

The System Registers define basic functionality that remains the same across all products based on the MAXQ20 architecture. This includes all register locations that are used to implement higher level op codes as well as the following common system features.

- ALU (MAXQ20: 16 bits) and associated status flags (zero, equals, carry, sign, overflow)

- 16 working accumulator registers (MAXQ20: 16-bit width), along with associated control registers

- Instruction pointer

- Registers for interrupt control, handling, and identification

- Auto-decrementing loop counters for fast, compact looping

- Two data pointer registers and a frame pointer for data memory access

The MAXQ7665/MAXQ7666 peripheral register space (modules 0 to 5) contains registers that access the following peripherals:

- General-purpose, 8-bit, I/O port (P0)
- External interrupts (up to 8)
- Three programmable Type 2 timer/counters
- Serial UART interface
- SPI
- CAN interface
- Analog input/output module
- Hardware multiplier
- JTAG debug engine

The lower 8 bits of all registers in modules 0 to 5 (as well as the AP module M8) are bit addressable.

Table 1-2. MAXQ7665/MAXQ7666 Register Modules

REGISTER INDEX	MODULE NAME (BASE SPECIFIER)												
	M0	M1	M2	M3	M4	M5	M8	M9	M11	M12	M13	M14	M15
00h	PO0	MCNT	T2CNA0	T2CNA2	C0C	VMC	AP	A[0]	PFX[0]	IP			
01h		MA	T2H0	T2H2	C0S	APE	APC	A[1]	PFX[1]		SP		
02h		MB	T2RH0	T2RH2	C0IR	ACNT		A[2]	PFX[2]		IV		
03h	EIF0	MC2	T2CH0	T2CH2	C0TE	DCNT		A[3]	PFX[3]			OFFS	DP0
04h		MC1	T2CNA1		C0RE	DACI	PSF	A[4]	PFX[4]			DPC	
05h		MC0	T2H1		COR		IC	A[5]	PFX[5]			GR	
06h		SPIB	T2RH1		C0DP	DACO	IMR	A[6]	PFX[6]		LC0	GRL	
07h	SBUF0	SPICN	T2CH1		C0DB			A[7]	PFX[7]		LC1	BP	DP1
08h	PI0	SPICF	T2CNB0	T2CNB2	C0RMS	ADCD	SC	A[8]				GRS	
09h		SPICK	T2V0	T2V2	C0TMA	TSO		A[9]				GRH	
0Ah		FCNTL	T2R0	T2R2		AIE		A[10]				GRXL	
0Bh	EIE0	FDATA	T2C0	T2C2		ASR	IIR	A[11]				FP	
0Ch		MC1R	T2CNB1			OSCC		A[12]					
0Dh		MC0R	T2V1					A[13]					
0Eh			T2R1				CKCN	A[14]					
0Fh			T2C1				WDCN	A[15]					
10h	PD0		T2CFG0	T2CFG2									
11h			T2CFG1		C0M1C								
12h					C0M2C								
13h	EIES0				C0M3C								
14h					C0M4C								
15h					C0M5C								
16h					C0M6C								

MAXQ7665/MAXQ7666 User's Guide

Table 1-2. MAXQ7665/MAXQ7666 Register Modules (continued)

REGISTER INDEX	MODULE NAME (BASE SPECIFIER)													
	M0	M1	M2	M3	M4	M5	M8	M9	M11	M12	M13	M14	M15
17h					C0M7C								
18h			ICDT0		C0M8C								
19h			ICDT1		C0M9C								
1Ah			ICDC		C0M10C								
1Bh			ICDF		C0M11C								
1Ch		FADDR	ICDB		C0M12C								
1Dh	SCON0		ICDA		C0M13C								
1Eh	SMD0		ICDD		C0M14C								
1Fh	PR0				C0M15C								

RESERVED OR OPCODE	PORT PINS (GPIO)	SERIAL AND SPI	INTERRUPT CONTROL	HARDWARE MULTIPLIER	TIMERS	CAN	ANALOG I/O	ACC ARRAY, CONTROL	OTHER FUNCTIONS

1.2.3 Memory Organization

Beyond the internal register space, memory on the MAXQ7665/MAXQ7666 microcontrollers is organized according to a Harvard architecture, with a separate address space and bus for program memory and data memory. Stack memory is also separate and is accessed through a dedicated register set.

To provide additional memory map flexibility, an MMU allows data memory space to be mapped into a predefined program memory segment, thus affording the possibility of code execution from data memory. Additionally, program memory space can be made accessible as data space, allowing access to constant data stored in program memory. All memory is internal, and physical memory segments (other than the stack and register memories) can be accessed as either program memory or as data memory, but not both at once.

1.2.3.1 Program Memory

The MAXQ7665/MAXQ7666 contain up to 64k x 16 (128kB) of flash memory, which normally serves as program memory. When executing from the data SRAM or utility ROM, this memory is mapped to data space and can be used for lookup tables and similar functions. Flash memory mapped into data space can be read from directly, like any other type of data memory. However, writing to flash memory must be done by calling the in-application functions provided by the utility ROM. The utility ROM provides routines to carry out the necessary operations (erase, write) on flash memory.

Table 1-3 summarizes the features of the flash memory supported in the MAXQ7665A–MAXQ7665D devices. The MAXQ7666 device features a 256B data flash in addition to 16kB program flash. The MAXQ7666 flash is different from the MAXQ7665A–MAXQ7665D and its features are summarized in Tables 1-4 and 1-5. Refer to the respective data sheets for additional information.

Program memory begins at address 0000h and is contiguous through the internal program memory. The actual size of the on-chip program memory available for user application is product dependent. Given a 16-bit program address bus, the maximum program space is 64kWords. Since the codewords are 16 bits, the program memory is, therefore, a 64k x 16 linear space.

Table 1-3. MAXQ7665A–MAXQ7665D Flash Memory Features

FEATURE	MAXQ7665A	MAXQ7665B	MAXQ7665C	MAXQ7665D
Flash Type	Type A	Type A	Type A	Type A
Flash Size	128kB (64k x 16)	64kB (32k x 16)	48kB (24k x 16)	32kB (16k x 16)
Flash Organization	5 Sectors	4 Sectors	3 Sectors	3 Sectors
Sector Address/Size	0000h–7FFFh (32k x 16)	0000h–3FFFh (16k x 16)	0000h–3FFFh (16k x 16)	0000h–1FFFh (8k x 16)
	8000h–BFFFh (16k x 16)	4000h–5FFFh (8k x 16)	4000h–4FFFh (4k x 16)	2000h–2FFFh (4k x 16)
	C000h–DFFFh (8k x 16)	6000h–6FFFh (4k x 16)	5000h–5FFFh (4k x 16)	3000h–3FFFh (4k x 16)
	E000h–EFFFh (4k x 16)	7000h–7FFFh (4k x 16)	—	—
	F000h–FFFFh (4k x 16)	—	—	—
Flash Erase	Erase All	Erase All	Erase All	Erase All
	Sector Erase	Sector Erase	Sector Erase	Sector Erase
Flash Program	Word Write	Word Write	Word Write	Word Write
In Application Programming	Yes, using utility ROM routines. See *Section 15* for more information.			
In System Programming	Yes, using utility ROM JTAG bootstrap loader. See *Section 12* for more information.			

Table 1-4. MAXQ7666 Program Flash Features

FEATURE	MAXQ7666
Flash Type	Type F
Program Flash Size	16kB (8k x 16)
Program Flash Organization	256 Pages
	1 Page = 64B (32 x 16)
Program Flash Page Address	0000h–001Fh (Page 0)
	0020h–003Fh (Page 1)
	0040h–005Fh (Page 2)
	. . .
	1FC0h–1FDFh (Page 254)
	1FE0h–1FFFh (Page 255)
Program Flash Erase	Erase All
	2 Page Erase
Program Flash Write	1 Page Write
In Application Programming	Yes, using utility ROM routines. See *Section 16* for more information.
In System Programming	Yes, using utility ROM JTAG bootstrap loader. See *Section 12* for more information.

MAXQ7665/MAXQ7666 User's Guide

Table 1-5. MAXQ7666 Data Flash Features

FEATURE	MAXQ7666
Flash Type	Type F
Data Flash Size	256B (128 x 16)
Data Flash Organization (Regular Mode)	128 Pages
	1 Page = 2B (1 x 16)
Data Flash Page Address (Regular Mode)	4000h (Page 0)
	4001h (Page 1)
	4002h (Page 2)
	….
	407Fh (Page 127)
Data Flash Erase (Regular Mode)	Erase All
	2 Page Erase
Data Flash Write (Regular Mode)	1 Page Write
Data Flash Organization (Even Mode)	64 Even Pages
	1 Page = 2B (1 x 16)
Data Flash Page Address (Even Mode)	4000h (Page 0)
	4002h (Page 1)
	4004h (Page 2)
	. . .
	407Eh (Page 63)
Data Flash Erase (Even Mode)	Erase All
	1 Page Erase
Data Flash Write (Even Mode)	1 Page Write
In Application Programming	Yes, using utility ROM routines. See *Section 16* for more information.
In System Programming	Not supported, only in application programming.

Program memory is accessed directly by the program fetching unit and is addressed by the Instruction Pointer register. From an implementation perspective, system interrupts and branching instructions simply change the contents of the Instruction Pointer and force the op code fetch from a new program location. The Instruction Pointer is direct read/write accessible by the user software; write access to the Instruction Pointer will force program flow to the new address on the next cycle following the write. The contents of the Instruction Pointer will be incremented by 1 automatically after each fetch operation. The Instruction Pointer defaults to 8000h, which is the starting address of the utility ROM. The default IP setting of 8000h is assigned to allow initial in-system programming to be accomplished with utility ROM code assistance. The utility ROM code interrogates a specific register bit in order to decide whether to execute in-system programming or jump immediately to user code starting at 0000h. The user code reset vector should always be stored in the lowest bytes of the program memory.

1.2.3.2 Utility ROM

A utility ROM (4k x 16) is placed in the upper 32kWord program memory space starting at address 8000h. This utility ROM provides the following system utility functions:

- Reset vector
- Bootstrap function for system initialization
- In-application programming
- In-circuit debug

Following each reset, the processor automatically starts execution at address 8000h in the utility ROM, allowing ROM code to perform any necessary system support functions. Next, the System Programming Enable (SPE) bit is examined to determine whether system programming should commence or whether that code should be bypassed, instead forcing execution to vector to the start of user program code. When the SPE bit is set to logic 1, the processor will execute the prescribed Bootstrap Loader mode program that resides in utility ROM. The SPE bit defaults to 0. To enter the Bootstrap Loader mode, the SPE bit can be set to 1 during reset via the JTAG interface. When in-system programming is complete, the Bootstrap Loader can clear the SPE bit and reset the device such that the in-system programming routine is subsequently bypassed.

The MAXQ7665/MAXQ7666 application programming routines available as part of the utility ROM are covered in Sections 15 and 16. The MAXQ7665/MAXQ7666 JTAG test access port, in-circuit debug, and bootstrap loader mode for in-system programming are covered in Sections 10, 11, and 12.

1.2.3.3 Data Memory

The MAXQ7665/MAXQ7666 contain 256 x 16 (512 bytes) of on-chip data SRAM that can be mapped into either program or data space. The contents of this SRAM are indeterminate after power-on reset, but are maintained during stop mode and across non-POR resets, as long as the DVDD supply stays within the acceptable range.

On-chip data memory begins at address 0000h and is contiguous through the internal data memory. Data memory is accessed via indirect register addressing through a Data Pointer (@DP[n]) or Frame Pointer (@BP[OFFS]). The Data Pointer is used as one of the operands in a MOVE instruction. If the Data Pointer is used as source, the core performs a Load operation that reads data from the data memory location addressed by the Data Pointer. If the Data Pointer is used as destination, the core executes a Store operation that writes data to the data memory location addressed by the Data Pointer. The Data Pointer can be directly accessed by the user software.

The core incorporates two 16-bit Data Pointers (DP[0] and DP[1]) to support data memory accessing. All Data Pointers support indirect addressing mode and indirect addressing with auto-increment or auto-decrement. Data Pointers DP[0] and DP[1] can be used as post increment/decrement source pointers by a MOVE instruction or pre increment/decrement destination pointers by a MOVE instruction. Using Data Pointer indirectly with "++" will automatically increase the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]++) or immediately preceding the execution of a write operation (@++DP[n]). Using Data Pointer indirectly with "--" will decrease the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]--) or immediately preceding the execution of a write operation (@--DP[n]).

The Frame Pointer (BP[OFFS]) is formed by 16-bit unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (OFFS). Frame Pointer can be used as a post increment/decrement source pointer by a MOVE instruction or as a pre increment/decrement destination pointer. Using Frame Pointer indirectly with "++" (@BP[++OFFS] for a write or @BP[OFFS++] for a read) will automatically increase the content of the Frame Pointer Offset by 1 immediately before or after the execution of data transfer depending upon whether it is used as a destination or source pointer respectively. Using Frame Pointer indirectly with "--" (@BP[--OFFS] for a write or @BP[OFFS--] for a read) will decrease the content of the Frame Pointer Offset by 1 immediately before/after execution of data transfer depending upon whether it is used as a destination or source pointer respectively. Note that the increment/decrement function affects the content of the OFFS register only, while the contents of the BP register remain unaffected by the borrow/carry out from the OFFS register.

A data memory cycle contains only one system clock period to support fast internal execution. This allows read or write operations on SRAM to be completed in one clock cycle. Data memory mapping and access control are handled by the MMU. Read/write access to the data memory can be in word or in byte.

When using the in-circuit debugging features of the MAXQ7665/MAXQ7666, the top 19 bytes (bytes 0x1ED to 0x1FF) of the SRAM must be reserved for saved state storage and working space for the debugging routines in the utility ROM. If in-circuit debug will not be used, the entire SRAM is available for application use.

1.2.3.4 Stack Memory

The MAXQ7665/MAXQ7666 provide a 16 x 16 hardware stack to support subroutine calls and system interrupts. A 16-bit wide on-chip stack is provided by the MAXQ7665/MAXQ7666 for storage of program return addresses and general-purpose use. The stack is used automatically by the processor when the CALL, RET, and RETI instructions are executed and when an interrupt is serviced; it can also be used explicitly to store and retrieve data by using the @SP- - source, @++SP destination, or the PUSH, POP, and POPI instructions. The POPI instruction acts identically to the POP instruction except that it additionally clears the INS bit.

The width of the stack is 16 bits to accommodate the instruction pointer size. The stack depth is 16 for the MAXQ7665/MAXQ7666. As the stack pointer register SP is used to hold the index of the top of the stack, the maximum size of the stack allowed is defined by the number of bits defined in the SP register (e.g., 4 bits for stack depth of 16).

On reset, the stack pointer SP initializes to the top of the stack (e.g. 0Fh for a 16-word stack). The CALL, PUSH, and interrupt vectoring operations increment SP and then store a value at @SP. The RET, RETI, POP, and POPI operations retrieve the value at @SP and then decrement SP.

As with the other RAM-based modules, the stack memory is initialized to indeterminate values upon reset or power-up. Stack memory is dedicated for stack operations only and cannot be accessed through program or data address spaces.

When using the in-circuit debugging features of the MAXQ7665/MAXQ7666, one word of the stack must be reserved to store the return location when execution branches into the debugging routines in the utility ROM. If in-circuit debug will not be used, the entire stack is available for application use.

1.2.3.5 Pseudo-Von Neumann Memory Mapping

The MAXQ7665/MAXQ7666 support a pseudo-Von Neumann memory structure that can merge program and data into a linear memory map. This is accomplished by mapping the data memory into the program space or mapping program memory segment into the data space. Program memory from 0000h to 7FFFh is the normal user code segment, followed by the utility ROM segment. The uppermost part of the 64kWord memory is the logical area for data memory when accessed as a code segment.

The program memory is logically divided into four program pages:

- P0 contains the lower 16kWords,
- P1 contains the second 16kWords,
- P2 contains the third 16kWords, and
- P3 contains the fourth 16kWords.

By default, P2 and P3 are not accessible for program execution until they are explicitly activated by the user software. The Upper Program Access (UPA) bit must be set to logic 1 to activate P2 and P3. Once UPA is set, P2 and P3 will occupy the upper half of the 64kWord program space. In this configuration (UPA = 1), the utility ROM cannot be accessed as program memory and the physical data memory cannot be accessed logically in program space.

The logical mapping of physical program memory page(s) into data space depends upon two factors: physical memory currently in use for program execution; and word/byte data memory access selection. If execution is from the utility ROM, physical program memory page(s) can logically be mapped to the upper half of data memory space. If logical data memory is used for execution, physical program memory page(s) can logically be mapped to the lower half of data memory space. If byte access mode is selected, only one page (16kWords) may be logically mapped, as just defined, to either the upper or lower half of data memory. If word access mode is selected, two pages (32kWords total) may be logically mapped to data memory. To avoid memory overlapping in the byte access mode, the physical data memory should be confined to the address range 0000h to 3FFFh in word mode. The selection of physical memory page or pages to be logically mapped to data space is determined by the Code Access Bits (CDA1:0):

CDA1:0	SELECTED PAGE IN BYTE MODE	SELECTED PAGE IN WORD MODE
00	P0	P0 and P1
01	P1	P0 and P1
10	P2	P2 and P3
11	P3	P2 and P3

Figure 1-4 summarizes the default memory maps for this memory structure. The primary difference lies in the reset default settings for the data pointer Word/Byte Mode Select (WBSn) bits. The WBSn bits of the MAXQ7665/MAXQ7666 default to word access mode (WBSn = 1).
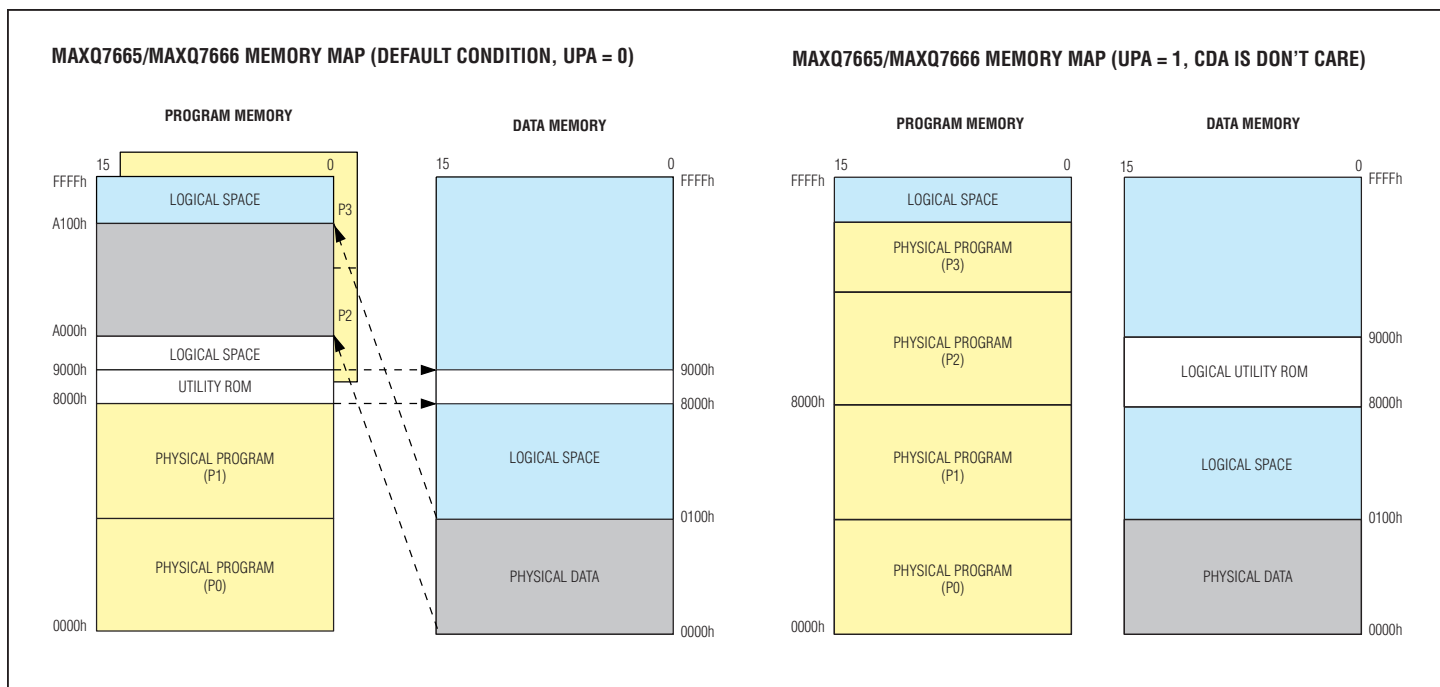
Figure 1-4. Pseudo-Von Neumann Memory Map (MAXQ7665/MAXQ7666 Default)

1.2.3.6 Pseudo-Von Neumann Memory Access

The pseudo-Von Neumann memory mapping is straightforward if there is no memory overlapping among the program, utility ROM, and data memory segments. However, for applications requiring large-size program memory, the paging scheme can be used to selectively activate those overlapped memory segments. The UPA bit can be used to activate the upper half of the physical program code (P2 and P3) for program execution. When accessing the program memory as data, the CDA bits can be used to select one of the four program pages as needed. Full data memory access to any of the four physical program memory pages is based on the assumption that the maximum physical data memory is in the range of 16k x 16. The other restriction for accessing the pseudo-Von Neumann map is that when program execution is in a particular memory segment, the same memory segment cannot be simultaneously be accessed as data.

When executing from the lower 32k program space (P0 and P1):

- The upper half of the code segment (P2 and P3) is accessible as program if the UPA bit is set to 1.
- The physical data memory is available for accessing as a code segment with offset at A000h if the UPA bit is 0.
- Load and Store operations addressed to physical data memory are executed as normal.
- The utility ROM can be read as data, starting at 8000h of the data space.

When executing from the utility ROM (only allowable when UPA = 0):

- The lower 32k program space (P0 and P1) functions as normal program memory.
- The upper half of the code segment (P2 and P3) is not accessible as program (since UPA = 0).
- The physical data memory is available for accessing as a code segment with offset at A000h.
- Load and Store operations addressed to physical data memory are executed as normal.
- One page (byte access mode) or two pages (word access mode) can be accessed as data with offset at 8000h as determined by the CDA1:0 bits.

When executing from the data memory (only allowable when UPA = 0):

- Program flows freely between the lower 32k user code (P0 and P1) and the utility ROM segment.
- The upper half of the code segment (P2 and P3) is not accessible as program (since UPA = 0).

- The utility ROM can be accessed as data with offset at 8000h.
- One page (byte access mode) or two pages (word access mode) can be accessed as data with offset at 0000h as determined by the CDA1:0 bits.

1.2.3.7 Data Alignment

To support merged program and data memory operation while maintaining efficiency on memory space usage, the data memory must be able to support both byte-wide and word-wide accessing. Data is aligned in data memory as word, but the effective data address is resolved to bytes. This data alignment allows direct program fetching in its native word size while maintaining accessibility at the byte level. It is important to realize that this accessibility requires strict word alignment. All executable words must align to an even address in byte mode. Care must be taken when updating the code segment in the unified data memory space as misalignment of words will likely result in loss of program execution control. Worst yet, this situation may not be detected if the watchdog timer is also disabled.

Data memory is organized as two byte-wide memory banks with common word address decode but two 8-bit data buses. The data memory will always be read as a complete word, independent of operation, whether program fetch or data access. The program decoder always uses the full 16-bit word, whereas the data access can utilize a word or an individual byte.

In byte mode, data pointer hardware reads out the word containing the selected byte using the effective data word address pointer (the least significant bit of the byte data pointer is not initially used). Then, the least significant data pointer bit functions as the byte select that is used to place the target byte to the data path. For write access, data pointer hardware addresses a particular word using the effective data word address while the least significant bit selects the corresponding data bank for write, leaving the contents of the other memory bank unaffected.

1.2.3.8 Memory Management Unit

Memory allocation and accessing control for program and data memory can be managed by the memory management unit (MMU). A single memory management unit option is discussed in this user's guide, however the memory management unit implementation for any given product depends upon the type and amount of memory addressable by the device. Users should consult the individual product data sheet(s) and/or user's guide supplement(s) for detailed information.

Although supporting less than the maximum addressable program and data memory segments, the MMU implementation presented provides a high degree of programming and access control flexibility. It supports the following:

- User program memory up to 32k x 16 (up to 64k x 16 with inclusion of UPA bit).
- Utility ROM up to 8k x 16.
- Data memory SRAM up to 16k x 16.
- In-system and in-application programming of embedded EEPROM, flash, or SRAM memories.
- Access to any of the three memory areas (SRAM, code memory, utility ROM) using the data memory pointers.
- Execution from any of the three memory areas (SRAM, code memory, factory written and tested utility-ROM routines).

Given these capabilities, the following rules apply to the memory map:

- A particular memory segment cannot be simultaneously accessed as both program and data.
- The offset address is A000h when logically mapping data memory into the program space.
- The offset for logically mapping the utility ROM into the data memory space is 8000h.
- Program memory:
  - The lower half of the program memory (P0 and P1) is always accessible, starting at 0000h.
  - The upper half of the program memory (P2 and P3) must be activated by setting the UPA bit to 1 when accessing for code execution, starting at 8000h.
  - Setting the UPA bit to 1 disallows access to the utility ROM and logical data memory as program.
  - Physical program memory pages (P0, P1, P2, P3) are logically mapped into data space based upon the memory segment currently being used for execution, selection of byte/word access mode, and CDA1:0 bit settings (described in the *Pseudo-Von Neumann Memory Map* and *Pseudo-Von Neumann Memory Access* sections).
- Data memory
  - Access can be either word or byte.
  - All 16 data pointer address bits are significant in either access mode (word or byte).

MAXQ7665/MAXQ7666 MEMORY MAP (UPA = 0, EXECUTING FROM UTILITY ROM)

PROGRAM MEMORY

FFFFh	LOGICAL SPACE
A100h	
	LOGICAL DATA
A000h	
9000h	LOGICAL SPACE
8000h	UTILITY ROM
	PHYSICAL PROGRAM (P1)
	PHYSICAL PROGRAM (P0)
0000h	

15 ... 0 / P3 / P2

DATA MEMORY

CDA1 = 1
CDA1 = 0

8000h	
0100h	
	PHYSICAL DATA
0000h	

MAXQ7665/MAXQ7666 MEMORY MAP (UPA = 0, EXECUTING FROM LOGICAL DATA MEMORY)

PROGRAM MEMORY

FFFFh	LOGICAL SPACE
A100h	
	LOGICAL DATA MEMORY
A000h	
9000h	LOGICAL SPACE
8000h	UTILITY ROM
	PHYSICAL PROGRAM (P1)
	PHYSICAL PROGRAM (P0)
0000h	

15 ... 0 / P3 / P2

DATA MEMORY

FFFFh	LOGICAL SPACE
8000h	LOGICAL UTILITY ROM
0000h	

CDA1 = 1
CDA1 = 0

Figure 1-5. CDA Functions (Word Access Mode)

EXECUTING FROM UTILITY ROM (UPA = 0, ONLY P1, P2 PRESENT)
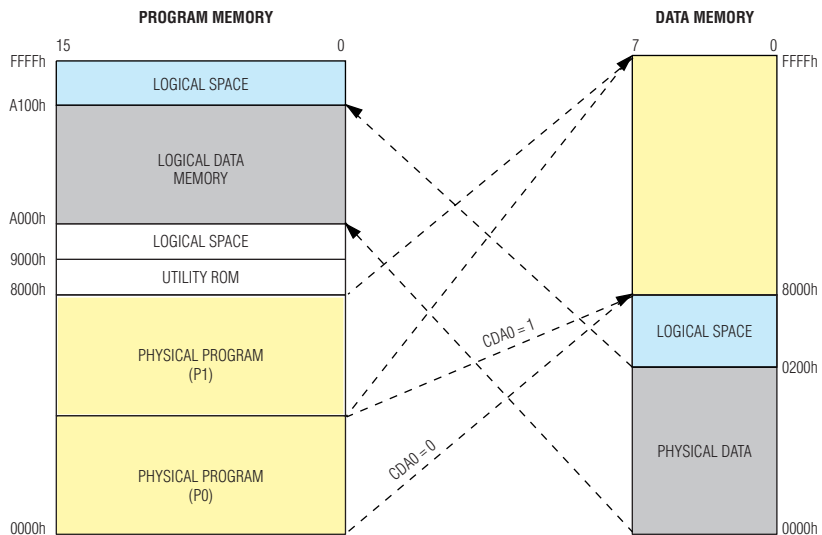
EXECUTING FROM LOGICAL DATA MEMORY (UPA = 0, ONLY P1, P2 PRESENT)
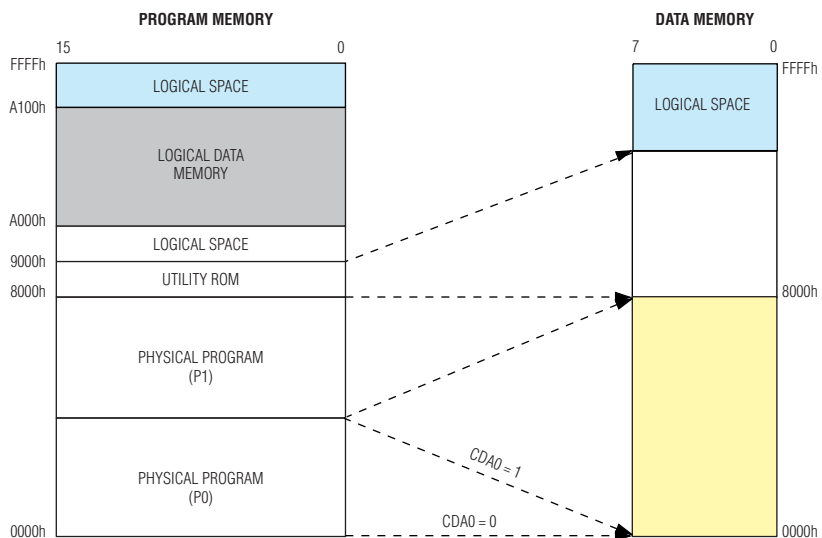
Figure 1-6. CDA Functions (Byte Access Mode)

1.2.3.9 Program and Data Memory Mapping Example 1: MAXQ7665B

Figures 1-7, 1-8, and 1-9 show the mapping of physical memory segments into the program and data memory space for the MAXQ7665B with 32k x 16 (64kB) program flash memory. In this case and all cases when program flash memory size is ≤ 32k x 16, the memory mapping is straightforward as there is no overlapping among the program, utility ROM, and data memory segments. The mapping of memory segments into program space is always the same. The mapping of memory segments into data space varies depending on which memory segment is currently being executed from.

In all cases, whichever memory segment is currently being executed from in program space cannot be accessed in data space.
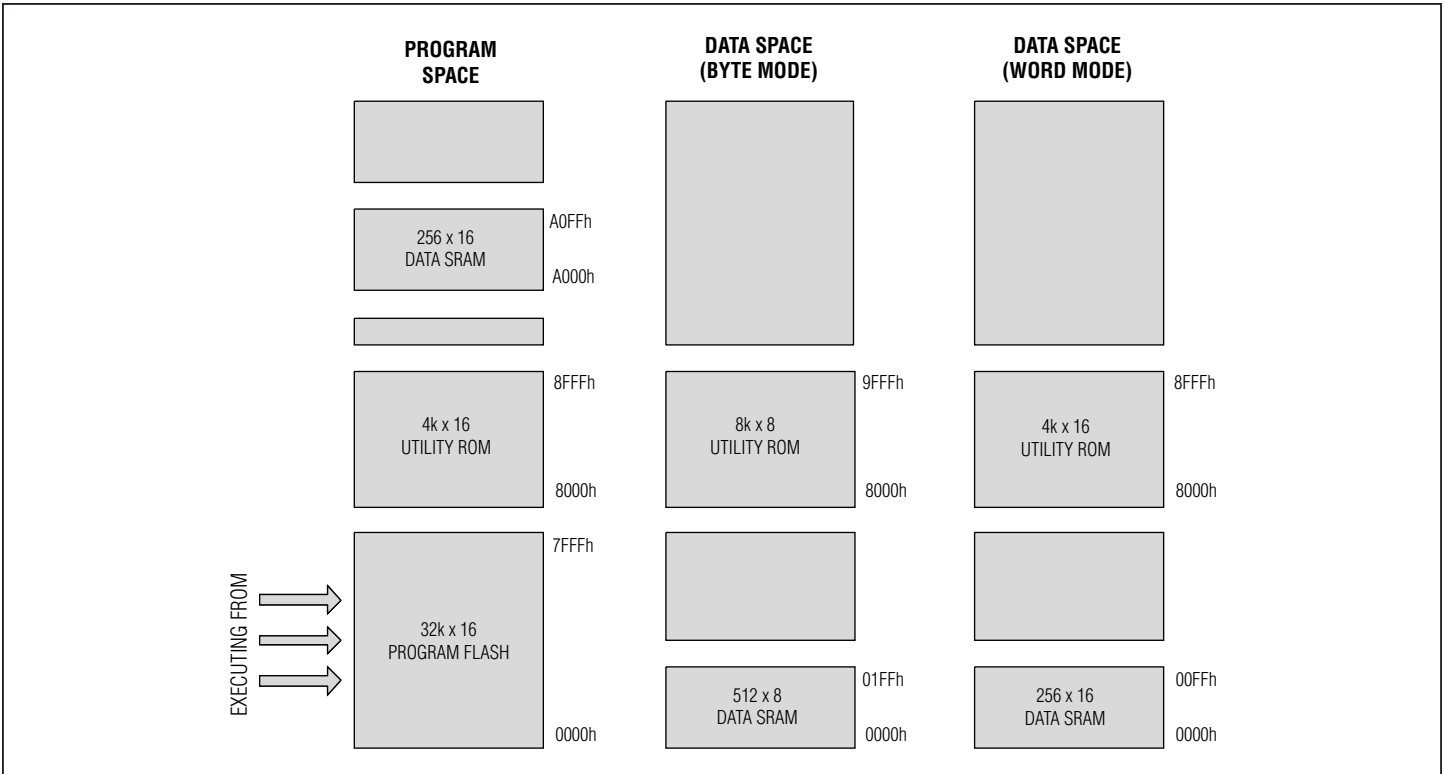
Figure 1-7. MAXQ7665B Memory Map When Executing from Application Flash
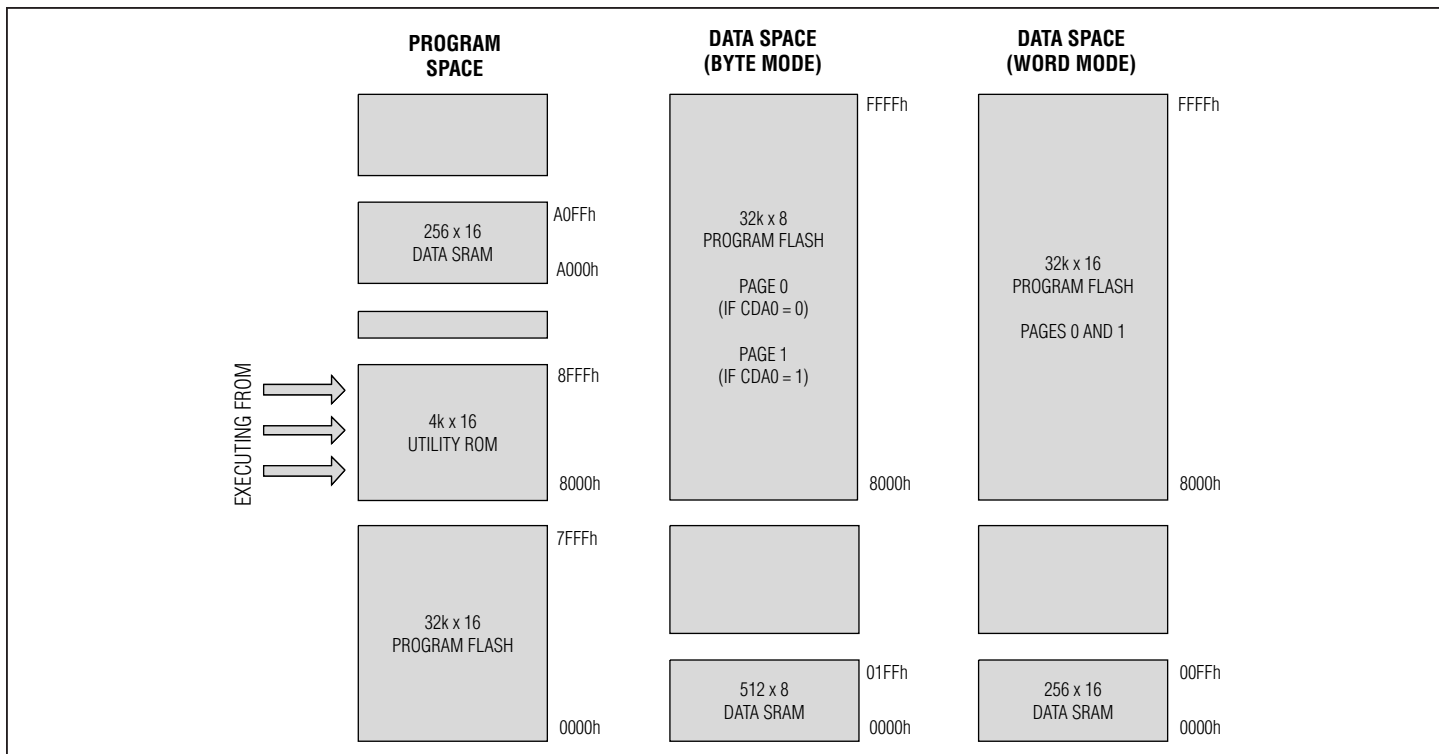
Figure 1-8. MAXQ7665B Memory Map When Executing from Utility ROM
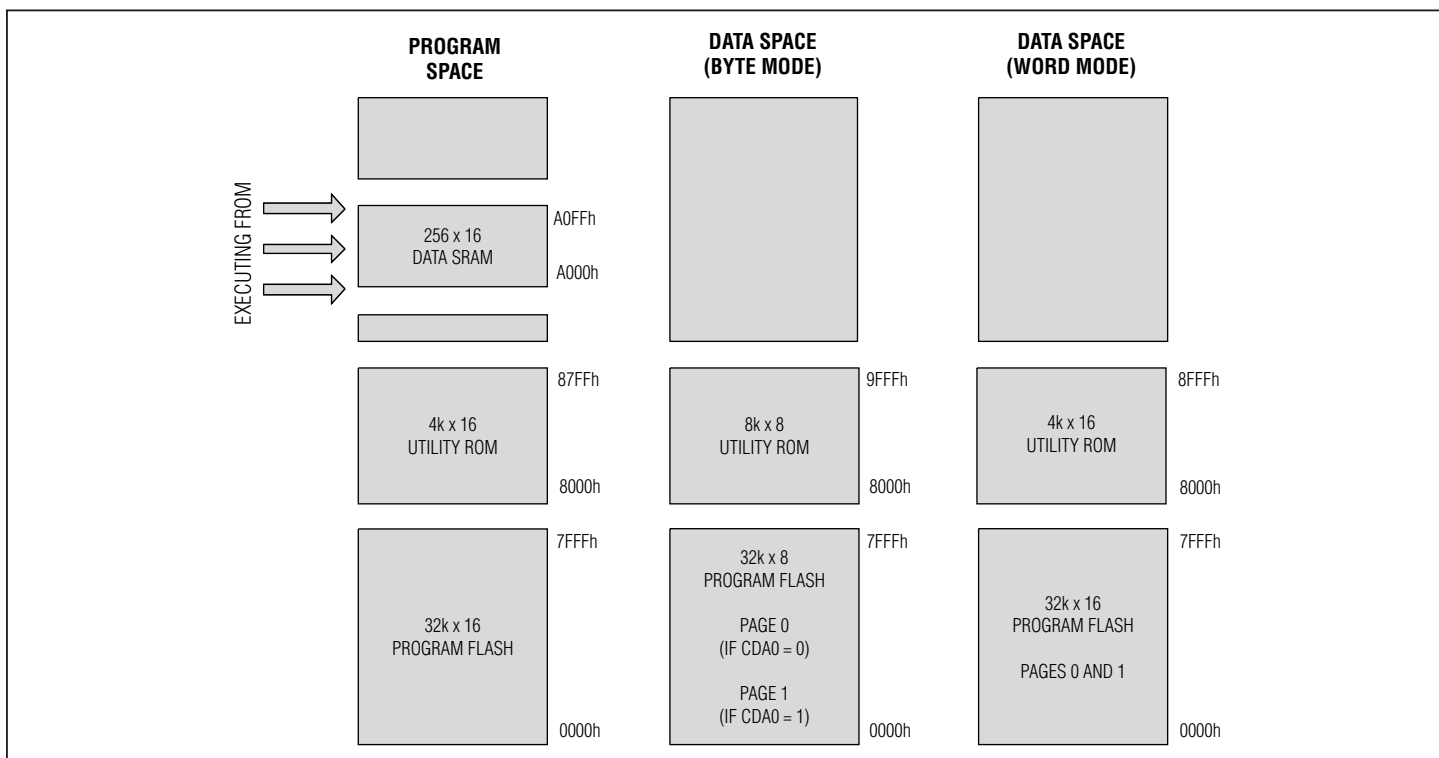
Figure 1-9. MAXQ7665B Memory Map When Executing from Data SRAM

1.2.3.10 Program and Data Memory Mapping Example 2: MAXQ7666

Figures 1-10, 1-11, and 1-12 show the mapping of physical memory segments into the program and data memory space for the MAXQ7666 with 8k x 16 (16kB) program flash memory, 256B data flash memory, and 512B data RAM.
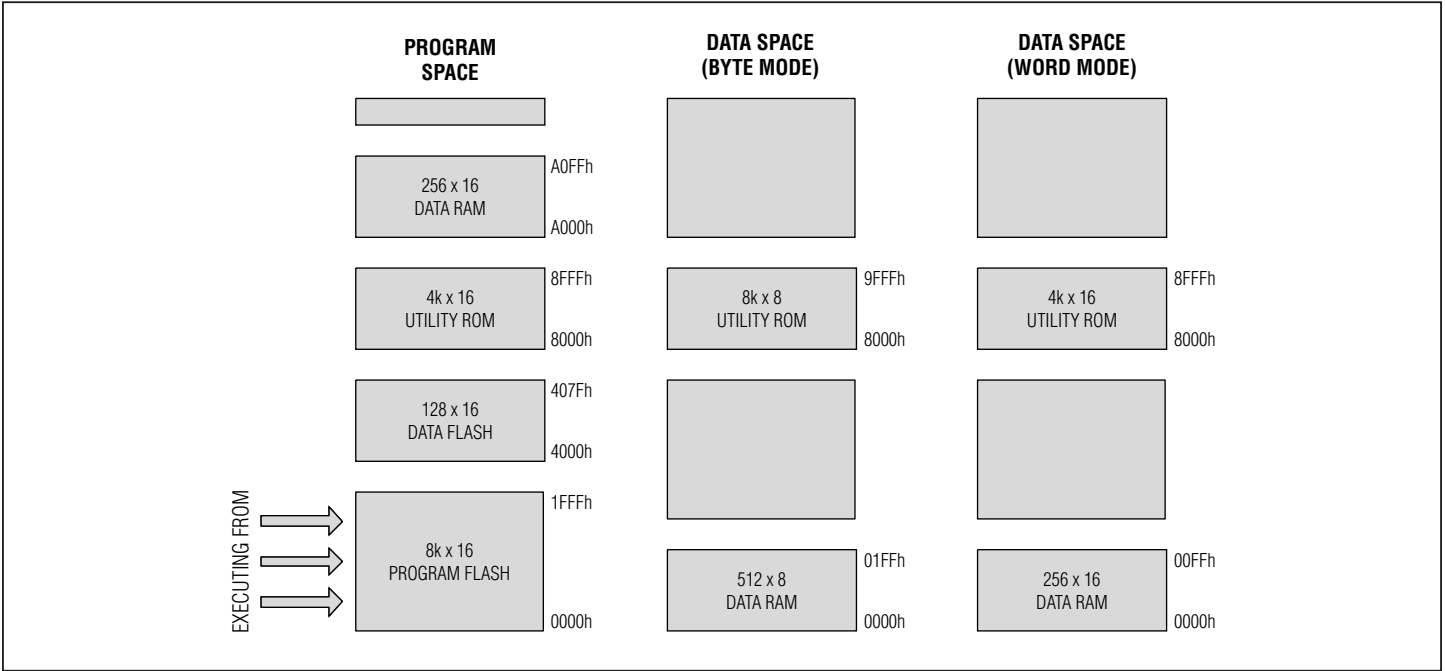
Figure 1-10. MAXQ7666 Memory Map When Executing from Application Flash
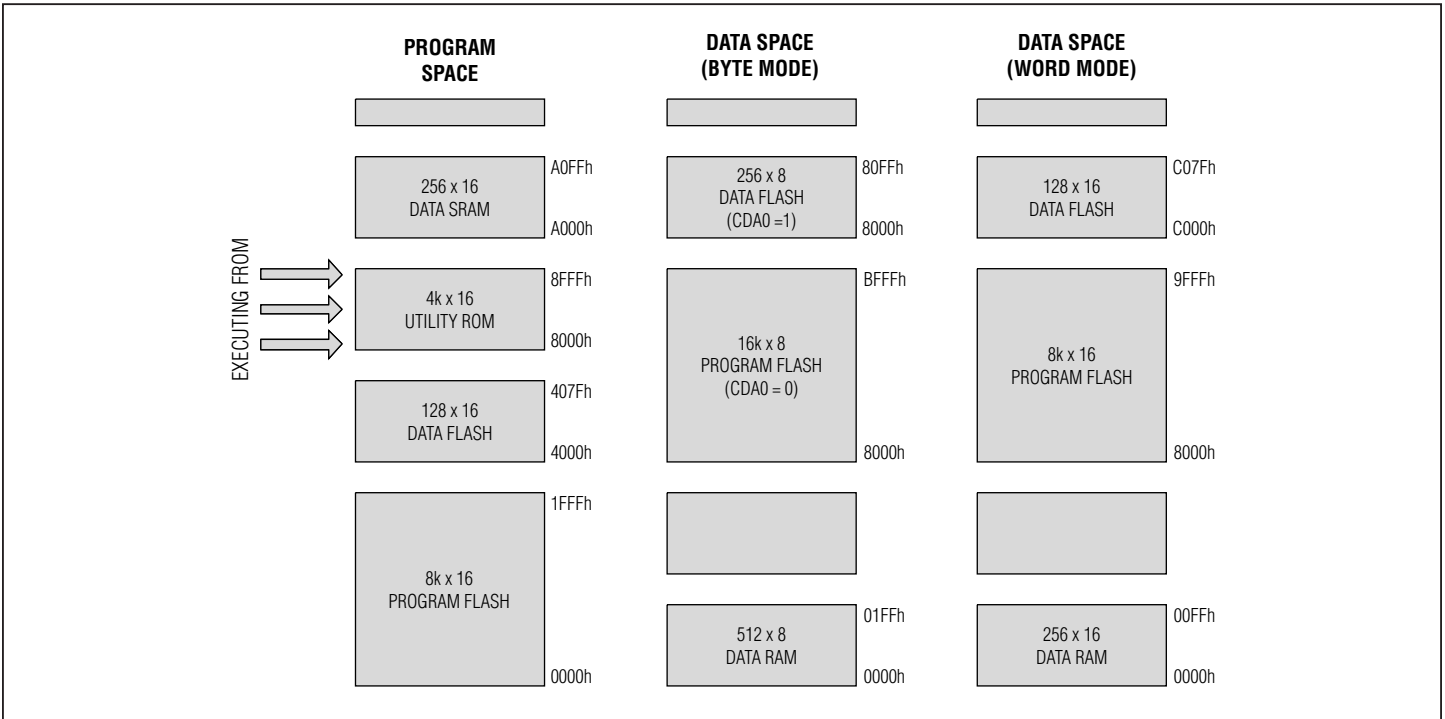
Figure 1-11. MAXQ7666 Memory Map When Executing from Utility ROM
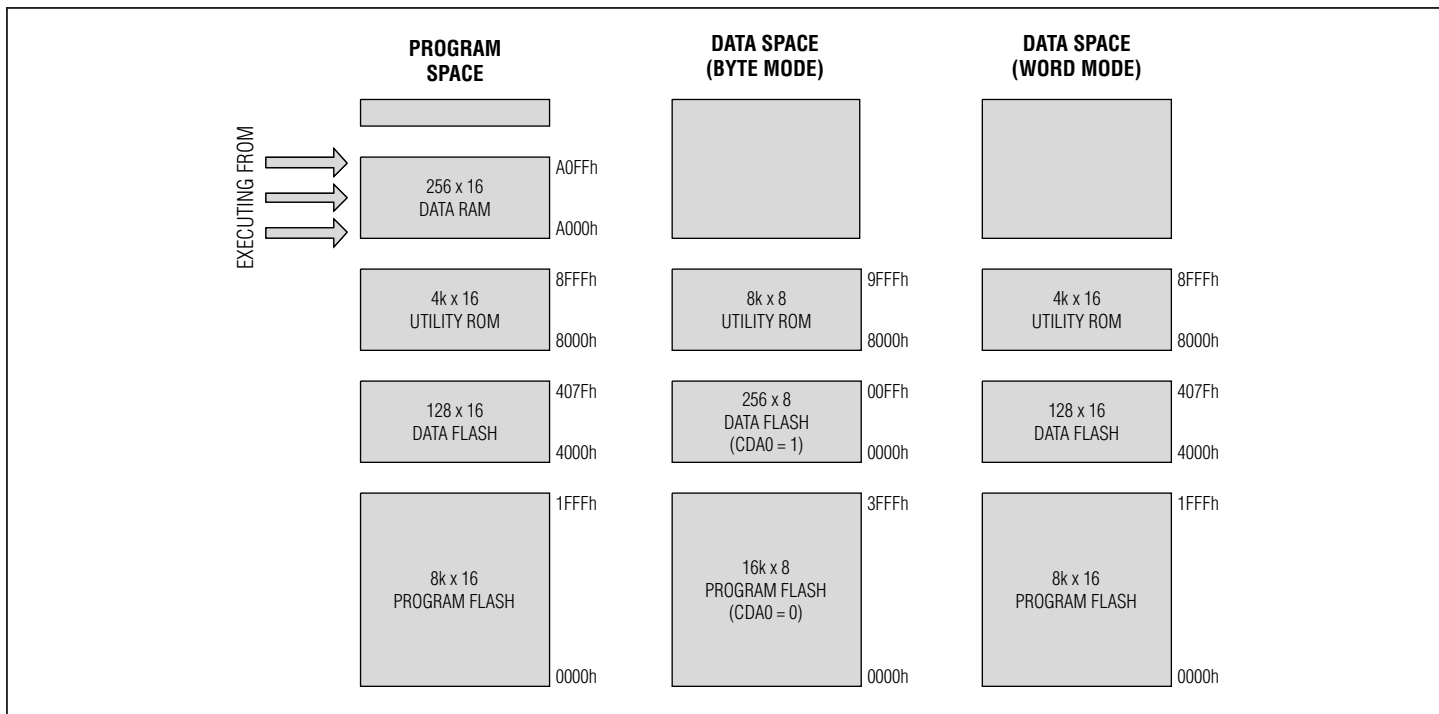
Figure 1-12. MAXQ7666 Memory Map When Executing from Data RAM

1.2.4 Interrupts

The MAXQ7665/MAXQ7666 provide a single, programmable interrupt vector (IV) that can be used to handle internal and external interrupts. Interrupts can be generated from system level sources (e.g., watchdog timer) or by sources associated with the peripheral modules included in the specific MAXQ7665/MAXQ7666 microcontrollers. Only one interrupt can be handled at a time, and all interrupts naturally have the same priority. A programmable interrupt mask register allows software-controlled prioritization and nesting of high-priority interrupts.

1.2.4.1 Servicing Interrupts

For the MAXQ7665/MAXQ7666 to service an interrupt, interrupts must be enabled globally, modularly, and locally. The Interrupt Global Enable (IGE) bit located in the Interrupt Control (IC) register acts as a global interrupt mask. This bit defaults to 0, and it must be set to 1 before any interrupt takes place.

The local interrupt-enable bit for a particular source is in one of the peripheral registers associated with that peripheral module, or in a system register for any system interrupt source. Between the global and local enables are intermediate per-module and system interrupt mask bits. These mask bits reside in the Interrupt Mask system register. By implementing intermediate per-module masking capability in a single register, interrupt sources spanning multiple modules can be selectively enabled/disabled in a single instruction. This promotes a simple, fast, and user-definable interrupt prioritization scheme. The interrupt source-enable hierarchy is illustrated in Figure 1-13.

When an interrupt condition occurs, its individual flag is set, even if the interrupt source is disabled at the local, module, or global level. Interrupt flags must be cleared within the user interrupt routine to avoid repeated interrupts from the same source.

Since all interrupts vector to the address contained in the Interrupt Vector (IV) register, the Interrupt Identification Register (IIR) may be used by the interrupt service routine to determine the module source of an interrupt. The IIR contains a bit flag for each peripheral module and one flag associated with all system interrupts; if the bit for a module is set, then an interrupt is pending that was initiated by that module. If a module is capable of generating interrupts for different reasons, then peripheral register bits inside the module provide a means to differentiate among interrupt sources.

The Interrupt Vector (IV) register provides the location of the interrupt service routine. It may be set to any location within program memory. The IV register defaults to 0000h on reset or power-up, so if it is not changed to a different address, the user program must determine whether a jump to 0000h came from a reset or interrupt source.

1.2.4.2 Interrupt System Operation

The interrupt handler hardware responds to any interrupt event when it is enabled. An interrupt event occurs when an interrupt flag is set. All interrupt requests are sampled at the rising edge of the clock and can be serviced by the processor one clock cycle later, assuming the request does not hit the interrupt exception window. The one-cycle stall between detection and acknowledgement/servicing is due to the fact that the current instruction may also be accessing the stack. For this reason, the CPU must allow the current instruction to complete before pushing the stack and vectoring to IV. If an interrupt exception window is generated by the currently executing instruction, the following instruction must be executed, so the interrupt service routine will be delayed an additional cycle.

Interrupt operation in the MAXQ7665/MAXQ7666 CPU is essentially a state machine generated long CALL instruction. When the interrupt handler services an interrupt, it temporarily takes control of the CPU to perform the following sequence of actions:

1) The next instruction fetch from program memory is cancelled.

2) The return address is pushed on to the stack.

3) The INS bit is set to 1 to prevent recursive interrupt calls.

4) The instruction pointer is set to the location of the interrupt service routine (contained in the Interrupt Vector register).

5) The CPU begins executing the interrupt service routine.

Once the interrupt service routine completes, it should use the RETI instruction to return to the main program. Execution of RETI involves the following sequence of actions:

1) The return address is popped off the stack.

2) The INS bit is cleared to 0 to re-enable interrupt handling.

3) The instruction pointer is set to the return address that was popped off the stack.

4) The CPU continues execution of the main program.

Pending interrupt requests will not interrupt an RETI instruction; a new interrupt will be serviced after first being acknowledged in the execution cycle which follows the RETI instruction and then after the standard one stall cycle of interrupt latency. This means there will be at least two cycles between back-to-back interrupts.

1.2.4.3 Synchronous vs. Asynchronous Interrupt Sources

Interrupt sources can be classified as either asynchronous or synchronous. All internal interrupts are synchronous interrupts. An internal interrupt is directly routed to the interrupt handler that can be recognized in one cycle. All external interrupts are asynchronous interrupts by nature. When the device is not in stop mode, asynchronous interrupt sources are passed through a 3-clock sampling/glitch filter circuit before being routed to the interrupt handler. The sampling/glitch filter circuit is running on the undivided source clock (i.e., before PMME, CD1:0-controlled clock divide) such that the number of system clocks required to recognize an asynchronous interrupt request depends upon the system clock divide ratio:

- if the system clock divide ratio is 1, the interrupt request is recognized after 3 system clock;
- if the system clock divide ratio is 2, the interrupt request is recognized after 2 system clock;
- if the system clock divide ratio is 4 or greater, the interrupt request is recognized after 1 system clock;

An interrupt request with a pulse width less than three undivided clock cycles is not recognized. Note that the granularity of interrupt source is at module level. Synchronous interrupts and sampled asynchronous interrupts assigned to the same module produce a single interrupt to the interrupt handler.

External interrupts, when enabled, can be used as switchback sources from power management mode. There is no latency associated with the switchback because the circuit is being clocked by an undivided clock source versus the divide-by-256 system clock. For the same reason, there is no latency for other switchback sources that do not qualify as interrupt sources.
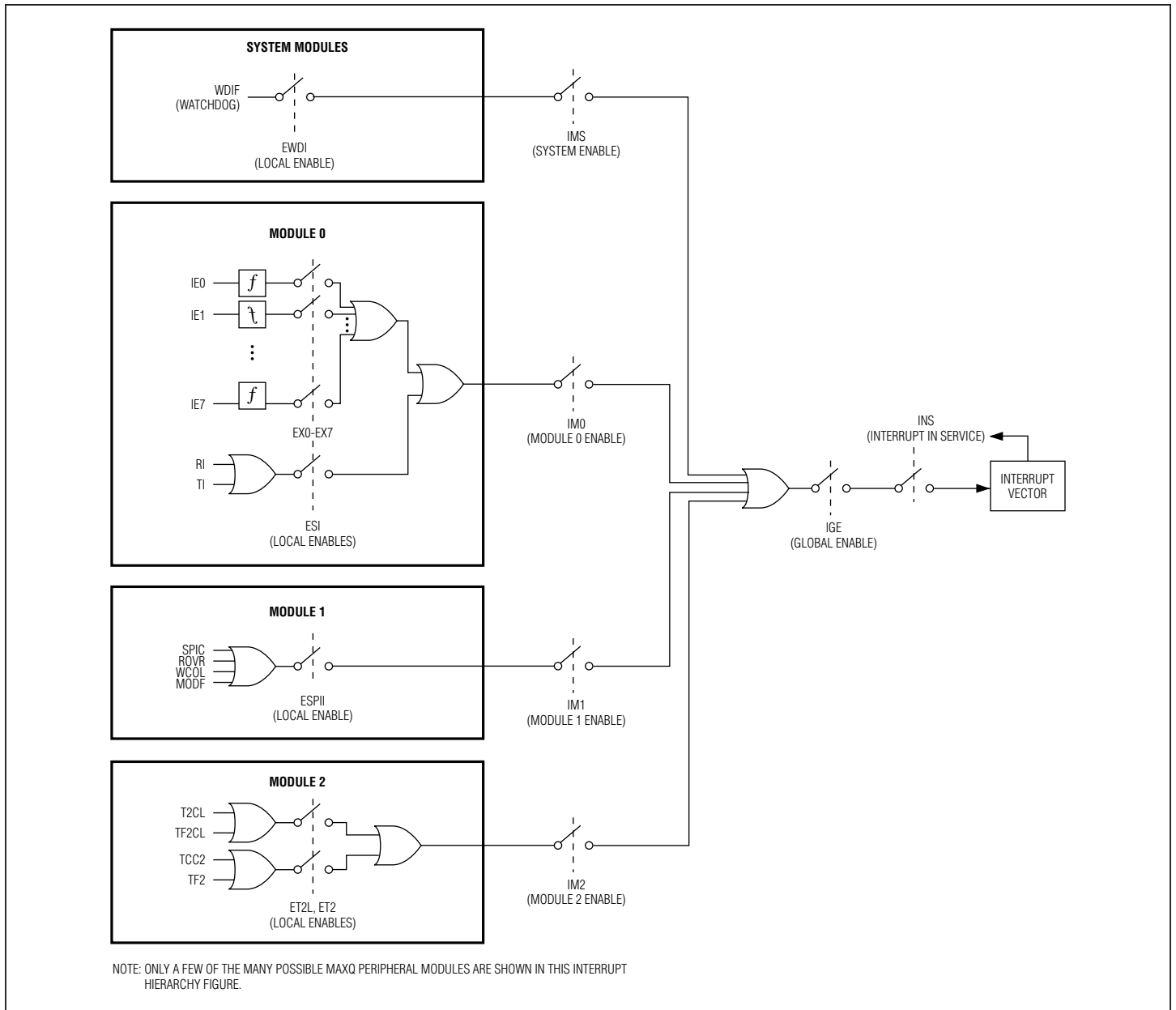
Figure 1-13. MAXQ7665/MAXQ7666 Interrupt Source Hierarchy Example

1.2.4.4 Interrupt Prioritization by Software

All interrupt sources of the MAXQ7665/MAXQ7666 microcontrollers naturally have the same priority. However, when CPU operation vectors to the programmed Interrupt Vector address, the order in which potential interrupt sources are interrogated is left entirely up to the user, as this often depends upon the system design and application requirements. The Interrupt Mask system register provides the ability to knowingly block interrupts from modules considered to be of lesser priority and manually re-enable the interrupt servicing by the CPU (by setting INS = 0). Using this procedure, a given interrupt service routine can continue executing, only to be interrupted by higher priority interrupts. An example demonstrating this software prioritization is provided in *Section 1.3.8: Handling Interrupts*.

1.2.4.5 Interrupt Exception Window

An interrupt exception window is a noninterruptable execution cycle. During this cycle, the interrupt handler does not respond to any interrupt requests. All interrupts that would normally be serviced during an interrupt exception window are delayed until the next execution cycle.

Interrupt exception windows are used when two or more instructions must be executed consecutively without any delays in between. Currently, there is a single condition in the MAXQ7665/MAXQ7666 microcontrollers that causes an interrupt exception window: activation of the prefix (PFX) register.

When the prefix register is activated by writing a value to it, it retains that value only for the next clock cycle. For the prefix value to be used properly by the next instruction, the instruction that sets the prefix value and the instruction that uses it must always be executed back to back. Therefore, writing to the PFX register causes an interrupt exception window on the next cycle. If an interrupt occurs during an interrupt exception window, an additional latency of one cycle in the interrupt handling will be caused as the interrupt will not be serviced until the next cycle.

1.2.4.6 MAXQ7665/MAXQ7666 Interrupt Sources

Table 1-6 lists all possible interrupt sources for the MAXQ7665/MAXQ7666, along with their corresponding module interrupt enable bits, local interrupt enable bits, and interrupt flags.

- Each module interrupt enable bit, when cleared to 0, will block interrupts originating in that module from being acknowledged. When the module interrupt enable bit is set to 1, interrupts from that module are acknowledged (unless the interrupts have been disabled globally).

- Each local interrupt enable bit, when cleared to 0, will disable the corresponding interrupt. When the local interrupt enable bit is set to 1, the interrupt will be triggered whenever the interrupt flag is set to 1 (either by software or hardware).

- All interrupt flag bits cause the corresponding interrupt to trigger when the bit is set to 1. These bits are typically set by hardware and must be cleared by software (generally in the interrupt handler routine).

Note that for an interrupt to fire, the following five conditions must exist:

1) Interrupts must be enabled globally by setting IGE (IC.0) to 1.

2) The module interrupt enable bit for that interrupt source's module must be set to 1.

3) The local interrupt enable bit for that specific interrupt source must be set to 1.

4) The interrupt flag for that interrupt source must be set to 1. Typically, this is done by hardware when the condition that requires interrupt service occurs.

5) The Interrupt In Service (INS) bit must be cleared to 0. This bit is set automatically upon vectoring to the interrupt handler address and cleared automatically upon exit (RETI/POPI), so the only reason to clear this bit manually (inside the interrupt handler routine) is allow nested interrupt handling.

Table 1-6. MAXQ7665/MAXQ7666 Interrupt Sources and Control Bits

INTERRUPT	MODULE ENABLE BIT	LOCAL ENABLE BIT	INTERRUPT FLAG
Watchdog Interrupt	IMS (IMR.7)	EWDI (WDCN.6)	WDIF (WDCN.3)
External Interrupt 0	IM0 (IMR.0)	EX0 (EIE0.0)	IE0 (EIF0.0)
External Interrupt 1	IM0 (IMR.0)	EX1 (EIE0.1)	IE1 (EIF0.1)
External Interrupt 2	IM0 (IMR.0)	EX2 (EIE0.2)	IE2 (EIF0.2)
External Interrupt 3	IM0 (IMR.0)	EX3 (EIE0.3)	IE3 (EIF0.3)
External Interrupt 4	IM0 (IMR.0)	EX4 (EIE0.4)	IE4 (EIF0.4)
External Interrupt 5	IM0 (IMR.0)	EX5 (EIE0.5)	IE5 (EIF0.5)
External Interrupt 6	IM0 (IMR.0)	EX6 (EIE0.6)	IE6 (EIF0.6)
External Interrupt 7	IM0 (IMR.0)	EX7 (EIE0.7)	IE7 (EIF0.7)
Serial Port Receive	IM0 (IMR.0)	ESI (SMD0.2)	RI (SCON0.0)
Serial Port Transmit	IM0 (IMR.0)	ESI (SMD0.2)	TI (SCON0.1)
SPI Mode Fault	IM1 (IMR.1)	ESPII (SPICF.7)	MODF (SPICN.3)
SPI Write Collision	IM1 (IMR.1)	ESPII (SPICF.7)	WCOL (SPICN.4)
SPI Receive Overrun	IM1 (IMR.1)	ESPII (SPICF.7)	ROVR (SPICN.5)
SPI Transfer Complete	IM1 (IMR.1)	ESPII (SPICF.7)	SPIC (SPICN.6)
Timer 0—Low Compare	IM2 (IMR.2)	ET2L (T2CNB0.7)	T2CL (T2CNB0.0)
Timer 0—Low Overflow	IM2 (IMR.2)	ET2L (T2CNB0.7)	TF2L (T2CNB0.2)
Timer 0—Capture/Compare	IM2 (IMR.2)	ET2 (T2CNA0.7)	TCC2 (T2CNB0.1)
Timer 0—Overflow	IM2 (IMR.2)	ET2 (T2CNA0.7)	TF2 (T2CNB0.3)
Timer 1—Low Compare	IM2 (IMR.2)	ET2L (T2CNB1.7)	T2CL (T2CNB1.0)
Timer 1—Low Overflow	IM2 (IMR.2)	ET2L (T2CNB1.7)	TF2L (T2CNB1.2)
Timer 1—Capture/Compare	IM2 (IMR.2)	ET2 (T2CNA1.7)	TCC2 (T2CNB1.1)
Timer 1—Overflow	IM2 (IMR.2)	ET2 (T2CNA1.7)	TF2 (T2CNB1.3)
Timer 2—Low Compare	IM3 (IMR.3)	ET2L (T2CNB2.7)	T2CL (T2CNB2.0)
Timer 2—Low Overflow	IM3 (IMR.3)	ET2L (T2CNB2.7)	TF2L (T2CNB2.2)
Timer 2—Capture/Compare	IM3 (IMR.3)	ET2 (T2CNA2.7)	TCC2 (T2CNB2.1)
Timer 2—Overflow	IM3 (IMR.3)	ET2 (T2CNA2.7)	TF2 (T2CNB2.3)
CAN 0 Message Center 1 Receive	IM4 (IMR.4)	ERI (C0M1C.5)	INTRQ (C0M1C.4)
CAN 0 Message Center 1 Transmit	IM4 (IMR.4)	ETI (C0M1C.6)	INTRQ (C0M1C.4)
CAN 0 Message Center 2 Receive	IM4 (IMR.4)	ERI (C0M2C.5)	INTRQ (C0M2C.4)
CAN 0 Message Center 2 Transmit	IM4 (IMR.4)	ETI (C0M2C.6)	INTRQ (C0M2C.4)

Table 1-6. MAXQ7665/MAXQ7666 Interrupt Sources and Control Bits (continued)

INTERRUPT	MODULE ENABLE BIT	LOCAL ENABLE BIT	INTERRUPT FLAG
CAN 0 Message Center 3 Receive	IM4 (IMR.4)	ERI (C0M3C.5)	INTRQ (C0M3C.4)
CAN 0 Message Center 3 Transmit	IM4 (IMR.4)	ETI (C0M3C.6)	INTRQ (C0M3C.4)
CAN 0 Message Center 4 Receive	IM4 (IMR.4)	ERI (C0M4C.5)	INTRQ (C0M4C.4)
CAN 0 Message Center 4 Transmit	IM4 (IMR.4)	ETI (C0M4C.6)	INTRQ (C0M4C.4)
CAN 0 Message Center 5 Receive	IM4 (IMR.4)	ERI (C0M5C.5)	INTRQ (C0M5C.4)
CAN 0 Message Center 5 Transmit	IM4 (IMR.4)	ETI (C0M5C.6)	INTRQ (C0M5C.4)
CAN 0 Message Center 6 Receive	IM4 (IMR.4)	ERI (C0M6C.5)	INTRQ (C0M6C.4)
CAN 0 Message Center 6 Transmit	IM4 (IMR.4)	ETI (C0M6C.6)	INTRQ (C0M6C.4)
CAN 0 Message Center 7 Receive	IM4 (IMR.4)	ERI (C0M7C.5)	INTRQ (C0M7C.4)
CAN 0 Message Center 7 Transmit	IM4 (IMR.4)	ETI (C0M7C.6)	INTRQ (C0M7C.4)
CAN 0 Message Center 8 Receive	IM4 (IMR.4)	ERI (C0M8C.5)	INTRQ (C0M8C.4)
CAN 0 Message Center 8 Transmit	IM4 (IMR.4)	ETI (C0M8C.6)	INTRQ (C0M8C.4)
CAN 0 Message Center 9 Receive	IM4 (IMR.4)	ERI (C0M9C.5)	INTRQ (C0M9C.4)
CAN 0 Message Center 9 Transmit	IM4 (IMR.4)	ETI (C0M9C.6)	INTRQ (C0M9C.4)
CAN 0 Message Center 10 Receive	IM4 (IMR.4)	ERI (C0M10C.5)	INTRQ (C0M10C.4)
CAN 0 Message Center 10 Transmit	IM4 (IMR.4)	ETI (C0M10C.6)	INTRQ (C0M10C.4)
CAN 0 Message Center 11 Receive	IM4 (IMR.4)	ERI (C0M11C.5)	INTRQ (C0M11C.4)
CAN 0 Message Center 11 Transmit	IM4 (IMR.4)	ETI (C0M11C.6)	INTRQ (C0M11C.4)
CAN 0 Message Center 12 Receive	IM4 (IMR.4)	ERI (C0M12C.5)	INTRQ (C0M12C.4)
CAN 0 Message Center 12 Transmit	IM4 (IMR.4)	ETI (C0M12C.6)	INTRQ (C0M12C.4)
CAN 0 Message Center 13 Receive	IM4 (IMR.4)	ERI (C0M13C.5)	INTRQ (C0M13C.4)
CAN 0 Message Center 13 Transmit	IM4 (IMR.4)	ETI (C0M13C.6)	INTRQ (C0M13C.4)
CAN 0 Message Center 14 Receive	IM4 (IMR.4)	ERI (C0M14C.5)	INTRQ (C0M14C.4)
CAN 0 Message Center 14 Transmit	IM4 (IMR.4)	ETI (C0M14C.6)	INTRQ (C0M14C.4)
CAN 0 Message Center 15 Receive	IM4 (IMR.4)	ERI (C0M15C.5)	INTRQ (C0M15C.4)
CAN 0 Message Center 15 Transmit	IM4 (IMR.4)	ETI (C0M15C.6)	INTRQ (C0M15C.4)
CAN 0 Bus Off Status	IM4 (IMR.4)	ERIE(C0C.7), C0IE (COR.0)	BSS (C0S.7)
CAN 0 Error Count > 96/128 Status	IM4 (IMR.4)	ERIE(C0C.7), C0IE (COR.0)	EC96/128 (C0S.6)
CAN 0 Wake-Up Status	IM4 (IMR.4)	STIE(C0C.6), C0IE (COR.0)	WKS (C0S.5)
CAN 0 Receive Status	IM4 (IMR.4)	STIE(C0C.6), C0IE (COR.0)	RXS (C0S.4)
CAN 0 Transmit Status	IM4 (IMR.4)	STIE(C0C.6), C0IE (COR.0)	TXS (C0S.3)
CAN 0 Bus Error Status	IM4 (IMR.4)	STIE(C0C.6), C0IE (COR.0)	ER2:ER0 (C0S.2:C0S.0)
CAN 0 Bus Activity Status	IM4 (IMR.4)	C0BIE (COR.1)	CAN0BA (COR.7)
ADC Data Ready	IM5 (IMR.5)	ADCIE (AIE.1)	ADCRY (ASR.1)
ADC Overrun	IM5 (IMR.5)	AORIE (AIE.2)	ADCOV (ASR.2)

MAXQ7665/MAXQ7666 User's Guide

Table 1-6. MAXQ7665/MAXQ7666 Interrupt Sources and Control Bits (continued)

INTERRUPT	MODULE ENABLE BIT	LOCAL ENABLE BIT	INTERRUPT FLAG
Digital Brownout	IM5 (IMR.5)	DVBIE (AIE.4)	DVBI (ASR.4)
I/O Voltage Brownout	IM5 (IMR.5)	VIOBIE (AIE.5)	VIOBI (ASR.5)
High-Frequency Oscillator Failure	IM5 (IMR.5)	HFFIE (AIE.6)	HFFINT (ASR.6)

1.3 Programming

The following section provides a programming overview of the MAXQ7665/MAXQ7666. For full details on the instruction set, as well as System Register and Peripheral Register detailed bit descriptions, see the appropriate sections in this user's guide.

1.3.1 Addressing Modes

The instruction set for the MAXQ7665/MAXQ7666 provides three different addressing modes: direct, indirect, and immediate.

The direct addressing mode can be used to specify either source or destination registers, such as:

```
move  A[0], A[1]           ; copy accumulator 1 to accumulator 0
push  A[0]                 ; push accumulator 0 on the stack
add   A[1]                 ; add accumulator 1 to the active accumulator
```

Direct addressing is also used to specify addressable bits within registers.

```
move  C, Acc.0             ; copy bit zero of the active accumulator
                           ;   to the carry flag
move  PO0.3, #1            ; set bit three of port 0 Output register
```

Indirect addressing, in which a register contains a source or destination address, is used only in a few cases.

```
move  @DP[0], A[0]         ; copy accumulator 0 to the data memory
                           ;   location pointed to by data pointer 0
move  A[0], @SP--          ; where @SP-- is used to pop the data pointed to
                           ;   by the stack pointer register
```

Immediate addressing is used to provide values to be directly loaded into registers or used as operands.

```
move  A[0], #10h           ; set accumulator 1 to 10h/16d
```

1.3.2 Prefixing Operations

All instructions on the MAXQ7665/MAXQ7666 are 16 bits long and execute in a single cycle. However, some operations require more data than can be specified in a single cycle or require that high-order register-index bits be set to achieve the desired transfer. In these cases, the prefix register module PFX is loaded with temporary data and/or required register index bits to be used by the following instruction. The PFX module only holds loaded data for a single cycle before it clears to zero.

Instruction prefixing is required for the following operations, which effectively makes them two-cycle operations.

- When providing a 16-bit immediate value for an operation (e.g., loading a 16-bit register, ALU operation, supplying an absolute program branch destination), the PFX module must be loaded in the previous cycle with the high byte of the 16-bit immediate value unless that high byte is zero. One exception to this rule is when supplying an absolute branch destination to 00xxh. In this case, PFX still must be written with 00h. Otherwise, the branch instruction would be considered a relative one instead of the desired absolute branch.

- When selecting registers with indexes greater than 07h within a module as destinations for a transfer or registers with indexes greater than 0Fh within a module as sources, the PFX[n] register must be loaded in the previous cycle. This can be combined with the previous item.

Generally, prefixing operations can be inserted automatically by the assembler as needed, so that (for example)

```
        move   DP[ 0], #1234h
```

actually assembles as

```
        move   PFX[ 0], #12h
        move   DP[ 0], #34h
```

However, the operation

```
        move   DP[ 0], #0055h
```

does not require a prefixing operation even though the register DP[0] is 16-bit. This is because the prefix value defaults to zero, so the line

```
        move   PFX[ 0], #00h
```

is not required.

1.3.3 Reading and Writing Registers

All functions in the MAXQ7665/MAXQ7666 are accessed through registers, either directly or indirectly. This section discusses loading registers with immediate values and transferring values between registers of the same size and different sizes.

1.3.3.1 Loading an 8-Bit Register with an Immediate Value

Any writeable 8-bit register with a subindex from 0h to 7h within its module can be loaded with an immediate value in a single cycle using the MOVE instruction.

```
        move   AP, #05h             ; load accumulator pointer register with 5 hex
```

Writeable 8-bit registers with subindexes 8h and higher can be loaded with an immediate value using MOVE as well, but an additional cycle is required to set the prefix value for the destination.

```
        move   WDCN, #33h           ; assembles to:  move PFX[ 2], #00h
                                     ;                move (WDCN-80h), #33h
```

1.3.3.2 Loading a 16-Bit Register with a 16-Bit Immediate Value

Any writeable 16-bit register with a subindex from 0h to 07h can be loaded with an immediate value in a single cycle if the high byte of that immediate value is zero.

```
        move   LC[ 0], #0010h       ; prefix defaults to zero for high byte
```

If the high byte of that immediate value is not zero or if the 16-bit destination subindex is greater than 7h, an extra cycle is required to load the prefix value for the high byte and/or the high-order register index bits.

```
                                     ; high byte <> #00h
        move   LC[ 0], #0110h        ; assembles to:  move PFX[ 0], #01h
                                     ;                move LC[ 0], #10h
                                     ; destination sub-index > 7h
        move   A[ 8], #0034h         ; assembles to:  move PFX[ 2], #00h
                                     ;                move (A[ 8]-80h), #34h
```

1.3.3.3 Moving Values Between Registers of the Same Size

Moving data between same-size registers can be done in a single-cycle MOVE if the destination register's index is from 0h to 7h and the source register index is between 0h and Fh.

```
        move   A[ 0], A[ 8]         ; copy accumulator 8 to accumulator 0
        move   LC[ 0], LC[ 1]       ; copy loop counter 1 to loop counter 0
```

If the destination register's index is greater than 7h or if the source register index is greater than Fh, prefixing is required.

```
        move   A[ 15], A[ 0]        ; assembles to:  move PFX[ 2], #00h
                                     ;                move (A[ 15]-80h), A[ 0]
```

1.3.3.4 Moving Values Between Registers of Different Sizes

Before covering some transfer scenarios that might arise, a special register must be introduced that will be used in many of these cases. The 16-bit General Register (GR) is expressly provided for performing byte singulation of 16-bit words. The high and low bytes of GR are individually accessible in the GRH and GRL registers respectively. A read-only GRS register makes a byte-swapped version of GR accessible and the GRXL register provides a sign-extended version of GRL.

MAXQ7665/MAXQ7666 User's Guide

1.3.3.4.1 8-Bit Destination ← Low Byte (16-Bit Source)

The simplest transfer possibility would be loading an 8-bit register with the low byte of a 16-bit register. This transfer does not require use of GR and requires a prefix only if the destination or source register are outside of the single cycle write or read regions, 0–7h and 0–Fh, respectively.

```
        move   OFFS, LC[ 0]        ; copy the low byte of LC[ 0]  to the OFFS register
        move   IMR, @DP[ 1]        ; copy the low byte @DP[ 1]  to the IMR register
        move   WDCN, LC[ 0]        ; assembles to: move PFX[ 2], #00h
                                   ;               move (WDCN-80h), LC[ 0]
```

1.3.3.4.2 8-Bit Destination ← High Byte (16-Bit Source)

If, however, we needed to load an 8-bit register with the high byte of a 16-bit source, it would be best to use the GR register. Transferring the 16-bit source to the GR register adds a single cycle.

```
        move   GR, LC[ 0]         ; move LC[ 0]  to the GR register
        move   IC, GRH            ; copy the high byte into the IC register
```

1.3.3.4.3 16-Bit Destination ← Concatenation (8-Bit Source, 8-Bit Source)

Two 8-bit source registers can be concatenated and stored into a 16-bit destination by using the prefix register to hold the high-order byte for the concatenated transfer. An additional cycle may be required if either source byte register index is greater than 0Fh or the 16-bit destination is greater than 07h.

```
        move   PFX[ 0], IC        ; load high order source byte IC into PFX
        move   @DP[ 0], AP        ; store @DP[ 0]  the concatenation of IC:AP

                                  ; 16-bit destination sub-index: dst=08h
                                  ;  8-bit source sub-indexes:
                                  ;  high=10h, low=11h
        move   PFX[ 1], #00h      ;
        move   PFX[ 3], high      ; PFX=00:high
        move   dst, low           ; dst=high:low
```

1.3.3.4.4 Low (16-Bit Destination) ← 8-Bit Source

To modify only the low byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the high byte can be singulated and the low byte written exclusively. An additional cycle is required if the destination index is greater than 0Fh.

```
        move   GR, DP[ 0]         ; move DP[ 0]  to the GR register
        move   PFX[ 0], GRH       ; get the high byte of DP[ 0]  via GRH
        move   DP[ 0], #20h       ; store the new DP[ 0]  value
                                  ; 16-bit destination sub-index: dst=10h
                                  ;  8-bit source sub-index: src=11h
        move   PFX[ 1], #00h      ;
        move   GR, dst            ; read dst word to the GR register
        move   PFX[ 5], GRH       ; get the high byte of dst via GRH
        move   dst, src           ; store the new dst value
```

1.3.3.4.5 High (16-Bit Destination) ← 8-Bit Source

To modify only the high byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the low byte can be singulated and the high byte can be written exclusively. Additional cycles are required if the destination index is greater than 0Fh or if the source index is greater than 0Fh.

```
        move   GR, DP[ 0]         ; move DP[ 0]  to the GR register
        move   PFX[ 0], #20h      ; get the high byte of DP[ 0]  via GRH
        move   DP[ 0], GRL        ; store the new DP[ 0]  value
                                  ; 16-bit destination sub-index: dst=10h
                                  ;  8-bit source sub-index: src=11h
        move   PFX[ 1], #00h      ;
        move   GR, dst            ; read dst word to the GR register
        move   PFX[ 1], #00h
        move   PFX[ 4], src       ; get the new src byte
        move   dst, GRL           ; store the new dst value
```

If the high byte needs to be cleared to 00h, the operation can be shortened by transferring only the GRL byte to the 16-bit destination (example follows):

```
        move  GR, DP[ 0]              ; move DP[ 0]  to the GR register
        move  DP[ 0], GRL            ; store the new DP[ 0]  value, 00h used for high byte
```

1.3.4 Reading and Writing Register Bits

The MOVE instruction can also be used to directly set or clear any one of the lowest 8 bits of a peripheral register in module 0h-5h or a system register in module 8h. The set or clear operation will not affect the upper byte of a 16-bit register that is the target of the set or clear operation. If a set or clear instruction is used on a destination register that does not support this type of operation, the register high byte will be written with the prefix data and the low byte will be written with the bit mask (i.e. all 0's with a single 1 for the set bit operation or all ones with a single 0 for the clear bit operation).

Register bits can be set or cleared individually using the MOVE instruction as follows.

```
        move  IGE, #1               ; set IGE (Interrupt Global Enable) bit
        move  APC.6, #0             ; clear IDS bit (APC.6)
```

As with other instructions, prefixing is required to select destination registers beyond index 07h.

The MOVE instruction may also be used to transfer any one of the lowest 8 bits from a register source or any bit of the active accumulator (Acc) to the Carry flag. There is no restriction on the source register module for the 'MOVE C, src.bit' instruction.

```
        move  C, IIR.3             ; copy IIR.3 to Carry
        move  C, Acc.7             ; copy Acc.7 to Carry
```

Prefixing is required to select source registers beyond index 15h.

1.3.5 Using the Arithmetic and Logic Unit

The MAXQ7665/MAXQ7666 provide a 16-bit (MAXQ20) ALU, which allows operations to be performed between the active accumulator and any other register. The ALU configuration provides 16 accumulator registers that are also 16 bits (MAXQ20) wide, of which any one may be selected as the active accumulator.

1.3.5.1 Selecting the Active Accumulator

Any of the 16 accumulator registers A[0] through A[15] may be selected as the active accumulator by setting the low four bits of the Accumulator Pointer Register (AP) to the index of the accumulator register you want to select.

```
        move  AP, #01h             ; select A[ 1]  as the active accumulator
        move  AP, #0Fh             ; select A[ 15]  as the active accumulator
```

The current active accumulator can be accessed as the Acc register, which is also the register used as the implicit destination for all arithmetic and logical operations.

```
        move  A[ 0], #55h          ; set A[ 0] =   55 hex (MAXQ10)
                                   ;            = 0055 hex (MAXQ20)
        move  AP, #00h             ; select A[ 0]  as active accumulator
        move  Acc, #55h            ; set A[ 0] =   55 hex (MAXQ10)
                                   ;            = 0055 hex (MAXQ20)
```

1.3.5.2 Enabling Auto-Increment and Auto-Decrement

The accumulator pointer AP can be set to automatically increment or decrement after each arithmetic or logical operation. This is useful for operations involving a number of accumulator registers, such as adding or subtracting two multibyte integers.

If auto-increment/decrement is enabled, the AP register increments or decrements after any of the following operations:

- ADD src (Add source to active accumulator)
- ADDC src (Add source to active accumulator with carry)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source from active accumulator with borrow)
- AND src (Logical AND active accumulator with source)
- OR src (Logical OR active accumulator with source)

- XOR src (Logical XOR active accumulator with source)
- CPL (Bit-wise complement active accumulator)
- NEG (Negate active accumulator)
- SLA (Arithmetic shift left on active accumulator)
- SLA2 (Arithmetic shift left active accumulator two bit positions)
- SLA4 (Arithmetic shift left active accumulator four bit positions)
- SRA (Arithmetic shift right on active accumulator)
- SRA2 (Arithmetic shift right active accumulator two bit positions)
- SRA4 (Arithmetic shift right active accumulator four bit positions)
- RL (Rotate active accumulator left)
- RLC (Rotate active accumulator left through Carry flag)
- RR (Rotate active accumulator right)
- RRC (Rotate active accumulator right through Carry flag)
- SR (Logical shift active accumulator right)
- MOVE Acc, src (Copy data from source to active accumulator)
- MOVE dst, Acc (Copy data from active accumulator to destination)
- MOVE Acc, Acc (Recirculation of active accumulator contents)
- XCHN (Exchange nibbles within each byte of active accumulator)
- XCH (Exchange active accumulator bytes)

The active accumulator may not be the source in any instruction where it is also the implicit destination.

There is an additional notation that can be used to refer to the active accumulator for the instruction "MOVE dst, Acc." If the instruction is instead written as "MOVE dst, A[AP]," the source value is still the active accumulator, but no AP auto-increment or auto-decrement function will take place, even if this function is enabled. Note that the active accumulator may not be the destination for the MOVE dst, A[AP] instruction (i.e., MOVE Acc, A[AP] is prohibited).

So, the two instructions

```
move  A[ 7] , Acc
move  A[ 7] , A[ AP]
```

are equivalent, except that the first instruction triggers auto-increment/decrement (if it is enabled), while the second one will never do so.

The Accumulator Pointer Control Register (APC) controls the auto-increment/decrement mode as well as selects the range of bits (modulo) in the AP register that will be incremented or decremented. There are nine different unique settings for the APC register, as listed in Table 1-7.

Table 1-7. Accumulator Pointer Control Register Settings

APC.2 (MOD2)	APC.1 (MOD1)	APC.0 (MOD0)	APC.6 (IDS)	APC	AUTO-INCREMENT/DECREMENT SETTING
0	0	0	X	00h	No auto-increment/decrement (default mode)
0	0	1	0	01h	Increment bit 0 of AP (modulo 2)
0	0	1	1	41h	Decrement bit 0 of AP (modulo 2)
0	1	0	0	02h	Increment bits [1:0] of AP (modulo 4)
0	1	0	1	42h	Decrement bits [1:0] of AP (modulo 4)
0	1	1	0	03h	Increment bits [2:0] of AP (modulo 8)
0	1	1	1	43h	Decrement bits [2:0] of AP (modulo 8)
1	0	0	0	04h	Increment all 4 bits of AP (modulo 16)
1	0	0	1	44h	Decrement all 4 bits of AP (modulo 16)

For the modulo increment or decrement operation, the selected range of bits in AP are incremented or decremented. However, if these bits roll over or under, they simply wrap around without affecting the remaining bits in the accumulator pointer. So, the operations can be defined as follows:

- Increment modulo 2: AP = AP[3:1] + ((AP[0] + 1) mod 2)

- Decrement modulo 2: AP = AP[3:1] + ((AP[0] - 1) mod 2)

- Increment modulo 4: AP = AP[3:2] + ((AP[1:0] + 1) mod 4)

- Decrement modulo 4: AP = AP[3:2] + ((AP[1:0] - 1) mod 4)

- Increment modulo 8: AP = AP[3] + ((AP[2:0] + 1) mod 8)

- Decrement modulo 8: AP = AP[3] + ((AP[2:0] - 1) mod 8)

- Increment modulo 16: AP = (AP + 1) mod 16

- Decrement modulo 16:   AP = (AP - 1) mod 16

For this example, assume that all 16 accumulator registers are initially set to zero.

```
move   AP, #02h          ; select A[ 2] as active accumulator
move   APC, #02h         ; auto-increment AP[ 1:0] modulo 4
                         ; AP   A[ 0]   A[ 1]   A[ 2]   A[ 3]
                         ; 02   0000    0000    0000    0000
add    #01h              ; 03   0000    0000    0001    0000
add    #02h              ; 00   0000    0000    0001    0002
add    #03h              ; 01   0003    0000    0001    0002
add    #04h              ; 02   0003    0004    0001    0002
add    #05h              ; 03   0003    0004    0006    0002
```

1.3.5.3 ALU Operations Using the Active Accumulator and a Source

The following arithmetic and logical operations can use any register or immediate value as a source. The active accumulator Acc is always used as the second operand and the implicit destination. Also, Acc may not be used as the source for any of these operations.

```
add    A[ 4]             ; Acc = Acc + A[ 4]
addc   #32h              ; Acc = Acc + 0032h + Carry

sub    A[ 15]            ; Acc = Acc − A[ 15]
subb   A[ 1]             ; Acc = Acc − A[ 1] - Carry
cmp    #00h              ; If (Acc == 0000h), set Equals flag

and    A[ 0]             ; Acc = Acc AND A[ 0]
or     #55h              ; Acc = Acc OR #0055h

xor    A[ 1]             ; Acc = Acc XOR A[ 1]
```

MAXQ7665/MAXQ7666 User's Guide

1.3.5.4 ALU Operations Using Only the Active Accumulator
The following arithmetic and logical operations operate only on the active accumulator.

```
cpl                          ; Acc = NOT Acc
neg                          ; Acc = (NOT Acc) + 1
rl                           ; Rotate accumulator left (not using Carry)
rlc                          ; Rotate accumulator left through Carry
rr                           ; Rotate accumulator right (not using Carry)
rrc                          ; Rotate accumulator right through Carry
sla                          ; Shift accumulator left arithmetically once
sla2                         ; Shift accumulator left arithmetically twice
sla4                         ; Shift accumulator left arithmetically four times
sr                           ; Shift accumulator right, set Carry to Acc.0,
                             ; set Acc.15 to zero (MAXQ20)
sra                          ; Shift accumulator right arithmetically once
sra2                         ; Shift accumulator right arithmetically twice
sra4                         ; Shift accumulator right arithmetically four times
xchn                         ; Swap low and high nibbles of each Acc byte
xch   (MAXQ20 only)          ; Swap low byte and high byte of Acc
```

1.3.5.5 ALU Bit Operations Using Only the Active Accumulator
The following operations operate on single bits of the current active accumulator in conjunction with the Carry flag. Any of these operations may use an Acc bit from 0 to 15.

```
move  C, Acc.0               ; copy bit 0 of accumulator to Carry
move  Acc.5, C               ; copy Carry to bit 5 of accumulator
and   Acc.3                  ; Acc.3 = Acc.3 AND Carry
or    Acc.0                  ; Acc.0 = Acc.0 OR Carry
xor   Acc.1                  ; Acc.1 = Acc.1 OR Carry
```

None of the above bit operations cause the auto-increment, auto-decrement, or modulo operations defined by the accumulator pointer control (APC) register.

1.3.5.6 Example: Adding Two 4-Byte Numbers Using Auto-Increment
```
move  A[0], #5678h           ; First number - 12345678h
move  A[1], #1234h
move  A[2], #0AAAAh          ; Second number - 0AAAAAAAh
move  A[3], #0AAAh
move  APC, #81h              ; Active Acc = A[0], increment low bit = mod 2
add   A[2]                   ; A[0] = 5678h + AAAAh = 0122h + Carry
addc  A[3]                   ; A[1] = 1234h + AAAh + 1 = 1CDFh
; 12345678h + 0AAAAAAAh = 1CDF0122h
```

1.3.6 Processor Status Flag Operations

The Processor Status Flag (PSF) register contains five flags that are used to indicate and store the results of arithmetic and logical operations, four of which can also be used for conditional program branching.

1.3.6.1 Sign Flag
The Sign flag (PSF.6) reflects the current state of the high bit of the active accumulator (Acc.15 for the MAXQ20). If signed arithmetic is being used, this flag indicates whether the value in the accumulator is positive or negative.

Since the Sign flag is a dynamic reflection of the high bit of the active accumulator, any instruction that changes the value in the active accumulator can potentially change the value of the Sign flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Sign flag.

The following operation uses the Sign flag:

- JUMP S, src (Jump if Sign flag is set)

1.3.6.2 Zero Flag

The Zero flag (PSF.7) is a dynamic flag that reflects the current state of the active accumulator Acc. If all bits in the active accumulator are zero, the Zero flag equals 1. Otherwise, it equals 0.

Since the Zero flag is a dynamic reflection of (Acc = 0), any instruction that changes the value in the active accumulator can potentially change the value of the Zero flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Zero flag.

The following operations use the Zero flag:

- JUMP Z, src (Jump if Zero flag is set)
- JUMP NZ, src (Jump if Zero flag is cleared)

1.3.6.3 Equals Flag

The Equals flag (PSF.0) is a static flag set by the CMP instruction. When the source given to the CMP instruction is equal to the active accumulator, the Equals flag is set to 1. When the source is different from the active accumulator, the Equals flag is cleared to 0.

The following instructions use the value of the Equals flag. Please note that the 'src' for the JUMP E/NE instructions must be immediate.

- JUMP E, src (Jump if Equals flag is set)
- JUMP NE, src (Jump if Equals flag is cleared)

In addition to the CMP instruction, any instruction using PSF as the destination can alter the Equals flag.

1.3.6.4 Carry Flag

The Carry flag (PSF.1) is a static flag indicating that a carry or borrow bit resulted from the last ADD/ADDC or SUB/SUBB operation. Unlike the other status flags, it can be set or cleared explicitly and is also used as a generic bit operand by many other instructions.

The following instructions can alter the Carry flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)
- SLA, SLA2, SLA4 (Arithmetic shift left active accumulator)
- SRA, SRA2, SRA4 (Arithmetic shift right active accumulator)
- SR (Shift active accumulator right)
- RLC/RRC (Rotate active accumulator left / right through Carry)
- MOVE C, Acc.<b> (Set Carry to selected active accumulator bit)
- MOVE C, #i (Explicitly set, i = 1, or clear, i = 0, the Carry flag)
- CPL C (Complement Carry)
- AND Acc.<b>
- OR Acc.<b>
- XOR Acc.<b>
- MOVE C, src.<b> (Copy bit addressable register bit to Carry)
- any instruction using PSF as the destination

The following instructions use the value of the Carry flag:

- ADDC src (Add source and Carry to active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)
- RLC/RRC (Rotate active accumulator left/right through Carry)
- CPL C (Complement Carry)

- MOVE Acc.<b>, C (Set selected active accumulator bit to Carry)
- AND Acc.<b> (Carry = Carry AND selected active accumulator bit)
- OR Acc.<b> (Carry = Carry OR selected active accumulator bit)
- XOR Acc.<b> (Carry = Carry XOR selected active accumulator bit)
- JUMP C, src (Jump if Carry flag is set)
- JUMP NC, src (Jump if Carry flag is cleared)

1.3.6.5 Overflow Flag

The Overflow flag (PSF.2) is a static flag indicating that the carry or borrow bit (Carry status Flag) resulting from the last ADD/ADDC or SUB/SUBB operation did not match the carry or borrow of the high order bit of the active accumulator. The overflow flag is useful when performing signed arithmetic operations.

The following instructions can alter the Overflow flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)

1.3.7 Controlling Program Flow

The MAXQ7665/MAXQ7666 provide several options to control program flow and branching. Jumps may be unconditional, conditional, relative, or absolute. Subroutine calls store the return address on the hardware stack for later return. Built-in counters and address registers are provided to control looping operations.

1.3.7.1 Obtaining the Next Execution Address

The address of the next instruction to be executed can be read at any time by reading the Instruction Pointer (IP) register. This can be particularly useful for initializing loops. Note that the value returned is actually the address of the current instruction plus 1, so this will be the address of the next instruction executed as long as the current instruction does not cause a jump.

1.3.7.2 Unconditional Jumps

An unconditional jump can be relative (IP +127/-128 words) or absolute (to anywhere in program space). Relative jumps must use an 8-bit immediate operand, such as

```
Label1:                          ; must be within +127/-128 words of the JUMP
...
jump  Label1
```

Absolute jumps can use a 16-bit immediate operand, a 16-bit register, or an 8-bit register.

```
jump  LongJump               ; assembles to: move PFX[ 0], #high(LongJump)
                             ;                jump        #low(LongJump)
jump  DP[ 0]                 ; absolute jump to the address in DP[ 0]
```

If an 8-bit register is used as the jump destination, the prefix value is used as the high byte of the address and the register is used as the low byte.

1.3.7.3 Conditional Jumps

Conditional jumps transfer program execution based on the value of one of the status flags (C, E, Z, S). Except where noted for JUMP E and JUMP NE, the absolute and relative operands allowed are the same as for the unconditional JUMP command.

```
        jump c, Label1            ; jump to Label1 if Carry is set
        jump nc, LongJump         ; jump to LongJump if Carry is not set
        jump z, LC[ 0]            ; jump to 16-bit register destination if
                                  ;    Zero is set
        jump nz, Label1           ; jump to Label1 if Zero is not set (Acc<>0)
        jump s, A[ 2]             ; jump to A[ 2] if Sign flag is set
        jump e, Label1            ; jump to Label1 if Equal is set
        jump ne, Label1           ; jump to Label1 if Equal is cleared
```

JUMP E and JUMP NE may only use immediate destinations.

1.3.7.4 Calling Subroutines

The CALL instruction works the same as the unconditional JUMP, except that the next execution address is pushed on the stack before transferring program execution to the branch address. The RET instruction is used to return from a normal call, and RETI is used to return from an interrupt handler routine.

```
        call   Label1             ; if Label1 is relative,
                                  ; assembles to : call #immediate
        call   LongCall           ; assembles to:  move PFX[ 0], #high(LongCall)
                                  ;                call #low(LongCall)
        call   LC[ 0]             ; call to address in LC[ 0]
    LongCall:
        ret                       ; return from subroutine
```

1.3.7.5 Looping Operations

Looping over a section of code can be performed by using the conditional jump instructions. However, there is built-in functionality, in the form of the 'DJNZ LC[n], src' instruction, to support faster, more compact looping code with separate loop counters. The 16-bit registers LC[0], and LC[1] are used to store these loop counts. The 'DJNZ LC[n], src' instruction automatically decrements the associated loop counter register and jumps to the loop address specified by src if the loop counter has not reached 0.

To initialize a loop, set the LC[n] register to the count you wish to use before entering the loop's main body.

The desired loop address should be supplied in the src operand of the 'DJNZ LC[n], src' instruction. When the supplied loop address is relative (+127/-128 words) to the DJNZ LC[n] instruction, as is typically the case, the assembler automatically calculates the relative offset and inserts this immediate value in the object code.

```
        move  LC[ 1], #10h        ; loop 16 times
    LoopTop:                      ; loop addr relative to djnz LC[ n],src instruction
        call  LoopSub
        djnz  LC[ 1], LoopTop     ; decrement LC[ 1] and jump if nonzero
```

When the supplied loop address is outside the relative jump range, the prefix register (PFX[0]) is used to supply the high byte of the loop address as required.

```
        move  LC[ 1], #10h        ; loop 16 times
    LoopTop:                      ; loop addr not relative to djnz LC[ n],src
        call  LoopSub
        ...
        djnz  LC[ 1], LoopTop     ; decrement LC[ 1] and jump if nonzero
                                  ; assembles to: move PFX[ 0], #high(LoopTop)
                                  ;               djnz LC[ 1], #low(LoopTop)
```

If loop execution speed is critical and a relative jump cannot be used, one might consider preloading an internal 16-bit register with the src loop address for the 'DJNZ LC[n], src' loop. This ensures that the prefix register will not be needed to supply the loop address and always yields the fastest execution of the DJNZ instruction.

```
        move  LC[ 0], #LoopTop       ; using LC[ 0] as address holding register
                                     ; assembles to: move PFX[ 0], #high(LoopTop)
                                     ;               move LC[ 0], #low(LoopTop)
        move  LC[ 1], #10h           ; loop 16 times
        ...
   LoopTop:                          ; loop address not relative to djnz LC[ n],src
        call  LoopSub
        ...
        djnz  LC[ 1], LC[ 0]         ; decrement LC[ 1] and jump if nonzero
```

If opting to preload the loop address to an internal 16-bit register, the most time and code efficient means is by performing the load in the instruction just prior to the top of the loop:

```
        move  LC[ 1], #10h           ; Set loop counter to 16
        move  LC[ 0], IP             ; Set loop address to the next address
   LoopTop:                          ; loop addr not relative to djnz LC[ n],src
        ...
```

1.3.7.6 Conditional Returns

Similar to the conditional jumps, the MAXQ7665/MAXQ7666 microcontrollers also support a set of conditional return operations. Based upon the value of one of the status flags, the CPU can conditionally pop the stack and begin execution at the address popped from the stack. If the condition is not true, the conditional return instruction does not pop the stack and does not change the instruction pointer. The following conditional return operations are supported:

```
        RET C                        ; if C=1, a RET is executed
        RET NC                       ; if C=0, a RET is executed
        RET Z                        ; if Z=1 (Acc=00h), a RET is executed
        RET NZ                       ; if Z=0 (Acc<>00h), a RET is executed
        RET S                        ; if S=1, a RET is executed
```

1.3.8 Handling Interrupts

Handling interrupts in the MAXQ7665/MAXQ7666 is a three-part process. First, the location of the interrupt handling routine must be set by writing the address to the 16-bit Interrupt Vector (IV) register. This register defaults to 0000h on reset, but this will usually not be the desired location since this will often be the location of reset/power-up code.

```
        move  IV, IntHandler         ; move PFX[ 0], #high(IntHandler)
                                     ; move IV, #low(IntHandler)
                                     ; PFX[ 0] write not needed if IntHandler addr=00xxh
```

Next, the interrupt must be enabled. For any interrupts to be handled, the IGE bit in the Interrupt and Control register (IC) must first be set to 1. Next, the interrupt itself must be enabled at the module level and locally within the module itself. The module interrupt enable is located in the Interrupt Mask register, while the location of the local interrupt enable will vary depending on the module in which the interrupt source is located.

Once the interrupt handler receives the interrupt, the Interrupt in Service (INS) bit will be set by hardware to block further interrupts, and execution control is transferred to the interrupt service routine. Within the interrupt service routine, the source of the interrupt must be determined. Since all interrupts go to the same interrupt service routine, the Interrupt Identification Register (IIR) must be examined to determine which module initiated the interrupt. For example, the II0 (IIR.0) bit will be set if there is a pending interrupt from module 0. These bits cannot be cleared directly; instead, the appropriate bit flag in the module must be cleared once the interrupt is handled.

INS is set automatically on entry to the interrupt handler and cleared automatically on exit (RETI).

```
IntHandler:
        push  PSF                 ; save C since used in identification process
        move  C, IIR.X            ; check highest priority flag in IIR
        jump  C, ISR_X            ; if IIR.X is set, interrupt from module X
        move  C, IIR.Y            ; check next highest priority int source
        jump  C, ISR_Y            ; if IIR.Y is set, interrupt from module Y
        ...
ISR_X:
        ...
        reti
```

To support high priority interrupts while servicing another interrupt source, the IMR register may be used to create a user-defined prioritization. The IMR mask register should not be utilized when the highest priority interrupt is being serviced because the highest priority interrupt should never be interrupted. This is default condition when a hardware branch is made the Interrupt Vector address (INS is set to 1 by hardware and all other interrupt sources are blocked). The code below demonstrates how to use IMR to allow other interrupts.

```
ISR_Z:
        pop   PSF                 ; restore PSF
        push  IMR                 ; save current interrupt mask
        move  IMR, #int_mask      ; new mask to allow only higher priority ints
        move  INS, #0             ; re-enable interrupts
        ...
        (interrupt servicing code)
        ...
        pop   IMR                 ; restore previous interrupt mask
        ret                       ; back to code or lower priority interrupt
```

Please note that configuring a given IMR register mask bit to 0 only prevents interrupt conditions from the corresponding module or system from generating an interrupt request. Configuring an IMR mask bit to 0 does not prevent the corresponding IIR system or module identification flag from being set. This means that when using the IMR mask register functionality to block interrupts, there may be cases when both the mask (IMR.x) and identifier (IIR.x) bits should be considered when determining if the corresponding peripheral should be serviced.

1.3.8.1 Conditional Return from Interrupt

Similar to the conditional returns, the MAXQ7665/MAXQ7666 microcontrollers also support a set of conditional return from interrupt operations. Based upon the value of one of the status flags, the CPU can conditionally pop the stack, clear the INS bit to 0, and begin execution at the address popped from the stack. If the condition is not true, the conditional return from interrupt instruction leaves the INS bit unchanged, does not pop the stack and does not change the instruction pointer. The following conditional return from interrupt operations are supported:

```
        RETI C                    ; if C=1, a RETI is executed
        RETI NC                   ; if C=0, a RETI is executed
        RETI Z                    ; if Z=1 (Acc=00h), a RETI is executed
        RETI NZ                   ; if Z=0 (Acc<>00h), a RETI is executed
        RETI S                    ; if S=1, a RETI is executed
```

1.3.9 Accessing the Stack

The hardware stack is used automatically by the CALL, RET and RETI instructions, but it can also be used explicitly to store and retrieve data. All values stored on the stack are 16 bits wide.

The PUSH instruction increments the stack pointer SP and then stores a value on the stack. When pushing a 16-bit value onto the stack, the entire value is stored. However, when pushing an 8-bit value onto the stack, the high byte stored on the stack comes from the prefix register. The @++SP stack access mnemonic is the associated destination specifier that generates this push behavior, thus the following two instruction sequences are equivalent:

```
        move  PFX[0], IC
        push  PSF                 ; stored on stack: IC:PSF

        move  PFX[0], IC
        move  @++SP, PSF          ; stored on stack: IC:PSF
```

The POP instruction removes a value from the stack and then decrements the stack pointer. The @SP-- stack access mnemonic is the associated source specifier that generates this behavior, thus the following two instructions are equivalent:

```
pop   PSF
move  PSF, @SP--
```

The POPI instruction is equivalent to the POP instruction but additionally clears the INS bit to 0. Thus, the following two instructions would be equivalent:

```
popi  IP
reti
```

The @SP-- mnemonic can be used by the MAXQ microcontroller so that stack values may be used directly by ALU operations (e.g. ADD src, XOR src, etc.) without requiring that the value be first popped into an intermediate register or accumulator.

```
add   @SP--              ; sum the last three words pushed onto the stack
add   @SP--              ;  with Acc, disregarding overflow
add   @SP--
```

The stack pointer SP can be set explicitly, however only those least significant bits needed to represent the stack depth for the associated MAXQ device are used. For a MAXQ device that has a stack depth of 16 words, only the lowest four bits are used and setting SP to 0Fh will return it to its reset state.

Since the stack is 16 bits wide, it is possible to store two 8-bit register values on it in a single location. This allows more efficient use of the stack if it is being used to save and restore registers at the start and end of a subroutine.

```
SubOne:
      move  PFX[ 0], IC
      push  PSF                 ; store IC:PSF on the stack
      ...
      pop   GR                  ; 16-bit register
      move  IC, GRH             ; IC was stored as high byte
      move  PSF, GRL            ; PSF was stored as low byte
      ret
```

1.3.10 Accessing Data Memory

Data memory is accessed through the data pointer registers DP[0] and DP[1] or the Frame Pointer BP[OFFS]. Once one of these registers is set to a location in data memory, that location can be read or written as follows, using the mnemonic @DP[0], @DP[1], or @BP[OFFS] as a source or destination.

```
      move  DP[ 0], #0000h      ; set pointer to location 0000h
      move  A[ 0], @DP[ 0]      ; read from data memory
      move  @DP[ 0], #55h       ; write to data memory
```

Either of the data pointers may be post-incremented or post-decremented following any read or may be pre-incremented or pre-decremented before any write access by using the following syntax.

```
      move  A[ 0], @DP[ 0]++    ; increment DP[ 0] after read
      move  @++DP[ 0], A[ 1]    ; increment DP[ 0] before write
      move  A[ 5], @DP[ 1]--    ; decrement DP[ 1] after read
      move  @--DP[ 1], #00h     ; decrement DP[ 1] before write
```

The Frame Pointer (BP[OFFS]) is actually composed of a base pointer (BP) and an offset from the base pointer (OFFS). For the frame pointer, the offset register (OFFS) is the target of any increment or decrement operation. The base pointer (BP) is unaffected by increment and decrement operations on the Frame Pointer. Similar to DP[n], the OFFS register may be pre-incremented/decremented when writing to data memory and may be post-incremented/decremented when reading from data memory.

```
      move  A[ 0], @BP[ OFFS--]   ; decrement OFFS after read
      move  @BP[ ++OFFS], A[ 1]   ; increment OFFS before write
```

All three data pointers support both byte and word access to data memory. Each data pointer has its own word/byte select (WBSn) special-function register bit to control the access mode associated with the data pointer. These three register bits (WBS2, which controls BP[OFFS] access; WBS1, which controls DP[1] access; and WBS0, which controls DP[0] access) reside in the Data Pointer Control (DPC) register. When a given WBSn control bit is configured to 1, the associated pointer is operated in the word access mode. When the WBSn bit is configured to 0, the pointer is operated in the byte access mode. Word access mode allows addressing of 64kWords of memory while byte access mode allows addressing of 64kBytes of memory.

Each data pointer (DP[n]) and Frame Pointer base (BP) register is actually implemented internally as a 17-bit register (e.g., 16:0). The Frame Pointer offset register (OFFS) is implemented internally as a 9-bit register (e.g., 8:0). The WBSn bit for the respective pointer controls whether the highest 16 bits (16:1) of the pointer are in use, as is the case for word mode (WBSn = 1) or whether the lowest 16 bits (15:0) are in use, as will be the case for byte mode (WBSn = 0). The WBS2 bit also controls whether the high 8 bits (8:1) of the offset register are in use (WBS2 = 1) or the low 8 bits (7:0) are used (WBS2 = 0). All data pointer register reads, writes, auto-increment/decrement operations occur with respect to the current WBSn selection. Data pointer increment and decrement operations only affect those bits specific to the current word or byte addressing mode (e.g., incrementing a byte mode data pointer from FFFFh does not carry into the internal high order bit that is utilized only for word mode data pointer access). Switching from byte to word access mode or vice versa does not alter the data pointer contents. Therefore, it is important to maintain the consistency of data pointer address value within the given access mode.

```
move   DPC, #0              ; DP[ 0]  in byte mode
move   DP[ 0], #2345h       ; DP[ 0]=2345h (byte mode)
                            ; internal  bits 15:0 loaded
move   DPC, #4              ; DP[ 0]  in word mode
move   DP[ 0], #2345h       ; DP[ 0]=2345h (word mode)
                            ; internal bits 16:1 loaded
move   DPC, #0              ; DP[ 0]  in byte mode
move   GR, DP[ 0]           ; GR = 468Bh (looking at bits 15:0)
```

The three pointers share a single read/write port on the data memory and thus, the user must knowingly activate a desired pointer before using it for data memory read operations. This can be done explicitly using the data pointer select bits (SDPS1:0; DPC.1:0), or implicitly by writing to the DP[n], BP, or OFFS registers as shown below. Any indirect memory write operation using a data pointer will set the SDPS bits, thus activating the write pointer as the active source pointer.

```
move   DPC, #2              ; (explicit) selection of FP as the pointer
move   DP[ 1], DP[ 1]       ; (implicit) selection of DP[ 1]; set SDPS1:0=01b
move   OFFS, src            ; (implicit) selection of FP; set SDPS1=1
move   @DP[ 0], src         ; (implicit) selection of DP[ 0]; set SDPS1:0=00b
```

Once the pointer selection has been made, it will remain in effect until:

- the source data pointer select bits are changed via the explicit or implicit methods described above (i.e., another data pointer is selected for use)

- the memory to which the active source data pointer is addressing is enabled for code fetching using the Instruction Pointer, or

- a memory write operation is performed using a data pointer other than the current active source pointer.

```
move   DP[ 1], DP[ 1]       ; select DP[ 1]  as the active pointer
move   dst, @DP[ 1]         ; read from pointer
move   @DP[ 1], src         ; write using a data pointer
                            ; DP[ 0]  is needed
move   DP[ 0], DP[ 0]       ; select DP[ 0]  as the active pointer
```

To simplify data pointer increment/decrement operations without disturbing register data, a virtual NUL destination has been assigned to system module 6, subindex 7 to serve as a bit bucket. Data pointer increment/decrement operations can be done as follows without altering the contents of any other register:

```
move   NUL, @DP[ 0] ++      ; increment DP[ 0]
move   NUL, @DP[ 0] --      ; decrement DP[ 0]
```

The following data pointer related instructions are invalid:

```
move @++DP[ 0], @DP[ 0] ++
move @++DP[ 1], @DP[ 1] ++
move @BP[ ++Offs], @BP[ Offs++]
move @--DP[ 0], @DP[ 0] --
move @--DP[ 1], @DP[ 1] --
move @BP[ --Offs], @BP[ Offs--]
move @++DP[ 0], @DP[ 0] --
move @++DP[ 1], @DP[ 1] --
move @BP[ ++Offs], @BP[ Offs--]
move @--DP[ 0], @DP[ 0] ++
move @--DP[ 1], @DP[ 1] ++
```

```
move @BP[ --Offs], @BP[ Offs++]
move @DP[ 0], @DP[ 0]++
move @DP[ 1], @DP[ 1]++
move @BP[ Offs], @BP[ Offs++]
move @DP[ 0], @DP[ 0]--
move @DP[ 1], @DP[ 1]--
move @BP[ Offs], @BP[ Offs--]
move DP[ 0], @DP[ 0]++
move DP[ 0], @DP[ 0]--
move DP[ 1], @DP[ 1]++
move DP[ 1], @DP[ 1]--
move Offs, @BP[ Offs--]
move Offs, @BP[ Offs++]
```

1.4 System Register Descriptions

The MAXQ7665/MAXQ7666 system register map is shown in Table 1-8. The system register bit functions and reset value are shown in Table 1-9. Those registers defined in the MAXQ7665/MAXQ7666 system register map are described in the following sections. The address for each register are given in the format *module[index]*, where *module* is the module specifier from 8h to Fh and *index* is the register subindex from 0h to Fh.

Table 1-8. MAXQ7665/MAXQ7666 System Register Map

CYCLES TO READ	CYCLES TO WRITE	REGISTER INDEX	MODULE NAME (BASE SPECIFIER)						
			AP (8h)	A (9h)	PFX (Bh)	IP (Ch)	SP (Dh)	DPC (Eh)	DP (Fh)
1	1	0h	AP	**A[0]**	**PFX[0]**	IP			
1	1	1h	APC	**A[1]**	**PFX[1]**		**SP**		
1	1	2h		**A[2]**	**PFX[2]**		**IV**		
1	1	3h		**A[3]**	**PFX[3]**			OFFS	**DP0**
1	1	4h	PSF	**A[4]**	**PFX[4]**			**DPC**	
1	1	5h	IC	**A[5]**	**PFX[5]**			**GR**	
1	1	6h	IMR	**A[6]**	**PFX[6]**		**LC0**	GRL	
1	1	7h		**A[7]**	**PFX[7]**		**LC1**	**BP**	**DP1**
1	2	8h	SC	**A[8]**				*GRS*	
1	2	9h		**A[9]**				GRH	
1	2	Ah		**A[10]**				*GRXL*	
1	2	Bh	*IIR*	**A[11]**				*FP*	
1	2	Ch		**A[12]**					
1	2	Dh		**A[13]**					
1	2	Eh	CKCN	**A[14]**					
1	2	Fh	WDCN	**A[15]**					

Note: Names that appear in italics indicate that all bits of a register are read-only. Names that appear in bold indicate that a register is 16 bits wide.

Table 1-9. MAXQ7665/MAXQ7666 System Register Bit Functions and Reset Value

REGISTER		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AP 08h[00h]										—	—	—	—	AP.3	AP.2	AP.1	AP.0
	reset									0	0	0	0	0	0	0	0
APC 08h[01h]										CLR	IDS	—	—	—	MOD2	MOD1	MOD0
	reset									0	0	0	0	0	0	0	0
PSF 08h[04h]										Z	S	—	GPF1	GPF0	OV	C	E
	reset									1	0	0	0	0	0	0	0
IC 08h[05h]										—	—	CGDS	—	—	—	INS	IGE
	reset									1	0	0	0	0	0	0	0
IMR 08h[06h]										IMS	—	IM5	IM4	IM3	IM2	IM1	IM0
	reset									0	0	0	0	0	0	0	0
SC 08h[08h]										TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
	reset									1	0	0	0	0	0	s	—
IIR 08h[0Bh]										IIS	—	II5	II4	II3	II2	II1	II0
	reset									0	0	0	0	0	0	0	0
CKCN 08h[0Eh]										XT	—	RGMD	STOP	SWB	PMME	CD1	CD0
	reset									s	0	s	0	0	0	0	1
WDCN 08h[0Fh]										POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
	reset									s	s	0	0	s	s	s	s
A[n] (0...15) 09h[0nh]		A[n].15	A[n].14	A[n].13	A[n].12	A[n].11	A[n].10	A[n].9	A[n].8	A[n].7	A[n].6	A[n].5	A[n].4	A[n].3	A[n].2	A[n].1	A[n].0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PFX[n] (0...7) 0Bh[0nh]		PFX[n].15	PFX[n].14	PFX[n].13	PFX[n].12	PFX[n].11	PFX[n].10	PFX[n].9	PFX[n].8	PFX[n].7	PFX[n].6	PFX[n].5	PFX[n].4	PFX[n].3	PFX[n].2	PFX[n].1	PFX[n].0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IP 0Ch[00h]		IP.15	IP.14	IP.13	IP.12	IP.11	IP.10	IP.9	IP.8	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
	reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SP 0Dh[01h]		—	—	—	—	—	—	—	—	—	—	—	—	SP.3	SP.2	SP.1	SP.0
	reset	—	—	—	—	—	—	—	—	—	—	—	—	1	1	1	1
IV 0Dh[02h]		IV.15	IV.14	IV.13	IV.12	IV.11	IV.10	IV.9	IV.8	IV.7	IV.6	IV.5	IV.4	IV.3	IV.2	IV.1	IV.0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LC[0] 0Dh[06h]		LC[0].15	LC[0].14	LC[0].13	LC[0].12	LC[0].11	LC[0].10	LC[0].9	LC[0].8	LC[0].7	LC[0].6	LC[0].5	LC[0].4	LC[0].3	LC[0].2	LC[0].1	LC[0].0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LC[1] 0Dh[07h]		LC[1].15	LC[1].14	LC[1].13	LC[1].12	LC[1].11	LC[1].10	LC[1].9	LC[1].8	LC[1].7	LC[1].6	LC[1].5	LC[1].4	LC[1].3	LC[1].2	LC[1].1	LC[1].0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OFFS 0Eh[03h]										OFFS.7	OFFS.6	OFFS.5	OFFS.4	OFFS.3	OFFS.2	OFFS.1	OFFS.0
	reset									0	0	0	0	0	0	0	0
DPC 0Eh[04h]										—	—	—	WBS2	WBS1	WBS0	SDPS1	SDPS0
	reset									0	0	0	1	1	1	0	0

Table 1-9. MAXQ7665/MAXQ7666 System Register Bit Functions and Reset Value (continued)

REGISTER	REGISTER BIT															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
GR 0Eh[05h]	GR.15	GR.14	GR.13	GR.12	GR.11	GR.10	GR.9	GR.8	GR.7	GR.6	GR.5	GR.4	GR.3	GR.2	GR.1	GR.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRL 0Eh[06h]									GRL.7	GRL.6	GRL.5	GRL.4	GRL.3	GRL.2	GRL.1	GRL.0
									0	0	0	0	0	0	0	0
BP 0Eh[07h]	BP.15	BP.14	BP.13	BP.12	BP.11	BP.10	BP.9	BP.8	BP.7	BP.6	BP.5	BP.4	BP.3	BP.2	BP.1	BP.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRS 0Eh[08h]	GRS.15	GRS.14	GRS.13	GRS.12	GRS.11	GRS.10	GRS.9	GRS.8	GRS.7	GRS.6	GRS.5	GRS.4	GRS.3	GRS.2	GRS.1	GRS.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRH 0Eh[09h]									GRH.7	GRH.6	GRH.5	GRH.4	GRH.3	GRH.2	GRH.1	GRH.0
									0	0	0	0	0	0	0	0
GRXL 0Eh[0Ah]	GRXL.15	GRXL.14	GRXL.13	GRXL.12	GRXL.11	GRXL.10	GRXL.9	GRXL.8	GRXL.7	GRXL.6	GRXL.5	GRXL.4	GRXL.3	GRXL.2	GRXL.1	GRXL.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FP 0Eh[0Bh]	FP.15	FP.14	FP.13	FP.12	FP.11	FP.10	FP.9	FP.8	FP.7	FP.6	FP.5	FP.4	FP.3	FP.2	FP.1	FP.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DP[0] 0Fh[03h]	DP[0].15	DP[0].14	DP[0].13	DP[0].12	DP[0].11	DP[0].10	DP[0].9	DP[0].8	DP[0].7	DP[0].6	DP[0].5	DP[0].4	DP[0].3	DP[0].2	DP[0].1	DP[0].0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DP[1] 0Fh[07h]	DP[1].15	DP[1].14	DP[1].13	DP[1].12	DP[1].11	DP[1].10	DP[1].9	DP[1].8	DP[1].7	DP[1].6	DP[1].5	DP[1].4	DP[1].3	DP[1].2	DP[1].1	DP[1].0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

s = Bit affected only by power-on reset and not by other forms of reset. See the register description for more information.

1.4.1 Accumulator Pointer Register (AP)

Register Description: **Accumulator Pointer Register**
Register Name: **AP**
Register Address: **Module 08h, Index 00h**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	AP.3	AP.2	AP.1	AP.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 7 to 4: Reserved. Read 0, write ignored.

Bits 3 to 0: Accumulator Select 3 to 0 (AP.3 to AP.0). These bits select which of the 16 accumulator registers are used for arithmetic and logical operations. If the APC register has been set to perform automatic increment/decrement of the active accumulator, this setting will be automatically changed after each arithmetic or logical operation. If a MOVE AP, Acc instruction is executed, any enabled AP inc/dec/modulo control will take precedence over the transfer of Acc data into AP.

1.4.2 Accumulator Pointer Control Register (APC)

Register Description: **Accumulator Pointer Control Register**
Register Name: **APC**
Register Address: **Module 08h, Index 01h**

Bit #	7	6	5	4	3	2	1	0
Name	CLR	IDS	—	—	—	MOD2	MOD1	MOD0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	r	r	r	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bit 7: Accumulator Pointer Clear (CLR). Writing this bit to 1 clears the accumulator pointer AP to 0. Once set, this bit will automatically be reset to 0 by hardware. If a MOVE APC, Acc instruction is executed requesting that AP be set to 0 (i.e., CLR = 1), the AP clear function overrides any enabled inc/dec/modulo control. All reads from this bit return 0.

Bit 6: Accumulator Pointer Increment/Decrement Select (IDS). If this bit is set to 0, the accumulator pointer AP is incremented following each arithmetic or logical operation according to MOD2:MOD0. If this bit is set to 1, the accumulator pointer AP is decremented following each arithmetic or logical operation according to MOD2:MOD0. If MOD2:MOD0 is set to 000, the setting of this bit is ignored.

Bits 5 to 3: Reserved. Read 0, write ignored.

Bits 2 to 0: Accumulator Pointer Auto-Increment/Decrement Modulus (MOD2 to MOD0). If these bits are set to a non-zero value, the accumulator pointer (AP3:AP0) will be automatically incremented or decremented following each arithmetic or logical operation. The mode for the auto-increment/decrement is determined as follows:

MOD2:MOD0	AUTO-INCREMENT/DECREMENT MODE
000	No auto-increment/decrement (default).
001	Increment/decrement AP[0] modulo 2.
010	Increment/decrement AP[1:0] modulo 4.
011	Increment/decrement AP[2:0] modulo 8.
100	Increment/decrement AP modulo 16.
101 to 111	Reserved (modulo 16 when set).

1.4.3 Processor Status Flags Register (PSF)

Register Description: **Processor Status Flags Register**
Register Name: **PSF**
Register Address: **Module 08h, Index 04h**

Bit #	7	6	5	4	3	2	1	0
Name	Z	S	—	GPF1	GPF0	OV	C	E
Reset	1	0	0	0	0	0	0	0
Access	r	r	r	rw	rw	r	rw	rw

r = read, w = write
Note: *This register is cleared to 80h on all forms of reset.*

Bit 7: Zero Flag (Z). The value of this bit flag equals 1 whenever the active accumulator is equal to zero, and it equals 0 otherwise.

Bit 6: Sign Flag (S). This bit flag mirrors the current value of the high bit of the active accumulator (Acc.15).

Bit 5: Reserved. Read 0, write ignored.

Bits 4 and 3: General-Purpose Software Flag 1 and 0 (GPF1 and GPF0). These general-purpose register bits are provided for user software control.

Bit 2: Overflow Flag (OV). This flag is set to 1 if there is a carry out of bit 14 but not out of bit 15, or a carry out of bit 15 but not out of bit 14 from the last arithmetic operation, otherwise, the OV flag remains as 0. OV indicates a negative number resulted as the sum of two positive operands, or a positive sum resulted from two negative operands.

Bit 1: Carry Flag (C). This bit flag is set to 1 whenever an add or subtract operation (ADD, ADDC, SUB, SUBB) returns a carry or borrow. This bit flag is cleared to 0 whenever an add or subtract operation does not return a carry or borrow. Many other instructions potentially affect the carry bit. See *Section 14: MAXQ7665/MAXQ7666 Instruction Set Summary* for details.

Bit 0: Equals Flag (E). This bit flag is set to 1 whenever a compare operation (CMP) returns an equal result. If a CMP operation returns not equal, this bit is cleared.

1.4.4 Interrupt and Control Register (IC)

Register Description: **Interrupt and Control Register**
Register Name: **IC**
Register Address: **Module 08h, Index 05h**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	CGDS	—	—	—	INS	IGE
Reset	0	0	0	0	0	0	0	0
Access	r	r	rw	r	r	r	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 7, 6, 4, 3, and 2: Reserved. Read 0, write ignored.

Bit 5: System Clock Gating Disable (CGDS). If this bit is set to 0 (default mode), system clock gating circuitry is active. If this bit is set to 1, the clock gating circuitry is disabled.

Bit 1: Interrupt In Service (INS). The INS is set by hardware automatically when an interrupt is acknowledged. No further interrupts occur as long as the INS remains set. The interrupt service routine can clear the INS bit to allow interrupt nesting. Otherwise, the INS bit is cleared by hardware upon execution of an RETI or POPI instruction.

Bit 0: Interrupt Global Enable (IGE). If this bit is set to 1, interrupts are globally enabled, but still must be locally enabled to occur. If this bit is set to 0, all interrupts are disabled.

1.4.5 Interrupt Mask Register (IMR)

The first six bits in this register are interrupt mask bits for modules 0 to 5, one bit per module. The eighth bit, IMS, serves as a mask for any system module interrupt sources. Setting a mask bit allows the enabled interrupt sources for the associated module or system (for the case of IMS) to generate interrupt requests. Clearing the mask bit effectively disables all interrupt sources associated with that specific module or all system interrupt sources (for the case of IMS). The interrupt mask register is intended to facilitate user-definable interrupt prioritization.

Register Description: **Interrupt Mask Register**
Register Name: **IMR**
Register Address: **Module 08h, Index 06h**

Bit #	7	6	5	4	3	2	1	0
Name	IMS	—	IM5	IM4	IM3	IM2	IM1	IM0
Reset	0	0	0	0	0	0	0	0
Access	rw	r	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bit 7: Interrupt Mask for System Modules (IMS)

Bit 6: Reserved. Read 0, write ignored.

Bits 5 to 0: Interrupt Mask for Register Module 5 to 0 (IM5 to IM0)

1.4.6 System Control Register (SC)

Register Description: **System Control Register**
Register Name: **SC**
Register Address: **Module 08h, Index 08h**

Bit #	7	6	5	4	3	2	1	0
Name	TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
Reset	1	0	0	0	0	0	1	0
Access	rw	r	rw	rw	rw	rw	rw	r

r = read, w = write
Note: *This register is reset to 100000s0b on all forms of reset. Bit 1 (PWL) is set to 1 on a power-on reset only.*

Bit 7: Test Access (JTAG) Port Enable (TAP). This bit controls whether the Test Access Port special-function pins are enabled. The TAP defaults to being enabled. Clearing this bit to 0 disables the TAP special function pins. See *Section 10* for more information about JTAG and TAP.

Bits 6 and 0: Reserved. Read 0, write ignored.

Bits 5 and 4: Code Data Access Bits 1 and 0 (CDA1 and CDA0). The CDA bits are used to logically map physical program memory page to the data space for read/write access (see table below).

The logical data memory addresses of the program pages depend on whether execution is from Utility ROM or logical data memory. Note that CDA1 is not implemented if the upper 32k of the program space is not used for the user code. No CDA bits are needed if only one page of program space is incorporated.

CDA1:CDA0	BYTE MODE ACTIVE PAGE	WORD MODE ACTIVE PAGE
00	P0	P0 and P1
01	P1	P0 and P1
10	P2	P2 and P3
11	P3	P2 and P3

Bit 3: Upper Program Access (UPA). The physical program memory is logically divided into four pages; P0 and P1 occupy the lower 32kWords while P2 and P3 occupy the upper 32kWords. P0 and P1 are assigned to the lower half of the program space and are always active. P2 and P3 must be explicitly activated in the upper half of the program space by setting the UPA bit to 1. When UPA bit is cleared to 0, the upper program memory space is occupied by the Utility ROM and the logical data memory, which is accessible as program memory. Note that the UPA is not implemented if the upper 32k of the program space is not used for the user code.

Bit 2: ROM Operation Done (ROD). This bit is used to signify completion of a ROM operation sequence to the control units. This allows the Debug engine to determine the status of a ROM sequence. Setting this bit to logic 1 causes an internal system reset if the JTAG SPE bit is also set. Setting the ROD bit will clear the JTAG SPE bit if it is set and the ROD bit will be automatically cleared by hardware once the control unit acknowledges the done indication. See *Section 11* for more information.

Bit 1: Password Lock (PWL). This bit defaults to 1 on a power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the program space before allowing access to the password protected in-circuit debug or bootstrap loader ROM routines. Clearing this bit to 0 disables the password protection for these ROM routines. See *Section 12* for more information.

1.4.7 Interrupt Identification Register (IIR)

The first six bits in this register indicate interrupts pending in modules 0 to 5, one bit per module. The eighth bit, IIS, indicates a pending system interrupt, such as from the watchdog timer. The interrupt pending flags will be set only for enabled interrupt sources waiting for service. The interrupt pending flag will be cleared when the pending interrupt sources within that module are disabled or when the interrupt flags are cleared by software.

Register Description: **Interrupt Identification Register**
Register Name: **IIR**
Register Address: **Module 08h, Index 0Bh**

Bit #	7	6	5	4	3	2	1	0
Name	IIS	—	II5	II4	II3	II2	II1	II0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 00h on all forms of reset.*

Bit 7: Interrupt Identifier Flag for System Modules (IIS)

Bit 6: Reserved. Read 0, write ignored.

Bits 5 to 0: Interrupt Identifier Flag for Register Module 5 to 0 (II5 to II0)

1.4.8 System Clock Control Register (CKCN)

The 8-bit CKCN register is part of the system register group and used to support system clock generation. It controls the system clock speed and power management mode selection. See *Section 5* for the description of this register.

Register Description: **System Clock Control Register**
Register Name: **CKCN**
Register Address: **Module 08h, Index 0Eh**

Bit #	7	6	5	4	3	2	1	0
Name	XT	—	RGMD	STOP	SWB	PMME	CD1	CD0
Reset	0	0	1	0	0	0	0	1
Access	rw	r	r	rw	rw	rw	rw	rw

r = read, w = write
Note: *Bits 4:0 are set to 00001b on all forms of reset. See bit description for bits 7 and 5.*

MAXQ7665/MAXQ7666 User's Guide

1.4.9 Watchdog Timer Control Register (WDCN)

The 8-bit WDCN register is part of the system register group and used to provide system control. It controls the watchdog timeout period and interrupt or reset generation on watchdog timeout. The watchdog timer is clocked by the internal 7.6MHz RC oscillator. See *Section 5* for a description of this register.

Register Description: **Watchdog Timer Control Register**
Register Name: **WDCN**
Register Address: **Module 08h, Index 0Fh**

Bit #	7	6	5	4	3	2	1	0
Name	POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *Bits 5, 4, 3, and 0 are cleared to 0 on all forms of reset; for others, see the individual bit descriptions.*

1.4.10 Accumulator n Register (A[n])

Register Description: **Accumulator n Register**
Register Name: **A[n]**
Register Address: **Module 09h, Index 0nh**

The MAXQ7665/MAXQ7666 support 16 accumulator registers (A[0] to A[15]).

Bit #	15	14	13	12	11	10	9	8
Name	A[n].15	A[n].14	A[n].13	A[n].12	A[n].11	A[n].10	A[n].9	A[n].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	A[n].7	A[n].6	A[n].5	A[n].4	A[n].3	A[n].2	A[n].1	A[n].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Accumulator n Register Bits 15 to 0 (A[n].15 to A[n].0). This register acts as the accumulator for all ALU arithmetic and logical operations when selected by the accumulator pointer (AP). It can also be used as a general-purpose working register.

1.4.11 Prefix Register (PFX[n])

Register Description: **Prefix Register**
Register Name: **PFX[n]**
Register Address: **Module 0Bh, Index 0nh**

Bit #	15	14	13	12	11	10	9	8
Name	PFX[n].15	PFX[n].14	PFX[n].13	PFX[n].12	PFX[n].11	PFX[n].10	PFX[n].9	PFX[n].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	PFX[n].7	PFX[n].6	PFX[n].5	PFX[n].4	PFX[n].3	PFX[n].2	PFX[n].1	PFX[n].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Prefix Register Bits 15 to 0 (PFX[n].15 to PFX[n].0). The prefix register provides a means of supplying an additional 8 bits of high-order data for use by the succeeding instruction as well as providing additional indexing capabilities. This register will only hold any data written to it for one execution cycle, after which it will revert to 0000h. Although this is a 16-bit register, only the lower 8 bits are actually used for prefixing purposes by the next instruction. Writing to or reading from any index in the Prefix module will select the same 16-bit register. However, when the prefix register is written, the index n used for the PFX[n] write also determines the high-order bits for the register source and destination specified in the following instruction.

The index selection reverts to 0 (default mode allowing selection of registers 0h to 7h for destinations) after one cycle in the same manner as the contents of the prefix register.

WRITE TO	SOURCE, DESTINATION INDEX SELECTION	
	SOURCE REGISTER RANGE	DESTINATION REGISTER RANGE
PFX[0]	0h to Fh	0h to 7h
PFX[1]	10h to 1Fh	0h to 7h
PFX[2]	0h to Fh	8h to Fh
PFX[3]	10h to 1Fh	8h to Fh
PFX[4]	0h to Fh	10h to 17h
PFX[5]	10h to 1Fh	10h to 17h
PFX[6]	0h to Fh	18h to 1Fh
PFX[7]	10h to 1Fh	18h to 1Fh

MAXQ7665/MAXQ7666 User's Guide

1.4.12 Instruction Pointer Register (IP)

Register Description: **Instruction Pointer Register**
Register Name: **IP**
Register Address: **Module 0Ch, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	IP.15	IP.14	IP.13	IP.12	IP.11	IP.10	IP.9	IP.8
Reset	1	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 8000h on all forms of reset.*

Bits 15 to 0: Instruction Pointer Register Bits 15 to 0 (IP.15 to IP.0). This register contains the address of the next instruction to be executed and is automatically incremented by 1 after each program fetch. Writing an address value to this register will cause program flow to jump to that address. Reading from this register will not affect program flow.

1.4.13 Stack Pointer Register (SP)

Register Description: **Stack Pointer Register**
Register Name: **SP**
Register Address: **Module 0Dh, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	SP.3	SP.2	SP.1	SP.0
Reset	0	0	0	0	1	1	1	1
Access	r	r	r	r	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 000Fh on all forms of reset.*

Bits 15 to 4: Reserved. Read 0, write ignored.

Bits 3 to 0: Stack Pointer Register Bits 3 to 0 (SP.3 to SP.0). These four bits indicate the current top of the hardware stack, from 0h to Fh. This pointer is incremented after a value is pushed on the stack and decremented before a value is popped from the stack.

1.4.14 Interrupt Vector Register (IV)

Register Description: **Interrupt Vector Register**
Register Name: **IV**
Register Address: **Module 0Dh, Index 02h**

Bit #	15	14	13	12	11	10	9	8
Name	IV.15	IV.14	IV.13	IV.12	IV.11	IV.10	IV.9	IV.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	IV.7	IV.6	IV.5	IV.4	IV.3	IV.2	IV.1	IV.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Interrupt Vector Register Bits 15 to 0 (IV.15 to IV.0). This register contains the address of the interrupt service routine. The interrupt handler will generate a CALL to this address whenever an interrupt is acknowledged.

1.4.15 Loop Counter 0 Register (LC[0])

Register Description: **Loop Counter 0 Register**
Register Name: **LC[0]**
Register Address: **Module 0Dh, Index 06h**

Bit #	15	14	13	12	11	10	9	8
Name	LC[0].15	LC[0].14	LC[0].13	LC[0].12	LC[0].11	LC[0].10	LC[0].9	LC[0].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	LC[0].7	LC[0].6	LC[0].5	LC[0].4	LC[0].3	LC[0].2	LC[0].1	LC[0].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Loop Counter 0 Register Bits 15 to 0 (LC[0].15 to LC[0].0). This register is used as the loop counter for the DJNZ LC[0], src operation. This operation decrements LC[0] by one and then jumps to the address specified in the instruction by src.

1.4.16 Loop Counter 1 Register (LC[1])

Register Description: **Loop Counter 1 Register**
Register Name: **LC[1]**
Register Address: **Module 0Dh, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	LC[1].15	LC[1].14	LC[1].13	LC[1].12	LC[1].11	LC[1].10	LC[1].9	LC[1].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	LC[1].7	LC[1].6	LC[1].5	LC[1].4	LC[1].3	LC[1].2	LC[1].1	LC[1].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Loop Counter 1 Register Bits 15 to 0 (LC[1].15 to LC[1].0). This register is used as the loop counter for the DJNZ LC[1], src operation. This operation decrements LC[1] by one and then jumps to the address specified in the instruction by src.

1.4.17 Frame Pointer Offset Register (OFFS)

Register Description: **Frame Pointer Offset Register**
Register Name: **OFFS**
Register Address: **Module 0Eh, Index 03h**

Bit #	7	6	5	4	3	2	1	0
Name	OFFS.7	OFFS.6	OFFS.5	OFFS.4	OFFS.3	OFFS.2	OFFS.1	OFFS.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 7 to 0: Frame Pointer Offset Register Bits 7 to 0 (OFFS.7 to OFFS.0). This 8-bit register provides the frame pointer (FP) offset from the base pointer (BP). The frame pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (OFFS). The contents of this register can be post-incremented or post-decremented when using the frame pointer for read operations and may be preincremented or pre-decremented when using the frame pointer for write operations. A carry out or borrow resulting from an increment/decrement operation has no effect on the Frame Pointer Base Register (BP).

1.4.18 Data Pointer Control Register (DPC)

Register Description: **Data Pointer Control Register**
Register Name: **DPC**
Register Address: **Module 0Eh, Index 04h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	WBS2	WBS1	WBS0	SDPS1	SDPS0
Reset	0	0	0	1	1	1	0	0
Access	r	r	r	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 001Ch on all forms of reset.*

Bits 15 to 5: Reserved. Read 0, write ignored.

Bit 4: Word/Byte Select 2 (WBS2). This bit selects access mode for BP[OFFS]. When WBS2 is set to logic 1, the BP[OFFS] is operated in word mode for data memory access; when WBS2 is cleared to logic 0, BP[OFFS] is operated in byte mode for data memory access.

Bit 3: Word/Byte Select 1 (WBS1). This bit selects access mode for DP[1]. When WBS1 is set to logic 1, the DP[1] is operated in word mode for data memory access; when WBS1 is cleared to logic 0, DP[1] is operated in byte mode for data memory access.

Bit 2: Word/Byte Select 0 (WBS0). This bit selects access mode for DP[0]. When WBS0 is set to logic 1, the DP[0] is operated in word mode for data memory access; when WBS0 is cleared to logic 0, DP[0] is operated in byte mode for data memory access.

Bits 1 and 0: Source Data Pointer Select Bits 1 and 0 (SDPS1 and SDPS0). These bits select one of the three data pointers as the active source pointer for the load operation. A new data pointer must be selected before being used to read data memory (see table below).

These bits default to 00b but do not activate DP[0] as an active source pointer until the SDPS bits are explicitly cleared to 00b or the DP[0] register is written by an instruction. Also, modifying the register contents of a data/frame pointer register (DP[0], DP[1], BP, or OFFS) will change the setting of the SDPS bits to reflect the active source pointer selection.

SDPS1	SDPS0	SOURCE POINTER SELECTION
0	0	DP[0]
0	1	DP[1]
1	0	FP (BP[OFFS])
1	1	Reserved (select FP if set)

1.4.19 General Register (GR)

Register Description: **General Register**
Register Name: **GR**
Register Address: **Module 0Eh, Index 05h**

Bit #	15	14	13	12	11	10	9	8
Name	GR.15	GR.14	GR.13	GR.12	GR.11	GR.10	GR.9	GR.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	GR.7	GR.6	GR.5	GR.4	GR.3	GR.2	GR.1	GR.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: General Register Bits 15 to 0 (GR.15 to GR.0). This register is intended primarily for supporting byte operations on 16-bit data. The 16-bit register is byte-readable, byte-writeable through the corresponding GRL and GRH 8-bit registers and byte-swappable through the GRS 16-bit register.

1.4.20 General Register Low Byte (GRL)

Register Description: **General Register Low Byte**
Register Name: **GRL**
Register Address: **Module 0Eh, Index 06h**

Bit #	7	6	5	4	3	2	1	0
Name	GRL.7	GRL.6	GRL.5	GRL.4	GRL.3	GRL.2	GRL.1	GRL.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 7 to 0: General Register Low Byte Bits 7 to 0 (GRL.7 to GRL.0). This register reflects the low byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRL register will also be stored in the low byte of the GR register.

1.4.21 Frame Pointer Base Register (BP)

Register Description: **Frame Pointer Base Register**
Register Name: **BP**
Register Address: **Module 0Eh, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	BP.15	BP.14	BP.13	BP.12	BP.11	BP.10	BP.9	BP.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	BP.7	BP.6	BP.5	BP.4	BP.3	BP.2	BP.1	BP.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Frame Pointer Base Register Bits 15 to 0 (BP.15 to BP.0). This register serves as the base pointer for the Frame Pointer (FP). The Frame Pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (OFFS). The content of this base pointer register is not affected by increment/decrement operations performed on the offset (OFFS) register.

1.4.22 General Register Byte-Swapped (GRS)

Register Description: **General Register Byte-Swapped**
Register Name: **GRS**
Register Address: **Module 0Eh, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	GRS.15	GRS.14	GRS.13	GRS.12	GRS.11	GRS.10	GRS.9	GRS.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	GRS.7	GRS.6	GRS.5	GRS.4	GRS.3	GRS.2	GRS.1	GRS.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: General Register Byte-Swapped Bits 15 to 0 (GRS.15 to GRS.0). This register is intended primarily for supporting byte operations on 16-bit data. This 16-bit read-only register returns the byte-swapped value for the data contained in the GR register.

1.4.23 General Register High Byte (GRH)

Register Description: **General Register High Byte**
Register Name: **GRH**
Register Address: **Module 0Eh, Index 09h**

Bit #	7	6	5	4	3	2	1	0
Name	GRH.7	GRH.6	GRH.5	GRH.4	GRH.3	GRH.2	GRH.1	GRH.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 7 to 0: General Register High Byte Bits 7 to 0 (GRH.7 to GRH.0). This register reflects the high byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRH register will also be stored in the high byte of the GR register.

1.4.24 General Register Sign Extended Low Byte (GRXL)

Register Description: **General Register Sign Extended Low Byte**
Register Name: **GRXL**
Register Address: **Module 0Eh, Index 0Ah**

Bit #	15	14	13	12	11	10	9	8
Name	GRXL.15	GRXL.14	GRXL.13	GRXL.12	GRXL.11	GRXL.10	GRXL.9	GRXL.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	GRXL.7	GRXL.6	GRXL.5	GRXL.4	GRXL.3	GRXL.2	GRXL.1	GRXL.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: General Register Sign Extended Low Byte Bits 15 to 0 (GRXL.15 to GRXL.0). This register provides the sign extended low byte of GR as a 16-bit source.

1.4.25 Frame Pointer Register (FP)

Register Description: **Frame Pointer Register**
Register Name: **FP**
Register Address: **Module 0Eh, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	FP.15	FP.14	FP.13	FP.12	FP.11	FP.10	FP.9	FP.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	FP.7	FP.6	FP.5	FP.4	FP.3	FP.2	FP.1	FP.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Frame Pointer Register Bits 15 to 0 (FP.15 to FP.0). This register provides the current value of the frame pointer (BP[OFFS]).

1.4.26 Data Pointer 0 Register (DP[0])

Register Description: **Data Pointer 0 Register**
Register Name: **DP[0]**
Register Address: **Module 0Fh, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	DP[0].15	DP[0].14	DP[0].13	DP[0].12	DP[0].11	DP[0].10	DP[0].9	DP[0].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	DP[0].7	DP[0].6	DP[0].5	DP[0].4	DP[0].3	DP[0].2	DP[0].1	DP[0].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Data Pointer 0 Register Bits 15 to 0 (DP[0].15 to DP[0].0). This register is used as a pointer to access data memory. DP[0] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

1.4.27 Data Pointer 1 Register (DP[1])

Register Description: **Data Pointer 1 Register**
Register Name: **DP[1]**
Register Address: **Module 0Fh, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	DP[1].15	DP[1].14	DP[1].13	DP[1].12	DP[1].11	DP[1].10	DP[1].9	DP[1].8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	DP[1].7	DP[1].6	DP[1].5	DP[1].4	DP[1].3	DP[1].2	DP[1].1	DP[1].0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Data Pointer 1 Register Bits 15 to 0 (DP[1].15 to DP[1].0). This register is used as a pointer to access data memory. DP[1] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

1.5 Peripheral Register Modules

The MAXQ7665/MAXQ7666 microcontrollers use peripheral registers to control and monitor peripheral modules. These registers reside in Modules 0h to 5h, with subindex values 0h to 1Fh. The MAXQ7665/MAXQ7666 peripheral register map is shown in Table 1-10. The peripheral register module bit function and reset values are shown in Table 1-11. Each peripheral module and its associated registers/bits are covered separately in the following sections.

Table 1-10. MAXQ7665/MAXQ7666 Peripheral Register Map

REGISTER INDEX	MODULE NAME (BASE SPECIFIER)					
	M0	M1	M2	M3	M4	M5
00h	PO0	MCNT	T2CNA0	T2CNA2	C0C	VMC
01h		MA	T2H0	T2H2	C0S	APE
02h		MB	T2RH0	T2RH2	*C0IR*	ACNT
03h	EIF0	MC2	T2CH0	T2CH2	C0TE	DCNT
04h		MC1	T2CNA1		C0RE	DACI
05h		MC0	T2H1		COR	
06h		SPIB	T2RH1		C0DP	DACO
07h	SBUF0	SPICN	T2CH1		C0DB	
08h	*PI0*	SPICF	T2CNB0	T2CNB2	*C0RMS*	*ADCD*
09h		SPICK	T2V0	T2V2	*C0TMA*	*TSO*
0Ah		FCNTL	T2R0	T2R2		AIE
0Bh	EIE0	FDATA	T2C0	T2C2		*ASR*
0Ch		*MC1R*	T2CNB1			OSCC
0Dh		*MC0R*	T2V1			
0Eh			T2R1			
0Fh			T2C1			
10h	PD0		T2CFG0	T2CFG2		
11h			T2CFG1		C0M1C	
12h					C0M2C	
13h	EIES0				C0M3C	
14h					C0M4C	
15h					C0M5C	
16h					C0M6C	
17h					C0M7C	
18h			ICDT0		C0M8C	
19h			ICDT1		C0M9C	
1Ah			ICDC		C0M10C	
1Bh			ICDF		C0M11C	
1Ch		FADDR	ICDB		C0M12C	
1Dh	SCON0		ICDA		C0M13C	
1Eh	SMD0		ICDD		C0M14C	
1Fh	PR0				C0M15C	

Note: *Names that appear in bold italics indicate that all bits of a register are read-only.*

MAXQ7665/MAXQ7666 User's Guide

Table 1-11. MAXQ7665/MAXQ7666 Module 0 Register Bit Functions and Reset Values

REGISTER		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								REGISTER BIT									
PO0 00h[00h]	function	—	—	—	—	—	—	—	—	PO0.7	PO0.6	PO0.5	PO0.4	PO0.3	PO0.2	PO0.1	PO0.0
	reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
EIF0 00h[03h]	function	—	—	—	—	—	—	—	—	IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SBUF0 00h[07h]	function	—	—	—	—	—	—	—	—	SBUF0.7	SBUF0.6	SBUF0.5	SBUF0.4	SBUF0.3	SBUF0.2	SBUF0.1	SBUF0.0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PI0 00h[08h]	function	—	—	—	—	—	—	—	—	PI0.7	PI0.6	PI0.5	PI0.4	PI0.3	PI0.2	PI0.1	PI0.0
	reset	0	0	0	0	0	0	0	0	s	s	s	s	s	s	s	s
EIE0 00h[0Bh]	function	—	—	—	—	—	—	—	—	EX7	EX6	EX5	EX4	EX3	EX2	EX1	EX0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PD0 00h[10h]	function	—	—	—	—	—	—	—	—	PD0.7	PD0.6	PD0.5	PD0.4	PD0.3	PD0.2	PD0.1	PD0.0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIES0 00h[13h]	function	—	—	—	—	—	—	—	—	IT7	IT6	IT5	IT4	IT3	IT2	IT1	IT0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SCON0 00h[1Dh]	function	—	—	—	—	—	—	—	—	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SMD0 00h[1Eh]	function	—	—	—	—	—	—	—	—	—	—	—	—	—	ESI	SMOD	FEDE
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PR0 00h[1Fh]	function	PR0.15	PR0.14	PR0.13	PR0.12	PR0.11	PR0.10	PR0.9	PR0.8	PR0.7	PR0.6	PR0.5	PR0.4	PR0.3	PR0.2	PR0.1	PR0.0
	reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

s = Dependent on the pin's state.

Table 1-12. MAXQ7665/MAXQ7666 Module 1 Register Bit Functions and Reset Values

REGISTER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MCNT 01h[00h]	—	—	—	—	—	—	—	—	OF	MCW	CLD	SQU	OPCS	MSUB	MMAC	SUS
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MA 01h[01h]	MA.15	MA.14	MA.13	MA.12	MA.11	MA.10	MA.9	MA.8	MA.7	MA.6	MA.5	MA.4	MA.3	MA.2	MA.1	MA.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MB 01h[02h]	MB.15	MB.14	MB.13	MB.12	MB.11	MB.10	MB.9	MB.8	MB.7	MB.6	MB.5	MB.4	MB.3	MB.2	MB.1	MB.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MC2 01h[03h]	MC2.15	MC2.14	MC2.13	MC2.12	MC2.11	MC2.10	MC2.9	MC2.8	MC2.7	MC2.6	MC2.5	MC2.4	MC2.3	MC2.2	MC2.1	MC2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MC1 01h[04h]	MC1.15	MC1.14	MC1.13	MC1.12	MC1.11	MC1.10	MC1.9	MC1.8	MC1.7	MC1.6	MC1.5	MC1.4	MC1.3	MC1.2	MC1.1	MC1.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MC0 01h[05h]	MC0.15	MC0.14	MC0.13	MC0.12	MC0.11	MC0.10	MC0.9	MC0.8	MC0.7	MC0.6	MC0.5	MC0.4	MC0.3	MC0.2	MC0.1	MC0.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPIB 01h[06h]	SPIB.15	SPIB.14	SPIB.13	SPIB.12	SPIB.11	SPIB.10	SPIB.9	SPIB.8	SPIB.7	SPIB.6	SPIB.5	SPIB.4	SPIB.3	SPIB.2	SPIB.1	SPIB.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPICN 01h[07h]	—	—	—	—	—	—	—	—	STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPICF 01h[08h]	—	—	—	—	—	—	—	—	ESPII	—	—	—	—	CHR	CKPHA	CKPOL
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPICK 01h[09h]	—	—	—	—	—	—	—	—	CKR.7	CKR.6	CKR.5	CKR.4	CKR.3	CKR.2	CKR.1	CKR.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FCNTL* (Type A Flash) 01h[0Ah]	—	—	—	—	—	—	—	—	FBUSY	FERR	FINE	FBYP	DQ5	FC2	FC1	—
	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
FCNTL* (Type F Flash) 01h[0Ah]	—	—	—	—	—	—	—	—	FRDY	FERR	—	—	FCRA3	FCRA2	FCRA1	FCRA0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FDATA* 01h[0Bh]	FDATA.15	FDATA.14	FDATA.13	FDATA.12	FDATA.11	FDATA.10	FDATA.9	FDATA.8	FDATA.7	FDATA.6	FDATA.5	FDATA.4	FDATA.3	FDATA.2	FDATA.1	FDATA.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MC1R 01h[0Ch]	MC1R.15	MC1R.14	MC1R.13	MC1R.12	MC1R.11	MC1R.10	MC1R.9	MC1R.8	MC1R.7	MC1R.6	MC1R.5	MC1R.4	MC1R.3	MC1R.2	MC1R.1	MC1R.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MC0R 01h[0Dh]	MC0R.15	MC0R.14	MC0R.13	MC0R.12	MC0R.11	MC0R.10	MC0R.9	MC0R.8	MC0R.7	MC0R.6	MC0R.5	MC0R.4	MC0R.3	MC0R.2	MC0R.1	MC0R.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FADDR* (Type F Flash Only) 01h[1Ch]	FADDR.15	FADDR.14	FADDR.13	FADDR.12	FADDR.11	FADDR.10	FADDR.9	FADDR.8	FADDR.7	FADDR.6	FADDR.5	FADDR.4	FADDR.3	FADDR.2	FADDR.1	FADDR.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

*FCNTL, FDATA, and FADDR are not accessible by program code inside the flash memory (blocked by hardware) and are accessible only to the utility ROM and data RAM.

Table 1-13. MAXQ7665/MAXQ7666 Module 2 Register Bit Functions and Reset Values

REGISTER		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							REGISTER BIT										
T2CNA0 02h[00h]		—	—	—	—	—	—	—	—	ET2	T2OE0	T2POL0	TR2L	TR2	CPRL2	SS2	G2EN
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2H0 02h[01h]		—	—	—	—	—	—	—	—	T2H0.7	T2H0.6	T2H0.5	T2H0.4	T2H0.3	T2H0.2	T2H0.1	T2H0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2RH0 02h[02h]		—	—	—	—	—	—	—	—	T2RH0.7	T2RH0.6	T2RH0.5	T2RH0.4	T2RH0.3	T2RH0.2	T2RH0.1	T2RH0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CH0 02h[03h]		—	—	—	—	—	—	—	—	T2CH0.7	T2CH0.6	T2CH0.5	T2CH0.4	T2CH0.3	T2CH0.2	T2CH0.1	T2CH0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CNA1 02h[04h]		—	—	—	—	—	—	—	—	ET2	T2OE0	T2POL0	TR2L	TR2	CPRL2	SS2	G2EN
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2H1 02h[05h]		—	—	—	—	—	—	—	—	T2H1.7	T2H1.6	T2H1.5	T2H1.4	T2H1.3	T2H1.2	T2H1.1	T2H1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2RH1 02h[06h]		—	—	—	—	—	—	—	—	T2RH1.7	T2RH1.6	T2RH1.5	T2RH1.4	T2RH1.3	T2RH1.2	T2RH1.1	T2RH1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CH1 02h[07h]		—	—	—	—	—	—	—	—	T2CH1.7	T2CH1.6	T2CH1.5	T2CH1.4	T2CH1.3	T2CH1.2	T2CH1.1	T2CH1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CNB0 02h[08h]		—	—	—	—	—	—	—	—	ET2L	—	—	—	TF2	TF2L	TCC2	TC2L
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2V0 02h[09h]		T2V0.15	T2V0.14	T2V0.13	T2V0.12	T2V0.11	T2V0.10	T2V0.9	T2V0.8	T2V0.7	T2V0.6	T2V0.5	T2V0.4	T2V0.3	T2V0.2	T2V0.1	T2V0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2R0 02h[0Ah]		T2R0.15	T2R0.14	T2R0.13	T2R0.12	T2R0.11	T2R0.10	T2R0.9	T2R0.8	T2R0.7	T2R0.6	T2R0.5	T2R0.4	T2R0.3	T2R0.2	T2R0.1	T2R0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2C0 02h[0Bh]		T2C0.15	T2C0.14	T2C0.13	T2C0.12	T2C0.11	T2C0.10	T2C0.9	T2C0.8	T2C0.7	T2C0.6	T2C0.5	T2C0.4	T2C0.3	T2C0.2	T2C0.1	T2C0.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CNB1 02h[0Ch]		—	—	—	—	—	—	—	—	ET2L	—	—	—	TF2	TF2L	TCC2	TC2L
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2V1 02h[0Dh]		T2V1.15	T2V1.14	T2V1.13	T2V1.12	T2V1.11	T2V1.10	T2V1.9	T2V1.8	T2V1.7	T2V1.6	T2V1.5	T2V1.4	T2V1.3	T2V1.2	T2V1.1	T2V1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2R1 02h[0Eh]		T2R1.15	T2R1.14	T2R1.13	T2R1.12	T2R1.11	T2R1.10	T2R1.9	T2R1.8	T2R1.7	T2R1.6	T2R1.5	T2R1.4	T2R1.3	T2R1.2	T2R1.1	T2R1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2C1 02h[0Fh]		T2C1.15	T2C1.14	T2C1.13	T2C1.12	T2C1.11	T2C1.10	T2C1.9	T2C1.8	T2C1.7	T2C1.6	T2C1.5	T2C1.4	T2C1.3	T2C1.2	T2C1.1	T2C1.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CFG0 02h[10h]		—	—	—	—	—	—	T2DIV2	—	—	T2DIV2	T2DIV1	T2DIV0	T2MD	CCF1	CCF0	C/T2
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1-13. MAXQ7665/MAXQ7666 Module 2 Register Bit Functions and Reset Values (continued)

REGISTER		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													REGISTER BIT				
T2CFG1 02h[11h]		—	—	—	—	—	—	—	—	—	T2DIV2	T2DIV1	T2DIV0	T2MD	CCF1	CCF0	C/T2
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ICDT0 02h[18h]		ICDT0.15	ICDT0.14	ICDT0.13	ICDT0.12	ICDT0.11	ICDT0.10	ICDT0.9	ICDT0.8	ICDT0.7	ICDT0.6	ICDT0.5	ICDT0.4	ICDT0.3	ICDT0.2	ICDT0.1	ICDT0.0
		db	db	db	db	db	db	db	db	db	db	db	db	db	db	db	db
ICDT1 02h[19h]		ICDT1.15	ICDT1.14	ICDT1.13	ICDT1.12	ICDT1.11	ICDT1.10	ICDT1.9	ICDT1.8	ICDT1.7	ICDT1.6	ICDT1.5	ICDT1.4	ICDT1.3	ICDT1.2	ICDT1.1	ICDT1.0
		db	db	db	db	db	db	db	db	db	db	db	db	db	db	db	db
ICDC 02h[1Ah]		—	—	—	—	—	—	—	—	DME	—	REGE	—	CMD.3	CMD.2	CMD.1	CMD.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ICDF 02h[1Bh]		—	—	—	—	—	—	—	—	—	—	—	—	PSS1	PSS0	SPE	TXC
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ICDB 02h[1Ch]		—	—	—	—	—	—	—	—	ICDB.7	ICDB.6	ICDB.5	ICDB.4	ICDB.3	ICDB.2	ICDB.1	ICDB.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ICDA 02h[1Dh]		ICDA.15	ICDA.14	ICDA.13	ICDA.12	ICDA.11	ICDA.10	ICDA.9	ICDA.8	ICDA.7	ICDA.6	ICDA.5	ICDA.4	ICDA.3	ICDA.2	ICDA.1	ICDA.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ICDD 02h[1Eh]		ICDD.15	ICDD.14	ICDD.13	ICDD.12	ICDD.11	ICDD.10	ICDD.9	ICDD.8	ICDD.7	ICDD.6	ICDD.5	ICDD.4	ICDD.3	ICDD.2	ICDD.1	ICDD.0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

db = Special: read/write access only in background or debug mode.
dw = Special: write-only by debug engine.

Table 1-14. MAXQ7665/MAXQ7666 Module 3 Register Bit Functions and Reset Values

REGISTER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												REGISTER BIT				
T2CNA2 03h[00h]	—	—	—	—	—	—	—	—	ET2	T2OE0	T2POL0	TR2L	TR2	CPRL2	SS2	G2EN
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2H2 03h[01h]	—	—	—	—	—	—	—	—	T2H2.7	T2H2.6	T2H2.5	T2H2.4	T2H2.3	T2H2.2	T2H2.1	T2H2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2RH2 03h[02h]	—	—	—	—	—	—	—	—	T2RH2.7	T2RH2.6	T2RH2.5	T2RH2.4	T2RH2.3	T2RH2.2	T2RH2.1	T2RH2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CH2 03h[03h]	—	—	—	—	—	—	—	—	T2CH2.7	T2CH2.6	T2CH2.5	T2CH2.4	T2CH2.3	T2CH2.2	T2CH2.1	T2CH2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CNB2 03h[08h]	—	—	—	—	—	—	—	—	ET2L	—	—	—	TF2	TF2L	TCC2	TC2L
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2V2 03h[09h]	T2V2.15	T2V2.14	T2V2.13	T2V2.12	T2V2.11	T2V2.10	T2V2.9	T2V2.8	T2V2.7	T2V2.6	T2V2.5	T2V2.4	T2V2.3	T2V2.2	T2V2.1	T2V2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2R2 03h[0Ah]	T2R2.15	T2R2.14	T2R2.13	T2R2.12	T2R2.11	T2R2.10	T2R2.9	T2R2.8	T2R2.7	T2R2.6	T2R2.5	T2R2.4	T2R2.3	T2R2.2	T2R2.1	T2R2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2C2 03h[0Bh]	T2C2.15	T2C2.14	T2C2.13	T2C2.12	T2C2.11	T2C2.10	T2C2.9	T2C2.8	T2C2.7	T2C2.6	T2C2.5	T2C2.4	T2C2.3	T2C2.2	T2C2.1	T2C2.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2CFG2 03h[10h]	—	—	—	—	—	—	—	—	—	T2DIV2	T2DIV1	T2DIV0	T2MD	CCF1	CCF0	C/T2
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1-15. MAXQ7665/MAXQ7666 Module 4 Register Bit Functions and Reset Values

REGISTER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C0C	—	—	—	—	—	—	—	—	ERIE	STIE	PDE	SIESTA	CRST	AUTOB	ERCS	SWINT
04h[00h]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
C0S	—	—	—	—	—	—	—	—	BSS	EC96/128	WKS	RXS	TXS	ER2	ER1	ER0
04h[01h]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
C0IR	—	—	—	—	—	—	—	—	INTIN7	INTIN6	INTIN5	INTIN4	INTIN3	INTIN2	INTIN1	INTIN0
04h[02h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0TE	—	—	—	—	—	—	—	—	C0TE.7	C0TE.6	C0TE.5	C0TE.4	C0TE.3	C0TE.2	C0TE.1	C0TE.0
04h[03h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0RE	—	—	—	—	—	—	—	—	C0RE.7	C0RE.6	C0RE.5	C0RE.4	C0RE.3	C0RE.2	C0RE.1	C0RE.0
04h[04h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0R	—	—	—	—	—	—	—	—	CAN0BA	INCDEC	AID	C0BPR7	C0BPR6	—	C0BIE	C0IE
04h[05h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0DP	C0DP.15	C0DP.14	C0DP.13	C0DP.12	C0DP.11	C0DP.10	C0DP.9	C0DP.8	C0DP.7	C0DP.6	C0DP.5	C0DP.4	C0DP.3	C0DP.2	C0DP.1	C0DP.0
04h[06h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0DB	C0DB.15	C0DB.14	C0DB.13	C0DB.12	C0DB.11	C0DB.10	C0DB.9	C0DB.8	C0DB.7	C0DB.6	C0DB.5	C0DB.4	C0DB.3	C0DB.2	C0DB.1	C0DB.0
04h[07h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0RMS	—	C0RMS.15	C0RMS.14	C0RMS.13	C0RMS.12	C0RMS.11	C0RMS.10	C0RMS.9	C0RMS.8	C0RMS.7	C0RMS.6	C0RMS.5	C0RMS.4	C0RMS.3	C0RMS.2	C0RMS.1
04h[08h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0TMA	—	C0TMA.15	C0TMA.14	C0TMA.13	C0TMA.12	C0TMA.11	C0TMA.10	C0TMA.9	C0TMA.8	C0TMA.7	C0TMA.6	C0TMA.5	C0TMA.4	C0TMA.3	C0TMA.2	C0TMA.1
04h[09h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M1C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[11h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M2C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[12h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M3C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[13h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M4C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[14h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M5C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[15h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M6C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[16h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M7C	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
04h[17h]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

REGISTER BIT

Table 1-15. MAXQ7665/MAXQ7666 Module 4 Register Bit Functions and Reset Values (continued)

REGISTER	REGISTER BIT															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C0M8C 04h[18h]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M9C 04h[19h]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M10C 04h[1Ah]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M11C 04h[1Bh]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M12C 04h[1Ch]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M13C 04h[1Dh]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M14C 04h[1Eh]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0M15C 04h[1Fh]	—	—	—	—	—	—	—	—	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1-16. MAXQ7665/MAXQ7666 Module 5 Register Bit Functions and Reset Values

REGISTER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VMC 05h[00h]	—	—	—	—	—	—	—	—	—	—	VIOBI1	VIOBI0	VDBI1	VDBI0	VDBR1	VDBR0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s	s
APE 05h[01h]	—	—	—	VIBE	VDBE	VDPE	—	—	PGG2	PGG1	PGG0	TSE	PGAE	—	DACE	ADCE
	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
ACNT 05h[02h]	ADCMX4	ADCMX3	ADCMX2	ADCMX1	ADCMX0	ADCDIF	ADCBIP	—	—	ADCDUL	—	ADCASD	ADCBY	ADCS2	ADCS1	ADCS0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DCNT 05h[03h]	—	—	—	—	—	—	—	—	—	DACLD2	DACLD1	DACLD0	—	—	—	—
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DACI 05h[04h]	—	—	—	—	DACI.11	DACI.10	DACI.9	DACI.8	DACI.7	DACI.6	DACI.5	DACI.4	DACI.3	DACI.2	DACI.1	DACI.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DACO 05h[06h]	—	—	—	—	DACO.11	DACO.10	DACO.9	DACO.8	DACO.7	DACO.6	DACO.5	DACO.4	DACO.3	DACO.2	DACO.1	DACO.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADCD 05h[08h]	—	—	—	—	ADCD.11	ADCD.10	ADCD.9	ADCD.8	ADCD.7	ADCD.6	ADCD.5	ADCD.4	ADCD.3	ADCD.2	ADCD.1	ADCD.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TSO 05h[09h]	TSO.15	TSO.14	TSO.13	TSO.12	TSO.11	TSO.10	TSO.9	TSO.8	TSO.7	TSO.6	TSO.5	TSO.4	TSO.3	TSO.2	TSO.1	TSO.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AIE 05h[0Ah]	—	—	—	—	—	—	—	—	—	HFFIE	VIOBIE	DVBIE	—	AORIE	ADCIE	—
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ASR 05h[0Bh]	VIOLVL	DVLVL	—	—	XHFRY	—	—	—	—	HFFINT	VIOBI	DVBI	—	ADCOV	ADCRY	—
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OSCC* 05h[0Ch]	—	—	—	—	HFOC1	HFOC0	HFIC1	HFIC0	ADCCD2	ADCCD1	ADCCD0	—	—	EXTHF	RCE	HFE
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

*OSCC is cleared to 0002h on power-on reset and is not affected by other forms of reset.

s = Bit affected only by power-on reset and not by other forms of reset. See the register description for more information.

SECTION 2: POWER-SUPPLY/SUPERVISORY MONITORING MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 2: POWER-SUPPLY/SUPERVISORY MONITORING MODULE

The MAXQ7665/MAXQ7666 power-supply/supervisory monitoring module supports dedicated supply pins to independently power analog, digital I/O, and digital core functions. The analog functions and digital I/O are powered from an external +5V supply, while the internal digital core is powered from a +3.3V supply, which can be supplied by an on-chip linear regulator. Except where explicitly noted, the MAXQ7665 and MAXQ7666 support identical features.

The MAXQ7665/MAXQ7666 power-supply/supervisory monitoring module features include the following.

- Dedicated analog supply (+5.0V) and ground pins
- Dedicated digital I/O supply (+5.0V) and ground pins
- Dedicated digital core supply (+3.3V) and ground pins
- On-chip +3.3V linear regulator
- Digital core brownout interrupt and reset voltage monitors
- Digital I/O brownout voltage monitor (can also be used to monitor analog supply)
- User-programmable thresholds for digital core brownout reset and interrupt generation
- User-programmable thresholds for digital I/O brownout interrupt generation
- Five reset sources: power-on, brownout, external, WDT, and internal system

2.1 Architecture

Figure 2-1 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 power-supply/supervisory monitoring module. The MAXQ7665/MAXQ7666 microcontrollers are +5V-powered devices. Three power supplies are used to operate the various modules in the microcontroller. The MAXQ7665/MAXQ7666's digital I/O supply (DVDDIO) uses two +5.0V supply pins to power the digital I/Os. An internal +3.3V linear regulator powers the digital core functions composed of internal CPU, memory, oscillator, and digital peripherals. If required, an external +3.3V supply (DVDD) can instead be used by disabling ($\overline{REGEN}$ pin connected to DVDDIO) the internal regulator. The analog module uses a separate power-supply line (AVDD) to allow additional filtering to maintain superior analog performance.

The MAXQ7665/MAXQ7666 contain two brownout power-supply monitors. One power-supply monitor is dedicated to the DVDDIO for brownouts, while the other monitors brownouts of the DVDD core supply of the microcontroller, and can actually cause a reset if DVDD is too low. The AVDD supply can be connected to the DVDDIO supply lines, and can then also be monitored by the DVDDIO monitor. The power-on reset circuit is integrated into the DVDD power-supply monitor and the default trip level is between 2.7V and 2.99V. The DVDDIO and DVDD brownout detection thresholds are user selectable, and can be configured independently to interrupt the microcontroller when either of the selected thresholds are crossed.
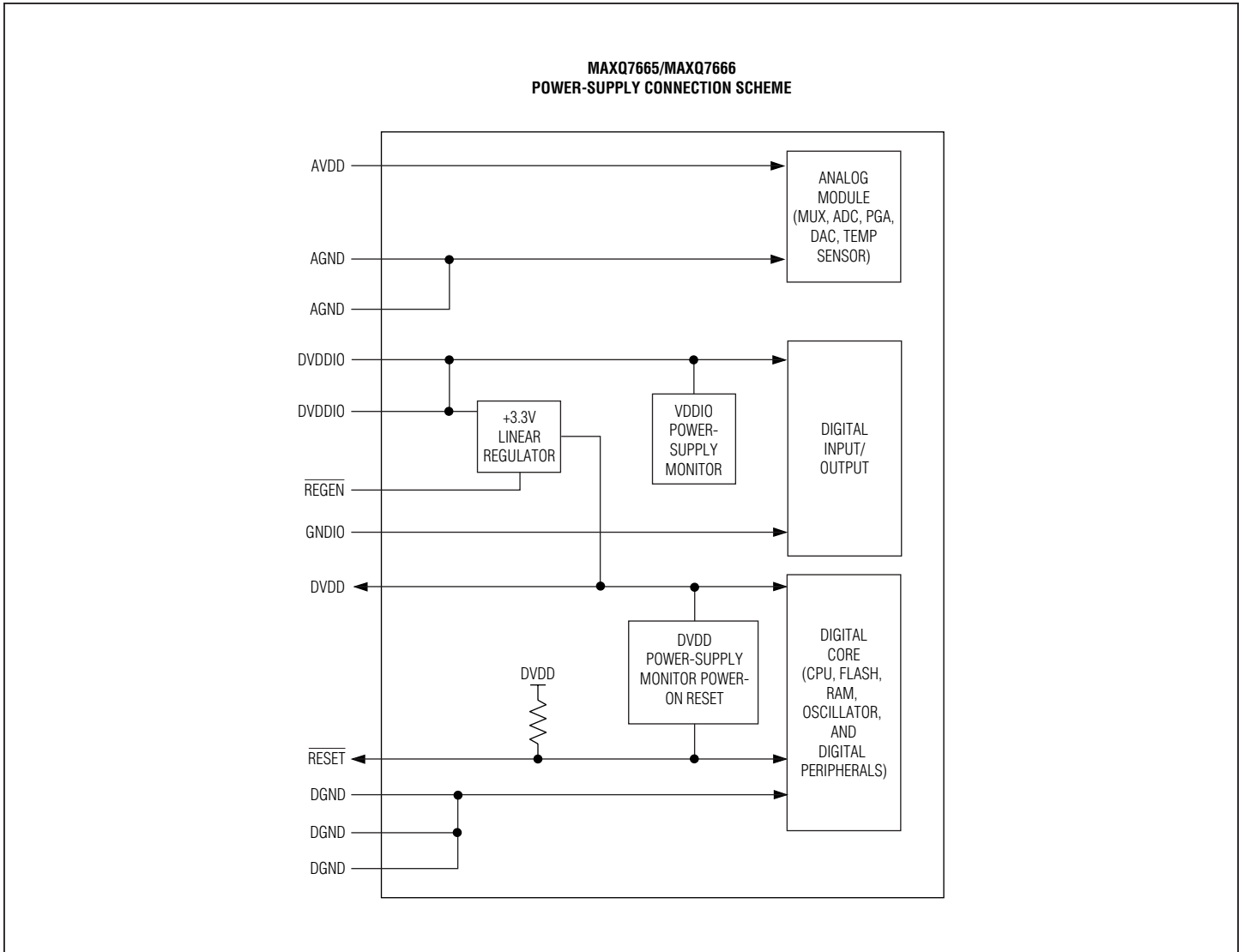
Figure 2-1. MAXQ7665/MAXQ7666 Power-Supply Block Diagram

2.1.1 Power-Supply/Supervisory Module Pins

The power-supply module signals are shown in Table 2-1.

Table 2-1. MAXQ7665/MAXQ7666 Power-Supply/Supervisory Module Pins

POWER-SUPPLY SIGNAL	PIN NUMBER		FUNCTION
	48	**56**	
AVDD	44	50	Analog V_{DD} Supply. AVDD is the power supply for all analog input/output functions including ADC, PGA, DAC, and temperature sensor. For the MAXQ7665/MAXQ7666, the analog supply voltage is +5.0V. If required, connect AVDD to DVDDIO through some supply filtering, which can allow for voltage monitoring on the AVDD line. If AVDD is a separate supply, no voltage monitoring is applied and the supply voltage should not deviate more than ±300mV from DVDDIO. Bypass AVDD to AGND with a 0.1µF capacitor placed as close to the device as possible.
AGND	5, 8	5, 8	Analog Ground*
DVDDIO	26, 39	30, 44	Digital Input/Output Supply Voltage. DVDDIO is the power supply for all digital input/output pins (except XIN, XOUT, and $\overline{RESET}$). For the MAXQ7665/MAXQ7666, the digital I/O supply voltage is +5.0V. DVDDIO also powers the internal +3.3V linear regulator (if used). Bypass DVDDIO to GNDIO with a 0.1µF capacitor placed as close to the device as possible.
GNDIO	27	31	Digital Input/Output Ground. GNDIO is the ground for all the digital I/O pins (except XIN, XOUT, $\overline{RESET}$).*
$\overline{REGEN}$	38	43	Active-Low Linear Power Regulator Enable Input. $\overline{REGEN}$ controls the internal +3.3V linear regulator. When $\overline{REGEN}$ is connected GNDIO, the linear regulator is enabled; when $\overline{REGEN}$ is connected to DVDDIO, the linear regulator is disabled and an external +3.3V supply must be provided to the DVDD pin.
DVDD	40	45	Digital Supply Voltage. DVDD is the power supply for all core CPU functions, flash, RAM, oscillator, and digital peripherals. For the MAXQ7665/MAXQ7666, the digital supply voltage is +3.3V and can be generated by the internal +3.3V linear regulator. Bypass DVDD to DGND with a 4.7µF ±20% capacitor with maximum ESR of 0.5Ω. In addition, bypass DVDD with a 0.1µF capacitor. Place both bypass capacitors as close to the device as possible.
DGND	18, 19, 31	20, 21, 36	Digital Ground. These pins serve as the digital ground for the CPU core functions, flash, SRAM, digital peripherals, and oscillator port.*
$\overline{RESET}$	41	47	Active-Low Reset I/O. This is an active-low open-drain signal with an internal pullup resistor to DVDD. During POR, this pin remains low until DVDD rises above the default power-on reset threshold and a timeout period expires. $\overline{RESET}$ is pulled low by the internal voltage monitoring circuitry if DVDD falls below the selected brownout reset threshold. This pin can also be pulled low externally by the user or internally by the watchdog timer. All these events reset the MAXQ7665/MAXQ7666.

For PCB layout guidelines, refer to Application Note 801 (www.maxim-ic.com/AN801) and Application Note 637 (www.maxim-ic.com/AN637).

2.2 Power-Supply/Supervisory Monitoring Registers

The MAXQ7665/MAXQ7666 power-supply/supervisory monitoring peripheral registers are described here. All these peripheral registers are directly accessible by the microcontroller through the module/index address.

2.2.1 Voltage Monitor Control Register (VMC)

The VMC register contains the DVDD and DVDDIO voltage-monitor threshold select bits. This register is cleared to a default value of 0000h by all forms of reset except bits 1 and 0, which are cleared by power-on reset only.

Register Description: **Voltage Monitor Control Register**
Register Name: **VMC**
Register Address: **Module 05h, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	—	VIOBI1	VIOBI0	VDBI1	VDBI0	VDBR1	VDBR0
Reset	0	0	0	0	0	0	0	0
Access	r	r	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *The VDBR1 and VDBR0 bits are reset only by POR. Other bits are cleared by all forms of reset.*

Bits 15 to 6: Reserved. Read 0, write ignored.

Bits 5, 4: DVDDIO Brownout Interrupt Threshold Bits 1, 0 (VIOBI1, VIOBI0). These bits are used to select the brownout interrupt threshold level for the DVDDIO voltage supply. An interrupt flag (VIOBI) is set if the DVDDIO brownout detection is enabled (VIBE = 1 in the APE register) and the DVDDIO voltage falls in the threshold range selected in the following table. To convert the interrupt flag to an interrupt, the DVDDIO brownout interrupt enable bit (VIOBIE in the AIE register) must be set. Also, global interrupt mask bits IM5 (in the IMR register) and IGE (in the IC register) must be enabled.

VIOBI1:VIOBI0	BROWNOUT INTERRUPT THRESHOLD RANGE (V)*
00	4.25–4.74 (default)
01	4.30–4.79
10	4.35–4.84
11	4.40–4.89

* Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.

Bits 3, 2: DVDD Brownout Interrupt Threshold Bits 1, 0 (VDBI1, VDBI0). These bits are used to select the brownout interrupt threshold level for the DVDD voltage supply. An interrupt flag (DVBI) is set if the DVDD brownout detection is enabled (VDBE = 1 in the APE register) and the DVDD voltage falls in the threshold range (see table below). To convert the interrupt flag to an interrupt, the DVDD brownout interrupt enable bit (DVBIE in the AIE register) must be set. Also, global interrupt mask bits IM5 (in the IMR register) and IGE (in the IC register) must be enabled.

VDBI1: VDBI0	BROWNOUT INTERRUPT THRESHOLD RANGE (V)*
00	2.77–2.99 (default)
01	2.84–3.13
10	2.91–3.20
11	2.99–3.27

Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.

Bits 1, 0: DVDD Brownout Reset Threshold Bits 1, 0 (VDBR1, VDBR0). These bits are used to select the brownout reset threshold level for the DVDD voltage supply. A reset state is generated to halt program execution if the DVDD brownout reset supervisor is enabled (VDPE = 1 in the APE register) and the DVDD voltage falls in the threshold range (see table below). **Note:** The DVDD brownout reset supervisor is enabled (VDPE = 1) by default after all forms of reset.

VDBR1: VDBR0	BROWNOUT RESET THRESHOLD RANGE (V)*
00	2.70–2.99 (default)
01	2.77–3.06
10	2.84–3.13
11	2.91–3.20

Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.

2.2.2 Analog Power Enable Register (APE)

The APE register contains the power-enable bits to control and turn on/off the DVDDIO and DVDD power-supply voltage monitoring.

Register Description: **Analog Power Enable Register**
Register Name: **APE**
Register Address: **Module 05, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	VIBE	VDBE	VDPE	—	—
Reset	0	0	0	0	0	1	0	0
Access	r	r	r	rw	rw	rw	r	r

Bit #	7	6	5	4	3	2	1	0
Name	PGG2	PGG1	PGG0	TSE	PGAE	—	DACE	ADCE
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	r	rw	rw

r = read, w = write
Note: *This register is cleared to 0400h on all forms of reset.*

Bits 15, 14, 13, 9, 8, and 2: Reserved. Read 0, write ignored.

Bit 12: I/O Voltage Brownout Detection Enable (VIBE). The DVDDIO brownout detection is enabled when this bit is set to logic 1. An interrupt request is generated if the DVDDIO brownout interrupt enable (VIOBIE in the AIE register) bit is set and the voltage monitor detects the DVDDIO voltage falling in the threshold range determined by the VIOBI[1:0] bits in the VMC register.

Note: To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and the IM5 mask in the IMR peripheral register.

Bit 11: Digital Voltage Brownout Detection Enable (VDBE). The DVDD brownout detection is enabled when this bit is set to logic 1. An interrupt request is generated if the DVDD brownout interrupt enable (DVBIE in the AIE register) bit is set and the voltage monitor detects the DVDD voltage falls in the threshold range determined by the VDBI[1:0] bits in the VMC register.

Note: To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and the IM5 mask in the IMR peripheral register.

Bit 10: Digital Voltage Brownout Reset Enable (VDPE). The DVDD brownout reset supervisor is enabled when this bit is set to logic 1. A reset state is generated to halt program execution if the DVDD voltage falls in the threshold range determined by the VDBR[1:0] bits in the VMC register. This bit defaults to logic 1 on reset. Clearing this bit to 0 disables the brownout reset supervisor.

Bits 7, 6, 5: PGA Gain Setting Bits 2, 1, 0 (PGG2, PGG1, PGG0). See *Section 3* for more information on these register bits.

Bit 4: Temperature Sensor Enable (TSE). See *Section 3* for more information on this register bit.

Bit 3: Programmable Gain Amp Enable (PGAE). *Section 3* for more information on this register bit.

Bit 1: DAC Enable (DACE). See *Section 3* for more information on this register bit.

Bit 0: ADC Enable (ADCE). See *Section 3* for more information on this register bit.

2.2.3 Analog Interrupt Enable Register (AIE)

The AIE register is used to enable interrupts from a variety of analog sources including DVDDIO and DVDD brownout detection.

Register Description: **Analog Interrupt Enable Register**
Register Name: **AIE**
Register Address: **Module 05h, Index 0Ah**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFIE	VIOBIE	DVBIE	—	AORIE	ADCIE	—
Reset	0	0	0	0	0	0	0	1
Access	r	rw	rw	rw	r	rw	rw	rw

r = read, w = write

Bits 15 to 7 and 3: Reserved. Read 0, write ignored.

Bit 6: External High-Frequency Oscillator Failure Interrupt Enable (HFFIE). See *Section 5* for more information on this register bit.

Bit 5: I/O Voltage Brownout Interrupt Enable (VIOBIE). This bit must be set to logic 1 to generate an interrupt request when a brownout condition is detected on the DVDDIO voltage and the VIOBI flag (in the ASR register) is set to logic 1. Clearing this bit to 0 disables the interrupt capability from the VIOBI flag. **Note:** To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and the IM5 mask in the IMR peripheral register.

Bit 4: Digital Brownout Interrupt Enable (DVBIE). This bit must be set to logic 1 to generate an interrupt request when a brownout condition is detected on the DVDD voltage and the DVBI flag (in the ASR register) is set to logic 1. Clearing this bit to 0 disables the interrupt capability from the DVBI flag. **Note:** To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and the IM5 mask in the IMR peripheral register.

Bit 2: ADC Overrun Interrupt Enable (AORIE). See *Section 3* for more information on this register bit.

Bit 1: ADC Data Ready Interrupt Enable (ADCIE). See *Section 3* for more information on this register bit.

Bit 0: This bit is implemented and available to be used as a user-software-controlled bit.

2.2.4 Analog Status Register (ASR)

The ASR register reports the status of the DVDD and DVDDIO supply brownout detection.

Register Description: **Analog Status Register**
Register Name: **ASR**
Register Address: **Module 05h, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	VIOLVL	DVLVL	—	—	XHFRY	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFINT	VIOBI	DVBI	—	ADCOV	ADCRY	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *The ADCOV bit is cleared by all forms of reset. All other bits are reset only by POR. Reading the ASR resets to 0 all the status flag bits except VIOLVL and DVLVL.*

Bit 15: I/O Voltage Brownout Comparator Level (VIOLVL). This bit reflects the DVDDIO voltage brownout comparator's current output state when read. This bit is set to logic 1 when the DVDDIO supply is higher than the threshold level (as programmed by the VIOBI[1:0] threshold bits in the VMC register) and is cleared to logic 0 when the supply voltage is below the threshold level. At power-up or when the DVDDIO voltage monitor is disabled (VIBE = 0), this bit is cleared to 0.

Bit 14: Digital Voltage Brownout Comparator Level (DVLVL). This bit reflects the DVDD voltage brownout comparator's current output state when read. This bit is set to logic 1 when the DVDD supply is higher than the threshold level (as programmed by the VDBI[1:0] threshold bits in the VMC register) and is cleared to logic 0 when the supply voltage is below the threshold level. At power-up or when the DVDD voltage monitor is disabled (VDBE = 0), this bit is cleared to 0.

Bits 13, 12, 10 to 7, 3, and 0: Reserved. Read 0, write ignored.

Bit 11: High-Frequency Oscillator Ready (XHFRY). See *Section 5* for more information on this register bit.

Bit 6: External High-Frequency Oscillator Failure Flag (HFFINT). See *Section 5* for more information on this register bit.

Bit 5: I/O Voltage Brownout Flag (VIOBI). This flag is set to logic 1 when a brownout interrupt condition is detected on the DVDDIO supply voltage. This bit is cleared after reading from the ASR register. If enabled (VIOBIE = 1), the DVDDIO brownout interrupt is generated by this register bit.

Bit 4: Digital Brownout Flag (DVBI). This flag is set to logic 1 when a brownout interrupt condition is detected on the DVDD supply voltage. This bit is cleared after reading from the ASR register. If enabled (DVBIE = 1), the DVDD brownout interrupt is generated by this register bit.

Bit 2: ADC Overrun Flag (ADCOV). See *Section 3* for more information on this register bit.

Bit 1: ADC Data Ready Flag (ADCRY). See *Section 3* for more information on this register bit.

2.3 Supply Configuration

The MAXQ7665/MAXQ7666 use three supplies to power the internal analog, digital core, and digital I/O circuits. The supplies are configured as listed:

- AVDD = +5V
- DVDD = +3.3V (with internal linear regulator enabled or through an external supply)
- DVDDIO = +5V

Figure 2-2 and Figure 2-3 show the recommended supply configurations. Bypass capacitors should be mounted as close as possible to the body of the MAXQ7665/MAXQ7666 to reduce noise. **Note:** If AVDD is a separate supply, no voltage monitoring is applied and the supply voltage should not deviate more than ±300mV from DVDDIO. For PCB layout guidelines, refer to *Application Note 801* (**www.maxim-ic.com/AN801**) and *Application Note 637* (**www.maxim-ic.com/AN637**).
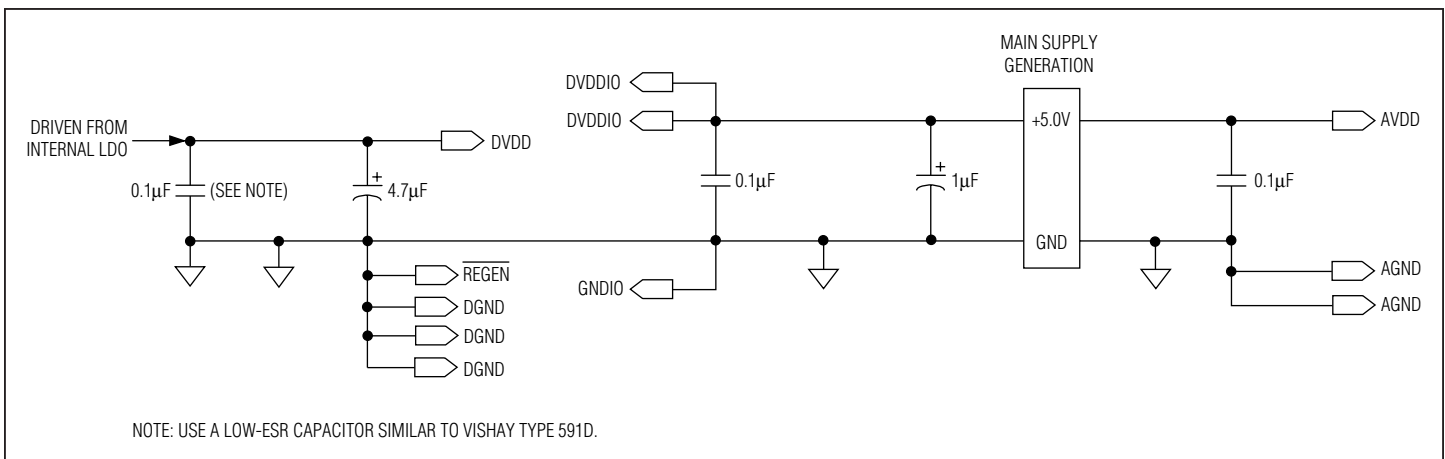
Figure 2-2. Supply Configuration 1 (Using Internal Linear Regulator)
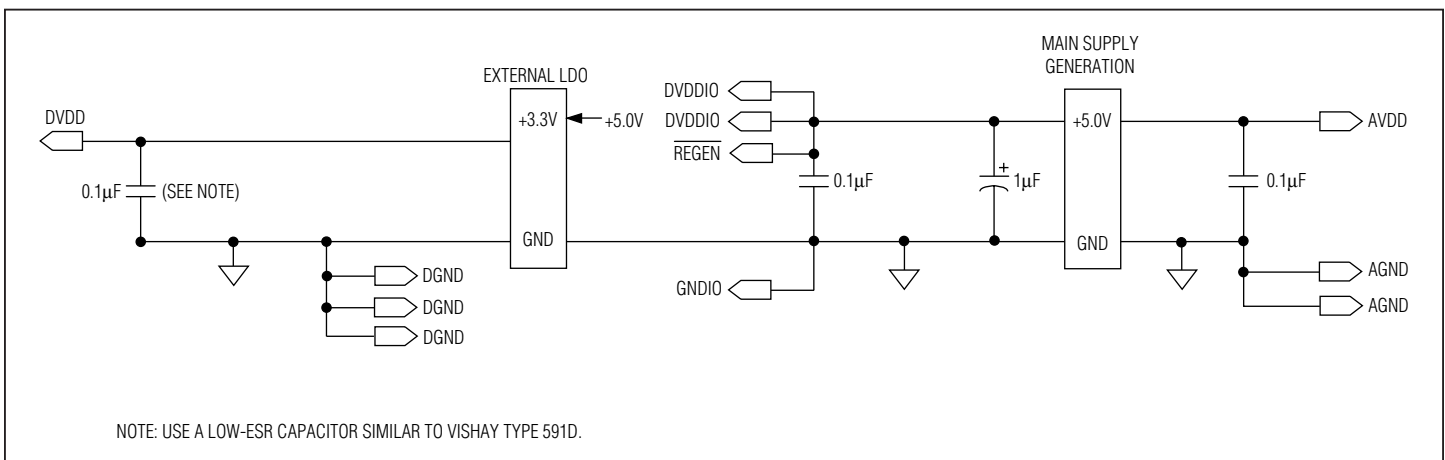
Figure 2-3. Supply Configuration 2 (External DVDD)

2.4 Linear Regulator

The MAXQ7665/MAXQ7666 contain a +3.3V, low dropout (LDO) linear regulator. The regulator powers the MAXQ7665/MAXQ7666 digital core functions including the CPU, flash, SRAM, oscillator, and all the digital peripherals. The linear regulator is powered by the +5V DVDDIO supply. The $\overline{REGEN}$ signal must be connected to GNDIO to enable the internal regulator. When the internal linear regulator is disabled ($\overline{REGEN}$ connected to DVDDIO), an external +3.3V supply must be used.

2.5 Power-On Reset

The MAXQ7665/MAXQ7666 support an on-chip power-on reset (POR) circuit to ensure proper initialization of internal device states. The POR circuit provides a power-on rising voltage threshold and a minimum power-on delay sufficient to accomplish this initialization. When power is first applied to the MAXQ7665/MAXQ7666, the MAXQ7665/MAXQ7666 are held in a power-on reset state (Figure 2-4). The MAXQ7665/MAXQ7666 power-on circuitry (POR) monitors the DVDD supply voltage in relation to the on-chip band gap voltage reference. On power-up, once DVDD exceeds ~1.2V, the $\overline{RESET}$ pin is asserted to be logic-low. All the internal system and peripheral registers are reset if DVDD from cold start exceeds ~1.2. Also, above this voltage, the power-on-reset delay counter is started.

For the MAXQ7665/MAXQ7666 to exit power-on reset, the following two conditions must apply:

- DVDD is above the power-on-reset rising voltage threshold level V_{RST} (2.7V–2.99V power-on default)
- The internal RC oscillator has completed 65,536 cycles (power-on-reset delay for power supply to stabilize; about 8.6ms at 7.6MHz)

Once the power-up period has elapsed, the reset condition is removed automatically ($\overline{RESET}$ pin goes high) and software execution will begin at the reset vector location 8000h (in the utility ROM). Software can determine whether a reset was caused by a power-on reset by checking the POR flag in the WDCN register. This flag is set to 1 following a power-on reset, and should be cleared by software after it has been checked.
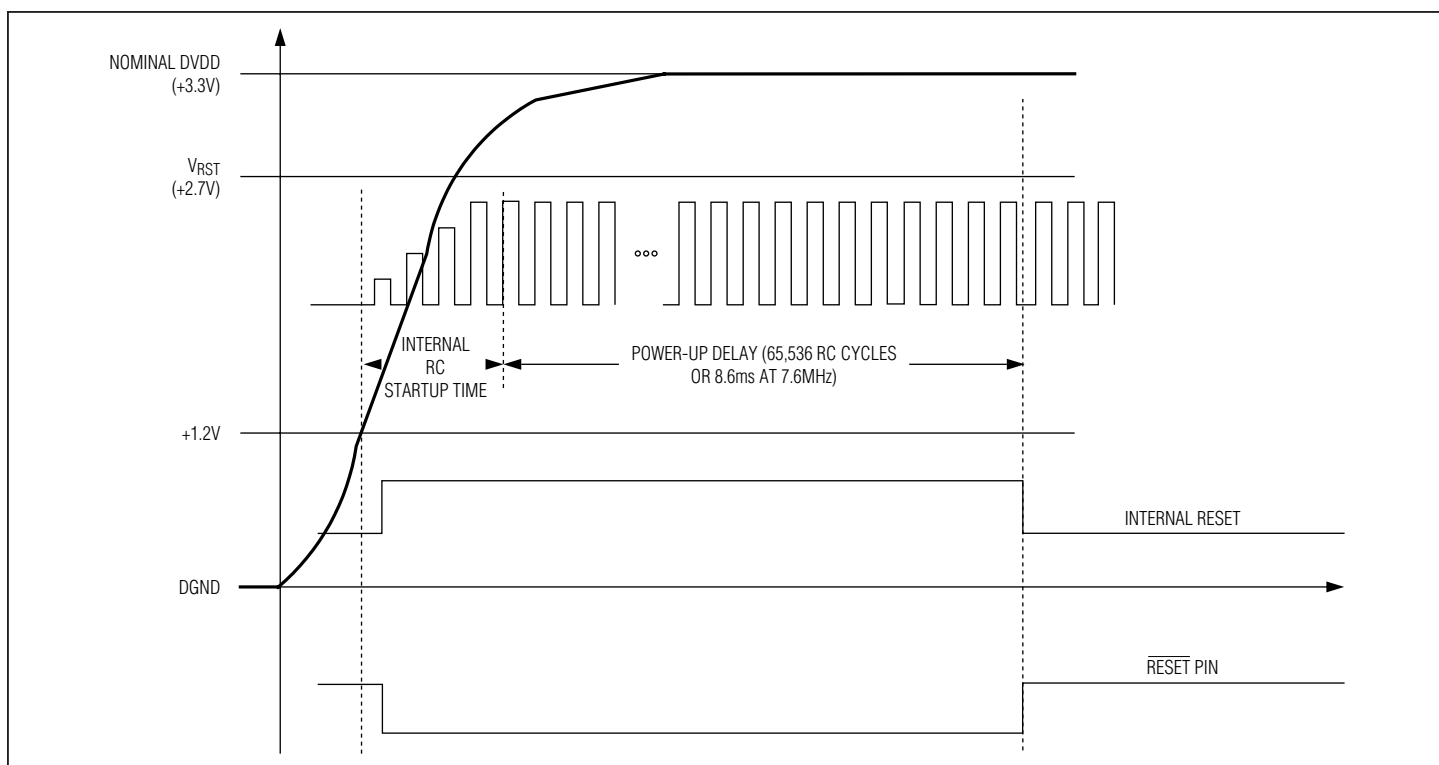
Figure 2-4. MAXQ7665/MAXQ7666 Power-On Reset

Note: In a brownout reset (BOR) situation (see *Section 2.5.2*), where the voltage drops below the DVDD BOR threshold (e.g., 2.7V) and rises back above the default power-on-reset rising voltage threshold level (2.7V), the POR flag in WDCN register will not be set unless DVDD drops below ~1.2V. The POR flag will be set if DVDD voltage drops below ~1.2V and rises back above the default POR rising voltage threshold (2.7V). In such a case, the MAXQ7665/MAXQ7666 go through a complete POR reset as described above.

2.5.1 Power-Up Counter

An independent power-up counter functions as the startup counter to count 65,536 cycles of the internal 7.6MHz RC oscillator from initial power-on. This time period is verified by the counter after the DVDD level reaches the reset threshold (V_{RST}). The counter is active only during initial power-up and is completely shut off during normal operation.

2.5.2 DVDD Brownout Reset (BOR)

The DVDD brownout reset monitoring is enabled when the VDPE bit in the APE register is set to logic 1. The BOR circuitry monitors the DVDD voltage and invokes a brownout reset state to halt program execution if the DVDD voltage falls in the threshold range determined by the VDBR[1:0] bits in the VMC register. The MAXQ7665/MAXQ7666 are held in the brownout reset state (Figure 2-5) while the DVDD voltage is below the reset threshold level and the $\overline{RESET}$ pin is asserted to be logic-low. Table 2-2 shows the supported brownout reset threshold range. When the DVDD power sources return above the threshold level, a brownout reset cycle is performed. For the MAXQ7665/MAXQ7666 to exit brownout reset, the following condition must apply:

- DVDD is above the brownout reset threshold level determined by the VDBR[1:0] bits.

Once the above condition is satisfied, the brownout reset condition is removed automatically ($\overline{RESET}$ pin goes high) and software execution will begin at the reset vector location 8000h (in the utility ROM). A brownout reset cycle is similar to power-on reset cycle shown in Figure 2-4, except that there is no power-up counter delay, V_{RST} is determined by the VDBR[1:0] bits, and some selected register bits are maintained and not reset to default state. For example, the VDBR1 and VDBR0 bits are only cleared by POR, not by BOR.

A brownout reset caused by a DVDD drop below the selected threshold level appears to be the same as a power-on reset, only if DVDD voltage falls below ~1.2V and rises back above the default POR rising voltage threshold. In such a case, the MAXQ7665/MAXQ7666 go through a complete POR reset (see Figure 2-6) as described in Section 2.5 and the POR flag in the WDCN register will be set.

Note: The DVDD brownout reset monitoring is enabled (VDPE = 1) by default after all forms of reset. The VDBR1 and VDBR0 bits are only cleared by POR (only if DVDD goes below ~1.2V in the case of BOR) and retain the selected level after all other forms of reset.

Table 2-2. DVDD Brownout Reset Threshold Range

VDBR1:VDBR0	DVDD BROWNOUT RESET THRESHOLD RANGE (V)*
00	2.70–2.99 (default)
01	2.77–3.06
10	2.84–3.13
11	2.91–3.20

* Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.
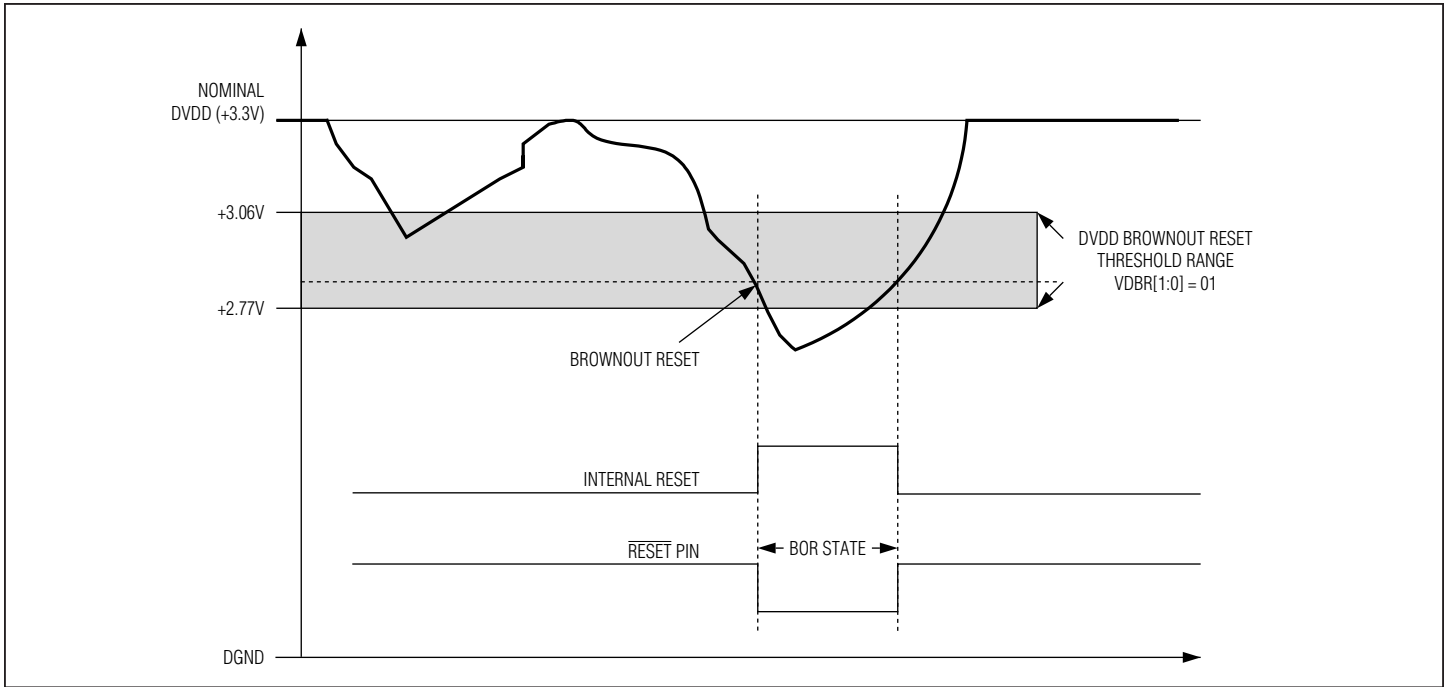
Figure 2-5. MAXQ7665/MAXQ7666 Brownout Reset
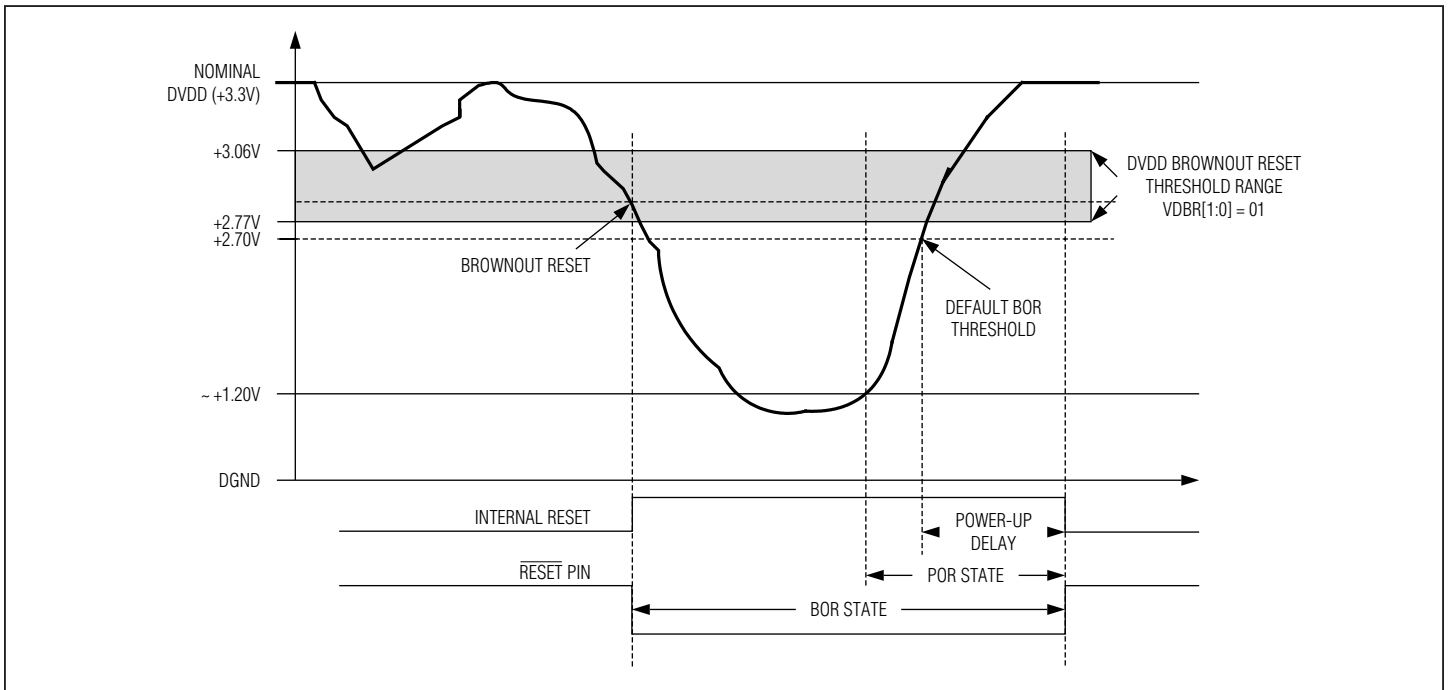
Figure 2-6. MAXQ7665/MAXQ7666 Brownout/Power-On Reset

2.5.3 Reset Output

The MAXQ7665/MAXQ7666 assert the $\overline{RESET}$ signal during power-up and also during reset conditions caused by an internal source (such as brownout, watchdog, or internal reset). On power-up, once DVDD exceeds 1.2V, $\overline{RESET}$ is asserted to be logic-low. As DVDD rises, $\overline{RESET}$ remains low. When DVDD exceeds the default BOR threshold, $\overline{RESET}$ is kept low until the internal RC oscillator has completed 65,536 cycles; after this period, if DVDD remains above the default BOR threshold, $\overline{RESET}$ goes high. If a brownout reset condition occurs, $\overline{RESET}$ is asserted low. Each time a DVDD BOR reset is triggered, it stays low until DVDD exceeds the BOR reset threshold.

Note: The $\overline{RESET}$ pin is an output and an input. The MAXQ7665/MAXQ7666 is placed into an external reset mode if the $\overline{RESET}$ pin is held low for at least four clock cycles. See *Section 2.7.2* for more information on external reset.

2.6 Power-Supply Voltage Monitors

The MAXQ7665/MAXQ7666 contain two power-supply voltage monitors that can be used to continually monitor the DVDD and DVDDIO supply voltages for brownout conditions and initiate interrupt requests if enabled. The DVDD and DVDDIO voltage monitors can be independently activated by programming the corresponding enable bits (VDBE and VIBE) in the analog power-enable (APE) register.

2.6.1 Digital Core Supply (DVDD) Monitor

The digital core supply monitor detects a brownout condition on the +3.3V DVDD supply. The DVDD supply monitor can be independently activated by programming the corresponding enable bit (VDBE) in the analog power enable (APE) register. A brownout is detected when the DVDD supply voltage drops below the programmed brownout detection threshold (Figure 2-7). The brownout interrupt threshold level is user selectable, and can be programmed using the brownout interrupt threshold bits (VDBI[1:0]) in the VMC register. The supported threshold levels are listed in Table 2-3. If enabled, a DVDD brownout interrupt can be generated that allows for saving data and the present state of the MAXQ7665/MAXQ7666. A DVDD brownout interrupt is generated only if the interrupt enable bit (DVBIE) in the analog interrupt enable (AIE) register is set. Also, global interrupt mask bits IM5 (in the IMR register) and IGE in (the IC register) must be enabled.

If the DVDD supply falls further, then the brownout reset threshold is tripped, terminating program operation and holding the MAXQ7665/MAXQ7666 in the brownout reset state. The MAXQ7665/MAXQ7666 remain in the brownout reset state until the supply rises above the reset threshold. The DVDD monitor brownout interrupt and reset trip points can differ from device to device within the programmed threshold range. This tolerance error is caused by the monitor comparator offsets, and threshold setting circuitry. The brownout interrupt and reset thresholds will track each other to some degree. If the brownout interrupt trip point is trending towards the lower side of the threshold level, then the brownout reset trip point will also trend towards the lower side of the threshold level. The brownout reset is always below the brownout interrupt threshold for equivalent settings, ensuring adequate notice of a failing supply condition.

Table 2-3. DVDD Brownout Interrupt Threshold Range

VDBI[1:0]	DVDD (CORE) (V)*
00 (default)	2.77–2.99
01	2.84–3.13
10	2.91–3.20
11	2.99–3.27

* Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.
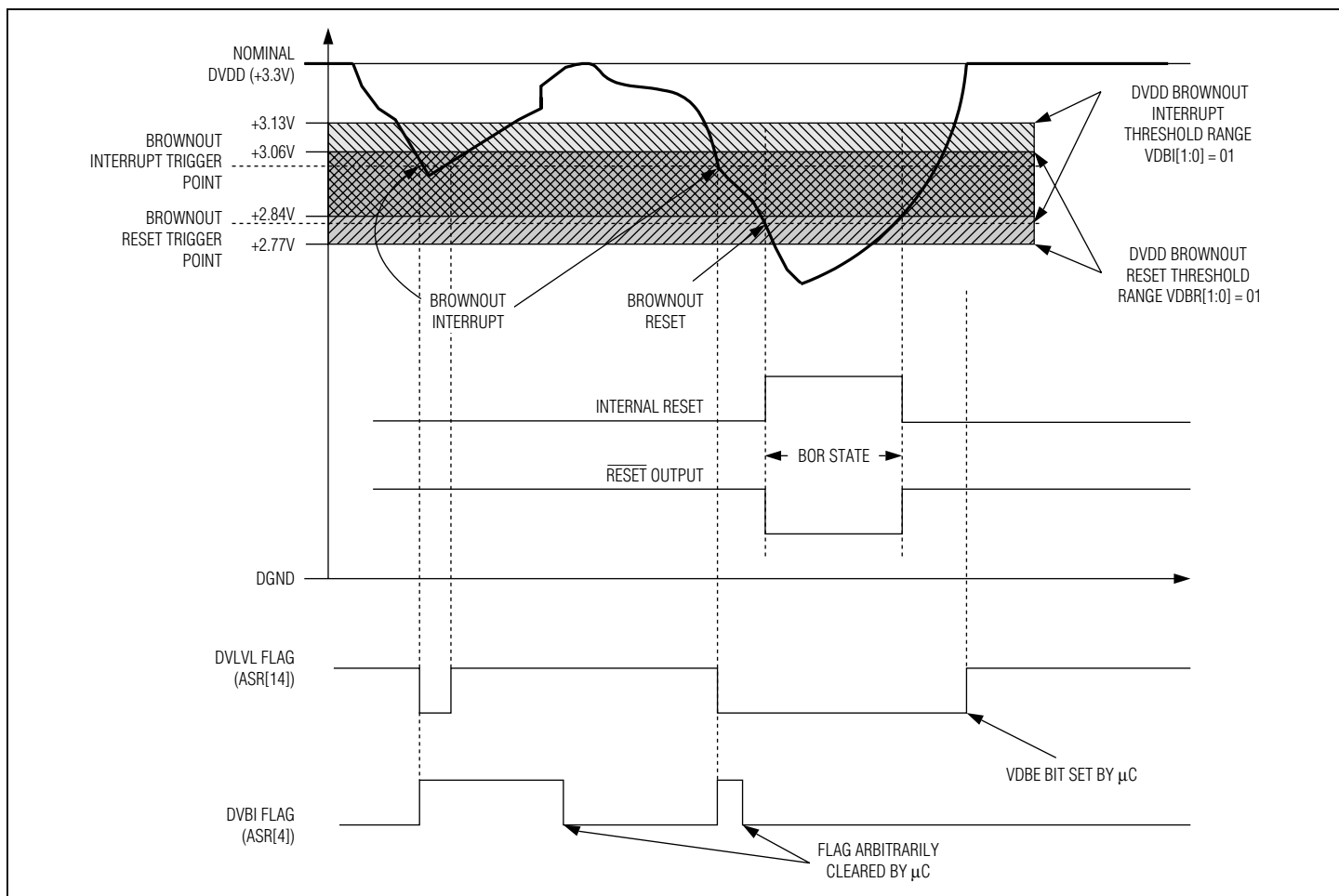
Figure 2-7. DVDD Brownout Interrupt Threshold Detection

2.6.2 Digital I/O Supply (DVDDIO) Monitor

The DVDDIO monitor detects a brownout condition on the +5V digital I/O supply. The DVDDIO supply monitor can be independently activated by programming the corresponding enable bit (VIBE) in the APE register. A brownout is detected when the DVDDIO supply voltage falls in the programmed DVDDIO brownout detection threshold range (Figure 2-8). The brownout interrupt threshold range is user selectable, and can be programmed using the brownout interrupt threshold bits (VIOBI[1:0]) in the VMC register. The supported threshold range are listed in Table 2-4. If enabled, an interrupt can be generated that allows for saving data and the present state of the MAXQ7665/MAXQ7666. A DVDDIO brownout interrupt is generated only if the interrupt enable bit (VIOBIE) in the AIE register is set. Also, global interrupt mask bits IM5 (in the IMR register) and IGE in (the IC register) must be enabled.

Table 2-4. DVDDIO Brownout Interrupt Threshold Range

VIOBI[1:0]	DVDDIO (V)*
00 (default)	4.25–4.74
01	4.30–4.79
10	4.35–4.84
11	4.40–4.89

* Reconfirm the values provided in this table with those in the latest MAXQ7665 and MAXQ7666 data sheets.
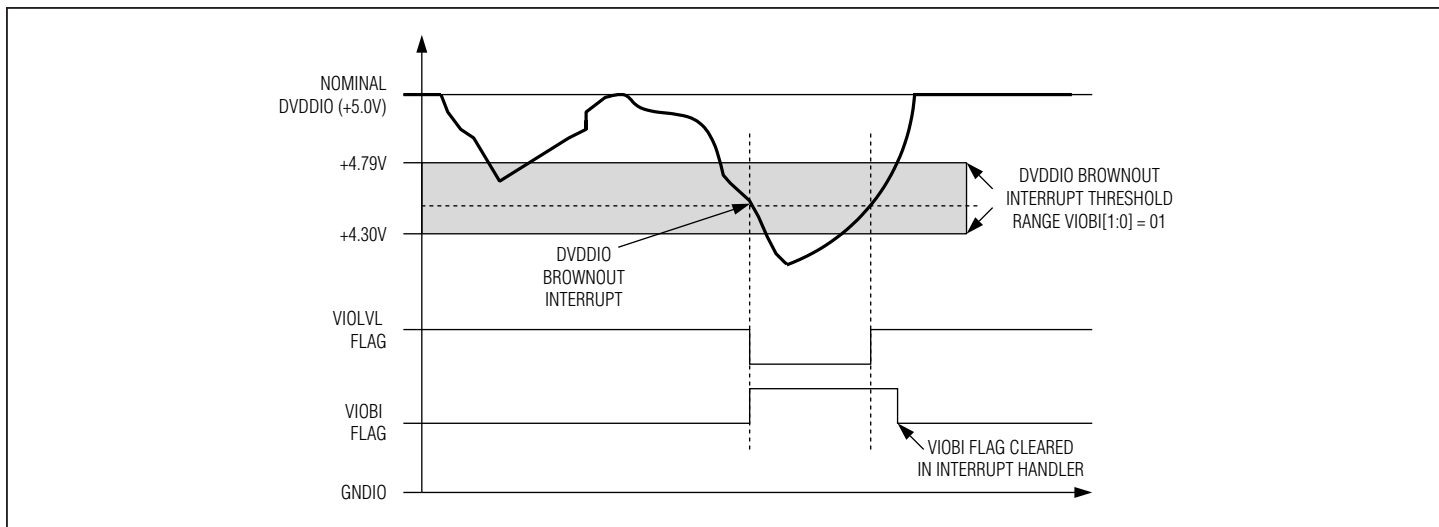
Figure 2-8. DVDDIO Brownout Interrupt Threshold Detection

2.7 Reset Mode

When the MAXQ7665/MAXQ7666 are in reset mode, the enabled system clock oscillator continues running, but no instruction execution or other system or peripheral operations occur, and all input/output pins return to default states. Once the condition that caused the reset (whether internal or external) is removed, code execution resumes at address 8000h for all reset types. Some of the reset sources will also trigger a delaying count of 65,536 clocks (as discussed above) before execution starts.

There are five different sources that can cause the MAXQ7665/MAXQ7666 to enter reset mode. See *Section 2.5* for information on power-on and brownout reset.

- Power-on reset
- Brownout reset
- Watchdog timer reset
- External reset
- Internal system reset

2.7.1 Watchdog Timer Reset

The MAXQ7665/MAXQ7666 watchdog timer is described in *Section 5*. The watchdog timer is a programmable hardware timer that can be set to reset the MAXQ7665/MAXQ7666 in the case of a software lockup or other unrecoverable error. Once the watchdog is enabled in this manner, the processor must refresh the watchdog periodically to avoid a reset. If the processor does not reset the watchdog timer before it elapses, the watchdog will initiate a reset state. When running at 7.6MHz, the maximum watchdog time period before reset is approximately 276ms.

If the watchdog resets the MAXQ7665/MAXQ7666, it remains in reset and holds the $\overline{RESET}$ pin low for four clock cycles. Once the reset condition has completed, the processor will begin executing program code at address 8000h. When a reset occurs due to a watchdog timeout, the watchdog timer reset flag in the WDCN register is set to 1 and can only be cleared by software. User software can examine this bit following a reset to determine if that reset was caused by a watchdog timeout.

Since the XT bit in the CKCN register and the HFE bit in the OSCC register are cleared to 0 only on power-on reset, it is possible to exit a watchdog reset with the clock source set to the high frequency crystal oscillator. In this case, execution resumes running from the RC oscillator, and the switchover to the high-frequency oscillator occurs automatically when the crystal oscillator is ready.
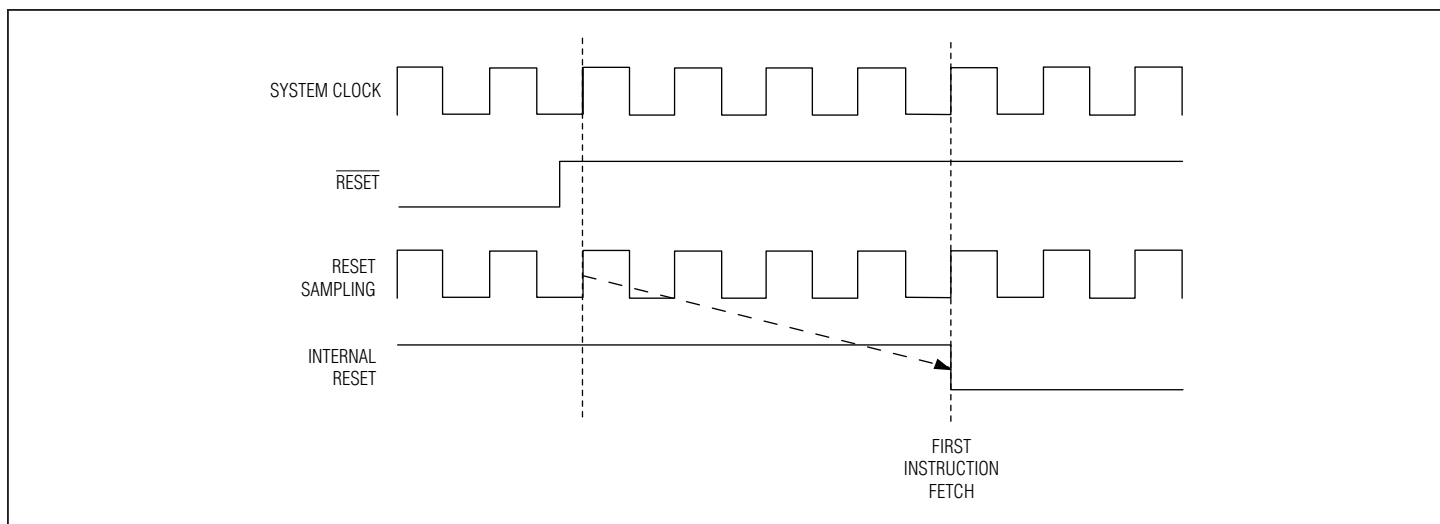
Figure 2-9. MAXQ7665/MAXQ7666 External Reset

2.7.2 External Reset

During normal operation, the MAXQ7665/MAXQ7666 devices are placed into an external reset mode by holding the $\overline{RESET}$ pin low for at least four clock cycles. If the MAXQ7665/MAXQ7666 devices are in the low-power stop mode (i.e., system clock is not active), the $\overline{RESET}$ pin becomes an asynchronous source, forcing the reset state immediately after being taken to logic 0. Once the MAXQ7665/MAXQ7666 enter reset mode, it remains in reset as long as the $\overline{RESET}$ pin is held at logic 0. After the $\overline{RESET}$ pin returns to logic 1, the processor starts the internal 7.6MHz RC oscillator if necessary and exits the reset state within four clock cycles (Figure 2-9) and begins program execution at address 8000h.

The $\overline{RESET}$ pin is an output and an input. If a reset condition is caused by an internal source (such as a brownout reset, watchdog, or internal reset), an output reset pulse or low level is generated at the $\overline{RESET}$ pin as long as the MAXQ7665/MAXQ7666 remain in reset. If the $\overline{RESET}$ pin is connected to an incompatible external reset circuit, it may not be able to drive the output reset signal. However, if this occurs it does not affect the internal reset condition.

Because the XT bit in the CKCN register and the HFE bit in the OSCC register are cleared to 0 only on power-on reset, it is possible to exit an external reset with the clock source set to the high-frequency crystal oscillator. In this case, execution resumes running from the RC oscillator, and the switchover to the high-frequency oscillator occurs automatically when the crystal oscillator is ready.

2.7.3 Internal System Reset

The MAXQ7665/MAXQ7666 support internal system reset capability from in-system programming mode. An internal system reset is generated when the ROD bit in the system control register is set. The SPE bit in the ICDF register must also be set. The bootloader software can use this capability to initiate an internal system reset when the flash loader completes its operation. See *Section 12* for more details on in-system programming.

SECTION 3: ANALOG I/O MODULE

This section contains the following information:

MAXQ7665/MAXQ7666 User's Guide

LIST OF FIGURES

LIST OF TABLES

MAXQ7665/MAXQ7666 User's Guide

SECTION 3: ANALOG I/O MODULE

The MAXQ7665/MAXQ7666 contain an ultra-low-power precision analog I/O module for measuring and controlling a host of sensors, motors, bridges, and other analog peripherals. The analog I/O module has all the components to make the MAXQ7665/MAXQ7666 stand-alone data-acquisition machines ideal for harsh environment applications. Except where explicitly noted, the MAXQ7665 and MAXQ7666 support identical features.

The analog I/O module includes the following features:

- 8 differential analog-input multiplexer
- Low-power, 12-bit, 500ksps successive approximation ADC
- 12-bit, buffered, voltage-output DAC
- On-chip ±1°C accurate temperature sensor (typ)
- Remote temperature sensor drive circuit
- Internal programmable gain amplifier (x1, x2, x4, x8, x16, x32)
- Individual external reference inputs for the ADC and DAC

3.1 Architecture

The analog-input multiplexer supports 8 differential measurements and feeds the programmable gain amplifier (PGA) and the 12-bit SAR ADC. The low noise, programmable gain amplifier with gains of x1 to x32 allows interfacing to a variety of devices with different signal amplitudes. The ADC conversion clock source is the same as the system clock source (internal oscillator or external crystal/clock) and has a user-programmable clock division ratio. The 12-bit voltage-out DAC has internal feedback resistors for reducing external component count. The internal temperature sensor performs temperature measurements with an internal diode-connected transistor. In the remote temperature sensor drive configuration, the device provides the proper bias necessary to measure temperature with up to two external diode-connected transistor sensors. The MAXQ7665/MAXQ7666 support independent external reference inputs for the ADC and DAC to allow the use of high precision, high quality reference sources.

Figure 3-1 shows a functional block diagram of the MAXQ7665/MAXQ7666 analog I/O module.
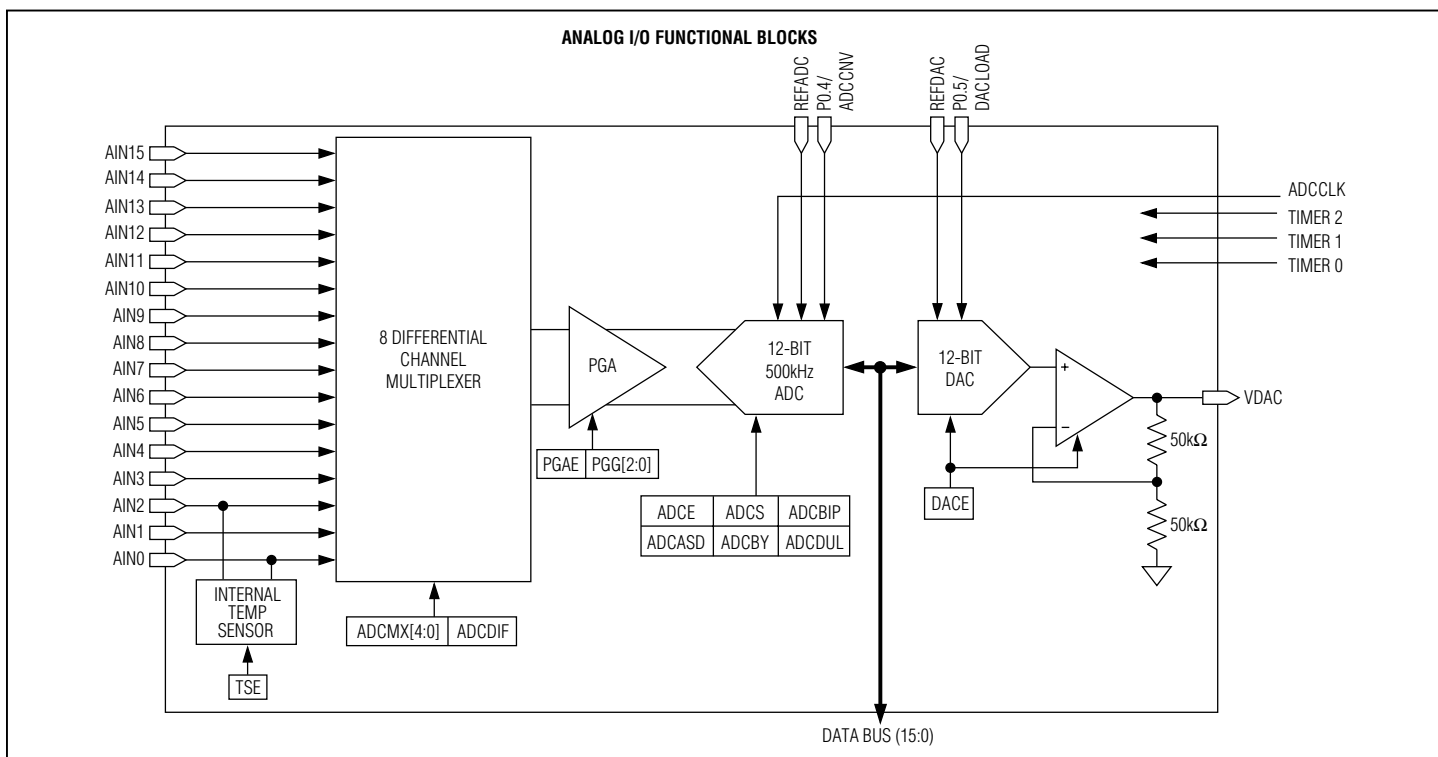
Figure 3-1. Analog I/O Module Block Diagram

3.1.1 Analog I/O Pins

The analog I/O module has 24 pins associated with the analog functions on the microcontroller. Table 3-1 shows the external interface signals used by the analog I/O module.

Table 3-1. Analog I/O Module Signals

SIGNAL	FUNCTION
AIN15	
AIN14	
AIN13	
AIN12	
AIN11	
AIN10	
AIN9	ADC Analog Input #. These are dedicated analog input pins connected through the internal analog multiplexer to the PGA and ADC. The analog multiplexer supports 8 differential-input measurements. In differential-input mode, the inputs are paired: AIN14 to AIN15, AIN12 to AIN13, AIN10 to AIN11, AIN8 to AIN9, AIN6 to AIN7, AIN4 to AIN5, AIN2 to AIN3, AIN0 to AIN1.
AIN8	
AIN7	
AIN6	
AIN5	
AIN4	
AIN3	
AIN1	
AIN2/TS2	ADC Analog Input/Remote Temperature Sensor. Analog input pins AIN2 and AIN0 are shared with the remote temperature-sensor drive line. If the remote temperature-sensor drive circuit is not selected, the pin can be used as a differential input to the multiplexer. In differential-input configuration, AIN2 is referenced to AIN3 while AIN0 is referenced to AIN1. When selected, the remote temperature-sensor drive circuit supplies suitable current to drive an external diode-connected transistor to monitor temperature away from the microcontroller. The remote temperature measurement can be made either in single-ended or differential configuration. Note: In differential configuration, AIN3 is used as the return path for AIN2 and AIN1 is used as the return path for AIN0.
AIN0/TS0	
DACOUT	DAC Voltage Output. DACOUT is a dedicated output pin. If the DAC is disabled, the pin is configured as a 100kΩ pulldown resistor to AGND. The DACOUT line can be used for precision drive applications.
P0.4/ADCCNV	ADC Conversion Start Input/Port 0 Data Bit 4. The ADC conversion start is a shared pin with the general-purpose digital I/O port 0 bit 4. As ADCCNV, this pin can trigger ADC sampling and conversion on a rising or falling edge. After power-up or a reset this pin defaults to a digital I/O port pin with pullup enabled.
P0.5/DACLOAD	DAC Load Input/Port 0 Data Bit 5. The DAC load is a shared pin with the general-purpose digital I/O port 0 bit 5. As DACLOAD, this pin can trigger DAC conversion by loading the DAC output register on a rising or falling edge. After power-up or a reset this pin defaults to a digital I/O port pin with pullup enabled.
REFADC	ADC Reference Input. The REFADC input pin is used to supply an external precision voltage reference to the ADC. The REFADC can handle a voltage range from 1V to AVDD. The REFADC input determines the full-scale range (FSR) of the internal 12-bit ADC.
REFDAC	DAC Reference Input. The REFDAC input pin is used to supply an external precision voltage reference to the DAC. The REFDAC can handle a voltage range from 0 to AVDD. The REFDAC input determine the full-scale range (FSR) of the internal 12-bit DAC.
AVDD	Analog V_{DD} Supply. The analog supply voltage is +5.0V for the MAXQ7665/MAXQ7666.
AGND	Analog Ground (2 pins)
EP	Exposed Paddle. The MAXQ7665/MAXQ7666 TQFN package has an exposed paddle on the bottom of the package, providing a very low thermal resistance path for heat removal from the IC, as well as low-inductance path to ground. The pad is electrically connected to AGND and should be soldered to the circuit board analog ground plane for proper thermal and electrical performance. Refer to Maxim's Application Note HFAN-08.1: *Thermal Considerations for QFN and Other Exposed Pad Packages* for additional information.

MAXQ7665/MAXQ7666 User's Guide

3.2 Analog I/O Module Control and Status Registers

The analog I/O module uses the following control and status registers.

3.2.1 Analog Power Enable Register (APE)

Register Description: **Analog Power Enable Register**
Register Name: **APE**
Register Address: **Module 05h, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	VIBE	VDBE	VDPE	—	—
Reset	0	0	0	0	0	1	0	0
Access	r	r	r	rw	rw	rw	r	r

Bit #	7	6	5	4	3	2	1	0
Name	PGG2	PGG1	PGG0	TSE	PGAE	—	DACE	ADCE
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	r	rw	rw

r = read, w = write
Note: *This register is cleared to 0400h on all forms of reset.*

Bits 15, 14, 13, 9, 8, and 2: Reserved. Read 0, write ignored.

Bit 12: I/O Voltage Brownout Detection Enable (VIBE). See *Section 2* for details on this bit.

Bit 11: Digital Voltage Brownout Detection Enable (VDBE). See *Section 2* for details on this bit.

Bit 10: Digital Voltage Reset Enable (VDPE). See *Section 2* for details on this bit.

Bits 7, 6, 5: PGA Gain Setting Bits 2, 1, 0 (PGG2, PGG1, PGG0). These bits set the PGA gain as shown in the following table. The PGA is bypassed when the PGA gain selected is 1.

PGG2:PGG0	PGA GAIN
000	1 (Default)
001	2
010	4
011	8
100	16
101	32
110	Reserved, should not be used
111	Reserved, should not be used

Bit 4: Temperature Sensor Enable (TSE). Setting this bit to logic 1 enables the temperature sensor. Clearing this bit to logic 0 turns off the power to the temperature sensor and disables its operation. The ADCMX4:ADCMX0 bits in the ADC control register determine if the internal or external temperature sensor configuration is used.

Bit 3: PGA Enable (PGAE). The PGA is enabled when this bit is set to logic 1. Clearing this bit to 0 disables the PGA. The PGAE should be enabled 5µs before attempting a conversion with a PGA gain other than 1. **Note:** To bypass the PGA, select a PGA gain of 1 (PGG2:PGG0) and clear the PGAE bit to 0. Setting PGAE = 0 significantly reduces power consumption.

Bit 1: DAC Enable (DACE). Setting this bit to logic 1 enables the DAC block to be ready for conversion. Clearing this bit to logic 0 turns off the power to the DAC block and disables its operation.

Bit 0: ADC Enable (ADCE). Setting this bit to logic 1 enables the ADC block to be ready for conversion. Clearing this bit to logic 0 turns off the power to the ADC block and disables its operation.

3.2.2 ADC Control Register (ACNT)

Register Description: **ADC Control Register**
Register Name: **ACNT**
Register Address: **Module 05h, Index 02h**

Bit #	15	14	13	12	11	10	9	8
Name	ADCMX4	ADCMX3	ADCMX2	ADCMX1	ADCMX0	ADCDIF	ADCBIP	—
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	r

Bit #	7	6	5	4	3	2	1	0
Name	—	ADCDUL	—	ADCASD	ADCBY	ADCS2	ADCS1	ADCS0
Reset	0	0	0	0	0	0	0	0
Access	r	rw	r	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 11: ADC Input Multiplexer Bits 4 to 0 (ADCMX4 to ADCMX0). These multiplexer bits select the inputs to the ADC and control the state of the temperature sensor.

ADCMX4:ADCMX0	ADCDIF = 1		MEASURE
	POSITIVE	NEGATIVE	
00000	AIN0	AIN1	Voltage
00001	AIN1	AIN1	Offset Voltage
00010	AIN2	AIN3	Voltage
00011	AIN3	AIN3	Offset Voltage
00100	AIN4	AIN5	Voltage
00101	AIN5	AIN5	Offset Voltage
00110	AIN6	AIN7	Voltage
00111	AIN7	AIN7	Offset Voltage
01000	AIN8	AIN9	Voltage
01001	AIN9	AIN9	Offset Voltage
01010	AIN10	AIN11	Voltage
01011	AIN11	AIN11	Offset Voltage
01100	AIN12	AIN13	Voltage
01101	AIN13	AIN13	Offset Voltage
01110	AIN14	AIN15	Voltage
01111	AIN15	AIN15	Offset Voltage
1x00x	Internal	Internal	Internal Temperature
1x01x	AIN0	AIN1	Remote Temperature 1
1x10x	AIN2	AIN3	Remote Temperature 2
11110	—	—	Reserved
11111	—	—	Reserved

MAXQ7665/MAXQ7666 User's Guide

When ADCMX4 is cleared, the ADC input channel is configured for a differential voltage measurement. When ADCMX0 is set, the ADC's positive and negative inputs are internally connected to the same analog input pin so the user can measure zero offset error, if any. When ADCMX4 is set, the ADC input channel is configured to measure remote or internal temperature, and the bits ADCMX3:ADCMX0 control the temperature measurement.

The bits ADCMX1 and ADCMX2 determine if internal or external temperature sense mode is selected.

ADCMX2	ADCMX1	FUNCTION
0	0	Internal diode-connected transistor based temperature measurement
0	1	Remote diode-connected transistor based temperature measurement on AIN0
1	0	Remote diode-connected transistor based temperature measurement on AIN2
1	1	Reserved

The bits ADCMX0 and ADCMX3 determine the state of the temperature sensor when measuring temperature.

ADCMX0: TS Auto-Zero Control. This bit puts the temperature sensor in auto-zero state when it is set to logic 1. The auto-zero-ing is used to cancel internal offset effects.

ADCMX3: TS Current Control. This bit sets the temperature sensor current to its high value when set to logic 1, and sets the current to its low value when cleared to logic 0.

The ADC performs temperature measurement by measuring the voltage across a diode-connected transistor at two different current levels.

Note: The temperature measurement process is fully automated in the MAXQ7665/MAXQ7666 ROM utility routine "tempConv." All the required setup and temperature measurement algorithm steps for both internal and external temperature measurements are handled in the utility routine and it returns the local or remote temperature result.

Bit 10: Differential Input (ADCDIF). The ADC operates only on differential inputs and this bit must be set to logic 1 if the remote or internal temperature sensor drive circuit is not selected. For the remote temperature sensor, this bit determines if the input is single-ended or differential. When this bit is set to logic 1, the remote temperature sense-diode anode connects to the designated positive input, and cathode connects to the designated negative input. When this bit is cleared to 0, the sense-diode cathode connects to AGND. If using internal temperature sense mode, leave this bit as 0.

Bit 9: ADC Bipolar Mode Select (ADCBIP). When this bit is set to logic 1, the ADC is in bipolar mode. When this bit is cleared to 0, the ADC is in unipolar mode.

Bits 8, 7, 5: Reserved. Read returns 0, write ignored.

Bit 6: ADC Dual-Mode Select (ADCDUL). This bit determines the ADC's acquisition time. When ADCDUL is set to 1, the ADC operates in dual-edge mode. The rising edge of ADC_CNVST (internal signal formed by a combination of all three conversion start sources described below) causes the ADC to power up and begin acquisition; the falling edge causes it to sample and perform a conversion. When ADC-DUL is 0, the ADC operates in single-edge mode. The rising edge controls the entire conversion, i.e., power-up, acquisition, and conversion sequence if the ADC was off; if the ADC was on, it stays in acquisition mode until the rising edge and then starts conversion.

Note: Setting ADCDUL = 1 and PGA gain = 1 is illegal. If ADCDUL is set as 1, make sure the PGA gain (selected by the PGG2:PGG0 bits in the APE register) is greater than 1.

Bit 4: ADC Auto Shutdown (ADCASD). Setting this bit to logic 1 shuts down the ADC automatically after the conversion is completed. Clearing this bit to 0 disables the auto shutdown function, and leaves the ADC powered on.

Bit 3: ADC Start/Busy (ADCBY). Setting this bit to logic 1 enables the ADC to perform a conversion when ADCS2:ADCS0 is also set to 111. ADCBY remains set while the conversion is in progress. A read of this bit reflects the busy status of the ADC. ADCBY is cleared by hardware when the conversion is complete and the data is ready. Attempting to change ADCBY from 1 to 0 by software is blocked by hardware in order to allow the conversion to complete.

Note that if software-controlled conversions are implemented (by setting ADCS2:ADCS0 to 111) when ADCDUL is also set, then to complete a conversion the user must first write ADCBY to 1 and then write a second time to attempt to set it back to 0. Setting ADCBY to 1 puts the ADC into acquisition mode. The second write attempting to set ADCBY back to 0 moves the ADC from acquisition to the conversion phase. The second write will not affect the value of ADCBY until the ADC cycle has completed.

Bits 2 to 0: ADC Source Select Bits 2 to 0 (ADCS2 to ADCS0). These bits select the ADC conversion start source used to trigger analog-to-digital conversion:

ADCS2:ADCS0	CONVERSION START SOURCE
000	Timer 0.
001	Timer 1.
010	Timer 2.
011	Reserved, functions as 010 if set.
100	From ADC conversion start pin: P0.4/ADCCNV.
101	From ADC conversion start pin with inverted data.
110	Continuous conversion every 16 clocks.
111	From ADC start bit: ACNT.3.

In mode 110, the ADC completes a conversion every 16 clocks with a PGA gain of 1. For other gains the PGA is active and conversions complete every 56 clocks.

Note that the ADC conversion start source could be one of the timers, ADC conversion start pin, or software writes to ADC start bit. All three conversion start sources support single-edge or dual-edge modes of operation. Single- or dual-edge mode is controlled by ADC-DUL bit. Also, all three conversion start sources support auto-shutdown after a conversion. See the ADCASD control bit description.

Note: It is recommended that the ADCS bits are updated before triggering conversions so the ADC conversion start source selection fully takes effect. As an example, the ADCBY bit should not be set in the same ACNT register write which changes the ADCS bits to 111. The same recommendation also applies to other conversion start sources. When the ADCS bits are being updated, avoid generating an ADC conversion trigger from the timers or the ADC conversion start pin.

3.2.3 DAC Control Register (DCNT)

Register Description: **DAC Control Register**
Register Name: **DCNT**
Register Address: **Module 05h, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	DACLD2	DACLD1	DACLD0	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	r	r	r	r

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 7, 3 to 0: Reserved. Read returns 0, write ignored.

Bits 6 to 4: DAC Load Select Bits 2 to 0 (DACLD2 to DACLD0). These bits determine the mechanism of data transfer for DAC conversion by generating the DACLOAD signal:

DACLD2:DACLD0	DATA TRANSFER
000	DAC conversion data is sourced from the DACO register under the control of the external DACLOAD input signal. On the rising edge of the DACLOAD input, the DACO register is loaded with the contents of DACI register and converted. The DAC output signal (DACOUT) then immediately tracks the DACO value. Note that this selection enables DACLOAD alternate function on the shared DACLOAD/P0.5 pin.
001	DAC conversion data is sourced from the DACO register under the control of software write to DACI register. The DACO register is loaded with the contents of DACI when the DACI register is written. The DAC output signal (DACOUT) then immediately tracks the DACO value.
010	Reserved.
011	Reserved, functions as 001 if set.
100	DAC conversion data is sourced from the DACO register under the control of the external DACLOAD input signal. On the falling edge of the DACLOAD input, the DACO register is loaded with the contents of DACI register and converted. The DAC output signal (DACOUT) then immediately tracks the DACO value. Note that this selection enables DACLOAD alternate function on the shared DACLOAD/P0.5 pin.
101	Square-wave mode. The data source for the DAC depends upon edges supplied by the DACLOAD pin. 1) A falling edge on DACLOAD after entering square-wave mode supplies the data in DACI to the DAC. 2) A rising edge on DACLOAD after entering square-wave mode supplies the data in DACO to the DAC. The DAC output signal (DACOUT) tracks the DACI value on a falling edge and the DACO value on a rising edge. Note that as the DAC settling time is up to 15µs, toggling DACLOAD at rates substantially faster than that may not allow the DAC to settle at either of the intended output values.
110	Reserved.
111	Reserved.

3.2.4 DAC Input Data Register (DACI)

Register Description: **DAC Input Data Register**
Register Name: **DACI**
Register Address: **Module 05h, Index 04h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	DACI.11	DACI.10	DACI.9	DACI.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	DACI.7	DACI.6	DACI.5	DACI.4	DACI.3	DACI.2	DACI.1	DACI.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 12: Reserved. Read returns 0, write ignored.

Bits 11 to 0: DAC Input Data 11 to 0 (DACI.11 to DACI.0). This register holds input data for DAC conversion.

3.2.5 DAC Output Data Register (DACO)

Register Description: **DAC Output Data Register**
Register Name: **DACO**
Register Address: **Module 05h, Index 06h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	DACO.11	DACO.10	DACO.9	DACO.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	DACO.7	DACO.6	DACO.5	DACO.4	DACO.3	DACO.2	DACO.1	DACO.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 12: Reserved. Read returns 0, write ignored.

Bits 11 to 0: DAC Output Data 11 to 0 (DACO.11 to DACO.0). This register holds output data for DAC conversion. Data from the DAC input register is transferred to this output register for DAC conversion as controlled by the DACLD2:DACLD0 bits.

MAXQ7665/MAXQ7666 User's Guide

3.2.6 ADC Data Register (ADCD)

Register Description: **ADC Data Register**
Register Name: **ADCD**
Register Address: **Module 05h, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	ADCD.11	ADCD.10	ADCD.9	ADCD.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ADCD.7	ADCD.6	ADCD.5	ADCD.4	ADCD.3	ADCD.2	ADCD.1	ADCD.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 12: Reserved. Read returns 0, write ignored.

Bits 11 to 0: ADC Data 11 to 0 (ADCD.11 to ADCD.0). This register holds output data from ADC conversion. Data from the ADC is latched in to this register at the falling edge of the ADCBY signal.

3.2.7 Temperature Sense Offset Register (TSO)

Register Description: **Temperature Sense Offset Register**
Register Name: **TSO**
Register Address: **Module 05h, Index 09h**

Bit #	15	14	13	12	11	10	9	8
Name	TSO.15	TSO.14	TSO.13	TSO.12	TSO.11	TSO.10	TSO.9	TSO.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	TSO.7	TSO.6	TSO.5	TSO.4	TSO.3	TSO.2	TSO.1	TSO.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Temperature Sense Offset Data 15 to 0 (TSO.15 to TSO.0). This 16-bit read-only register holds sign-extended two's-complement offset data for temperature sensing conversion. The effective bit length of this register is 12 bits. The upper four bits are sign extended. The offset stored in this register is added to the final temperature result to convert the raw temp sensor output (in degrees Kelvin) to degrees Celsius. The temperature conversion utility ROM routine automatically adds the offset stored in this register to the final temperature result.

3.2.8 Analog Interrupt Enable Register (AIE)

Register Description: **Analog Interrupt Enable Register**
Register Name: **AIE**
Register Address: **Module 05h, Index 0Ah**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFIE	VIOBIE	DVBIE	—	AORIE	ADCIE	—
Reset	0	0	0	0	0	0	0	1
Access	r	rw	rw	rw	r	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0001h on all forms of reset.*

Bits 15 to 7 and 3: Reserved. Read returns 0, write ignored.

Bit 6: High-Frequency Oscillator Failure Interrupt Enable (HFFIE). See *Section 5* for details on this bit.

Bit 5: I/O Voltage Brownout Interrupt Enable (VIOBIE). See *Section 2* for details on this bit.

Bit 4: Digital Brownout Interrupt Enable (DVBIE). See *Section 2* for details on this bit.

Bit 2: ADC Overrun Interrupt Enable (AORIE). This bit must be set to logic 1 to generate an interrupt request when an ADC result overrun occurs and the ADCOV flag is set to logic 1. Clearing this bit to 0 disables the interrupt capability from ADCOV. Note: To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and IM5 mask in the IMR peripheral register.

Bit 1: ADC Data Ready Interrupt Enable (ADCIE). This bit must be set to logic 1 to generate an interrupt request when the ADC completes a conversion and the ADCRY flag is set to logic 1. Clearing this bit to 0 disables the interrupt capability from ADCRY. Note: To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by the IGE bit in the IC register and IM5 mask in the IMR peripheral register.

Bit 0: This bit is implemented and available to be used as a user-software-controlled bit.

3.2.9 Analog Status Register (ASR)

Register Description: **Analog Status Register**
Register Name: **ASR**
Register Address: **Module 05h, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	VIOLVL	DVLVL	—	—	XHFRY	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFINT	VIOBI	DVBI	—	ADCOV	ADCRY	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *The ADCOV bit is cleared by all forms of reset. All other bits are only reset by POR. Reading the ASR register resets to 0 all the status flag bits except VIOLVL and DVLVL.*

Bit 15: I/O Voltage Brownout Comparator Level (VIOLVL). See *Section 2* for details on this bit.

Bit 14: Digital Voltage Brownout Comparator Level (DVLVL). See *Section 2* for details on this bit.

Bits 13, 12, 10 to 7, 3, and 0: Reserved. Read returns 0, write ignored.

Bit 11: High-Frequency Oscillator Ready (XHFRY). See *Section 5* for details on this bit.

Bit 6: External High-Frequency Oscillator Failure Flag (HFFINT). See *Section 5* for details on this bit.

Bit 5: I/O Voltage Brownout Flag (VIOBI). See *Section 2* for details on this bit.

Bit 4: Digital Voltage Brownout Flag (DVBI). See *Section 2* for details on this bit.

Bit 2: ADC Overrun Flag (ADCOV). This flag signifies that an ADC result overrun has occurred when it is set to logic 1. This bit is cleared after reading from the ASR register. ADC overrun occurs if prior ADC data gets overwritten before it is read (i.e., ADCRY = 1 at falling edge of ADCBY). If enabled, the ADC overrun interrupt is generated by this register bit.

Bit 1: ADC Data Ready Flag (ADCRY). This flag is set to logic 1 when the ADC completes its conversion and data is ready for access. This bit is cleared after reading data from the ADCD register. If enabled, the ADC data ready interrupt is generated by this register bit.

3.2.10 Oscillator Control Register (OSCC)

Register Description: **Oscillator Control Register**
Register Name: **OSCC**
Register Address: **Module 05h, Index 0Ch**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	HFOC1	HFOC0	HFIC1	HFIC0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	ADCCD2	ADCCD1	ADCCD0	—	—	EXTHF	RCE	HFE
Reset	0	0	0	0	0	0	1	0
Access	rw	rw	rw	r	r	rw	rw	rw

r = read, w = write (Note that some clock control bits may have locking mechanisms for write.)
Note: *This register is cleared to 0002h on power-on reset and is not affected by other forms of reset.*

Bits 15 to 12, 4, and 3: Reserved. Read returns 0, write ignored.

Bits 11 and 10: High-Frequency Crystal Output Capacitance Select 1 and 0 (HFOC1 and HFOC0). See *Section 5* for details on these bits.

Bits 9 and 8: High-Frequency Crystal Input Capacitance Select 1 and 0 (HFIC1 and HFIC0). See *Section 5* for details on these bits.

Bits 7 to 5: ADC Clock Divider Bits 2 to 0 (ADCCD2 to ADCCD0). These bits determine the ADC clock frequency that is divided down from the system clock. The MAXQ7665/MAXQ7666 ADC uses the divided-system clock to clock the multiplexer front-end selection, track and hold acquisition, and each step of the successive approximation conversion. Note that there is no ADC clock in stop mode.

ADCCD2:ADCCD0	DIVIDE RATIO	ADC CLOCK
000	1 (default)	ADC Clock = System Clock
001	2	ADC Clock = System Clock/2
010	4	ADC Clock = System Clock/4
011	8	ADC Clock = System Clock/8
100	16	ADC Clock = System Clock/16
101–111	Reserved, divide by 16 if set	ADC Clock = System Clock/16

Bit 2: External High-Frequency Clock Enable (EXTHF). See *Section 5* for details on this bit.

Bit 1: Internal RC Oscillator Enable (RCE). See *Section 5* for details on this bit.

Bit 0: High-Frequency Crystal Oscillator Enable (HFE). See *Section 5* for details on this bit.

MAXQ7665/MAXQ7666 User's Guide

3.3 Analog-to-Digital Converter (ADC) Port

The MAXQ7665/MAXQ7666 contain a low-power, high-precision, 12-bit, 500ksps successive approximation analog-to-digital converter (ADC) and 8 differential-input channel multiplexer. The ADC can be configured to run from a variety of conversion start sources both internal and external to the microcontroller. The ADC conversion rate is reduced to 142ksps when PGA gains greater than 1 are used to allow for adequate analog signal settling. PGA functional details are covered in *Section 3.3.6*. The on-chip temperature sensor and the remote temperature sense drive makes use of the ADC for temperature measurements (see *Section 3.4* for details).

The ADC features the following:

- 12-bit SAR converter
- 8 differential analog-input channels
- Integrated track-and-hold (T/H) input circuit
- Integrated PGA on the input path with 1x, 2x, 4x, 8x, 16x, and 32x gains
- 500ksps sampling rate for PGA gain = 1
- 142ksps sampling rate for PGA gain > 1
- Bipolar/unipolar selection
- Selectable ADC conversion start source from on-chip timers, ADC conversion pin, and software write
- Acquisition time control
- Auto shutdown on conversion
- End of conversion and overrun status interrupts

Figure 3-2 shows a simplified functional block diagram of the ADC in differential input configuration.
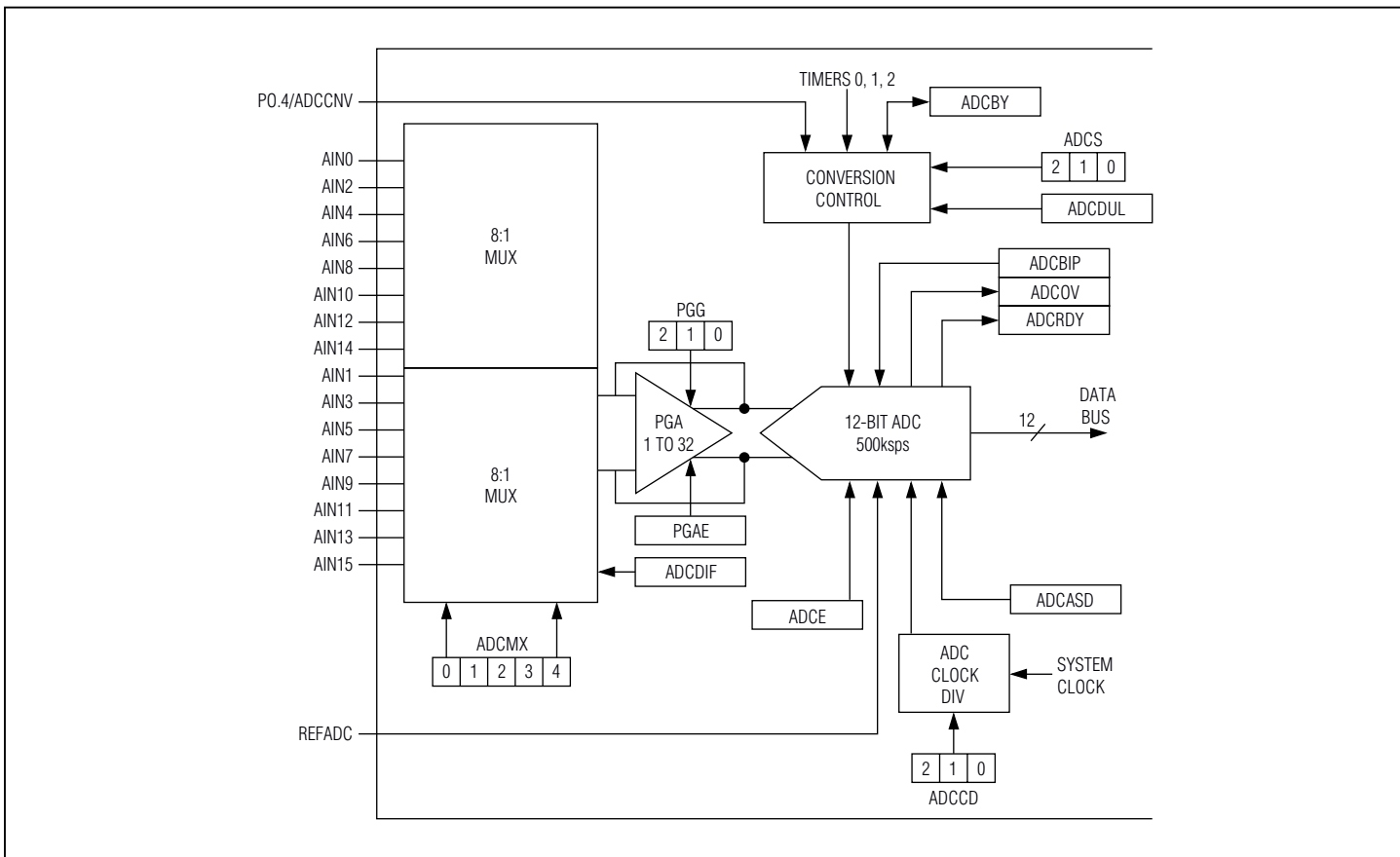
Figure 3-2. Differential Input ADC Block Diagram

3.3.1 ADC Signals

The MAXQ7665/MAXQ7666 ADC uses 18 external signals (other than analog supply and ground) as explained in Table 3-2.

Table 3-2. ADC Signals

SIGNAL	PIN NUMBER		FUNCTION
	48-PIN	**56-PIN**	
AIN15	45	53	
AIN14	46	54	
AIN13	47	55	
AIN12	48	56	
AIN11	1	1	
AIN10	2	2	
AIN9	3	3	ADC Analog Input 15 to 0. These are dedicated analog input pins connected through the internal analog multiplexer to the PGA and ADC. The analog multiplexer supports 8 differential-input measurements. In differential input mode, the inputs are paired: AIN14 to AIN15, AIN12 to AIN13, AIN10 to AIN11, AIN8 to AIN9, AIN6 to AIN7, AIN4 to AIN5, AIN2 to AIN3, AIN0 to AIN1.
AIN8	4	4	
AIN7	9	11	
AIN6	10	12	(See *Section 3.4: Temperature Sensor* for discussion on using analog inputs AIN2 and AIN0 for measuring remote temperature.)
AIN5	11	13	
AIN4	12	14	
AIN3	13	15	
AIN2	14	16	
AIN1	15	17	
AIN0	16	18	
P0.4/ADCCNV	36	41	Port 0 Data Bit 4/ADC Conversion Start. The ADC conversion start is a shared pin with the digital I/O port 0 bit 4. As ADCCNV, this pin can trigger ADC sampling and conversion on a rising or falling edge. After power-up or a reset this pin defaults to a digital I/O port pin with pullup enabled.
REFADC	6	6	ADC Reference Input. The REFADC input pin is used to supply an external precision voltage reference to the ADC. The REFADC can handle a voltage range from 1V to AVDD. The REFADC input determines the full-scale range (FSR) of the internal 12-bit ADC.
AVDD	44	52	Analog V_{DD} Supply. For the MAXQ7665/MAXQ7666, the analog supply voltage is +5.0V.
AGND	5, 8	5, 8	Analog Ground

3.3.2 Differential Inputs

The MAXQ7665/MAXQ7666 ADC use a fully differential successive-approximation register (SAR) conversion technique and an on-chip T/H block to convert temperature and voltage signals into a 12-bit digital result. Differential configurations are supported using an analog input channel multiplexer that supports 8 differential channels.

In differential input configuration, analog inputs AIN+ and AIN- are selected from the following pairs: AIN0/AIN1, AIN2/AIN3, AIN4/AIN5, AIN6/AIN7, AIN8/AIN9, AIN10/AIN11, AIN12/AIN13, and AIN14/AIN15. The differential input configuration references all input signals to the complementary multiplexer channel input, minimizing common-mode DC offsets and noise. Figure 3-3 shows the multiplexer connection scheme. The analog input is configured for differential conversion by writing logic 1 to the ADCDIF control bit, while analog input channel selection is controlled by the ADCMX control field in the ACNT peripheral register.

The remote temperature sensor configuration in differential mode uses analog input channel pairs AIN2/AIN3 and/or AIN0/AIN1. In single-ended remote temperature sensor configuration, only channels AIN2 and AIN0 are used. Internal temperature sensor configuration measures local die temperature and does not use any analog input channel.
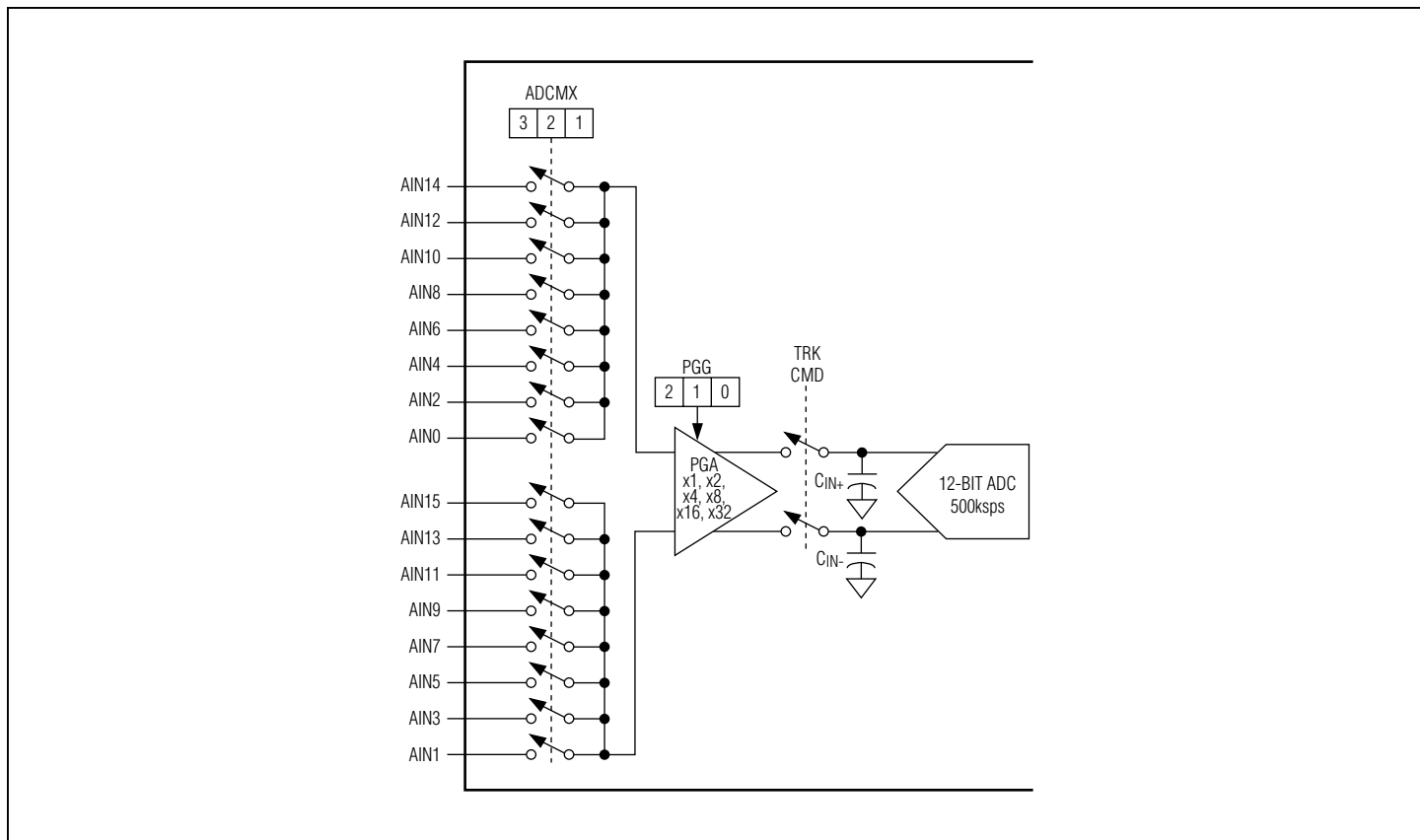
Figure 3-3. Multiplexer Input Connection Scheme

3.3.3 True-Differential Analog Input T/H

The equivalent input circuit of Figure 3-4 A and B shows the MAXQ7665/MAXQ7666's analog input architecture when the PGA is bypassed (PGA disabled and PGA gain = 1). In track mode, a positive input capacitor is connected to AIN0, AIN2, AIN4…AIN14 and a negative input capacitor is connected to AIN1, AIN3, AIN5…AIN15 in differential mode. T/H timing is controlled by the ADC source select (ADCS2:ADCS0) and ADC dual-mode select (ADCUL) fields in the ADC control register (ACNT). ADCS selects an ADC conversion start source, which could be one of the timers, ADCCNV pin, or software writes to the ADC start bit. All three conversion start sources support single-edge or dual-edge modes of operation, which are determined by the ADCDUL bit. When ADCDUL is set to 1, the ADC operates in dual-edge mode. The rising edge of the selected conversion start source causes the ADC to power up and begin acquisition; the falling edge causes it to sample and perform a conversion. If ADCDUL is set as 1, make sure the PGA gain (selected by the PGG2:PGG0 bits in the APE register) is greater than 1. Setting ADCDUL = 1 and PGA gain = 1 is illegal. When ADCDUL is 0, the ADC operates in single-edge mode. The rising edge controls the entire conversion, i.e., power-up, acquisition, and conversion sequence if the ADC was off; if the ADC was on, it stays in acquisition mode until the rising edge and then starts conversion. Once a conversion has been initiated, the T/H enters acquisition mode for the next conversion on the 13th falling edge of ADCCLK, if auto shut-down (ADCASD = 0 in ADC control register) is disabled. See *Section 3.3.10* for ADC conversion start sources and timing details.

The time required for the T/H to acquire an input signal is determined by how quickly its input capacitance is charged. If the input signal's source impedance is high, the acquisition time lengthens. The acquisition time, t_{ACQ}, is the minimum time needed for the signal to be acquired. It is calculated by the following equation:

$$t_{ACQ} \geq k \times (R_{SOURCE} + R_{IN}) \times C_{IN}$$

Where:

$$k = 9 \approx \ln(2 \times 2^{12})$$

The constant, k, is the number of RC time constants required so that the voltage on the internal sampling capacitor reaches 12-bit accuracy, i.e., so that the difference between the input voltage and the sampling capacitor voltage is equal to 0.5 LSB.

R_{SOURCE} is the source impedance of the input signal, $R_{IN} = 1k\Omega$ is the equivalent differential analog input resistance, and $C_{IN} = 14pF$ is the equivalent differential analog input capacitance when PGA = 1. Note that for PGA = 1, t_{ACQ} is never less than 375ns (3 ADCCLK periods at 8MHz), and any source impedance less than $1k\Omega$ does not significantly affect the ADC's AC performance. For higher source impedance, a longer acquisition time is required.

For PGA > 1, t_{ACQ} requires an additional 5µs (40 ADCCLK cycles at 8MHz). The additional cycles are due to PGA settling time and automatically introduced by the internal hardware. The PGA uses a switched capacitor technique (see *Section 3.3.6*), and channel input capacitance C_{IN} increases with gain as shown in Table 3-3.

With PGA > 1, any source impedance less than $5k\Omega$ does not significantly affect the ADC's AC performance.

Table 3-3. PGA Gain and Channel Input Capacitance

PGA	C_{IN} (pF) ($C_{IN} = C_{IN+} = C_{IN-}$)
x2	2
x4	4
x8	8
x16	16
x32	32
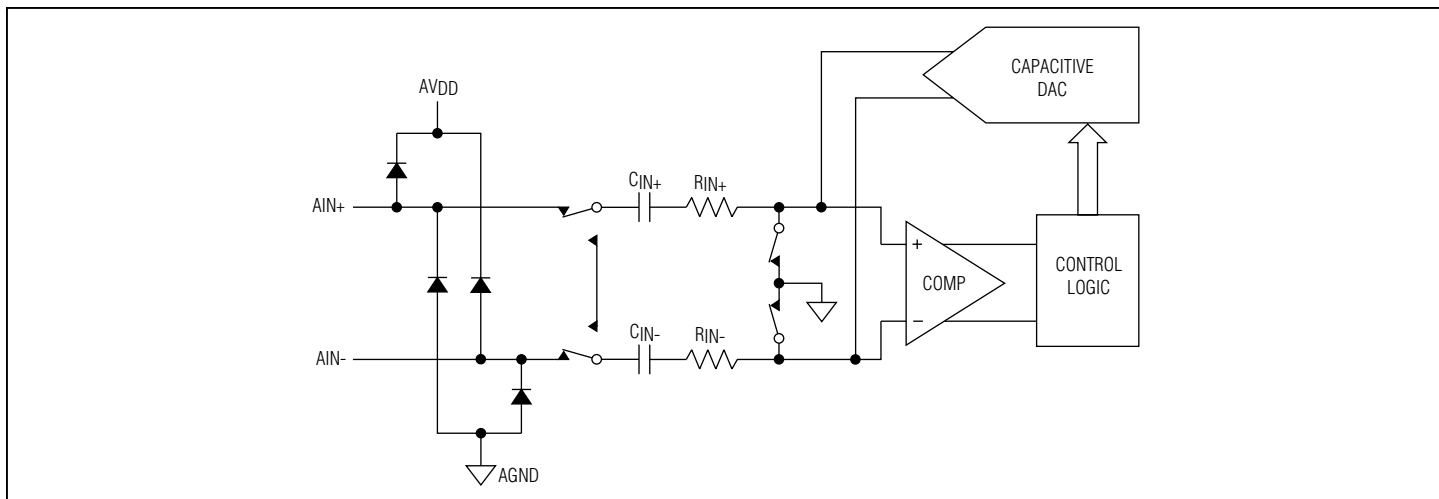
Figure 3-4A. Equivalent Input Circuit (Acquisition Mode with PGA Bypassed)
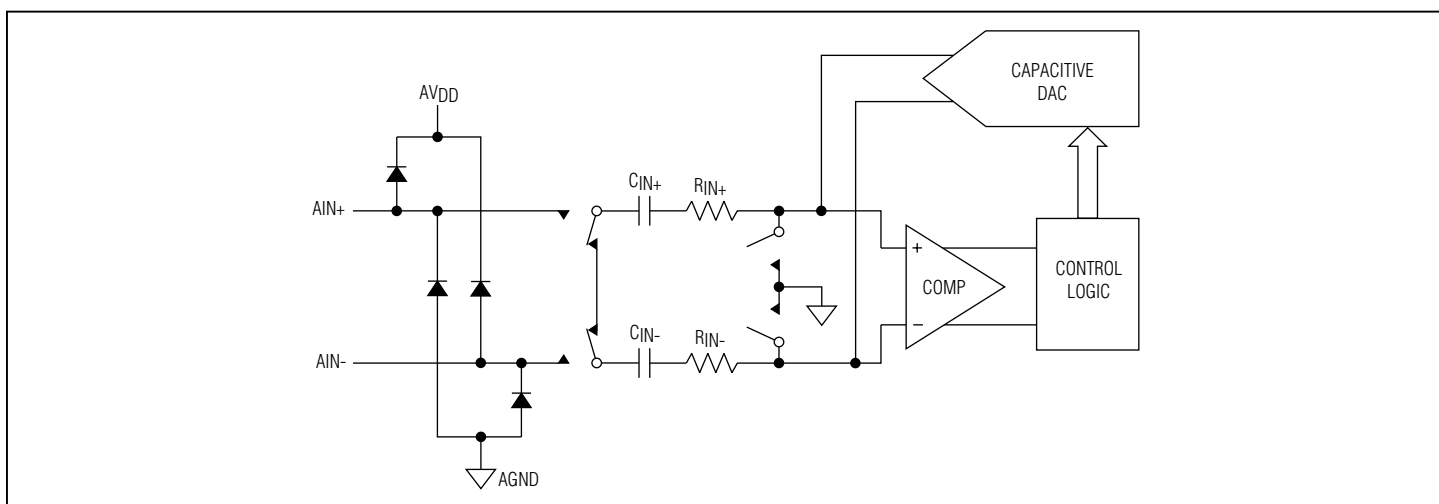
Figure 3-4B. Equivalent Input Circuit (Hold/Conversion Mode with PGA Bypassed)

3.3.4 Unipolar/Bipolar

The MAXQ7665/MAXQ7666 ADC produces a digital output that corresponds to the differential analog input voltage as long as the differential analog inputs are within the specified range. The analog inputs are configured for differential conversion when the ADCDIF control bit is set. When performing differential conversions, the control bit ADCBIP in analog control register selects between unipolar and bipolar operation modes. Unipolar mode sets the differential input range from 0 to REFADC (for PGA gain of 1, it is less if the gain is > 1). A negative differential analog input in unipolar mode causes the digital output code to be 0. Selecting bipolar mode sets the differential input range to -REFADC/2 to +REFADC/2 (for PGA gain of 1, it is less if the gain is > 1). The digital output code is straight binary in unipolar mode and two's complement in bipolar mode (see *Section 3.3.5: Transfer Function*).

3.3.5 Transfer Function

The MAXQ7665/MAXQ7666 ADC output is straight binary in unipolar mode. Figure 3-5 shows the MAXQ7665/MAXQ7666 ADC unipolar transfer function for PGA gain of 1. Table 3-4 shows the unipolar relationship between the differential analog input voltage and the digital output code for PGA gain of 1.
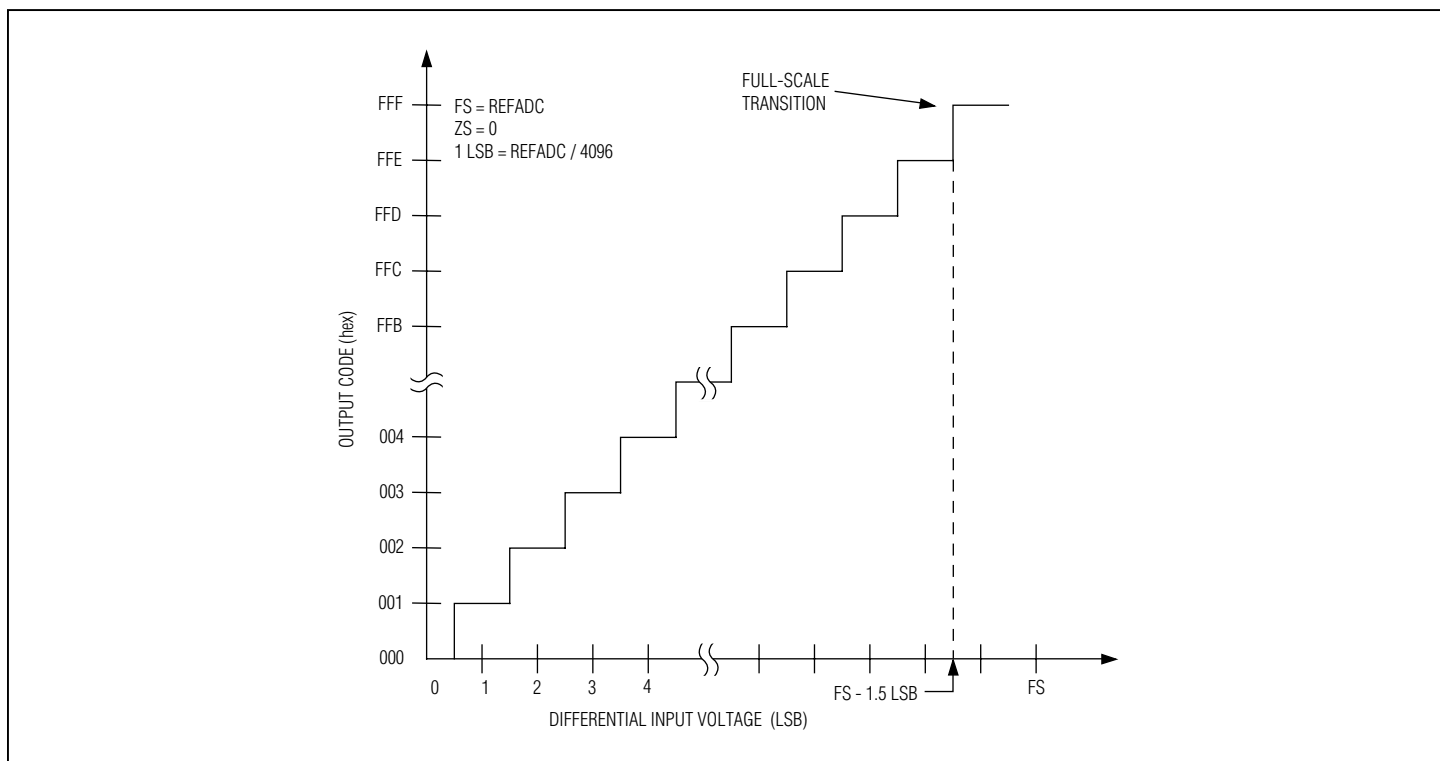
Figure 3-5. Unipolar Transfer Function (PGA Gain = 1)

Table 3-4. Unipolar Code Table (PGA Gain = 1)

BINARY DIGITAL OUTPUT CODE ADCD11:ADCD0	HEXADECIMAL EQUIVALENT OF ADCD11:ADCD0	DECIMAL EQUIVALENT OF ADCD11:ADCD0 (CODE$_{12}$)	IDEAL DIFFERENTIAL INPUT VOLTAGE (V) (REFADC = 5.0V)
1111 1111 1111	0xFFF	4095	+4.99878 ± 0.5 LSB
1111 1111 1110	0xFFE	4094	+4.99756 ± 0.5 LSB
1000 0000 0001	0x801	2049	+2.50122 ± 0.5 LSB
1000 0000 0000	0x800	2048	+2.5 ± 0.5 LSB
0111 1111 1111	0x7FF	2047	+2.49878 ± 0.5 LSB
0000 0000 0001	0x001	1	+0.00122 ± 0.5 LSB
0000 0000 0000	0x000	0	+0.000 + 0.5 LSB

Table 3-5 shows the input range for various PGA settings (PGG2:PGG0) in unipolar mode. When the PGA is used (gain > 1), the differential input range at the analog multiplexer input is reduced by the gain factor. The maximum PGA output value is limited to 3.2V. For gain > 1, differential input range x PGA gain ≤ 3.2V and the ADC output code range is limited to 0–2621 (decimal).

Table 3-5. Unipolar Input Scaling

PGG2	PGG1	PGG0	GAIN FACTOR	DIFFERENTIAL INPUT RANGE AT MUX INPUTS	DIFFERENTIAL INPUT RANGE WITH 5.0V EXTERNAL REFERENCE	LSB SIZE WITH 5.0V EXTERNAL REFERENCE	OUTPUT CODE RANGE (DECIMAL)
0	0	0	x1	0 to +REFADC	0 to 4.99878V	1.22mV	0–4095
0	0	1	x2	0 to +REFADC/2 (1.6V max)	0 to 1.6V	610.35µV	0–2621 (REFADC = +5V)
0	1	0	x4	0 to +REFADC/4 (0.8V max)	0 to 0.8V	305.18µV	
0	1	1	x8	0 to +REFADC/8 (0.4V max)	0 to 0.4V	152.59µV	
1	0	0	x16	0 to +REFADC/16 (0.2V max)	0 to 0.2V	76.29µV	
1	0	1	x32	0 to +REFADC/32 (0.1V max)	0 to 0.1V	38.15µV	

The MAXQ7665/MAXQ7666 ADC output is two's complement in bipolar mode. Figure 3-6 shows the MAXQ7665/MAXQ7666 ADC bipolar transfer function for PGA gain = 1. Table 3-6 shows the bipolar relationship between the differential analog input voltage and the digital output code for PGA gain = 1. In bipolar mode, the inputs are measured in a truly differential fashion where either input can exceed the other by up to REFADC/2 (for gain = 1, but limited for gain > 1).
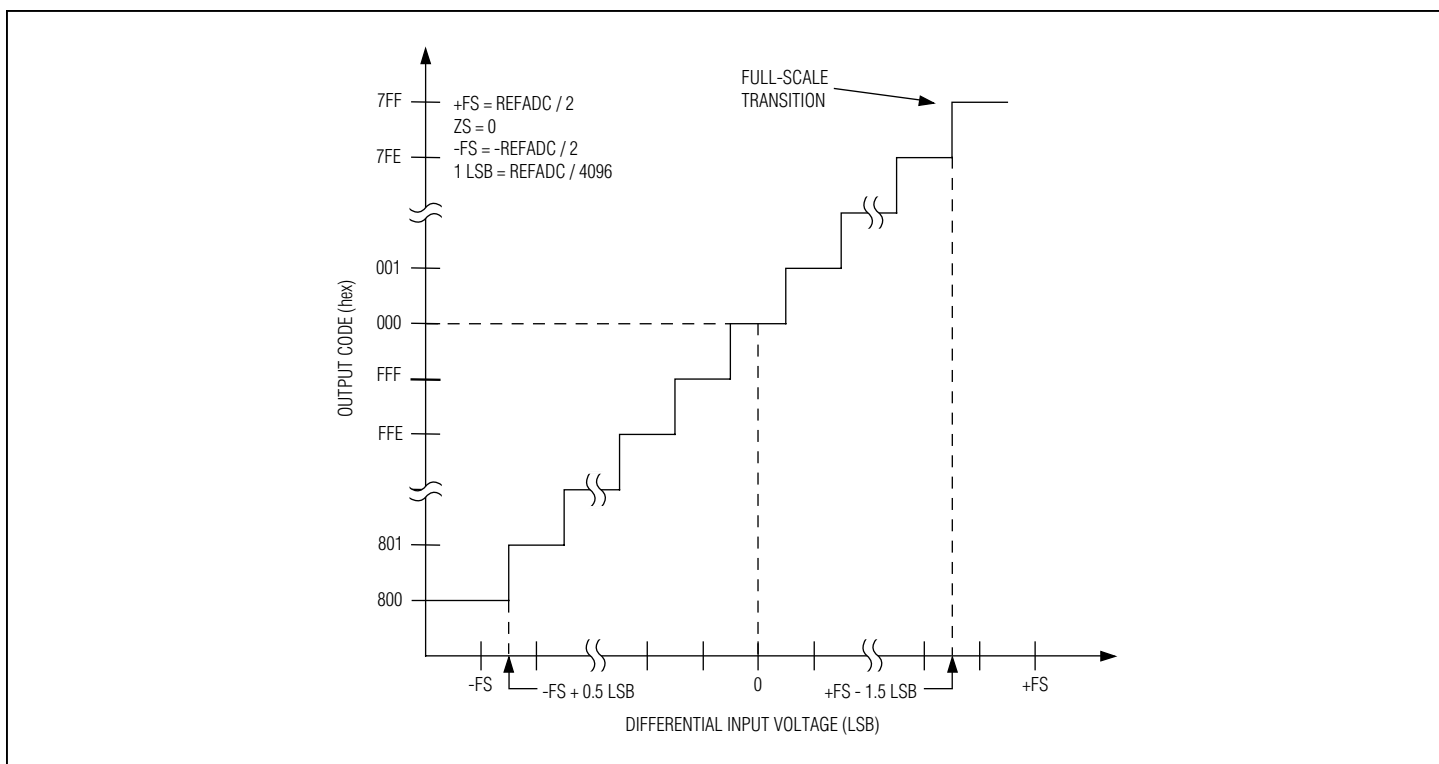
Figure 3-6. Bipolar Transfer Function (PGA Gain = 1)

Table 3-6. Bipolar Code Table (PGA Gain = 1)

BINARY DIGITAL OUTPUT CODE ADCD11:ADCD0	HEXADECIMAL EQUIVALENT OF ADCD11:ADCD0	DECIMAL EQUIVALENT OF ADCD11:ADCD0 (CODE$_{12}$)	IDEAL DIFFERENTIAL INPUT VOLTAGE (V) (REFADC = 5.0V)
0111 1111 1111	0x7FF	+2047	+2.49878 ± 0.5 LSB
0111 1111 1110	0x7FE	+2046	+2.49756 ± 0.5 LSB
0000 0000 0001	0x001	+1	+0.00122 ± 0.5 LSB
0000 0000 0000	0x000	0	0.000 ± 0.5 LSB
1111 1111 1111	0xFFF	-1	-0.00122 ± 0.5 LSB
1000 0000 0001	0x801	-2047	-2.49878 ± 0.5 LSB
1000 0000 0000	0x800	-2048	-2.5 ± 0.5 LSB

Table 3-7 shows the input range for various PGA settings (PGG2:PGG0) in bipolar mode. When the PGA is used (gain > 1), the differential input range at the analog multiplexer input is reduced by the gain factor:

full-scale differential input range = ±REFADC / (2 x PGA gain factor)

Table 3-7. Bipolar Input Scaling

PGG2	PGG1	PGG0	GAIN FACTOR	DIFFERENTIAL INPUT RANGE AT MUX INPUTS	DIFFERENTIAL INPUT RANGE WITH 5.0V EXTERNAL REFERENCE	LSB SIZE WITH 5.0V EXTERNAL REFERENCE	OUTPUT CODE RANGE (DECIMAL)
0	0	0	x1	±REFADC/2	-2.5V to 2.5V	1.22mV	
0	0	1	x2	±REFADC/4	-1.25V to 1.25V	610.35µV	
0	1	0	x4	±REFADC/8	-0.625V to 0.625V	305.18µV	-2048 to +2047
0	1	1	x8	±REFADC/16	-0.313V to 0.313V	152.59µV	
1	0	0	x16	±REFADC/32	-0.156V to 0.156V	76.29µV	
1	0	1	x32	±REFADC/64	-0.078V to 0.078V	38.15µV	

The differential analog input voltage as a function of REFADC, PGA gain factor (PGA_GF), and the ideal (error-free) digital output code are determined with the following equation:

$$\Delta V_{AIN} = LSB \times CODE_{12} \pm 0.5 \times LSB$$

Where:

$\Delta V_{AIN} = V_{AIN+} - V_{AIN-}$

$LSB = REFADC / (PGA\_GF \times 2^{12}) = REFADC / (PGA\_GF \times 4096)$

$CODE_{12}$ = the decimal equivalent of the digital output code (see Table 3-4 and Table 3-6)

PGA_GF = PGA gain factor (1, 2, 4, 8, 16, or 32)

±0.5 x LSB represents the quantization error that is inherent to any ADC

When using a 4.096V reference and a PGA gain factor of 1, 1 LSB equals 1.0mV. When using a 5.0V reference and a PGA gain factor of 1, 1 LSB equals 1.22mV.

3.3.6 Programmable Gain Amplifier

The MAXQ7665/MAXQ7666 programmable gain amplifier (PGA) receives its inputs from the input multiplexer and feeds its outputs to the 12-bit ADC. Figure 3-7 shows the MAXQ7665/MAXQ7666 PGA block diagram. The PGA has software-selectable gains of x1, x2, x4, x8, x16, and x32. The PGA uses a switched capacitor technique that reduces power and improves linearity and accuracy. The PGA has a 5µs warmup/turn-on time from the PGA enable (bit 3 in APE register). The analog front-end should be configured before attempting any conversions to ensure that the PGA and ADC are enabled and fully functional. This is most important after a power-down or reset condition.

The MAXQ7665/MAXQ7666 PGA is bypassed when set to gain = 1 (PGG2:PGG0 = 000). If the PGAE bit in the APE register is cleared (PGA disabled), power consumption is also reduced significantly. To bypass the PGA, disabling the PGA alone is not enough; the gain must be set to 1. This allows for full-speed analog signal acquisition and conversion. The analog front-end can acquire and convert an analog signal in 2µs (500ksps) when a gain of x1 is selected. For higher gains (≥ x2) the PGA settling time requires an additional 40 cycles (5µs at 8MHz clock) to settle to a final value. So the maximum throughput rate for gains greater than or equal to x2 is 142ksps. Longer settling time through the analog front-end can be experienced when the device that is being measured has a source imped-ance of ≥ 5kΩ. In that case, the ADC conversion clock should be programmed to allow for additional settling time caused by high source impedance. To properly budget the appropriate amount of time for the signal to settle, see *Section 3.3.10*. **Note:** When the PGA is bypassed (PGA gain = 1 and PGAE = 0), make sure the ADC single-edge mode (ADCDUL = 0) is selected. In single-edge mode, the ADC control logic provides the necessary power-up, acquisition, and conversion delay to ensure maximum throughput. When using the ADC dual-edge mode (ADCDUL = 1), power-up and acquisition period is under user control. It is valid to use this mode only for PGA gain > 1.

See *Section 3.3.3* for discussion of the effect of the PGA on the ADC equivalent input circuit and *Section 3.3.5* for the effect of the PGA on the ADC transfer function. *Section 3.3.10* discusses the effect of the PGA on the ADC conversion timing.
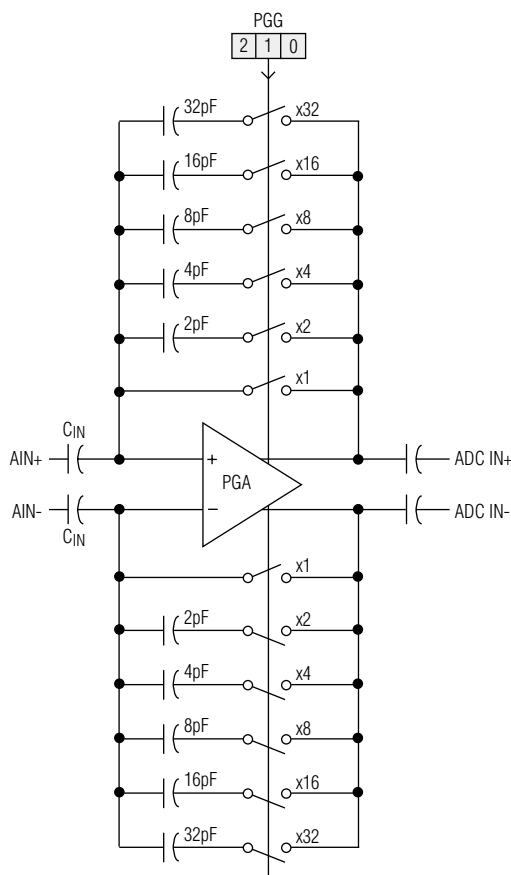
Figure 3-7. PGA Block Diagram

3.3.7 Analog Input Protection

Internal ESD protection diodes limit all analog inputs to AVDD and AGND, allowing the inputs to swing from (AGND - 0.3V) to (AVDD + 0.3V) without damage. However, for accurate conversions near full scale, the inputs should not exceed AVDD by more than +50mV or be lower than AGND by -50mV. Input voltages beyond AGND - 0.3V and AVDD + 0.3V forward bias the internal protection diodes. In this situation, limit the forward diode current to 50mA to avoid damaging the MAXQ7665/MAXQ7666.

The MAXQ7665/MAXQ7666's common-mode analog input range or absolute analog input range is specified from AGND to AVDD. Signals may run outside that range but will be interpreted as an overrange (ADC data output set to 0xFFF in unipolar mode and to 0x7FF in bipolar mode) or an underrange condition (ADC data output set to 0x000 in unipolar mode and to 0x800 in bipolar mode). Analog input signals cannot excurse outside of the ABS max input signal ratings of +0.3V above AVDD or -0.3V below AGND.

Figure 3-8 shows the common mode or absolute analog input range as specified with a selected REFADC value. In unipolar input configuration (ADCBIP = 0, register bit location ACNT.9), the output data coding of the ADC is straight binary.

In bipolar input configuration (ADCBIP = 1, register bit location ACNT.9), the output data coding of the ADC is two's complement. Bipolar mode is commonly used with a differential analog input configuration (ADCDIF = 1, register bit location ACNT.10), where the analog input signals are referenced to their complementary analog input (AIN+/-) pin. Figure 3-9 shows how a negative ADC value is created. The absolute voltage of each analog input signal must be within the MAXQ7665/MAXQ7666 supply range so as to satisfy the critical absolute maximum tolerance ratings of the analog inputs, and must also be within the REFADC range to produce useful information from the ADC.
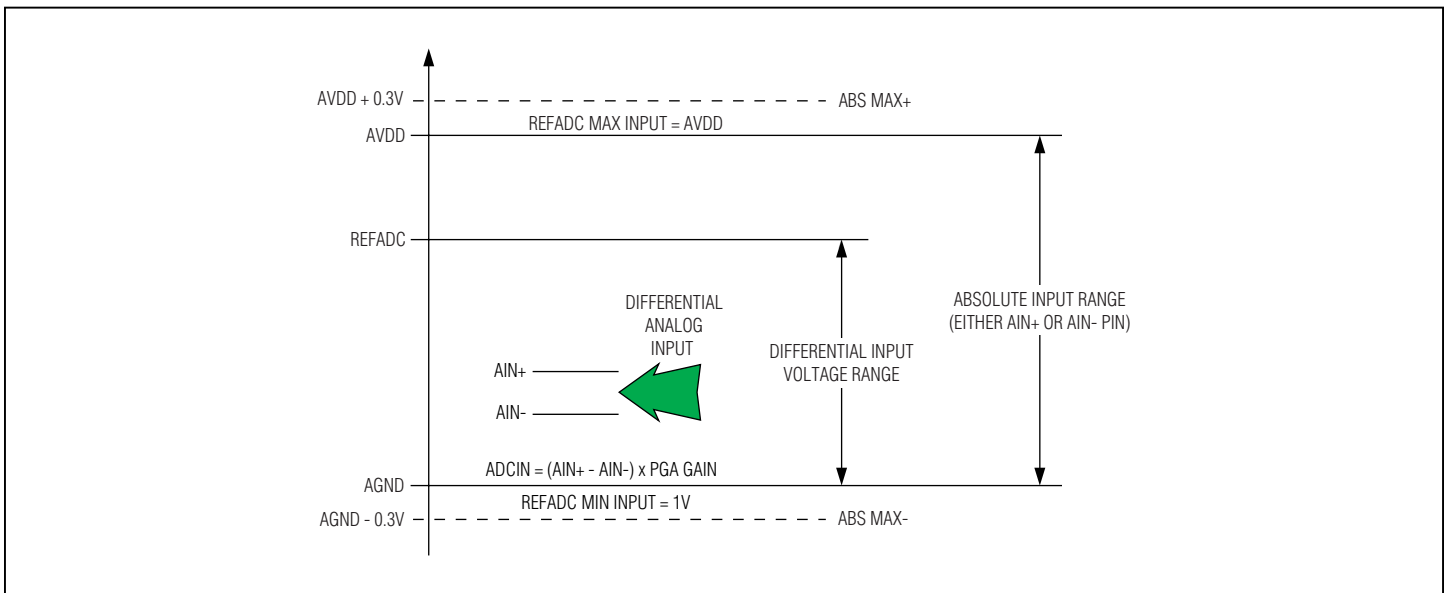
Figure 3-8. Analog Input Range Measuring a Positive Analog Input Value
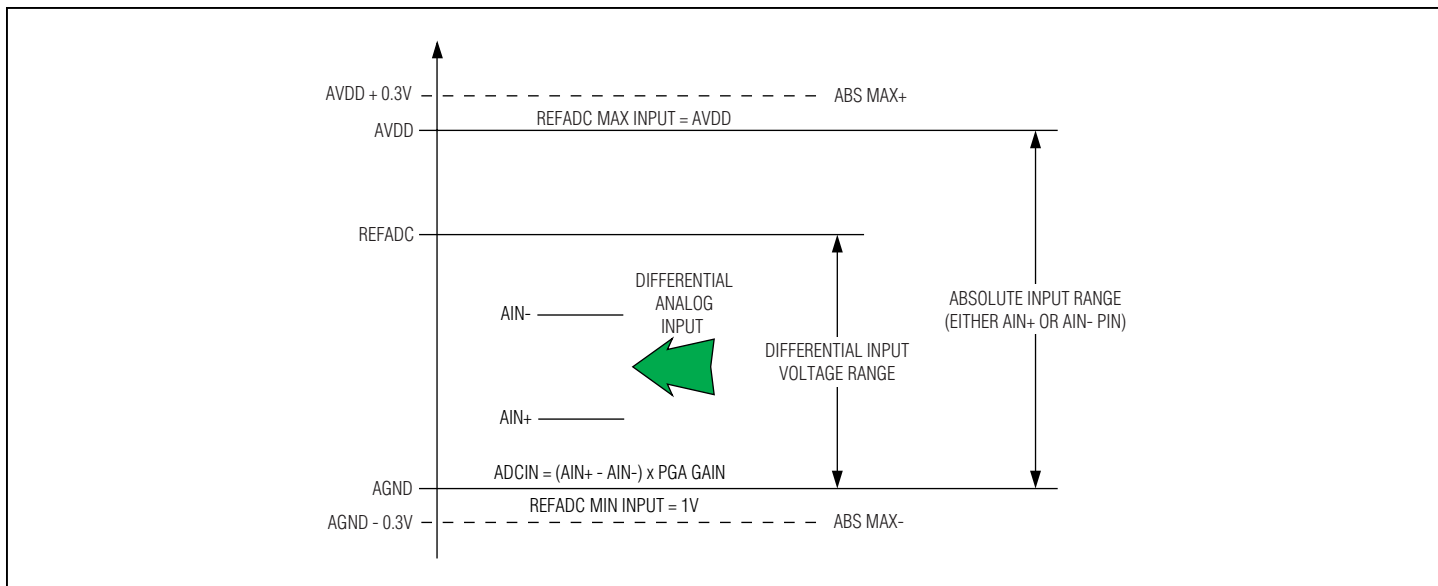
Figure 3-9. Analog Input Range Measuring a Negative Analog Input Value

3.3.8 ADC Clock

The MAXQ7665/MAXQ7666 ADC clock frequency is controlled by the ADCCD2:ADCCD0 bits in the OSCC control register and the system clock speed. These bits determine the ADC clock frequency that is divided down from the system clock. Clock divide ratios of 1, 2, 4, 8, or 16 are supported. The MAXQ7665/MAXQ7666 ADC uses the divided system clock to clock the multiplexer front-end selection, track-and-hold acquisition, and each step of the successive approximation conversion. Note the system clock speed is determined by the divide ratio selected through the CD1 and CD0 bits in the CKCN register. By default, the CD1 and CD0 bits selected divide ratio is 2 and the system clock speed is 3.8MHz if the internal RC oscillator is selected as the system clock source (XT = 0). The ADCCD2:ADCCD0 bits selection further divides the system clock frequency to form the ADC clock.

The XT bit in the system clock control register, CKCN, selects the system clock source. If the XT bit is 0 (reset value), the internal RC oscillator is configured as the system clock source. The internal RC oscillator runs at a nominal 7.6MHz frequency and is trimmed to 1% accuracy at room temperature. If the XT bit is set as 1, the external crystal/clock is configured as the system clock source. See *Section 5* for additional details on the system clock sources.

3.3.9 Auto Shutdown Mode

Power consumption is reduced significantly by placing the MAXQ7665/MAXQ7666 ADC in auto shutdown mode after a conversion. Auto shutdown is ideal for infrequent data sampling and fast wake-up time applications. The ADCASD bit in the ACNT register controls auto shutdown. If the ADCASD bit is set, the ADC automatically shuts down when a conversion is complete and the ADC data ready (ADCRY) flag in the analog status register is set. If the ADCASD is not set, the ADC returns to acquisition mode after a conversion. Auto shutdown reduces the ADC supply current (refer to the MAXQ7665/MAXQ7666 data sheet for exact current saving), but there is a power-up delay of 10 ADC clock cycles (1.25μs at 8MHz) after an auto shutdown.

Note that auto shutdown is different from a full power-down state. The ADC is disabled and fully powered down if the ADCE bit in the APE register is cleared. Full power-down reduces ADC supply current (refer to the MAXQ7665/MAXQ7666 data sheet for exact current saving) and is ideal for infrequent data sampling. The ADCE bit is the master control for ADC operation and, unless set, no ADC conversion is possible. From full power-down state (ADCE = 0), the ADC requires 10 ADC clock cycles (1.25μs at 8MHz) to power up.

Data in the ADC peripheral registers is not lost when the ADC is in auto shutdown or full power-down state. Setting the ADC auto shutdown affects the PGA response. There is an additional delay of 40 cycles introduced in the PGA because of the ADC entering auto shutdown state.

3.3.10 ADC Conversion Start Sources and Timing

The MAXQ7665/MAXQ7666 ADC supports three different conversion start sources: timers, ADC convert pin, and software writes. The conversion start source provides the input trigger for the ADC to start acquisition and conversion. The ADC enable bit (ADCE) in the analog power control register (APE) must be set so the ADC block is enabled for operation. If PGA > 1 is required, the PGA enable (PGAE) bit and the PGA gain selection bits (PGG) must also be set. The ADC source select field (ADCS2:ADCS0) in the ADC control register selects the ADC conversion start source, as shown in Table 3-8.

Table 3-8. ADC Conversion Start Source Selection

ADC SOURCE SELECT (ADCS2:ADCS0)	ADC CONVERSION START SOURCE	DESCRIPTION
000	Timer 0	• Timer output is internally connected to ADC to act as the ADC conversion trigger control.
001	Timer 1	• Configure timer for 8-bit or 16-bit PWM output operation.
010	Timer 2	
011	Reserved, functions as 010	—
100	ADC Conversion Pin	This configures the P0.4/ADCCNV pin as ADC conversion trigger control input pin.
101	ADC Conversion Pin with Inverted Data	This configures the P0.4/ADCCNV pin as ADC conversion trigger control input pin. ADCCNV pin input is inverted and used as ADC conversion trigger control.
110	Continuous Conversion	Writing 110 to ADCS triggers conversion. Once started, for PGA =1, ADC continuously performs a conversion every 16 ADC clock cycles. For PGA > 1, ADC continuously performs a conversion every 56 ADC clock cycles.
111	Start/Busy Bit	Write to the start/busy bit triggers conversion.

All three conversion start sources support single-edge or dual-edge modes of operation, which are determined by the ADCDUL bit. When ADCDUL is set to 1, the ADC operates in dual-edge mode. The rising edge of the selected conversion start source causes the ADC to power up and begin acquisition; the falling edge causes it to sample and perform a conversion. The ADC dual-edge mode is valid only with PGA gain > 1. If ADCDUL is set to 1, make sure the PGA gain (selected by the PGG2:PGG0 bits in the APE register) is greater than 1. The ADC dual-edge mode allows user control of the power-up and acquisition period. An ADC power-up delay is required only if ADC is in auto shutdown from a prior conversion, otherwise, there is no ADC power-up delay. When ADCDUL is 0, the ADC operates in single-edge mode. The rising edge of the selected conversion start source controls the entire conversion, i.e., power-up, acquisition, and conversion. There is no restriction on the PGA gain selection when ADCDUL = 0. Table 3-9 summarizes ADC operation in dual- and single-edge modes.

Table 3-9. ADC Dual- and Single-Edge Modes

ADC DUAL-MODE (ADCDUL)	ADC CONVERSION SOURCE (ADCS2:ADCS0)	ADC CONVERSION TRIGGER	ADC CONVERSION DESCRIPTION
1 (Dual-Edge Mode) (Note: This mode is valid only with PGA gain > 1.)	000 (Timer 0) 001 (Timer 1) 010 (Timer 2) 100 (ADCCNV)		Rising Edge of Conversion Source • Sets T/H into track mode. • Track duration is under user control. • If ADC is in auto shutdown, a minimum of 10 ADC clock cycles power-up delay is required in addition to 80 cycles PGA settling delay (PGA gain > 1), and 3 cycles acquisition delay. • If ADC is not in auto shutdown, a minimum of 40 ADC clock cycles PGA settling delay (PGA gain > 1) and 3 cycles acquisition delay is required. Falling Edge of ADCNV • Sets T/H into hold mode. • Then SAR conversion executes (13 ADC clock cycles).
	101 (Inverted ADCCNV)		Falling Edge of ADCNV • Sets T/H into track mode. • Track duration is under user control. • If ADC is in auto shutdown, a minimum of 10 ADC clock cycles power-up delay is required in addition to 80 cycles PGA settling delay (PGA gain > 1), and 3 cycles acquisition delay. • If ADC is not in auto shutdown, a minimum of 40 ADC clock cycles PGA settling delay (PGA gain > 1) and 3 cycles acquisition delay is required. Rising Edge of ADCNV • Sets T/H into hold mode. • Then SAR conversion executes (13 ADC clock cycles).
	110 (Continuous)	Write 110 to ADCS	Write 110 to ADCS • Sets T/H into track mode. • ADC control logic provides the required track duration. • T/H placed in hold after 43 ADC clock cycles (PGA gain > 1). • Then SAR conversion executes (13 ADC clock cycles). Conversion continuously repeated every 56 ADC clock cycles.
	111 (Start/Busy Bit)	Start/Busy Bit: Write 1 followed by a write 0	Write 1 to Start/Busy Bit • Sets T/H into track mode. • Track duration is under user control. • If ADC is in auto shutdown, a minimum of 10 ADC clock cycles power-up delay is required in addition to 80 cycles PGA settling delay (PGA gain > 1), and 3 cycles acquisition delay. • If ADC is not in auto shutdown, a minimum of 40 ADC clock cycles PGA settling delay (PGA gain > 1) and 3 cycles acquisition delay is required. Write 0 to Start/Busy Bit • Sets T/H into hold mode. • Then SAR conversion executes (13 ADC clock cycles).

Table 3-9. ADC Dual- and Single-Edge Modes (continued)

ADC DUAL-MODE (ADCDUL)	ADC CONVERSION SOURCE (ADCS2:ADCS0)	ADC CONVERSION TRIGGER	ADC CONVERSION DESCRIPTION
0 (Single-Edge Mode)	000 (Timer 0) 001 (Timer 1) 010 (Timer 2) 100 (ADCCNV)		Rising Edge of Conversion Source • Sets T/H into track mode. • ADC control logic provides the required track duration composed of power-up delay (10 cycles), acquisition delay (3 cycles for PGA =1), and settling delay (40 cycles for PGA>1). • If ADC is in auto shutdown, T/H placed in hold after 13 clock cycles if PGA =1 and after 93 clock cycles if PGA > 1. • If ADC is not in auto shutdown, T/H placed in hold after 3 clock cycles if PGA =1 and after 43 cycles if PGA > 1. • Then SAR conversion executes (13 ADC clock cycles).
	101 (Inverted ADCCNV)		Falling Edge of ADCNV • Sets T/H into track mode. • ADC control logic provides the required track duration composed of power-up delay (10 cycles), acquisition delay (3 cycles for PGA =1), and settling delay (40 cycles for PGA>1). • If ADC is in auto shutdown, T/H placed in hold after 13 clock cycles if PGA =1 and after 93 clock cycles if PGA > 1. • If ADC is not in auto shutdown, T/H placed in hold after 3 clock cycles if PGA =1 and after 43 cycles if PGA > 1. • Then SAR conversion executes (13 ADC clock cycles).
	110 (Continuous)	Write 110 to ADCS	Write 110 to ADCS • Sets T/H into track mode. • ADC control logic provides the required track duration. • T/H placed in hold after 3 clock cycles if PGA =1 and after 43 clock cycles if PGA > 1. • Then SAR conversion executes (13 ADC clock cycles). Conversion continuously repeated every16 or 56 ADC clock cycles.
	111 (Start/Busy bit)	Write 1 to Start/ Busy Bit	Write 1 to Start/Busy Bit • Sets T/H into track mode. • ADC control logic provides the required track duration composed of power-up delay (10 cycles), acquisition delay (3 cycles for PGA =1), and settling delay (40 cycles for PGA > 1). • If ADC is in auto shutdown, T/H placed in hold after 13 clock cycles if PGA =1 and after 93 clock cycles if PGA > 1. • If ADC is not in auto shutdown, T/H placed in hold after 3 clock cycles if PGA =1 and after 43 cycles if PGA > 1. • Then SAR conversion executes (13 ADC clock cycles).

MAXQ7665/MAXQ7666 User's Guide

Figure 3-10 shows single-edge-controlled ADC conversion timing when the ADC is in auto shutdown state and the PGA is bypassed. The power-up and acquisition is triggered by the rising edge of the ADC conversion start source signal ADC_CNVST. ADC_CNVST is an internal signal generated from a combination of all the three conversion start sources previously described.
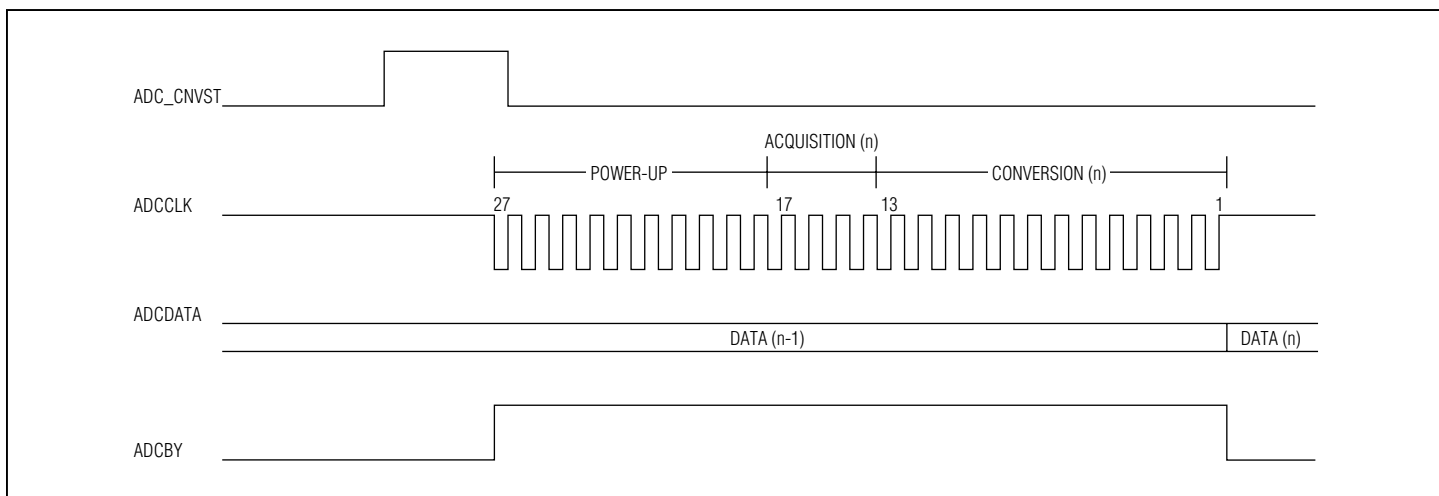
Figure 3-10. Single-Edge ADC Conversion Timing; ADC Previously Off and PGA Bypassed

In single-edged conversions, the ADC control logic provides the necessary power-up, acquisition, and conversion delay.

  a) If ADC is in auto shutdown state, it takes 27 ADC clock cycles before the 12-bit result is available when PGA gain = 1. For PGA gain > 1, it takes an additional 80 cycles for a total of 107 ADC clock cycles before the 12-bit output result is available.

  b) If ADC is not in auto shutdown state, it takes 17 ADC clock cycles before the 12-bit result is available when PGA gain = 1. For PGA gain > 1, it takes a total of 57 ADC clock cycles before the 12-bit result is available.

Figure 3-11 shows single-edge-controlled ADC conversion when the ADC is not in auto shutdown state and the PGA is bypassed.
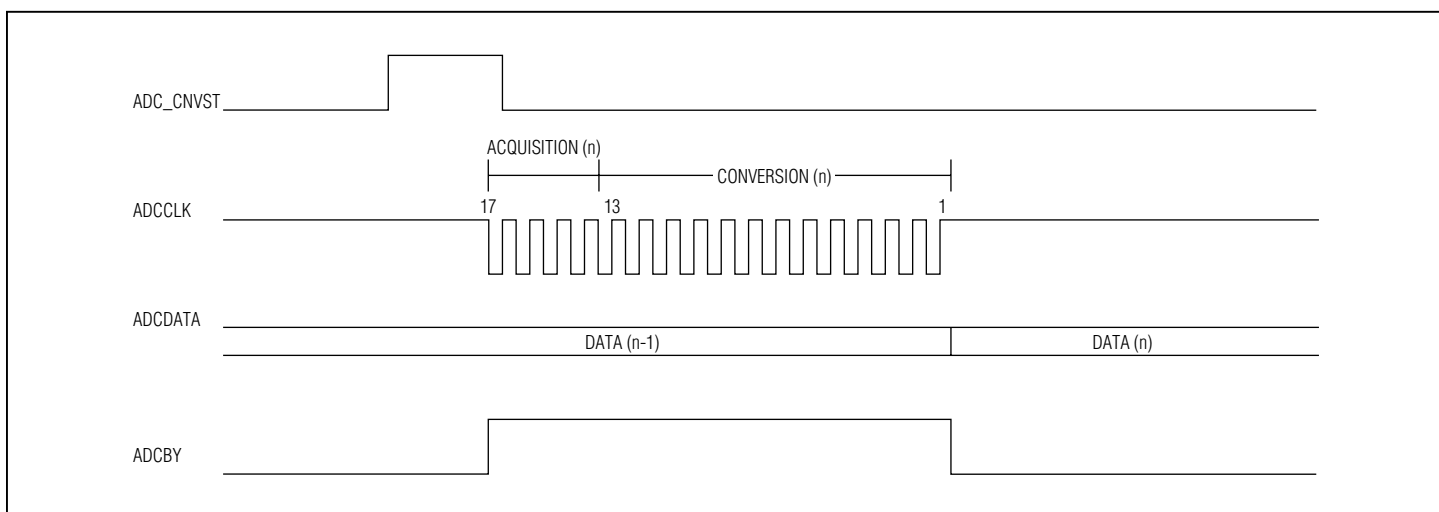
Figure 3-11. Single-Edge ADC Conversion Timing; ADC Previously On and PGA Bypassed

In dual-edged conversions, it is up to the user to provide the required power-up and acquisition delay as explained in Table 3-9.

  a) If ADC is in auto shutdown state, a minimum of 13 ADC clock cycles power-up and acquisition delay is required, in addition to 80 cycles PGA settling delay (PGA gain > 1) and 13 cycles ADC conversion delay for a total of at least 107 ADC clock cycles before the 12-bit result is available.

  b) If ADC is not in auto shutdown state, a minimum of 3 ADC clock cycles acquisition delay is required, in addition to 40 cycles PGA settling delay (PGA gain > 1) and 13 cycles ADC conversion delay for a total of at least 57 ADC clock cycles before the 12-bit result is available.

Figure 3-12 shows dual-edge-controlled ADC conversion when the ADC is in auto shutdown state and the PGA is > 1.
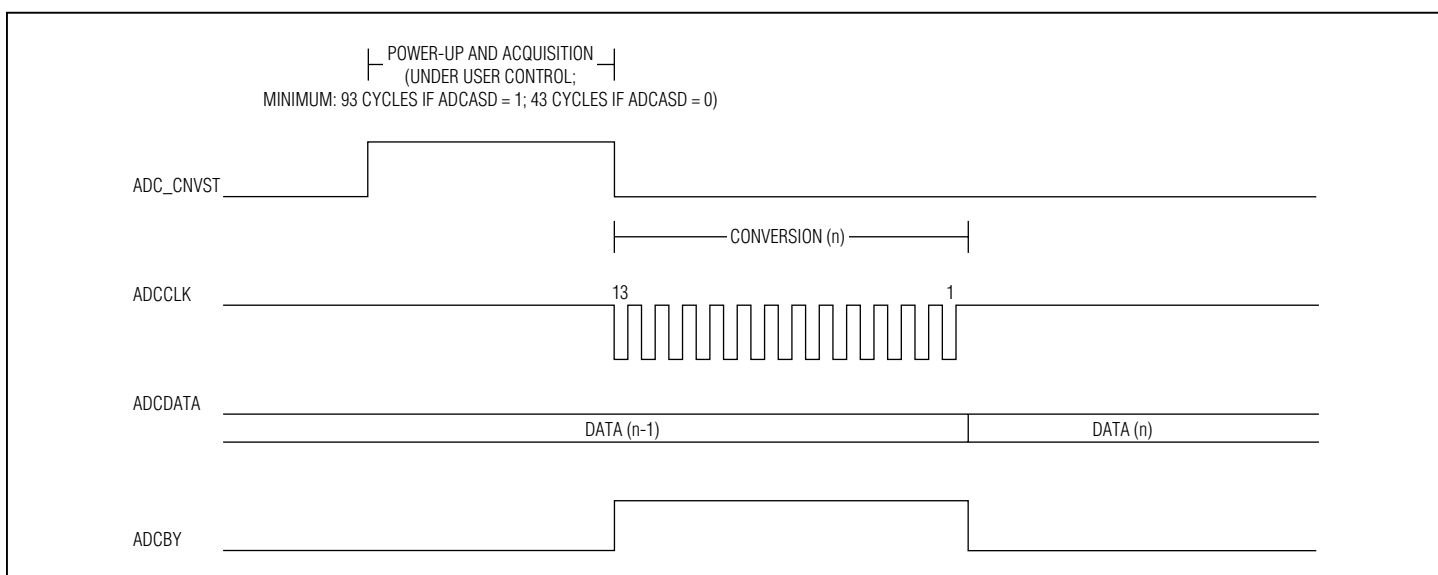
Figure 3-12. Dual-Edge ADC Conversion Timing; ADC Previously Off and PGA > 1

In dual-edge conversion, the power-up and acquisition is triggered by the rising edge of the ADC conversion start source signal ADC_CNVST. At the falling edge, the ADC starts conversion and a 12-bit result is written to the ADC result register in 13 ADC clock cycles. The advantage of dual-edge mode is, depending on the analog input signal's source impedance, the user can provide additional acquisition time if required. Also, in dual-edge mode the user can determine the exact sample instant. This can be very useful in applications where a signal must be sampled precisely every so many microseconds, for example.

3.3.11 ADC Interrupts

The MAXQ7665/MAXQ7666 ADC can generate an interrupt under the following conditions:

  • ADC Data Ready
  • ADC Overrun

The ADC data ready interrupt is generated when the conversion on a channel is complete and a 12-bit result is written into the ADC data register. The ADC data ready (ADCRY) flag in the analog status register (ASR) is also set when a conversion is complete. The ADCIE bit in the analog interrupt register (AIE) must be set for the interrupt to be generated. Otherwise, only the ADCRY status flag is set and the interrupt is not generated. The ADCRY flag is cleared when the ADC data register (ADCD) is read.

The ADC overrun interrupt is generated when an ADC result overrun occurs. The ADC result overrun occurs if the ADC data register is overwritten with a new result before the previous result is read. The ADC overrun (ADCOV) flag in the ASR is set when an overrun occurs. An interrupt is generated only if the AORIE bit in the AIE register is set, otherwise, only the status flag is set. The ADCOV flag is cleared when the analog status register (ASR) is read.

The ADC data ready and ADC overrun interrupts are globally enabled/disabled by the IM5 bit (in the IMR register) and the IGE bit (in the IC register).

3.3.12 Using the ADC

The flow chart in Figure 3-13 highlights all the steps required for initializing and using the ADC.
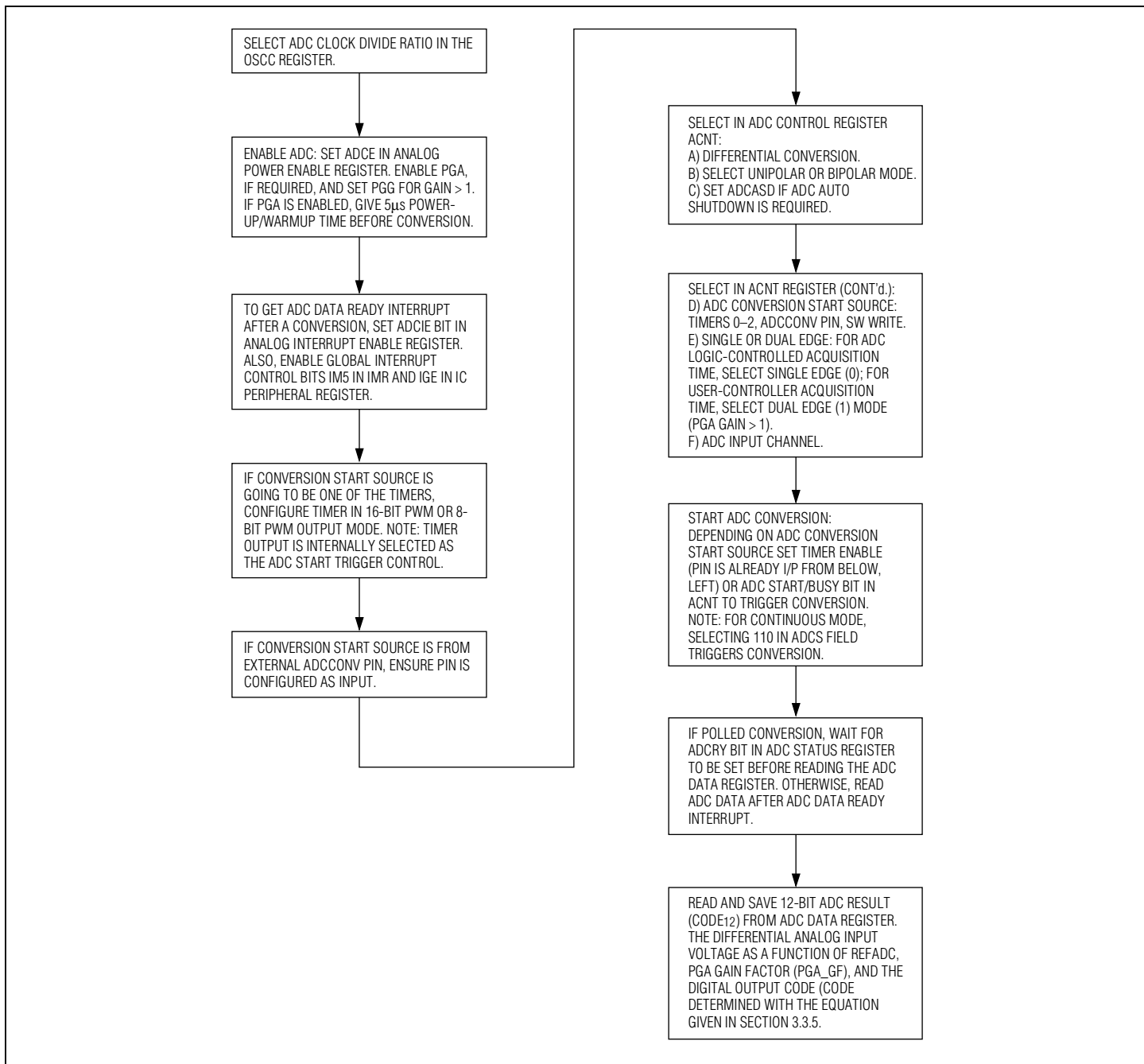

Figure 3-13. Flow Chart for Initializing and Using the ADC

3.4 Temperature Sensor

The MAXQ7665/MAXQ7666 support an internal temperature sensor for local die temperature measurement and a remote temperature sensor drive to measure outside temperature. The internal temperature sensor performs local die temperature measurements with an internal diode-connected transistor. In the remote temperature sensor drive configuration, the device provides the proper bias necessary to measure outside temperature with up to two external diode-connected transistor sensors. Figure 3-14 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 temperature sensor.
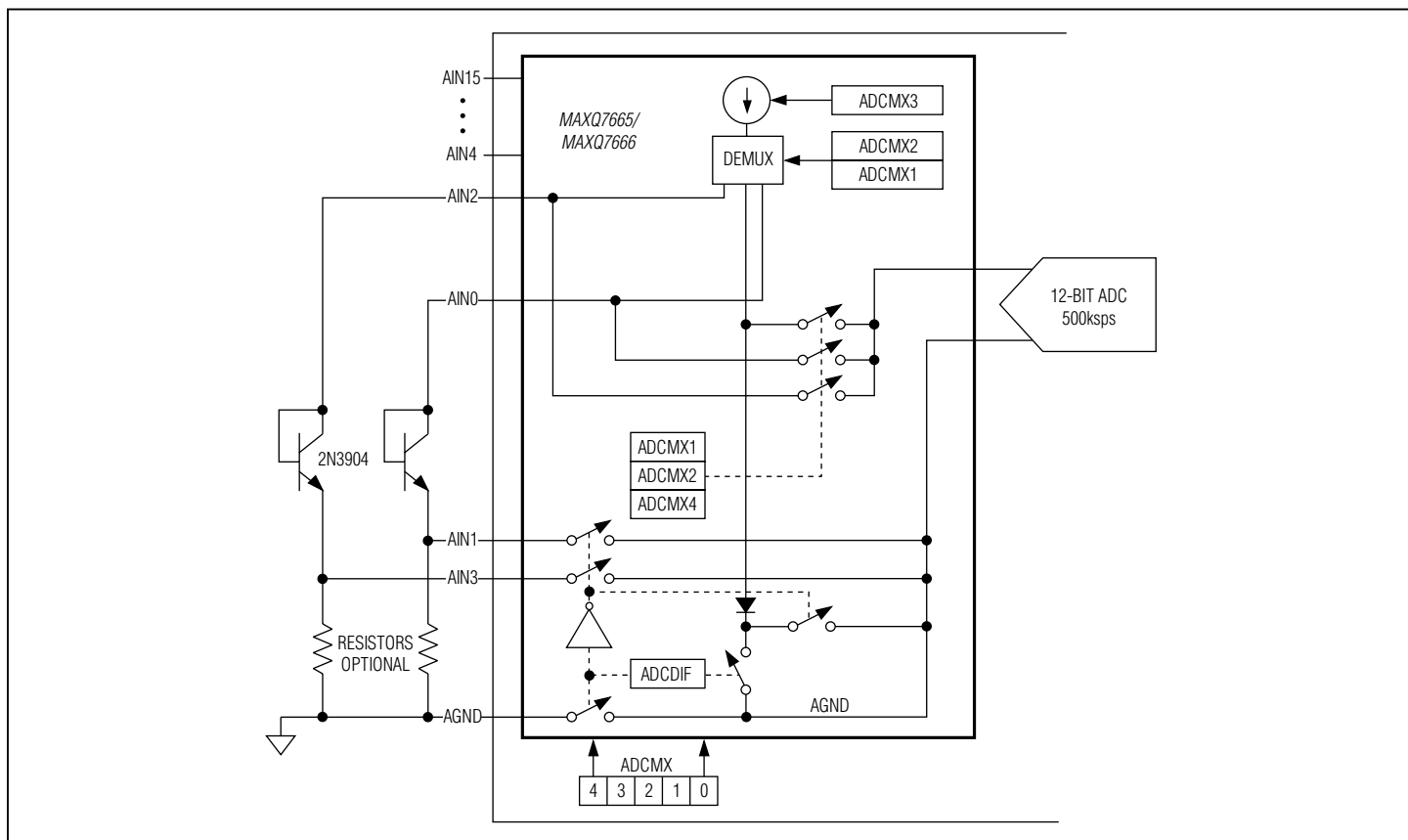
Figure 3-14. MAXQ7665/MAXQ7666 Temperature Sensor Block Diagram

The MAXQ7665/MAXQ7666 perform temperature measurement by measuring the voltage across a diode-connected transistor (internal or remote) at two different current levels. The following equation illustrates the algorithm used for temperature calculations:

$$\text{Temperature} = (V_{HIGH} - V_{LOW}) \times (q/k) / (n \times \ln[I_{HIGH} / I_{LOW}])$$

Where:

V_{HIGH} = sensor-diode voltage with high current flowing (I_{HIGH})

V_{LOW} = sensor-diode voltage with low current flowing (I_{LOW})

q = charge of electron = 1.602×10^{-19} coulombs

k = Boltzman constant = 1.38×10^{-23} J/K

n = ideality factor (slightly greater than 1)

The temperature measurement process is fully automated in the MAXQ7665/MAXQ7666 ROM utility routine "tempConv." All the required setup and temperature measurement algorithm steps for both internal and external temperature measurements are handled in the utility routine and it returns the local or remote temperature result. See the utility ROM section (*Sections 15* and *16*) for details of the routine.

3.4.1 Temperature Sensor Signals

The MAXQ7665/MAXQ7666 temperature sensor uses four (one external diode can be connected between AIN0/AIN1, and a second diode between AIN2/AIN3) external signals in remote temperature sensor drive configuration as explained in Table 3-10.

Table 3-10. Temperature Sensor Signals

SIGNAL	PIN NUMBER		FUNCTION
	48-PIN	56-PIN	
AIN3	13	15	ADC Analog Input/Remote Temperature Sensor. Analog input pins AIN2 and AIN0 are shared with the remote temperature sensor drive line. If the remote temperature sensor drive circuit is not selected, the pin can be used
AIN2	14	16	as a differential input to the multiplexer. In differential input configuration, AIN2 is referenced to AIN3 while AIN0 is referenced to AIN1. When selected, the remote temperature sensor drive circuit supplies suitable current levels for biasing an external diode-connected transistor to monitor temperature away from the microcontroller. The
AIN1	15	17	remote temperature measurement can be made either in single-ended or differential configuration (differential measurements are likely to be more accurate). Note, in differential configuration, AIN3 is used as the return path
AIN0	16	18	for AIN2, and AIN1 is used as the return path for AIN0.

The MAXQ7665/MAXQ7666 temperature sensor block works with the on-chip SAR ADC. Therefore, ensure the ADC is enabled (utility ROM routine handles this) with the required external reference and supply. The temperature sensor has been calibrated for operation with a +5V ADC reference level. It is possible to use other reference levels, but with diminished accuracy.

3.4.2 Using the Temperature Sensor

The following is an overview of the setup required for using the temperature sensor (internal or remote). The full details are available as part of the utility ROM routine.

1) Temperature sensor and ADC must be enabled in the analog power enable register (APE).

2) Set up ADC configuration as follows in the ADC control register (ACNT).

    ADCCS: conversion start from ADC start bit (111)

    ADCDUL: single edge (0)

    ADCBIP: unipolar conversion (0)

    ADCDIF: single-ended or differential input (only for remote configuration)

3) Configure ADCMX4:ADCMX0 bits in the ACNT register as follows to enable/control temperature measurement.

    ADCMX4: This bit must be set to configure ADC input channel for temperature measurement.

    ADCMX3: Set this bit to configure temperature sensor drive current to high value. Clear this bit to configure temperature sensor drive current to low value.

    ADCMX2 and ADCMX1: These bits determine if internal or external temperature sense mode is selected. (See table below.)

    ADCMX0: This bit puts the temperature sensor in auto zero state when it is set to logic 1. The autozeroing is used to cancel internal offset effects.

    Note: The above setup is not required if the temperature conversion ROM utility routine is used. All the required setup and temperature measurement algorithm steps are handled in the utility routine and it returns the local or remote temperature result. The temperature conversion utility ROM routine automatically adds the 12-bit signed offset stored in temperature offset register to the final temperature result. See the utility ROM section (*Sections 15* and *16*) for details of the routine.

ADCMX2	ADCMX1	FUNCTION
0	0	Internal diode-connected transistor based temperature measurement
0	1	Remote diode-connected transistor based temperature measurement on AIN0
1	0	Remote diode-connected transistor based temperature measurement on AIN2
1	1	Reserved

Figure 3-15 shows the nominal transfer function for temperature conversions. Output coding is two's complement with 1 LSB = +0.125°C.
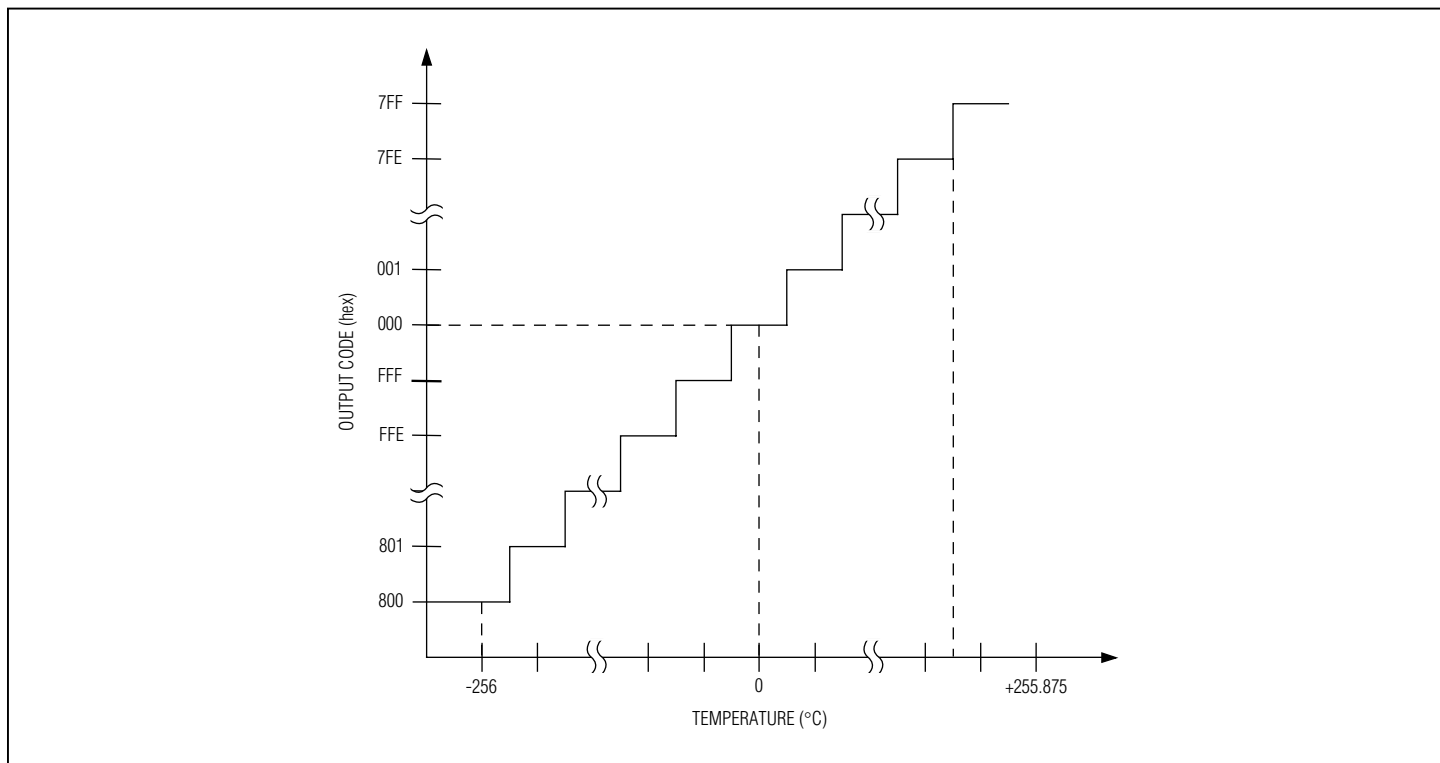

Figure 3-15. Temperature Transfer Function

3.4.3 Internal Temperature Sensor

The MAXQ7665/MAXQ7666 perform local die temperature measurements with an internal diode-connected transistor. The diode bias current is changed from 68μA to 4μA (4.096V reference; for 5V use 74.7μA and 4μA) to produce a temperature-dependent bias voltage difference. The on-chip ADC is used to measure the voltage generated at different drives. The ADC conversion result at 4μA is subtracted from the first at 68μA to calculate a digital value that is proportional to absolute temperature. The final result output is the previously mentioned digital code minus an offset to adjust from Kelvin to Celsius.

The reference voltage used for the temperature measurements is derived from the external ADC reference source to ensure that 1 LSB corresponds to 1/8th of a degree. The internal temperature sensor does not use any analog input channel.

The temperature measurement process is fully automated in the MAXQ7665/MAXQ7666 ROM utility routine "tempConv." All the required setup and temperature measurement algorithm steps are handled in the utility routine and it returns the local or remote temperature result. See the utility ROM section (*Sections 15* and *16*) for details of the routine.

3.4.4 Remote Temperature Sensor Driver

The MAXQ7665/MAXQ7666 temperature sensor supports remote temperature sensor driver on two input channels. The device supports both singled-ended and differential temperature measurements. Remote temperature sensor in differential mode uses analog input channel pairs AIN2/AIN3 and AIN0/AIN1. In single-ended mode, only channels AIN2 and AIN0 are used. The superior common-mode rejection and lower noise of the differential mode reduces measurement errors and provides higher accuracy, while single-ended measurements require a lower number of connections, resulting in a simpler implementation.

3.4.4.1 Differential Temperature Measurement

For differential temperature measurements, connect the anode of a diode-connected transistor to the even input channel and the cathode to the odd input channel of an input pair AIN0/AIN1 or AIN2/AIN3. Run the two sensor connection lines parallel to each other with minimum spacing. This improves temperature measurement accuracy by minimizing the differential noise between the two lines, since they have equal exposure to most sources of noise. For further improved noise rejection, shield the two sensor connections by running them between ground planes, when available.

Configure the MAXQ7665/MAXQ7666 temperature sensor and ADC as explained in *Section 3.4.2* for differential mode (ADCDIF = 1) and enable the remote temperature measurement on an even channel AIN0 or AIN2 (ADCMX2 and ADCMX1 = 01 or 10).

3.4.4.2 Single-Ended Temperature Measurement

For single-ended temperature measurements, connect the anode of a diode-connected transistor to the even input channel AIN0 or AIN2 and the cathode to the ground. Choose ground connections for sensors away from high-current return paths to avoid the introduction of errors caused by voltage drops in the boards/system ground, which is the main drawback for single-ended measurements. Practical options for better accuracy are the use of a star-configured subsystem ground or a signal ground plane; to isolate the anode sensor connection trace away from board and system noise sources; or to shield it with ground lines and ground planes (when available) to prevent accuracy degradation in the temperature measurements caused by magnetic/electric noise induction.

Configure the MAXQ7665/MAXQ7666 temperature sensor and ADC as explained in *Section 3.4.2* for single-ended mode (ADCDIF = 0) and enable the remote temperature measurement on an even channel AIN0 or AIN2 (ADCMX2 and ADCMX1 = 01 or 10).

3.4.5 Remote Temperature Sensor Selection

Temperature-sensing accuracy depends on having a good quality, diode-connected, small-signal transistor as a sensor. Accuracy has been experimentally verified for 2N3904-type devices. The transistor must be a small-signal type with low base resistance. See Table 3-11 for recommended devices.

Table 3-11. Remote Sensor Transistor Manufacturers

MANUFACTURER	MODEL NUMBER
Central Semiconductor (CMPT)	CMPT3904
Fairchild Semiconductors (USA)	MMBT3904
ON Semiconductor (USA)	MMBT3904
Rohm Semiconductor (Japan)	SST3904
Zetex (England)	FMMT3904CT-ND
Diodes Inc.	MMBT3904

3.5 Digital-to-Analog Converter (DAC) Port

The MAXQ7665/MAXQ7666 have a true 12-bit voltage-output DAC with buffered outputs that supports a 15µs maximum settling time at a 12-bit level. The DAC provides a gain of 1 relative to the external REFDAC reference voltage. The MAXQ7665/MAXQ7666 DAC features include:

- 12-bit voltage-output DAC
- 8µs typical and 15µs maximum settling time
- Unity gain output buffer
- External reference
- Straight binary input
- Double-buffered data
- DAC load control from external pin or software write

Figure 3-16 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 DAC.
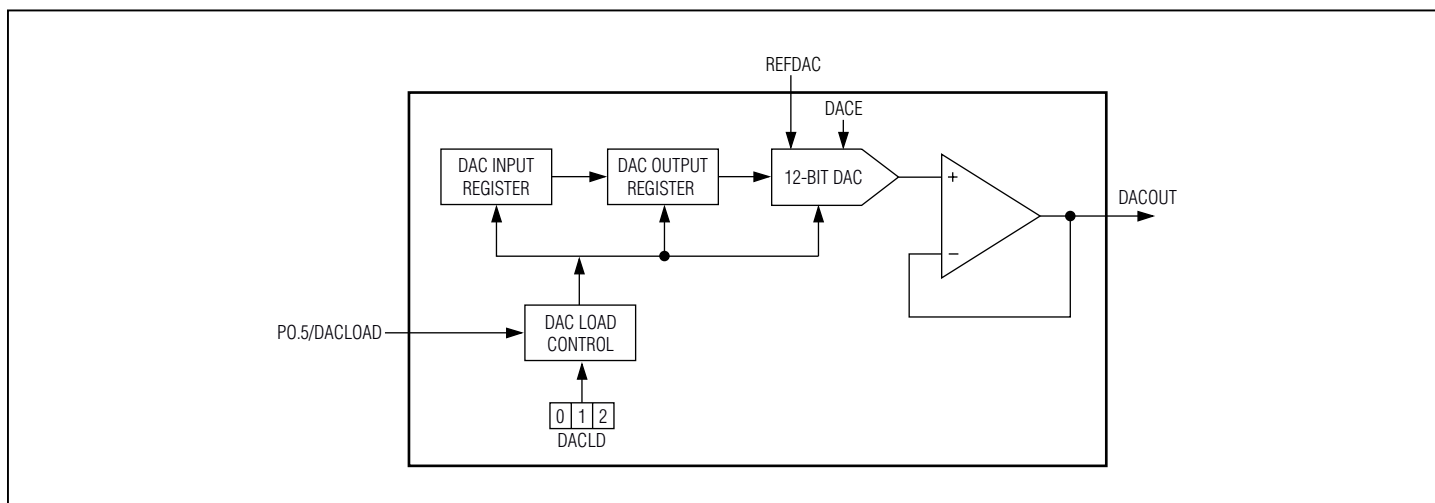

Figure 3-16. DAC Block Diagram

3.5.1 DAC Signals

The MAXQ7665/MAXQ7666 DAC external signals are explained in Table 3-12.

Table 3-12. DAC Signals

SIGNAL	PIN NUMBER		FUNCTION
	48-PIN	**56-PIN**	
REFDAC	7	7	Dedicated DAC Reference Input Pin. The REFDAC pin is used to supply an external precision voltage reference to the internal DAC. The REFDAC can handle a voltage range from 0 to AV_{DD}.
P0.5/DACLOAD	37	43	The DAC load is a shared pin with the digital I/O port 0 bit 5. As the DAC load input signal, it can trigger DAC conversion by loading the DAC output register on a rising or falling edge. After power-up or a reset this pin defaults to a digital I/O port pin with pullup enabled.
DACOUT	17	19	Dedicated DAC Output Pin. If the DAC is disabled, this pin is configured as a 100kΩ pulldown resistor to ground.
AVDD	44	52	Analog V_{DD} Supply. For the MAXQ7665/MAXQ7666, the analog supply voltage is +5.0V.
AGND	5, 8	5, 8	Analog Ground

3.5.2 External Reference Input and Output Buffer

The MAXQ7665/MAXQ7666 DAC external reference input, REFDAC, accepts a voltage range extending from 0 to AVDD. The voltage at REFDAC sets the full-scale output of the DAC. The output voltage is determined using the following equation:

$$DACOUT = (REFDAC \times CODE_{12}) / 2^{12}$$

Where:

CODE$_{12}$ is the numeric value of the DAC's straight binary input code

DACOUT is the output voltage on the external DACOUT pin

REFDAC is the reference voltage on the external REFDAC pin

The MAXQ7665/MAXQ7666 DAC output-buffer amplifier is unity-gain stable with rail-to-rail output voltage swings.

Table 3-13 illustrates the relationship between the straight binary input and the analog output voltage.

Table 3-13. DAC Input Code to Output Voltage (Gain = 1)

BINARY DIGITAL INPUT CODE D11:D0	HEXADECIMAL EQUIVALENT OF D11:D0	DECIMAL EQUIVALENT OF D11:D0 (CODE$_{12}$)	NOMINAL OUTPUT ANALOG VOLTAGE (V)	NOMINAL OUTPUT VOLTAGE (V) (REFDAC = 4.096V)
1111 1111 1111	0xFFF	4095	REFDAC x (4095/4096)	+4.095
1111 1111 1110	0xFFE	4094	REFDAC x (4094/4096)	+4.094
1000 0000 0001	0x801	2049	REFDAC x (2049/4096)	+2.049
1000 0000 0000	0x800	2048	REFDAC x (2048/4096)	+2.048
0111 1111 1111	0x7FF	2047	REFDAC x (2047/4096)	+2.047
0000 0000 0001	0x001	1	REFDAC x (1/4096)	+0.001
0000 0000 0000	0x000	0	0	+0.000

3.5.3 Loading DAC Data Register for Conversion

The MAXQ7665/MAXQ7666 DAC conversion is controlled by the DACLD bits in the DAC control register (DCNT). DACLD selects when and from where the conversion data is sourced. The DAC conversion data is normally sourced from the 12-bit DAC output register (DACO) except in square-wave mode when the 12-bit DAC input register also acts as a source. The DAC output signal (DACOUT) then immediately tracks the conversion data source value. The DACLD bits functionality is explained in Table 3-14.

Table 3-14. DAC Load Control Selection

DACLD2:DACLD0	DAC CONVERSION SOURCE	DAC CONVERSION TRIGGER	DAC DATA TRANSFER AND CONTROL
000	DACO Register	⟋	DAC conversion data is sourced from DACO register under the control of the external DACLOAD input signal. On the rising edge of the DACLOAD input, the DACO register is loaded with the contents of DACI register and converted. The DAC output signal (DACOUT) then immediately tracks the DACO value. Note that this selection enables DACLOAD alternate function on the shared P0.5/DACLOAD pin.
001	DACO Register	Load conversion data to DACI	DAC conversion data is sourced from DACO register under the control of software write to DACI register. The DACO register is loaded with the new contents of DACI when the DACI register is written. The DAC output signal (DACOUT) then immediately tracks the DACO value.
010	—	—	Reserved
011	—	—	Reserved, functions as 001 if set.
100	DACO Register	⟍	DAC conversion data is sourced from DACO register under the control of the external DACLOAD input signal. On the falling edge of the DACLOAD input, the DACO register is loaded with the contents of DACI register and converted. The DAC output signal (DACOUT) then immediately tracks the DACO value. Note that this selection enables DACLOAD alternate function on the shared P0.5/DACLOAD pin.
101	DACI and DACO Registers	⟍⟋	Square-Wave Mode. The data source for conversion is dependent upon edges supplied by the DACLOAD input signal. A falling edge on DACLOAD after entering square-wave mode will supply the data in DACI to the DAC. A rising edge on DACLOAD after entering square-wave mode will supply the data in DACO to the DAC. The DAC output signal (DACOUT) tracks the DACI value on a falling edge and the DACO value on a rising edge.*
110	—	—	Reserved, functions as 000 if set.
111	—	—	Reserved, functions as 000 if set

Note that as the DAC settling time is up to 15µs, toggling DACLOAD at rates substantially faster than that may not allow to settle at either of the intended output values.

3.5.4 DAC Power-Down

The DAC is disabled and fully powered down if the DACE bit in the APE register is cleared. Full power-down reduces analog supply current (refer to the MAXQ7665/MAXQ7666 data sheet for exact current saving) and is ideal for infrequent data conversion. The DACE bit is the master control for DAC operation and, unless set, no DAC conversion is possible. From full power-down state (DACE = 0), the DAC may require up to 1µs to power-up and takes 15µs to settle the final value. This occurs in the worst case when no other analog peripheral is enabled and the bias circuit has, therefore, automatically shut down.

Note: The DACI and DACO registers continue to work even if the DAC is powered down, so user could change the DAC output in power-down mode and then power-up and settle to the new value.

3.5.5 Using the DAC

The following setup is required for using the MAXQ7665/MAXQ7666 DAC.

1) Set DACE bit in analog power enable (APE) register to enable the DAC.

2) If DAC data register loading is going to be controlled from external DACLOAD pin, make sure pin is configured as input in port 0 direction register (PD0).

3) Set up DACLD (DAC load) control bits in DAC control register (DCNT) from external DACLOAD pin or by software write.

4) Initialize DAC input register for load control from DACLOAD falling or rising edge signal.

5) Initialize both DAC input and output register for load control from DACLOAD square-wave signal.

6) For load control by software write, initializing the DAC input register triggers conversion.

SECTION 4: CONTROLLER AREA NETWORK (CAN) MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 4: CONTROLLER AREA NETWORK (CAN) MODULE

The MAXQ7665/MAXQ7666 smart data-acquisition microcontrollers incorporate a single CAN controller (CAN 0), which provides operating modes that are fully compliant with the CAN2.0B specification. The CAN unit provides 15 message centers, each with capability to use 11-bit standard or 29-bit extended acceptance identifiers. Except where explicitly noted, the MAXQ7665 and MAXQ7666 features are identical.

The CAN controller features include the following:

- Full CAN implementation with compliance to CAN2.0A/B protocol standard

- Programmable bit rates from 10kbps to 1Mbps

- 15 Message Centers (14 Tx or Rx, 1 Rx only with FIFO)

- Standard 11-bit or extended 29-bit identification modes

- Support for DeviceNET™, SDS, and higher level CAN protocols

- Remote frame support

- SIESTA low-power mode

- Wakeup on CANRXD edge transition

- Programmable loopback mode

- Support for multiple prioritized interrupt sources: message center interrupts, status interrupts, and error interrupts

- 256 bytes internal dual port memory for information exchange between CAN controller and microcontroller

4.1 Architecture

The microcontroller interface to the CAN controller is broken into two groups of registers. To simplify the software associated with the operation of the CAN controller, all the global CAN status and controls, as well as the individual message center control/status registers, are located in the directly accessible peripheral register map. The remaining registers associated with the data identification, identification masks, format, and data for each message center is located in 256 bytes dual port memory. The access to the dual port data memory by the CAN controller is direct while the access to the dual port memory by the microcontroller is through the CAN 0 data pointer (C0DP) and CAN 0 data buffer (C0DB) registers located in the peripheral register map.

The basic functions covered by the CAN controllers begin with the capability to use 11-bit standard or 29-bit extended acceptance identifiers, as programmed by the microcontroller for each message center. The CAN unit provides 15 message centers, each having a standard 8-byte data field. The first 14 message centers are programmable in either transmit or receive mode. Message center 15 is designed as a receive-only message center with a FIFO buffer to prevent the inadvertent loss of data when the microcontroller is busy. This FIFO buffer is utilized when the microcontroller is not allowed time to retrieve the incoming message prior to the acceptance of a second message into message center 15. Message center 15 also utilizes an independent set of mask registers and identification registers, which are only applied once an incoming message has not been accepted by any of the first 14 message centers. A second filter test is also supported for all message centers (1 to 15) to allow the CAN controller to use two separate 8-bit media masks and media arbitration fields to verify the contents of the first two bytes of data of each incoming message, before accepting an incoming message. This feature allows the CAN unit to directly support the use of higher CAN protocols, which make use of the first and/or second byte of data as a part of the acceptance layer for storing incoming messages. Each message center can also be programmed independently to perform testing of the incoming data with or without the use of the global masks.

Global controls and status registers in the CAN module allow the microcontroller to evaluate error messages, validate new data and the location of such data, establish the bus timing for the CAN bus, establish the identification mask bits, and verify the source of individual messages. Each message center register in dual-port memory is individually equipped with the necessary status and controls to establish direction, interrupt generation, identification mode (standard or extended), data field size, data status, automatic remote frame request and acknowledgment, and masked or nonmasked identification acceptance testing.

DeviceNET is a trademark of Open DeviceNet Vendor Association.

MAXQ7665/MAXQ7666 User's Guide

The priority order associated with the CAN module transmitting or receiving a message is determined by the inverse of the number of the message center, and is independent of the arbitration bits assigned to the message center. Thus, message center 2 has a higher priority than message center 14. To avoid a priority inversion the CAN modules are configured to reload the transmit buffer with the message of the highest priority (lowest message center number) whenever an arbitration is lost or an error condition occurs.

The MAXQ7665/MAXQ7666 CAN controller block diagram is shown in Figure 4-1.
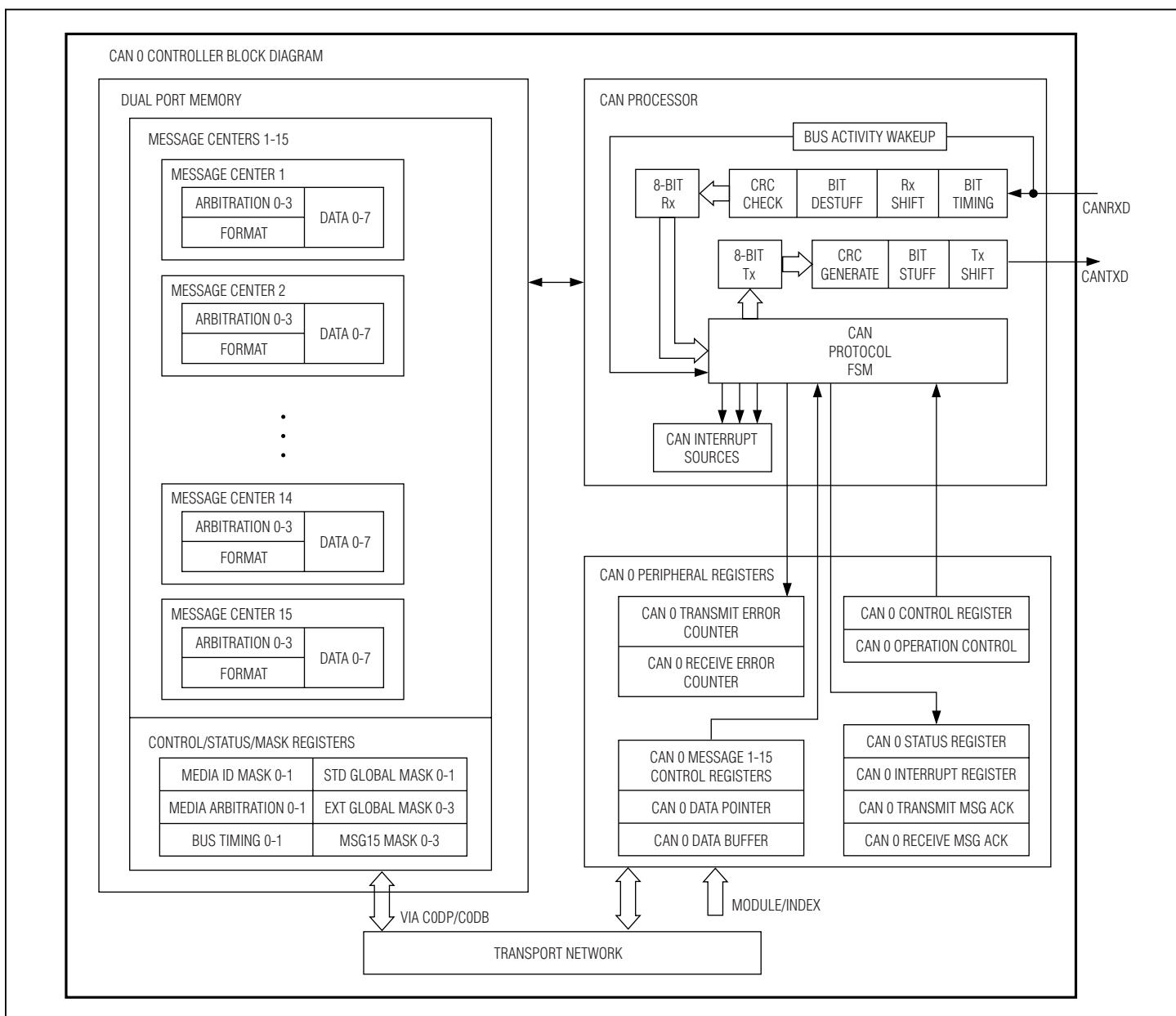

Figure 4-1. MAXQ7665/MAXQ7666 CAN 0 Controller Block Diagram

4.2 CAN Controller Registers

4.2.1 Dual Port Memory Space Registers

This section summarizes CAN 0 control/status/mask information and CAN 0 message center registers that are located in the dual-port memory space. The CAN 0 control/status/mask information is organized in sixteen 8-bit registers. For the 15 CAN 0 message centers, each message center contains sixteen 8-bit registers.

The dual port memory address for the CAN message centers is located from 00h to 7Fh as illustrated in the CAN dual port memory address map (see Figure 4-2). Since the CPU memory access is in 2-byte words, two registers are accessed at one time. A register is shown as the high-order byte (H) or the low-order byte (L) by the word address. The dual port memory control/status/mask registers and message center registers and bits are summarized in the tables that follow.

All the dual port memory locations are accessed by the microcontroller through the CAN 0 data pointer (C0DP) and the CAN 0 data buffer (C0DB) registers.

MESSAGE CENTER 1	ADDRESS	ADDRESS MAP	ADDRESS	CONTROL/STATUS/MASK REGISTER	ADDRESS
RESERVED	08h(L)	**CONTROL/STATUS/MASK REG**	00-07h	MEDIA ID MASK REGISTER 0	00h(L)
RESERVED	08h(H)	**MESSAGE CENTER 1**	08-0Fh	MEDIA ARBITRATION REGISTER 0	00h(H)
MESSAGE 1 ARBITRATION REG 0	09h(L)	**MESSAGE CENTER 2**	10-17h	MEDIA ID MASK REGISTER 1	01h(L)
MESSAGE 1 ARBITRATION REG 1	09h(H)	**MESSAGE CENTER 3**	18-1Fh	MEDIA ARBITRATION REGISTER 1	01h(H)
MESSAGE 1 ARBITRATION REG 2	0Ah(L)	**MESSAGE CENTER 4**	20-27h	BUS TIMING REGISTER 0	02h(L)
MESSAGE 1 ARBITRATION REG 3	0Ah(H)	**MESSAGE CENTER 5**	28-2Fh	BUS TIMING REGISTER 1	02h(H)
MESSAGE 1 FORMAT REGISTER	0Bh(L)	**MESSAGE CENTER 6**	30-37h	STANDARD GLOBAL MASK REG 0	03h(L)
MESSAGE 1 DATA BYTE 0	0Bh(H)	**MESSAGE CENTER 7**	38-3Fh	STANDARD GLOBAL MASK REG 1	03h(H)
MESSAGE 1 DATA BYTE 1	0Ch(L)	**MESSAGE CENTER 8**	40-47h	EXTENDED GLOBAL MASK REG 0	04h(L)
MESSAGE 1 DATA BYTE 2	0Ch(H)	**MESSAGE CENTER 9**	48-4Fh	EXTENDED GLOBAL MASK REG 1	04h(H)
MESSAGE 1 DATA BYTE 3	0Dh(L)	**MESSAGE CENTER 10**	50-57h	EXTENDED GLOBAL MASK REG 2	05h(L)
MESSAGE 1 DATA BYTE 4	0Dh(H)	**MESSAGE CENTER 11**	58-5Fh	EXTENDED GLOBAL MASK REG 3	05h(H)
MESSAGE 1 DATA BYTE 5	0Eh(L)	**MESSAGE CENTER 12**	60-67h	MESSAGE CENTER 15 MASK REG 0	06h(L)
MESSAGE 1 DATA BYTE 6	0Eh(H)	**MESSAGE CENTER 13**	68-6Fh	MESSAGE CENTER 15 MASK REG 1	06h(H)
MESSAGE 1 DATA BYTE 7	0Fh(L)	**MESSAGE CENTER 14**	70-77h	MESSAGE CENTER 15 MASK REG 2	07h(L)
RESERVED	0Fh(H)	**MESSAGE CENTER 15**	78-7Fh	MESSAGE CENTER 15 MASK REG 3	07h(H)

Figure 4-2. CAN Dual Port Memory Address Map

4.2.1.1 Dual Port Memory Space Registers for CAN 0

REGISTER	7	6	5	4	3	2	1	0	DUAL PORT ADDRESS
				CAN 0 CONTROL/STATUS/MASK REGISTERS					
C0MID0	MID07	MID06	MID05	MID04	MID03	MID02	MID01	MID00	00h(L)
C0MA0	M0AA7	M0AA6	M0AA5	M0AA4	M0AA3	M0AA2	M0AA1	M0AA0	00h(H)
C0MID1	MID17	MID16	MID15	MID14	MID13	MID12	MID11	MID10	01h(L)
C0MA1	M1AA7	M1AA6	M1AA5	M1AA4	M1AA3	M1AA2	M1AA1	M1AA0	01h(H)
C0BT0	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	02h(L)
C0BT1	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10	02h(H)
C0SGM0	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21	03h(L)
C0SGM1	MASK20	MASK19	MASK18	0	0	0	0	0	03h(H)
C0EGM0	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21	04h(L)
C0EGM1	MASK20	MASK19	MASK18	MASK17	MASK16	MASK15	MASK14	MASK13	04h(H)
C0EGM2	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5	05h(L)
C0EGM3	MASK4	MASK3	MASK2	MASK1	MASK0	0	0	0	05h(H)
C0M15M0	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21	06h(L)
C0M15M1	MASK20	MASK19	MASK18	MASK17	MASK16	MASK15	MASK14	MASK13	06h(H)
C0M15M2	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5	07h(L)
C0M15M3	MASK4	MASK3	MASK2	MASK1	MASK0	0	0	0	07h(H)

REGISTER	7	6	5	4	3	2	1	0	DUAL PORT ADDRESS
				CAN 0 MESSAGE CENTER 1					
—				Reserved					08h(HL)
C0M1AR0				CAN 0 Message 1 Arbitration Register 0					09h(L)
C0M1AR1				CAN 0 Message 1 Arbitration Register 1					09h(H)
C0M1AR2				CAN 0 Message 1 Arbitration Register 2					0Ah(L)
C0M1AR3				CAN 0 Message 1 Arbitration Register 3			WTOE		0Ah(H)
C0M1F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX$\overline{ST}$	MEME	MDME	0Bh(L)
C0M1D0:7				CAN 0 Message 1 Data Bytes 0–7					0Bh(H)–0Fh(L)
—				Reserved					0Fh(H)

4.2.1.1 Dual Port Memory Space Registers for CAN 0 (continued)

CAN 0 MESSAGE CENTERS 2–14									
REGISTER	7	6	5	4	3	2	1	0	DUAL PORT ADDRESS
	Message Center 2 Registers (Similar to Message Center 1)								10h–17h
	Message Center 3 Registers (Similar to Message Center 1)								18h–1Fh
	Message Center 4 Registers (Similar to Message Center 1)								20h–27h
	Message Center 5 Registers (Similar to Message Center 1)								28h–2Fh
	Message Center 6 Registers (Similar to Message Center 1)								30h–37h
	Message Center 7 Registers (Similar to Message Center 1)								38h–3Fh
	Message Center 8 Registers (Similar to Message Center 1)								40h–47h
	Message Center 9 Registers (Similar to Message Center 1)								48h–4Fh
	Message Center 10 Registers (Similar to Message Center 1)								50h–57h
	Message Center 11 Registers (Similar to Message Center 1)								58h–5Fh
	Message Center 12 Registers (Similar to Message Center 1)								60h–67h
	Message Center 13 Registers (Similar to Message Center 1)								68h–6Fh
	Message Center 14 Registers (Similar to Message Center 1)								70h–77h

CAN 0 MESSAGE CENTER 15									
REGISTER	7	6	5	4	3	2	1	0	DUAL PORT ADDRESS
—	Reserved								78h(HL)
C0M15AR0	CAN 0 Message 15 Arbitration Register 0								79h(L)
C0M15AR1	CAN 0 Message 15 Arbitration Register 1								79h(H)
C0M15AR2	CAN 0 Message 15 Arbitration Register 2								7Ah(L)
C0M15AR3	CAN 0 Message 15 Arbitration Register 3							WTOE	7Ah(H)
C0M15F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	0	EX/$\overline{\text{ST}}$	MEME	MDME	7Bh(L)
C0M15D0:7	CAN 0 Message 15 Data Bytes 0–7								7Bh(H)–7Fh(L)
—	Reserved								7Fh(H)

MAXQ7665/MAXQ7666 User's Guide

4.2.2 Control/Status/Mask Register Descriptions

The CAN control/status/mask registers are located at either the higher order (H) or the lower order (L) byte of the dual port address location from 00h to 07h. Write access to these registers in dual port memory space is allowed only during a software initialization (SWINT = 1). A write by the microcontroller to any of these registers when SWINT = 0 will not alter any of the data in these registers. The CAN control/status/mask registers can be read at any time independent of the state of SWINT. All the CAN control/status/mask registers are cleared to 00h after system reset.

All the dual port memory locations are accessed by the microcontroller through the CAN 0 data pointer and CAN 0 data buffer registers located in the peripheral register map.

CAN 0 Media ID Mask Register 0 (C0MID0)

Bit #	7	6	5	4	3	2	1	0
Name	MID07	MID06	MID05	MID04	MID03	MID02	MID01	MID00
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	00h(L)							

CAN 0 Media ID Mask Register 1 (C0MID1)

Bit #	7	6	5	4	3	2	1	0
Name	MID17	MID16	MID15	MID14	MID13	MID12	MID11	MID10
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	01h(L)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

CAN 0 Media ID Mask Registers 0 and 1 (C0MID0 and C0MID1). These registers function as the mask when performing the media identification test. These registers can only be modified during a software initialization (SWINT = 1). If MDME = 0, the media identification test cannot be performed, and the contents of these registers is ignored. If MDME = 1, the CAN module performs an additional qualifying test on data bytes 0 and 1 of the incoming message, regardless of the state of the EX/$\overline{ST}$ bit. Data byte 1 is compared against CAN media byte arbitration register 1 using C0MID1 as a mask; data byte 0 is compared against CAN media byte arbitration register 0 using C0MID0 as a mask. Any bit in the C0MID1, C0MID0 masks programmed to 0 will ignore the state of the corresponding data byte bit when performing the test. Any bit in the C0MID1, C0MID0 masks programmed to 1 will force the state of the corresponding data byte bit and CAN media byte arbitration registers 1 and 0 to match before considering the incoming message a match. Programming either media ID mask register to 00h effectively disables the media ID test for that byte. As such, the C0MID1, C0MID0 masks act as a don't care following a system reset.

CAN 0 Media Arbitration Register 0 (C0MA0)

Bit #	7	6	5	4	3	2	1	0
Name	M0AA7	M0AA6	M0AA5	M0AA4	M0AA3	M0AA2	M0AA1	M0AA0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	00h(H)							

CAN 0 Media Arbitration Register 1 (C0MA1)

Bit #	7	6	5	4	3	2	1	0
Name	M1AA7	M1AA6	M1AA5	M1AA4	M1AA3	M1AA2	M1AA1	M1AA0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	01h(H)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

CAN 0 Media Arbitration Registers 0 and 1 (C0MA0 and C0MA1). These registers function as the arbitration field when performing the media identification test. If MDME = 0, the media identification test cannot be performed and the contents of these registers is ignored. If MDME = 1, the CAN module performs an additional qualifying test on data bytes 0 and 1 of the incoming message, as mentioned in the description of the CAN media ID mask registers. These registers can only be modified during a software initialization (SWINT = 1).

CAN 0 Bus Timing Register 0 (C0BT0)

Bit #	7	6	5	4	3	2	1	0
Name	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	02h(L)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

Bits 7 and 6: CAN Synchronization Jump Width Select (SJW1 and SJW0). These bits specify the maximum number of time quanta (t_{QU}) cycles that a bit can be lengthened or shortened in one resynchronization to compensate for phase errors detected by the CAN controller when receiving data. These bits can only be modified during a software initialization (SWINT = 1).

SJW1	SJW0	SYNCRHONIZATION JUMP WIDTH
0	0	1 t_{QU} (1)
0	1	2 t_{QU} (2)
1	0	3 t_{QU} (3)
1	1	4 t_{QU} (4)

Note: *Number in parentheses is the SJW value used in bit timing calculations.*

Bits 5 to 0: CAN Baud-Rate Prescaler (BPR5 to BPR0). These bits specify the lower six bits (BPR5–BRR0) of the 8-bit prescale value (BPR7–BPR0). The 256 states defined by the binary combinations of the BPR7–BPR0 bits determine the value of the prescale, which in turn defines the cycle time associated with one time quanta. These bits can only be modified during a software initialization (SWINT = 1). The BPR7, BPR6 bits are located in the COR peripheral register.

BPR7, BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	BAUD-RATE PRESCALE VALUE (BRPV)
00	0	0	0	0	0	0	1
00	0	0	0	0	0	1	2
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
11	1	1	1	1	1	0	255
11	1	1	1	1	1	1	256

CAN 0 Bus Timing Register 1 (C0BT1)

Bit #	7	6	5	4	3	2	1	0
Name	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	02h(H)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

Bit 7: CAN Sampling Rate (SMP). The SMP bit determines the number of samples to be taken during each receive bit time. Programming SMP = 0 takes only one sample during each bit time. Programming SMP = 1 directs the CAN logic to take three samples during each bit time, and to use a majority voting circuit to determine the final bit value. When SMP is set to 1, two additional t_{QU} clock cycles are be added to time segment 1. SMP should not be set to 1 when the baud-rate prescale value (BRPV) is less than 4. This bit can only be modified during a software initialization (SWINT = 1).

Bits 6, 5, 4: CAN Time Segment 2 Select (TSEG26, TSEG25, TSEG24). The eight states defined by the TSEG26, TSEG25, and TSEG24 bits determine the number of clock cycles in the phase segment 2 portion of the nominal bit time, which occurs after the sample time. These bits can only be modified during a software initialization (SWINT = 1).

TSEG26	TSEG25	TSEG24	TIME SEGMENT 2 LENGTH
0	0	0	Invalid
0	0	1	2 t_{QU} (2)
0	1	0	3 t_{QU} (3)
. . .	. . .	. . .	. . .
1	1	0	7 t_{QU} (7)
1	1	1	8 t_{QU} (8)

Note: *Number in parentheses is the TS2_LEN value used in bit timing calculations.*

Bits 3 to 0: CAN Time Segment 1 Select (TSEG13 to TSEG10). The 16 states defined by the TSEG13–TSEG10 bits determine the number of clock cycles in the phase segment 1 portion of the nominal bit time, which occurs before the sample time. These bits can only be modified during a software initialization (SWINT = 1).

TSEG13	TSEG12	TSEG11	TSEG10	TIME SEGMENT 2 LENGTH
0	0	0	0	Invalid
0	0	0	1	2 t_{QU} (2)
0	1	1	0	3 t_{QU} (3)
. . .	. . .	. . .	. . .	. . .
1	1	1	0	15 t_{QU} (15)
1	1	1	1	16 t_{QU} (16)

Note: *Number in parentheses is the TS1_LEN value used in bit timing calculations.*

CAN 0 Standard Global Mask Register 0 (C0SGM0)

Bit #	7	6	5	4	3	2	1	0
Name	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	03h(L)							

CAN 0 Standard Global Mask Register 1 (C0SGM1)

Bit #	7	6	5	4	3	2	1	0
Name	MASK20	MASK19	MASK18	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	03h(H)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

CAN Standard Global Mask Registers 0 and 1 (C0SGM0 and C0SGM1). These registers function as the mask when performing the 11-bit global identification test on incoming messages for message centers 1–14. If message identification masking is disabled (MEME = 0), the incoming message ID field must match the corresponding message center arbitration value exactly, effectively ignoring the contents of these registers. These registers are only used when performing the message identification test for message centers configured as standard receivers (EX/$\overline{ST}$ = 0) having message ID masking enabled (MEME = 1). Thus, the contents are ignored by message centers configured to receive messages with extended identifiers (EX/$\overline{ST}$ = 1). These registers can only be modified during a software initialization (SWINT = 1).

When MEME = 1, any mask bit in the C0SGM1, C0SGM0 mask programmed to 0 creates a don't care condition when the respective bit in the incoming message ID field is compared with the corresponding arbitration bits in message centers 1–14. Any bit in these masks programmed to a 1 forces the respective bit in the incoming message ID field to match identically with the corresponding arbitration bits in message centers 1–14 before said message is loaded into message centers 1–14.

The five least significant bits in the C0SGM1 register are not used and do not perform any comparison of these bit locations. A read of these bits will return 0, writes are ignored.

CAN 0 Extended Global Mask Register 0 (C0EGM0)

Bit #	7	6	5	4	3	2	1	0
Name	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	04h(L)							

CAN 0 Extended Global Mask Register 1 (C0EGM1)

Bit #	7	6	5	4	3	2	1	0
Name	MASK20	MASK19	MASK18	MASK17	MASK16	MASK15	MASK14	MASK13
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	04h(H)							

CAN 0 Extended Global Mask Register 2 (C0EGM2)

Bit #	7	6	5	4	3	2	1	0
Name	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	05h(L)							

CAN 0 Extended Global Mask Register 3 (C0EGM3)

Bit #	7	6	5	4	3	2	1	0
Name	MASK4	MASK3	MASK2	MASK1	MASK0	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	05h(H)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

CAN 0 Extended Global Mask Registers 0 to 3 (C0EGM0 to COEGM3). These registers function as the mask when performing the extended global identification test (EX/$\overline{ST}$ = 1) when message ID masking is enabled (MEME = 1) for message centers 1–14. When EX/$\overline{ST}$ = 0 or MEME = 0 for a given message center, the contents of this register are ignored. These registers can only be modified during a software initialization (SWINT = 1).

When EX/$\overline{ST}$ = 1, the 29 bits of the message ID are compared against the 29 bits of the CAN message center arbitration registers, using the 29 bits of the CAN extended global mask registers as a mask. Any bit in the extended global mask registers set to 0 ignores the state of the corresponding bit in the incoming message ID field when performing the test. Any bit in the extended global mask registers set to 1 forces the state of the corresponding bit in the incoming message ID field and CAN message center arbitration registers 0–3 to match before considering the incoming message a match.

The three least significant bits in the C0EGM3 are not used and do not perform any comparison of these bit locations. A read of these bits always returns 0, and writes to these bits are ignored.

Programming all mask registers to 00h effectively disables the global ID test for that message, accepting all messages. As such, the global mask registers act as a don't care following a system reset.

CAN 0 Message Center 15 Mask Register 0 (C0M15M0)

Bit #	7	6	5	4	3	2	1	0
Name	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	06h(L)							

CAN 0 Message Center 15 Mask Register 1 (C0M15M1)

Bit #	7	6	5	4	3	2	1	0
Name	MASK20	MASK19	MASK18	MASK17	MASK16	MASK15	MASK14	MASK13
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	06h(H)							

CAN 0 Message Center 15 Mask Register 2 (C0M15M2)

Bit #	7	6	5	4	3	2	1	0
Name	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	07h(L)							

CAN 0 Message Center 15 Mask Register 3 (C0M15M3)

Bit #	7	6	5	4	3	2	1	0
Name	MASK4	MASK3	MASK2	MASK1	MASK0	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	07h(H)							

r = read, w = write (allowed only when SWINT = 1 via C0DP/C0DB)

CAN Message Center 15 Mask Registers 0 to 3 (C0M15M0 to C0M15M3). These registers function as the mask for the standard ($EX/\overline{ST}$ = 0) or extended ($EX/\overline{ST}$ = 1) global identification test ($EX/\overline{ST}$ = 1) when message ID masking has been enabled (MEME = 1) for message center 15. These registers can only be modified during a software initialization (SWINT = 1).

When $EX/\overline{ST}$ = 1, the 29 bits of the message ID are compared against the 29 bits of the CAN message center 15 arbitration registers, using the 29 bits of the CAN message center 15 mask registers as a mask. When $EX/\overline{ST}$ = 0, the 11 bits of the message ID are compared against the most significant 11 bits of the CAN message center 15 arbitration registers, using the most significant 11 bits of the CAN message center 15 mask registers as a mask. Any bit in the CAN 0 message center 15 mask registers set to 0 will ignore the state of the corresponding bit in the incoming message ID field when performing the test. Any bit in the CAN message center 15 mask registers set to 1 forces the state of the corresponding bit in the incoming message ID field and CAN message center arbitration registers 0–3 to match before considering the incoming message a match.

The three least significant bits in the C0M15M3 register are not used and will not perform any comparison of these bit locations. A read of these bits always returns 0, and writes to these bits are ignored.

Programming all mask registers to 00h effectively disables the message center 15 ID test, accepting all messages. As such the message center 15 mask registers act as a don't care following a system reset.

MAXQ7665/MAXQ7666 User's Guide

4.2.3 CAN Message Center Register Descriptions

The CAN message center registers are located at either the higher order (H) or the lower order (L) byte of the dual port address locations from 08h to 7Fh. The microcontroller has read/write access to these locations at any time independent of the state of SWINT. All message centers (y = 1–15) are identical, with the exception of 15, which has some minor differences noted in the register descriptions. To simplify the discussion, only one set of registers is shown, with the generic notation y to denote a message center (y = 1–15). All the CAN message center register values are indeterminate after a system reset.

All dual port memory locations are accessed by the microcontroller through the CAN 0 data pointer (C0DP) and the CAN 0 data buffer (C0DB) registers.

CAN 0 Message Center y Arbitration Register 0 (C0MyAR0)

Bit #	7	6	5	4	3	2	1	0
Name	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+1)h(L)							

CAN 0 Message Center y Arbitration Register 1 (C0MyAR1)

Bit #	7	6	5	4	3	2	1	0
Name	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+1)h(H)							

CAN 0 Message Center y Arbitration Register 2 (C0MyAR2)

Bit #	7	6	5	4	3	2	1	0
Name	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+2)h(L)							

CAN 0 Message Center y Arbitration Register 3 (C0MyAR3)

Bit #	7	6	5	4	3	2	1	0
Name	ID4	ID3	ID2	ID1	ID0	—	—	WTOE
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+2)h(L)							

X = Don't care
r = read, w = write (via C0DP/C0DB)

CAN 0 Message Center y Arbitration Registers 0 to 3 (C0MyAR0 to C0MyAR3). These bits form the arbitration value/identification number for the message center y. When the message center is configured in a transmit mode, these registers are the source of the 29-bit ID message field (when $EX/\overline{ST}$ = 1) or the 11-bit ID message field (when $EX/\overline{ST}$ = 0). When $EX/\overline{ST}$ = 1, the 29 message ID bits correspond to ID28–ID0 as shown above. When $EX/\overline{ST}$ = 0, the message ID bits 10–0 correspond to ID28–ID18 in C0MyAR0 and C0MyAR1.

When configured in a receive mode, these registers serve as the arbitration value for message center y, against which incoming messages are compared to ascertain if they are valid for that message center. When $EX/\overline{ST}$ = 1, all 29 bits of the arbitration are used, but when $EX/\overline{ST}$ = 0, only the most significant 11 bits are used.

Note that when a message is successfully loaded, the entire message is loaded to the message center. So, if message ID masking was enabled (MEME = 1), it is possible to overwrite the arbitration register bits that were defined as don't cares for incoming message acceptance.

Bits 2, 1 (C0MyAR3 Only): Reserved. These bits are not used in arbitration. These bits can be modified by the application software. A read of these bits will always return 0.

Bit 0 (C0MyAR3 Only): Writeover Enable (WTOE). This bit controls the ability of a new message to overwrite an existing message in the corresponding message center in receive mode. The DTUP and EXTRQ bits for the message center in question must also be considered to determine the effect of this bit as shown below. The WTOE bit can only be programmed when the SWINT bit is set.

WTOE	DTUP	EXTRQ	RESULT WHEN NEW MESSAGE IS DETECTED
0	0	0	There is currently no unread message or pending external frame request in the message center, so the matching message is written to appropriate message center (1–15).
0	1	x	The message center (1–15) has an unread message or pending external frame request. The incoming matching message is ignored and the message center remains unchanged. The CAN module proceeds to the next lower priority message center to evaluate the incoming message ID and arbitration bits and related masking operations. (No overwrite.)
0	x	1	The message center (1–15) has an unread message or pending external frame request. The incoming matching message is ignored and the message center remains unchanged. The CAN module proceeds to the next lower priority message center to evaluate the incoming message ID and arbitration bits and related masking operations. (No overwrite.)
1	0	x	There is currently no unread message or pending external frame request in the message center, so the matching message is written to appropriate message center (1–15).
1	1	x	The new matching message is stored, overwriting the previously stored message. The ROW bit is set to indicate the overwrite operation.

Special Notes for Message Center 15: The ROW bit in message center 15 is associated with an overwrite of the shadow buffer for message center 15. The EXTRQ and DTUP bits are also shadow buffered to allow the buffered message and the message center 15 value to take on different relationships. The EXTRQ and DTUP values read by software are the current message center 15 values, rather than those of the shadow buffer, as is the case with the ROW bit. The shadow buffer is automatically loaded into message center 15 when **both** the DTUP bit and EXTRQ bit are cleared. If either DTUP = 1 or EXTRQ = 1 when clearing the other, any message in the shadow buffer will not be transferred to the message 15 registers, and any incoming messages for message 15 will be stored in the shadow buffer if WTOE = 1, or will be lost if WTOE = 0.

Special Notes Concerning Remote Frames: For remote frames, which can be received by transmit message centers (1–14) in case of a matching identifier, WTOE and EXTRQ are evaluated. If [(WTOE = 1) or (WTOE = 0 and EXTRQ = 1)], the respective transmit message center (1–14) arbitration bits can be overwritten.

CAN 0 Message Center y Format Register (C0MyF)

Bit #	7	6	5	4	3	2	1	0
Name	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX/$\overline{ST}$	MEME	MDME
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+3)h(L)							

X = Don't care
r = read, w = write (via C0DP/C0DB)

Bits 7 to 4: Data Byte Count (DTBYC3 to DTBYC0). These bits indicate the number of bytes within the data field of the message. When performing a transmit, software sets the DTBYC bits to establish the number of bytes that are to be transmitted. Any value above 8 acts as if 8 had been programmed. When receiving a message, the DTBYC bits indicate the (binary) number of bytes of data in the incoming message (i.e., 0000b = 0 data bytes and 1000b = 8 data bytes).

Bit 3: Transmit/Receive Select (T/$\overline{R}$). This bit is programmed by the application software to indicate if the message is to be transmitted (T/$\overline{R}$ = 1) or received (T/$\overline{R}$ = 0). This bit can only be modified when MSRDY = 0. This bit does not exist for message center 15 and always returns 0 when read from message center 15.

Bit 2: Extended or Standard Identifier (EX/$\overline{ST}$). This bit determines whether the respective message is to use the extended 29-bit identification format (EX/$\overline{ST}$ = 1) or the standard 11-bit Identification format (EX/$\overline{ST}$ = 0). Message centers programmed for one format will only receive/send extended messages in that format and will ignore the alternate format. This bit can only be modified when MSRDY = 0.

Bit 1: Message Identification Mask Enable (MEME). The MEME bit enables (MEME = 1) or disables (MEME = 0) the use of the message identification masking process, associated with the testing of the identification field in the incoming message. This bit can only be modified when MSRDY = 0.

0 = The mask registers are ignored when evaluating the identification bits of the incoming message, and the identification bits of the incoming message and the message center arbitration bits must match exactly to allow receipt of the incoming message. This is equivalent to programming the mask with all zeros. An exact match is also required before a remote data request is allowed.

1 = The mask registers are enabled, comparing only those bits message identification and arbitration bits that correspond to a 1 in the mask register. Since the entire message is loaded on a successful ID match, note that it is possible to overwrite the corresponding arbitration register bits that were defined as don't cares (0) in the standard or extended global ID mask.

Bit 0: Media Identification Mask Enable (MDME). The MDME bit enables (MDME = 1) or disables (MDME = 0) the use of the first 2 bytes of the data field as a message qualifier. This bit can only be modified when MSRDY = 0.

0 = The first 2 bytes of the data field are ignored and not compared.

1 = The first 2 data bytes are masked by the respective media mask ID register and then compared with the media arbitration register 0 and 1 bytes. Only those bits in the first 2 data bytes and the arbitration registers corresponding to a 1 in the mask register are compared. When MDME = 1, the test is also performed before a remote request of data from a remote node is accepted.

CAN 0 Message Center y Data Byte 0 (C0MyD0)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+3)h(H)							

CAN 0 Message Center y Data Byte 1 (C0MyD1)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+4)h(L)							

CAN 0 Message Center y Data Byte 2 (C0MyD2)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+4)h(H)							

CAN 0 Message Center y Data Byte 3 (C0MyD3)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+5)h(L)							

CAN 0 Message Center y Data Byte 4 (C0MyD4)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+5)h(H)							

CAN 0 Message Center y Data Byte 5 (C0MyD5)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+6)h(L)							

CAN 0 Message Center y Data Byte 6 (C0MyD6)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+6)h(H)							

CAN 0 Message Center y Data Byte 7 (C0MyD7)

Bit #	7	6	5	4	3	2	1	0
Name								
Reset	X	X	X	X	X	X	X	X
Access	rw	rw	rw	rw	rw	rw	rw	rw
Dual Port Address	(8y+7)h(L)							

X = Don't care
r = read, w = write (via C0DP/C0DB)

CAN 0 Message Center y Data Bytes 0 to 7 (C0MyD0 to C0MyD7). These bytes hold data to be transmitted or received data.

4.2.4 CAN Global Control and Status Register Descriptions

All the global CAN controls and status, as well as the individual message center control/status registers, are located in the peripheral register map. These registers are located in Module 4, indexes 0h–9h, 11h–1Fh.

Note: All the registers located in the peripheral register map are directly accessible by the microcontroller using the module/index address.

4.2.4.1 CAN 0 Control Register (C0C)

Register Description: **CAN 0 Control Register**
Register Name: **C0C**
Register Address: **Module 04h, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ERIE	STIE	PDE	SIESTA	CRST	AUTOB	ERCS	SWINT
Reset	0	0	0	0	1	0	0	1
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved. Read 0, write ignored.

Bit 7: CAN 0 Error Interrupt Enable (ERIE). Programming the ERIE bit to 1 enables the CAN 0 status bus status (BSS) or error count greater than 96 bit (EC96) to issue an interrupt to the microcontroller, if the C0IE bit in the COR peripheral register is also set. When ERIE is cleared to 0, the error interrupt is disabled.

Bit 6: CAN 0 Status Interrupt Enable (STIE). Programming the STIE bit to 1 allows the CAN 0 status error bits (ER0:ER2), transmit status bit (TXS), receive status bit (RXS), or the wake-up status bit (WKS) to issue an interrupt to the microcontroller if the C0IE bit in the COR peripheral register is also set. When STIE is cleared to 0, the status interrupt is disabled.

Bit 5: CAN 0 Power-Down Enable (PDE). Programming the PDE bit to 1 places the CAN 0 controller into a fully static power-down mode after completion of the last reception, transmission, or after the arbitration was lost or an error condition occurred. Note that the term "after arbitration lost" denotes the fact that the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. Programming PDE = 0 disables the power-down mode. The PDE mode forces all the CAN 0 logic to a static state. The PDE mode can only be removed by either software reprogramming the PDE bit or through a system reset. A read of PDE establishes when the power-down mode has been enabled or removed as per the PDE bit. In all cases, the CAN controller begins operation after 11 recessive bits (a power-up sequence) on the CAN bus per the configuration settings for bit timing, which were programmed prior to entering the power-down mode. Since WKS reflects when the CAN has entered the low-power state as per the SIESTA and/or PDE bit states, a read of the PDE bit establishes when the PDE bit is actually allowed to enable the low-power state. If the low-power state was previously enabled by setting the SIESTA bit, a read of PDE reflects the actual PDE bit value and not the low-power mode. If the low-power mode has not been previously enabled and the PDE bit is set to 1 by software, a read of PDE returns a 0, until such time the PDE bit actually enables the low-power mode following an active transmit or receive operation. When the PDE and SIESTA bit are not used together, a read of the PDE bit, by default, also reflects the actual state of the low-power mode. Setting PDE does not alter any CAN block controls or error status relationships.

Bit 4: Low-Power Siesta Mode (SIESTA). Setting the SIESTA bit to 1 places the CAN 0 controller into a low-power static state after completion of the last reception, transmission, or after the arbitration was lost or an error condition occurred. Note that the term "after arbitration lost" denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. Programming SIESTA = 0 disables the low-power mode. The state of when the SIESTA mode is actually enabled or removed, as per the SIESTA bit programmed value, is reflected in the read of the SIESTA bit. The SIESTA mode is removed when the CAN 0 controller detects CAN 0 bus activity, by reprogramming the SIESTA bit to 0, or by setting either CRST or SWINT to 1. When the SIESTA bit is cleared by either a microcontroller write or activity on the CAN 0 bus, the CAN controller begins operation after 11 recessive bits on the CAN bus (after a power-up sequence) using the configuration settings that were programmed prior to entering the power-down mode. Changing the SIESTA bit from 0 to 1 does not disrupt a currently active receive or transmit, but allows the completion of CAN 0 bus activity prior to entering into the static state. If the CAN 0 logic issues an interrupt as a result of an active CAN 0 receive or transmit while SIESTA is being set, the SIESTA bit is cleared and the CAN 0 logic does not enter the low-power mode. Since WKS reflects when the CAN has entered the low-power state as per the SIESTA and/or PDE bit states, a read of the SIESTA bit establishes when the SIESTA bit is actually allowed to enable the low-power state. If the low-power state was previously enabled by setting the PDE bit, a read of SIESTA reflects the actual SIESTA bit value and not the low-power mode. If the low-power mode has not been previously enabled and the SIESTA bit is set to 1 by software, a read of SIESTA returns a 0 until such time that the SIESTA bit actually enables the low-power mode following an active transmit or receive operation. When the PDE and SIESTA bit are not used together, a read of the SIESTA bit, by default, also reflects the actual state of the low-power mode. Setting SIESTA does not alter any CAN block controls or error status relationships. Note that the PDE and SIESTA bits act independently of each other. Setting both bits leaves the CAN processor in a low-power state until both bits have been cleared by their respective mechanisms.

Bit 3: CAN 0 Reset (CRST). When CRST is set to 1 and after completion of the last reception, transmission, or after arbitration was lost or an error condition occurred, all CAN registers located in the peripheral register memory map, with the exception of the CAN 0 control register, are cleared to 00h. The CAN 0 control register is set to 09h. Note that the term "after arbitration lost" denotes the fact that the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. In accordance with waiting until after the completion of the last reception, transmission, or after arbitration was lost or an error condition occurred, a read of the CRST bit, when previously programmed to 1, reads as a 0 until such time that the CRST = 1 state is actually allowed to place the CAN processor into the reset state. As such, a read of the CRST bit verifies when the CAN reset has been engaged or removed. CAN registers located in the dual port memory map are left in the last state prior to setting CRST. Setting CRST also clears both the receive- and transmit-error counters in the CAN controllers and sets the SWINT bit to 1. CRST must be cleared by software to remove the CAN reset and allow the CAN 0 processor to be initialized. When the CAN processor is not in a BUSOFF mode (BSS = 0) and the CAN processor exits either the software initialization mode (SWINT programmed from 1 to 0) or when the CAN reset is removed (CRST bit is cleared from 1 to 0 and the SWINT bit is cleared from 1 to 0), the CAN processor performs a power-up sequence of 11 consecutive recessive bits before the CAN controller enters into normal operation. If the CAN reset is removed and SWINT is left in the software initialization state, the microcontroller is allowed to immediately start programming the CAN registers and dual port data memory prior to the completion of the power-up sequence. Exiting the software initialization mode (SWINT ≥ 0) requires a power-up sequence of 11 consecutive recessive bits before the CAN controller enters into normal operation. Clearing CRST to 0 from a previous 0 state does not alter CAN processor operation.

Bit 2: Autobaud (AUTOB). When AUTOB is set to 1, an internal loop back is enabled to AND the data from the external CAN bus with the transmitted data of the CAN 0 processor. The ANDed data is then connected to the internal input of the CAN 0 processor. At the same time, the transmitted data is disabled from reaching the external CANTXD pin. The CANTXD pin is placed into a recessive state when AUTOB = 1. The purpose of the internal loopback and the disabled CANTXD pin is to allow the CAN processor to establish the proper CAN bus timing without disrupting the normal data flow between other nodes on the CAN bus. Disabling the CANTXD pin and setting the CANTXD pin to a recessive state prevents the CAN processor from driving nonsynchronized data onto the CAN bus (creating CAN bus errors to other nodes) when being programmed with various frequencies to synchronize the processor with the CAN bus. With AUTOB = 1, the microcontroller autobaud algorithm makes use of the CAN 0 status register RXS and error status bits to determine when a message is successfully received (when AUTOB =1, a successful receive, does not require a store). Each successive baud rate attempt is preceded by the microcontroller clearing the transmit- and receive-error counters by means of a write of 00 to the transmit-error peripheral register and a read of the CAN 0 status register to clear the previous status change interrupt. Note that a write to the transmit-error peripheral register automatically resets the CAN fault confinement state machine to an initial (error active) state if the error counters are cleared to 00h. If, however, the error counters are programmed to a value greater than 128, the CAN processor is in an error-passive state. Appropriate flags are set when the error counter is written with any value. A write of the status register is also used to remove the previous error value in the ER2:ER0 bits. Clearing the error counters also clears the EC96 bit, if set. When BSS = 1, the CAN processor locks out the ability for the microcontroller to write to the error counters by virtue of the fact that the SWINT bit is also forced to a 0 state during the period that the CAN processor performs a bus recovery and power-up sequence. Once the CAN

processor has removed itself from the BUSOFF condition, it also clears BSS = 0, sets SWINT = 1, and clears both the transmit- and receive-error counters to 00h.

The following two situations are examples of how the autobaud function works on the CAN processor. In the first case, consider three nodes, A, B and C, with nodes A and B operating in the normal CAN operational mode (nonautobaud) and node C (a MAXQ7665/MAXQ7666 CAN processor) is attempting to establish a proper baud rate using the autobaud features. If node A transmits a message, node B acknowledges this message, and node C also receives the acknowledged message if it has the same baud rate. If node C does not have the same baud rate as nodes A and B, node C detects the mismatch via the respective error count. Node C then proceeds to adapt its baud rate and attempt to receive the following message.

In the second case, consider a system with only two nodes on the CAN bus. Consider node A in the autobaud mode and the second node on the bus in the normal CAN operational mode. Node B transmits a message and does not receive an acknowledgment, since there is no third node on the bus that is also properly synchronized with the bus and in the normal CAN operational mode. Once node B enters into an error-passive mode (after 16 repeated messages), it begins to send passive error flags. Note that when node B is operating in an error-passive mode, it does not send any dominant errors flags to the bus. Once node A has established the proper baud rate, it receives the correct message. The internal autobaud loopback path also allows the passive acknowledgment error sent by node B to be ANDed with the dominant internally transmitted acknowledgment bit from node A. As such, node A sees no errors, which establishes the fact that it is properly synchronized with the bus. Node A now exits out of the autobaud mode (AUTOB = 0) and enters into the normal CAN operational mode (with full transmit capability to the CAN bus). In this mode, node A then acknowledges the next message from node B.

Bit 1: Error Count Select (ERCS). The ERCS bit establishes in which level the error counters set or clear the EX96/128 bit in the CAN 0 status register. When ERCS = 0, the EC96/128 flag operates in an EC96 mode. In this mode, the EC96/128 bit sets to 1 whenever the error count of either the transmit- or receive-error counters reach a level of 96 or greater. When ERCS = 1, the EC96/128 flag operates in an EC128 mode. In the EC128 mode, the EC96/128 flag is set to 1 whenever the error count of either the transmit- or receive-error counters reach a level of 128 or greater.

Bit 0: Software Initialization (SWINT). (Unrestricted read/write if BSS = 0 and read-only if BSS = 1.) The SWINT bit establishes the initialization state for CAN 0, which disables CAN 0 Bus activity to allow the processor to modify the dual port CAN control/status/mask registers assigned to the message centers without corrupting messages. When SWINT is set to 1 and after completion of the last reception, transmission, or after arbitration was lost or an error condition occurred, all CAN 0 bus activity is disabled, allowing the processor to initialize any or all of the CAN 0 dual port memory. Note that the term "after arbitration lost" denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. A read of the SWINT bit verifies when the CAN processor software initialization mode has been engaged or removed. Although the transmit- and receive-error counters are not cleared when the SWINT bit is set, the CAN 0 transmit- and receive-error counters can be altered by software through the use of the CAN 0 transmit-error peripheral register, as long SWINT = 1. Setting SWINT to 1 also clears the SIESTA bit independently of what is stored to the SIESTA bit location during or prior to the write of the C0C register. Clearing SWINT = 0 also disables the microcontroller from writing to the first 16 bytes of the CAN dual port memory. These 16 locations make up the CAN 0 control/status/mask registers. When SWINT = 0, the microcontroller is allowed to write to any of the MOVX dual port message center register sites. All dual port registers are readable at any time, independent of the SWINT bit. Also note that the SWINT bit does not alter the read or write access to any of the CAN 0 peripheral registers or dual port CAN message center registers. SWINT is programmed to 0 when the processor has completed the dual port control/status/mask initialization and CAN 0 bus activity has started. Software write access to the error counters is disabled when SWINT is cleared to 0. A BUSOFF condition is caused by a high number of errors on the CAN bus. When a BUSOFF condition occurs, the CAN processor clears the SWINT bit to 0 and immediately starts a bus recover and power-up sequence. During this time, the microcontroller is limited to only reading this bit. All microcontroller write access to SWINT is disabled when BSS = 1.

If the SWINT bit is set by a system reset, programming the CRST bit or setting the SWINT bit without the prior detection of a BUSOFF condition can cause an adverse condition. Clearing SWINT by software allows the CAN processor to synchronize itself to the CAN bus after the CAN processor executes a power-up sequence (11 recessive bits). The power-up sequence requires the CAN processor to detect 11 consecutive recessive bits. (In CAN protocol, this is termed a power-up sequence.) When SWINT = 0 by a BUSOFF condition, BUSOFF forces the CAN processor to initiate a standard BUSOFF recovery sequence (128 x 11 recessive bits). This is followed by entering into a reset state, requiring a power-up sequence (11 recessive bits), after which the CAN processor enters into the idle state (normal operation, BSS = 0) and sets the SWINT bit to 1. This bit is not intended for use in changing data within the message centers after the CAN processor is placed into operation. Changes to the arbitration or data fields in the message centers should be done through the use of the MSRDY bit in the respective message (1–15) control registers. The SWINT bit is locked into the SWINT = 1 state until the bus timing registers are programmed to valid states. (The invalid states are 00h. See the CAN bus timing registers in the CAN control/status/mask registers section.)

4.2.4.2 CAN 0 Status Register (C0S)

Register Description: **CAN 0 Status Register**
Register Name: **C0S**
Register Address: **Module 04h, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	BSS	EC96/128	WKS	RXS	TXS	ER2	ER1	ER0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	rw	rw	rw	r	r

r = read, w = write

The bits BSS, EC96, WKS, ER2, ER1, and ER0 in the CAN 0 status register are read-only by the microcontroller. The CAN processor sets or clears these flags (and interrupt sources) as defined by the system aspects associated with each bit. A CAN status register read clears the internal status-change interrupt flag. Unlike RXS and TXS, however, the individual mechanisms that set the ER2, ER0, BSS, EC96, and WKS bits do not reoccur without first being removed by the CAN processor. As a result, a new (0 ≥ 1) change by BSS, EC96, or (1 ≥ 0) change by WKS is required to set a new internal status-change interrupt flag through these bits. In a similar fashion, a read of the CAN status register (which automatically sets ER2:ER0 to 111), followed by a new transmit or receive error, is required to set a new internal status-change interrupt flag. If any one of these bits changes state from a previous 0 to 1 (other than WKS, which changes from 1 to 0) and STIE is set to 1with no other interrupt pending, the INTIN vector in CAN interrupt register is set to 01h. If TXS or RXS are set to 1 and a second message is successfully transmitted or received, and STIE is set to 1 while no other interrupt is pending, the INTIN vector in the CAN interrupt register is also set to 01h. If ER2:ER0 changes from either a 000 or 111, binary state to any state other than 000 or 111, the INTIN vector in CAN interrupt register is also set to 01h. This issues a status-change interrupt request if at least one of the conditions is valid and no other interrupt is pending.

Bits 15 to 8: Reserved. Read 0, write ignored.

Bit 7: CAN 0 Bus Status (BSS). (Read-only.) The BSS bit reflects the current status of the CAN 0 bus. When BSS = 1, the CAN 0 bus is disabled (BUSOFF) and is not capable of receiving or transmitting messages. This condition is the result of the transmit-error counter reaching a count of 256. When the CAN processor detects an error count of 256, the CAN processor automatically sets BSS = 1 and clears SWINT = 0. BSS is cleared to 0 to enable CAN 0 bus activity when the CAN processor completes both the BUSOFF recovery (128 x 11 consecutive recessive bits) and the power-up sequence (11 consecutive recessive bits). Once the CAN processor has completed this relationship, it sets SWINT = 1 and enters into the software initialization state. Once the microcontroller has cleared SWINT to 0, the CAN processor is enabled to transmit and receive messages. BSS is set to 1 whenever the transmit-error counter for CAN 0 reaches the 256 limit. When BSS = 0, the CAN 0 bus is enabled to receive or transmit messages. A change in the state of BSS from a previous 0 to 1 generates an interrupt if the ERIE, C0IE, IM4, and IGE* peripheral register bits are set. All microcontroller writes to the SWINT bit are disabled when BSS = 1. Both the transmit- and receive-error counters are cleared to 00h when the BUSOFF condition is cleared by the CAN module and BSS is cleared to 0.

Bit 6: CAN 0 Error Count Greater Than 96/128 Status (EC96/128). (Read-only.) The EC96/128 bit operates in one of two modes. These two modes are determined by the state of the C0C.1 bit in the CAN 0 control register. Following a system or CAN reset, the C0C.1 bit is cleared to 0, which in turn enables the EC96 mode.

C0C.1 = 0, EC96/128 = EC96. In this mode, when EC96/128 = 1 the interrupt flag indicates that either the CAN 0 transmit-error counter or the CAN 0 receive-error counter has reached an error count of 96, which represents an exceptionally high number of errors. EC96/128 = 0 indicates that the current transmit-error counter and receive-error counter both have an error count of less than 96. A change in the state of EC96/128 from a previous 0 to 1 generates an interrupt if the ERIE, C0IE, IM4, and IGE* peripheral register bits are set. When C0C.1 is programmed to 1, the EC96/128 bit is reconfigured into an EC128 bit flag mode.

C0C.1 = 1, EC96/128 = EC128. In this mode, when EC96/128 = 1 the interrupt flag indicates that either the CAN 0 transmit-error counter or the CAN 0 receive-error counter has reached an error count of 128, which represents an exceptionally high number of errors. EC96/128 = 0 indicates that the current transmit-error counter and receive error-counter both have an error count of less than 128. A change in the state of EC96/128 **from either a previous 0 to 1 or from a previous 1 to 0** generates an interrupt if the ERIE, C0IE, IM4, and IGE* peripheral register bits are set.

Bit 5: CAN 0 Wake-Up Status (WKS). (Read-only). WKS = 0 indicates that CAN 0 is not in a low-power mode. WKS = 1 indicates that CAN 0 is in a low-power mode, based on the setting of either the SIESTA bit or the power-down mode bit to 1. Clearing both the SIESTA bit and power-down enable (PDE) bit forces the WKS bit to 0. A change in the state of WKS from a previous 1 to 0 generates an interrupt if the STIE, C0IE, IM4, and IGE* peripheral register bits are set.

Bit 4: Receive Status (RXS). The RXS bit functions in two different modes. When the AUTOB bit is set to 1, RXS = 1 indicates that a message has been successfully received by CAN 0 since the last read of the CAN 0 status register. Note that this does not mean that the incoming message was or was not stored in a message center, but that the message did not have any errors associated with it during the reception. Messages that are successfully received but are not stored do not pass the arbitration filtering tests required by the internal message centers. When the AUTOB bit is cleared to 0, RXS = 1 indicates that a message has been both successfully received and stored in one of the message centers by CAN 0 since the last read of the CAN 0 status register.

RXS = 0 indicates that no message has been successfully received since the last read of the CAN 0 status register. RXS is only set by the CAN 0 logic and is not cleared by the CAN controller but is only cleared by the microcontroller software, the CRST bit, or a system reset.

When the RXS bit (0 > 1) provides the interrupt source for an interrupt, the microcontroller is required to read the CAN status register to clear the internal status-change interrupt flag. (This flag is seen externally by the presence of the 01 state in the CAN interrupt register.) Once this flag is cleared, the 01 state in the CAN interrupt register is replaced with either the 00 state for no interrupts pending, or a lower priority interrupt code related to one of the message centers. If a second successful reception is detected prior to or after the clearing of the RXS bit in the status register, a second status-change interrupt flag is set to allow a second interrupt to be issued. Each new successful reception generates an interrupt request independent of the previous state of the RXS bit, as long as the CAN status register has been read to clear the previous status-change interrupt flag. Note that if the microcontroller sets the RXS bit from a previous low, it generates an artificial status change interrupt (STIE = 1).

Thus, if RXS is previously set to 0 and a reception was successful, RXS is set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to RXS when RXS was previously 0, RXS is set to 1 and an interrupt can be asserted if enabled. If RXS is previously set to 1 and a reception was successful, RXS remains set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to RXS when RXS was previously 1, RXS remains 1 and no interrupt is asserted.

Bit 3: Transmit Status (TXS). TXS = 1 indicates that a message has been successfully transmitted by CAN 0 (error free and acknowledged) since the last read of the CAN 0 status register. TXS = 0 indicates that no message has been successfully transmitted since the last read of the CAN 0 status register. TXS is only set by the CAN 0 logic and is not cleared by the CAN controller, but is only cleared by the microcontroller software, the CRST bit, or a system reset.

When the TXS bit (0 > 1) provides the interrupt source for an interrupt, the microcontroller is required to read the CAN status register to clear the internal status-change interrupt flag. (This flag is seen externally by the presence of the 01 state in the CAN interrupt register). Once this flag is cleared, the 01 state in the CAN interrupt register is replaced with either the 00 state for no interrupts pending or a lower priority interrupt code related to one of the message centers. If a second successful transmission is detected prior to or after the clearing of the TXS bit in the status register, a second status-change interrupt flag is set to allow a second interrupt to be issued. Each new successful transmission generates an interrupt request independent of the previous state of the TXS bit, as long as the CAN status register has been read to clear the previous status-change interrupt flag. Note that if the microcontroller sets the TXS bit from a previous low, it generates an artificial status change interrupt (STIE = 1).

Thus, if TXS is previously set to 0 and a reception was successful, TXS is set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to TXS when TXS was previously 0, TXS is set to 1 and an interrupt can be asserted if enabled. If TXS is previously set to 1 and a reception was successful, TXS remains set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to TXS when TXS was previously 1, TXS remains 1 and no interrupt is asserted.

IM4 enables interrupt requests from Module 4 and is part of the IMR (Module 8, index 6) register. IGE is interrupt global enable and is part of the IC (Module 8, index 5) register.

Bits 2, 1, 0: CAN 0 Bus Error Status 2, 1, 0 (ER2, ER1, ER0). The ER2:ER0 bits indicate the first type of error that is encountered within a CAN 0 bus frame. The following states outline the specific error type. The eighth state (111 binary) is automatically programmed into ER2:ER0, following a read of the CAN 0 status register to establish if there has been a change in an error condition when doing a future read of the CAN 0 status register. The status data (ER2:ER0) read by the processor must be analyzed or stored in a separate SRAM location, since the ER2:ER0 bits are automatically set to the 111 state following a read. The 111 state remains in the register until a new frame is either transmitted or received, at which time the ER2:ER0 data is undated in relation to the associated transmit or receive message. The ER2:ER0 bits are read-only. Any attempted write to these bits does not affect the bits or the interrupt relationship associated with their value.

The interrupt error represented by the ER2:ER0 bus-error bits is updated following each reception or transmission. Since the stored error from one reception or transmission can be reproduced in the next attempted reception or transmission, a new interrupt is generated whenever a new error condition is detected. This occurs during a reception or transmission, as long as the previous error condition was removed by a read of the CAN 0 status register.

Thus, if ER2:ER0 is set to 000 or 111 and an error condition occurs, this error condition is stored in the ER2:ER0 bits. An interrupt request is made to the microcontroller whenever the ER2:ER0 values change from either a 000 or 111 binary state to any state other than 000 or 111. If a second error occurs prior to the microcontroller performing a read of the CAN status register, the second error is not stored and the first error condition continues to reside in the ER2:ER0 bits. Once the CAN status register is read by the microcontroller, the error status bits are set to 111. If another error occurs after the microcontroller read of the CAN status register, the ER2:ER0 bits are updated with the new error condition.

If two errors come up at the same time, only the one with the higher priority (as given in the following table) is shown. Priority 1 is the highest and 6 is the lowest. The format error is higher than the bit 1 error, since the format error is always a bit 1 error, but a bit 1 error is not necessarily a format error. The error value displayed is selected according to relevance, if the two errors occur at the same time. This is based on which error is the main error and which one is an accompanying error.

ER2	ER1	ER0	PRIORITY	ERROR CONDITIONS
0	0	0	N/A	No Error in Last Frame
0	0	1	2	Bit Stuff Error
0	1	0	5	Format Error
0	1	1	4	Transmit Not Acknowledged Error
1	0	0	6 (lowest)	Bit 1 Error
1	0	1	1 (highest)	Bit 0 Error
1	1	0	3	CRC Error
1	1	1	N/A	No Change Since Last C0S Read

The following are descriptions of the different error types.

Bit Stuff Error: The CAN controller detects more than five consecutive bits of an identical state are received in an incoming message.

Format Error: A received message has the wrong format.

Transmit Not Acknowledged Error: A data frame was sent and the requested node did not acknowledged the message.

Bit 1 Error: The CAN attempted to transmit a message and when a recessive bit was transmitted, the CAN bus was found to have a dominant bit level. This error is not generated when the bit is a part of the arbitration field (identifier and remote retransmission request).

Bit 0 Error: The CAN attempted to transmit a message and when a dominant bit was transmitted, the CAN bus was found to have a recessive bit level. This error is not generated when the bit is a part of the arbitration field. The bit 0 error is set each time a recessive bit is received during the period that the CAN processor is recovering from a BUSOFF recovery period.

CRC Error: The calculated CRC of a received message does not match the CRC embedded in the message.

4.2.4.3 CAN 0 Interrupt Register (C0IR)

Register Description: **CAN 0 Interrupt Register**
Register Name: **C0IR**
Register Address: **Module 04h, Index 02h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	INTIN7	INTIN6	INTIN5	INTIN4	INTIN3	INTIN2	INTIN1	INTIN0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 8: Reserved. Read 0, write ignored.

Bits 7 to 0: CAN 0 Interrupt Indicator 7 to 0 (INTIN7 to INTIN0). The C0IR register provides an indication as to the status of the interrupt sources in the CAN 0 processor. The contents of C0IR indicate that no interrupt is pending (00h), if an interrupt is due to a change in the CAN 0 status register (01h), or if an interrupt has been generated from the successful reception or transmission of one of the 15 message centers (02h–10h). The C0IR register is cleared to 00h following a reset.

The following table shows the values of the INTIN7:INTIN0 bits for each interrupt source along with the respective priority of each.

INTERRUPT SOURCE	INTIN7:INTIN0 HEX VALUE	INTERRUPT PRIORITY
No Pending Interrupt	00	N/A
CAN 0 Status Register	01	Highest = 1
Message 15	02	2
Message 1	03	3
Message 2	04	4
Message 3	05	5
Message 4	06	6
Message 5	07	7
Message 6	08	8
Message 7	09	9
Message 8	0A	10
Message 9	0B	11
Message 10	0C	12
Message 11	0D	13
Message 12	0E	14
Message 13	0F	15
Message 14	10	Lowest = 16

The INTIN vector value does not change when a new interrupt source becomes active and the previous one has not yet been acknowledged and removed (i.e., microcontroller read of CAN 0 status register or microcontroller clear of the appropriate INTRQ bit in the respective CAN 0 message control register), regardless of the fact that the new interrupt has a higher priority or not.

If two properly enabled interrupt sources become active at the same time, the interrupt of highest priority is indicated. For example, if a message center completes a successful transmission or reception and both STIE and ERI, ETI are set, the interrupt indicated by the INTIN7:INTIN0 vector is that of the status-change interrupt (i.e., INTIN7 = 01h and not the message center interrupt; i.e., INTIN7:INTIN0 = MCV.

RXS and TXS are always activated when a transmission or reception is successfully completed. These bits are reset by the microcontroller writing 0 to them. Reading the CAN 0 status register only removes the INTIN7:INTIN0 = 01h vector, but does not clear these bits. These bits (RXS and TXS) can be set by either the CAN processor or microcontroller, but are never reset by the CAN controller.

The CAN 0 interrupt is active when an active interrupt source is indicated in the interrupt vector INTIN7:INTIN0. Changes in the INTIN7:INTIN0 value from a previous 00h state indicate the interrupt source first detected by the CAN processor following the nonactive-interrupt state. The INTIN7:INTIN0 interrupt values displayed in C0IR remain in place until the respective interrupt source is removed, independent of other higher (or lower) priority interrupts that become active prior to clearing the currently displayed interrupt source. The CAN 0 interrupt to the microcontroller is not active when INTIN7:INTIN0 = 00h. In all the other cases, the interrupt line is asserted and the INTIN7:INTIN0 vector must be read to determine the current interrupt source.

When the current (INTIN7:INTIN0) interrupt source is cleared, INTIN7:INTIN0 is changed to reflect the next active interrupt with the highest priority. The status-change interrupt is asserted if there has been a change in the CAN 0 status register (if enabled by the appropriate ERIE and/or STIE bit) and the CAN status interrupt state is set. A message center interrupt is indicated if the INTRQ bit in the respective CAN message control register is set.

The priority of the next interrupt displayed is fixed. For example, consider the case when the current INTIN7:INTIN0 value is that of a message center interrupt. The current INTIN7:INTIN0 interrupt source is cleared (INTRQ = 0), and the status-change interrupt and another message center interrupt are both active. The next interrupt indicated by INTIN7:INTIN0 would be the status-change interrupt that has a higher priority than that of the message center interrupt.

When the current INTIN7:INTIN0 interrupt indicated is a status interrupt and the status register is read, the INTIN7:INTIN0 vector is changed to the next lowest INTIN7:INTIN0 value (which is the next highest priority) of the corresponding message center whose INTRQ bit is set to 1. During this time the interrupt line to the microcontroller remains active. The microcontroller either does an RETI and is then forced back into the same interrupt routine via the active interrupt line, or it remains in the interrupt routine until the microcontroller has cleared all active interrupt sources (INTIN7:INTIN0 = 00h).

An active message center interrupt is cleared by writing 0 to the INTRQ bit in the respective CAN message control register. The interrupt line to the microcontroller goes to an inactive state and the INTIN7:INTIN0 vector resets to 00h if there are no other interrupts active and enabled.

Example Case:

```
t<i>: moment in time
STIE = 1, ERI = 1, ETI = 1
t1: INTRQ[ 1]  = 1, RXS = 1                 [ INTIN = 1, interrupt line = active]
t2: INTRQ[ 15] = 1, TXS = 1                 [ INTIN = 1, interrupt line = active]
t3: ERR[ 2:0]  = 3' b101                     [ INTIN = 1, interrupt line = active]
t4: Begin processing interrupts by micro    [ INTIN = 1, interrupt line = active]
t5: TXS = 1 ≥ 0                             [ INTIN = 1, interrupt line = active]
t6: RXS = 1 ≥ 0                             [ INTIN = 1, interrupt line = active]
t7: ERR[ 2:0]  = 101 ≥ 111                  [ INTIN = 2, interrupt line = active]
t8: INTRQ[ 15]  = 1 ≥ 0                     [ INTIN = 3, interrupt line = active]
t9: INTRQ[ 1]  = 1 ≥ 0                      [ INTIN = 0, interrupt line = inactive]
```

To properly reflect the value of each interrupt source in the C0IR register, each source must be enabled via the respective interrupt enable. These include ERIE and/or STIE enable in the case of status-change-related interrupt (01) sources, and either the ETI or ERI enable for each message center interrupt (02h–10h) source. The status values of the interrupt sources in C0IR do not, however, require setting either the EA or C0IE bits in the IE and EIE peripheral registers.

There are two methods for verifying message center interrupts. One method uses the ETI/ERI interrupt enables within each CAN message control register, and the other method uses the STIE interrupt enable in the CAN status register.

STIE = 1: When a transmission or a reception by the corresponding message center was successfully completed, the status change interrupt and the RXS/TXS bit are asserted. To understand how each bit in the status register acts as an interrupt source, review the bit descriptions of each bit in the status register. Note that a successful receive in relation to the RXS bit is dependent on the AUTOB bit (AUTOB = 1 is successful receive only and AUTOB = 0 is successful receive and store). This is not the case with the following ERI relationship, in which a receive is considered a successful receive only if the data was stored in the respective message center. The STIE interrupt method requires the microcontroller to poll each message center to establish the respective interrupt source following each status change interrupt.

ETI = 1 and/or ERI = 1: When a successful transmission or a successful reception and store by the corresponding message center are completed, the interrupt is asserted according to its priority. This method relies on the hardwired priority of the message centers. Minimal microcontroller intervention is required.

Terms used in the following descriptions:

Value A: Value that was indicated before and is not zero.

MCV (Message Center's Value): Interrupt indicator value that corresponds to the message center that received or transmitted a message, i.e., 02 for MC15, 03 for MC1, etc.

1a. STIE = 1 Only (Polling Method: ETI = ERI 0) with No Prior Interrupt Active

CASE	STIE	CHANGE DETECTED IN BITS 5:0 OF C0S PERIPHERAL REGISTER	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	Not affected	Inactive
B	0	Yes	Value A or 0	Not affected	Inactive
C	1	No	Value A or 0	Not affected	Inactive
D	1	Yes	Value A or 0 > 1	Not affected	Active

It is important to note that additional changes in bits 4–0 (RXS, TXS) of the CAN 0 status register can be detected even if these bits have not been cleared by the microcontroller. The only requirement for the second status-change interrupt is for the microcontroller to read the CAN 0 status register to clear the previous interrupt. Multiple changes in the CAN 0 status register, which are read from the CAN 0 status register and occur without the microcontroller clearing the status-change interrupt, appear as one interrupt. The WKS bit is a read-only bit and is not altered by a write from the microcontroller. The ER2:ER0 bits are automatically set to 111 following a read of the CAN status register.

Although not related to a successful transmission or reception, ERIE =1 also enables a similar interrupt relationship when bits 6 or 7 are changed in the CAN status register, with ERIE = 1.

1b. ERIE = 1 with No Prior Interrupt Active

CASE	ERIE	CHANGE DETECTED IN BIT 7 OR 6 OF C0S PERIPHERAL REGISTER	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	Not affected	Inactive
B	0	Yes	Value A or 0	Not affected	Inactive
C	1	No	Value A or 0	Not affected	Inactive
D	1	Yes	Value A or 0 > 1	Not affected	Active

2. ERI = 1 and/or ETI = 1 Only (STIE = 0: Hardwired Method) with No Prior Interrupt Active

CASE	ERIE	RECEPTION SUCCESSFUL?	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	0	Inactive
B	0	Yes	Value A or 0	0	Inactive
C	1	No	Value A or 0	0	Inactive
D	1	Yes	Value A or (MCV > INTIN)	1	Active

4.2.4.4 CAN 0 Transmit-Error Register (C0TE)

Register Description: **CAN 0 Transmit-Error Register**
Register Name: **C0TE**
Register Address: **Module 04h, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r*	r*	r*	r*	r*	r*	r*	r*

Bit #	7	6	5	4	3	2	1	0
Name	C0TE.7	C0TE.6	C0TE.5	C0TE.4	C0TE.3	C0TE.2	C0TE.1	C0TE.0
Reset	0	0	0	0	0	0	0	0
Access	r*	r*	r*	r*	r*	r*	r*	r*

r = read, * = write only when SWINT = 1 and BUSOFF = 0

Bits 15 to 8: Reserved. Read 0, write ignored.

Bits 7 to 0: CAN 0 Transmit-Error Register 7 to 0 (C0TE.7 to C0TE.0). This register indicates the number of accumulated CAN 0 transmit errors. The CAN 0 module responds in different ways to varying number of errors as shown in the following table. This register can only be modified by software when SWINT = 1 and BUSOFF = 0. All software writes to this register simultaneously load the same value into the CAN 0 transmit error register and the CAN 0 receive error register. Writing 00h to this register also clears the CAN 0 error-count-exceeded bit, CECE (C0S.6). This register is cleared following all hardware resets and software resets enabled by the CRST bit in the CAN 0 control register.

C0TE VALUE	CAN 0 STATE
Value < 96	Error Active Mode, CAN 0 Bus On (BUSOFF = 0)
128 > Value ≥ 96	Error Active Mode, CAN 0 Bus On (BUSOFF = 0), Warning Level
255 ≥ Value ≥ 128	Error Passive Mode, CAN 0 Bus On (BUSOFF = 0)
Value > 255	CAN 0 Bus Off (BUSOFF = 1)

4.2.4.5 CAN 0 Receive-Error Register (C0RE)

Register Description: **CAN 0 Receive-Error Register**
Register Name: **C0RE**
Register Address: **Module 04h, Index 04h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	C0RE.7	C0RE.6	C0RE.5	C0RE.4	C0RE.3	C0RE.2	C0RE.1	C0RE.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 8: Reserved. Read 0, write ignored.

Bits 7 to 0: CAN 0 Receive-Error Register 7 to 0 (C0RE.7 to C0RE.0). This register provides a means of reading the CAN 0 receive-error counter. New values can be loaded into the receive-error counter through the CAN 0 transmit-error register. C0RE is cleared to 00h following all hardware resets and software resets enabled by the CRST bit in the CAN 0 control register.

4.2.4.6 CAN 0 Operation Control Register (COR)

Register Description: **CAN 0 Operation Control Register**
Register Name: **COR**
Register Address: **Module 04h, Index 05h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	CAN0BA	INCDEC	AID	C0BPR7	C0BPR6	—	C0BIE	C0IE
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
***Note:** This register is cleared to 00h on all forms of reset.*

Bits 15 to 8, 2: Reserved. Read 0, write ignored.

Bit 7: CAN 0 Bus Activity Status (CAN0BA). The CAN0BA signal is a latched status bit that is set if a CAN bus activity is detected. This bit is cleared by a reset or software once set.*

---

*A change in the state of CAN0BA from a previous 0 to 1 generates an interrupt if the C0BIE, IM4, and IGE peripheral register bits are set.

Bit 6: Increment/Decrement Select (INCDEC). This bit determines the C0DP's auto-increment/decrement function when AID bit is set to logic 1. When INCDEC is set to logic 0, the contents of C0DP are decremented by 1 after a read/write access to the C0DB register. When INCDEC is 1, the contents of the C0DP are incremented by 1 after a read/write access to the C0DB register.

Bit 5: Automatic Increment/Decrement Enable (AID). This bit enables automatic increment or decrement of the CAN data pointer (C0DP) after the CAN data buffer (C0DB) has been accessed. The actual increment/decrement function is dependent on the setting of the INCDEC bit. When AID is set to logic 1, the contents of the C0DP are incremented (or decremented) by 1 after a read/write access to the C0DB register. When AID is cleared to 0, a read/write access to the C0DB register has no effect on the contents of the C0BP register.

Bits 4 and 3: CAN 0 Baud-Rate Prescale Bits 7 and 6 (C0BPR7 and C0BPR6). The C0BPR7 and C0BPR6 bits establish the two high-order bits associated with the 8-bit baud-rate prescaler in the CAN 0 controller. Note that the C0BPR7 and C0BPR6 bits cannot be written when the SWINT bit in the CAN 0 control register is cleared to 0. The remaining CAN baud-rate prescale bits are in the CAN 0 bus timing register (C0BT0).

Bit 1: CAN 0 Bus Activity Interrupt Enable (C0BIE). When this bit is set to logic 1, detecting a CAN bus activity initiates an interrupt. When this bit is set to logic 0, the interrupt capability caused by CAN bus activity is disabled.

Bit 0: CAN 0 Interrupt Enable (C0IE). When this bit is set to logic 1, a change of CAN status register initiates an interrupt if the corresponding ERIE or STIE bit in the CAN control register is also set. When this bit is set to logic 0, the interrupt capability caused by change of CAN status register is disabled.

4.2.4.7 CAN 0 Data Pointer Register (C0DP)

Register Description: **CAN 0 Data Pointer Register**
Register Name: **C0DP**
Register Address: **Module 04h, Index 06h**

Bit #	15	14	13	12	11	10	9	8
Name	C0DP.15	C0DP.14	C0DP.35	C0DP.12	C0DP.11	C0DP.10	C0DP.9	C0DP.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	C0DP.7	C0DP.6	C0DP.5	C0DP.4	C0DP.3	C0DP.2	C0DP.1	C0DP.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: CAN 0 Data Pointer Register Bits 15 to 0 (C0DP.15 to C0DP.0). This register is used as a pointer for direct memory access to the dual port memory. Only the lower seven bits are significant; the other high-order bits are tied to 0. To access the dual port memory, a valid address in the range of 00h to 7Fh must be presented in this register. The contents of this register can be automatically incremented or decremented after a read/write access to the C0DB register by setting the AID and INCDEC bit in the COR register.

Note that the dual port memory is synchronous memory and its read pointer must be activated before a memory read. A write to C0DP or a read from C0DB automatically activates the C0DP as a read pointer and remains in effect until the C0DP is used as a write pointer. In this case, C0DP must be reactivated by a write of C0DP before reading data from C0DB, or it must be reactivated by a back-to-back read from C0DB if the auto-increment/decrement function is disabled. Valid data is presented by the second read operation. If it is suspected that the data at the memory location addressing by the C0DP has been changed, the C0DP must be reactivated to ensure the new data has been pushed to the C0DB register.

4.2.4.8 CAN 0 Data Buffer Register (C0DB)

Register Description: **CAN 0 Data Buffer Register**
Register Name: **C0DB**
Register Address: **Module 04h, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	C0DB.15	C0DB.14	C0DB.35	C0DB.12	C0DB.11	C0DB.10	C0DB.9	C0DB.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	C0DB.7	C0DB.6	C0DB.5	C0DB.4	C0DB.3	C0DB.2	C0DB.1	C0DB.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *The contents of the buffer are 0000h on all forms of reset.*

Bits 15 to 0: CAN 0 Data Buffer Register Bits 1 to 15 (C0DB.15 to C0DB.0). This register location refers to both the input and output buffers of the dual port memory, depending on write or read operation. Writing to this register location triggers a write operation that transfers the source data to the dual port memory as addressed by the C0DP register at the end of the cycle. Reading from this register location transfers data from the dual port memory to the destination, as selected by the C0DP register at the end of the cycle. Read/write access to this register location also triggers increment/decrement function to the C0DP if the AID bit is set to logic 1. The auto-increment/decrement operation is initiated immediately after the completion of the memory access cycle.

Note that the dual port memory is synchronous memory and its read pointer must be activated before a memory read. A write to C0DP or a read from C0DB automatically activate the C0DP as a read pointer and will remain in effect until the C0DP is used as a write pointer. In this case, C0DP must be reactivated by a write of C0DP before reading data from C0DB, or be reactivated by a back-to-back read from C0DB if the auto-increment/decrement function is disabled. Valid data is presented by the second read operation. If it is suspected that the data at the memory location addressed by the C0DP has been changed, the C0DP must be reactivated to ensure the new data has been pushed to the C0DB register.

Note that, while using the auto-increment/decrement feature for C0DB reads and writes, the RAM is preread in order to get the data out to the transport network in one cycle. If it is suspected that the data at the memory location specified by C0DP has changed, then C0DP must be rewritten (i.e., prime the pump). If C0DP is not rewritten, the data returned to C0DB can be incorrect. Sample pseudo code follows:

```
Set COR peripheral register to value 0x60;      #enable auto increment
Set C0DP peripheral register to value 0x0B;     #pre-read dual port memory location 0x0B
Read C0DB peripheral register value;            #read from location 0x0B, pre-read 0x0C
Read C0DB peripheral register value;            #read from location 0x0C, pre-read 0x0D
##MESSAGE RECEIVED IN MC1, OVERWRITING          0x0D##

Read C0DB peripheral register value;            #read from location 0x0D, data is incorrect!!
```
Solution: Rewrite C0DP after MC1 is updated as shown below.

```
...
Read C0DB peripheral register value;            #read from location 0x0C, pre-read 0x0D
##MESSAGE RECEIVED IN MC1, OVERWRITING 0x0D##
Set C0DP peripheral register to value 0x0D;     #set C0DP, pre-read 0x0D (priming the pump)
Read C0DB register value;                       #read from location 0x0D, data is correct!!
```

4.2.4.9 CAN 0 Receive Message Stored Register (C0RMS)

Register Description: **CAN 0 Receive Message Stored Register**
Register Name: **C0RMS**
Register Address: **Module 04h, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	—	C0RMS.15	C0RMS.14	C0RMS.13	C0RMS.12	C0RMS.11	C0RMS.10	C0RMS.9
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	C0RMS.8	C0RMS.7	C0RMS.6	C0RMS.5	C0RMS.4	C0RMS.3	C0RMS.2	C0RMS.1
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Note: *This register is cleared to 0000h on all forms of reset, including the reset established by the CRST bit.*

Bit 15: Reserved. Read 0, write ignored.

Bits 14 to 0: CAN 0 Message Center 15 to 1 Receive and Store (C0RMS.15 to C0RMS.1). The C0RMS bits indicate which message center (1 to 15) has successfully received and stored the last incoming message. The contents of the C0RMS register is updated each time a new message is successfully received and stored. The contents of the C0RMS register are automatically cleared following each read of C0RMS by the microcontroller. A bit value of 1 indicates that the assigned message center has successfully received and stored new data since the last read of the C0RMS register. A bit value of 0 indicates no new messages have been successfully received and stored since the last read of the C0RMS register. No interrupts are asserted because of the C0RMS settings. This register works fully independent of the status bits in the CAN status register and the INTIN7:INTIN0 vector in the CAN interrupt register and independent of the INTRQ bit in the CAN message control registers.

4.2.4.10 CAN 0 Transmit Message Acknowledgement Register (C0TMA)

Register Description: **CAN 0 Transmit Message Acknowledgement Register**
Register Name: **C0TMA**
Register Address: **Module 04h, Index 09h**

Bit #	15	14	13	12	11	10	9	8
Name	—	C0TMA.15	C0TMA.14	C0TMA.13	C0TMA.12	C0TMA.11	C0TMA.10	C0TMA.9
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	C0TMA.8	C0TMA.7	C0TMA.6	C0TMA.5	C0TMA.4	C0TMA.3	C0TMA.2	C0TMA.1
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Note: *This register is cleared to 0000h on all forms of reset, including the reset established by the CRST bit.*

Bit 15: Reserved. Read 0, write ignored.

Bits 14 to 0: Message Center 15 to 1 Transmit (C0TMA.15 to C0TMA.0). The C0TMA bits indicate which message center (1 to 15) has been successfully transmitted. The contents of the C0TMA register are updated each time a new message is successfully transmitted. The contents of the C0TMA0 register are automatically cleared following each read of C0TMA by the microcontroller. A bit value of 1 indicates that the assigned message center has been successfully transmitted since the last read of the C0TMA register. A bit value of 0 indicates no new message has been successfully transmitted since the last read of the C0TMA register. The corresponding C0TMA bits are assigned to the following message centers. No interrupts are asserted because of the C0TMA settings. This register works fully independent of the status bits in the CAN status register and the INTIN7:INTINT0 vector in the CAN interrupt register, and independent of the INTRQ bit in the CAN message control registers.

MAXQ7665/MAXQ7666 User's Guide

4.2.4.11 CAN 0 Message Center 1 to 15 Control Registers (C0M1C to C0M15C)

Register Description: **CAN 0 Message Center 1 Control Register**
Register Name: **C0M1C**
Register Address: **Module 04h, Index 11h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Bits 15 to 8: Reserved. Read 0, write ignored.

Read/Write Access: MSRDY, ETI, ERI, and INTRQ are unrestricted read/write bits. EXTRQ is read/clear-only. When $T/\overline{R}$ = 0, ROW is read-only; when $T/\overline{R}$ = 1, TIH is unrestricted read/write. MTRQ is unrestricted read and can only be set to 1 when written to by the microcontroller or by the CAN controller in case of a remote frame reception in a transmit message center. A write of 0 to MTRQ leaves the MTRQ bit unchanged. DTUP is unrestricted read. When $T/\overline{R}$ = 0, DTUP can only be cleared to 0 when written by the microcontroller. A write of 1 to DTUP with $T/\overline{R}$ = 0 leaves the DTUP bit unchanged. DTUP is unrestricted read/write when $T/\overline{R}$ = 1.

Bit 7: CAN 0 Message Center 1 Ready (MSRDY). (Unrestricted read/write.) MSRDY is programmed by the microcontroller to notify the CAN 0 logic when the associated message is ready for communication on the CAN 0 bus. When MSRDY = 0, the CAN 0 processor does not access this message center for either transmissions or to receive data or remote frame requests. MSRDY = 1 indicates the message is ready for communication, and MSDRY = 0 indicates that the associated message is either not configured for use or is not required at the present time. This bit is used by the microcontroller to prevent the CAN 0 logic from accessing a message while the microcontroller is updating message attributes. These include identifiers (arbitration registers 0–3), data byte registers 0–7, data byte count (DTBYC3, DTBYC0), direction control ($T/\overline{R}$), the extended or standard mode bit ($EX/\overline{ST}$), and the mask enables (MEME and MDME) associated with this message center. MSRDY is cleared to 0 following a microcontroller hardware reset or a reset generated by the CRST bit in the CAN 0 control register, and must also remain in a cleared mode until all the CAN 0 initialization has been completed. Individual message MSRDY controls can be changed after initialization to reconfigure specific messages, without interrupting the communication of other messages on the CAN 0 bus.

Bit 6: CAN 0 Message Center 1 Enable Transmit Interrupt (ETI). (Unrestricted read/write.) When ETI is cleared to 0, a successful transmission does not set INTRQ and, as such, does not generate an interrupt. Setting ETI to 1 enables a successful CAN 0 transmission to set the INTRQ bit, which in turn issues an interrupt to the microcontroller. Note that the CAN processor ignores the ETI bit located in message center 15, since message center 15 is a receive-only message center.

Bit 5: CAN 0 Message Center 1 Enable Receive Interrupt (ERI). (Unrestricted read/write.) When ERI is cleared to 0, a successful reception does not set INTRQ and, as such, does not generate an interrupt. When ERI is set to 1, the INTRQ bit only sets when the CAN processor successfully receives and stores the incoming message into one of the message centers. Setting INTRQ, in turn, issues an interrupt request to the microcontroller.

Bit 4: Interrupt Request (INTRQ). (Unrestricted read/write.) INTRQ is automatically set to 1 by the CAN 0 logic when the ERI is set and the CAN 0 logic completes a successful reception and store. The INTRQ bit is also set to 1 when the ETI is set and the CAN 0 logic completes a successful transmission. The INTRQ interrupt request must be also enabled via the IGE global mask in the IC peripheral register and the IM4 mask in the IMR peripheral register, if the interrupt is to be acknowledged by the microcontroller interrupt logic. An active message center interrupt is cleared by writing a 0 to the INTRQ bit in the respective CAN message control register.

Bit 3: External Transmit Request (EXTRQ). (Read/clear only.) When EXTRQ is cleared to 0, there are no pending requests by external CAN nodes for this message. When EXTRQ is set to 1, a request has been made for this message by an external CAN node, but the service request has not been completed by the CAN 0 controller at the time of the read of EXTRQ. Following the completion of a requested transmission by a message center programmed for transmission (T/$\overline{R}$ = 1), the EXTRQ bit is cleared by the CAN 0 controller. A remote request is only answered by a message center programmed for transmission (T/$\overline{R}$ = 1) when DTUP = 1 and TIH = 0 (i.e., when new data was loaded and is not being currently modified by the microcontroller). Note that a message center programmed for a receive mode (T/$\overline{R}$ = 0) also detects a remote frame request and sets the EXTRQ bit in a similar manner, but does not automatically transmit a data frame and, as such, does not automatically clear the EXTRQ bit.

Bit 2: Microcontroller Transmit Request (MTRQ). MTRQ is unrestricted read and can only be set to 1 when written to by the microcontroller. A write of 0 to MTRQ leaves the MTRQ bit unchanged. MTRQ can only be cleared as a result of a successful transmission by the respective message center, or when the CRST bit is set or the CAN processor experiences a system reset from the reset sources outlined in the functional description in the reset option and reset timing section of this user's guide.

The MTRQ is a read, limited-write bit, and is designed to allow the microcontroller to request a message to be transmitted. MTRQ is programmed to 1 when the microcontroller is requesting the respective message to be transmitted. MTRQ remains set until such time that the message transmission is successfully completed, at which time the CAN 0 controller clears the MTRQ bit. Setting MTRQ with T/$\overline{R}$ = 1 (directional = transmit) results in the sending of a data frame for the transmitted message, and setting MTRQ with T/$\overline{R}$ = 0 (directional = receive) results in the sending of a remote frame request. When the associated message is programmed for transmit (T/$\overline{R}$ = 1), the MTRQ bit is also set by the CAN 0 controller at the same time that the EXTRQ bit is set by a message request from an external node. MTRQ is cleared by the CAN 0 controller at the same time as the EXTRQ bit, once a successful transmission of the message is completed. Note that the MTRQ bit located in message center 15 is ignored by the CAN processor, since message center 15 is a receive-only message center.

Bit 1: Receive Overwrite/Transmit Inhibit (ROW/TIH). The ROW and TIH bits share the same bit 1 location in the CAN 0 message control register. The ROW function is only supported when the associated message is programmed via the T/$\overline{R}$ = 0 bit in the message format register to function in the receive mode. Similarly, the TIH function is only supported when the associated message is programmed via the T/$\overline{R}$ = 1 bit in the message format register to function in the transmit mode.

> **Receive Overwrite (ROW).** (T/$\overline{R}$ = 0, read-only.) The CAN 0 controller automatically sets the ROW bit to 1 if a new message is received and stored while the DTUP bit was still set. When set, ROW indicates that the previous message was potentially lost and may not have been read, since the microcontroller had not cleared the DTUP bit prior to the new load. When ROW = 0, no new message has been received and stored while DTUP was set to 1 since this bit was last cleared. Note that the ROW bit is not set when the WTOE bit is cleared to 0, since all overwrites are disabled. Thus, if the incoming message matches the respective message center and DTUP = 1 in the respective message center, the combination of WTOE = 0 and DTUP = 1 forces the CAN processor to ignore the respective message center when the CAN is processing the incoming data.

> The CAN processor clears ROW when the microcontroller clears the DTUP bit associated with the same message center. It must be pointed out that the ROW bit for message center 15 is related to the overwrite of the buffer associated with message center 15, as opposed to the actual message center 15. ROW reflects the actual message center relationships for message centers 1–14. The ROW bit for the message center 15 shadow buffer is cleared once the shadow buffer is loaded into message center 15, and the shadow buffer is cleared to allow a new message to be loaded. The shadow buffer is automatically loaded into message center 15 when the microcontroller clears the DTUP and EXTRQ bits in message center 15.

> **Transmit Inhibit (TIH).** (T/$\overline{R}$ = 1, unrestricted read/write.) The TIH bit allows the microcontroller to disable the transmission of the message when the data contents of the message are being updated. TIH = 1 directs the CAN 0 controller not to transmit the associated message. TIH = 0 enables the CAN 0 controller to transmit the message. If TIH = 1, EXTRQ is set to 1 when a remote frame request is received by the message. Following the remote frame request and after the microcontroller has established the proper data to be sent, the microcontroller clears the TIH bit to 0, allowing the CAN processor to send the data requested by the previous remote frame request. Note that the TIH bit located in message center 15 is ignored by the CAN processor, since message center 15 is a receive-only message center.

> If the message center being set up with WTOE = 1 was previously a transmit message center, ensure that the TIH bit is cleared to 0. (TIH can only be written while T/$\overline{R}$ is set to 1.) If TIH is set to 1 and that message center is changed to receive with WTOE = 1, the ROW bit always reads back 1, even though a receive overwrite condition may not have occurred.

Bit 0: Data Updated (DTUP). (Unrestricted read.) When $T/\overline{R}$ = 0, DTUP can only be cleared to 0 when written by the microcontroller. A write of 1 to DTUP with $T/\overline{R}$ = 0 leaves the DTUP bit unchanged. A write of 1 to DTUP with $T/\overline{R}$ = 1 leaves the MTRQ bit unchanged. DTUP is unrestricted read/write when $T/\overline{R}$ = 1.

The DTUP bit has a dual function depending on whether a message is configured for transmit or receive via the $T/\overline{R}$ bit in the CAN 0 message format register. The DTUP bit is set to 1 by either the microcontroller (when in transmit) or by the CAN 0 controller (when in receive) to signify that new data has been loaded into the data portion of the message.

> **Transmission Mode ($T/\overline{R}$ = 1).** The microcontroller sets TIH = 1 and clears DTUP = 0 prior to doing an update of the associated message center. This prevents the CAN processor from transmitting the data while the microcontroller is updating it. Once the microcontroller has finished configuring the message center, the microcontroller clears TIH = 0 and sets MSRDY =1, MTRQ = 1, and DTUP = 1 to enable the CAN processor to transmit the data.

> The CAN processor does not clear the DTUP after the transmission, but the microcontroller can determine that the transmission has been completed by checking the MTRQ bit, which is cleared (MTRQ = 0) after the transmission has been successfully completed.

> **Receive Mode ($T/\overline{R}$ = 0).** The CAN processor sets the DTUP bit when it has completed a successful reception and storage of the incoming message to the respective message center. The CAN processor does not clear the DTUP after the microcontroller has read the associated data. This function is left to the microcontroller.

> When operating in the receive mode ($T/\overline{R}$ = 0), the DTUP = 1 signal notifies the microcontroller that the respective message center has new data to be read by the microcontroller. The DTUP bit is used in two different ways when doing the read of the message center, as determined by the WTOE bit in the CAN 0 message 1 arbitration register 3 (C0M1AR3).

> When WTOE = 1 and the CAN processor is allowed to perform overwrites of respective message centers, the microcontroller uses the DTUP bit to establish the validity of each message read. Clearing DTUP = 0 before a read of a receive message center and then reading the DTUP bit after finishing the message center read, the microcontroller can determine if new data was loaded (DTUP = 1) or not (DTUP = 0) into the message center during the microcontroller read of the message center.

> If DTUP = 1, then there was new data stored to the message center while the microcontroller was performing the message center read. This status condition requires the microcontroller to again clear the DTUP bit and perform a second read of the message center to verify that the data it reads is completely updated.

> If DTUP = 0, the message center data read by the microcontroller had not been updated while it was being read by the microcontroller, and the data is complete.

> When WTOE = 0 and the CAN processor is not allowed to perform overwrites of respective message centers, the microcontroller only needs to clear DTUP = 0 after performing the read of the message center. The CAN processor is not allowed to write into a message center where the DTUP = 1 state exists.

> The DTUP bit is never cleared by the CAN processor, but is set as per the above discussion. The only mechanism used to clear the DTUP bit is by the microcontroller or a system reset or the setting of the CRST bit.

> When $T/\overline{R}$ = 1, all message center transmissions are automatically disabled until both DTUP = 1 and TIH = 0. This mechanism prevents the CAN from sending incomplete data.

> Remote frame transmissions are not affected by the TIH bit in the receive mode ($T/\overline{R}$ = 0), since this function does not exist in this mode. In a similar fashion, the state of the DTUP bit does not inhibit remote frame request transmissions in the receive mode. The only gating item for remote frame transmissions in the receive mode ($T/\overline{R}$ = 0) is the setting of both the MSRDY = 1 and MTRQ = 1 bits.

Note: The CAN 0 message center 2 to 15 control register bits are identical to those found in the CAN 0 message center 1 control register. Refer to these descriptions for the following registers.

Register Description: **CAN 0 Message Center 2 Control Register**
Register Name: **C0M2C**
Register Address: **Module 04h, Index 12h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 3 Control Register**
Register Name: **C0M3C**
Register Address: **Module 04h, Index 13h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

MAXQ7665/MAXQ7666 User's Guide

Register Description: **CAN 0 Message Center 4 Control Register**
Register Name: **C0M4C**
Register Address: **Module 04h, Index 14h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 5 Control Register**
Register Name: **C0M5C**
Register Address: **Module 04h, Index 15h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 6 Control Register**
Register Name: **C0M6C**
Register Address: **Module 04h, Index 16h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 7 Control Register**
Register Name: **C0M7C**
Register Address: **Module 04h, Index 17h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 8 Control Register**
Register Name: **C0M8C**
Register Address: **Module 04h, Index 18h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 9 Control Register**
Register Name: **C0M9C**
Register Address: **Module 04h, Index 19h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 10 Control Register**
Register Name: **C0M10C**
Register Address: **Module 04h, Index 1Ah**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 11 Control Register**
Register Name: **C0M11C**
Register Address: **Module 04h, Index 1Bh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 12 Control Register**
Register Name: **C0M12C**
Register Address: **Module 04h, Index 1Ch**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 13 Control Register**
Register Name: **C0M13C**
Register Address: **Module 04h, Index 1Dh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 14 Control Register**
Register Name: **C0M14C**
Register Address: **Module 04h, Index 1Eh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

Register Description: **CAN 0 Message Center 15 Control Register**
Register Name: **C0M15C**
Register Address: **Module 04h, Index 1Fh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rc	r*	r*	r*

*r = read, w = write, c = clear only, * = see description*

4.3 CAN Operations

The CAN2.0B protocol specifies two different message formats: the standard 11-bit (CAN2.0A) and the extended 29-bit (CAN2.0B), and four different frame types for CAN bus communications. The standard format, as shown in Figure 4-3, makes use of an 11-bit identifier. The extended format, as shown in Figure 4-4, makes use of a 29-bit identifier.
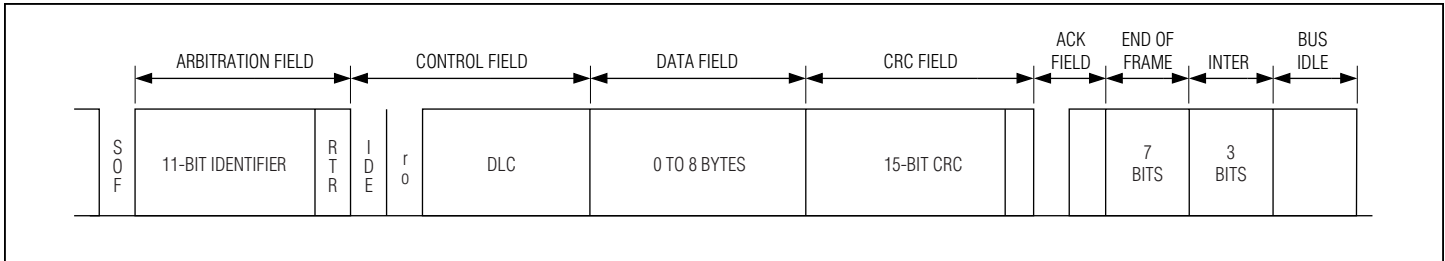
Figure 4-3. CAN2.0A (Standard) Format
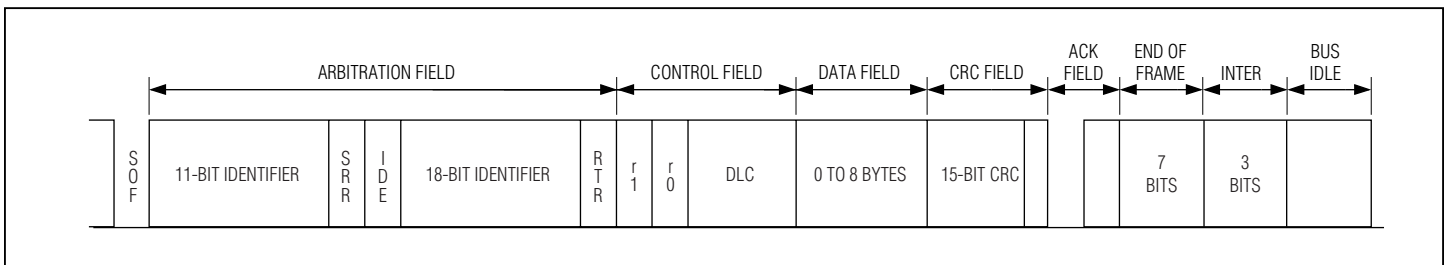
Figure 4-4. CAN2.0B (Extended) Format

4.3.1 Frame Types

The four different frame types for CAN bus communications are data frame, remote frame, error frame, and overload frame.

4.3.1.1 Data Frame

The data frame is formulated to carry data from a transmitter to a receiver. Figure 4-3 and Figure 4-4 show examples of data frames in the standard and extended formats. The data frame is composed of seven fields that include the start of frame, arbitration field, control field, data field, CRC field, acknowledge field, and an end of frame. A description of these fields follows.

4.3.1.1.1 Start of Frame (SOF)

(Standard and extended format.) The start of frame (SOF) is a dominant bit that signals the start of a data or remote frame. The dominant bit forces a hard synchronization, initiating the CAN controller receive mode.

4.3.1.1.2 Arbitration Field

(Standard and extended format.) The arbitration field contains the identifier of the message and a dominant remote request (RTR) bit. The identifier is composed of one field in the standard 11-bit format or two fields in the extended 29-bit format. Two additional bits, the substitution remote request (SRR) bit and the identifier extension (IDE) bit, separate the two fields in the extended format.

- **Remote Request (RTR) Bit:** (Standard and extended format.) The remote request (RTR) bit is a dominant bit in data frames and a recessive bit in remote frames.

- **Substitution Remote Request (SRR) Bit:** (Extended format.) The substitution remote request (SRR) bit is a recessive bit and is substituted for the RTR bit when using the extended format.

- **Identifier Extension (IDE) Bit:** (Extended format.) The identifier extension (IDE) bit is a dominant bit in the standard format and a recessive bit in the extended format. The IDE bit is located in the control field in the standard format and is located in the arbitration field in the extended format.

4.3.1.1.3 Control Field

(Standard and extended format.) The control field is composed of six bits in two fields. The first field is made up of two reserved bits that are transmitted as dominant bits (Figure 4-5). The second field contains four bits that comprise the data length code (DLC). The DLC determines the number of data bytes in the data field of the data frame, and is programmed through the use of the CAN message format registers, located in each of the 15 message centers.

4.3.1.1.4 Data Field

(Standard and extended format.) The data field is composed of 0 to 8 bytes in a data frame and 0 bytes in a remote frame. The number of data bytes associated with a message center is programmed through the use of the CAN message format registers, located in each of the 15 message centers. The data field contents are saved to the respective message center if the identifier test is successful, no errors are detected through the last bit of the end of frame, and an error frame does not immediately follow the data or remote frame. The data field is transmitted least significant byte first, with the most significant bit of each byte transmitted first.

4.3.1.1.5 CRC Field

(Standard and extended format.) The CRC field is composed of a 15-bit code that is the computed cyclic redundancy check (after destuffing bits) from the start of frame, through the arbitration, control, and data fields (when present), and a CRC delimiter (Figure 4-6). The CRC calculation is limited to a 127-bit maximum code word (a shortened BCH code) with a CRC sequence length of 15 bits.

4.3.1.1.6 Acknowledge (ACK) Field

(Standard and extended format). The ACK field is composed of two bits (Figure 4-7). The transmitting node sends two recessive bits in the ACK field. The receiving nodes that have received the message and found the CRC sequence to be correct reply by driving the ACK slot with a dominant bit. The ACK delimiter is always a recessive bit.

4.3.1.1.7 End of Frame

(Standard and extended format). The end of frame for both the data and remote frame is established by the transmitter sending seven recessive bits.

4.3.1.1.8 Interframe Spacing (Intermission)

(Standard and extended format). Data frames and remote frames are separated from preceding frames by three recessive bits termed the intermission (Figure 4-8). During the intermission, the only allowed signaling to the bus is by an overload condition. No node is allowed to start a message transmission of a data or remote frame during this period. If no node becomes active following the interframe space, an indeterminate number of recessive bit times transpires in the bus-idle condition until the next transmission of a new data or remote frame by a node.
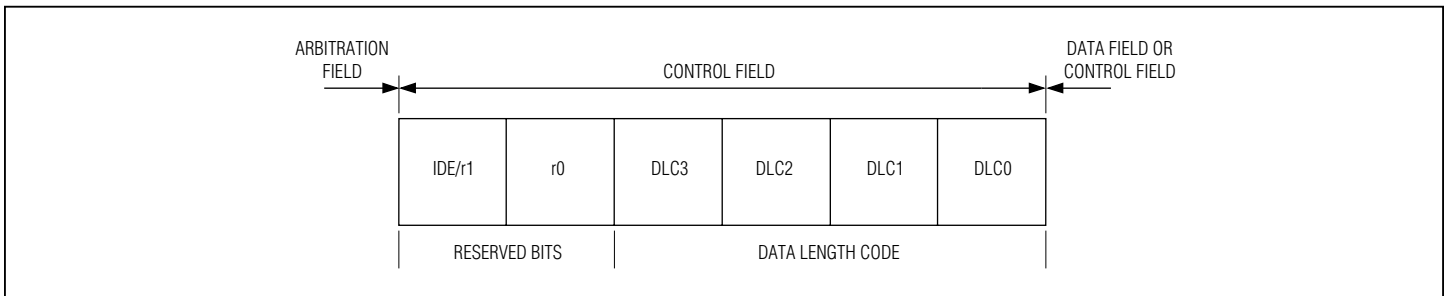
Figure 4-5. Control Field
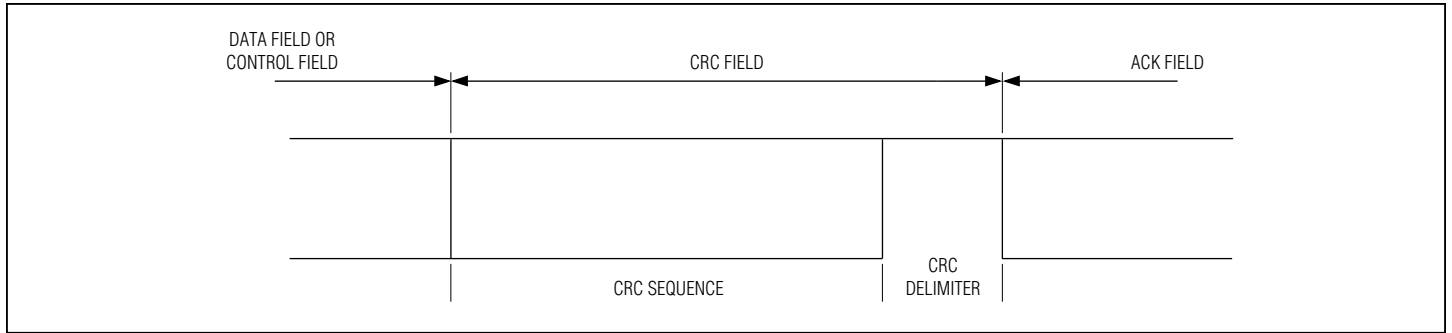
Figure 4-6. CRC Field
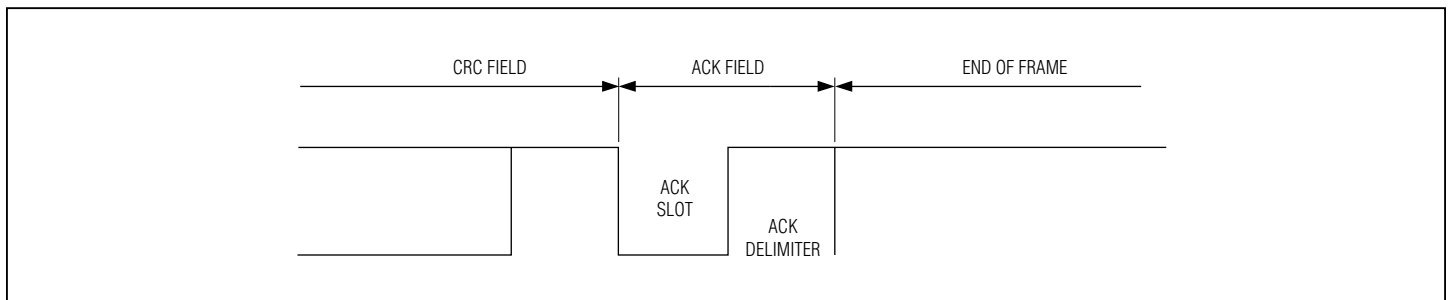
Figure 4-7. Acknowledge Field
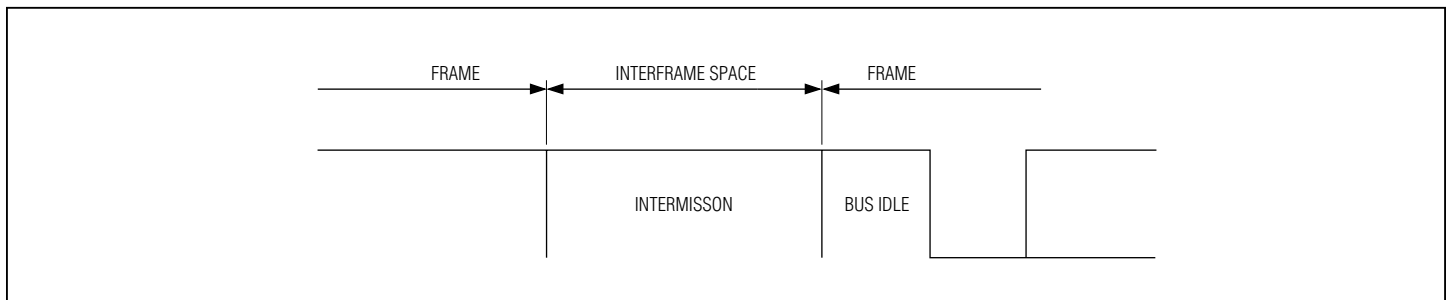
Figure 4-8. Intermission

4.3.1.2 Remote Frame

(Standard and extended format.) The remote frame is transmitted by a CAN controller to request the transmission of the data frame with the same identifier (Figure 4-9). The remote frame is composed of seven fields, which include start of frame, arbitration field, control field, data field, CRC field, acknowledge field, and end of frame.

The remote frame is used when a CAN processor wishes to request data from another node. Sending a remote frame initiates a transmission of data from a source node with the same identifier (masked groups included). The primary bit pattern difference between a data frame and a remote frame is the RTR bit. In the remote frame, the RTR bit is sent as a recessive bit; in the data frame, the RTR bit is sent as a dominant bit. Additionally, the remote frame does not contain a data field, independent of the programmed values in the DTBYC3:DTBYC0 bits in the respective CAN message format register.
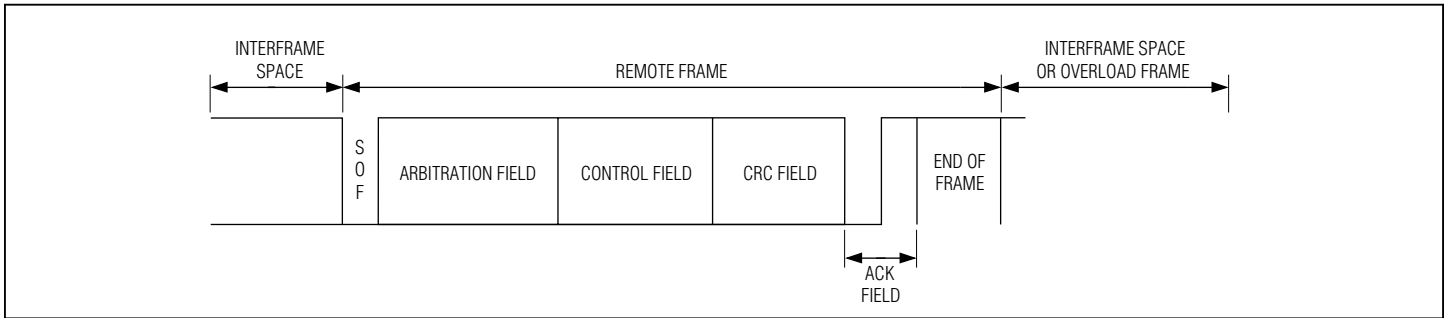
Figure 4-9. Remote Frame
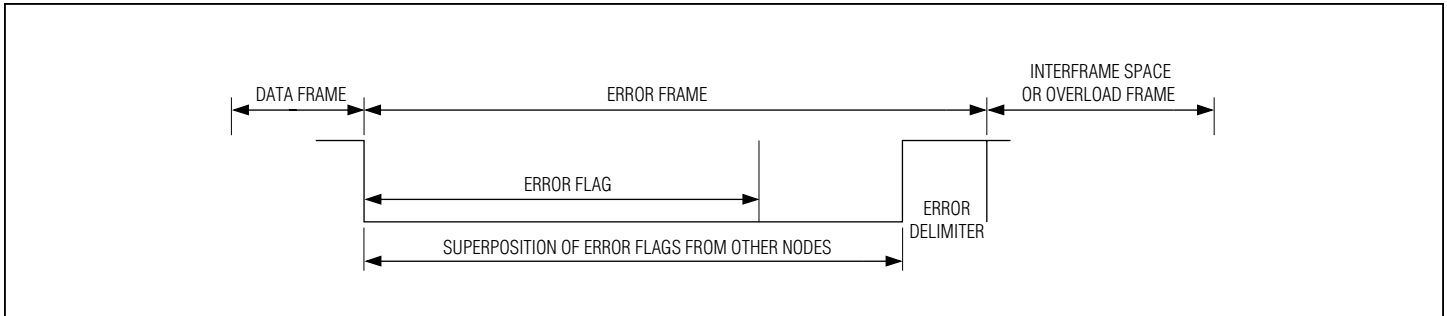
Figure 4-10. Error Frame
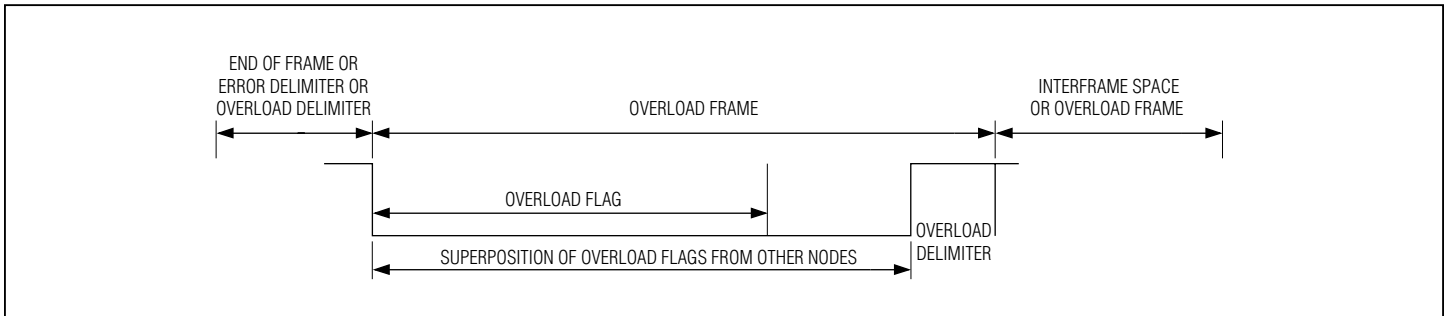
Figure 4-11. Overload Frame

4.3.1.3 Error Frame

The error frame is transmitted by a CAN controller when the CAN processor detects a bus error. The error frame is composed of two different fields: the superposition of the error flags from different nodes and the error delimiter.

The error frame is composed of six dominant bits that violate the CAN specification bit stuffing rule. If either of the CAN processors detects an error condition, that CAN processor transmits an error frame. When this happens, all nodes on the bus detect the bit stuff error condition and transmit their own error frame. The superpositioning of all these error frames leads to a total error frame length between 6 and 12 bits, depending on the response time and number of nodes in the system. Any messages (data or remote frame) received by the CAN processors (successful or not) that are followed by an error frame are discarded. After the transmission of an error flag, each CAN processor sends an error delimiter (eight recessive bits) and monitors the bus until it detects the change from the dominant to recessive bit level. The CAN modules issue an error frame each time an error frame is detected. Following a series of error frames, the CAN modules enter into an error passive mode. In the error passive mode, the CAN processors transmit six recessive bits and wait until six equal bits of the same polarity have been detected. At this point, the CAN processor begins the next internal receive or transmission operation.

4.3.1.4 Overload Frame

The overload frame provides an extra delay between data or remote frames. The overload frame is composed of two different fields: the overload flag and the overload delimiter.

There are three conditions that lead to the transmission of an overload flag:

1) The internal conditions of a CAN receiver require a delay before the next data or remote frame is sent. The MAXQ7665/ MAXQ7666 CAN controller is designed to prevent this condition for data rates at or below the 1Mbps data rate.

2) The CAN processor detects a dominant bit at the first and second bit position of the intermission.

3) If the CAN processor detects a dominant bit at the 8th bit of an error delimiter or overload delimiter, it starts transmitting an overload frame.

The error counters are not incremented as a result of number 3. The CAN processor only starts an overload frame at the first bit of an expected Intermission if initiated by condition 1. Conditions 2 and 3 result in the CAN processor transmitting an overload frame, starting one bit after detecting the dominant bit. The overload flag consists of six dominant bits that correspond to an error flag. Because the overload frame is only transmitted at the first bit time of the interframe space, it is possible for the CAN processor to discriminate between an error frame and an overload frame. The overload flag destroys the intermission field. When such a condition is detected, the CAN processor detects the overload condition and begins transmitting an overload frame. After the transmission of an overload frame, the CAN processors monitor the bus for a dominant to recessive level change. The CAN processor then begins the transmission of six additional recessive bits, for a total of seven recessive bits on the bus. The overload delimiter consists of eight recessive bits.

4.4 General CAN Protocol-Related Issues

4.4.1 Bit Stuffing

The CAN processor performs a function termed bit stuffing in accordance with the CAN2.0 protocol. The bit stuffing is a mechanism that is done on both the transmitting and receiving end of the transmission. When the CAN processor detects (in transmit or receive mode) five consecutive bits of identical polarity, the CAN processor inserts (when transmitting) or removes (when receiving) a compliment bit from the data stream. The bit stuffing is only used within the start of frame, arbitration field, control field, data field, and CRC sequence. All other fields are unaffected. The bit stuffing in the CAN specification provides the required changes in the bus to allow all nodes in the system to maintain synchronization.

4.4.2 Simultaneous Transmissions

The CAN processor monitors its own transmission and performs test on the outgoing data through the receive inputs. This is done to verify that the message being sent is not in conflict with another node on the bus that may also be transmitting at the same time. If the CAN processor detects that a transmitted recessive bit has been overwritten to the bus by a dominant bit from another node, the CAN processor stops the transmission and waits until the next available time slot to try and retransmit the data or remote frame. This allows the node with the higher priority to dominate the bus, without the possibility of a collision that would destroy data. The assignment of unique identifiers to each message center within the CAN processor also establishes the natural priority of each message center within the CAN processor, as well as when the message center is transmitted to the bus. Since the MSB of the identifier is transmitted first, the message with the highest value quickly establishes the priority of which CAN unit is allowed to continue to use the bus during a simultaneous transmission time segment. To eliminate possible problems with identical identifier, it is best that all nodes on the system use unique identifiers. The issue of simultaneous transmission of a data frame and a remote frame with the same identifier is handled through the use of the RTR bit. The RTR bit establishes the data frame as the higher priority to guarantee that the previously requested data frame takes precedence over the newer remote frame request.

4.4.3 Transmit- and Receive-Error Counters

The CAN processor contains an 8-bit transmit-error counter and a second 8-bit receive-error counter. The CAN processor monitors both its own transmissions as well as those of other nodes. Whenever an error condition is detected, the appropriate error counter is incremented by a given value associated with the type of error detected. The MAXQ7665/MAXQ7666 CAN controller meets all the standard error-logging conditions outlined in the CAN2.0B specification (Part B, Sept. 1991) under the heading of *Fault Confinement*. Both transmit- and receive-error counters can be read by the microcontroller at any time. During software initialization (SWINT = 1), the error counters can be written to establish a common value in both registers via a write capability supplied through the CAN transmit-error peripheral register.

4.5 External Pins

The CAN controller uses two external signals, CANRXD and CANTXD, that are on dedicated pins. CANRXD is receive data, a digital input that connects to a CAN transceiver output. CANTXD is transmit data, a digital output that connects to a CAN transceiver input. The CANRXD and CANTXD signals are CAN2.0B-interface compliant with the logic level 0 (low voltage) representing a dominant state, and logic level 1 (high voltage) representing a recessive state. The MAXQ7665/MAXQ7666 CAN controller pins are summarized in Table 4-1.

Table 4-1. MAXQ7665/MAXQ7666 CAN Controller Pins

CAN EXTERNAL SIGNAL	PIN		DESCRIPTION
	48-PIN	56-PIN	
CANRXD	20	22	CAN Bus Receiver Input
CANTXD	21	24	CAN Bus Transmitter Output

4.6 Initializing the CAN Controller

Software initialization of the CAN controller begins with the setting of the software initialization bit (SWINT) in the CAN 0 control peripheral register. When SWINT = 1, the CAN module is disabled and the CAN transmit output (CANTXD) is placed in a recessive state. This, in turn, allows the microcontroller to write information into the CAN MOVX SRAM control/status/mask registers without the possibility of corrupting data transmissions or receptions in progress. Setting SWINT does not clear the receive- and transmit-error counters, but allows the microcontroller to write a common value to both error counters through the CAN 0 transmit-error peripheral register. See the description of the SWINT bit for specifics of the software initialization process.

All CAN registers located in the peripheral register map, with the exception of the CAN 0 control register, are cleared to 00h following a system reset. The CAN 0 control register is set to 09h following a system reset. CAN registers located in the dual port memory map are indeterminate following a system reset. A system reset also clears both the receive- and transmit-error counters in the CAN controller, takes the CAN processor offline, and sets the SWINT bit in the CAN 0 control register.

Following a reset, the CAN-related registers in Table 4-2 must be initialized for proper operation of the CAN module. These registers are in addition to specific registers associated with mask, format, or specific message centers.

Table 4-2. Registers to Be Initialized for Proper CAN Module Operation

REGISTER	SIGNIFICANCE
C0BT0, C0BT1 (Dual Port Address 02h(L), 02h(H))	These dual port control registers must be set to configure CAN 0 bus timing. The exact values are dependent on the network configuration and environment.
COR (Module 4, Index 5)	C0BPR7, C0BPR6 (COR.4, COR3) must be configured as part of the CAN 0 bus timing.

4.7 CAN Interrupts

The CAN processor is assigned an interrupt that is individually enabled via the C0IE bit in the EIE register and globally enabled/disabled by the IM4 bit in the IMR register and the IGE bit in the IC peripheral register. A CAN 0 interrupt can be generated by either a receive/transmit acknowledgment from one of the 15 message centers or by a change in the CAN 0 status register.

CAN 0 transmit/receive interrupt sources are derived from a successful transmit or receive of data within one of the 15 message centers as signaled by the INTRQ bit in the associated CAN 0 message (1–15) control register. Each message center (1–15) provides a separate receive interrupt enable (ERI) and transmit interrupt enable (ETI) bits in the respective CAN 0 message (1–15) control register to allow setting of the INTRQ bit in response to successful transmission or reception. The CAN 0 interrupt register (C0IR) can then be used to determine which message center generated the interrupt request. Software must clear the respective INTRQ bit in the associated CAN 0 message (1–15) control register to clear the interrupt source before leaving the interrupt routine.

The CAN 0 interrupt source can also be connected to a change in the CAN 0 status register. Each of the bits in the CAN 0 status register represents a potential source for the interrupt. To simplify the application and testing of a device, these sources are broken into two groups that, for interrupt purposes, are enabled separately by the ERIE and STIE bit of the CAN 0 control (C0C) register. This allows the nonstandard errors typically associated with development to be grouped under the STIE enable. These include the successful receive RXS, successful transmit TXS, wake status WKS, and general set of error conditions reported by ER2:ER0. Also note that since the RXS and TXS bit are cleared by software, if a second message is received or transmitted before the RXS or TXS bits are cleared and after a read of the CAN 0 status register, a second interrupt is generated. The remaining error sources comprise the BSS and EC96/128 bits in the CAN 0 status register. These read-only bits are separately enabled via the ERIE bit in the CAN 0 control register. A read of the CAN 0 status register is required to clear either of the two groups of error interrupts. It is possible that multiple changes to the status register can occur before the register is read. In that case, the status register generates only one interrupt. Figure 4-12 provides a graphical illustration of the interrupt sources and their respective interrupt enables.
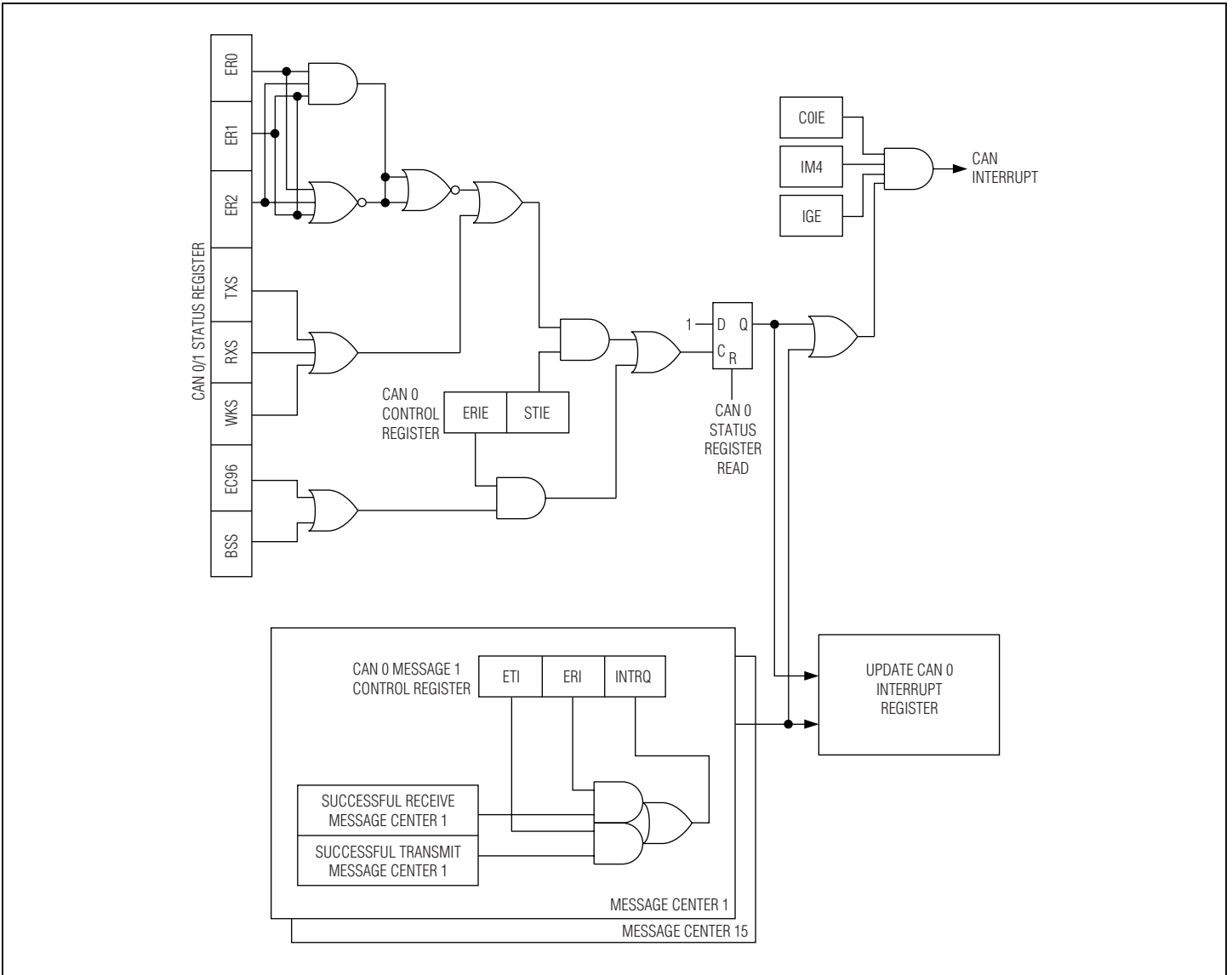
Figure 4-12. CAN Interrupt Logic

4.8 Arbitration/Masking Considerations

The CAN processor is designed to evaluate and determine if an incoming message is loaded into one of the 15 message centers. Acceptance of a message is determined by comparing the message's ID and/or data field against the corresponding arbitration information defined for each message center. Messages that contain bit errors or which fail arbitration are discarded. The incoming message is tested in order against each enabled message center (enabled by the MSRDY bit in the CAN message control register) from 1 to 15. The first message center to successfully pass the test receives the incoming message and ends the testing, and the message is loaded into the respective message center.

The MAXQ7665/MAXQ7666 CAN module supports two types of arbitration: basic and media. Basic arbitration compares either the 29-bit (EX/$\overline{ST}$ = 1) or 11-bit (EX/$\overline{ST}$ = 0) incoming message ID against the corresponding bits in the message center CAN arbitration registers (C0MxAR0:C0MxAR3). This depends upon whether the message center has been configured for 29- or 11-bit operation. An optional masking feature can also be used with basic arbitration. The format register (C0MxF) for each message center contains a message identification mask enable bit (MEME). If MEME is set, the CAN module factors in the standard global mask registers (C0SGM0:C0SGM1) when EX/$\overline{ST}$ = 0 or the extended global mask registers (C0EGM0:C0EGM3) when EX/$\overline{ST}$ = 1 when deciding if there is an ID match. A comparison between bits in the incoming message ID and arbitration register bits is only made for bit positions that correspond to 1 in the appropriate mask register. Bits corresponding to 0 in the mask register are ignored, creating a don't care condition. Filling the mask register with all 0s while MEME = 1 causes the arbitration circuitry to automatically match all message IDs. Filling the mask register with all 1s while MEME = 1 requires an exact match between the incoming message ID and the arbitration registers, just as is the case when MEME = 0.

Media arbitration is an optional second arbitration performed if the media identification mask enable bit (MDME) is set in the C0MxF message center register. Media arbitration compares the first and second byte of the data field in each incoming message against two 8-bit media arbitration bytes (stored at locations C0MA0, C0MA1). Each media arbitration byte has an associated media identification mask: C0MID0 for C0MA0 and C0MID1 for C0MA1. Media byte comparison is made only for those bits corresponding to 1 in the media identification mask registers. When MDME = 1, the first two data bytes of the incoming message must pass media byte arbitration as defined by C0MA0:C0MA1 and C0MID0:C0MID1 before being loaded into the respective message center. Unlike the identification mask enable (MEME), however, when MDME = 0 no testing is performed of the first two bytes of the incoming data field.

4.8.1 Message Center 15

Message center 15 supports an additional set of masks to supplement basic arbitration. While this message center performs basic and media arbitration as per message centers 1–14, it also uses the C015M3:C015M0 mask registers to perform an additional level of filtering during basic (i.e., not media) arbitration. When determining arbitration for message center 15, the contents of C015M3:C015M0 are logically ANDed with either C0EGM3:C0EGM0 (if EX/$\overline{ST}$ = 1 for message center 15) or C0SGM1:C0SGM0 (if EX/$\overline{ST}$ = 0 for message center 15). This ANDed value is then used in place of C0EGM3:C0EGM0 or C0SGM1:C0SGM0 when performing basic arbitration as previously described. If the MDME bit is set, the incoming message must also pass the media arbitration test.

Message center 15 has a buffered FIFO arrangement to allow up to two received messages to be received without being lost prior to the microcontroller reading of the first message. The first message received by message center 15 is stored in the normal dual port memory location for message center 15, if the previous message has been already read by the microcontroller. If the first message has not been read, then the incoming message is buffered internally until the first message is read, at which time the second message is automatically loaded into the first (dual port) message 15 slot, allowing software to then read the second message. The CAN module determines that the first message has been read (allowing the buffered message to be transferred) when software clears the DTUP and EXTRQ bits. If a third message arrives before the second message has been copied into the dual port message 15 slot, then the third message writes over the second buffered message. Software should clear the INTRQ bit as well as the DTUP and EXTRQ bits after reading each message in the dual port message center 15. The WTOE bit associated with message center 15 has unique operating considerations, described in this section.

4.9 Transmitting and Receiving Messages

All CAN data is sent and received through message centers. All CAN message centers are identical with the exception of message center 15. Message center 15 has been designed as a receive-only center and is shadow buffered to help prevent the loss of incoming messages when software is unable to read one message before the next one should be loaded. All message centers, with the exception of message center 15, are capable of four different operations, which include the following.

- Transmitting a data message
- Receiving a data message
- Transmitting a remote frame request
- Receiving a remote frame request

4.9.1 Transmitting Data Messages

Starting with the lowest numbered message center (highest priority), the CAN module sequentially scans each message center until it finds a message center that is properly enabled for transmission (T/$\overline{R}$ = 1, TIH = 0, DTUP = 1, MSRDY = 1, and MTRQ = 1). The contents of the respective message center are then transferred to the transmit buffer, and the CAN module attempts to transmit the message. If successful, the appropriate MTRQ bit is cleared to 0, indicating that the message was successfully sent. Following a successful transmission, loss of arbitration, or an error condition, the CAN module again searches for a properly configured message center, starting with the lowest numbered message center. This search relationship always allows the highest priority message center to be transmitted, independent of the last successful (MTRQ = 0) or unsuccessful (MTRQ = 1) message transmission.

4.9.2 Receiving Data Messages

Each incoming data message is compared sequentially with each receive enabled (T/$\overline{R}$ = 0) message center starting with the lowest numbered message center (highest priority) and proceeding to the highest numbered message center. This testing continues until a match is found (incorporating masking functions as required), at which time the incoming message is stored in the respective message center. Higher numbered message centers that are not reviewed prior to the match are not evaluated during the current message test. When the WTOE = 1, the CAN module can overwrite receive message centers that have DTUP = 1, which, in turn, set ROW = 1. When WTOE = 0, incoming messages do not overwrite receive message centers that have DTUP = 1.

Message center 15 is a special receive-only, FIFO-buffered message center designed to receive messages not accepted by the other message centers. The ROW bit in message center 15 is associated with the overwrite of the shadow buffer for message center 15. The EXTRQ and DTUP bits are shadow buffered to allow the buffered message and the message center 15 values to take on different relationships. The EXTRQ and DTUP values read by the microcontroller are not those of the shadow buffer, as is the case with the ROW bit, but are the current values associated with message center 15. The shadow buffer is automatically loaded into message center 15 when both the DTUP bit and the EXTRQ bit are cleared. If either DTUP or EXTRQ are left set when clearing the other, any message in the shadow buffer is not transferred to the message 15 registers, and any incoming messages for message 15 are stored in the shadow buffer (if WTOE = 1) or are lost (if WTOE = 0).

4.9.3 Transmitting Remote Frame Requests

Starting with the lowest numbered message center (highest priority), the CAN module sequentially scans each message center. When it finds a message center properly enabled to transmit a remote frame (T/$\overline{R}$ = 0, MSRDY = 1, and MTRQ = 1), the contents of the respective message center is then transferred to the transmit buffer and the CAN module attempts to transmit the message. If successful, the appropriate MTRQ bit is cleared to 0, indicating that the message was successfully sent. Following a successful transmission, loss of arbitration, or an error condition, the CAN module again searches for a properly configured message center, starting with the lowest numbered message center. This search relationship always allows the highest priority message center to be transmitted, independent of the last successful (MTRQ = 0) or unsuccessful (MTRQ = 1) message transmission. The state of the TIH bit does not effect the transmission of a remote frame request.

If the microcontroller wants to request data from another node, it first clears the respective MSRDY bit to 0 and then writes the identifier and control bits in this message center, configures the message center as a receive message center (T/$\overline{R}$ = 0), and then sets the MTRQ and MSRDY bits. After a successful transmission, the CAN module clears MTRQ = 0 and sets TXS = 1. In addition to the TXS bit, if the ETI bit is set, the successful transmission also sets the corresponding INTRQ bit. Requesting data from another node is possible in message centers 1–14. As seen above, the CAN module sends a remote frame request and receives the data frame in any other mailbox for which the answering incoming data frame passes the acceptance filtering of identifier and first two data bytes. Therefore, only one mailbox is necessary to do a remote request. Remote frame requests are not supported during autobaud mode.

4.9.4 Receiving/Responding to Remote Frame Requests

The remote frame request is handled like a data frame with data length zero and the EXTRQ and RXS bits are set. Each incoming remote frame request (RFR) message is compared sequentially with each enabled (MSRDY = 1) message center starting with the lowest numbered message center (highest priority) and proceeding to the highest numbered message center. Testing continues until a match is found (incorporating masking functions as required), at which time the incoming RFR message is stored in the respective message center, the DTBYC bits are updated to indicate the requested number of return bytes, and EXTRQ and MTRQ are both set to 1. When the message is successfully received and stored, an interrupt of the corresponding message center is asserted, if enabled by the ERI bit. The EXTRQ bit can be left set if the message center is reconfigured to perform a transmit (T/$\overline{R}$ = 1) and used in the standard reply of a remote frame operating with transmit message centers. EXTRQ can also be cleared by software if the current message center is not being used to reply to the remote frame request. Higher numbered message centers (lower priority) that are not reviewed prior to the match are not evaluated during the current message test. Depending on the state of the transmit/receive bit for that message center, the CAN module performs one of two responses.

When a remote frame request is successfully received, the message centers enabled for transmission (T/$\overline{R}$ = 1) set the EXTRQ and MTRQ bits in the corresponding message center to mark the message as a "to be sent" message. The CAN module attempts to automatically transmit the requested message if the message center is fully enabled to do so (MSRDY = 1, TIH = 0, DTUP = 1). After the transmission, the TXS bit in the status register is set, the EXTRQ and MTRQ bits are reset to 0, and a message center interrupt of the corresponding message center is asserted, if enabled by the respective ETI bit. If the transmit inhibit bit is set (TIH = 1), the message center receives the RFR, modifying the DTBYC and/or arbitration bits if necessary, but the return data is not transmitted until TIH = 0.

If software wants to modify the data in a message center configured for transmission of an answer to a remote request (EXTRQ set to 1), the microcontroller must set TIH = 1 and DTUP = 0. The microcontroller can then access the data byte registers 0–7, data byte count (DTBYC3:DTBYC0), the extended or standard mode bit (EX/$\overline{ST}$), and the mask enables (MEME and MDME) of the message center to load the required settings. Following the setup, the software should reset TIH to 0 and set DTUP to a 1 bit to signal the CAN that the access is finished. Until DTUP = 1 and TIH = 0, the transmission of this mailbox is not permitted. Thus, the CAN transmits the newest data and resets EXTRQ = 0 after the transmission is complete. The message center must first be disabled to change the identifier or the direction control (T/$\overline{R}$).

Message centers enabled for reception (T/$\overline{R}$ = 0) do not automatically transmit the requested data. The remote frame request does, however, continue to store the requested number of return bytes in DTBYC and set EXTRQ = 1. No data bytes are received or stored from a remote frame request. The message center can then be configured by software to function as transmitter (T/$\overline{R}$ = 1) and transmit the requested data, or the microcontroller can configure another message center in a transmit mode (T/$\overline{R}$ = 1) to send the requested data. Note that, when T/$\overline{R}$ = 0, the MTRQ bit is not set upon loading of a matching remote frame request.

When a remote frame is received, the CAN module can be configured to either automatically transmit data back to the remote node or to allow the microcontroller to intervene and establish the conditions for the transmission of the return message. The following examples outline various options to respond to remote frame requests.

Case 1: Automatic Reply

CAN controller receives a remote frame request (RFR) and automatically transmits data without additional software intervention.

1) Software sets T/$\overline{R}$ = 1, MSRDY = 0, DTUP = 0, and TIH = 1.
2) Software loads data into respective message center.
3) Software sets MSRDY = 1, DTUP = 1, and TIH = 0 in same instruction. **Note:** Software does not change MTRQ = 0 from previously completed transmission.
4) CAN does not transmit data (MTRQ = 0), but waits for RFR.
5) CAN successfully receives RFR.
6) CAN forces MTRQ = 1 and EXTRQ = 1
7) CAN loads DTBYC from RFR and ID into arbitration registers.
8) CAN automatically transmits data in respective message center.
9) CAN clears EXTRQ = 0 and MTRQ = 0.

Case 2: Software-Initiated Reply

(Using TIH as gating control.) CAN module wants to receive an RFR and wait for software to determine when and what is transmitted in reference to RFR.

1) Software sets $T/\overline{R}$ = 1, MSRDY = 0, DTUP = 0, and TIH = 1.

2) Software loads data into respective message center.

3) Software sets MSRDY = 1, DTUP = 1, and TIH = 1 in same instruction. **Note:** Software does not change MTRQ = 0 from previously completed transmission.

4) CAN does not transmit data (MTRQ = 0), but waits for RFR.

5) CAN successfully receives RFR.

6) CAN forces MTRQ = 1 and EXTRQ = 1.

7) CAN loads DTBYC from RFR and ID into arbitration registers.

8) CAN waits for software to read message center and determine the fact that EXTRQ = 1.

9) Software may load data into message center (or it may already have the data established).

10) Software writes MSRDY = 1, DTUP = 1, and TIH = 0 in same instruction.

11) CAN automatically transmits data (as per RFR DTBYC) in respective message center.

12) CAN clears EXTRQ = 0 and MTRQ = 0.

Case 3: Software-Initiated Reply

(Reply through same message center, using TIH as gating control.) CAN module wants to receive an RFR in a receive-configured ($T/\overline{R}$ = 0) message center. When the data is received, the message center is reconfigured to send data back to the remote request node. This relationship is not possible for message center 15.

1) Software sets $T/\overline{R}$ = 0, MSRDY = 1, and DTUP = 0 and awaits either data frame or RFR. **Note:** Software does not change MTRQ = 0 from previously completed transmission.

2) CAN successfully receives RFR.

3) CAN forces EXTRQ = 1 and DTUP = 1.

4) MTRQ cannot be written to 1 by the CAN when $T/\overline{R}$ = 0 and is left as MTRQ = 0.

5) CAN loads DTBYC from RFR and ID into arbitration registers.

6) CAN waits for software to read message center and determine the fact that EXTRQ = 1.

7) Software disables message center and converts message center into transmit message center.

   a. Software clears MSRDY = 0 to disable message center. Software leaves EXTRQ = 1.

   b. Software then forces message center to transmit mode, $T/\overline{R}$ = 1.

8) Software writes MSRDY = 0, DTUP = 0, and TIH = 1 in preparation to load data.

9) Software loads data into message center.

10) Software writes MSRDY = 1, MTRQ = 1, DTUP = 1, and TIH = 0 in same instruction. **Note:** Software leaves EXTRQ = 1.1

11) CAN automatically transmits data (as per RFR DTBYC) in respective message center.

12) CAN clears EXTRQ = 0 and MTRQ = 0.

Case 4: Software-Initiated Reply

(Reply through different message center, using TIH as gating control.) CAN controller wants to receive an RFR in a message center (denoted MC1) configured to receive data (T/$\overline{R}$ = 0) and to wait for software to select another message center (denoted MC2) to send data back to remote request node.

1) Software sets T/$\overline{R}$ = 0, MSRDY = 1, and DTUP = 0 in MC1 and awaits either data frame or RFR. **Note:** Software does not change MTRQ = 0 in MC1 from previously completed transmission.

2) CAN successfully receives RFR in MC1.

3) CAN forces EXTRQ = 1 and DTUP = 1 in MC1. MTRQ cannot be written to 1 by the CAN when T/$\overline{R}$ = 0 and is left as MTRQ = 0.

4) CAN loads DTBYC from RFR and ID into arbitration registers in MC1.

5) CAN waits for software to read message center and determine the fact that EXTRQ = 1.

6) Software disables MC1 to transfer information to MC2.

    a. Software clears MSRDY = 0 to disable MC1. Software leaves EXTRQ = 1.

    b. Software clears MSRDY = 0 in MC2.

7) Software forces MC2 to transmit mode T/$\overline{R}$ = 1.

8) Software loads ID and DTBYC values from MC1 into ID and DTBYC values in MC2.

9) Software writes MSRDY = 0, DTUP = 0, and TIH = 1 in MC2 in preparation to load data to MC2.

10) Software loads data into MC2.

11) Software writes MSRDY = 1, MTRQ = 1, EXTRQ = 0, DTUP = 1, and TIH = 0 in MC2 in same instruction. Note that CAN has not set EXTRQ in MC2, and is not required to be set for transmission of data from MC2.

12) CAN automatically transmits data (as per RFR requested DTBYC) in MC2.

13) CAN clears MTRQ = 0 (leaving previous EXTRQ = 0 cleared).

14) Software sets T/$\overline{R}$ = 0, MSRDY = 1, EXTRQ = 0, and DTUP = 0 in MC1 and awaits either next RFR or data frame. Note that MTRQ is still cleared in MC1, since MC1 has not been set to a transmit mode.

4.10 Remote Frame Handling in Relation to the DTBYC Bits

The DTBYC bits function slightly differently when remote frames are used. The data length code currently programmed in the message center is overwritten by the data length code field of the incoming remote request frame, so that the requested number of data bytes can be sent in response to the remote request. The following example demonstrates how the DTBYC bits are modified by a received remote frame request.

1) Assume the microcontroller has programmed the following into a message center:

    DTBYC = 5, data field = 75 AF 43 2E 12 78 90 00

    (Note that only the first through fifth data bytes are recognized because DTBYC = 5.)

2) When the CAN module successfully receives a remote frame with the following data:

    Identifier = ID, DTBYC = 2, RTR = 1

3) The incoming message overwrites the identifier and the data length code. The new data in the message center is: DTBYC = 2, data field = 75 AF 43 2E 12 78 90 00

    (Note that only the first and second data bytes are recognized because DTBYC is now 2.)

4) The outgoing response is a data frame containing the following information:

    DTBYC = 2, data field = 75 AF

Important Information Concerning ID Changes When Awaiting Data from a Previous Remote Frame Request

The use of acceptance filtering (MEME = 1) in conjunction with remote frame requests can result in a modification of the message center arbitration registers. Suppose, for example, that a message center is configured to transmit a remote frame request (MTRQ = 1, EXTRQ = 0, T/$\overline{R}$ = 0, and MSRDY = 1). If arbitration masks are used, it is possible for a second frame request from an external node to modify this requesting node's arbitration register value prior to reception of the previously requested data. When a remote frame request is received, the message ID is loaded into that message center's arbitration registers. When message identification masking is not used (MEME = 0), the message ID always matches the arbitration value, so the process is transparent. If masking is used, however, the message ID ANDed with the appropriate mask is loaded into that message center's arbitration registers, resulting in a change of the arbitration values for that message center. To prevent this situation, acceptance filtering should be disabled (MEME = 0) for any message center configured for remote frame handling.

4.11 Overwrite Enable/Disable Feature

The writeover enable bit (WTOE) located in each message center (C0MxAR3) enables or disables the overwriting of unread messages in message centers 1–15. Programming WTOE = 1 following a system reset or CRST bit-enabled reset allows newly received messages that pass arbitration to overwrite unread (i.e., message centers with DTUP = 1) messages. When an overwrite occurs, the receive overwrite bit (ROW) in the respective CAN message control register is set. When WTOE = 0, message centers that have data waiting to be read (indicated by DTUP = 1) or transmitted (EXTRQ = 1) are not overwritten by incoming data.

Special care must be taken when reading data from a message center with the overwrite feature enabled (WTOE = 1). Caution is necessary because the WTOE bit, when set, allows an incoming message to overwrite the message center. If an overwrite occurs at the same time that software is attempting to read several bytes from the message center (such as a multibyte data field), it is possible that the read could return a mix of information from the old and overwriting messages. If the message center being set up with TWOE = 1 was previously a transmit message center, ensure that the TIH bit is cleared to 0 (TIH can only be written while T/$\overline{R}$ is set to 1). If TIH is set to 1 and that message center is changed to receive with WTOE = 1, the ROW bit will always read back a 1, even though a receive overwrite condition may not have occurred. To avoid this situation, software should clear the DTUP bit to 0 prior to reading the message center, and then verify afterwards that the DTUP bit remained at 0. If DTUP remains cleared after the read, no overwrite occurred and the returned data was correct. If DTUP = 1 after the read, then software again should clear DTUP = 0 and reread the message center, since a possible overwrite has occurred. The original message will be lost (as planned since WTOE = 1), but a new message should be available on the next read.

One important use of the WTOE bit is to allow the microcontroller to program multiple message centers with the same ID when operating in the receive mode, with WTOE = 0. This allows the CAN module to store multiple incoming messages in a series of message centers, creating a large storage area for high-speed recovery of large amounts of data. The CPU is required to manage the use of these message centers to keep track of the incoming data, but the use of multiple message centers and disabling of their overwrite (WTOE = 0) function prevents the module from potentially losing data during a high-speed data transfer.

The following examples demonstrate the use of the WTOE and other bits when using multiple message centers with the same arbitration value. Case 2 illustrates the approach described above for configuring multiple message centers to capture a large amount of data at a relatively high rate.

Case 1: WTOE = 1 (Overwrites Allowed)

1) Software configures message centers 1 and 2 with the same arbitration value (abbreviated AV).

2) Software configures message centers 1 and 2 to receive (T/$\overline{R}$ = 0) and to allow message overwrite (WTOE = 1).

3) The first message received that matches AV is stored in message center 1, DTUP = 1.

4) The second message that matches AV is stored in message center 1, DTUP = ROW = 1.

5) The third message that matches AV is stored in message center 1.

6) Etc.

Note that in this example, message center 2 never receives a message and that, if software does not read message center 1 before the second message is received, the first message is lost.

Case 2: WTOE = 0 (Overwrites Disabled)

1) Software configures message centers 1 and 2 with the same arbitration value (abbreviated AV).

2) Software configures message centers 1 and 2 to receive (T/$\overline{R}$ = 0) and to disable message overwrite (WTOE = 0).

3) The first message received that matches AV is stored in message center 1, DTUP = 1.

4) The second message received that matches AV is stored in message center 2, DTUP = 1.

5) Software reads message center 1 and then programs message center 1, DTUP = 0.

6) The third message received that matches AV is stored in message center 1, DTUP = 1.

7) Software reads message center 2 and then programs message center 2, DTUP = 0.

8) The fourth message received that matches AV is stored in message center 2, DTUP = 1.

9) Etc.

Note that in this example, message center 1 or 2 is never overwritten. The user must ensure that the proper number of message centers be allocated to the same arbitration value when using this arrangement. If software fails to read the allocated message group, an incoming message can be lost without software realizing it (ROW is never set when WTOE = 0). To put a message center back into operation, software must force DTUP = 0 and EXTRQ = 0. This indicates that software has read the message center.

4.12 Special Considerations for Message Center 15

Message center 15 incorporates a shadow message center used to buffer incoming messages, in addition to the standard message center registers. When the message center is empty (DTUP = EXTRQ = 0), incoming messages are loaded directly into the message center registers. When the message center has unread data (DTUP = 1) or a pending remote frame request (EXTRQ = 1), incoming messages are loaded into the shadow message center. Unread contents of the shadow message center are automatically loaded into the message center when it becomes empty (DTUP = 0). An overwrite condition is possible when both the message center 15 and shadow message centers are full.

The response of message center 15 to the overwrite condition is dependent on the WTOE bit. When overwrite is enabled (WTOE = 1) and there is unread data (DTUP = 1) or a pending remote frame request (EXTRQ = 1), successfully received messages are stored in the shadow message center, overwriting existing data. If the shadow message center contained previously unread data at the time of the overwrite, the message center 15 ROW bit is set. If the shadow message center was empty at the time, then the incoming message is simply loaded into the shadow buffer and ROW is not set to 1. Note that the message center 15 ROW bit reflects only an overwrite of the shadow message center, not the message center registers (as with message centers 1–14).

When WTOE = 0, there is unread data (DTUP = 1), or a pending remote frame request (EXTRQ = 1) in message center 15, and there is already a message stored in the shadow buffer, incoming messages are not stored in either the message center or shadow buffer.

4.13 Using the Autobaud Feature

It is sometimes necessary to connect a CAN node to a network with an unknown baud rate. The MAXQ7665/MAXQ7666's autobaud feature provides a simple way for the CAN module to determine the baud rate of the network and reconfigure the MAXQ7665/MAXQ7666 to operate at that baud rate. Special hardware inside the CAN module allows it to interface to a fully functional CAN bus and perform the autobaud feature without disturbing other CAN nodes.

The theory behind the CAN autobaud feature is relatively simple. If a CAN module operating at a particular baud rate listens in on a CAN bus operating at a different baud rate, it sees a random bit stream. Because the bit stream does not conform to the CAN2.0B protocol, a large number of bus errors (bit 0 error, bit 1 error, bit stuff error, etc.) are seen by the "listening" CAN. These errors increment the CAN error-counter register. With only a moderate amount of CAN traffic, enough errors quickly accumulate to set the CAN error-count-exceeded (EC96/128) bit in the CAN 0 status register (C0S; A4h). This can be used as an indicator that the MAXQ7665/MAXQ7666 are not operating at the same baud rate as the CAN bus. The MAXQ7665/MAXQ7666 would then adjust their baud rate and repeat the process.

If, after a period of time, only a small number of errors have accumulated (most likely due to normal transmission noise), the MAXQ7665/MAXQ7666 are operating at the correct baud rate. The autobaud process is further simplified by the fact that most networks only operate at a small number of values. For example, DeviceNet operates at 125kbps, 250kbps, and 500kbps, so a device attempting to autobaud to a DeviceNet network would only have to test three baud rates.

The autobaud feature for the CAN module is enabled by setting the autobaud bit (C0C.2). Setting this bit activates a special loopback circuit within the CAN module that logically ANDs incoming network data received on the Rx pin with the Tx pin of the CAN module. While the autobaud bit is set, the CAN module disables its transmit output and places it in the recessive (high) state, so that error frames generated by the autobauding CAN module do not disturb other devices on the network during the procedure. Figure 4-13 outlines the CAN autobaud feature.

The following user-defined software procedure can be used with the autobaud feature to determine the baud rate of the network.

1) Set CRST = 1 to disable bus activity. Setting this bit also sets the SWINT bit, enabling access to control/status registers, and also clears the C0RE and C0TE registers.

2) Configure bus timing registers to set desired baud rate.

3) Set autobaud bit = 1.

4) Set SWINT = 0 to enable CAN module and begin listening for errors.

5) Delay approximately 500ms (allow enough time for > 128 errors to occur).

6) If CAN error-count-exceeded (EC96/128) bit is set, baud rate is incorrect. Select a new baud rate and repeat procedure. If EC96/128 bit is not set, the MAXQ7665/MAXQ7666 CAN module is set to the correct baud rate.
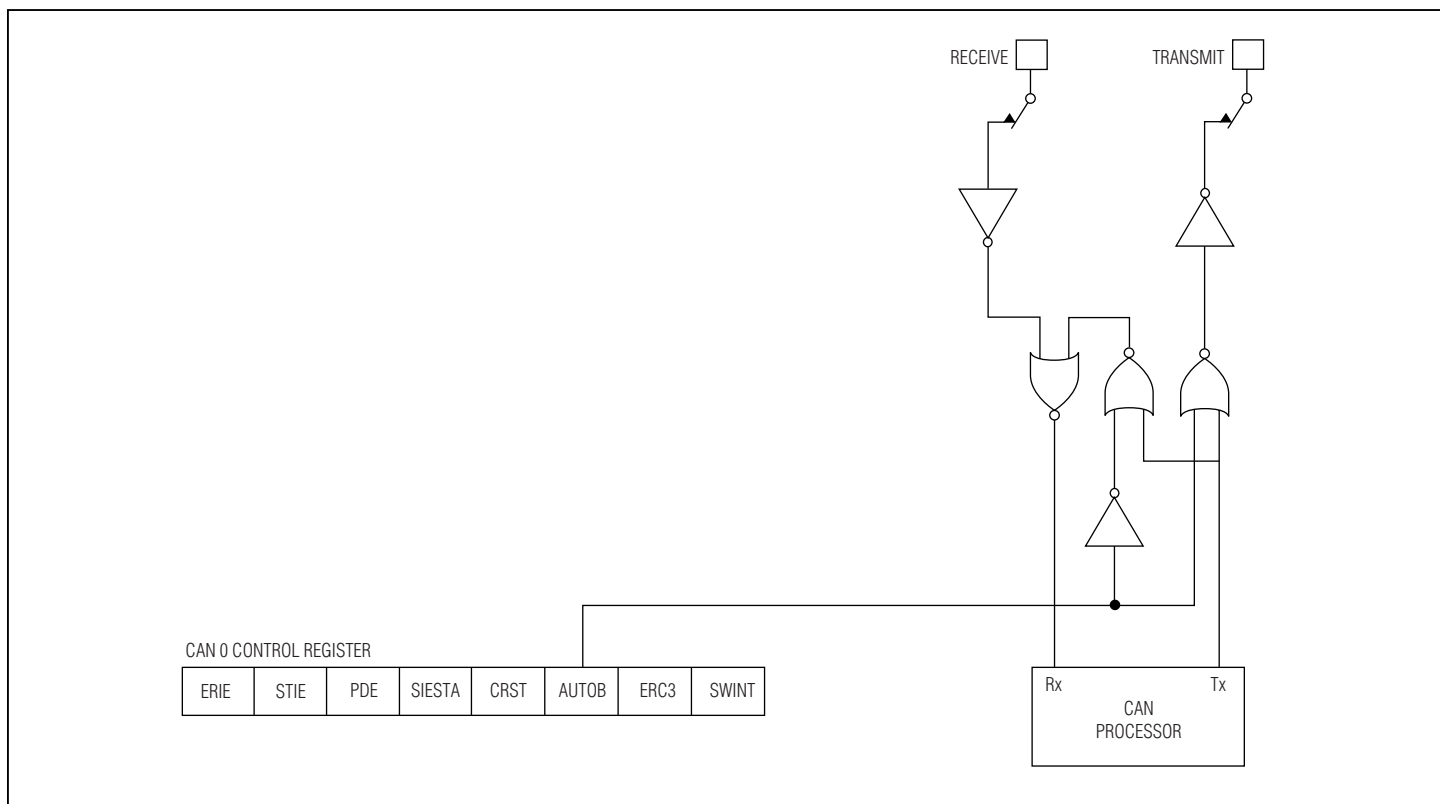
Figure 4-13. CAN Autobaud Feature

4.14 BUSON/BUSOFF Recovery and Error Counter Operation

The CAN module contains two peripheral registers that allow software to monitor and modify (under controlled conditions) the error counts associated with the transmit- and receive-error counters in the CAN module. These registers can be read at any time. Writing the CAN transmit-error counter registers updates both the transmit-error counter registers and the receive-error counter registers with the same value. Details are given in the peripheral registers description. These counters are incremented or decremented according to CAN specification version 2.0B, summarized in Table 4-3. The error counters are initialized by a CRST = 1 or a system reset to 00h. The error counters remain unchanged when the CAN module enters and exits from a low-power mode through the SIESTA or PDE bit.

Changes to the error counters are performed according to the following rules. This level of detail is not necessary for the average CAN user, and full information is provided in the CAN2.0B specification. More than one rule can apply to a given message.

Table 4-3. Rules for Changes to Error Counters

CONDITION	EFFECT ON ERROR COUNTERS
Error detected by receiver, unless the detected error was a bit error during the sending of an active error flag or an overload flag.	Receive-error counter incremented by 1.
Receiver detects a dominant bit as the first bit after sending an error flag.	Receive-error counter incremented by 8.
Transmitter sends an error flag. Note, however, that the transmit-error count does not change if: (1) The transmitter is error passive and detects an acknowledgement error because of not detecting a dominant acknowledge, and does not detect a dominant bit while sending its passive error flag. (2) Or, if the transmitter sends an error flag because a stuff error occurred during arbitration, and has been sent as recessive, but monitored as dominant.	Transmit-error counter incremented by 8.
Transmitter detects a bit error while sending an active error flag or an overload flag.	Transmit-error counter incremented by 8.
Receiver detects a bit error while sending an active error flag or an overload flag.	Receive-error counter incremented by 8.
Node detects the 14th consecutive dominant bit (in case of an active error flag or an overload flag), or detects the 8th consecutive dominant bit following a passive error flag, or after a sequence of additional eight consecutive dominant bits.	Transmit-error counter incremented by 8. Receive-error counter incremented by 8.
Message is successfully transmitted (acknowledge received and no error until end of frame is complete).	Transmit-error counter is decremented by 1 (unless it was already 0).
A message has been successfully received (reception without error up to the acknowledge slot and the successful sending of the acknowledge bit), and the receive-error count was between 1 and 127.	Receive-error counter decremented by 1.
A message has been successfully received (reception without error up to the acknowledge slot and the successful sending of the acknowledge bit), and the receive-error count was greater than 127.	Receive-error counter is set to a value between 119 and 127.

A node is error-active when the transmit- and receive-error counters are less than 128. When in an error-active state, an error condition causes the node to send an error frame on the bus. A node is error-passive when the transmit-error count equals or exceeds 128, or when the receive-error count equals or exceeds 128. An error-passive node does not transmit an error frame on the bus. An error-passive node becomes error-active again when both the transmit-error count and the receive-error count are less than or equal to 127.

A node is BUSOFF when the transmit-error count is greater than or equal to 256. A BUSOFF node becomes error-active (no longer BUSOFF) when its error counters are both set to 0 and after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus.

After exceeding the error-passive limit (128), the receive-error counter is not increased any further. When a message is received correctly, the counter is set again to a value between 119 and 127 (compare with CAN2.0B specification). After reaching BUSOFF status, the transmit-error counter is undefined while the receive-error counter is cleared and changes its function. The receive-error counter is incremented after every 11 consecutive recessive bits on the bus. These 11 bits correspond to the gap between two messages on the bus. If the receive-error counter reaches count = 128 following the BUSOFF recovery sequence, the CAN module changes automatically back to the status of BUSON and then sets SWINT = 1. After setting SWINT, all internal flags of the CAN module are reset and the error counters are cleared. A recovery from a BUSOFF condition does not alter any of the previously programmed dual port memory values or peripheral registers, apart from the transmit- and receive-error peripheral registers and the error conditions displayed in CAN status register. The bus timing remains as previously programmed.

4.15 Bit Timing

Bit timing in the CAN2.0B specification is based on a unit called the nominal bit time. The nominal bit time is further subdivided into four specific time periods.

1) The SYNC_SEG time segment is where an edge is expected when synchronizing to the CAN bus.

2) The PROP_SEG time segment is provided to compensate for the physical times associated with the CAN bus network.

3) The PHASE_SEG1 and PHASE_SEG2 time segments compensate for edge phase errors.

4) The PHASE_SEG1 and PHASE_SEG2 time segments can be lengthened or shorted through the use of the SJW1 and SJW0 bits in the CAN 0 bus timing register zero.

The CAN bus bit data is evaluated at a specific sample point. A time quantum (t_{QU}) is a unit of time derived from the division of the microcontroller system clock by both the baud-rate prescaler (programmed by the BPR7:BPR0 bits of the CAN 0 operation control register and CAN 0 bus timing register) and the system clock divider (programmed by the CD1:CD0 and PMME bits of the CKCN register). Combining the PROP_SEG and PHASE_SEG1 time segments into one time period termed t_{TSEG1}, and equating the SYNC_SEG time segment to t_{SYNC_SEG} and PHASE_SEG2 to t_{TSEG2}, provides the basis for the time segments outlined in Figure 4-14 and in the CAN bus timing peripheral register descriptions.
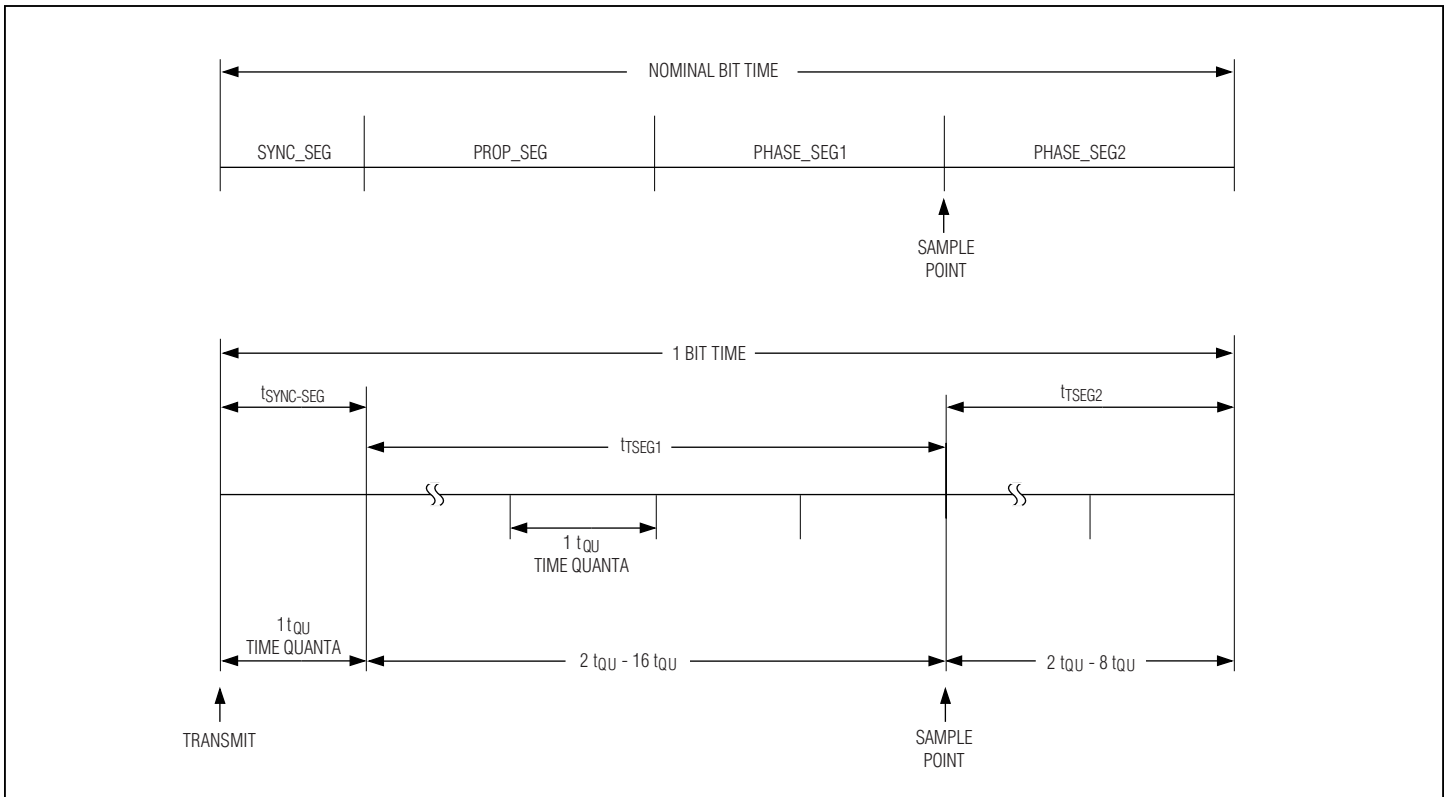
Figure 4-14. Bit Timing

MAXQ7665/MAXQ7666 User's Guide

The CAN 0 bus timing register 0 (C0BT0) contains the control bits for the PHASE_SEG1 and PHASE_SEG2 time segments as well as the baud-rate prescaler (BPR5:BPR0) bits. CAN 0 bus timing register 1 (C0BT1) controls the sampling rate, the time segment two bits that control the number of clock cycles assigned to the phase segment 2 portion, and the time segment one bits that determine the number of clock cycles assigned to the phase segment 1 portion. The value of both of the bus timing registers is automatically loaded into the CAN module following each software change of the SWINT bit from a 1 to a 0 by the microcontroller. The bit timing parameters must be configured before starting operation of the CAN module. These registers can only be modified during a software initialization (SWINT = 1), when the CAN module is NOT in a BUSOFF mode, and after the removal of a system reset or a CAN reset. To avoid unpredictable behavior of the CAN module, the bus timing registers should never be written with all zeros. To prevent this, the SWINT is forced to 0 when TSEG1 = TSEG2 = 00h.

The timing of the various time segments is determined by the following formulas. Most users never need to perform these calculations, as other devices already attached to the network dictate the bus timing parameters.

$$t_{QU} = \frac{BRPV \times CCD}{f_{OSC}}$$

$$t_{SYNC\_SEG} = 1 \times t_{QU}$$

$$t_{TSEG1} = (TS1\_LEN) \times t_{QU}$$

$$t_{TSEG2} = (TS1\_LEN) \times t_{QU}$$

$$t_{SJW} = (SJW) \times t_{QU}$$

$$t_{QU} \ per \ bit = \frac{1}{baud \ rate \times t_{QU}}$$

(Only integer values are permitted.)

where BRPV is the CAN baud-rate prescaler value found in the earlier description of the C0BT0 and COR registers, f_{OSC} is the crystal or external oscillator frequency of the microprocessor, and TS1_LEN and TS2_LEN are listed in the description of the TSEG26:TSEG24 and TSEG13:TSEG10 bits in the CAN bus timing register 1 (C0BT1). SJW is listed in the description of the SJW1:SJW0 bits in the CAN bus timing register 0 (C0BT0). The CAN clock divide (CCD) value is a factor tied to the current microcontroller system clock selection CKCN (see the peripheral register description) and can be referenced in the following table.

Table 4-4. CAN Clock Divide Selection

CD1	CD0	PMME	CCD
0	0	0	1
0	1	0	2
1	0	0	4
1	1	0	8
0	0	1	256
0	1	1	256
1	0	1	256
1	1	1	Reserved

The following restrictions apply to the above equations:

$$t_{TSEG1} \geq t_{TSEG2}$$
$$t_{TSEG2} \geq t_{SJW}$$
$$t_{SJW} < t_{TSEG1}$$
$$2 \leq TS1\_LEN \leq 16$$
$$2 \leq TS2\_LEN \leq 8$$
$$(TS1\_LEN + TS2\_LEN + 1) \leq 25$$

The nominal bit time applies when a synchronization edge falls within the t_{SYNC_SEG} period. The maximum bit time occurs when the synchronization edge falls outside of the t_{SYNC_SEG} period, and the synchronization jump width time is added to perform the resynchronization.

$$\text{nominal bit time} = t_{SYNC\_SEG} + t_{TSEG1} + t_{TSEG2}$$
$$= \frac{(BRPV)(CCD)\left[1 + (TS1\_LEN) + (TS2\_LEN)\right]}{f_{OSC}}$$
$$\text{maximum bit time} = t_{SYNC\_SEG} + t_{TSEG1} + t_{TSEG2} + t_{SJW}$$
$$= \frac{(BRPV)(CCD)\left[1 + (TS1\_LEN) + (TS2\_LEN) + (SJW)\right]}{f_{OSC}}$$
$$\text{CAN baud rate} = \frac{f_{OSC}}{(BRPV)(CCD)\left[1 + (TS1\_LEN) + (TS2\_LEN)\right]}$$

4.15.1 Threefold Bit Sampling

The MAXQ7665/MAXQ7666 support the ability to perform one or three samplings of each bit, based on the SMP bit (C0BT1.7). The single sample mode (SMP = 0) is available in all settings and takes one sample during each bit time. The triple sampling mode (SMP = 1) samples each bit three times for increased noise immunity. This mode can only be used when the baud-rate prescale value (BPRV) is greater than 3.

4.15.2 Bus Rate Timing Example

Table 4-5 shows an example bit timing setting for an 8MHz oscillator frequency and some baud-rate selections. Because of the large number of variables, there are many combinations not shown that can achieve a desired baud rate. There are a number of approaches to determining all the bit timing factors, but this uses the most common (i.e., the oscillator frequency and baud rate have already been determined by system constraints).

Table 4-5. Bit Timing Setting Example for 8MHz Oscillator Frequency

f_{OSC}	CCD	BRPV	t_{QU}	BAUD RATE	t_{QU} PER BIT	TS1_LEN	TS2_LEN	SJW	SMP = 1 PERMITTED?
8MHz	1	1	125ns	1Mbps	8	4	3	2	No
	1	1	125ns	500kbps	16	10	5	4	No
	1	1	250ns	250kbps	16	10	5	4	No
	2	2	500ns	125kbps	16	10	5	4	No

To understand the table data, the following is an explanation of how the 8MHz oscillator frequency and a 125kbps CAN baud rate data is derived.

Various combinations of BRPV are selected until one is located that meets the "t_{QU} per bit" criteria, i.e., an integer value less than 25. Selecting BRPV = 2, the previously described equations state that there should be 16 t_{QU} per bit. That leaves 16-1 or 15 t_{QU} remaining for TS1_LEN and TS2_LEN, which are arbitrarily assigned as shown. Because BRPV < 3, the triple sampling feature (SMP = 1) cannot be used.

4.16 CAN Bus Activity

The CAN bus activity (CAN0BA) status is active when a CAN bus activity is detected on the CAN input pin (Figure 4-15). This signal is used as one of the switchback sources for PMM mode or a wake-up source for stop mode if its interrupt function is also enabled. The status bit CAN0BA in the COR register can be used by software to determine the switchback or wake-up source.
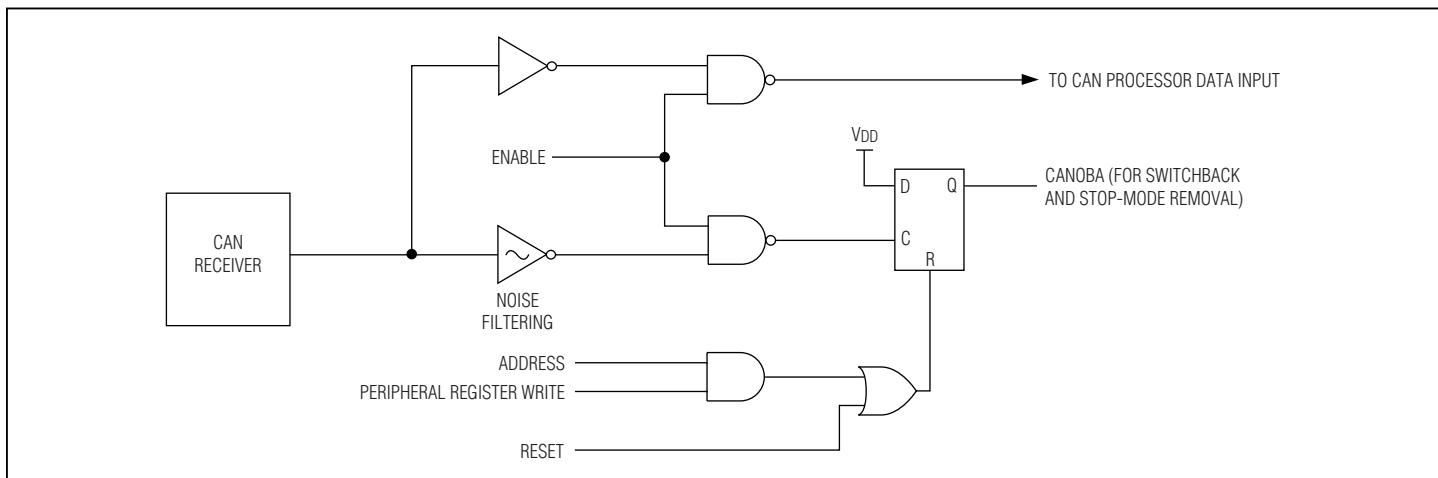

Figure 4-15. CAN Bus Activity

4.16.1 Issues with Stop Mode Entry While CAN is Active

When bits PDE, CRST, and SWINT are all cleared to logic 0, this condition indicates the CAN processor is active, even though this does not mean that the CAN is actively transmitting or receiving a message. However, if the microcontroller is trying to enter stop mode while the CAN processor is active, this could be catastrophic to the CAN network if it is actually transmitting a message.

The issue is directly related to the possibility of holding the CAN bus in dominate state when the system clock is stopped in stop mode while the CAN is transmitting a low value. Normally, if a CAN node is in a fault state, other nodes automatically send out error frames until the problem node takes itself offline. In this case, the problem node cannot take itself offline because it has no way to receive the error frame or to increment the error counts without the clock.

In order to confine error to the CAN processor when it is inadvertently entering stop mode, a hardware solution is implemented:

- The TXD output is forced high by hardware, and automatically takes the CAN processor offline.
- The SWINT bit is forced to logic 1 after the stop mode is exited and 11 consecutive recessive bits on the CAN bus have been received. Setting the SWINT bit to logic 1 inactivates the CAN processor.
- Once the SWINT bit is set, the TXD pin is released by the hardware, returning the control of the CAN bus to the CAN processor.
- The logic state of the SWINT bit also signifies the user that an error may occur and the CAN processor has been forced offline. To activate the CAN processor, the user can clear the SWINT bit.

The following are some side effects that may not be addressed by the hardware solution.

- There is neither an error associated with this condition in the CAN status register, nor an interrupt associated with this condition. The user software should be able to determine the SWINT has been changed from 0 to 1 by hardware.
- It is possible that if a transmitted message is interrupted, it may be lost and have to be retransmitted manually. It is also possible that the message will be retransmitted automatically when the software clears SWINT if the error count is below the threshold.
- The CAN error counters may have incremented due to the broken-up message and holding the bus high.

To avoid issues with stop mode entry, the user software should poll the CRST, SWINT, and PDE bits in the CAN control peripheral register before setting the STOP bit. At least one of these bits should be set to 1 before entering stop mode. If all three bits are cleared to 0 when stop mode is entered, the CAN is taken offline and deactivated.

SECTION 5: OSCILLATOR/CLOCK GENERATION MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 5: OSCILLATOR/CLOCK GENERATION MODULE

The MAXQ7665/MAXQ7666 oscillator/clock generation module supplies the system clock for the microcontroller core and all the peripheral modules. The MAXQ7665/MAXQ7666 are designed to operate up to 8MHz. Except where explicitly stated, the MAXQ7665 and MAXQ7666 have identical features.

The MAXQ7665/MAXQ7666 oscillator/clock generation module features include:

- Internal 7.6MHz RC oscillator
- Internal high-frequency oscillator, using an external crystal or resonator (up to 8MHz)
- External high-frequency clock signal (up to 8MHz)
- External crystal-fail detection and automatic switchover
- Power-up timer
- Power-saving management modes
- Watchdog timer

5.1 Architecture

Figure 5-1 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 oscillator/clock generation module. All functional modules in the MAXQ7665/MAXQ7666 are synchronized to a single system clock except the watchdog timer, which always operates using the internal, undivided 7.6MHz RC oscillator. The internal clock circuitry generates the system clock from one of three possible sources:

- Internal 7.6MHz RC oscillator
- Internal high-frequency oscillator, using an external crystal or resonator
- External high-frequency clock signal

These options provide the flexibility to select the clock source that best fits a particular application. The external crystal and clock are mutually exclusive since they share a common pin. The internal 7.6MHz RC oscillator provides a low-cost system solution that eliminates the need for a high-frequency crystal in some applications. The clock source selection is determined by the state of the XT bit in the CKCN register. When XT = 0, the internal 7.6MHz RC oscillator is used for clock generation; when XT = 1, the clock source is from external, either from an external clock or a crystal depending on the user system configuration. The internal 7.6MHz RC oscillator is selected as the default clock source on a power-on reset condition.

When the device is powered up, the power-on reset circuitry holds the device in reset to:

- Start up the internal 7.6MHz RC oscillator
- Reset the power-up counter, and
- Allow power-up delay of 65,536 internal 7.6MHz RC oscillator cycles (8.6ms typical) before releasing the reset and starting CPU operation

The oscillator/clock generation module includes a clock divider (CD1:CD0 bits) to select the number of oscillator clock source cycles per system clock. By default, one system clock is generated for every two oscillator cycles (divide by 2). Maximum performance is achieved by changing this to one system clock for each oscillator cycle (divide by 1).
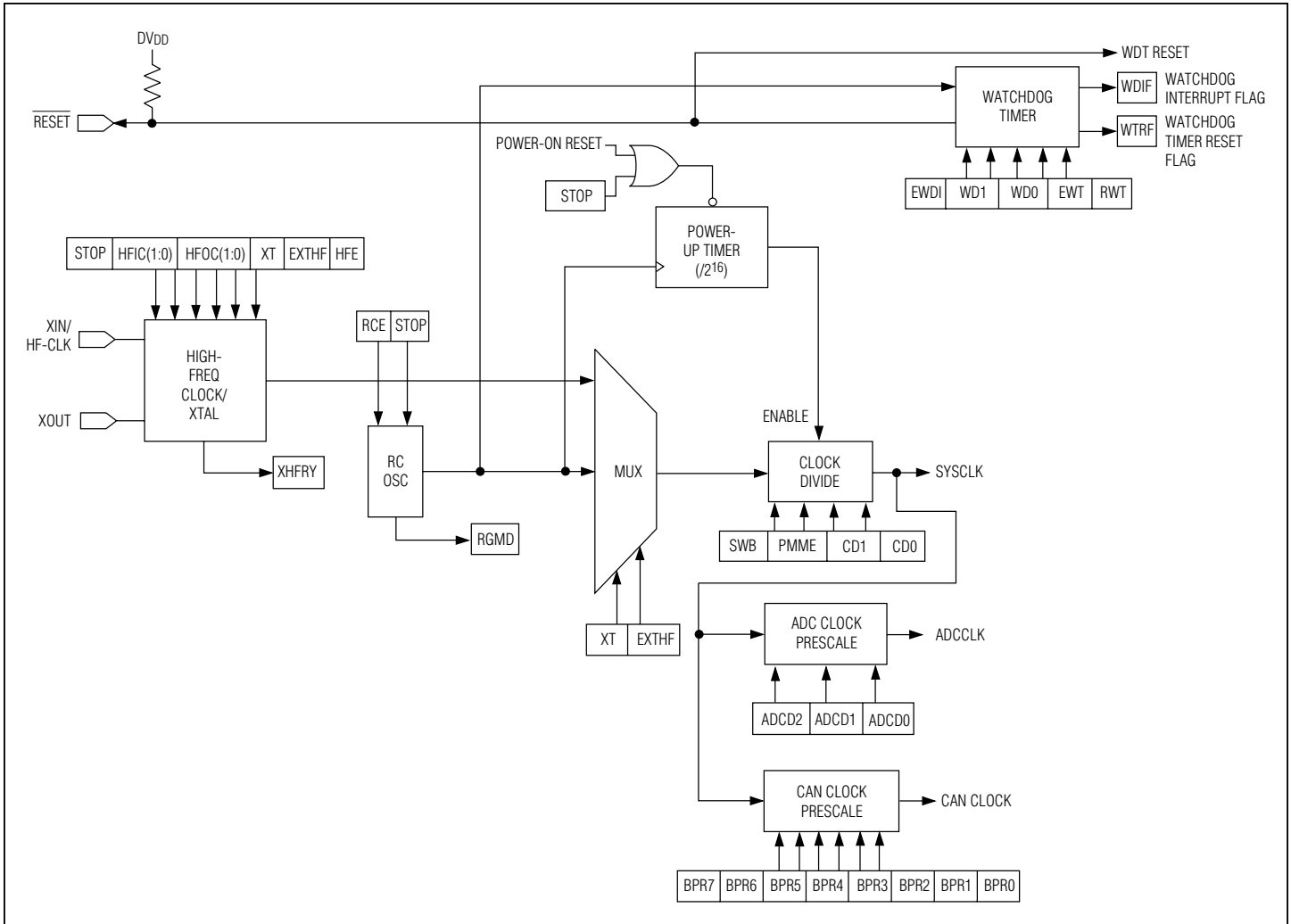
Figure 5-1. Oscillator/Clock Generation Module Block Diagram

5.1.1 Oscillator/Clock Generation Module Pins

Table 5-1 shows the MAXQ7665/MAXQ7666 oscillator/clock generation module signals.

Table 5-1. MAXQ7665/MAXQ7666 Oscillator/Clock Generation Module Pins

OSCILLATOR SIGNAL	PIN		DESCRIPTION
	48	56	
XIN	43	49	High-Frequency Crystal Input. Connect an external crystal or resonator between XIN and XOUT as the high-frequency oscillator clock. Alternatively, XIN is the input for an external high-frequency clock source when XOUT is floating. Leave XIN unconnected if an external clock source is not used.
XOUT	42	48	High-Frequency Crystal Output. Connect an external crystal or resonator between XIN and XOUT as the high-frequency oscillator clock. Alternatively, float XOUT when an external, high frequency clock source is connected to the XIN pin.

5.2 Oscillator/Clock Generation Registers

The MAXQ7665/MAXQ7666 oscillator/clock generation module registers are described here. All these registers are directly accessible by the microcontroller through the module/index address.

5.2.1 Analog Status Register (ASR)

The ASR register contains the high-frequency oscillator ready and failure flags. This register is cleared to its default state when it is read.

Register Description: **Analog Status Register**
Register Name: **ASR**
Register Address: **Module 05h, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	VIOLVL	DVLVL	—	—	XHFRY	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFINT	VIOBI	DVBI	—	ADCOV*	ADCRY	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *Reading the ASR resets to 0 all the status flag bits except VIOLVL and DVLVL.*
*The ADCOV bit is cleared by all forms of reset. All other bits are reset only by POR.

Bit 15: I/O Voltage Brownout Comparator Level (VIOLVL). See *Section 2* for more information on this bit.

Bit 14: Digital Voltage Brownout Comparator Level (DVLVL). See *Section 2* for more information on this bit.

Bits 13, 12, 10 to 7, 3, and 0: Reserved. Read 0, write ignored.

Bit 11: High-Frequency Oscillator Ready (XHFRY). This flag is set to logic 1 when the high-frequency crystal oscillator warmup is complete and ready for use. This bit is cleared after reading from the ASR register.

Bit 6: External High-Frequency Oscillator Failure Flag (HFFINT). This flag is set to logic 1 if the previously stable high-frequency clock source (XHFRY = 1) is sourced as the system clock (XT = 1) and a failure is detected (XHFRY = 0). This condition causes a hardware clock switchover by forcing the internal 7.6MHz RC oscillator enable (RCE = 1) and selecting it as the system clock (XT =0).

Bit 5: I/O Voltage Brownout Flag (VIOBI). See *Section 2* for more information on this bit.

Bit 4: Digital Brownout Flag (DVBI). See *Section 2* for more information on this bit.

Bit 2: ADC Overrun Flag (ADCOV). See *Section 3* for more information on this bit.

Bit 1: ADC Data Ready Flag (ADCRY). See *Section 3* for more information on this bit.

5.2.2 Oscillator Control Register (OSCC)

The OSCC register contains the oscillator enable and configuration bits.

Register Description: **Oscillator Control Register**
Register Name: **OSCC**
Register Address: **Module 05h, Index 0Ch**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	HFOC1	HFOC0	HFIC1	HFIC0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	ADCD2	ADCD1	ADCD0	—	—	EXTHF	RCE	HFE
Reset	0	0	0	0	0	0	1	0
Access	rw	rw	rw	r	r	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0002h on power-on reset and is not affected by other forms of reset.*

Bits 15 to 12, 4, and 3: Reserved. Read 0, write ignored.

Bits 11 and 10: High-Frequency Crystal Output Capacitance Select 1 and 0 (HFOC1 and HFOC0). These bits select the output capacitance of the on-chip high-frequency oscillator. The capacitor value is switched on to the XOUT pin of the MAXQ7665. The possible selections are given below. **Note:** For the default 00 setting, only a 1.3pF capacitor is switched on. The 6pF capacitor value is essentially due to stray capacitance.

HFOC1	HFOC0	CAPACITOR VALUE (pF)
0	0	6 (default)
0	1	17
1	0	27
1	1	34

Note: For the MAXQ7666, the HFIC1:HFIC0 bits select both the input and output capacitance of the on-chip high-frequency oscillator. The HFOC1:HFOC0 bits are used to select one of four possible crystal drive strengths as shown in the following table. Refer to the MAXQ7666 data sheet for typical drive strengths.

HFOC1	HFOC0	DRIVE STRENGTH
0	0	Drive 2 (default)
0	1	Drive 3 (highest)
1	0	Drive 0 (smallest)
1	1	Drive 1

Bits 9 and 8: High-Frequency Crystal Input Capacitance Select 1 and 0 (HFIC1 and HFIC0). These bits select the input capacitance of the on-chip high-frequency oscillator. The capacitor value is switched on to the XIN pin of the MAXQ7665. The possible selections are given below. **Note:** For the default 00 setting, only a 1.3pF capacitor is switched on. The 7pF capacitor value is essentially due to stray capacitance.

HFIC1	HFIC0	CAPACITOR VALUE (pF)
0	0	7 (default)
0	1	18
1	0	27
1	1	34

Note: For the MAXQ7666, the HFIC1:HFIC0 bits set both the input and output capacitance. The HFOC1:HFOC0 bits are used to select the crystal drive strength as previously explained.

Bits 7, 6, and 5: ADC Clock Divider Bits 2, 1, and 0 (ADCCD2, ADCCD1, ADCCD0). See *Section 3* for more information on these bits.

Bit 2: External High-Frequency Clock Enable (EXTHF). Setting this bit to logic 1 enables direct input of the external high-frequency clock to the XIN pin. Clearing this bit to logic 0 disables the high-frequency clock input. To use this as the system clock source, the XT bit in the CKCN register must be set to logic 1.

Bit 1: Internal 7.6MHz RC Oscillator Enable (RCE). Setting this bit to logic 1 enables the internal 7.6MHz RC oscillator. Clearing this bit to logic 0 disables the internal 7.6MHz RC oscillator. To use the internal 7.6MHz RC oscillator as the system clock source, the XT bit in the CKCN register must be set to 0. In the MAXQ7665/MAXQ7666, the internal 7.6MHz RC oscillator is the default system clock (RCE = 1, XT =0) after power-on reset. The watchdog timer is clocked by the internal 7.6MHz RC oscillator.

Note: The internal 7.6MHz RC oscillator is the default system clock source, so disabling it before the high-frequency crystal oscillator is stable (XHFRY = 1) and selected as the system clock source (XT = 1) can cause unrecoverable system errors.

Bit 0: High-Frequency Crystal Oscillator Enable (HFE). Setting this bit to logic 1 enables the on-chip high-frequency oscillator for use with an external crystal or resonator. Clearing this bit to 0 disables the high-frequency oscillator.

To use the external crystal or resonator as the system clock source, the HFE bit should first be set to logic 1 to allow the crystal oscillator to power-up. At some time later, the XT bit of the CKCN register should be set to logic 1 to swap the system clock source from the internal 7.6MHz RC oscillator to external crystal. If sufficient time has elapsed between setting HFE and setting XT to 1, the oscillator is ready and the clock source will swap immediately. Otherwise, the crystal oscillator continues to power-up and the clock source will swap to crystal when the power-up has completed. The XHFRY bit of the ASR indicates when the crystal oscillator circuit is ready (XHFRY = 1), and the clock source can swap from the internal 7.6MHz RC oscillator to external crystal.

5.2.3 System Clock Control Register (CKCN)

The 8-bit CKCN register is part of the system register group and used to support system clock generation. It controls the system clock speed and power management mode selection.

Register Description: **System Clock Control Register**
Register Name: **CKCN**
Register Address: **Module 08h, Index 0Eh**

Bit #	7	6	5	4	3	2	1	0
Name	XT	—	RGMD	STOP	SWB	PMME	CD1	CD0
Reset	0	0	1	0	0	0	0	1
Access	rw	r	r	rw	rw	rw	rw	rw

r = read, w = write
Note: *Bits 4:0 are set to 00001b on all forms of reset. See bit description for bits 7 and 5.*

Bit 7: External Crystal Select (XT). This bit selects the external crystal/clock or the internal 7.6MHz RC oscillator as the desired clock source. The XT bit is the inverse of RGMD except during the crystal warmup period when resuming from the stop mode through the 7.6MHz RC oscillator. This bit is cleared to 0 after a power-on reset, which selects the internal 7.6MHz RC oscillator as the clock source; otherwise, it is unchanged by other forms of reset.

Changing the XT bit from 0 to 1 causes the system clock source to swap from the internal RC to the high-frequency crystal oscillator. This change occurs automatically within a few clock cycles if sufficient crystal warmup time has elapsed since the HFE bit of the OSCC register was set. If the crystal has not finished warming up when XT is set to 1, the crystal oscillator continues to warm up and the clock source swaps to crystal when this is complete. The XHFRY bit of the ASR register indicates when the crystal oscillator circuit is ready and the clock source can swap from 7.6MHz RC oscillator to crystal.

Changing the XT bit from 1 to 0 selects the internal 7.6MHz RC oscillator as the system clock source. Allow four 7.6MHz RC oscillator cycles after enabling the 7.6MHz RC oscillator (RCE = 1) before switching XT to 0. To use the 7.6MHz RC oscillator, the RCE bit in the OSCC register must be set to logic 1, which enables the 7.6MHz RC oscillator.

Bit 6: Reserved. Read 0, write ignored.

Bit 5: 7.6MHz RC Oscillator Mode (RGMD). This read-only bit reflects the selection of clock source. RGMD = 1 indicates that the 7.6MHz RC oscillator is providing the system clock. RGMD = 0 indicates that the external crystal/clock is providing the system clock. Note that RGMD is set to 1 only for POR reset.

Bit 4: Stop Mode Select (STOP). Setting this bit to 1 causes the MAXQ7665/MAXQ7666 to enter stop mode. This will not change the currently selected clock divide ratio (CD0, CD1, and PMME). This bit is cleared by a reset or any of the enabled external interrupts. Stop mode disables all circuits within the MAXQ7665/MAXQ7666 including the watchdog timer and its clock source (the internal 7.6MHz RC oscillator). All clock sources, timers, and peripherals are halted, and no code execution occurs. The system clock is stopped, and all processing activity is halted.

Bit 3: Switchback Enable (SWB). If the SWB bit is cleared to 0, switchback mode is not active. If the SWB is set to 1, switchback mode is active. Switchback mode has no effect if power management mode is not active (PMME = 0). If power management mode is active and switchback mode is enabled, the PMME bit will be cleared to 0 when any of the following conditions occur:

- An external interrupt condition occurs on a port pin and the corresponding external interrupt is enabled.

- An active-low transition occurs on the UART serial receive-input line (modes 1, 2, and 3) and data reception is enabled.

- The SBUF0 register is written to send an outgoing byte through the UART and transmission is enabled.

- The SPIB register is written in master mode (STBY = 1) to send an outgoing character through the SPI module and transmission is enabled.

- The SPI module's $\overline{SS}$ (slave select input) signal is asserted in slave mode.
- A CAN bus activity on its data input (CANRXD) while its interrupt is enabled.
- Active debug mode is entered either by break point match or issuance of the debug command from background mode.

When any of these conditions cause switchback to clear PMME to 0, the system clock rate will then revert back to the divide-by-1 mode (CD1:CD0 = 00). After PMME is cleared to 0 by switchback, it may not be set back to 1 as long as any of the above conditions are true.

Bit 2: Power Management Mode Enable (PMME). If the PMME bit is cleared to 0, the values of CD1 and CD0 determine the number of oscillator (clock source) cycles per system clock cycle. If the PMME bit is set to 1, the values of CD1 and CD0 are ignored and the system clock operates in a fixed mode of 1 cycle per 256 oscillator cycles (divide by 256). If the PMME bit is set to 1 and switchback mode has been enabled (SWB = 1), when a switchback source (such as an enabled external interrupt) becomes active, PMME is cleared to 0 and cannot be set to 1 unless all switchback sources are inactive.

Note: The CD1 and CD0 (CKCN1:CKCN0) bits must both be cleared to 0 before setting the PMME bit to 1.

Bits 1 and 0: Clock Divide Control Bits 1 and 0 (CD1 and CD0). If the PMME bit is cleared, the CD1 and CD0 bits control the number of oscillator (clock source) cycles required to generate one system clock as follows:

PMME	CD1	CD0	OSCILLATOR CYCLES PER SYSTEM CLOCK CYCLE (CLOCK DIVIDE RATIO)
0	0	0	1
0	0	1	2 (default)
0	1	0	4
0	1	1	8
1	0	0	256

If the PMME bit is set to 1, the values of CD1 and CD0 may not be altered and do not affect the system clock frequency.

5.2.4 Watchdog Timer Control Register (WDCN)

The 8-bit WDCN register is part of the system register group and used to provide system control. It controls the watchdog timeout period and interrupt or reset generation on watchdog timeout. The watchdog timer is clocked by the internal 7.6MHz RC oscillator. Enabling the watchdog does not force the internal 7.6MHz RC oscillator enable (RCE) bit to logic 1.

Register Description: **Watchdog Timer Control Register**
Register Name: **WDCN**
Register Address: **Module 08h, Index 0Fh**

Bit #	7	6	5	4	3	2	1	0
Name	POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *Bits 5, 4, 3, and 0 are cleared to 0 on all forms of reset; for others, see the individual bit descriptions.*

Bit 7: Power-On Reset Flag (POR). This bit is set to 1 whenever a power-on reset occurs. It is unaffected by other forms of reset. This bit can be checked by software following a reset to determine if a power-on reset occurred. It should always be cleared by software following a reset so that the source of the next reset can be correctly determined by software.

Bit 6: Watchdog Interrupt Enable (EWDI). If this bit is set to 1, an interrupt request can be generated when the WDIF bit is set to 1 by any means. If this bit is cleared to 0, no interrupt will occur when WDIF is set to 1; however, it does not stop the watchdog timer or prevent watchdog timer resets from occurring if EWT = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWDI bit resets the watchdog interval and reset counter, and enables the watchdog timer. This bit is cleared to 0 by power-on reset and is unaffected by other forms of reset.

Note: *The watchdog timer is clocked by the internal 7.6MHz oscillator. Therefore, RCE must be set to 1 for the watchdog timer operation.*

Bits 5 and 4: Watchdog Timer Mode Select Bits 1 and 0 (WD1 and WD0). These bits determine the watchdog interval or the length of time between resetting of watchdog timer and the watchdog generated interrupt in terms of RC clocks. Modifying the watchdog interval via the WD1:WD0 bits automatically resets the watchdog timer unless the 512 RC clock reset counter is already in progress, in which case, changing the WD1:WD0 bits does not affect the watchdog timer or reset counter.

WD1	WD0	WATCHDOG TIMEOUT PERIOD			
		RC CLOCKS		MILLISECONDS (FOR RC = 7.6MHz)	
		UNTIL INTERRUPT	UNTIL RESET	UNTIL INTERRUPT	UNTIL RESET
0	0	2^{12} (default)	$2^{12} + 2^9$	0.539	0.606
0	1	2^{15}	$2^{15} + 2^9$	4.31	4.38
1	0	2^{18}	$2^{18} + 2^9$	34.49	34.56
1	1	2^{21}	$2^{21} + 2^9$	275.94	276.01

Bit 3: Watchdog Interrupt Flag (WDIF). This flag is set to 1 when the watchdog timer interval has elapsed or can be set to 1 by user software. When WDIF = 1, an interrupt request is generated if the watchdog interrupt has been enabled (EWDI = 1) and not otherwise masked or prevented by an interrupt already in service (i.e., IGE = 1, IMS = 1, and INS = 0 must be true for the interrupt to occur). This bit should be cleared by software before exiting the interrupt service routine to avoid repeated interrupts. Furthermore, if the watchdog reset has been enabled (EWT = 1), a reset is scheduled to occur 512 RC clock cycles following setting of the WDIF bit.

MAXQ7665/MAXQ7666 User's Guide

Bit 2: Watchdog Reset Flag (WTRF). This flag is set to 1 when the watchdog resets the processor. Software can check this bit following a reset to determine if the watchdog was the source of the reset. Setting this bit to 1 in software will not cause a watchdog reset. This bit is cleared by power-on reset only and is unaffected by other forms of reset. It should always be cleared by software following a reset so that the source of the next reset can be correctly determined by software. This bit is only set to 1 when a watchdog reset actually occurs, so if EWT is cleared to 0 when the watchdog timer elapses, this bit will not be set.

Bit 1: Enable Watchdog Timer Reset (EWT). If this bit is set to 1 when the watchdog timer elapses, the watchdog resets the processor 512 system clock cycles later unless action is taken to disable the reset event. Clearing this bit to 0 prevents a watchdog reset from occurring but does not stop the watchdog timer or prevent watchdog interrupts from occurring if EWDI = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWT bit resets the watchdog interval and reset counter, and enables the watchdog timer. This bit is cleared on power-on reset and is unaffected by other forms of reset.

Note: The watchdog timer is clocked by the internal 7.6MHz RC oscillator. Therefore, RCE should be set to 1 for the watchdog timer operation.

Bit 0: Reset Watchdog Timer (RWT). Setting this bit to 1 resets the watchdog timer count. If watchdog interrupt and/or reset modes are enabled, the software must set this bit to 1 before the watchdog timer elapses to prevent an interrupt or reset from occurring. This bit always returns 0 when read.

5.2.5 Analog Interrupt Enable Register (AIE)

The AIE register contains the enable bits for various analog interrupts. With respect to the clock generation logic, it contains the enable bit for high-frequency oscillator failure detection.

Register Description: **Analog Interrupt Enable Register**
Register Name: **AIE**
Register Address: **Module 05h, Index 0Ah**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	HFFIE	VIOBIE	DVBIE	—	AORIE	ADCIE	—
Reset	0	0	0	0	0	0	0	1
Access	r	rw	rw	rw	r	rw	rw	rw

r = read, w = write

Bits 15 to 7 and 3: Reserved. Read 0, write ignored.

Bit 6: External High-Frequency Oscillator Failure Interrupt Enable (HFFIE). This bit must be set to logic 1 to generate an interrupt request when high-frequency oscillator failure is detected and the HFFINT flag in the ASR register is set to logic 1. Clearing this bit to 0 disables the interrupt capability from the HFFINT flag.

Note: To be acknowledged by the microcontroller interrupt logic, this interrupt request must also be enabled by IGE bit in the IC register and IM5 mask in the IMR peripheral register.

Bit 5: I/O Voltage Brownout Interrupt Enable (VIOBIE). See *Section 2* for more information on this bit.

Bit 4: Digital Brownout Interrupt Enable (DVBIE). See *Section 2* for more information on this bit.

Bit 2: ADC Overrun Interrupt Enable (AORIE). See *Section 3* for more information on this bit.

Bit 1: ADC Data Ready Interrupt Enable (ADCIE). See *Section 3* for more information on this bit.

Bit 0: This bit is implemented and available to be used as a user-software-controlled bit.

5.3 System Clock Generation

All functional modules in the MAXQ7665/MAXQ7666 are synchronized to a single system clock. This system clock can be generated from one of three possible sources:

- Internal 7.6MHz RC oscillator
- Internal high-frequency oscillator using external crystal or resonator circuit
- External high-frequency clock signal

Table 5-2 shows the registers and bits used to control clock generation and selection. For more information, see the register descriptions in *Section 5.2*.

Table 5-2. Clock Generation and Selection Registers and Bits

REGISTER	ADDRESS	BIT	NAME	FUNCTION
CKCN	M8[0Eh]	[1:0]	CD[1:0]	Selects clock divide-by-1 (00), by-2 (01), by-4 (10), or by-8 (11) mode.
		2	PMME	Selects divide-by-256 mode (1) or normal clock divide mode (0).
		5	RGMD	Read-only. Indicates if RC oscillator (1) or external crystal/clock (0) is currently being used to provide the system clock.
		7	XT	Selects external crystal/clock (1) or internal RC (0) as the clock source.
OSCC	M5[0Ch]	0	HFE	Selects whether the internal high-frequency oscillator (for use with external crystal or resonator) is enabled (1) or not (0).
		1	RCE	Selects whether the internal RC oscillator is enabled (1) or not (0).
		2	EXTHF	Selects whether the external high-frequency clock (direct input) is enabled (1) or not (0).
		[9:8]	HFIC[1:0]	MAXQ7665: Selects the input capacitance of the on-chip high-frequency oscillator. MAXQ7666: Selects both the input and output capacitance of the on-chip high-frequency oscillator.
		[11:10]	HFOC[1:0]	MAXQ7665: Selects the output capacitance of the on-chip high-frequency oscillator. MAXQ7666: Selects the crystal drive strength of the on-chip high-frequency oscillator.
ASR	M5[0Bh]	11	XHFRY	Indicates if high-frequency oscillator warmup is complete (1) and ready. ASR read clears the bit.

5.3.1 Internal 7.6MHz Oscillator

The MAXQ7665/MAXQ7666 provide an internal 7.6MHz RC oscillator, which is used as the default source for the system clock following any power-on reset or exit from stop mode. This oscillator, which requires no external components, typically runs at 7.6MHz. The exact frequency may vary part to part and over temperature and supply voltage. For more details, refer to the MAXQ7665/MAXQ7666 data sheet.

Following a power-on reset, to start up the internal oscillator the DVDD power supply must be above the minimum power-on reset threshold of ~1.2V and the external $\overline{RESET}$ must be deasserted (1). When the DVDD power supply crosses the ~1.2V power-on reset threshold, the internal 7.6MHz RC oscillator starts running and the 16-bit power-up counter is enabled. After 65,536 counts of the internal 7.6MHz RC oscillator (8.6ms typical), the power-up counter is disabled. The internal 7.6MHz RC oscillator is enabled as the system clock (XT = 0, RGMD =1), and the MAXQ7665/MAXQ7666 start program execution at address 8000h in the utility ROM if the DVDD power supply is above the power-on reset rising threshold level (2.7V–2.99V). Figure 5-2 illustrates the internal RC oscillator startup and execution flow after a power-on reset. For more details on power-on reset, see *Section 2*.

To select the 7.6MHz RC oscillator as the system clock source, the XT bit (CKCN.7) must be set to 0 and RCE must be set to logic 1. Starting execution using the internal 7.6MHz RC oscillator requires a 4-cycle warmup delay under the following circumstances:

- After power-on reset
- After exiting stop mode
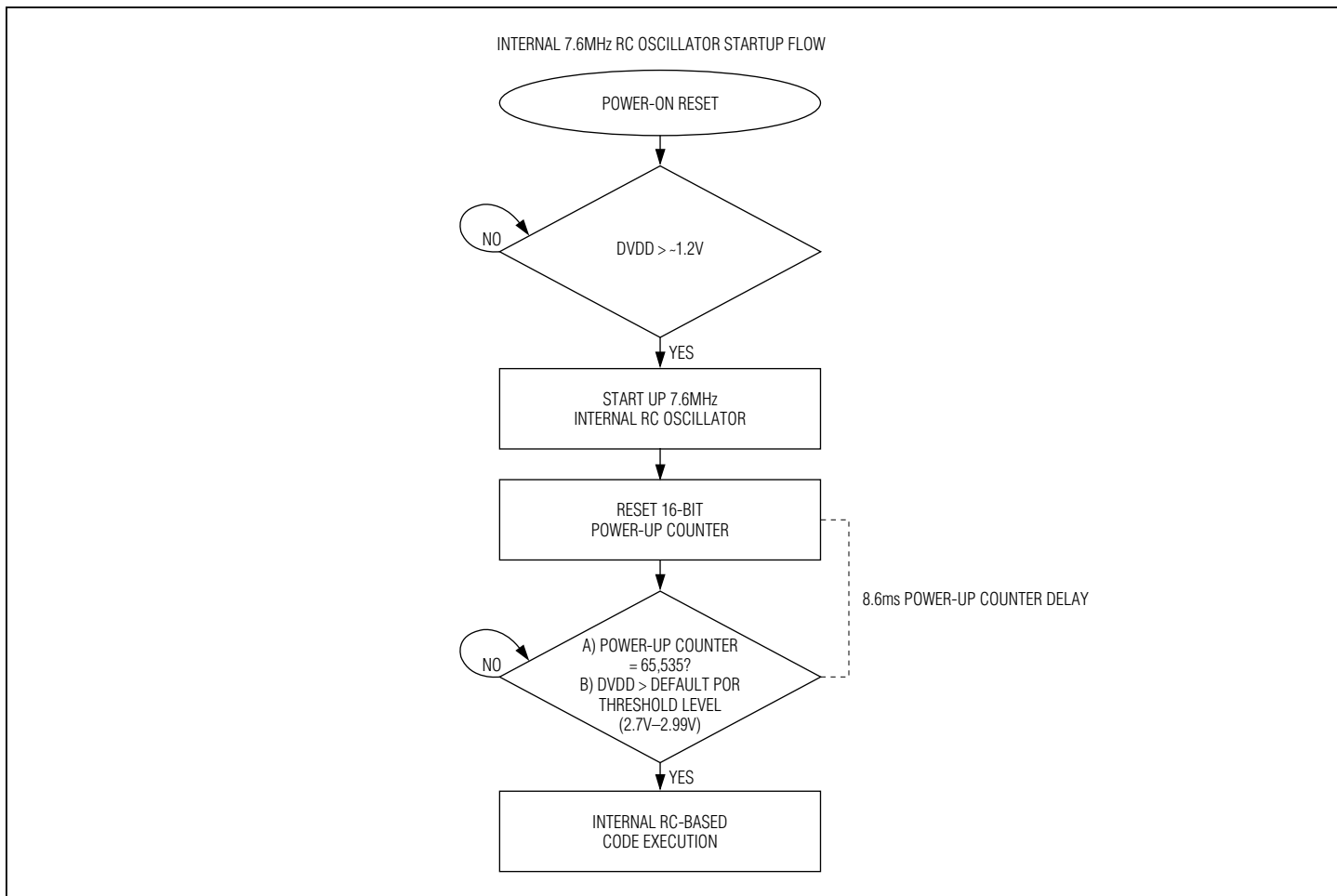- After enabling the 7.6MHz RC oscillator (RCE = 1) and before switching the XT bit from a 1 to 0

Figure 5-2. Oscillator Startup Flow

In all other cases, setting XT = 0 switches the system clock source to the internal RC immediately. The RGMD (CKCN.5) bit indicates the current system clock source. If the 7.6MHz RC oscillator is providing the system clock, RGMD equals 1; otherwise, RGMD equals 0. When used as the system clock, the 7.6MHz RC oscillator clock is divided down according to the PMME, CD1, CD0 bit selections (CKCN.2:CKCN.0) just the same as the external crystal/clock possibilities. The WDT alone is always clocked by the undivided 7.6MHz RC oscillator clock.

When the system clock source is switched from the 7.6MHz RC oscillator to the high-frequency external crystal/clock by setting XT = 1, the 7.6MHz RC oscillator will still be used as the system clock source until the warmup period has completed for the high-frequency oscillator. This is reflected by the value of the RGMD bit, which remains at 1 until the warmup for the high-frequency oscillator is complete and the clock switches over, at which point RGMD switches to 0.

Note that in order to use the 7.6MHz RC oscillator, the RCE bit in the OSCC register must be set to logic 1, which enables the 7.6MHz RC oscillator. The 7.6MHz RC oscillator is enabled (RCE = 1, XT = 0) by default after a power-on reset.

5.3.2 External Clock (Crystal/Resonator)

An external quartz crystal or a ceramic resonator can be connected from XIN to XOUT as the device determining the frequency of operation. The MAXQ7665/MAXQ7666 are designed to operate at a maximum frequency of 8.12MHz. For details of the high-frequency crystal oscillator specification, refer to the MAXQ7665/MAXQ7666 data sheet.

The crystal oscillator/resonator is disabled upon power-up, as the default mode for the MAXQ7665/MAXQ7666 is to run from the internal 7.6MHz RC oscillator. To use the external crystal/resonator, select the input (HFIC1:HFIC0) and output capacitance (HFOC1:HFOC0) of the internal high-frequency oscillator (to match the external crystal/resonator load capacitance requirement) in the OSCC register. Figure 5-3 shows the possible options. The HFE bit (OSCC.0) must be set to 1 to enable the internal high-frequency oscillator.

Note: For MAXQ7666, the HFIC1:HFIC0 bits select both the input and output capacitance. The HFOC1:HFOC0 bits are used to select the crystal drive strength as explained in the oscillator control register subsection.
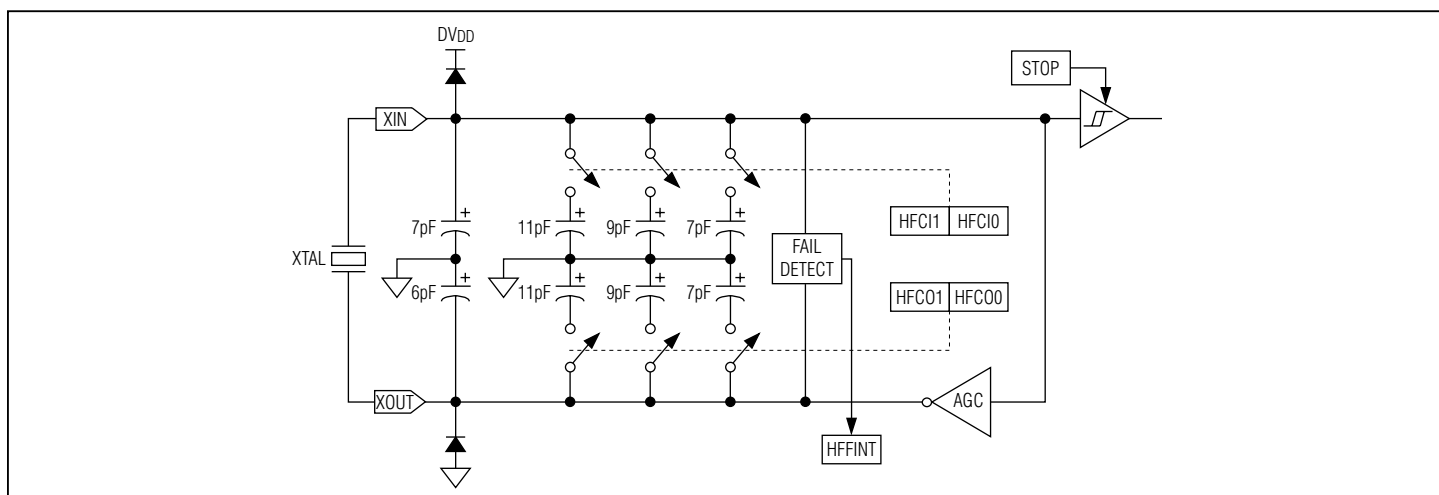

Figure 5-3. High-Frequency Crystal Oscillator Configuration

To select the crystal/resonator as the system clock source, the XT bit must be set to 1. When the system clock source is switched from the 7.6MHz RC oscillator to the high-frequency external crystal/resonator by setting XT = 1, the 7.6MHz RC oscillator will still be used as the system clock source until the warmup period has completed for the high-frequency oscillator. It is important that the 7.6MHz RC oscillator is not disabled (RCE = 0) before the system clock is switched to external crystal/resonator. The RGMD bit may be monitored by application software to determine when this switchover has actually occurred, as code execution continues during the warmup delay.

The switchover to the high-frequency oscillator requires a warmup delay of 4096 crystal/resonator clock cycles under the following circumstances:

- When resuming execution from stop mode with HFE = 1 and XT = 1.
- When the high-frequency oscillator has been shutdown (HFE = 0) or has not been started since the last POR or stop mode exit.

Setting the HFE bit to 1 allows the high-frequency oscillator to keep running even when the internal RC oscillator has been selected as the system clock. Once the high-frequency oscillator has been started and allowed to warm up for the first time (following a POR, exit from stop mode, shutdown of the oscillator), application code can switch to the high-frequency oscillator with a warmup delay of only four clock cycles instead of 4096. The XHFRY (ASR.11) bit can be monitored by application software to determine if the high-frequency oscillator warmup is complete (XHFRY = 1). Figure 5-4 illustrates the steps to select external crystal/resonator as the system clock source.

When the high-frequency oscillator (using external crystal/resonator) is used as the system clock source, the clock is divided down according to the PMME, CD1, CD0 bit selections (CKCN.2:CKCN.0).

5.3.2.1 High-Frequency Oscillator Application Configuration

The MAXQ7665/MAXQ7666 high-frequency oscillator is optimized for 8MHz operation. Figure 5-5 shows an example application configuration using an external crystal, two low-end capacitors, and an optional series resistor to limit current through the crystal. Pins XIN and XOUT connect the oscillator to the external crystal. By using the internal built-in capacitor (selected by HFIC1:HFIC0 and HFOC1:HFOC0 bits), the user can save board space and avoid the external end capacitors. The user should review the crystal specifications for appropriate values of the external components. Additionally, the user should verify the oscillator performance across the required DVDD and temperature range.

Note: For MAXQ7666, the HFIC1:HFIC0 bits select both the input and output capacitance. The HFOC1:HFOC0 bits are used to select the crystal drive strength as explained in the oscillator control register subsection.
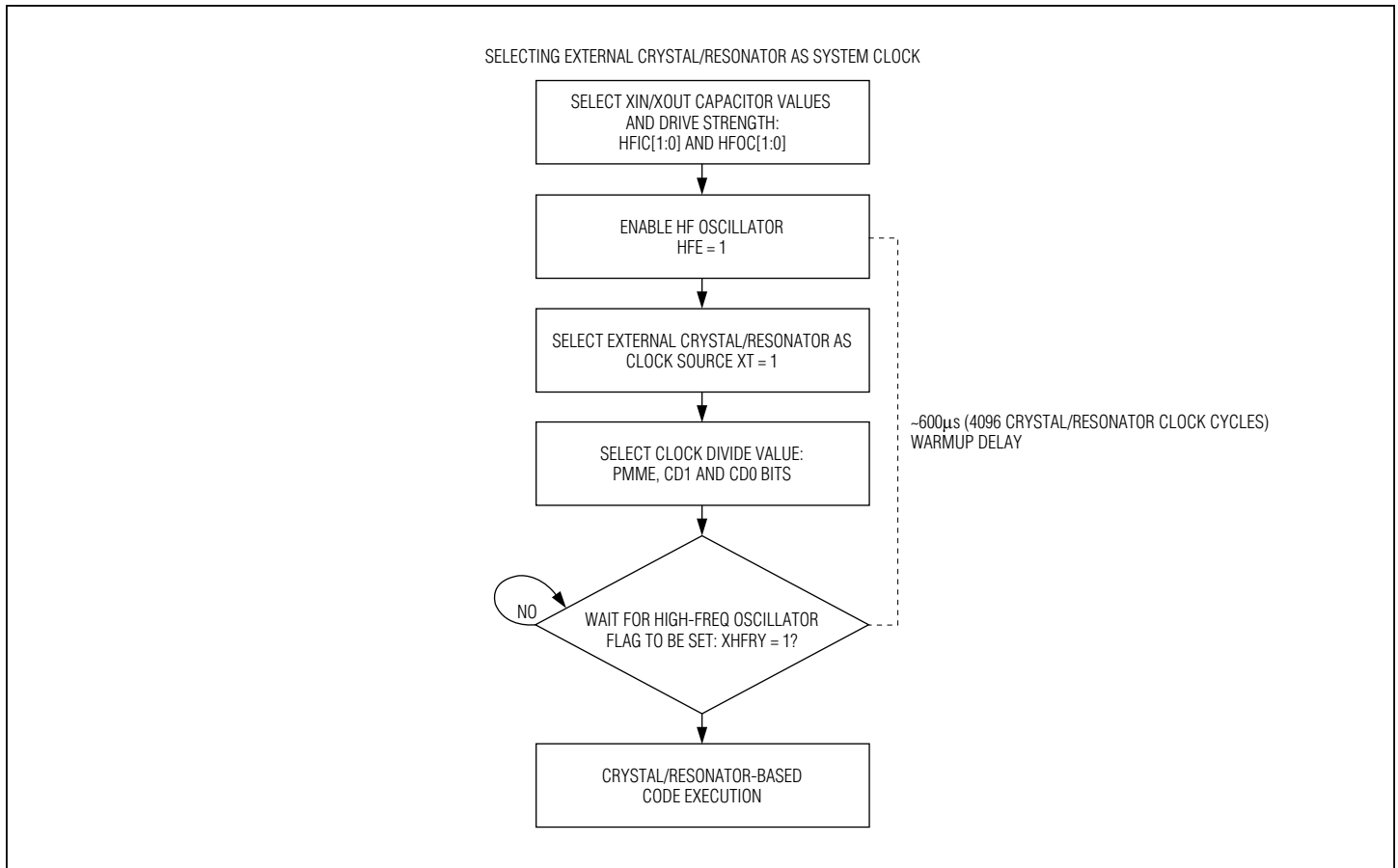
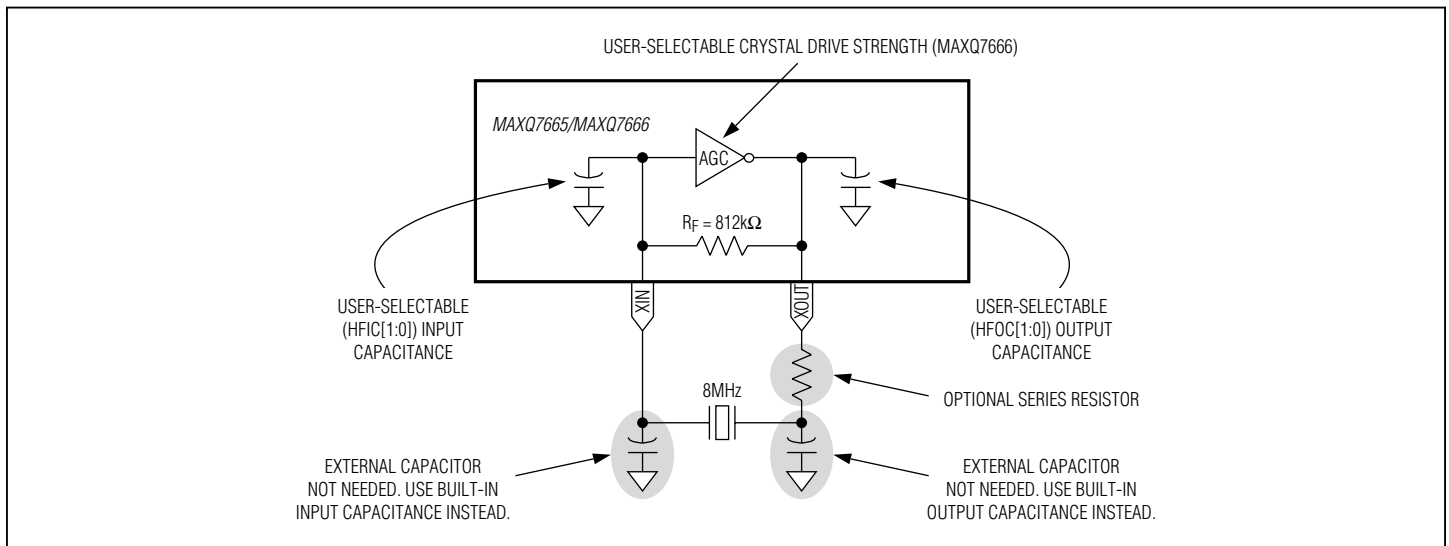Figure 5-4. Selecting External Crystal/Resonator as System Clock

Figure 5-5. High-Frequency Oscillator Application Configuration

5.3.3 External Clock (Direct Input)

The MAXQ7665/MAXQ7666 can also obtain the system clock signal directly from an external source. In this configuration, the clock generation circuitry is driven directly by an external clock.

To operate from an external clock, connect the clock source to the XIN pin and leave the XOUT pin floating. Figure 5-6 shows the external clock source configuration. The clock source should be driven through a CMOS driver at the same level as DVDD (nominally +3.3V). If the clock driver is a TTL gate, its output must be connected to DVDD through a pullup resistor to ensure a satisfactory logic level for active clock pulses. The MAXQ7665/MAXQ7666 are designed to operate at a maximum frequency of 8MHz. For more details, refer to the MAXQ7665/MAXQ7666 data sheet.
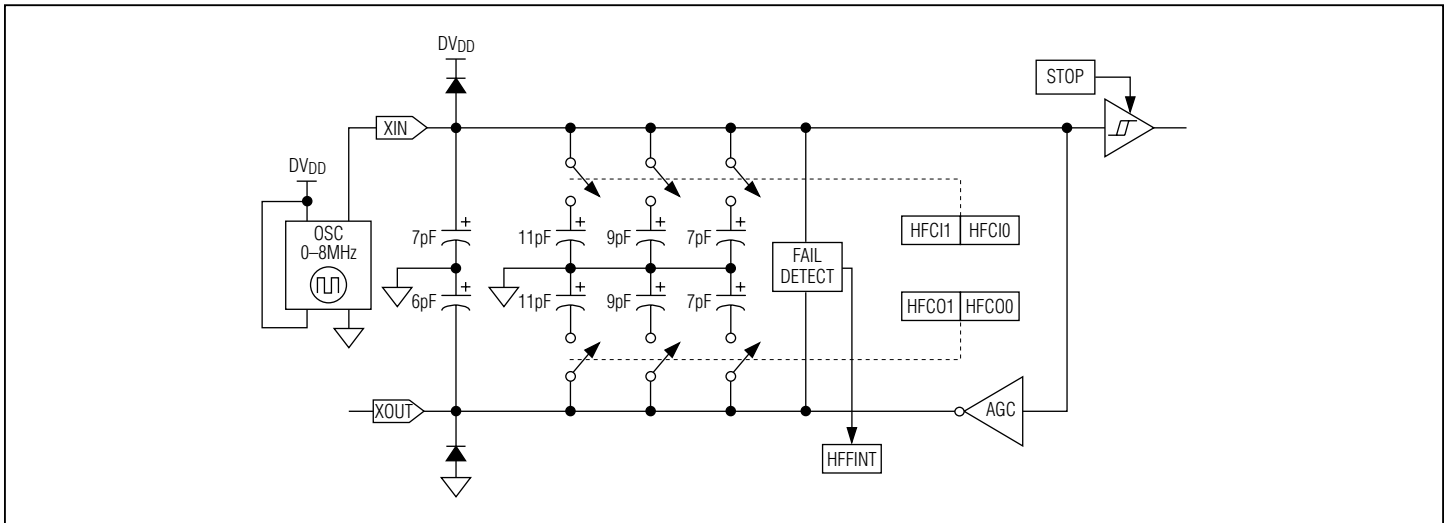

Figure 5-6. External Clock Source Configuration

The external clock input is disabled upon power-up, as the default mode for the MAXQ7665/MAXQ7666 is to run from the internal 7.6MHz RC oscillator. To use the external clock input, the EXTHF bit (OSCC.2) must be set to 1. To select the external clock as the system clock source, the XT bit (CKCN.7) must be set to 1. When the external clock input is used as the system clock source, the clock is divided down according to the PMME, CD1, CD0 bit selections (CKCN.2:CKCN.0).

5.3.4 Internal System Clock Generation

The internal system clock is derived from the currently selected system clock source. By default, two system clock cycles are generated per oscillator clock source cycle, but the number of oscillator clock cycles per system clock can also be increased by setting the power management mode enable (PMME) bit and the clock-divide control (CD1:CD0) bits per Table 5-3.

Table 5-3. System Clock Rate Control Settings

PMME	CD[1:0]	CYCLES PER SOURCE CLOCK
0	00	1
0	01	2 (default)
0	10	4
0	11	8
1	00	256

5.3.5 External Crystal-Fail Detection and Automatic Switchover

The MAXQ7665/MAXQ7666 have a high-frequency oscillator-fail detection circuit. An automatic clock switchover from crystal to 7.6MHz RC oscillator is forced if:

- XT = 1 (external crystal is selected as the system clock source).
- XHFRY = 1 (high-frequency oscillator warmup is complete).
- A clock failure (high-frequency source drops below 30kHz) is detected.

When the above condition has been detected, the clock circuitry will:

- Enable the internal 7.6MHz RC oscillator immediately (RCE forced to 1).
- Switch the system clock source from external to internal (XT forced to 0).
- Set the high-frequency crystal failure interrupt flag to 1 (HFFINT = 1).

The 7.6MHz RC oscillator system clock should be ready after a four-cycle delay. This switchover remains in effect until software reconfigures the clock structure. A crystal failure interrupt is generated if the high-frequency oscillator failure interrupt is enabled (HFFIE = 1). Also, for the interrupt to be acknowledged by the microcontroller interrupt logic, the interrupt request must also be enabled by the IGE bit in the IC register and IM5 mask in the IMR peripheral register. Figure 5-7 illustrates the external crystal-fail detection and automatic clock switchover flow.
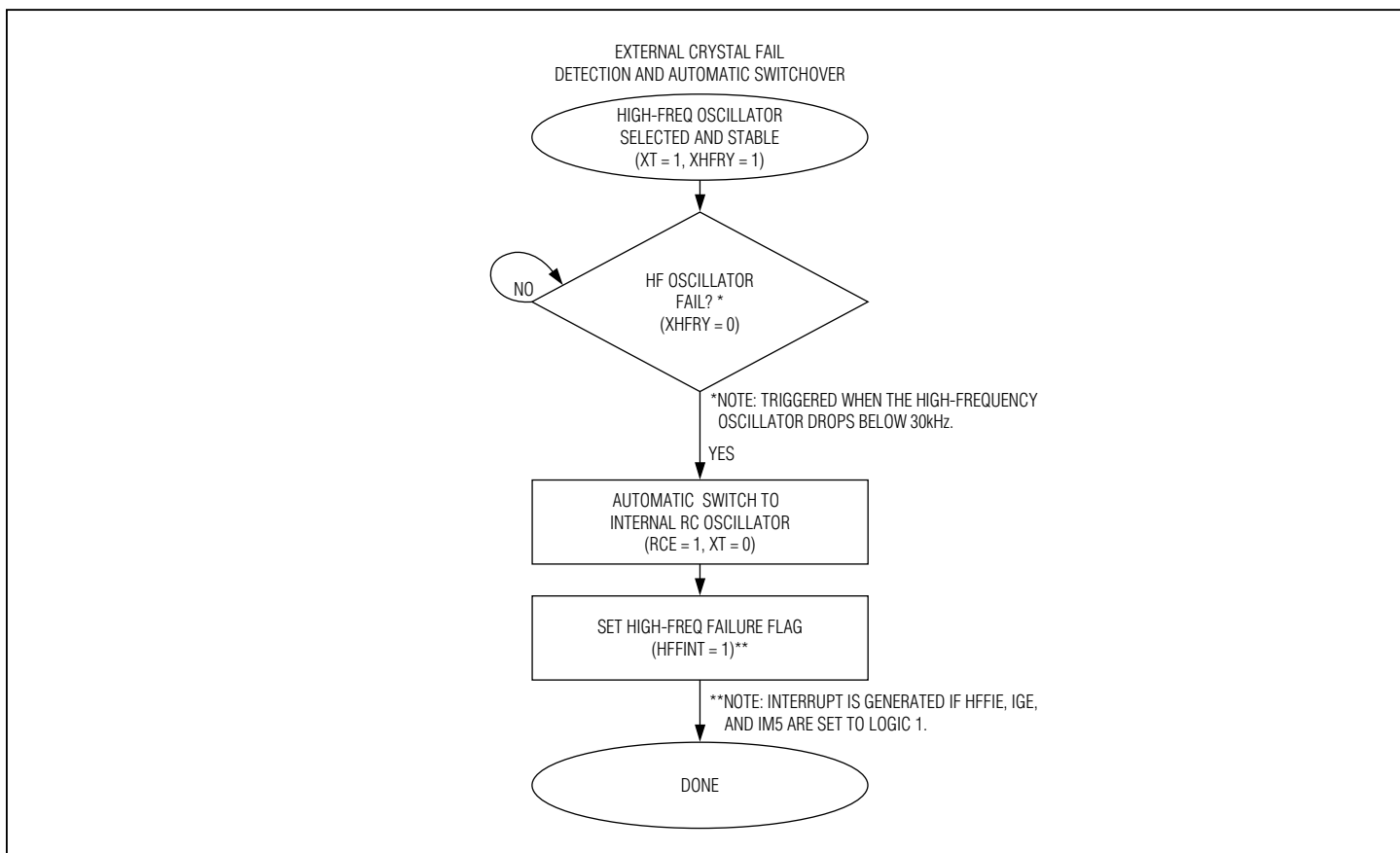

Figure 5-7. External Crystal-Fail Detection

5.4 Watchdog Timer

The watchdog timer is a user-programmable clock counter that can serve as a time-base generator, an event timer, or a system supervisor. As shown in Figure 5-8, the watchdog timer is driven by the internal 7.6MHz RC clock and is supplied to a series of dividers. If the watchdog interrupt and the watchdog reset are disabled (EWDI = 0 and EWT = 0), the watchdog timer is disabled and its input clock is gated off. Whenever the watchdog timer is disabled, the watchdog interval timer (per WD1:WD0 bits) and 512 clock reset counter will be reset if either the interrupt or reset function is enabled. When the watchdog timer is initially enabled, there will be a 1-clock to 3-clock cycle delay before it starts. The divider output is selectable and determines the interval between timeouts. When the timeout is reached, the interrupt flag WDIF is set, and if enabled, an interrupt occurs. A watchdog-reset function is also provided in addition to the watchdog interrupt. The reset and interrupt are completely discrete functions that can be acknowledged or ignored, together or separately for various applications.
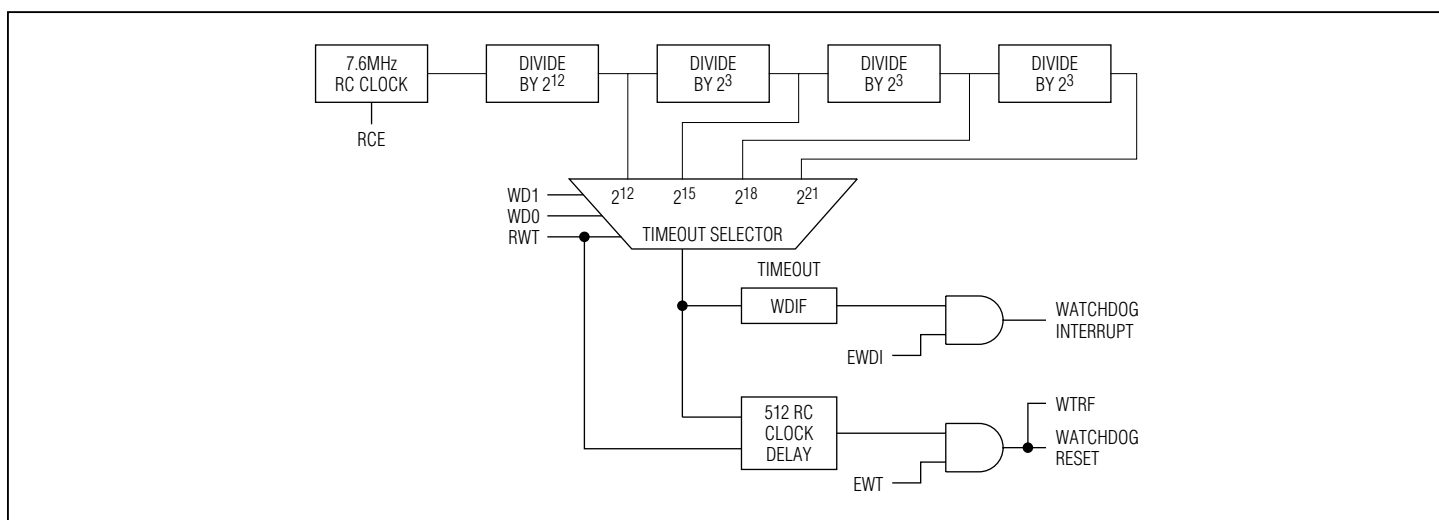

Figure 5-8. Watchdog Timer Block Diagram

The watchdog timer reset function works as follows. After initializing the correct timeout interval (discussed below), software can enable, if desired, the reset function by setting the enable watchdog timer reset (EWT = WDCN.1) bit. Setting the EWT bit will reset/restart the watchdog timer if the watchdog timer is not already enabled. At any time prior to reaching its user-selected terminal value, software can set the refresh the watchdog timer (RWT = WDCN.0). If the watchdog timer is refreshed (RWT bit written to a logic 1) before the timeout period expires, the timer will start over. Hardware will automatically clear RWT after software sets it.

If the timeout is reached without RWT being set, hardware will generate a watchdog interrupt if the interrupt source has been enabled. If no further action is taken to prevent a watchdog reset, in the 512 RC clock cycles following the timeout, hardware will reset the MAXQ7665/MAXQ7666 if EWT = 1. When the reset occurs, the watchdog timer reset flag (WTRF = WDCN.2) will automatically be set to indicate the cause of the reset; however, software must clear this bit manually after recovering from the reset.

The watchdog interrupt is also available for applications that do not need a true watchdog reset but rather a very long timer. The interrupt is enabled using the enable watchdog timer interrupt (EWDI = WDCN.6) bit. When the timeout occurs, the watchdog timer sets the WDIF bit (WDCN.3), and an interrupt occurs if the interrupt global enable (IGE = IC.0) and system interrupt mask (IMS = IMR.7) are set and the interrupt in service (INS) bit is clear. Note that WDIF is set 512 clocks before a potential watchdog reset. The watchdog interrupt flag must be cleared by software.

Note: The watchdog timer is clocked by the internal 7.6MHz RC oscillator. Therefore, RCE should be set to 1 for the watchdog timer operation. The interrupt and reset functions of the watchdog timer are summarized in Table 5-4.

Table 5-4. Interrupt and Reset Functions for Watchdog

RCE	EWT	EWDI	WDIF	OPERATION
0	x	x	x	Watchdog disabled.
1	0	0	x	Watchdog disable, clock is gated off.
1	0	1	0	Watchdog interrupt is enabled and has not occurred. Watchdog reset function disabled.
1	0	1	1	Watchdog interrupt is enabled and has occurred. Watchdog reset function disabled.
1	1	0	0	Watchdog reset function is enabled. Watchdog interrupt is disabled.
1	1	0	1	Watchdog reset function is enabled. No interrupt has been generated. Watchdog reset will occur in 512 RC cycles if RWT is not set.
1	1	1	0	Watchdog reset and interrupt functions are enabled.
1	1	1	1	Watchdog reset and interrupt functions are enabled. Watchdog interrupt has occurred. Watchdog reset will occur in 512 RC cycles if RWT is not set.

Table 5-5. Watchdog Timeout Selections

WD1	WD0	WATCHDOG TIMEOUT PERIOD			
		RC CLOCKS		MILLISECONDS (FOR RC = 7.6MHz)	
		UNTIL INTERRUPT	UNTIL RESET	UNTIL INTERRUPT	UNTIL RESET
0	0	2^{12} (default)	$2^{12} + 2^9$	0.539	0.606
0	1	2^{15}	$2^{15} + 2^9$	4.31	4.38
1	0	2^{18}	$2^{18} + 2^9$	34.49	34.56
1	1	2^{21}	$2^{21} + 2^9$	275.94	276

The watchdog timeout selection is made using bits WD1 (WDCN.5) and WD0 (WDCN.4). The watchdog has four timeout selections based on the internal RC clock as shown in Table 5-5.

Using the watchdog interrupt during software development can allow the user to select ideal watchdog reset locations. Code is first developed without enabling the watchdog interrupt or reset functions. Once the program is complete, the watchdog interrupt function is enabled to identify the required locations in code to set the RWT (WDCN.0) bit. Incrementally adding instructions to refresh the watchdog timer prior to each address location (identified by the watchdog interrupt) allows the code to eventually run without receiving a watchdog interrupt. At this point, the watchdog timer reset can be enabled without the potential of generating unwanted resets. At the same time the watchdog interrupt may also be disabled. Proper use of the watchdog interrupt and watchdog reset allows interrupt software to survey the system for errant conditions.

When using the watchdog timer as a system monitor, the watchdog reset function should be used. If the interrupt function were used, the purpose of the watchdog would be defeated. For example, assume the system is executing errant code prior to the watch dog interrupt. The interrupt would temporarily force the system back into control by vectoring the CPU to the interrupt service routine. Restarting the watchdog and exiting by an RETI or RET, would return the processor to the lost position prior to the interrupt. By using the watchdog reset function; the processor is restarted from the beginning of the program, and therefore placed into a known state.

5.5 Power Management Mode

There are two major sources of power dissipation in CMOS circuitry. The first is static dissipation caused by leakage current. The second is dynamic dissipation caused by transient switching current required to charge and discharge load capacitors, as well as short-circuit current dissipated by momentary connections between V_{DD} and ground during gate switching.

Usually, it is the dynamic switching power dissipation that dominates the total power consumption, and this power dissipation (P_D) for a CMOS circuit can be calculated in terms of load capacitance (C_L), power-supply voltage (V_{DD}), and operating frequency (f) as

$$P_D = C_L \times V_{DD}^2 \times f$$

Capacitance and supply voltage are technology dependent and relatively fixed. However, the operating frequency determines the clock rate, and the required clock rate may be different from application to application depending on the amount of processing power required. If an external crystal or oscillator is being used, the operating frequency can be adjusted by changing external components. However, it may be the case that a single application may require maximum power at sometimes and very little at others. Power management mode allows an application to reduce its clock frequency, and therefore its power consumption, under software control.

The MAXQ7665/MAXQ7666 provide the following features to assist in power management:

- Divide-by-256 (PMM) mode to reduce current consumption.
- Switchback mode to exit PMM automatically when rapid processing is required.
- Ultra-low-power stop mode.

Table 5-6 shows the system clock control register (CKCN) bits used to control power management features.

Table 5-6. System Power Management

REGISTER	ADDRESS	BIT	NAME	FUNCTION
CKCN	M8[0Eh]	1:0	CD[1:0]	Selects clock divide-by-1 (00), -2 (01), -4 (10), or -8 (11) mode. When PMM mode is enabled, selects divide-by-256 (00) mode.
		2	PMME	Selects PMM mode (when set to 1) or normal clock divide mode (when set to 0)
		3	SWB	When set to 1, enables automatic switchback from PMM (divide-by-256 mode) to normal clock divide mode under certain conditions.
		4	STOP	When set to 1, causes the processor to enter stop mode.

5.5.1 Divide-by-256 Mode (PMM)

In this power management mode, all operations continue as normal but at a reduced clock rate (the selected clock source divided by 256). This power management mode affects module clock rates as follows:

- Program execution occurs at the selected clock source rate divided by 256.
- All other functional modules (CAN, ADC, timers, UART, and SPI) operate at the selected clock source rate divided by 256.
- Watchdog timer, if enabled, continues to operate using the internal, undivided 7.6MHz RC oscillator as the clock source.

The power management mode is entered by setting the PMME bit (CKCN.2) to 1 while the CD1 and CD0 (CKCN1:CKCN0) bits are both cleared to 0. When PMM mode is exited (either by clearing the PMME bit or as a result of a switchback trigger), system operation will revert to the mode indicated by the values of the CD1 and CD0 bits, which in this case will be the standard divide-by-1 clock mode.

Note: The CD1 and CD0 (CKCN1:CKCN0) bits must both be cleared to 0 before setting the PMME bit to 1.

5.5.2 Switchback Mode

When power management mode is active, the MAXQ7665/MAXQ7666 operate at a reduced clock rate. Although execution continues as normal, peripherals that base their timing on the system clock such as the UART module and the SPI module may be unable to operate normally or at a high enough speed for proper application response. Additionally, interrupt latency is greatly increased.

The switchback feature is used to allow a processor running under power management mode to switch back to normal mode quickly under certain conditions that require rapid response. Switchback is enabled by setting the SWB bit (CKCN.3) to 1. If switchback is enabled, the MAXQ7665/MAXQ7666 running power management mode automatically clears the PMME bit to 0 and returns to normal undivided clock rate when any of the following conditions occur:

- An external interrupt condition occurs on a port pin and the corresponding external interrupt is enabled.

- An active-low transition occurs on the UART serial receive-input line (modes 1, 2, and 3) and data reception is enabled.

- The SBUF0 register is written to send an outgoing byte through the UART and transmission is enabled.

- The SPIB register is written in master mode (STBY = 1) to send an outgoing character through the SPI module and transmission is enabled.

- The SPI module's $\overline{SS}$ (slave select input) signal is asserted in slave mode.

- CAN bus activity on its data input (CANRXD) while its interrupt is enabled.

- Active debug mode is entered either by break point match or issuance of the debug command from background mode.

If any of the above conditions are true (a switchback source is active) and SWB has been set to logic 1, the PMME bit cannot be written to enter power management mode. This is to prevent the MAXQ7665/MAXQ7666 from accidentally reducing the clock rate during the service of an external interrupt or serial port activity.

5.5.3 Stop Mode

Stop mode disables all circuits within the MAXQ7665/MAXQ7666 including the watchdog timer and its clock source (the internal 7.6MHz RC oscillator). All clock sources, timers, and peripherals are halted, and no code execution occurs. The system clock is stopped, and all processing activity is halted. Once in stop mode, the MAXQ7665/MAXQ7666 are in a mostly static state, with power consumption determined mainly by leakage currents.

Stop mode is invoked by setting the STOP bit (CKCN.4) to 1. The MAXQ7665/MAXQ7666 enter stop mode immediately once the instruction that sets the STOP bit is executed. Entering stop mode does not affect the settings of the clock control bits; this allows the processor to return to its original operating frequency following stop mode removal.

The MAXQ7665/MAXQ7666 exit stop mode if any of the following conditions occur:

- An external reset signal is applied to the $\overline{RESET}$ pin.

- An external interrupt condition occurs on one of the port pins and the corresponding external interrupt is enabled.

- A CAN bus activity on its data input (CANRXD) while its interrupt is enabled.

- A brownout interrupt condition on DVDD or DVDDIO (if brownout detection and corresponding interrupt is enabled).

- A power-on/brownout reset (if brownout reset detection is enabled (VDPE = 1)).

Note that exiting stop mode via external reset or power-on/brownout reset causes the processor to undergo a normal reset cycle (see *Section 2*), as opposed to resuming execution at the point at which it entered stop mode. Exiting stop mode by means of an external interrupt or CAN bus activity interrupt causes the processor to resume execution at the instruction following the one which set the STOP bit (and then immediately vector to the interrupt service routine).

When stop mode is exited, the MAXQ7665/MAXQ7666 execution resumes as follows:

- If the internal 7.6MHz RC oscillator is selected as the system clock source (RCE = 1 and XT = 0), execution will resume using the 7.6MHz RC oscillator as the system clock source following a delay of four 7.6MHz RC oscillator cycles.

- If the high-frequency oscillator is selected as the system clock source (HFE = 1 and XT = 1), execution will resume using the internal 7.6MHz RC oscillator as the system clock source following a delay of four 7.6MHz RC oscillator cycles. After a warmup delay of 4096 high-frequency oscillator cycles, the system clock source will switch over to the high-frequency oscillator automatically.

SECTION 6: SERIAL I/O MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 6: SERIAL I/O MODULE

The MAXQ7665/MAXQ7666 serial I/O module provides access to a universal asynchronous receiver/transmitter (UART) for serial communication with framing error detection. The UART is a full-duplex communication channel capable of supporting asynchronous and synchronous data transfers. The UART allows the MAXQ7665/MAXQ7666 to conveniently communicate with other RS-232 interface-enabled devices and can support LIN-bus implementation. Except where explicitly noted, the MAXQ7665 and MAXQ7666 features are identical.

Features of the MAXQ7665/MAXQ7666 UART include:

- Asynchronous and synchronous data transfer
- Separate transmit and receive interrupts
- Framing error detection
- Baud rate based on system clock or baud-rate generator output

6.1 Architecture

The MAXQ7665/MAXQ7666 UART supports four basic modes of operation and is capable of synchronous and asynchronous communication, with different protocols and baud rates. In the synchronous mode, the microcontroller supplies the clock, and communication takes place in a half-duplex manner, while the asynchronous mode supports full-duplex operation. Table 6-1 shows the UART operating modes.

Table 6-1. UART Operation Modes

UART MODE	FUNCTION	BAUD CLOCK*	DATA BITS	START/STOP	9TH BIT FUNCTION	MAX BAUD RATE AT 8MHz
0	Synchronous	4 or 12 Clocks	8	None	None	2Mbps
1	Asynchronous	Baud Generation	8	1 Start, 1 Stop	None	250kbps
2	Asynchronous	32 or 64 Clocks	9	1 Start, 1 Stop	0, 1, Parity	250kbps
3	Asynchronous	Baud Generation	9	1 Start, 1 Stop	0, 1, Parity	250kbps

*Use of any system clock-divide modes or power management mode affects the baud clock.

See Figure 6-1 for a simplified functional block diagram of the MAXQ7665/MAXQ7666 UART in synchronous mode. Serial I/O occurs on the receive pin, which behaves as a bidirectional data line, and the shift clock is provided on the TXD pin. The MAXQ7665/MAXQ7666 UART in asynchronous mode is shown in Figure 6-2. In asynchronous mode, the UART is a full-duplex communication channel with a programmable baud-rate generator.

The MAXQ7665/MAXQ7666 UART has a control register (SCON0) and a transmit/receive register (SBUF0). The SBUF0 location provides access to both transmit and receive registers, where a read is directed to the receive buffer and a write is directed to the transmit buffer. There is a holding buffer that allows the UART to receive an incoming word before software has read the previous one. The UART baud clock is generated by the baud-rate generator or based directly on the system clock.
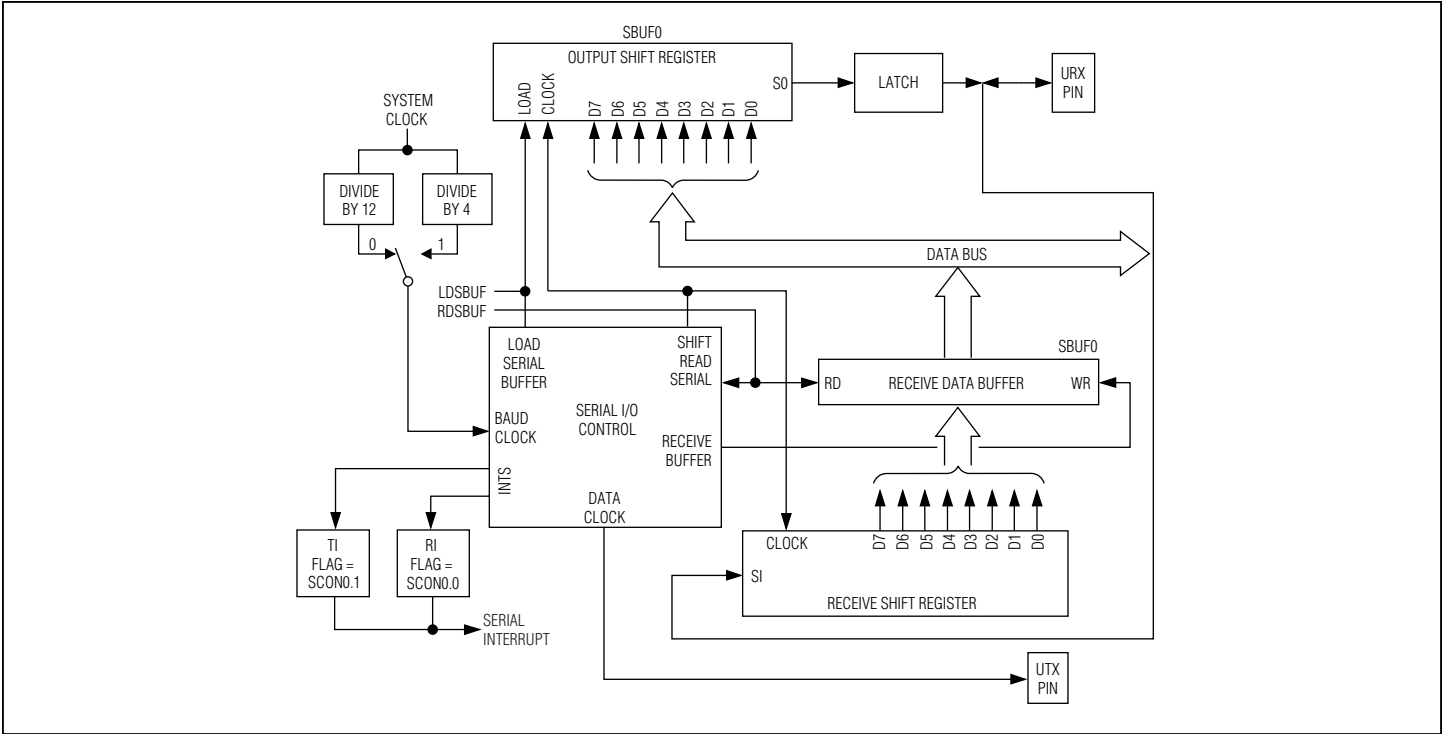
Figure 6-1. UART Synchronous Mode (Mode 0)
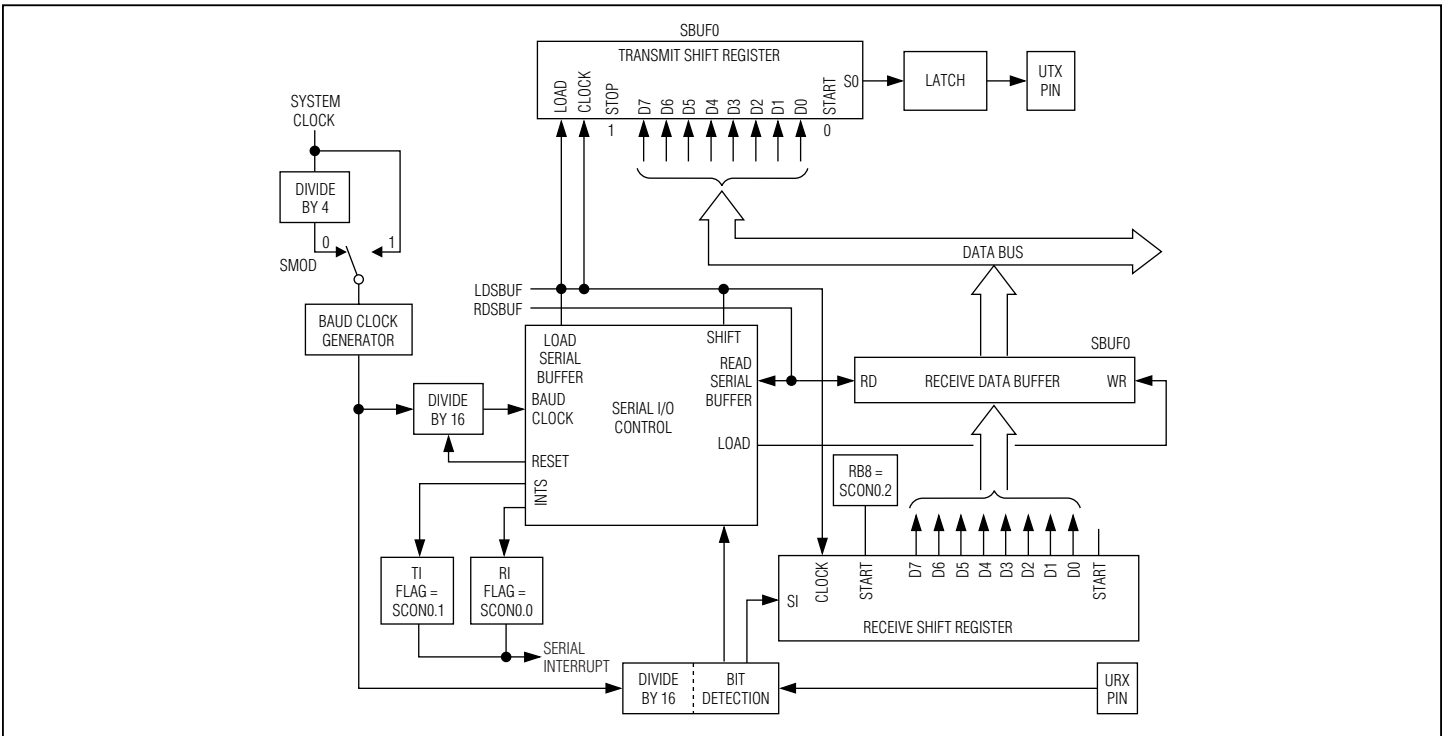
Figure 6-2. UART Asynchronous Mode (Mode 1)

6.1.1 UART Pins

The MAXQ7665/MAXQ7666 UART supports dedicated transmit and receive pins as described in Table 6-2.

Table 6-2. MAXQ7665/MAXQ7666 UART Pins

UART EXTERNAL SIGNAL	PIN NUMBER		FUNCTION
	48-PIN	56-PIN	
UTX	22	25	UART Transmitter Output. This signal is the transmit output from the UART. In synchronous mode, the shift clock is output on this pin.
URX	23	26	UART Receiver Input: This signal is the receive input for the UART. In synchronous mode, this pin behaves as a bidirectional data line.

6.2 UART Registers

The MAXQ7665/MAXQ7666 UART peripheral registers are described here.

6.2.1 Serial Port 0 Control Register (SCON0)

Register Description: **Serial Port 0 Control Register**
Register Name: **SCON0**
Register Address: **Module 00h, Index 1Dh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved.

Bit 7: Serial Port Mode Bit 0/Framing Error Flag (SM0/FE). When FEDE (SMD0.0) is set to 1, this bit is the framing error flag that is set upon detection of an invalid stop bit. It must be cleared by software. Modification of this bit when FEDE is set has no effect on the serial mode. This bit functions as the serial port mode bit 0 when FEDE is 0. SM0 is used in conjunction with the SM2 and SM1 bits to define the serial mode as shown in the *Serial Mode Definition* table.

Serial Mode Definition

UART MODE	SM2	SM1	SM0	FUNCTION	LENGTH (BITS)	PERIOD
0	0	0	0	Synchronous	8	12 System Clock
0	1	0	0	Synchronous	8	4 System Clock
1	X	1	0	Asynchronous	10	64/16 Baud Clock (SMOD = 0/1)
2	0	0	1	Asynchronous	11	64/32 System Clock (SMOD = 0/1)
2	1	0	1	Asynchronous (MP)	11	64/32 System Clock (SMOD = 0/1)
3	0	1	1	Asynchronous	11	64/16 Baud Clock (SMOD = 0/1)
3	1	1	1	Asynchronous (MP)	11	64/16 Baud Clock (SMOD = 0/1)

Bit 6: Serial Port Mode Bit 1 (SM1). See the *Serial Mode Definition* table.

Bit 5: Serial Port Mode Bit 2 (SM2). Setting this bit in mode 1 ignores received data if an invalid stop bit is detected. Setting this bit in mode 2 or 3 enables multiprocessor communications, and prevents the RI bit from being set and the interrupt from being asserted if the 9th bit received is 0. See the *Serial Mode Definition* table. This bit also used to support mode 0 for clock selection.

SM2 = 0: System clock is divided by 12.
SM2 = 1: System clock is divided by 4.

Bit 4: Receive Enable (REN)

REN = 0: Serial port 0 receiver disabled.
REN = 1: Serial port 0 receiver enabled for modes 1, 2 and 3. Initiate synchronous reception for mode 0.

Bit 3: 9th Transmission Bit State (TB8). This bit identifies the state of the 9th transmission bit in serial port modes 2 and 3.

Bit 2: 9th Received Bit State (RB8). This bit identifies the state of the 9th bit of received data in serial port modes 2 and 3. When SM2 is 0, it is the state of the stop bit in mode 1. This bit has no meaning in mode 0.

Bit 1: Transmit Interrupt Flag (TI). This bit indicates that the data in the serial port data buffer has been completely shifted out. It is set at the end of the last data bit for all modes of operation and must be cleared by software once set.

Bit 0: Receive Interrupt Flag (RI). This bit indicates that a data byte has been received in the serial port buffer. The bit is set at the end of the 8th bit for mode 0, after the last sample of the incoming stop bit for mode 1 subject to the value of the SM2 bit, or after the last sample of RB8 for modes 2 and 3. This bit must be cleared by software once set.

6.2.2 Serial Port 0 Mode Register (SMD0)

Register Description: **Serial Port 0 Mode Register**
Register Name: **SMD0**
Register Address: **Module 00h, Index 1Eh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	ESI	SMOD	FEDE
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	rw	rw

r = read, w = write

Bits 15 to 3: Reserved.

Bit 2: Enable Serial Port Interrupt (ESI). Setting this bit to 1 enables interrupt requests generated by the RI or TI flags in SCON0. Clearing this bit to 0 disables the serial port interrupt. **Note:** For interrupt requests to happen, global interrupt mask bits IM0 (in the IMR register) and IGE (in the IC peripheral register) must also be enabled.

Bit 1: Serial Port Baud Rate Select (SMOD). The SMOD bit selects the final baud rate for the asynchronous mode:

SMOD = 1: 16 times the baud clock for modes 1 and 3 (32 times the system clock for mode 2).
SMOD = 0: 64 times the baud clock for modes 1 and 3 (64 times the system clock for mode 2).

Bit 0: Framing Error Detection Enable (FEDE). This bit selects the function of the SM0/FE (SCON0.7) bit. **Note:** The information for bits SM0 and FE are actually stored in different registers. Changing FEDE only modifies which register is accessed, not the contents of either.

FEDE = 0: SM0/FE bit functions as SM0 for serial port mode selection.
FEDE = 1: SM0/FE is converted to the framing error (FE) flag.

6.2.3 Phase Register 0 Register (PR0)

Register Description: **Phase Register 0**
Register Name: **PR0**
Register Address: **Module 00h, Index 1Fh**

Bit #	15	14	13	12	11	10	9	8
Name	PR0.15	PR0.14	PR0.13	PR0.12	PR0.11	PR0.10	PR0.9	PR0.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	PR0.7	PR0.6	PR0.5	PR0.4	PR0.3	PR0.2	PR0.1	PR0.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Phase Register 15 to 0 (PR0.15 to PR0.0). This register is used to load and read the 16-bit value in the phase register that determines the baud rate of the serial port 0.

6.2.4 Serial Port 0 Data Buffer Register (SBUF0)

Register Description: **Serial Port 0 Data Buffer Register**
Register Name: **SBUF0**
Register Address: **Module 00h, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	SBUF0.7	SBUF0.6	SBUF0.5	SBUF0.4	SBUF0.3	SBUF0.2	SBUF0.1	SBUF0.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved.

Bits 7 to 0: Serial Port 0 Data Buffer Bits 7 to 0 (SBUF0.7 to SBUF0.0). Data for serial port 0 is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.

6.3 Modes of Operation

A detailed description of the MAXQ7665/MAXQ7666 UART modes is given in this section.

6.3.1 UART Mode 0

This mode is used to communicate in synchronous, half-duplex format with devices that accept the MAXQ7665/MAXQ7666 micro-controller as a master. Figure 6-3 shows a functional block diagram and basic timing of this mode. As can be seen, there is one bidi-rectional data line (URX) and one shift clock line (UTX) used for communication. Mode 0 requires that the MAXQ7665/MAXQ7666 be the master since it generates the serial shift clock for data transfers that occur in either direction.

The URX signal is used for both transmission and reception. Data bits enter and exit least significant bit first. The UTX pin provides the shift clock. The baud rate is equal to the shift clock frequency. When not using power management mode, the baud rate in mode 0 is equivalent to the system clock divided by either 12 or 4, as selected by the SM2 bit in the SCON0 register.

The UART begins transmitting when a write is performed to SBUF0. The internal shift register then begins to shift data out. The clock is activated and transfers data until the 8-bit value is complete. Data is presented one clock prior to the falling edge of the shift clock (UTX) so that an external device can latch the data using the rising edge of the shift clock.

The UART begins to receive data when the REN bit in the SCON0 register (SCON0.4) is set to logic 1 and the RI bit (SCON0.0) is set to logic 0. This condition indicates that there is data to be shifted in on the URX pin. The shift clock (UTX) is activated and data is latched on the rising edge. The external device should, therefore, present data on the falling edge. This process continues until all 8 bits have been received. The RI bit is automatically set to logic 1, one clock cycle following the last rising edge of the shift clock on UTX. This causes reception to stop until SBUF0 has been read and the RI bit cleared. When RI is cleared, another byte can be shift-ed in, if available.

6.3.2 UART Mode 1

This mode provides asynchronous, full-duplex communication. A total of 10 bits is transmitted, consisting of a start bit (logic 0), 8 data bits, and 1 stop bit (logic 1), as illustrated in Figure 6-4. The data is transferred least significant bit first. The baud rate is programma-ble through the baud-clock generator and is discussed in *Section 6.4.*

Following a write to SBUF0, the UART begins transmission five clock cycles after the first baud clock from the baud-clock generator. Transmission takes place on the UTX pin. It begins with the start bit being placed on the pin. Data is then shifted out onto the pin, least significant bit first. The stop bit follows. The TI bit is set by hardware after the stop bit is placed on the pin. All bits are shifted out at the rate determined by the baud-clock generator.

Once the baud-clock generator is active, reception can begin at any time. The REN bit (SCON0.4) must be set to logic 1 to allow recep-tion. The detection of a falling edge on the URX pin is interpreted as the beginning of a start bit, and will begin the reception process. Data is shifted in at the selected baud rate. At the middle of the stop bit time, certain conditions must be met to load SBUF0 with the received data in the receive shift register:

<div align="center">

RI must = 0, and either

if SM2 = 0, the state of the stop bit does not matter

or

if SM2 is 1, the state of the stop bit must be 1

</div>

If these conditions are true, SBUF0 will be loaded with the received byte, the RB8 bit (SCON0.2) is loaded with the stop bit, and the RI bit (SCON0.0) is set. If these conditions are false, then the received data is lost (SBUF0 and RB8 not loaded) and RI is not set. Regardless of the receive word status, after the middle of the stop bit time, the receiver goes back to looking for a 1-to-0 transition on the URX pin.

Each data bit received is sampled on the 7th, 8th, and 9th clock used by the divide-by-16 counter. Using majority voting, two equal samples out of the three determine the logic level for each received bit. If the start bit was determined to be invalid (= 1), then the receive logic goes back to looking for a 1-to-0 transition on the URX pin in order to start the reception of data.
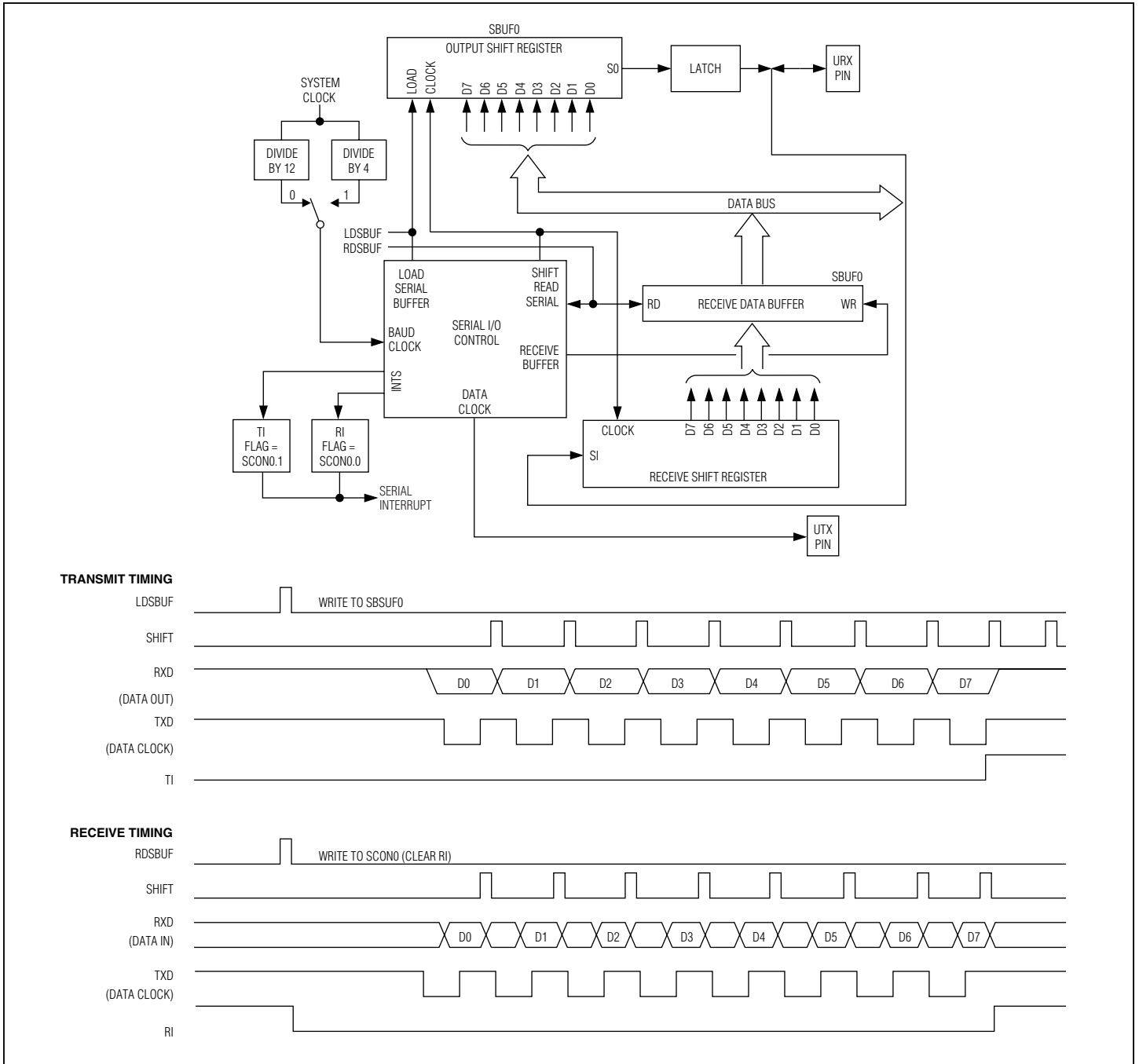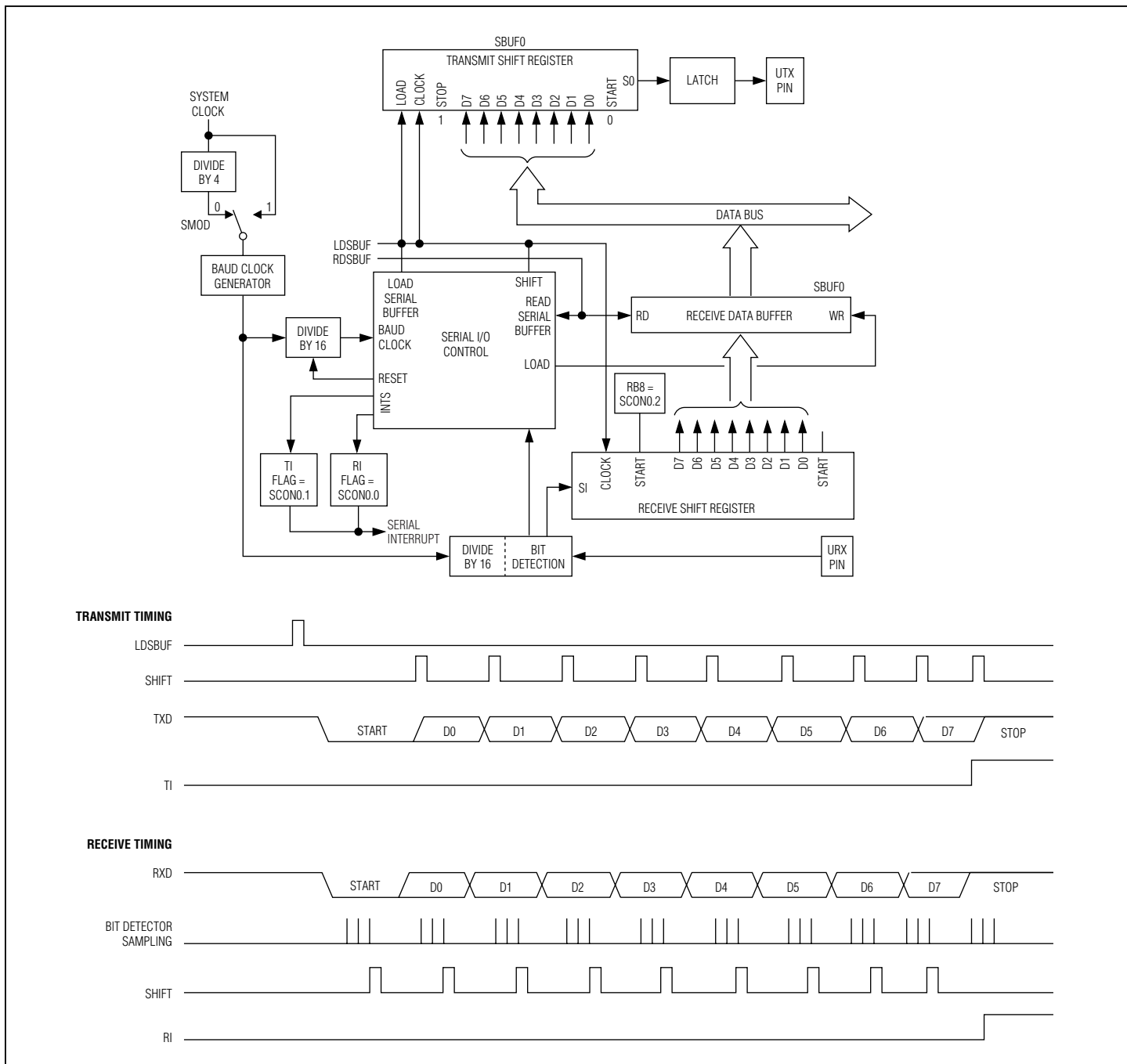
Figure 6-3. UART Mode 0

Figure 6-4. UART Mode 1

6.3.3 UART Mode 2

This mode uses a total of 11 bits in asynchronous, full-duplex communication as illustrated in Figure 6-5. The 11 bits consist of one start bit (logic 0), 8 data bits, a programmable 9th bit, and one stop bit (logic 1). Like mode 1, the transmissions occur on the UTX signal pin and receptions on URX.

For transmission purposes, the 9th bit can be stuffed as logic 0 or 1. A common use is to put the parity bit in this location. The 9th bit is transferred from the TB8 bit position in the SCON0 register following a write to SBUF0 to initiate a transmission. The UART transmission begins five clock cycles after the first rollover of the divide-by-16 counter following a software write to SBUF0. It begins with the start bit being placed on the UTX pin. The data is then shifted out onto the pin, least significant bit first, followed by the 9th bit, and finally the stop bit. The TI bit is set when the stop bit is placed on the pin.

Once the baud-rate generator is active and the REN bit has been set to logic 1, reception can begin at any time. Reception begins when a falling edge is detected as part of the incoming start bit on the URX pin. The URX pin is then sampled according to the baud rate speed. The 9th bit is placed in the RB8 bit location of the SCON0 register. At the middle of the 9th bit time, certain conditions must be met to load SBUF0 with the received data:

RI must = 0, and either

if SM2 = 0, the state of the 9th bit does not matter

or

if SM2 is 1, the state of the 9th bit must be 1

If these conditions are true, SBUF0 will be loaded with the received byte, the RB8 bit is loaded with the 9th bit, and the RI bit is set. If these conditions are false, then the received data is lost (SBUF0 and RB8 not loaded) and RI is not set. Regardless of the receive word status, after the middle of the stop bit time, the receiver goes back to looking for a 1-to-0 transition on the URX pin.

Data is sampled in a similar fashion to mode 1 with the majority voting on three consecutive samples. Mode 2 uses the sample divide-by-16 counter with either the system clock divided by 2 or 4, thus resulting in a baud clock of either system clock/32 or system clock/64.

6.3.4 UART Mode 3

This mode has the same operation as mode 2, except for the baud-rate source. As shown in Figure 6-6, mode 3 generates the baud rates through the baud-clock generator. The bit shifting and protocol are the same. The baud-clock generator is discussed in *Section 6.4*.
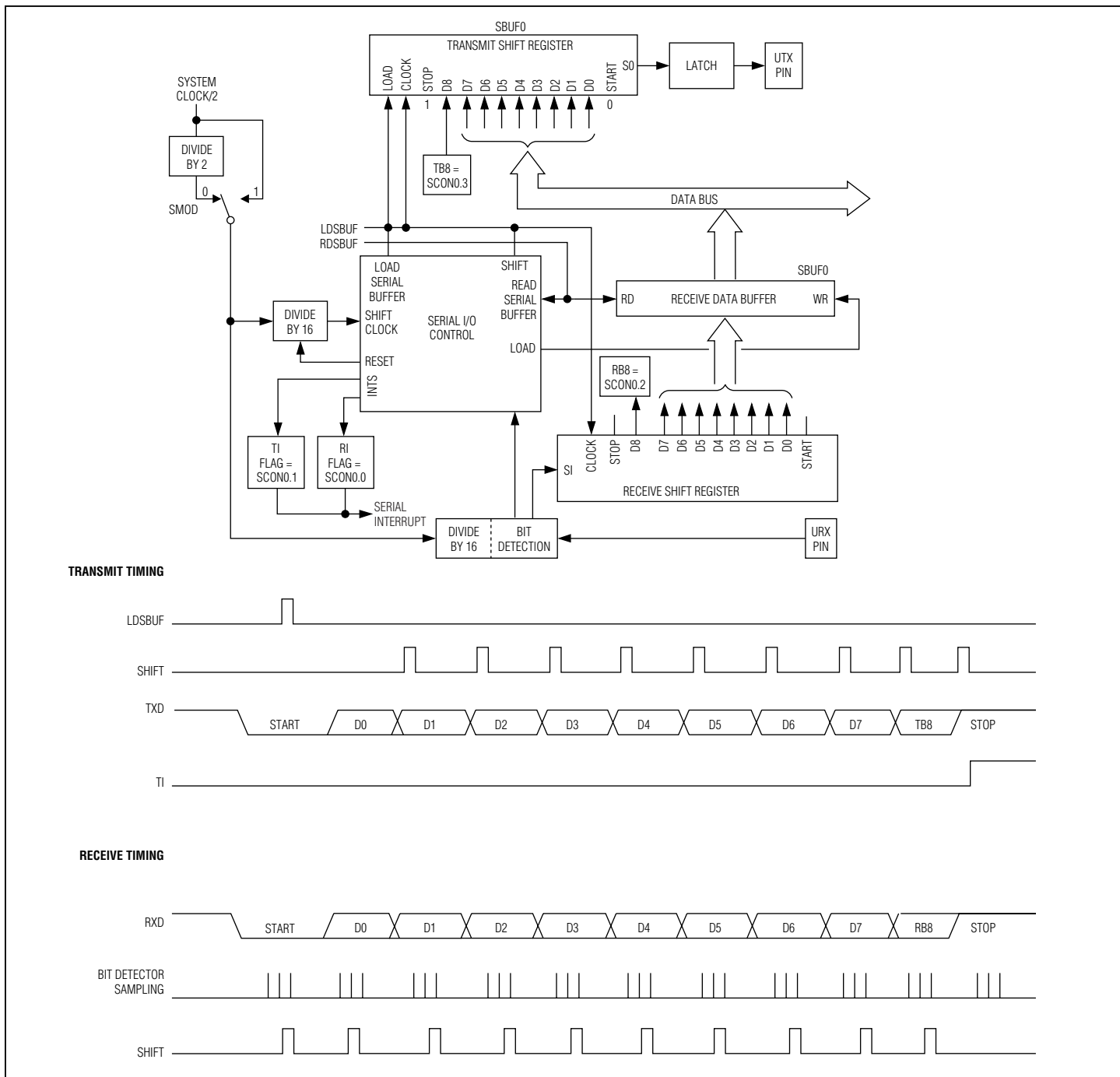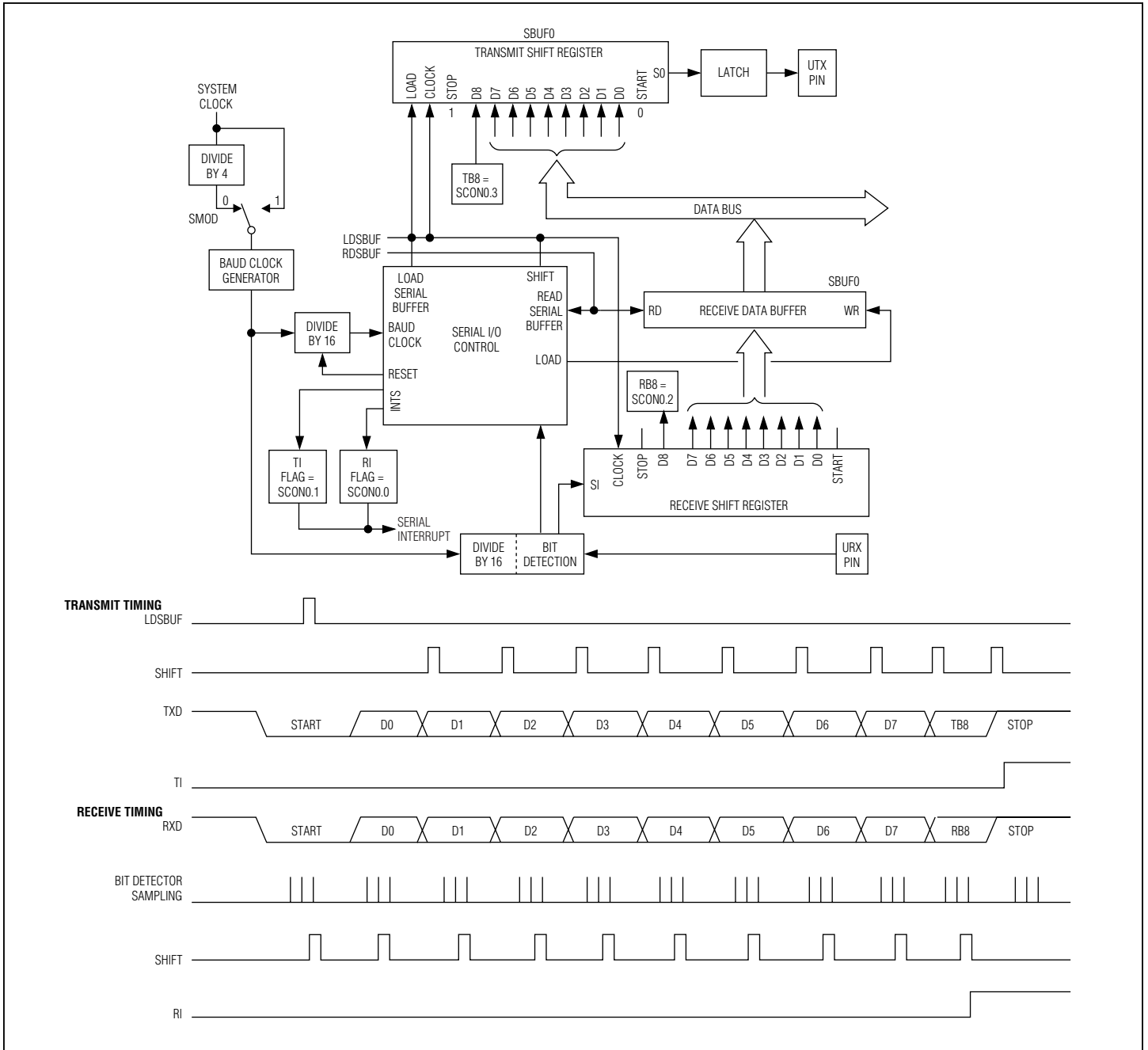
Figure 6-5. UART Mode 2

Figure 6-6. UART Mode 3

MAXQ7665/MAXQ7666 User's Guide

6.4 Baud-Rate Generation

Each mode of operation has a baud-rate generation technique associated with it. The baud-rate generation is affected by certain user options such as the power management mode enable (PMME) bit, serial mode 2 (SM2) bit, and baud-rate doubler (SMOD) bit. Table 6-3 summarizes the effects of the various user options on the UART baud clock.

Table 6-3. UART Baud-Clock Summary

SYSTEM CLOCK MODE	MODE 0		MODE 2		MODES 1, 3*	
	SM2 = 0	SM2 = 1	SMOD = 0	SMOD = 1	SMOD = 0	SMOD = 1
Divide by 1	CLK/12	CLK/4	CLK/64	CLK/32	BAUD/64	BAUD/16
Divide by 2 (default)	CLK/24	CLK/8	CLK/128	CLK/64	BAUD/64	BAUD/16
Divide by 4	CLK/48	CLK/16	CLK/256	CLK/128	BAUD/64	BAUD/16
Divide by 8	CLK/96	CLK/32	CLK/512	CLK/256	BAUD/64	BAUD/16
Power Management Mode (Divide by 256)	CLK/3072	CLK/1024	CLK/16384	CLK/8192	BAUD/64	BAUD/16

*The baud frequency is determined by the baud-clock generator.

6.4.1 Mode 0 Baud Rate

Baud rates for mode 0 are driven directly from the system clock divided by either 12 or 4, with the default case being divide-by-12. The user can select the shift clock frequency using the SM2 bit in the SCON0 register. When SM2 is set to logic 0, the baud rate is fixed at divide-by-12 of the system clock. When SM2 is set to logic 1, the baud rate is divide-by-4 of the system clock.

$$\text{Mode 0 Baud Rate} = \text{System Clock Frequency} \times (3^{SM2} / 12)$$

6.4.2 Mode 2 Baud Rate

In this asynchronous mode, baud rates are also generated from the system clock source. The user can effectively double the UART baud-clock frequency by setting the SMOD bit to logic 1. The SMOD bit is set to logic 0 on all resets, thus making divide-by-64 the default setting. The baud rate is given by the following formula:

$$\text{Mode 2 Baud Rate} = \text{System Clock Frequency} \times (2^{SMOD} / 64)$$

6.4.3 Mode 1 or 3 Baud Rate

These asynchronous modes are commonly used for communication with PCs, modems, and other similar interfaces. The baud rates are programmable using the baud-rate generator in the UART module. The baud-clock generator is basically a phase accumulator that generates a baud clock as the result of phase overflow into the most significant bit of the phase shifter. This baud-clock generator is driven by the system clock or system clock divided by 4 (depending upon the state of the SMOD bit). The baud-clock generator output is always divided by 16 to generate the exact baud rate.

6.4.4 Baud-Clock Generator

The baud-clock generator is basically a phase accumulator that produces a baud clock as the result of phase overflow from the most significant bit of the phase shift circuitry. As illustrated in Figure 6-7, a user-programmable 16-bit phase register (PR0) is used to select a suitable phase value for its baud clock. The phase value dictates the phase period of the accumulation process. The phase value (from PR0) is added to the current phase accumulator value on each system clock (SMOD = 1) or every 4th system clock (SMOD = 0). The baud clock is the result of the addition overflow out of the most significant bit of the phase accumulator (bit 16). The baud-clock generator output is always divided by 16 to produce the exact baud rate.
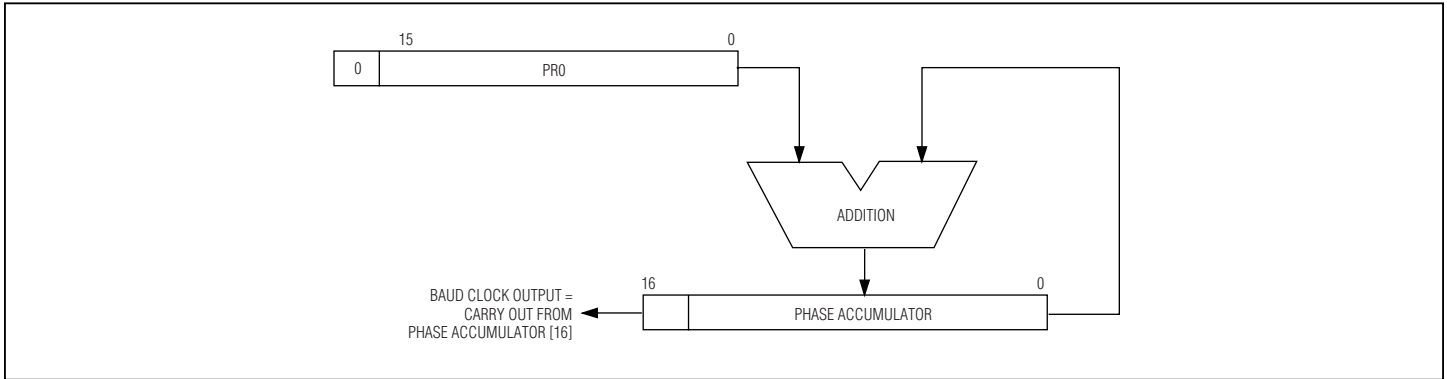
Figure 6-7. UART Baud-Clock Generator

The following two formulas can be used to calculate the output of the baud-clock generator and the resultant mode 1, 3 baud rates. Additionally, Table 6-4 gives example phase register (PR0) settings needed to produce some more common baud rates at certain system clock frequencies (assuming SMOD = 1).

$$\text{Baud-Clock Generator Output (BAUD)} = \text{System Clock Frequency} \times PR0 / 2^{17}$$

$$\text{Baud Rate for Modes 1 and 3} = \text{BAUD} \times 2^{(\text{SMOD} \times 2)} / 2^{6}$$

Table 6-4. Example Baud-Clock Generator Settings (SMOD = 1)

SYSTEM CLOCK FREQUENCY (MHz)	BAUD RATE	PR0 SETTINGS	SYSTEM CLOCK FREQUENCY (MHz)	BAUD RATE	PR0 SETTINGS
8	115,200	75F7h	3.579545	115,200	83D2h
	57,600	3AFBh		19,200	2BF1h
	19,200	13A9h		9600	15F8h
	9600	09D5h		2400	057Eh
	2400	0275h	2.4576	57,600	C000h
3.6864	115,200	FFFFh		19,200	4000h
	57,600	8000h		9600	2000h
	19,200	2AABh		2400	0800h
	9600	1555h	1	19,200	9D49h
	2400	0555h		9600	4EA5h
				2400	13A9h

6.5 Framing Error Detection

A framing error occurs when a valid stop bit is not detected. This results in the possible improper reception of the serial word. The UART can detect a framing error and notify the software. Typical causes of framing errors are noise and contention. The framing error condition is reported in the SCON0 register for the UART.

The framing error bit, FE, is located in SCON0.7. Note that this bit normally serves as SM0 and is described as SM0/FE0 in the register description. Framing error information is made accessible by the FEDE (framing error detection enable) bit located at SMD0.0. When FEDE is set to logic 1, the framing error information is shown in SM0/FE (SCON0.7). When FEDE is set to logic 0, the SM0 function is accessible. The information for bits SM0 and FE is actually stored in different registers. Changing FEDE only modifies which register is accessed, not the contents of either.

The FE bit is set to 1 when a framing error occurs. It must be cleared by software. Note that the FEDE state must be 1 while reading for writing the FE bit. Also note that receiving a properly framed serial word does not clear the FE bit. This must be done in software.

6.6 Serial UART Example: Asynchronous 10-Bit Output at 115,200 Baud

```
        move SCON0.6, #1                ; Set to mode 1 (10-bit asynchronous)
        move SCON0.4, #1                ; Enable receiver
        move SMD0.1, #1                 ; Baud rate = 16 x baud clock
        move PR0, #75F7h                ; PR0 = 2^21 * 115200 / 8MHz (crystal)

        move SCON0.0, #0                ; Clear received character flag
        move SCON0.1, #0                ; Clear transmit character flag

Loop1:
        move Acc, #'0'                  ; Start with '0' character
        move LC[ 0], #10                ; Transmit from '0' - '9'
Loop2:
        move SBUF0, Acc                 ; Send character
Transmit:
        move C, SCON0.1                 ; check transmit flag
        jump NC, Transmit               ; wait for transmit to complete
        move SCON0.1, #0                ; clear transmit flag
        add  #1                         ; increment character by 1
        djnz LC[ 0], Loop2
        jump Loop1
```

SECTION 7: TYPE 2 TIMER/COUNTER MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 7: TYPE 2 TIMER/COUNTER MODULE

The MAXQ7665/MAXQ7666 microcontrollers have three Type 2 timer/counter modules. The Type 2 timer/counter is an auto-reload 16-bit timer/counter with the following functions:

- 8-bit/16-bit timer/counter
- Up/down auto-reload
- Counter function of external pulse
- Capture
- Compare

The three Type 2 timer/counter modules supported in MAXQ7665/MAXQ7666 are referred to as timer 0, timer 1, and timer 2 in this document. To simplify the discussion, the generic notation x is appended to register and pin names to denote the MAXQ7665/MAXQ7666 timer/counter module they belong to (x = 0, 1, and 2). Except where explicitly noted, the MAXQ7665 and MAXQ7666 support identical features.

7.1 Architecture

The Type 2 timer/counter module is operable as a single 16-bit timer/counter or as a dual 8-bit timer/counter. In 16-bit mode, the timer is composed of three registers: T2Vx, T2Rx, and T2Cx. The T2Vx register is a 16-bit register that holds the current timer value. The reload value for the timer is held in the 16-bit T2Rx register. The T2Cx register is a 16-bit register that holds the compare value when operating in compare mode and gets the capture value when operating in capture mode. When operating in 16-bit mode (T2MD = 0), the full 16 bits of registers T2Vx, T2Rx, and T2Cx are read/write accessible.

When T2MD = 1, each 16-bit register associated with the Type 2 timer is split into separate upper and lower 8-bit registers to support dual 8-bit timers. Thus, the primary 8-bit timer is composed of T2Hx (value), T2RHx (reload), and T2CHx (capture/compare), and the secondary 8-bit timer is composed of T2Lx (value), T2RLx (reload), and T2CLx (capture/compare). In the dual 8-bit mode, the upper bytes of T2Vx, T2Rx, and T2Cx are inaccessible and always reads 00h. Separate T2Hx, T2RHx, and T2CHx registers are provided to facilitate high-byte access for dual 8-bit mode.

Note: For convenience, the lower byte of T2Vx (T2Vx.7–T2Vx.0) is referred to as T2Lx. Unlike T2Hx, there is no separate T2Lx register and the low byte is always accessed through T2Vx. Similarly, the lower byte of T2Rx (T2Rx.7–T2Rx.0) is referred to as T2RLx and the lower byte of T2Cx (T2Cx.7–T2Cx.0) is referred to as T2CLx. There are no separate T2RLx and T2CLx registers.

The input clock for the Type 2 timer is defined as the system clock divided by the ratio specified by the T2DIV2:T2DIV0 prescale bits. Two of the three Type 2 timers in the MAXQ7665/MAXQ7666 are connected to device pins as detailed in Table 7-1.

Figure 7-1 shows a simplified functional block diagram of a MAXQ7665/MAXQ7666 Type 2 timer/counter module in 16-bit mode. Figure 7-2 shows a MAXQ7665/MAXQ7666 Type 2 timer/counter in dual 8-bit mode. The input and output conditioning shown in the block diagrams is selected in the status/control registers T2CNAx (Type 2 timer/counter control register A), T2CNBx (Type 2 timer/counter control register B), and T2CFGx (Type 2 timer/counter configuration register) in addition to other functionalities. See *Section 7.2* for detailed discussion of these registers.
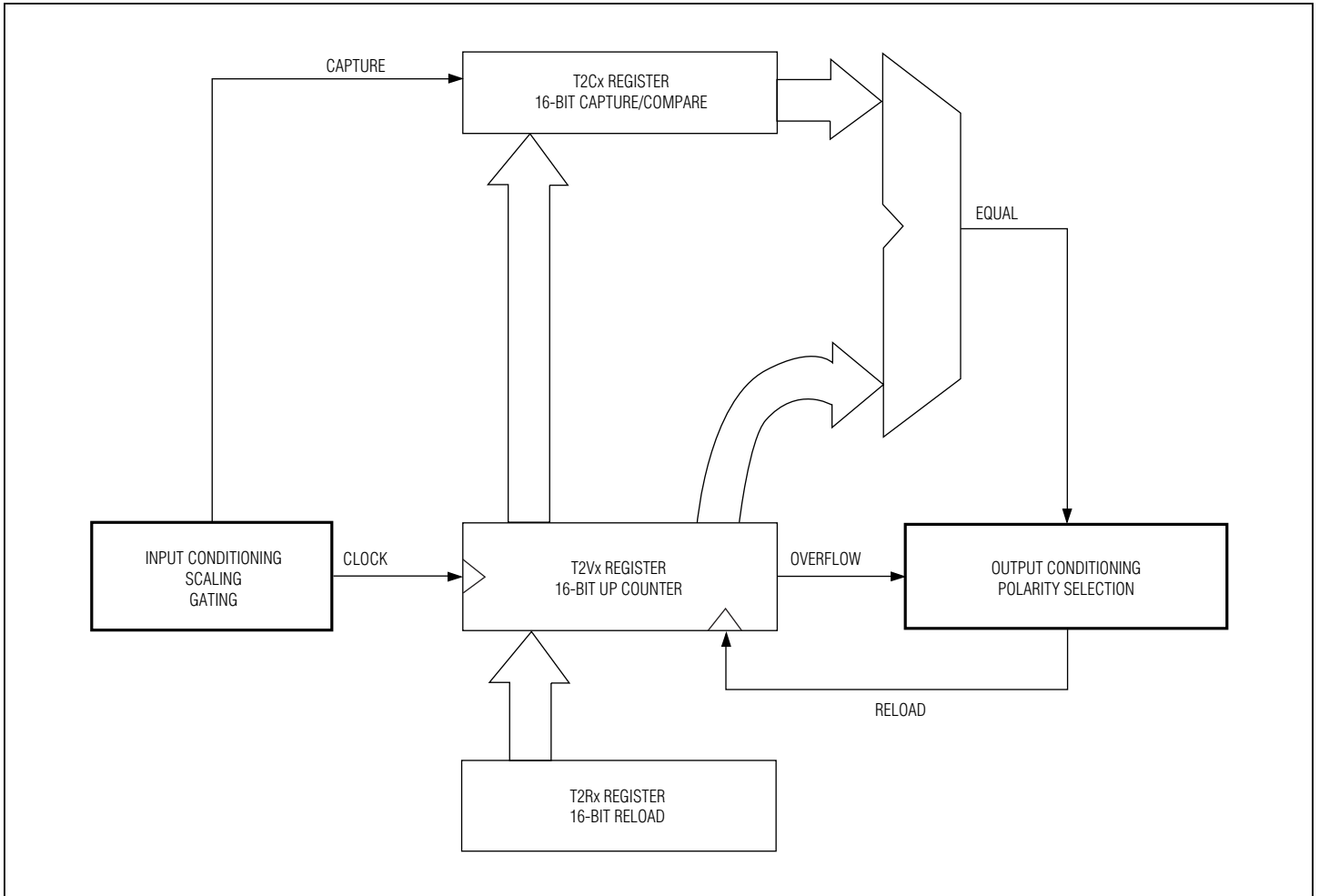
Figure 7-1. Type 2 Timer/Counter in 16-Bit Mode
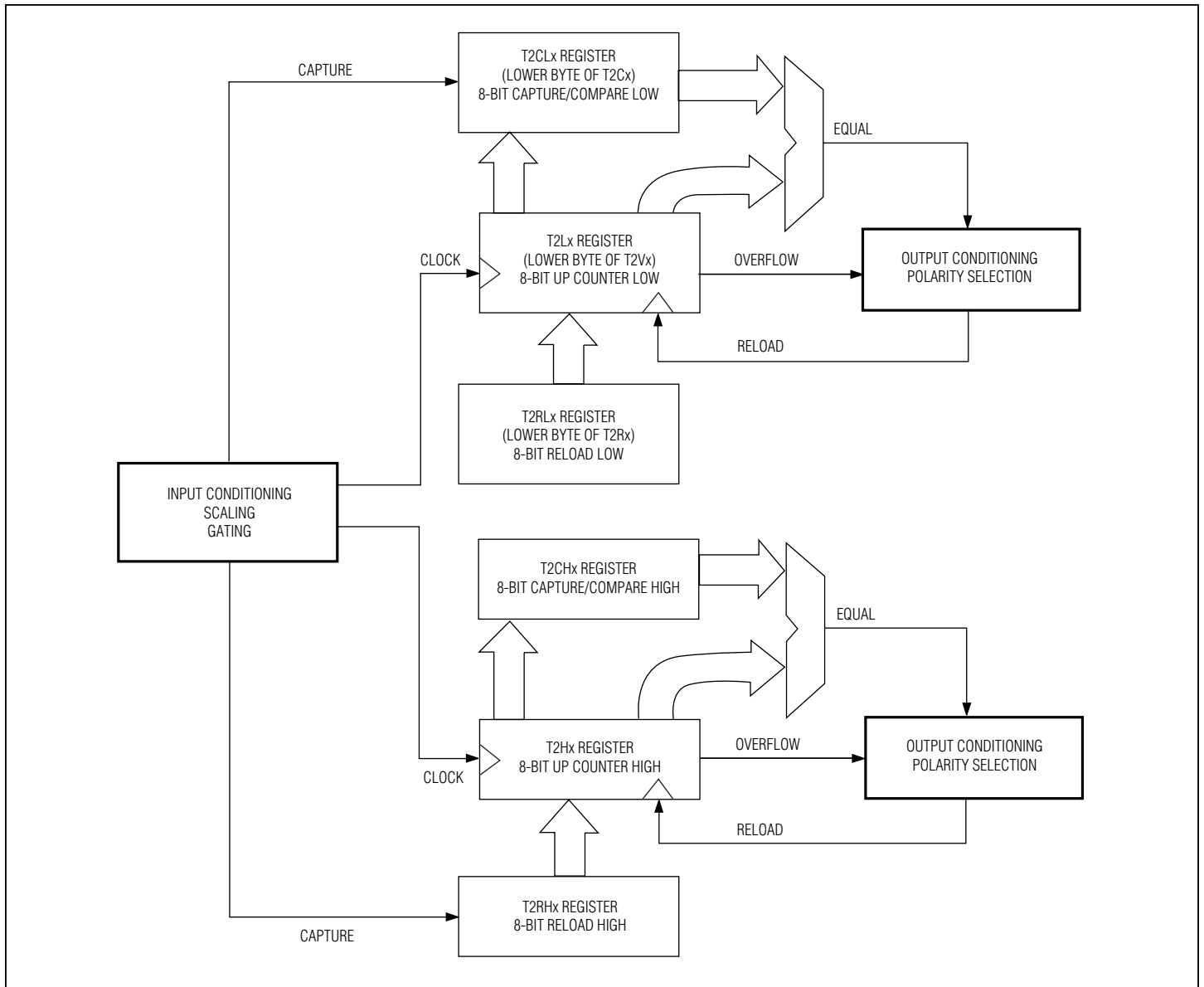
Figure 7-2. Type 2 Timer/Counter in 8-Bit Mode

7.1.1 Type 2 Timer/Counter I/O Pins

Each Type 2 timer/counter module normally supports one primary input/output pin that is referred to as Tx. Table 7-1 describes the pin assignments for the three MAXQ7665/MAXQ7666 timer/counter modules.

Table 7-1. Type 2 Timer/Counter Input and Output Pins

TIMER/COUNTER EXTERNAL SIGNAL	PIN		MULTIPLEXED WITH PORT PIN	FUNCTION
	48	56		
T0—Timer 0 Input/Output	24	27	P0.6	Timer 0 Input/Output. T0 is shared with GPIO port P0 bit 6. As timer 0 input/output, the pin supports clock gating, capture/compare, counter, and PWM functionalities.
T1—Timer 1 Input/Output	25	29	P0.7	Timer 1 Input/Output. T1 is shared with GPIO port P0 bit 7. As timer 1 input/output, the pin supports clock gating, capture/compare, counter, and PWM functionalities.

Note: In the MAXQ7665/MAXQ7666, the timer 2 input/output signal, T2, is not supported on the 48- and 56-pin packages. Thus, timer 2 can serve only as an internal timer.

7.2 Type 2 Timer/Counter Peripheral Registers

The MAXQ7665/MAXQ7666 provide three Type 2 timer/counter modules: timer 0, timer 1, and timer 2. Table 7-2 shows the associated peripheral registers for these timer/counter modules.

Table 7-2. Type 2 Timer/Counter Peripheral Registers

TIMER/COUNTER REGISTER	ADDRESS			FUNCTION
	TIMER 0	TIMER 1	TIMER 2	
Configuration Register	T2CFG0 M2[10h]	T2CFG1 M2[11h]	T2CFG2 M3[10h]	Controls counter/timer select, capture/compare function select, 8-bit/16-bit mode select, and clock divide modes
Control Register A	T2CNA0 M2[0h]	T2CNA1 M2[04h]	T2CNA2 M3[0h]	I/O settings, run enables, polarity modes
Control Register B	T2CNB0 M2[08h]	T2CNB1 M2[0Ch]	T2CNB2 M3[08h]	Contains capture, compare, overflow flags
Value Register	T2V0 M2[09h]	T2V1 M2[0Dh]	T2V2 M3[09h]	Holds current timer value
Value MSB Register	T2H0 M2[01h]	T2H1 M2[05h]	T2H2 M3[01h]	Provides access to high byte of T2Vx
Reload Register	T2R0 M2[0Ah]	T2R1 M2[0Eh]	T2R2 M3[0Ah]	Holds timer reload value
Reload MSB Register	T2RH0 M2[02h]	T2RH1 M2[06h]	T2RH2 M3[02h]	Provides access to high byte of T2Rx
Capture/Compare Register	T2C0 M2[0Bh]	T2C1 M2[0Fh]	T2C2 M3[0Bh]	Holds capture/compare value
Capture/Compare MSB Register	T2CH0 M2[03h]	T2CH1 M2[07h]	T2CH2 M3[03h]	Access to high byte of T2Cx

MAXQ7665/MAXQ7666 User's Guide

7.2.1 Type 2 Status/Control Registers

The MAXQ7665/MAXQ7666 timer/counter module registers T2CFGx (configuration), T2CNAx (control A), and T2CNBx (control B), where x = 0, 1, and 2, are described here.

7.2.1.1 Type 2 Timer/Counter 2 Configuration Register (T2CFGx)

Register Description: **Type 2 Timer/Counter 2 Configuration Register**
Register Name: **T2CFGx (x = 0, 1, 2)**
Register Address:

T2CFG0: **Module 02h, Index 10h**
T2CFG1: **Module 02h, Index 11h**
T2CFG2: **Module 03h, Index 10h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	—	T2DIV2	T2DIV1	T2DIV0	T2MD	CCF1	CCF0	C/T2
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 7: Reserved. Read 0, write ignored.

Bits 6 to 4: Type 2 Timer Clock Divide Bits 2 to 0 (T2DIV2 to T2DIV0). These three bits select the divide ratio for the timer clock input clock (as a function of the system clock).

T2DIV2:T2DIV0			DIVIDE RATIO
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Bit 3: Type 2 Timer Mode Select (T2MD). This bit enables the dual 8-bit mode of operation. The default reset state is 0, which selects the 16-bit mode of operation. When the dual 8-bit mode is established, the primary timer/counter (T2Hx) carries all the counter/capture functionality, while the secondary 8-bit timer (T2Lx) must operate in timer compare mode, sourcing the defined internal clock.

0 = 16-bit mode (default)
1 = dual 8-bit mode

Bits 2 and 1: Capture/Compare Function Select Bits (CCF1 and CCF0). These bits, in conjunction with the C/T2 bit, select the basic operating mode of the Type 2 timer. In the dual 8-bit mode of operation (T2MD = 1), the T2Lx timer only operates in compare mode.

CCF1	CCF0	EDGE(s)	C/T2 = 0 (TIMER MODE)	C/T2 = 1 (COUNTER MODE)
0	0	None	Compare Mode	Disabled
0	1	Rising	Capture/Reload	Counter
1	0	Falling	Capture/Reload	Counter
1	1	Rising and Falling	Capture/Reload	Counter

Bit 0: Counter/Timer Select (C/T2). This bit enables/disables the edge counter mode of operation for the 16-bit counter (T2Vx) or the 8-bit counter (T2Hx) when the dual 8-bit mode of operation is enabled (T2MD = 1). The edge for counting (rising/falling/both) is defined by the CCF1:CCF0 bits.

    0 = timer mode
    1 = counter mode

Note: Timer 2 in the MAXQ7665/MAXQ7666 does not support an input/output pin and serves only as an internal timer. Thus, counter mode (C/T2 = 1) functionality does not apply for timer 2.

7.2.1.2 Type 2 Timer/Counter 2 Control Register A (T2CNAx)

Register Description: **Type 2 Timer/Counter 2 Control Register A**
Register Name   **T2CNAx (x = 0, 1, 2**)
Register Address:

    T2CNA0: **Module 02h, Index 00h**
    T2CNA1: **Module 02h, Index 04h**
    T2CNA2: **Module 03h, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ET2	T2OE0	T2POL0	TR2L	TR2	CPRL2	SS2	G2EN
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved. Read 0, write ignored.

Bit 7: Enable Type 2 Timer Interrupts (ET2). This bit serves as the local enable for the Type 2 timer interrupt sources that fall under the TF2 and TCC2 interrupt flags.

Bit 6: Type 2 Timer Output Enable 0 (T2OE0). This register bit enables the timer output function for the external Tx pin. The following table shows the timer output possibilities for the external pin. **Note:** When the timer output function on the Tx pin is enabled, the polarity bit (T2POL0) selects the starting logic level for the pin output.

T2OE0	T2MD	TIMER INPUT/OUTPUT PIN (Tx)
0	X	Port latch data
1	0	16-bit PWM output
1	1	8-bit PWM output (T2Hx)

Bit 5: Type 2 Timer Polarity Select 0 (T2POL0). When the timer output function has been enabled (T2OE0 = 1), the polarity select bit defines the starting logic level for the Tx output waveform. When T2POL0 = 0, the starting state for the output is logic-low. When T2POL0 = 1, the starting state for the output is logic-high. The T2POL0 bit can be modified at any time, but takes effect on the external pin when T2OE0 is changed from 0 to 1. When the Type 2 timer pin is being used as an input (T2OE0 = 0), the polarity select bit defines which logic level can be used to gate the timer input clock (when CCF1:CCF0 ≠ 11b). When CCF1:CCF0 = 11b, T2POL0 defines which edge can start/stop a single-shot capture and which edge reload can be skipped (if CPRL2 = 1 and G2EN = 1).

Note: The MAXQ7665/MAXQ7666 timer 2 does not support an input/output pin and serves only as an internal timer. Thus, polarity select functionality does not apply for timer 2.

Bit 4: Type 2 Timer Low Run Enable (TR2L). This bit start/stops the low 8-bit timer (T2Lx) when dual 8-bit mode (T2MD = 1) is in effect. This bit has no effect when T2MD = 0.

    0 = timer low stopped
    1 = timer low run

Bit 3: Type 2 Timer Run Enable (TR2). This bit starts/stops the Type 2 timer. In the dual 8-bit mode of operation, this bit applies only to the T2Hx timer/counter. Otherwise, the bit applies to the full 16-bit T2Vx timer/counter. When the timer is stopped (TR2 = 0), the timer registers hold their count. The single-shot bit (SS2) can override and/or delay the effect of the TR2 bit.

    0 = timer stopped
    1 = timer run

Bit 2: Capture and Reload Enable (CPRL2). This bit enables a reload (in addition to a capture) on the edge specified by CCF1:CCF0 when operating in capture/reload mode (C/T2 = 0). If both edges are defined for capture/reload (CCF1:CCF0 = 11b), enabling the gating control (G2EN = 1) allows the T2POL0 bit to be used to prevent a reload on one of the edges; if T2POL0 is 0, no reload on the falling edge; or, if T2POL0 is 1, no reload on the rising edge.

    0 = capture on edge(s) specified by CCF1:CCF0 bits
    1 = capture and reload on edge(s) specified by CCF1:CCF0 bits

Bit 1: Single-Shot (SS2). This bit is used to automatically override or delay the effect of the TR2 bit setting. The single-shot bit is only useful in the timer mode of operation (C/T2 = 0) and should not be set to 1 when the counter mode of operation is enabled (C/T2 = 1).

Compare Mode: If SS2 is written to 1 while in compare mode, one cycle of the defined waveform (reload to overflow) is output to the Tx pin as prescribed by T2POL0 and T2OE0 controls. The only time that this does not immediately occur is when a gating condition is also defined. If a gating condition is defined, the single-shot cycle cannot occur until the gating condition is removed. If the specified nongated level is already in effect, the single-shot period starts. The gated single-shot output is not supported in dual 8-bit mode.

Capture Mode: If SS2 is written to 1 while in capture mode, the timer is halted and the single-shot capture cycle does not begin until 1) the edge specified by CCF1:CCF0 is detected, or 2) the defined gating condition is removed. Once running, the timer continues running (as allowed by the gate condition) until the defined capture single-shot edge is detected. In this way, the SS2 bit can be used to delay the running of a timer until an edge is detected (setting both SS2 and TR2 = 1) or override the TR2 = 0 bit setting for one capture cycle (setting only SS2 = 1). When both edges are defined for capture (CCF1:CCF0 = 11b), the T2POL0 bit serves to define the single-shot start/end edge: falling edge if T2POL0 = 1, rising edge if T2POL0 = 0. No interrupt flag is set when the starting edge for the single-shot capture cycle is detected. The single-shot capture cycle always ends when the next single-shot edge is detected. The start/end edge is defined by T2POL0. This bit is intended to automate pulse-width measurement (low or high) and duty cycle/period measurement.

Note: The MAXQ7665/MAXQ7666 timer 2 does not support an input/output pin and serves only as an internal timer. Thus, some single-shot functionality does not apply for timer 2.

Bit 0: Gating Enable (G2EN). This bit enables the external Tx pin to gate the input clock to the 16-bit (T2MD = 0) or highest 8-bit (T2MD = 1) timer. Gating uses Tx as an input, so it can only be used when T2OE0 = 0 and C/T2 = 0. Gating is not possible on the low 8-bit timer (T2Lx) when the Type 2 timer is operated in dual 8-bit mode. Gating is not supported for counter mode operation (C/T2 = 1). The G2EN bit serves a different purpose when capture and reload have been defined for both edges (CCF1:CCF0 = 11b and CPRL2 = 1). For this special case, setting G2EN = 1 allows the T2POL0 bit to specify which edge does not cause a reload. If T2POL0 is 0, there is no reload on the falling edge; if T2POL0 is 1, there is no reload on the rising edge.

> 0 = gating disabled
> 1 = gating enabled

Note: The MAXQ7665/MAXQ7666 timer 2 does not support an input/output pin and serves only as an internal timer. Thus, gating functionality does not apply for timer 2.

7.2.1.3 Type 2 Timer/Counter 2 Control Register B (T2CNBx)

Register Description: **Type 2 Timer/Counter 2 Control Register B**
Register Name: **T2CNBx (x = 0, 1, 2)**
Register Address:

    T2CNB0: **Module 02h, Index 08h**
    T2CNB1: **Module 02h, Index 0Ch**
    T2CNB2: **Module 03h, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ET2L	—	—	—	TF2	TF2L	TCC2	TC2L
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	r	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8, 6 to 4: Reserved. Read 0, write ignored.

Bit 7: Enable Type 2 Timer Low Interrupts (ET2L). This bit serves as the local enable for T2Lx interrupt sources that fall under the TF2L and TC2L interrupt flags.

Bit 3: Type 2 Timer Overflow Flag (TF2). This flag becomes set anytime there is an overflow of the full 16-bit T2Vx timer/counter (when T2MD = 0) or an overflow of the 8-bit T2Hx timer/counter when the dual 8-bit mode of operation is selected (T2MD = 1).

Bit 2: Type 2 Timer Low Overflow Flag (TF2L). This flag is meaningful only when in the dual 8-bit mode of operation (T2MD = 1). It is set whenever there is an overflow of the T2Lx 8-bit timer.

Bit 1: Type 2 Timer Capture/Compare Flag (TCC2). This flag is set on any compare match between the Type 2 timer value and compare register (T2Vx = T2Cx or T2Hx = T2CHx, respectively, for 16-bit and 8-bit compare modes) or when a capture event is initiated by an external edge.

Bit 0: Type 2 Timer Low Compare Flag (TC2L). This flag is meaningful only for the dual 8-bit mode of operation (T2MD = 1). It is set only when a compare match occurs between T2CLx and T2Lx. The Type 2 timer low does not have an associated capture function.

7.2.2 Type 2 Timer Value Registers

The MAXQ7665/MAXQ7666 timer/counter registers T2Vx (timer value) and T2Hx (timer value high), where x = 0, 1, and 2, are described here.

7.2.2.1 Type 2 Timer/Counter Value Register (T2Vx)

Register Description: **Type 2 Timer/Counter Value Register**
Register Name: **T2Vx (x = 0, 1, 2)**
Register Address:

      T2V0: **Module 02h, Index 09h**
      T2V1: **Module 02h, Index 0Dh**
      T2V2: **Module 03h, Index 09h**

Bit #	15	14	13	12	11	10	9	8
Name	T2Vx.15	T2Vx.14	T2Vx.13	T2Vx.12	T2Vx.11	T2Vx.10	T2Vx.9	T2Vx.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	T2Vx.7	T2Vx.6	T2Vx.5	T2Vx.4	T2Vx.3	T2Vx.2	T2Vx.1	T2Vx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Type 2 Timer/Counter Value Register (T2Vx.15 to T2Vx.0). The T2Vx register is a 16-bit register that holds the current timer value. When operating in 16-bit mode (T2MD = 0), the full 16 bits are read/write accessible. If the dual 8-bit mode of operation (T2MD = 1) is selected, the upper byte of T2Vx is inaccessible. T2Vx reads while in the dual 8-bit mode return 00h as the high byte and writes to the upper byte of T2Vx are blocked. A separate T2Hx register is provided to facilitate high-byte access for dual 8-bit mode.

Note: For convenience, the lower byte of T2Vx (T2Vx.7–T2Vx.0) is referred to as T2Lx. Unlike T2Hx, there is no separate T2Lx register and the low byte is always accessed through T2Vx.

7.2.2.2 Type 2 Timer/Counter Value High Register (T2Hx)

Register Description: **Type 2 Timer/Counter Value High Register**
Register Name: **T2Hx (x = 0, 1, 2)**
Register Address:

T2H0: **Module 02h, Index 01h**
T2H1: **Module 02h, Index 05h**
T2H2: **Module 03h, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	T2Hx.7	T2Hx.6	T2Hx.5	T2Hx.4	T2Hx.3	T2Hx.2	T2Hx.1	T2Hx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved.

Bits 7 to 0: Type 2 Timer/Counter Value High Register (T2Hx.7 to T2Hx.0). This register is used to load and read the most significant 8-bit timer value (T2Hx) for the dual 8-bit mode.

7.2.3 Type 2 Reload Registers

The MAXQ7665/MAXQ7666 timer/counter module registers T2Rx (timer reload) and T2RHx (timer reload high), where x = 0, 1, and 2, are described here.

7.2.3.1 Type 2 Timer/Counter Reload Register (T2Rx)

Register Description: **Type 2 Timer/Counter Reload Register**
Register Name: **T2Rx (x = 0, 1, 2)**
Register Address:

    T2R0: **Module 02h, Index 0Ah**
    T2R1: **Module 02h, Index 0Eh**
    T2R2: **Module 03h, Index 0Ah**

Bit #	15	14	13	12	11	10	9	8
Name	T2Rx.15	T2Rx.14	T2Rx.13	T2Rx.12	T2Rx.11	T2Rx.10	T2Rx.9	T2Rx.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	T2Rx.7	T2Rx.6	T2Rx.5	T2Rx.4	T2Rx.3	T2Rx.2	T2Rx.1	T2Rx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Type 2 Timer/Counter Reload Register (T2Rx.15 to T2Rx.0). This 16-bit register holds the reload value for the timer. When operating in 16-bit mode (T2MD = 0), the full 16 bits are read/write accessible. If the dual 8-bit mode of operation is selected, the upper byte of T2Rx is inaccessible. T2Rx reads while in the dual 8-bit mode return 00h as the high byte and writes to the upper byte of T2Rx are blocked. A separate T2RHx register is provided to facilitate high-byte access for the dual 8-bit mode.

Note: For convenience, the lower byte of T2Rx (T2Rx.7–T2Rx.0) is referred to as T2RLx. Unlike T2RHx, there is no separate T2RLx register and the low byte is always accessed through T2Rx.

7.2.3.2 Type 2 Timer/Counter Reload High Register (T2RHx)

Register Description: **Type 2 Timer/Counter Reload High Register**
Register Name: **T2RHx (x = 0, 1, 2)**
Register Address:

     T2RH0: **Module 02h, Index 02h**
     T2RH1: **Module 02h, Index 06h**
     T2RH2: **Module 03h, Index 02h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	T2RHx.7	T2RHx.6	T2RHx.5	T2RHx.4	T2RHx.3	T2RHx.2	T2RHx.1	T2RHx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved.

Bits 7 to 0: Type 2 Timer/Counter Reload High Register (T2RHx.7 to T2RHx.0). This register is used to load and read the most significant 8-bit reload value (T2RHx) in the timer for the dual 8-bit mode.

7.2.4 Type 2 Capture/Compare Registers

The MAXQ7665/MAXQ7666 timer/counter module registers T2Cx (timer capture/compare) and T2CHx (timer capture/compare high), where x = 0, 1, and 2, are described here.

7.2.4.1 Type 2 Timer/Counter Capture/Compare Register (T2Cx)

Register Description: **Type 2 Timer/Counter Capture/Compare Register**
Register Name: **T2Cx (x = 0, 1, 2)**
Register Address:

T2C0: **Module 02h, Index 0Bh**
T2C1: **Module 02h, Index 0Fh**
T2C2: **Module 03h, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	T2Cx.15	T2Cx.14	T2Cx.13	T2Cx.12	T2Cx.11	T2Cx.10	T2Cx.9	T2Cx.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	T2Cx.7	T2Cx.6	T2Cx.5	T2Cx.4	T2Cx.3	T2Cx.2	T2Cx.1	T2Cx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Type 2 Timer/Counter Capture/Compare Register (T2Cx.15 to T2Cx.0). This 16-bit register holds the compare value when operating in compare mode and gets the capture value when operating in capture mode. When operating in 16-bit mode (T2MD = 0), the full 16 bits are read/write accessible. If the dual 8-bit mode of operation is selected, the upper byte of T2Cx is inaccessible. T2Cx reads while in the dual 8-bit mode return 00h as the high byte and writes to the upper byte of T2Cx are blocked. A separate T2CHx register is provided to facilitate high-byte access.

Note: For convenience, the lower byte of T2Cx (T2Cx.7–T2Cx.0) is referred to as T2CLx. Unlike T2CHx, there is no separate T2CLx register and the low byte is always accessed through T2Cx.

7.2.4.2 Type 2 Timer/Counter Capture/Compare High Register (T2CHx)

Register Description: **Type 2 Timer/Counter Capture/Compare High Register**
Register Name: **T2CHx (x = 0, 1, 2)**
Register Address:

T2CH0: **Module 02h, Index 03h**
T2CH1: **Module 02h, Index 07h**
T2CH2: **Module 03h, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	T2CHx.7	T2CHx.6	T2CHx.5	T2CHx.4	T2CHx.3	T2CHx.2	T2CHx.1	T2CHx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved.

Bits 7 to 0: Type 2 Timer/Counter Capture/Compare High (T2CHx.7 to T2CHx.0). This register is used to load and read the most significant 8-bit capture/compare value (T2CHx) of the timer for the dual 8-bit mode.

7.3 Type 2 Timer/Counter Operation Modes

The MAXQ7665/MAXQ7666 Type 2 timer/counter supports six operation modes. Table 7-3 summarizes the modes supported by the Type 2 timer and the peripheral register bits associated with those modes.

The Type 2 timer operating mode selection is illustrated in Figure 7-3. Figure 7-4 shows the PWM timer output possibilities.

Table 7-3. Type 2 Timer/Counter Functions and Control

MODE	T2MD	C/T2	CCF1:CCF0	CONTROL BITS		
16-Bit Auto-Reload/Compare Timer	0	0	00	T2OE0: Output enable (PWM out)		
				T2POL0: Input/output polarity select		
				SS2: Single-shot pulse control		
				G2EN: Gate timer clock		
16-Bit Capture (CCF1:CCF0 bits define capture edge)	0	0	01, 10, or 11	**T2OE0 = 0**	T2POL0: Gate level/reload edge select	
					SS2: Single-shot capture	
					G2EN: Gate timer clock (or gate reload)	
					CPRL2: Reload enable	
16-Bit Counter (CCF1:CCF0 bits define count edge)	0	1	01, 10, or 11	**T2OE0 = 0**		
Dual 8-Bit Auto-Reload Timers	1	0	00	T2POL0: Output polarity select T2OE0: Output enable (PWM out)		
				T2Hx Only:	SS2: Single-shot pulse control	
8-Bit Capture and 8-Bit Timer/PWM (CCF1:CCF0 bits define capture edge)	1	0	01, 10, or 11	T2Hx Only:	**T2OE0 = 0**	
					T2POL0: Gate level/reload edge select	
					SS2: Single-shot capture	
					G2EN: Gate timer (or gate reload)	
					CPRL2: Reload enable	
8-Bit Counter and 8-Bit Timer/PWM (CCF1:CCF0 bits define count edge)	1	1	01, 10, or 11	T2Hx Only:	**T2OE0 = 0**	

Note: Timer 2 in the MAXQ7665/MAXQ7666 does not support an input/output pin and serves only as an internal timer.
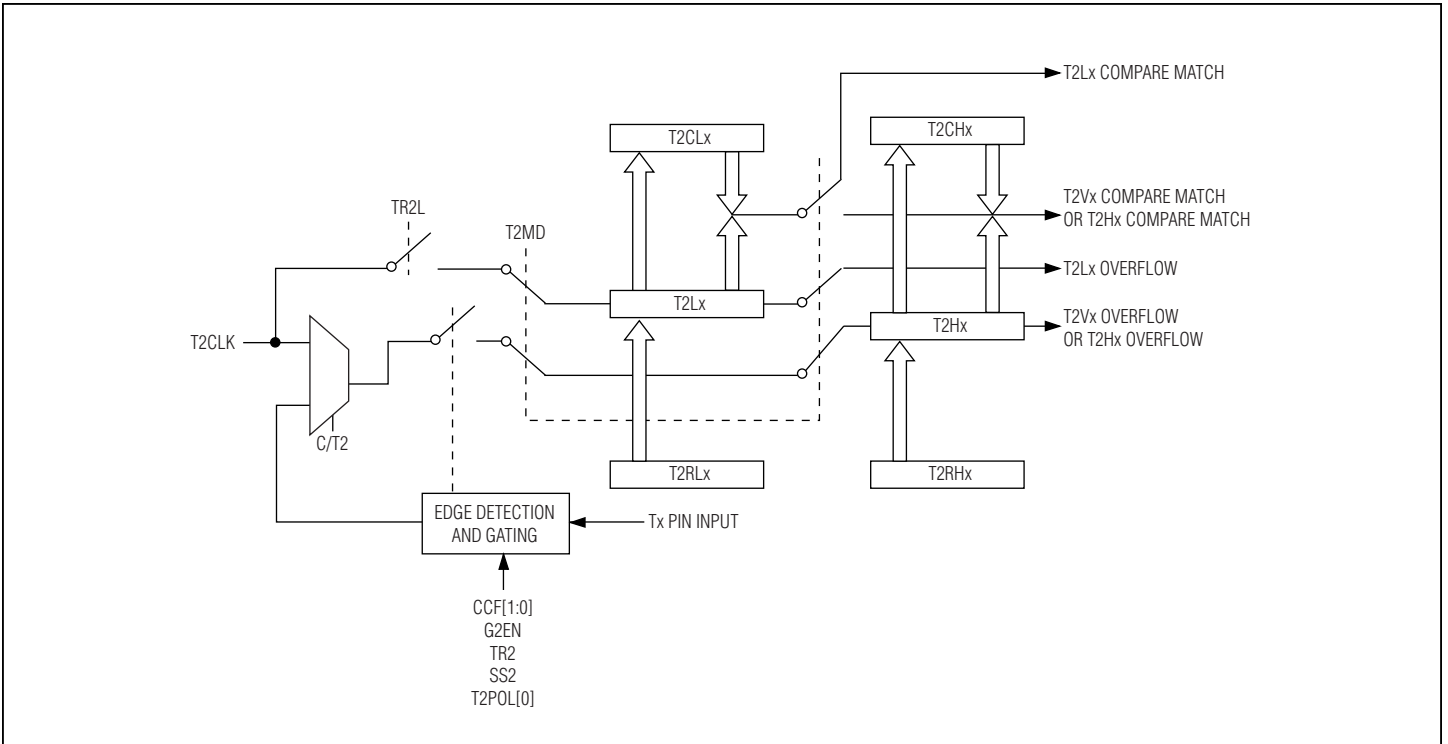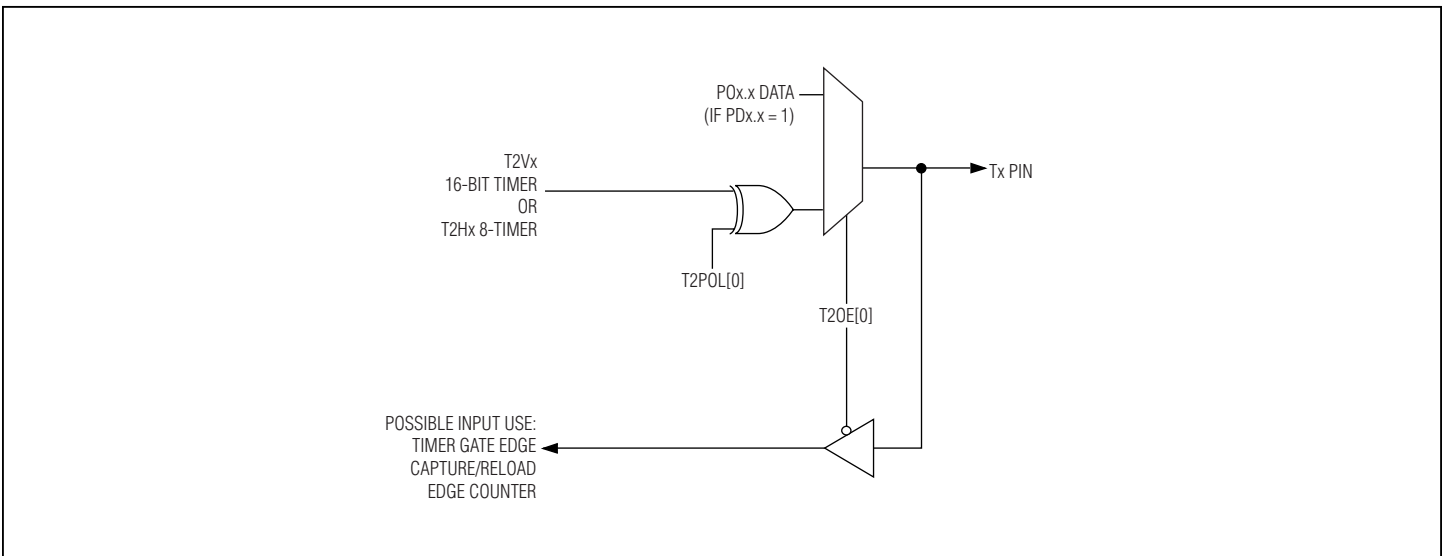
Figure 7-3. Type 2 Timer Mode Selection

Figure 7-4. Output Enable and Polarity Control

7.3.1 16-Bit Timer: Auto-Reload/Compare

The 16-bit auto-reload/compare mode for the Type 2 timer is in effect when the timer-mode select bit (T2MD) is cleared and the capture/compare function definition bits are both cleared (CCF1:CCF0 = 00b). The timer value is contained in the T2Vx register. The timer run control bit (TR2) starts and stops the 16-bit timer. The input clock for the 16-bit Type 2 timer is defined as the system clock divided by the ratio specified by the T2DIV2:T2DIV0 prescale bits. The timer begins counting from the value contained in the T2Vx register until an overflow occurs. When an overflow occurs, the reload value is reloaded instead of the x0000h state. The timer overflow flag (TF2) is set every time that an overflow condition (T2Vx = 0xFFFFh) is detected. If the Type 2 timer interrupts have been enabled (ET2 = 1), the TF2 flag can generate an interrupt request. When operating in compare mode, the capture/compare register, T2Cx, is compared versus the timer value registers. Whenever a compare match occurs, the capture/compare status flag (TCC2) is set. If the Type 2 timer interrupts have been enabled (ET2 = 1), this event can generate an interrupt request. If the capture/compare register is set to a value outside of the timer counting range, a compare match is not signaled and the TCC2 flag is not set. Internally, a timer output clock is generated that toggles on the cycle following any compare match or overflow, unless the compare match value has been set equal to the overflow condition, in which case, only one toggle occurs. This clock, if enabled by the T2OE0 bit, can be output on the timer pin Tx. **Note:** For an interrupt to occur, the global enable bits IM2 (for timer 0 and timer 1) and IM3 (for timer 2) in the IMR peripheral register and IGE in the IC peripheral register must also be enabled.

- **Output Enable (PWM Out).** The output enable bit (T2OE0) enables the timer output clock to be presented on the timer input/output pin Tx (0 for timer 0, 1 for timer 1). **Note:** T2 is not supported on the 48- and 56-pin packages. Thus, timer 2 can serve only as an internal timer.

- **Polarity Control.** The polarity control bit (T2POL0) can be used to modify (invert) the enabled clock output to the pin. The enabled clock output toggles on each compare match or overflow. The T2POL0 bit is logically XORed with the timer output signal, therefore setting T2POL0 will result in a high starting state. The T2POL0 bit can be changed any time, however, the assigned T2POL0 state will take effect on the external pin only when the corresponding T2OE0 bit is changed from 0 to 1. When generating PWM output, note that changing the compare match register can result in a perceived duty cycle inversion if a compare match is missed or multiple compare matches occur during the reload to overflow counting.

- **Gated.** To use the Tx pin as a timer input clock gate, the T2OE0 bit must be cleared to 0 and the G2EN bit must be set to 1. When T2OE0 = 1, the G2EN bit setting has no effect. When T2OE0 is cleared to 0, the respective polarity control bit is used to modify the polarity of the input signal to the timer. In the gated mode, the timer input clock is gated anytime the external signal matches the state of the T2POL0 bit. This means that the default clock gating condition for the Tx pin is logic-low (since T2POL0 = 0 default). Setting T2POL0 = 1 results in the timer input clock being gated when the Tx pin is high.

- **Single-Shot.** When operating in 16-bit compare mode, the single-shot is used to automate the generation of single pulses under software control or in response to an external signal (single-shot gated). To generate single-shot output pulses solely under software control, the G2EN bit should be cleared to 0, the output enables and polarity controls should be configured as desired, and the single-shot bit should be set to 1. Writing the single-shot bit effectively overrides the TR2 = 0 condition until timer overflow/reload occurs. The single-shot bit is automatically cleared once the overflow/reload occurs.

  Writing SS2 and TR2 = 1 at the same time still causes the SS2 bit to stay in effect until an overflow/reload occurs, however, since TR2 was also written to a 1, the specified PWM output continues even after SS2 becomes clear.

- **Capture/Reload Control.** For the 16-bit compare operating mode, the CPRL2 bit is not used.

7.3.2 16-Bit Timer: Capture Mode

The 16-bit capture mode requires that some event trigger the capture. Normally, this event will be an external edge. The CCF1:CCF0 bits define which edge(s) causes a capture to occur. If CCF1:CCF0 = 01b, a rising edge causes a capture. If CCF1:CCF0 = 10b, a falling edge causes a capture. If CCF1:CCF0 = 11b, rising and falling edges both cause a capture to occur. The CPRL2 bit enables both capture and reload to occur on the specified edge(s). Whenever a capture occurs, the capture/compare status flag (TCC2) is set. If the Type 2 timer interrupts have been enabled (ET2 = 1), this event can generate an interrupt request.

Note: For an interrupt to occur, the global enable bits IM2 (for timer 0 and timer 1) and IM3 (for timer 2) in the IMR peripheral register and IGE in the IC peripheral register must also be enabled.

- **Output Enable.** In 16-bit capture mode, the output enables are meaningless. No output waveform is allowed since the capture/compare registers are being used for the purpose of capturing the timer value.

- **Polarity Control.** The polarity control bit (T2POL0) has no specific meaning as related to the output function since there is no output function. The T2POL0 bit is used to establish the gating condition for the single-edge capture mode when gating is enabled (G2EN = 1). If capture and reload are defined (CPRL2 = 1 and CCF1:CCF0 = 11b) for both edges, the T2POL0 bit can be used to specify which edge does not have an associated edge reload when gating has also been enabled (G2EN bit = 1). When the SS2 bit is used to delay the timer run (for both edge capture), the T2POL0 bit also defines which edge starts/ends the single-shot process.

- **Edge Detection.** Edge detection was described above (CCF1:CCF0 controlled).

- **Gated.** If gating is specified, it uses the T2POL0 bit to define when the input clock to the timer is gated (just as described for the compare mode). This mode can easily be used to measure or incrementally capture high or low pulse durations. If a predefined high/low duration is required to generate an interrupt, the gated compare mode can also be used. Note that if capture is defined for both rising and falling edges, gating would serve no useful purpose because it would result in redundant capture data/interrupts. For this reason, when G2EN = 1 and CCF1:CCF0 = 11b, the T2POL0 bit is used to specify which edge is a capture-only edge when CPRL2 = 1 (gating of the reload event).

- **Single-Shot.** The single-shot bit overrides the TR2 = 0 bit setting for a single edge-to-edge capture cycle (as defined by the CCF1:CCF0 bits). The single-shot takes effect (starting the timer) only when 1) the edge defined by CCF1:CCF0 is detected or 2) the defined gating condition is removed. While a capture and/or reload may occur on this starting edge, the interrupt flag will not be set since a single-shot event has been requested. When rising or falling edge capture is defined, the single-shot mode is useful for measuring single periods. If gating is also specified for the single-shot, the high/low pulse widths are easily measured. If rising **and** falling edges are defined, the T2POL0 bit designates which edge starts/ends the single-shot cycle, but the starting edge will not cause the interrupt flag to set. If G2EN = 1 for the two-edge capture, the alternate edge (opposite of defined start/end edge can only be used for capture, not capture and reload). For T2POL0 = 1, the falling edge starts and stops the single shot. This is important for combined duty cycle and period measurement.

- **Capture and Reload.** The CPRL2 bit enables both capture and reload on the specified edge(s). The only exception to this rule is when the G2EN bit is set to a logic 1. When G2EN is set to 1, a reload does not occur on the edge specified by T2POL0. When T2POL0 = 0, the falling edge does not cause a reload; if T2POL0 = 1, the rising edge does not cause a reload.

7.3.3 16-Bit Counter

The 16-bit counter mode is enabled by setting the C/T2 bit to logic 1. When C/T2 = 1, rising, falling, or both rising and falling edges are counted as determined by the CCF1:CCF0 bits. If CCF1:CCF0 = 00b, neither edge is defined as a counted edge, and the T2Vx counter will hold its count. When an overflow occurs, the reload value (T2Rx) is reloaded instead of the x0000h state. The timer/counter 2 overflow flag (TF2) is set every time that an overflow occurs. If timer/counter 2 interrupts have been enabled (ET2 = 1), the TF2 flag can generate an interrupt request. In counter mode, the capture/compare register (T2Cx) is compared versus the timer/counter 2 value register. Whenever a compare match occurs, the capture/compare status flag (TCC2) is set. If timer/counter 2 interrupts have been enabled (ET2 = 1), this event can generate an interrupt request. If the capture/compare register is set to a value outside of the Type 2 timer counting range, a compare match is not signaled and the TCC2 flag is not set.

Note: For an interrupt to occur, the global enable bits IM2 (for timer 0 and timer 1) and IM3 (for timer 2) in the IMR peripheral register and IGE in the IC peripheral register must also be enabled.

- **Output Enable.** For the timer to serve as a counter, the Tx pin must be used as an input. Thus, when C/T2 = 1, the T2OE0 bit is ignored. **Note:** T2 is not supported on the 48- and 56-pin packages. Thus, timer 2 can only serve as an internal timer.

- **Polarity Control.** When C/T2 = 1, the T2OE0 and T2POL0 bits are ignored.

- **Gating and Single-Shot.** Neither gating nor single-shot modes are supported when operating in 16-bit counter mode. The G2EN and SS2 bits should not be set to 1 when operating in the counter mode (C/T2 = 1).

7.3.4 Dual 8-Bit Timers

The dual 8-bit timer mode of operation is initiated by setting the T2MD bit to logic 1. When T2MD = 1, each 16-bit register associated with the Type 2 timer is split into separate upper and lower 8-bit registers to support dual 8-bit timers. Thus, the primary 8-bit timer is composed of T2Hx (value), T2RHx (reload), T2CHx (capture/compare), and the secondary 8-bit timer is composed of T2Lx (value), T2RLx (reload), and T2CLx (capture/compare). There is but a single internal Type 2 timer input clock that can be sourced by either of these two 8-bit timers. The secondary 8-bit timer/counter has its own run control bit (TR2L) and interrupt flags (TF2L, TC2L).

- **Output Enable (PWM out).** The output enable bit (T2OE0) enables the primary T2Hx 8-bit timer output to be presented on the Tx pin. T2Lx can only serve as an internal timer.

- **Polarity Control.** The polarity control bit (T2POL0) can be used to modify (invert) the enabled clock output to the pin. The starting state of the enabled clock output is the logic state of T2POL0 and toggles on each compare match or overflow. The T2POL0 bit is logically XORed with the timer output signal, therefore setting the T2POL0 bit results in a high starting state. The T2POL0 bit can be changed any time, however the assigned T2POL0 state will take effect on the external pin only when the corresponding T2OE0 bit is changed from 0 to 1. When generating PWM output, note that changing the compare match register can result in a perceived duty cycle inversion if a compare match is missed or multiple compare matches occur during the reload to overflow counting.

- **Gated.** To use the Tx pin as a timer input clock gate, the T2OE0 bit must be cleared to 0 and the G2EN bit must be set to 1. When T2OE0 = 1, the G2EN bit setting has no effect. When T2OE0 is cleared to 0, the respective polarity control bit is used to modify the polarity of the input signal to the timer. In the gated mode, the input clock to T2Hx is gated anytime that the external signal matches the state of the T2POL0 bit. This means that the default clock gating condition is associated with the Tx pin being low (T2POL0 = 0). Note that the secondary 8-bit timer, T2Lx, cannot be gated.

- **Single-Shot.** The single-shot bit and mode apply only to the primary 8-bit timer (T2Hx). The single-shot mode is used to automate the generation of single pulses under software control. To generate single-shot output pulses under software control, the G2EN bit should be cleared to 0, the output enables and polarity controls should be configured as desired, and the single-shot bit should be set to 1. Writing the single-shot bit effectively overrides the TR2 = 0 condition until the timer overflow/reload occurs. Writing SS2 and TR2 = 1 at the same time still causes the SS2 bit to stay in effect until an overflow/reload occurs, however, the specified PWM output continues since TR2 was also written to 1.

7.3.5 8-Bit Timer/8-Bit Capture Mode

When the CCF1:CCF0 bits are configured to a state other than 00b, the edge capture mode is enabled for the primary timer (T2Hx). The secondary timer (T2Lx) always remains in the timer/compare mode and does not support any capture functionality. The capture controls for the 8-bit mode are identical to those specified for the 16-bit mode, however, they apply only to the upper timer, T2Hx.

7.3.6 8-Bit Timer/8-Bit Counter

Just as in the 16-bit mode, setting the C/T2 bit to logic 1 enables the external Tx pin to function as a counter input. The edges that are counted are determined by the CCF1:CCF0 bits. The counter mode of operation applies only to the primary timer/counter (T2Hx). In a similar fashion to the 16-bit counter mode, when an overflow occurs, an auto-reload of T2RHx occurs and the TF2 flag is set. The TCC2 flag is also set on a compare match between the T2Hx counter and the T2CHx compare register (except for the case where T2CHx is outside of the T2RHx to 0xFFh counting range). The secondary timer (T2Lx) always continues to operate in 8-bit compare mode.

7.3.7 Type 2 Timer Input Clock Selection

The Type 2 timer clock source is illustrated in Figure 7-5. System clock is used as the Type 2 timer clock source and is optionally divided down as defined by the T2DIV2:T2DIV0 bits.
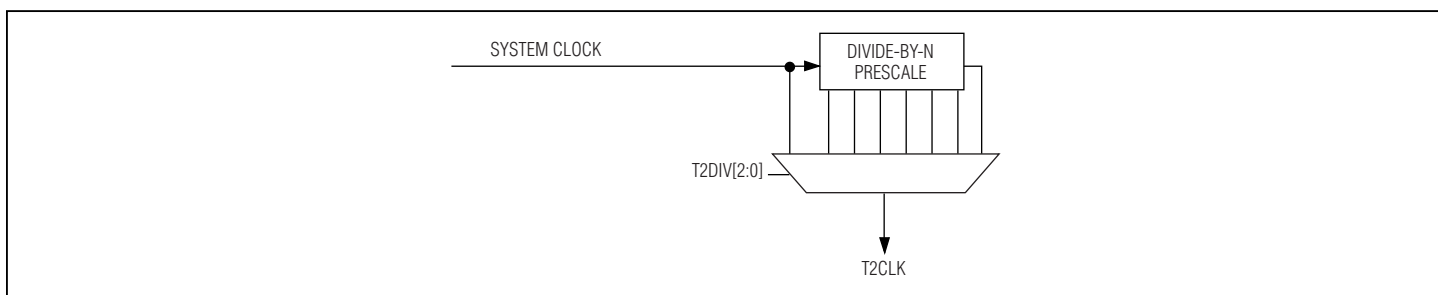
Figure 7-5. Type 2 Timer Clock

7.4 Type 2 Timer/Counter Capture Application Examples

The following examples are used to demonstrate some of the Type 2 timer capture functions. All examples assume that pulse and/or period measurements do not exceed 2^{16} (i.e., 65,536) input clocks and that capture register holds the desired result.

7.4.1 Measure Low-Pulse Duration

To measure the duration of the first full low pulse seen on the T0 input pin, the Type 2 timer is configured for a single-shot capture, gating enabled for logic-high, and capture on the rising edge. The CPRL2 bit can optionally be set to generate a reload on the same rising edge as the capture, if the preconfigured T2R0 value is expected to be needed next.

```
; ----------------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2CFG0, #00000010b          ;
                                 ; T2DIV[2:0]  =000     (/1)
                                 ; T2MD        =0       (16-bit)
                                 ; CCF[1:0]    =01      (rising edge)
                                 ; C/T2        =0       (timer/capture)
MOVE T2CNA0, #10100111b          ; ET2         =1       (enable Timer ints)
                                 ; T2OE0       =0       (input)
                                 ; T2POL0      =1       (gating level = '1')
                                 ; TR2L:TR2    =00      (don't start timer)
                                 ; CPRL2       =1       (reload on capture edge)
                                 ; SS2         =1       (single shot mode)
                                 ; G2EN        =1       (gating enabled)
; ----------------- TCC2 Interrupt : DURATION = T2C0
```
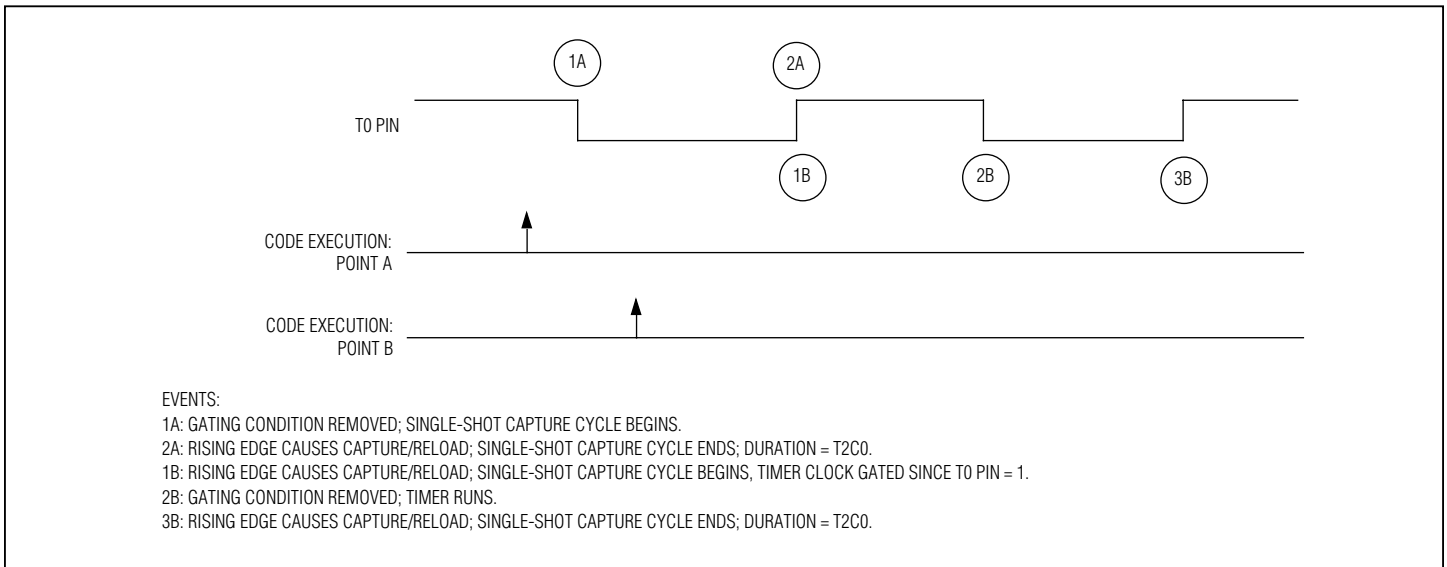

Figure 7-6. Type 2 Timer Application Example—Measure Low Pulse Width

7.4.2 Measure High-Pulse Duration Repeatedly

To measure the duration of high pulses seen on the T0 input pin repeatedly, the Type 2 timer is configured for a single-shot delayed run, gating enabled for logic-low, and capture on the falling edge. The CPRL2 bit can be set to generate a reload on each falling edge.

```
; ----------------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2CFG0, #00000100b          ;
                                 ; T2DIV[2:0]  =000     (/1)
                                 ; T2MD        =0       (16-bit)
                                 ; CCF[1:0]    =10      (falling edge)
                                 ; C/T2        =0       (timer/capture)
MOVE T2CNA0, #10001111b          ; ET2         =1       (enable Timer ints)
                                 ; T2OE0       =0       (input)
                                 ; T2POL0      =0       (gating level = '0')
                                 ; TR2L:TR2    =01      (start timer on single shot
                                 ;                      condition)
                                 ; CPRL2       =1       (reload on capture edge)
                                 ; SS2         =1       (single shot mode)
                                 ; G2EN        =1       (gating enabled)
; ----------------- TCC2 Interrupt : DURATION = T2C0
```


EVENTS:
1A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS; TIMER CLOCK GATED SINCE T0 PIN = 0.
2A: GATING CONDITION REMOVED; TIMER RUNS.
3A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C0. TIMER CONTINUES TO OPERATE SINCE TR2 = 1, BUT TIMER CLOCK GATED SINCE T0 PIN = 0.
4A: GATING CONDITION REMOVED; TIMER RUNS.

1B: GATING CONDITION REMOVED; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2B: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C0. TIMER CONTINUES TO OPERATE SINCE TR2 = 1, BUT TIMER CLOCK GATED SINCE T0 PIN = 0.
3B: GATING CONDITION REMOVED; TIMER RUNS.

Figure 7-7. Type 2 Timer Application Example—Measure High Pulse Width

7.4.3 Measure Period

To measure the period of the signal seen on the T0 input pin, the Type 2 timer is configured for a single-shot capture, no gating, either edge (selected by the CCF1:CCF0 bits). The CPRL2 bit can be set to generate a reload on each capture edge.

```
; ----------------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2CFG0, #00000100b              ;
                                     ; T2DIV[2:0]   =000     (/1)
                                     ; T2MD         =0       (16-bit)
                                     ; CCF[1:0]     =10      (falling edge)
                                     ; C/T2         =0       (timer/capture)
MOVE T2CNA0, #10000110b              ; ET2          =1       (enable Timer ints)
                                     ; T2OE0        =0       (input)
                                     ; T2POL0       =0       (gating level = '0')
                                     ; TR2L:TR2     =00      (don't start timer )
                                     ; CPRL2        =1       (reload on capture edge)
                                     ; SS2          =1       (single shot mode)
                                     ; G2EN         =0       (gating disabled)
; ----------------- TCC2 Interrupt : PERIOD = T2C0
```
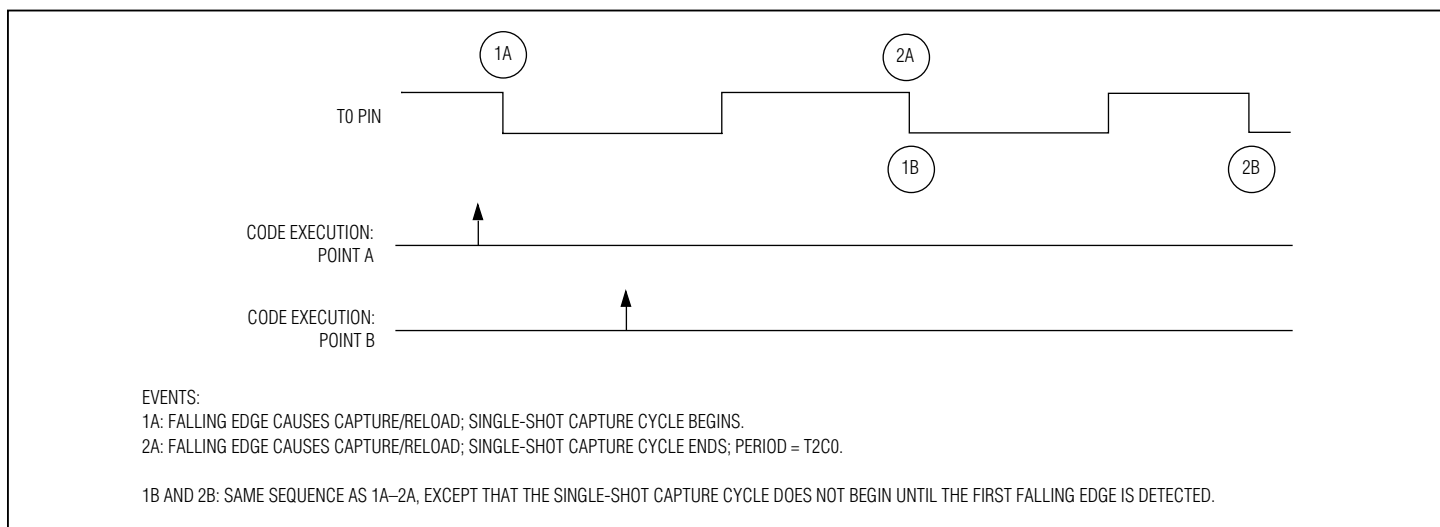

Figure 7-8. Type 2 Timer Application Example—Measure Period

7.4.4 Measure Duty Cycle Repeatedly

To measure the duty cycle of the signal seen on the T0 input pin, the Type 2 timer is configured for a single-shot delayed run with both edges defined for capture. The CPRL2 bits should be configured to 1 to request reloads on each edge. To prevent reloads on one of the edges, gating should be enabled. The T2POL0 bit specifies which edge starts/ends the capture cycle and which edge does not have a reload associated with it.

```
; ----------------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2CFG0, #00000110b           ;
                                  ; T2DIV[2:0]  =000     (/1)
                                  ; T2MD        =0       (16-bit)
                                  ; CCF[1:0]    =11      (both edges)
                                  ; C/T2        =0       (timer/capture)
MOVE T2CNA0, #10101111b           ; ET2         =1       (enable Timer ints)
                                  ; T2OE0       =0       (input)
                                  ; T2POL0      =1       (no reload on rising edge
                                  ;                      single shot start/end on falling edge)
                                  ; TR2L:TR2    =01      (start timer on single shot condition)
                                  ; CPRL2       =1       (reload on capture edge)
                                  ; SS2         =1       (single shot mode)
                                  ; G2EN        =1       (gating enabled)
; ----------------- TCC2 Interrupt : LOW TIME=T2C0
;------------------ TCC2 Interrupt : PERIOD = T2C0
```
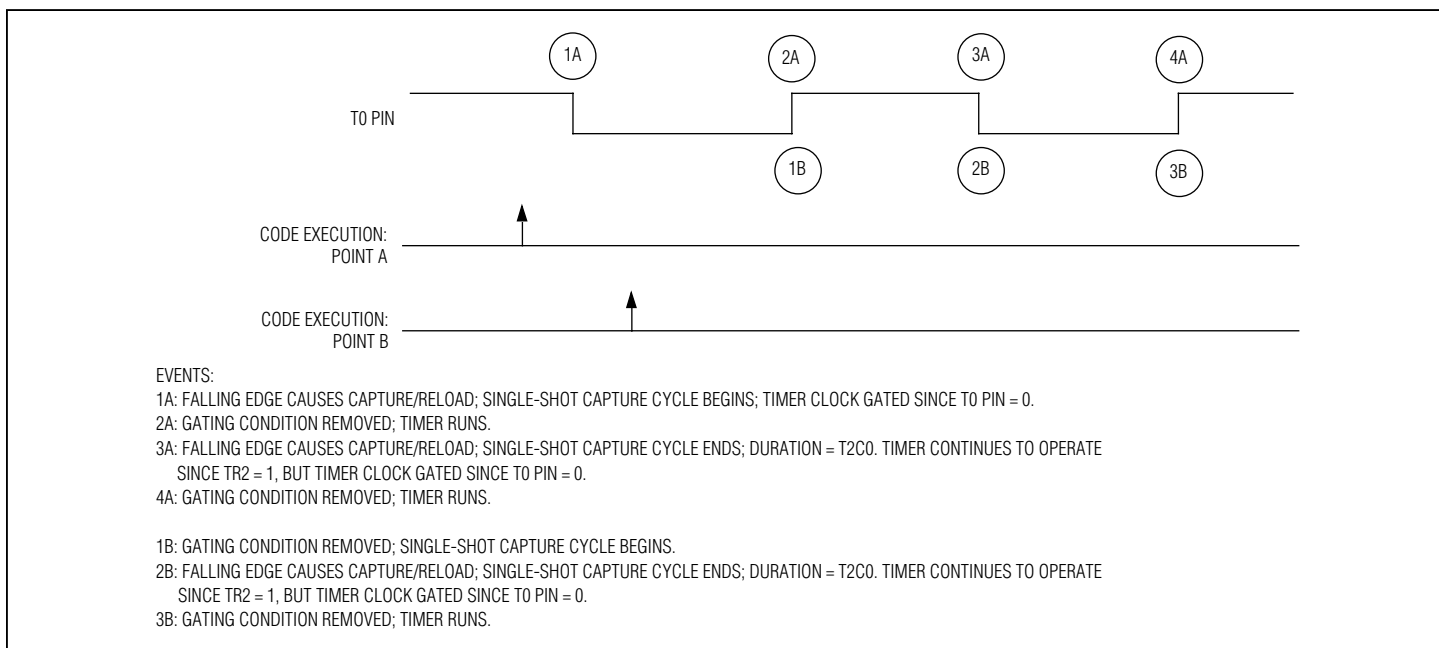

EVENTS:
1A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2A: RISING EDGE CAUSES CAPTURE; LOW TIME = T2C0.
3A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; PERIOD = T2C0. TIMER CONTINUES TO OPERATE
    SINCE TR2 = 1, ALLOWING THE NEXT LOW TIME/PERIOD TO BE MEASURED.
1B–3B: SAME SEQUENCE AS 1A–3A, EXCEPT THAT THE SINGLE-SHOT CAPTURE CYCLE DOES NOT BEGIN UNTIL THE FIRST FALLING EDGE IS DETECTED.

Figure 7-9. Type 2 Timer Application Example—Measure Duty Cycle

7.4.5 Overflow/Interrupt on Cumulative Time

To cause an overflow only when the T0 pin has been low for some cumulative duration, the Type 2 timer can be configured to the gated compare mode of operation with an initial starting value appropriate for the cumulative duration to be detected.

```
; ----------------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2V0, #1234h                  ; Overflow after T0 input low for (10000h-01234h) T2CLKs
MOVE T2CFG0, #01110000b            ;
                                   ; T2DIV[2:0]   =111   (/128)
                                   ; T2MD    =0    (16-bit)
                                   ; CCF[1:0]     =00    (no edges)
                                   ; C/T2    =0    (timer/compare)
MOVE T2CNA0, #10101001b            ; ET2     =1    (enable Timer ints)
                                   ; T2OE0   =0    (input)
                                   ; T2POL0  =1    (gating level = '1')
                                   ; TR2L:TR2     =01   (start timer)
                                   ; CPRL2   =0    (no capture possible)
                                   ; SS2     =0    (not single shot mode)
                                   ; G2EN    =1    (gating enabled)
; ----------------- TF2 Interrupt : Cumulative low duration reached
```
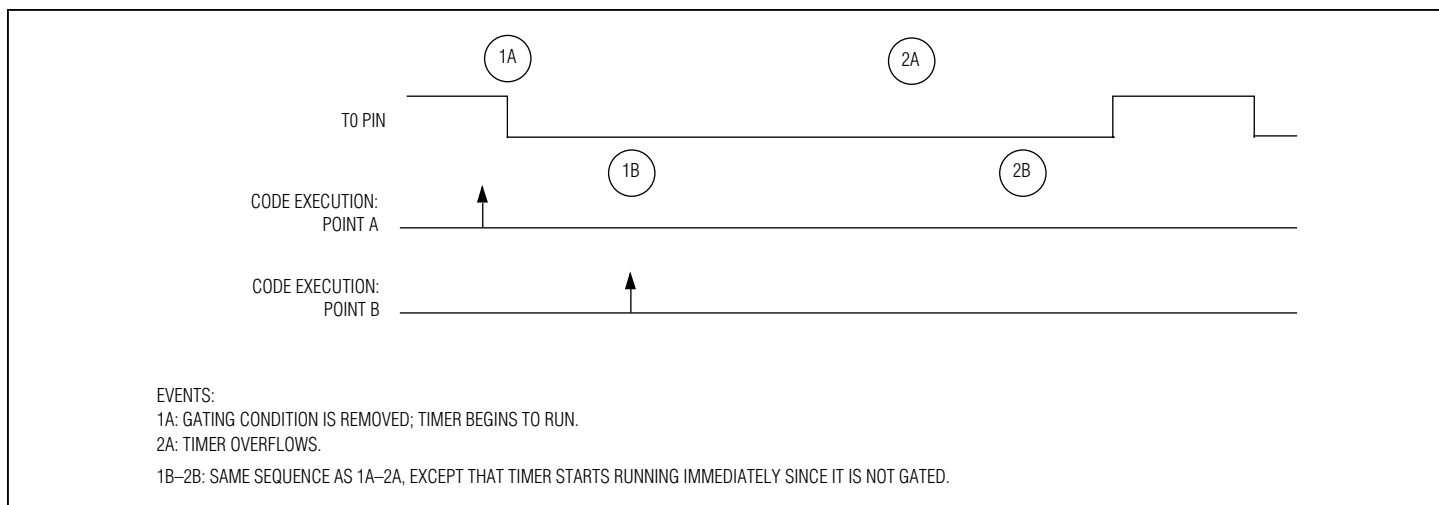

Figure 7-10. Type 2 Timer Application Example—Overflow/Interrupt on Cumulative Time

7.5 Type 2 Timer/Counter Compare Application Example

The following example is used to demonstrate the Type 2 timer compare function.

7.5.1 A Simple Waveform Output

To output a simple waveform on the T0 pin whose frequency and duty cycle can be configured with an appropriate initial starting value for T2R0 and T2C0 registers.

```
; --------- Reset State:  T2R0 = T2V0 = T2C0 = 0000h -----------------------
MOVE T2V0, #4000h                  ; Set to reload value to keep first pulse from being extra long
MOVE T2R0, #4000h                  ; Reload value
MOVE T2C0, #C000h                  ; T0 output high for (C000h – 4000h) T2CLKS
MOVE T2CFG0, #01110000b            ;
                                   ; T2DIV[2:0] = 111 (/128)
                                   ; T2MD = 0 (16-bit)
                                   ; CCF[1:0] = 00 (compare mode)
                                   ; C/T2 = 0 (timer/compare)
MOVE T2CNA0, #11101000b            ; ET2 = 1 (enable Timer ints)
                                   ; T2OE0 = 1 (pin enabled as output)
                                   ; T2POL0 = 0 (high start value on pin)
                                   ; TR2L:TR2 = 01 (start timer)
                                   ; CPRL2 = 0 (no capture possible)
                                   ; SS2 = 0 (not single shot mode)
                                   ; G2EN = 0 (gating disbled)
; -------- TCC2 Interrupt: Compare match; high duration reached
; -------- TF2 Interrupt: Overflow/Reload; low duration reached
```
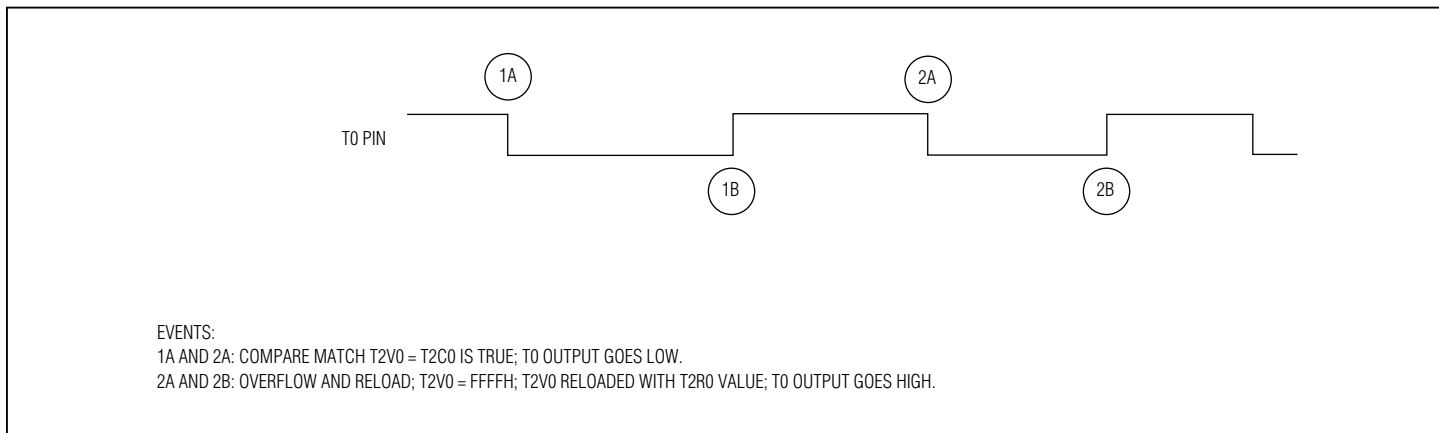

Figure 7-11. Type 2 Timer Compare Application Example—A Simple Waveform Output

SECTION 8: GENERAL-PURPOSE I/O MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 8: GENERAL-PURPOSE I/O MODULE

The MAXQ7665/MAXQ7666 smart data-acquisition microcontrollers provide 8 port pins for general-purpose I/O, which are grouped into the logical port P0. The P0 port pins have the following features:

- All pins are multiplexed with alternate functions
- CMOS-compatible I/O levels to VDDIO and GND rails
- User-selectable, weak pullup resistors when configured as inputs (power-on state)
- Push-pull or open-drain output (can use internal pullup for open drain)
- Rising or falling edge selectable interrupt or wakeup inputs on all digital I/O pins
- Low leakage

8.1 Architecture

The MAXQ7665/MAXQ7666 support one Type D port P0. Type D is a bidirectional I/O port that incorporates Schmitt trigger receivers and full CMOS output drivers, and can support alternate functions. The pin is either three-stated or weakly pulled up when defined as an input. All Type D pins also have interrupt capability.

All port P0 pins can support special function (SF). Enabling the special function automatically converts the pin to that function. Special function is usually implemented in another functional module and supported by individual enable or status bits.

Figure 8-1 illustrates a Type D port pin function. The pin logic of each port pin is identical.
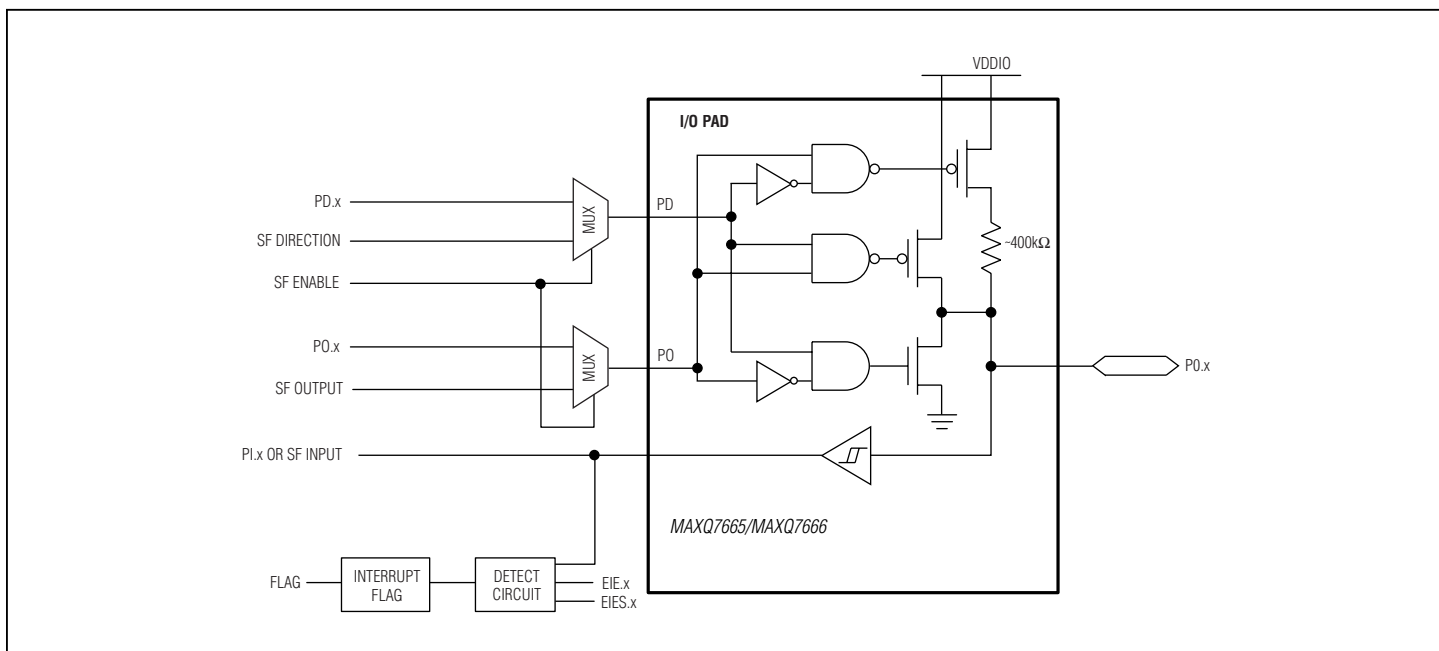
Figure 8-1. Type D Port Pin Schematic

MAXQ7665/MAXQ7666 User's Guide

8.1.1 Port Pins

The MAXQ7665/MAXQ7666 port P0 pins are summarized in Table 8-1.

Table 8-1. MAXQ7665/MAXQ7666 Port P0 Pins

PORT P0 SIGNALS	PIN NUMBER 48-PIN	PIN NUMBER 56-PIN	FUNCTION
P0.0/TDO	32	37	Port 0 Data 0/JTAG Serial Test Data Output. P0.0 is a general-purpose digital I/O with interrupt/wakeup capability. TDO is the JTAG serial test data output. After power-up or a reset this pin defaults to JTAG TDO pin.
P0.1/TMS	33	38	Port 0 Data 1/JTAG Test Mode Select. P0.1 is a general-purpose digital I/O with interrupt/wakeup capability. TMS is the JTAG test mode select input. After power-up or a reset this pin defaults to JTAG TMS pin.
P0.2/TDI	34	39	Port 0 Data 2/JTAG Test Data Input. P0.2 is a general-purpose digital I/O with interrupt/wakeup capability. TDI is the JTAG serial test data input. After power-up or a reset this pin defaults to JTAG TDI pin.
P0.3/TCK	35	40	Port 0 Data 3/JTAG Test Clock Input. P0.3 is a general-purpose digital I/O with interrupt/wakeup capability. TCK is the JTAG serial test clock input. After power-up or a reset this pin defaults to JTAG TCK pin.
P0.4/ADCCNV	36	41	Port 0 Data 4/ADC Conversion Start Input. P0.4 is a general-purpose digital I/O with interrupt/wakeup capability. ADCCNV is the ADC conversion start input signal that can trigger ADC sampling and conversion on a rising or falling edge. After power-up or a reset this pin defaults to a weakly pulled up general-purpose input.
P0.5/DACLOAD	37	43	Port 0 Data 5/DAC Load Input. P0.5 is a general-purpose digital I/O with interrupt/wakeup capability. DACLOAD is the DAC load input signal that can trigger DAC conversion by loading the DAC output register on a rising or falling edge. After power-up or a reset this pin defaults to a weakly pulled up general-purpose input.
P0.6/T0	24	27	Port 0 Data 6/Timer 0 Input/Output. P0.6 is a general-purpose digital I/O with interrupt/wakeup capability. As Timer 0 Input/Output, the pin supports clock gating, capture/compare, counter, and PWM functionalities. After power-up or a reset this pin defaults to a weakly pulled up general-purpose input.
P0.7/T1	25	29	Port 0 Data 7/Timer 1 Input/Output. P0.7 is a general-purpose digital I/O with interrupt/wakeup capability. As Timer 1 Input/Output, the pin supports clock gating, capture/compare, counter, and PWM functionalities. After power-up or a reset this pin defaults to a weakly pulled up general-purpose input.

8.2 Port Registers

The following peripheral registers control the general-purpose I/O and external interrupt features specific to the MAXQ7665/MAXQ7666.

8.2.1 Port 0 Output Register (PO0)

Register Description: **Port 0 Output Register**
Register Name: **PO0**
Register Address: **Module 00h, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	PO0.7	PO0.6	PO0.5	PO0.4	PO0.3	PO0.2	PO0.1	PO0.0
Reset	1	1	1	1	1	1	1	1
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to FFh on all forms of reset.*

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bits 7 to 0: Port 0 Output Register Bits 7 to 0 (PO0.7 to PO0.0). Port 0 is a Type D I/O port. The PO0 register stores output data for port 0 when it is defined as an output port and controls whether the internal pullup resistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 0 does not change the data contents of the register.

8.2.2 External Interrupt Flag Register (Port 0) (EIF0)

Register Description: **External Interrupt Flag Register (Port 0)**
Register Name: **EIF0**
Register Address: **Module 00h, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bit 7: Bit 7 Edge Detect (IE7). This bit is set when a negative edge (IT7 = 1) or a positive edge (IT7 = 0) is detected on the interrupt 7 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 6: Bit 6 Edge Detect (IE6). This bit is set when a negative edge (IT6 = 1) or a positive edge (IT6 = 0) is detected on the interrupt 6 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 5: Bit 5 Edge Detect (IE5). This bit is set when a negative edge (IT5 = 1) or a positive edge (IT5 = 0) is detected on the interrupt 5 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 4: Bit 4 Edge Detect (IE4). This bit is set when a negative edge (IT4 = 1) or a positive edge (IT4 = 0) is detected on the interrupt 4 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 3: Bit 3 Edge Detect (IE3). This bit is set when a negative edge (IT3 = 1) or a positive edge (IT3 = 0) is detected on the interrupt 3 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 2: Bit 2 Edge Detect (IE2). This bit is set when a negative edge (IT2 = 1) or a positive edge (IT2 = 0) is detected on the interrupt 2 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 1: Bit 1 Edge Detect (IE1). This bit is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the interrupt 1 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

Bit 0: Bit 0 Edge Detect (IE0). This bit is set when a negative edge (IT0 = 1) or a positive edge (IT0 = 0) is detected on the interrupt 0 pin. Setting this bit to 1 generates an interrupt to the CPU if enabled. This bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt will be generated as long as this bit is set. Note that this flag register simply indicates whether an edge has been detected at port 0 and is not affected by the state of EIE0.

8.2.3 Port 0 Input Register (PI0)

Register Description: **Port 0 Input Register**
Register Name: **PI0**
Register Address: **Module 00h, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	PI0.7	PI0.6	PI0.5	PI0.4	PI0.3	PI0.2	PI0.1	PI0.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

r = read, s = dependent on pin's state
Note: The reset value for this register is dependent on the logical states of the pins.

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bits 7 to 0: Port 0 Input Register Bits 7 to 0 (PI0.7 to PI0.0). Port 0 is a Type D I/O port. The PI0 register always reflects the logic state of its pins when read.

8.2.4 External Interrupt Enable Register (Port 0) (EIE0)

Register Description: **External Interrupt Enable Register (Port 0)**
Register Name: **EIE0**
Register Address: **Module 00h, Index 0Bh**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	EX7	EX6	EX5	EX4	EX3	EX2	EX1	EX0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bit 7: Enable External Interrupt 7 (EX7). Setting this bit to 1 enables external interrupt on port pin P0.7. Clearing this bit to 0 disables the interrupt function.

Bit 6: Enable External Interrupt 6 (EX6). Setting this bit to 1 enables external interrupt on port pin P0.6. Clearing this bit to 0 disables the interrupt function.

Bit 5: Enable External Interrupt 5 (EX5). Setting this bit to 1 enables external interrupt on port pin P0.5. Clearing this bit to 0 disables the interrupt function.

Bit 4: Enable External Interrupt 4 (EX4). Setting this bit to 1 enables external interrupt on port pin P0.4. Clearing this bit to 0 disables the interrupt function.

Bit 3: Enable External Interrupt 3 (EX3). Setting this bit to 1 enables external interrupt on port pin P0.3. Clearing this bit to 0 disables the interrupt function.

Bit 2: Enable External Interrupt 2 (EX2). Setting this bit to 1 enables external interrupt on port pin P0.2. Clearing this bit to 0 disables the interrupt function.

Bit 1: Enable External Interrupt 1 (EX1). Setting this bit to 1 enables external interrupt on port pin P0.1. Clearing this bit to 0 disables the interrupt function.

Bit 0: Enable External Interrupt 0 (EX0). Setting this bit to 1 enables external interrupt on port pin P0.0. Clearing this bit to 0 disables the interrupt function.

8.2.5 Port 0 Direction Register (PD0)

Register Description: **Port 0 Direction Register**
Register Name: **PD0**
Register Address: **Module 00h, Index 10h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	PD0.7	PD0.6	PD0.5	PD0.4	PD0.3	PD0.2	PD0.1	PD0.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bits 7 to 0: Port 0 Direction Register Bits 7 to 0 (PD0.7 to PD0.0). Port 0 is a Type D I/O port. The PD0 register is used to determine the direction of each pin that makes up the port. The port pins are independently controlled by their direction bit. When a bit in PD0 is set to 1, its corresponding pin is enabled as an output. The data value in the respective bit of the PO register will be driven on the pin. When a bit in PD0 is cleared to 0, its corresponding pin is available as an input, and allows an external signal to drive the pin. Note that each port pin has a weak pullup resistor when functioning as an input, which is controlled by the respective PO bit. If the PO bit is set to 1, the pullup is enabled; if the PO bit is cleared to 0, the pullup is disabled and the port pin is in high impedance three-state.

8.2.6 External Interrupt Edge Select Register (Port 0) (EIES0)

Register Description: **External Interrupt Edge Select Register (Port 0)**
Register Name: **EIES0**
Register Address: **Module 00h, Index 13h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	IT7	IT6	IT5	IT4	IT3	IT2	IT1	IT0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 00h on all forms of reset.*

Bits 15 to 8: Reserved Read returns 0, write ignored.

Bit 7: Edge Select for External Interrupt 7 (IT7)

> IT7 = 0: External interrupt 7 is positive-edge triggered.
> IT7 = 1: External interrupt 7 is negative-edge triggered.

Bit 6: Edge Select for External Interrupt 6 (IT6)

> IT6 = 0: External interrupt 6 is positive-edge triggered.
> IT6 = 1: External interrupt 6 is negative-edge triggered.

Bit 5: Edge Select for External Interrupt 5 (IT5)

> IT5 = 0: External interrupt 5 is positive-edge triggered.
> IT5 = 1: External interrupt 5 is negative-edge triggered.

Bit 4: Edge Select for External Interrupt 4 (IT4)

> IT4 = 0: External interrupt 4 is positive-edge triggered.
> IT4 = 1: External interrupt 4 is negative-edge triggered.

Bit 3: Edge Select for External Interrupt 3 (IT3)

> IT3 = 0: External interrupt 3 is positive-edge triggered.
> IT3 = 1: External interrupt 3 is negative-edge triggered.

Bit 2: Edge Select for External Interrupt 2 (IT2)

> IT2 = 0: External interrupt 2 is positive-edge triggered.
> IT2 = 1: External interrupt 2 is negative-edge triggered.

Bit 1: Edge Select for External Interrupt 1 (IT1)

> IT1 = 0: External interrupt 1 is positive-edge triggered.
> IT1 = 1: External interrupt 1 is negative-edge triggered.

Bit 0: Edge Select for External Interrupt 0 (IT0)

> IT0 = 0: External interrupt 0 is positive-edge triggered.
> IT0 = 1: External interrupt 0 is negative-edge triggered.

8.3 GPIO Operation

From a software perspective, the MAXQ7665/MAXQ7666 port P0 appears as a group of peripheral registers with unique addresses and is addressed as a byte or 8 individual bit locations. The port is designed to provide programming flexibility for the user application.

- All individual I/O bits are independently configured.
- Any combination of input, output, or alternate function in a port is permitted.
- All I/O pins have protection circuitry to VDDIO and ground.

8.3.1 Port P0 Direction Control and Input/Output

The port 0 direction register (PD0) controls the MAXQ7665/MAXQ7666 port P0 pin input/output direction. The port 0 input register (PI0) is a read-only register that always reflects the logic state on the pins. The port 0 output register (PO0) has a dual function. For pins defined as output, PO0 stores output data, and for pins defined as input, PO0 controls whether the internal weak pullup is enabled or disabled. The port P0 pins input/output states in standard mode (no special or alternate function enabled) are according to Table 8-2.

Table 8-2. Port P0 Pin Input/Output States (in Standard Mode)

PD0.x	PO0.x	PORT PIN MODE	PORT PIN (P0.x) STATE
0	0	Input	Three-state
0	1	Input	Weak pullup HIGH
1	0	Output	Strong drive LOW
1	1	Output	Strong drive HIGH

The port P0 can be used to support applications that require open-drain functionality. This can be achieved by using the PO0 and PD0 register of the port.

- Three-state the port pin needed to be open drain by setting the corresponding PD0 bit to 0.
- Clear the corresponding PO0 bit to 0.
- Use the corresponding PD0 bit to drive the port pin function, instead of the 00 register.

Note that the internal pullup has a relatively high impedance (typically ~400kΩ), so a particular system may require a stronger (external) pullup to meet the system level needs.

8.3.2 Port P0 External Interrupts

Each of the port P0 pins can function as an external interrupt with individual enable, flag, and active edge selection bits.

- External interrupt enable register (EIE0) bits determine if the external interrupt functionality at each pin is enabled or not.
- External interrupt edge select register (EIES0) bits determine if the external interrupt is generated on rising or falling edge of the interrupt pin input.
- External interrupt flag register (EIF0) bits indicate if a valid rising or falling edge has been detected on the interrupt pin input. An interrupt is generated only if the external interrupt functionality is enabled for the pin. Also, global interrupt mask bits IM0 (in the IMR register) and IGE (in the IC register) must be enabled.

Note: The detection of a valid interrupt edge on any of the external interrupt pins can act as a switchback-trigger source, causing the microcontroller to switch back from power management mode (PMME = 1, clock set to divide-by-256) to the standard divide-by-1 system clock frequency.

8.3.3 Port P0 Special and Alternate Functions

All the MAXQ7665/MAXQ7666's port pins are multiplexed with special functions as listed in Table 8-3. All these special functions are disabled by default with the exception of the JTAG interface pins, which are enabled by default following any reset. The behavior of these functions breaks down into two categories:

- Special functions override the PD0 and PO0 settings for the port pin when they are enabled. Once the special function takes control, normal control of the port pin is lost until the special function is disabled. Examples of special functions include timer 0 and timer 1 output.
- Alternate functions operate in parallel with PD0 and PO0 settings for the port pin, and generally consist of input-only functions such as external interrupts. When an alternate function is enabled for a port pin, the port pin's output state is still controlled in the usual manner.

Table 8-3. Port Pin Special and Alternate Functions

PORT PIN	FUNCTION TYPE	FUNCTION	ENABLED WHEN	MULTIPLEXING/PRIORITIZATION
P0.0	Special	TDO—JTAG Data Out, Output	TAP = 1	The JTAG is the default interface; this port pin is configured as an output and is ready for JTAG operation after a reset.
	Alternate	INT0—External Interrupt 0, Input	EX0 = 1	
P0.1	Special	TMS—JTAG Mode Select, Input	TAP = 1	This pin defaults to a weak pullup input and is ready for JTAG operation after a reset.
	Alternate	INT1—External Interrupt 1, Input	EX1 = 1	
P0.2	Special	TDI—JTAG Data In, Input	TAP = 1	This pin defaults to a weak pullup input and is ready for JTAG operation after a reset.
	Alternate	INT2—External Interrupt 2, Input	EX2 = 1	
P0.3	Special	TCK—JTAG Clock In, Input	TAP = 1	This pin defaults to a weak pullup input and is ready for JTAG operation after a reset.
	Alternate	INT3—External Interrupt 3, Input	EX3 = 1	
P0.4	Special	ADCCNV—ADC Conversion Start, Input	ADCS2:ADCS0 = 100 or 101	This pin defaults to a weak pullup input after a reset.
	Alternate	INT4—External Interrupt 4, Input	EX4 = 1	
P0.5	Special	DACLOAD—DAC Load, Input	DACLD2:DACLD0 = 000, 100, or 101	This pin defaults to a weak pullup input after a reset.
	Alternate	INT5—External Interrupt 5, Input	EX5 = 1	
P0.6	Special	Timer 0 (Type 2) Output	T2OE0 = 1	This pin defaults to a weak pullup input after a reset.
	Special	Timer 0 (Type 2) Counter Input	C/T2 = 1; CCF1 - CCF0 ≠ 00b; T2OE0 must be 0	
	Special	Timer 0 (Type 2) Gate Input	G2EN = 1 or CCF1:CCF0 = 11b and CPRL2 = 1; T2OE0 must be 0; C/T2 must be 0	
	Alternate	INT6—External Interrupt 6, Input	EX6 = 1	
P0.7	Special	Timer 1 (Type 2) Output	T2OE0 = 1	This pin defaults to a weak pullup input after a reset.
	Special	Timer 1 (Type 2) Counter Input	C/T2 = 1; CCF1:CCF0 ≠ 00b; T2OE0 must be 0	
	Special	Timer 1 (Type 2) Gate Input	G2EN = 1 or CCF1:CCF0 = 11b and CPRL2 = 1; T2OE0 must be 0; C/T2 must be 0	
	Alternate	INT7—External Interrupt 7, Input	EX7 = 1	

8.3.4 Port Pin Examples

8.3.4.1 Port Pin Example 1: Driving Outputs on Port 0

```
move PO0, #000h          ; Set all outputs low
move PD0, #0FFh          ; Set all P0 pins to output mode
```

8.3.4.2 Port Pin Example 2: Receiving Inputs on Port 0

```
move PO0, #0FFh          ; Set weak pullups ON on all pins
move PD0, #000h          ; Set all P1 pins to input mode

nop                      ; Wait for external source to drive P1

move Acc, PI0            ; Get input values from P0
                         ; (will return FF if no other source
                         ; drives the pins low)
```

SECTION 9: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 9: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

The MAXQ7665/MAXQ7666 serial peripheral interface (SPI) module provides an independent serial communication channel to communicate synchronously with peripheral devices in a multiple master or multiple slave system. The interface allows access to a 4-wire full-duplex serial bus that can be operated in either master mode or slave mode. The MAXQ7665/MAXQ7666 SPI features include the following:

- 4-wire synchronous full-duplex communication

- Master or slave mode

- 8-bit or 16-bit character lengths

- Four standard SPI clocking modes

- Programmable baud-rate generator

- Mode-fault detection

- Data overrun and collision detection

- Interrupt or polled operation

- Maximum data rate: 1/8 the system clock for slave and 1/2 the system clock for master mode

9.1 Architecture

Figure 9-1 shows a simplified block diagram of the MAXQ7665/MAXQ7666 SPI. The main element in the SPI module is the block containing the shift register and the read buffer. The shift register serves as the transmit and receive data buffer, while the read buffer is the holding register for data received from the network and ready for the CPU to read. Each time that an SPI transfer completes, the received character is transferred to the read buffer, giving double buffering on the receive side. No buffer overrun will occur as long as the first character is read out of the data buffer before the next character is ready to be transferred into the read buffer. The SPI is single buffered in the transmit direction. New data for transmission cannot be written to the shift register until the previous transfer is completed. The CPU has read/write access to the control unit and the SPI data buffer (SPIB). The SPIB provides access for both transmit data and receive data; reads are directed to the read buffer and writes to the shift register automatically.
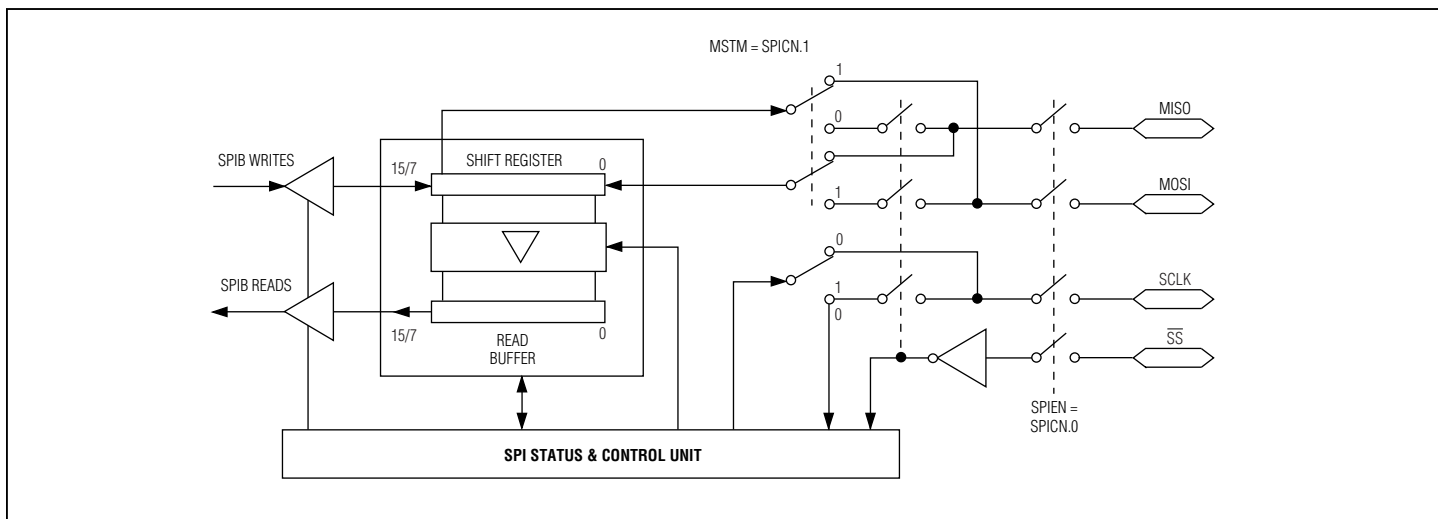
Figure 9-1. SPI Block Diagram

9.1.1 SPI Pins

The SPI signals are shown in Table 9-1.

Table 9-1. MAXQ7665/MAXQ7666 SPI Pins

SPI EXTERNAL SIGNAL	PIN NUMBER		MASTER MODE USE	SLAVE MODE USE
	48	56		
MISO—Master In, Slave Out		35	Input to serial shift register.	Output from serial shift register when selected. Data sent most significant bit first.
MOSI—Master Out, Slave In		34	Output from serial shift register. Data sent most significant bit first.	Input to serial shift register when selected.
SCLK	—	33	Serial shift clock sourced to slave device(s).	Serial shift clock from an external master.
$\overline{SS}$		32	(Optional) Mode-fault-detection input if enabled (MODFE = 1).	Slave select input.

9.2 SPI Peripheral Registers

The MAXQ7665/MAXQ7666 SPI peripheral registers are described here. All the SPI peripheral registers are directly accessible by the microcontroller through the module/index address.

9.2.1 SPI Data Buffer Register (SPIB)

Register Description: **SPI Data Buffer Register**
Register Name: **SPIB**
Register Address: **Module 01h, Index 06h**

Bit #	15	14	13	12	11	10	9	8
Name	SPIB.15	SPIB.14	SPIB.13	SPIB.12	SPIB.11	SPIB.10	SPIB.9	SPIB.8
Reset	0	0	0	0	0	0	0	0
Access	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

Bit #	7	6	5	4	3	2	1	0
Name	SPIB.7	SPIB.6	SPIB.5	SPIB.4	SPIB.3	SPIB.2	SPIB.1	SPIB.0
Reset	0	0	0	0	0	0	0	0
Access	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

*r = read, w = write, * = write allowed only when STBY = 0*

Bits 15 to 0: SPIB Data Bits 15 to 0 (SPIB.15 to SPIB.0). Data for SPI is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location. Write access is allowed only outside of the transfer cycle. When the STBY bit (SPICN.7) is set, write attempts are blocked and cause a write collision error.

9.2.2 SPI Control Register (SPICN)

Register Description: **SPI Control Register**
Register Name: **SPICN**
Register Address: **Module 01h, Index 07h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved. Read 0, write ignored.

Bit 7: SPI Transfer Busy Flag (STBY). This bit is used to indicate the current transmit/receive activity of the SPI module. STBY is set to 1 when an SPI transfer cycle starts and is cleared to 0 when the transfer cycle is completed. This bit is controlled by hardware and is read only for user software.

    0 = SPI module is idle—no transfer in progress.
    1 = SPI transfer in progress.

Bit 6: SPI Transfer Complete Flag (SPIC). This bit signals the completion of an SPI transfer cycle. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

    0 = No SPI transfers have completed since the bit was last cleared.
    1 = SPI transfer complete.

Bit 5: Receive Overrun Flag (ROVR). This bit indicates a receive overrun has occurred. A receive overrun results when a received character is ready to be transferred to the SPI receive data buffer before the previous character in the data buffer is read. The most recent receive data is lost. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

    0 = No receive overrun has occurred.
    1 = Receive overrun occurred.

Bit 4: Write Collision Flag (WCOL). This bit signifies that an attempt was made by software to write the SPI buffer (SPIB) while a transfer was in progress (STBY = 1). Such attempts will always be blocked. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

    0 = No write collision has been detected.
    1 = Write collision detected.

Bit 3: Mode-Fault Flag (MODF). This bit is the mode-fault flag for SPI master mode operation. When mode fault detection is enabled (MODFE = 1) in master mode, detection of high-to-low transition on the $\overline{SS}$ pin signifies a mode fault causes MODF to be set to 1. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled. This flag has no meaning in slave mode.

    0 = No mode fault has been detected.
    1 = Mode fault detected while operating as a master (MSTM = 1).

Bit 2: Mode-Fault Enable (MODFE). When to set logic 1, the $\overline{SS}$ input pin is used for mode fault detection during SPI master mode operation. When cleared to 0, the $\overline{SS}$ input has no function. In slave mode, the $\overline{SS}$ pin always functions as a slave-select input signal to the SPI module, independent of the setting of the MODFE bit.

Bit 1: Master Mode Enable (MSTM). The MSTM bit functions as the master mode enable bit for the SPI module. Note that this bit can be set from 0 to 1 only when the $\overline{SS}$ signal is deasserted. This bit is automatically cleared to 0 by hardware if a mode fault is detected.

    0 = SPI module operates in slave mode when enabled (SPIEN = 1).
    1 = SPI module operates in master mode when enabled (SPIEN = 1).

Bit 0: SPI Enable (SPIEN)

    0 = SPI module and its baud-rate generator are disabled.
    1 = SPI module and its baud-rate generator are enabled.

9.2.3 SPI Configuration Register (SPICF)

Register Description: **SPI Configuration Register**
Register Name: **SPICF**
Register Address: **Module 01h, Index 08h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ESPII	—	—	—	—	CHR	CKPHA	CKPOL
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	r	r	rw	rw	rw

r = read, w = write

Bits 15 to 8, 6 to 3: Reserved. Read 0, write ignored.

Bit 7: Enable SPI Interrupt (ESPII). This bit enables any of the SPI interrupt source flags (MODF, WCOL, ROVR, SPIC) to generate interrupt requests. **Note:** For interrupt requests to happen, global interrupt mask bits IM1 (in the IMR register) and IGE (in the IC peripheral register) must also be enabled.

    0 = SPI interrupt sources disabled.
    1 = SPI interrupt sources enabled.

Bit 2: Character Length Select (CHR). This bit determines the character length for an SPI transfer cycle. A character can be 8 bits in length or 16 bits in length.

    0 = 8-bit character length specified.
    1 = 16-bit character length specified.

Bit 1: Clock Phase Select (CKPHA). This bit selects the clock phase and is used with the CKPOL bit to define the SPI data transfer format.

    0 = Data sampled on the active clock edge.
    1 = Data sampled on the inactive clock edge.

Bit 0: Clock Polarity Select (CKPOL). This bit selects the clock polarity and is used with the CKPHA bit to define the SPI data transfer format.

    0 = Clock idles in the logic 0 state (rising = active clock edge).
    1 = Clock idles in the logic 1 state (falling = active clock edge).

9.2.4 SPI Clock Register (SPICK)

Register Description: **SPI Clock Register**
Register Name: **SPICK**
Register Address: **Module 01h, Index 09h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	CKR.7	CKR.6	CKR.5	CKR.4	CKR.3	CKR.2	CKR.1	CKR.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 8: Reserved. Read 0, write ignored.

Bits 7 to 0: Clock Divide Ratio (CKR.7 to CKR.0). This 8-bit value determines the system clock divide ratio to be used for SPI master mode baud-clock generation. This register has no function when operating in slave mode, as the SPI clock generation circuitry is disabled. The frequency of the SPI master mode baud rate is calculated using the following equation:

$$\text{SPI baud rate} = 0.5 \times \text{system clock frequency} / (CKR7{:}CKR0 + 1)$$

9.3 SPI Operation

The MAXQ7665/MAXQ7666 SPI can be viewed as a synchronous serial I/O port that shifts a data stream of 8 or 16 bits between peripheral devices. Data is shifted in and out of the SPI through the programmable shift registers that are formed by serially connecting the master's shift register and a slave shift register. The SPI bus is typically implemented with one master device and multiple slave devices. Each slave device has a unique $\overline{SS}$ pin that is used to enable transfers to that device. Figure 9-2 shows a typical SPI bus configuration.

During an SPI transfer, data is simultaneously transmitted and received. The serial clock signal (SCLK) synchronizes shifting and sampling of the bit stream on the two serial data pins. For both the master and the slave, data is shifted out of the shift registers on one edge of SCLK and latched into the shift registers on the opposite SCLK clock edge.

The SPI module operates in one of two modes once enabled by setting the SPI enable bit (SPIEN) in the SPI control register. The master mode bit (MSTM) selects the operating mode and the source of the SCLK signal.
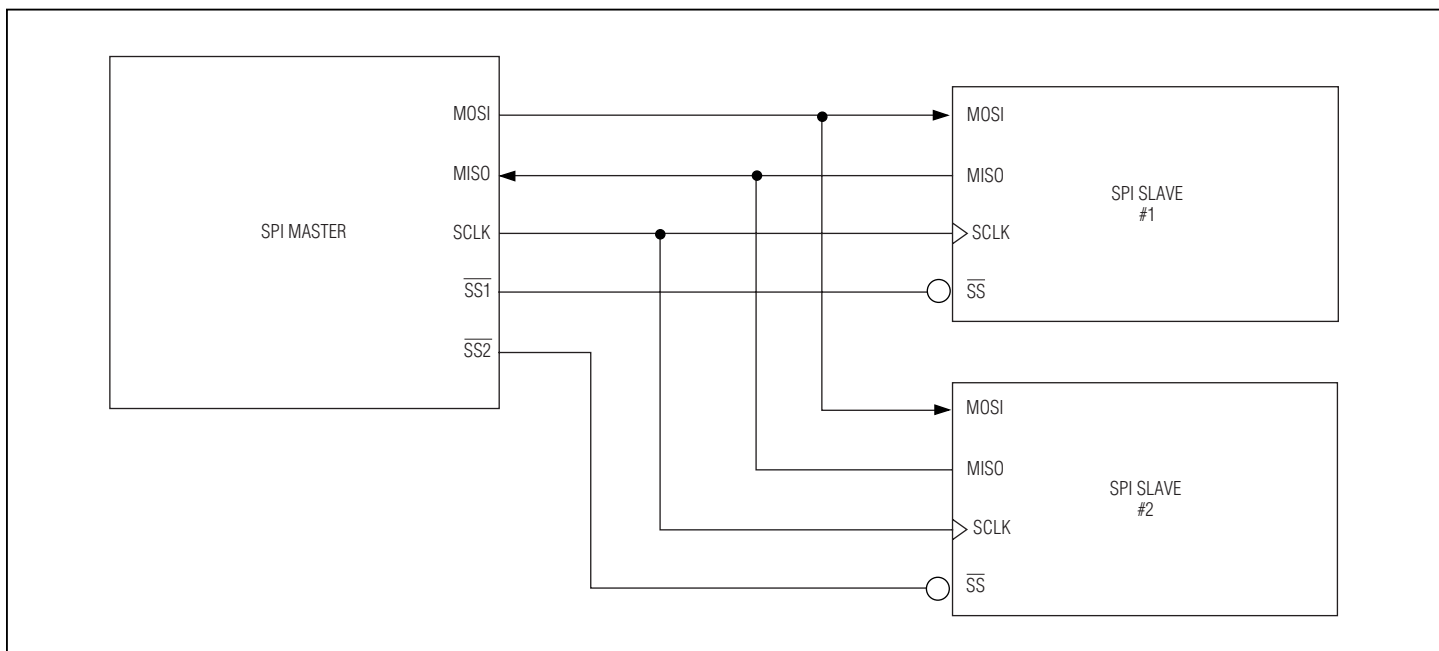
Figure 9-2. SPI Bus Configuration

9.3.1 SPI Master Operation

The MAXQ7665/MAXQ7666 SPI module is placed in master mode by setting the master mode enable (MSTM) bit in the SPI control register to logic 1.

Only an SPI master device can initiate a data transfer. The master is responsible for manually selecting/deselecting the desired slave devices. This can be done using a general-purpose output pin. Writing a data character to the SPI shift register (SPIB) while in master mode starts a data transfer. The SPI master immediately shifts out the data serially on the MOSI pin, most significant bit first, while providing the serial clock on SCLK output. New data is simultaneously received on the MISO pin into the least significant bit of the shift register.

The data transfer format (clock polarity and phase), character length, and baud rate are configurable as described in later sections. During the transfer, the SPI transfer busy (STBY) flag is set to indicate that a transfer is in progress. At the end of the transfer, the data contained in the shift register is moved into the receive data buffer, the STBY bit is cleared by hardware, and the SPI transfer complete flag (SPIC) is set. Setting the SPIC bit generates an interrupt request if SPI interrupt sources are enabled (ESPII = 1). Also, for an interrupt request to be generated global interrupt masks IM1 (in the IMR register) and IGE (in the IC peripheral register) must also be enabled.

9.3.2 SPI Slave Operation

The MAXQ7665/MAXQ7666 SPI module operates in slave mode when the MSTM bit is cleared to logic 0. In slave mode, the SPI module is dependent on the SCLK signal sourced from the master to control the data transfer. The SCLK input frequency should be no greater than the system clock of the MAXQ7665/MAXQ7666 slave device divided by 8.

The slave select ($\overline{SS}$) input must be externally asserted by a master before data exchange can take place. $\overline{SS}$ must be low before data transaction begins and must remain low for the duration of the transaction. If data is to be transmitted by the slave device, it must be written to its shift register before the beginning of a transfer cycle, otherwise the character already in the shift register will be transferred. The slave device considers a transfer to begin with the first clock edge or the falling edge of $\overline{SS}$, dependent on the data transfer format.

The SPI slave receives data from the external master MOSI pin, most significant bit first, while simultaneously transferring the contents of its shift register to the master on the MISO pin, also most significant bit first. Data received from the external master replaces data in the internal shift register until the transfer completes. Just like the master mode of operation, received data is loaded into the read buffer and the SPI transfer complete flag is set at the end of transfer. The setting of the transfer complete flag generates an interrupt request if enabled.

When $\overline{SS}$ is not asserted, the slave device ignores the SCLK clock and the shift register is disabled. Under this condition, the device is basically idle, no data is shifted out from the shift register, and no data is sampled from the MOSI pin. The MISO pin is placed in an input mode and is weakly pulled high to allow other devices on the bus to drive the bus. Deassertion of the $\overline{SS}$ signal by the master during a transfer (before a full character, as defined by the CHR, is received) aborts the current transfer. When the transfer is aborted no data is loaded into the read buffer, the SPIC flag is not set, and the slave logic and bit counter are reset.

In slave mode, the clock divide ratio bits (CKR7:CKR0) have no function since an external master supplies the serial clock. The transfer format (CKPOL, CKPHA settings) and the character length selection (CHR) for the slave device, however, should match the master for proper communication.

9.3.3 SPI Transfer Formats

During an SPI transfer, data is simultaneously transmitted and received over two serial data lines with respect to a single serial shift clock. The polarity and phase of the serial shift clock are the primary components in defining the SPI data transfer format. The polarity of the serial clock corresponds to the idle logic state of the clock line and therefore also defines which clock edge is the active edge. To define a serial shift clock signal that idles in a logic-low state (active clock edge = rising), the clock polarity select (SPICF.0: CKPOL) bit should be configured to 0, while CKPOL = 1 causes the shift clock to idle in a logic-high state (active clock edge = falling). The phase of the serial clock selects which edge is used to sample the serial shift data. The clock phase select (SPICF.1:CKPHA) bit controls whether the active or inactive clock edge is used to latch the data. When CKPHA is set to logic 1, data is sampled on the inactive clock edge (clock returning to the idle state). When CKPHA is set to logic 0, data is sampled on the active clock edge (clock transition to the active state). Together, the CKPOL and CKPHA bits allow the four possible SPI data transfer formats as illustrated in Figure 9-3.

Anytime that the active clock edge is used for sampling (CKPHA = 0), the transfer cycle must be started with assertion of the $\overline{SS}$ signal. This requirement necessitates that the $\overline{SS}$ signal be deasserted and reasserted between successive transfers. Conversely, when the inactive edge is used for sampling (CKPHA = 1), the $\overline{SS}$ signal may remain low through successive transfers allowing the active clock edge to signal the start of a new transfer.
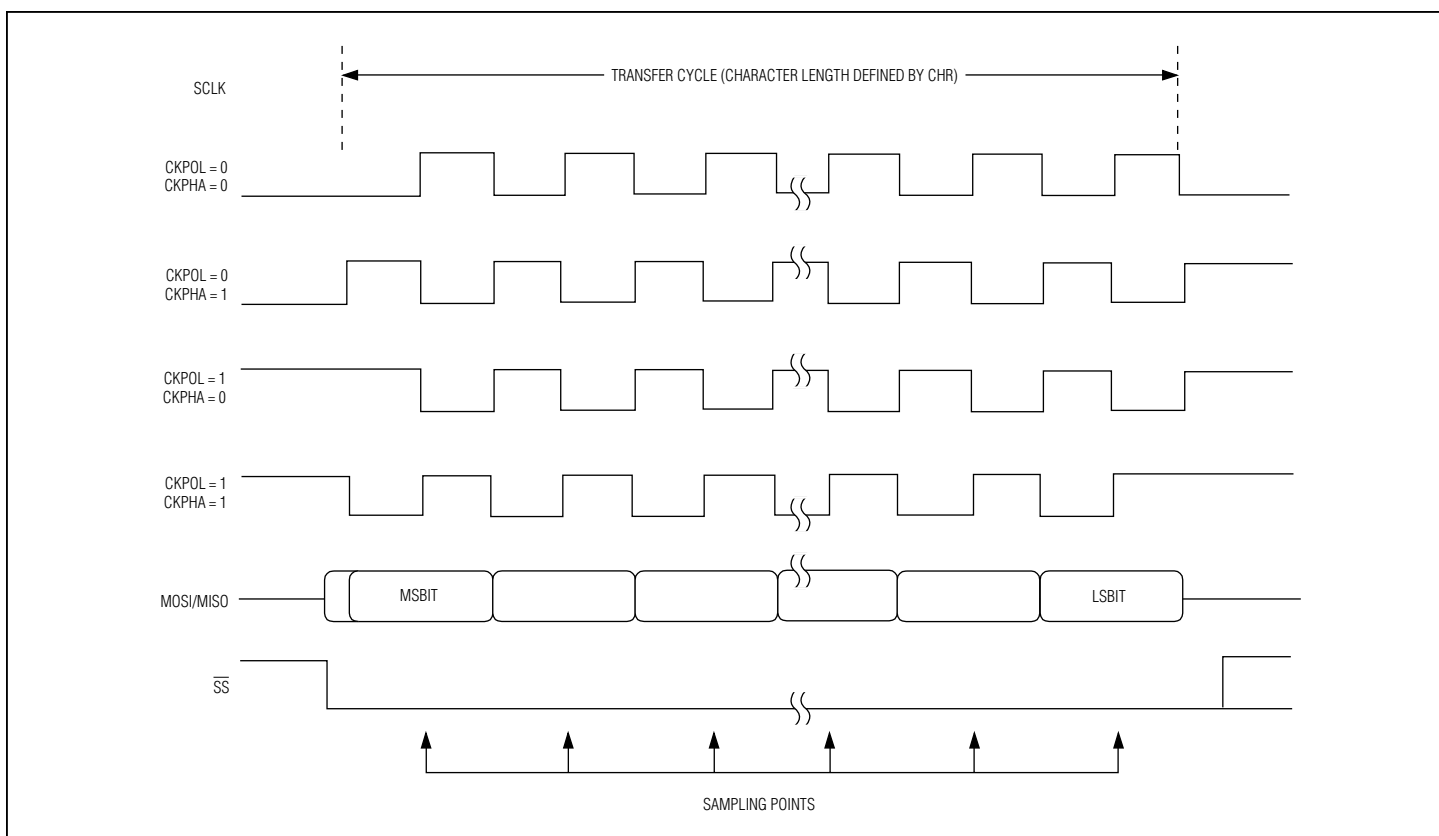

Figure 9-3. SPI Transfer Formats (CKPOL, CKPHA Control)

9.3.4 SPI Character Lengths

To flexibly accommodate different SPI transfer data lengths, the character length for any transfer is user configurable through the character length bit (CHR) in the SPI configuration register. The CHR bit allows selection of either 8-bit or 16-bit transfers.

When loading 8-bit characters into the SPIB data buffer, the byte for transmission should be right justified or placed in the least significant byte of the word. When a byte transfer completes, the received byte is right justified and can be read from the least significant byte of the SPIB word. The MSB of the SPIB data buffer is not significant when transmitting and receiving 8-bit characters.

9.4 SPI Transfer Baud Rates

When operating as a slave device, an external master drives the MAXQ7665/MAXQ7666 SPI serial clock. For proper slave operation, the serial clock provided by the external master should not exceed the system clock frequency divided by 8.

When operating in the master mode, the SPI serial clock is sourced to the external slave device(s). The serial clock baud rate is determined by the clock-divide ratio (CKR) specified in the SPI clock-divide ratio (SPICK) register. The MAXQ7665/MAXQ7666 SPI module supports 256 different clock divide ratio selections for serial clock generation. The SCLK clock rate is determined by the following formula:

$$\text{SPI baud rate} = 0.5 \times \text{system clock frequency} / (\text{CKR7:CKR0} + 1)$$

Since the SPI baud rate is a function of the system clock frequency, using any of the system clock-divide modes (including power management mode) alters the baud rate.

Note, however, that once in power management mode (PMME = 1), writes to SPIB in master mode and assertion of the $\overline{\text{SS}}$ pin slave mode both qualify as switchback sources if enabled (SWB = 1). The MAXQ7665/MAXQ7666 SPI module clocks are halted if the device is placed into stop mode.

9.5 SPI System Errors

The MAXQ7665/MAXQ7666 SPI module can detect three types of SPI system errors. A mode-fault error arises in a multiple master system when more than one SPI device simultaneously tries to be a master. A receive-overrun error occurs when an SPI transfer completes before the previous character has been read from the receive-holding buffer. The third kind of error, write collision, indicates that an attempted write to SPIB was detected while a transfer was in progress (STBY = 1).

9.5.1 Mode Fault

When the MAXQ7665/MAXQ7666 SPI device is configured as a master and its mode-fault enable bit (SPICN.2:MODFE) is also set, a mode-fault error occurs if the $\overline{\text{SS}}$ input signal is driven low by an external device. This error is typically caused when a second SPI device attempts to function as a master in the system. In the condition where more than one device is configured as master concurrently, there is a possibility of bus contention that can cause permanent damage to push-pull CMOS drivers. The mode-fault-error detection is to provide protection from such damage by disabling the bus drivers. When a mode fault is detected, the following actions are taken immediately.

1) The MSTM bit is forced to logic 0 to reconfigure the SPI device as a slave.

2) The SPIEN bit is forced to logic 0 to disable the SPI module.

3) The mode fault (SPICN.3: MODF) status flag is set. Setting the MODF bit can generate an interrupt if it is enabled.

The application software must correct the system conflict before resuming its normal operation. The MODF flag is set automatically by hardware, but must be cleared by software once set. Setting the MODF bit to logic 1 by software causes an interrupt if enabled.

Mode-fault detection is optional and can be disabled by clearing the MODFE bit to logic 0. Disabling the mode-fault detection disables the function of the $\overline{\text{SS}}$ signal during master mode operation.

Note that the mode-fault mechanism does not provide full protection from bus contention in multiple master, multiple slave systems. For example, if two devices are configured as master at the same time, the mode-fault-detect circuitry offers protection only when one of them selects the other as slave by asserting its $\overline{\text{SS}}$ signal. Also, if a master accidentally activates more than one slave and those devices try to simultaneously drive their output pins, bus contention can occur without a mode-fault error being generated.

9.5.2 Receive Overrun

Since the receive direction of the MAXQ7665/MAXQ7666 SPI is double buffered, there is no overrun condition as long as the received character in the read buffer is read before the next character in the shift register is ready to be transferred to the read buffer. However, if previous data in the read buffer has not been read out when a transfer cycle is completed and the new character is ready to be loaded into the read buffer, a receive overrun occurs and the receive overrun flag (SPICN.5: ROVR) is set. Setting the ROVR flag indicates that the most recent received character is lost. Setting the ROVR bit to logic 1 causes an interrupt if enabled. Once set, the ROVR bit is cleared only by software or a reset.

9.5.3 Write Collision While Busy

A write collision occurs if an attempt to write the SPIB data buffer is made during a transfer cycle (STBY = 1). Since the shift register is single buffered in the transmit direction, writes to SPIB are made directly into the shift register. Allowing the write to SPIB while another transfer is in progress could easily corrupt the transmit/receive data. When such a write attempt is made, the current transfer continues undisturbed, the attempted write data is not transferred to the shift register, and the control unit sets the write collision flag (SPICN.4:WCOL). Setting the WCOL bit to logic 1 causes an interrupt if SPI interrupt sources are enabled. Once set, the WCOL bit is cleared only by software or a reset.

Normally, write collisions are associated solely with slave devices since they do not control initiation of transfers and do not have access to as much information about SCLK as the master. As a master, write collisions are completely avoidable, however, the control unit detects write collisions for both master and slave modes.

9.6 SPI Interrupts

Four flags in the SPI control register (SPICN) can generate an SPI interrupt when enabled.

- Mode Fault (MODF)
- Write Collision (WCOL)
- Receive Overrun (ROVR)
- SPI Transfer Complete (SPIC)

These four bits serve as interrupt flags that allow the system programmer to determine the source of interrupts that can cause an interrupt request to the CPU. These bits default to 0 on a reset and must be cleared by software when set.

The ESPII bit in the SPI configuration register (SPICF) enables any of the SPI interrupt source flags (MODF, WCOL, ROVR, SPIC) to generate interrupt requests. For interrupt requests to happen, global interrupt mask bits IM1 (in the IMR register) and IGE (in the IC peripheral register) must also be enabled.

9.7 SPI Example: Enabling Master Mode

```
move    SPICN, #03h         ; Enable SPI for master mode communication
move    SPICF, #00h         ; Rising clock, active edge sample, 8-bit character
move    SPICK, #0Fh         ; Divide by 16 clock
```

SECTION 10: TEST ACCESS PORT (TAP)

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 10: TEST ACCESS PORT (TAP)

10.1 TAP Overview

The MAXQ7665/MAXQ7666 incorporate a test access port (TAP) and TAP controller for communication with a host device across a 4-wire synchronous serial interface. The MAXQ7665/MAXQ7666 use the TAP to support in-system flash programming, in-circuit debug, and device test functions. The MAXQ7665/MAXQ7666 TAP features include the following:

- 4-wire synchronous communication
- TAP signals compatible with JTAG IEEE Standard 1149.1
- Maximum TAP clock frequency limited to 1/8 the system clock

For detailed information on the TAP and TAP controller, refer to IEEE STD 1149.1 *"IEEE Standard Test Access Port and Boundary-Scan Architecture."* Except where explicitly noted, the MAXQ7665 and MAXQ7666 features are identical.

10.2 Architecture

The MAXQ7665/MAXQ7666 TAP controller is a synchronous state machine that responds to changes at the TMS and TCK signals. The TAP state control is achieved through host manipulation of the test mode select (TMS) and test clock (TCK) signals. Based on its state transition, the controller provides the clock and control sequence for TAP operation. The performance of the TAP is dependent on the TCK clock frequency. The maximum TCK clock frequency should be limited to 1/8 the system clock frequency. Figure 10-1 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 TAP and TAP controller.

The TAP provides an independent serial channel to communicate synchronously with the host system. The TMS signal is sampled at the rising edge of TCK and decoded by the TAP Controller to control movement between the TAP states. The TDI input and TDO output are meaningful once the TAP is in a serial shift state.

The TAP controller block has four working registers that control the operation of the port.

- TAP Debug Register
- TAP System Programming Register
- TAP Instruction Register
- TAP Bypass Register

These registers are accessed through the TAP port only and control the sequencing of the TAP state machine. These registers are not accessible from the CPU.
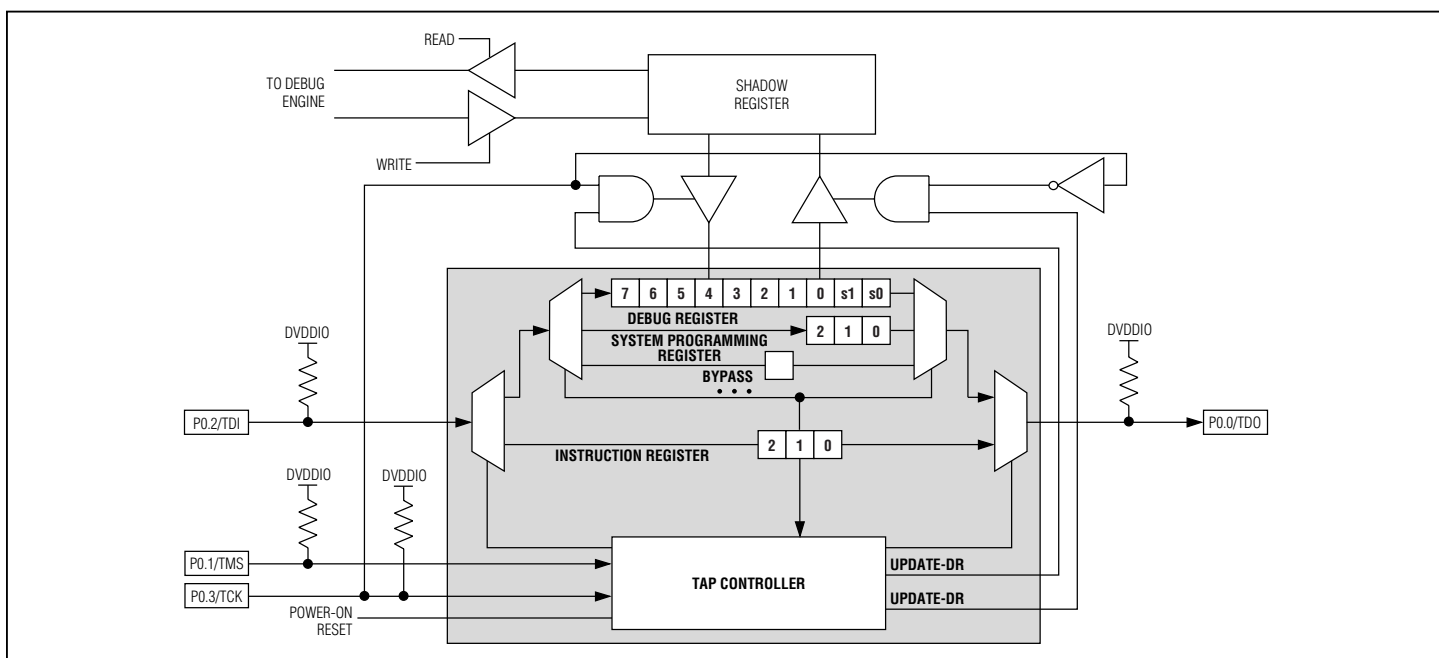
Figure 10-1. MAXQ7665/MAXQ7666 TAP and TAP Controller

10.2.1 TAP Pins

The TAP is formed by four interface signals as described in Table 10-1. The TAP signals are multiplexed with port pins P0.0, P0.1, P0.2, and P0.3. These pins default to their JTAG TAP function on reset, which means that the MAXQ7665/MAXQ7666 will always be ready for in-circuit debugging or in-circuit programming following any reset.

Once an application has been loaded and starts running, the JTAG TAP port can still be used for in-circuit debugging operations. If in-circuit debugging functionality is not needed, the P0.0, P0.1, P0.2, and P0.3 port pins can be reclaimed for application use by setting the TAP (SC.7) bit to 0. This disables the JTAG TAP interface and allows the four pins to operate as normal port pins.

Table 10-1. MAXQ7665/MAXQ7666 TAP Pins

PIN		NAME	MULTIPLEXED WITH PORT SIGNAL	FUNCTION
48	**56**			
32	37	TDO	P0.0	JTAG Serial Test Data Output. This signal is used to serially transfer internal data to the external host. Data is transferred least significant bit first. Data is driven out only on the falling edge of TCK, only during TAP Shift-IR or Shift-DR states and is otherwise inactive. This pin is weakly pulled high internally when inactive and/or when SC.7 (TAP) = 1. After power-up or a reset this pin defaults to JTAG TDO pin.
33	38	TMS	P0.1	JTAG Test Mode Select Input. This signal is sampled at the rising edge of TCK and controls movement between TAP states. TMS is weakly pulled high internally when TAP = 1. After power-up or a reset this pin defaults to JTAG TMS pin.
34	39	TDI	P0.2	JTAG Serial Test Data Input. This signal is used to receive data serially transferred by the host. Data is received least significant bit first and is sampled on the rising edge of TCK. TDI is weakly pulled high internally when TAP = 1. After power-up or a reset this pin defaults to JTAG TDI pin.
35	40	TCK	P0.3	JTAG Serial Test Clock Input. Provided by the host. When this signal is stopped at 0, storage elements in the TAP logic retain their data indefinitely. TCK is weakly pulled high internally when TAP = 1. After power-up or a reset this pin defaults to JTAG TCK pin.

MAXQ7665/MAXQ7666 User's Guide

10.3 TAP Interface Control

Once an application has been loaded and starts running, the MAXQ7665/MAXQ7666 JTAG TAP interface can be controlled by the TAP bit in the system control register as described in Section 10.3.1.

10.3.1 System Control Register (SC)

Register Description: **System Control Register**
Register Name: **SC**
Register Address: **Module 08h, Index 08h**

Bit #	7	6	5	4	3	2	1	0
Name	TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
Reset	1	0	0	0	0	0	1*	0
Access	rw	r	rw	rw	rw	rw	rw	r

r = read, w = write
*This register defaults to 80h on all forms of reset except after power-on reset. After power-on reset, the PWL bit is also set and this register defaults to 82h.

Bit 7: Test Access (JTAG) Port Enable (TAP). This bit controls whether the TAP special function pins are enabled. The TAP defaults to being enabled.

   0 = JTAG/TAP functions are disabled and P0.0–P0.3 can be used as general-purpose I/O pins
   1 = TAP special function pins P0.0–P0.3 are enabled to act as JTAG inputs and outputs

Bits 6 and 0: Reserved.

Bits 5 and 4: Code Data Access Bits 1 and 0 (CDA1:CDA0). See *Section 1* for more information on these bits.

Bit 3: Upper Program Access (UPA). See *Section 1* for more information on this bit.

Bit 2: ROM Operation Done (ROD). See *Section 11* for more information on this bit.

Bit 1: Password Lock (PWL). See *Section 12* for more information on this bit.

10.4 TAP Controller Operation

The MAXQ7665/MAXQ7666 TAP controller is formed by a finite state machine that provides 16 operational states for access control. The TAP state control is achieved through host manipulation of the TMS and TCK signals. The TMS signal is sampled at the rising edge of TCK and decoded by the TAP controller to control movement between the TAP states. The TDI input and TDO output are meaningful once the TAP is in a serial shift state. This section provides a brief description of the TAP controller state machine and its state transitions.

The state diagram in Figure 10-2 summarizes the transitions caused by the TMS signal sampling on the rising edge at TCK. The TMS signal value is shown adjacent to each state transition in the figure.
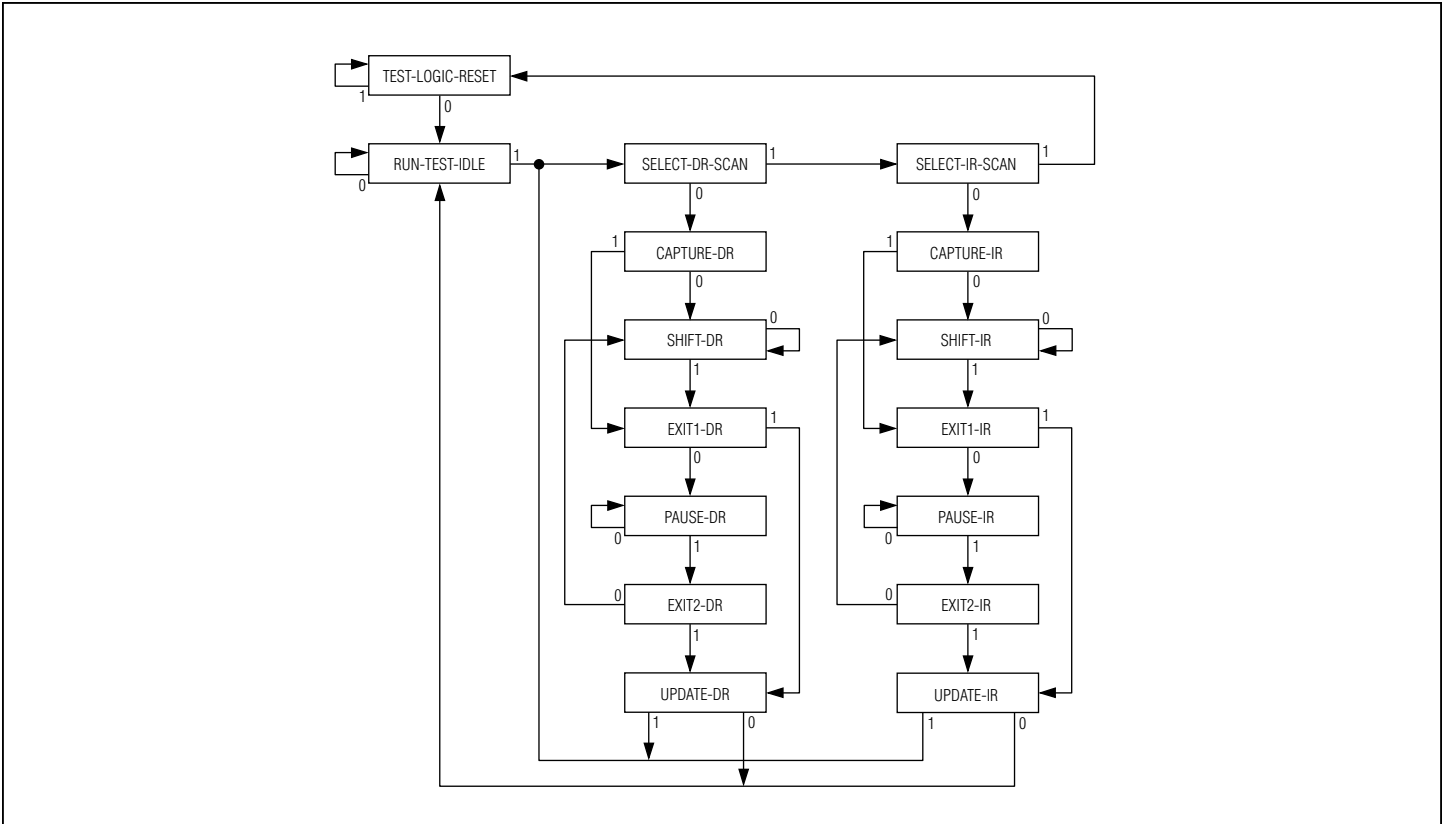
Figure 10-2. TAP Controller State Diagram

10.4.1 Test-Logic-Reset

On a power-on reset, the TAP controller is initialized to the test-logic-reset state and the instruction register (IR2:IR0) is initialized to the bypass instruction so that it does not affect normal system operation. No matter what the state of the controller, it enters test-logic-reset when TMS is held high for at least five rising edges of TCK. The controller remains in the test-logic-reset state if TMS remains high. An erroneous low signal on the TMS can cause the controller to move into the run-test-idle state, but no disturbance is caused to system operation if the TMS signal is returned and kept at the intended logic-high for three rising edges of TCK since this returns the controller to the test-logic-reset state.

10.4.2 Run-Test-Idle

As illustrated in Figure 10-2, the run-test-idle state is an intermediate state for getting to one of the two state sequences in which the TAP controller performs meaningful operations:

- Controller state sequence (IR-scan)
- Data register state sequence (DR-scan)

10.4.3 IR-Scan Sequence

The MAXQ7665/MAXQ7666 support a 3-bit TAP instruction register to allow certain device specific instructions (e.g., "Debug" or "System Programming") to be supported. The IR-Scan sequence allows instructions (e.g., "Debug" and "System Programming") to be shifted into the instruction register starting from the select-IR-scan state. In the TAP, the instruction register is connected between the TDI input and the TDO output. Inside the IR-scan sequence, the capture-IR state loads a fixed binary pattern (001b) into the 3-bit shift register and the shift-IR state causes shifting of TDI data into the shift register and serial output to TDO, least significant bit first. Once the desired instruction is in the shift register, the instruction can be latched into the parallel instruction register (IR2:IR0) on the falling edge of TCK in the update-IR state. The contents of the 3-bit instruction shift register and parallel instruction register (IR2:IR0) are summarized with respect to the TAP controller states in Table 10-2.

When the parallel instruction register (IR2:IR0) is updated, the TAP controller decodes the instruction and performs any necessary operations, including activation of the data shift register to be used for the particular instruction during data register shift sequences (DR-scan). The length of the activated shift register depends upon the value loaded to the instruction register (IR2:IR0). The supported instruction register encodings and associated data-register selections are shown in Table 10-3.

The extest (IR2:IR0 = 000b) and sample/preload (IR2:0 = 001b) instructions are mandated by the JTAG standard; however, the MAXQ7665/MAXQ7666 do not make use of these instructions. These instructions are treated as no operations and may be entered into the instruction register without affecting the on-chip system logic or pins and does not change the existing serial data register selection between TDI and TDO.

The bypass (IR2:IR0 = 011b, 101b, or 111b) instruction is also mandated by the JTAG standard. The bypass instruction is fully implemented by the MAXQ7665/MAXQ7666 to provide a minimum length serial data path between the TDI and the TDO pins. This is accomplished by providing a single-cell bypass shift register. When the instruction register is updated with the bypass instruction, a single bypass register bit is connected serially between TDI and TDO in the shift-DR state. The instruction register automatically defaults to the bypass instruction when the TAP is in the test-logic-reset state. The bypass instruction has no effect on the operation of the on-chip system logic.

The debug (IR2:IR0 = 010b) and system programming (IR2:IR0 = 100b) instructions are private instructions that are intended solely for in-circuit debug and in-system programming operations, respectively. If the instruction register is updated with the debug instruction, a 10-bit serial shift register is formed between the TDI and TDO pins in the shift-DR state. If the system programming instruction is entered into the instruction register (IR2:IR0), a 3-bit serial data shift register is formed between the TDI and TDO pins in the shift-DR state.

Instruction register (IR2:IR0) settings other than those listed and previously described are reserved for internal use. As can be seen in Figure 10-1, the instruction register serves to select the length of the serial data register between TDI and TDO during the shift-DR state.

Table 10-2. Instruction Register Content vs. TAP Controller State

TAP CONTROLLER STATE	INSTRUCTION SHIFT REGISTER	PARALLEL (3-BIT) INSTRUCTION REGISTER (IR2:IR0)
Test-Logic-Reset	Undefined	Set to bypass (011b) instruction
Capture-IR	Load 001b at the rising edge of TCK	Retain last state
Shift-IR	Input data via TDI and shift towards TDO at the rising edge of TCK	Retain last state
Exit1-IR, Exit2-IR, Pause-IR	Retain last state	Retain last state
Update-IR	Retain last state	Load from shift register at the falling edge of TCK
All other states	Undefined	Retain last state

Table 10-3. Instruction Register Commands

IR2:IR0			INSTRUCTION	FUNCTION	SERIAL DATA SHIFT REGISTER SELECTION
0	0	0	Extest	No operation	Unchanged (retain previous selection)
0	0	1	Sample/Preload	No operation	Unchanged (retain previous selection)
0	1	0	Debug	In-circuit debug mode	10-bit shift register
0	1	1	Bypass	No operation (default)	1-bit shift register
1	0	0	System Programming	Bootstrap function	3-bit shift register
1	0	1	Bypass	No operation (default)	1-bit shift register
1	1	0	Reserved		
1	1	1	Bypass	No operation (default)	1-bit shift register

10.4.4 DR-Scan Sequence

Once the instruction register has been configured to a desired state (mode), transactions are performed via a data buffer register associated with that mode. These data transactions are executed serially in a manner analogous to the process used to load the instruction register. The transactions are grouped in the TAP controller state sequence starting from the select-DR-scan state. In the TAP controller state sequence, the shift-DR state allows internal data to be shifted out through the TDO pin while the external data is shifted in simultaneously via the TDI pin. Once a complete data pattern is shifted in, input data can be latched into the parallel buffer of the selected register on the falling edge of TCK in the update-DR state. On the same TCK falling edge, in the update-DR state, the internal parallel buffer is loaded to the data shift register for output. This shift-DR/update-DR process serves as the basis for passing information between the external host and the MAXQ7665/MAXQ7666. These data register transactions occur in the data register portion of the TAP controller state sequence diagram and have no effect on the instruction register.

10.4.5 Communication via TAP

The TAP controller is in test-logic-reset state after a power-on-reset. During this initial state, the instruction register contains bypass instruction and the serial path defined between the TDI and TDO pins for the shift-DR state is the 1-bit bypass register. All TAP signals (TCK, TMS, TDI, and TDO) default to being weakly pulled high internally on any reset. The TAP controller remains in the test-logic-reset state as long as TMS is held high. The TCK and TMS signals can be manipulated by the host to transition to other TAP states. The TAP controller remains in a given state whenever TCK is held low.

For the host to establish a specific data communication link, a private instruction must be loaded into the IR2:IR0 register. Once the instruction is latched in the instruction parallel buffer at the update-IR state, it is recognized by the TAP controller and the communication channel is established. In-circuit debug or in-system programming commands and data can be exchanged between the host and the MAXQ7665/MAXQ7666 by operating in the data register portion of the state sequence (i.e., DR-scan). The TAP retains the private instruction that was loaded into IR2:IR0 until a new instruction is shifted in or until the TAP controller returns to the test-logic-reset state.

10.4.5.1 TAP Communication Examples IR-Scans and DR-Scans

Figures 10-3 and 10-4 illustrate examples of communication between the host JTAG controller and the TAP of the MAXQ7665/ MAXQ7666. The host controls the TCK and TMS signals to move through the desired TAP states while accessing the selected shift register through the TDI input and TDO output pair.
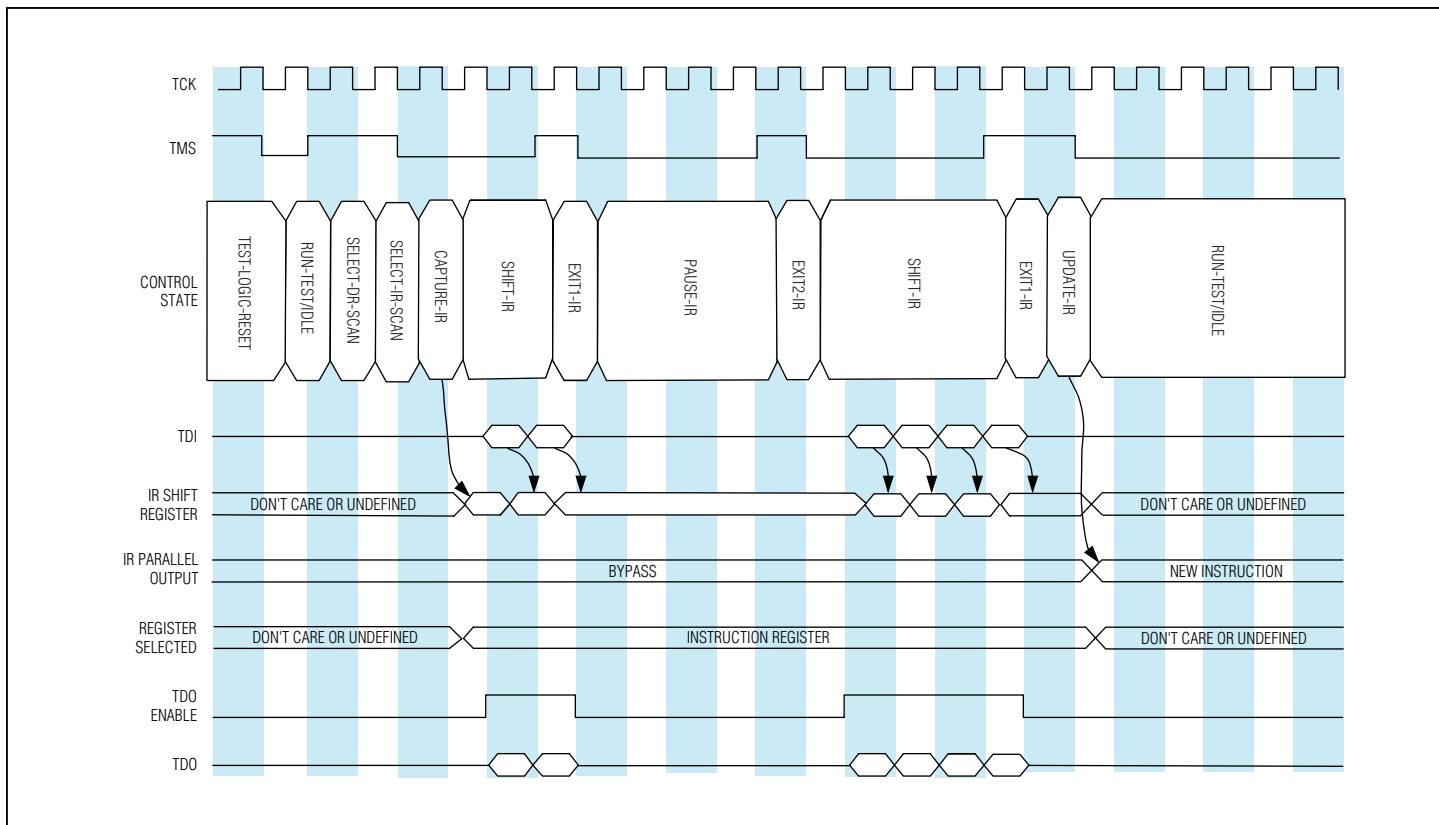
Figure 10-3. TAP Controller Debug Mode—IR-Scan Example
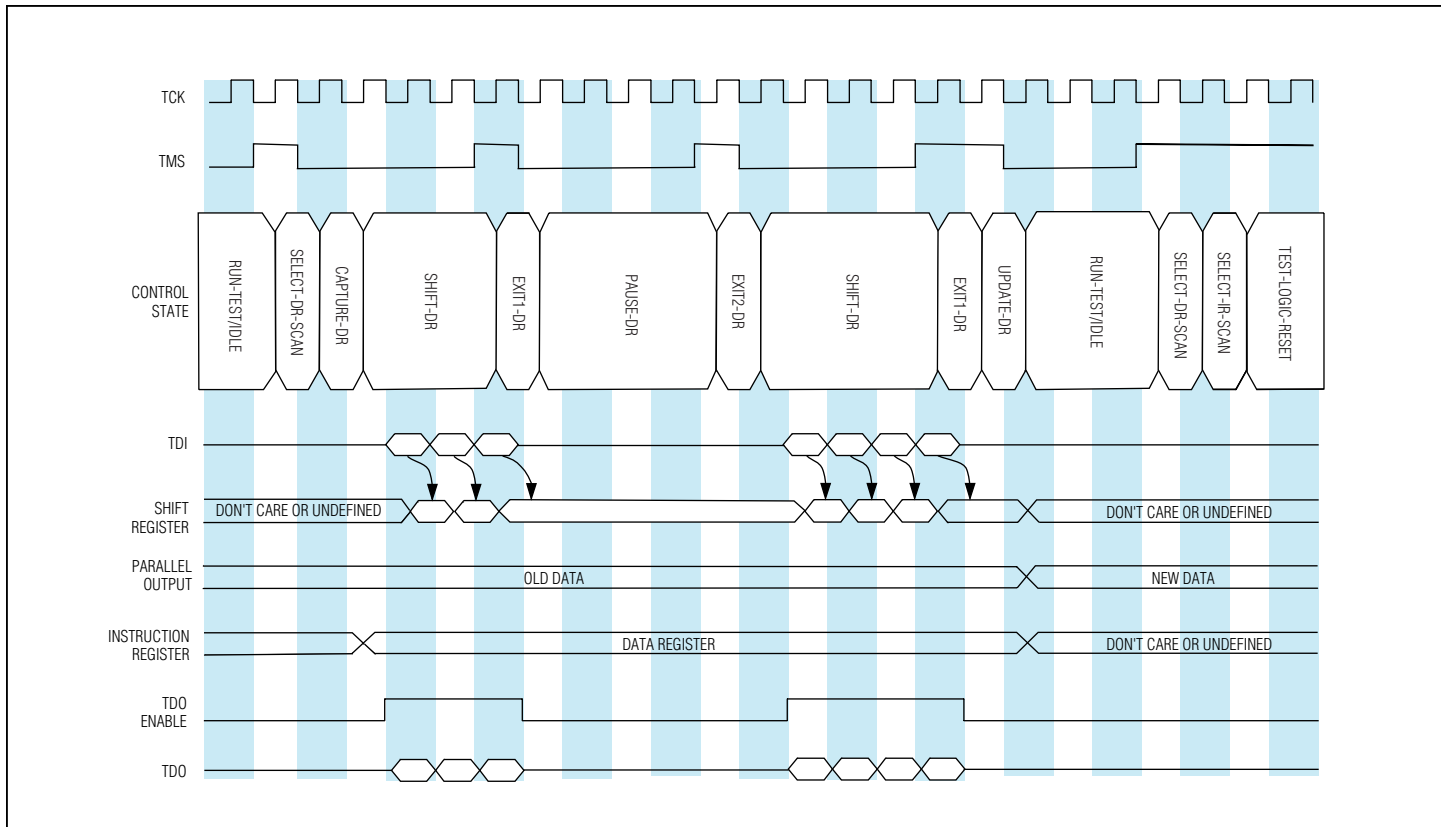
Figure 10-4. TAP Controller Debug Mode—DR-Scan Example

SECTION 11: IN-CIRCUIT DEBUG MODE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 11: IN-CIRCUIT DEBUG MODE

The MAXQ7665/MAXQ7666 are equipped with embedded debug hardware and embedded ROM firmware developed for the purpose of providing in-circuit debugging capability to the user application. The in-circuit debug mode uses the JTAG-compatible TAP as its means of communication between the host and MAXQ7665/MAXQ7666 microcontrollers. The in-circuit debug hardware and software features include the following:

- A debug engine
- A set of registers providing the ability to set breakpoints on register, code, or data
- A set of debug service routines stored in a ROM

Collectively, these hardware and software features allow two basic modes of in-circuit debugging:

- Background mode allows the host to configure and set up the in-circuit debugger while the CPU continues to execute the normal program. Debug mode can be invoked from background mode.
- Debug mode allows the debug engine to take control of the CPU, providing read-write access to internal registers and memory, and single-step trace operation.

11.1 Architecture

Figure 11-1 shows a simplified functional block diagram of the MAXQ7665/MAXQ7666 in-circuit debugger. The embedded hardware debug engine is implemented as a stand-alone hardware block in the MAXQ7665/MAXQ7666 microcontrollers. The debug engine can be enabled for monitoring internal activities and interacting with selected internal registers while the CPU is executing user code. This capability allows the user to employ the embedded debug engine to debug the actual system, in place of the in-circuit emulator that uses external hardware to duplicate operation of the microcontroller outside of the real application environment.
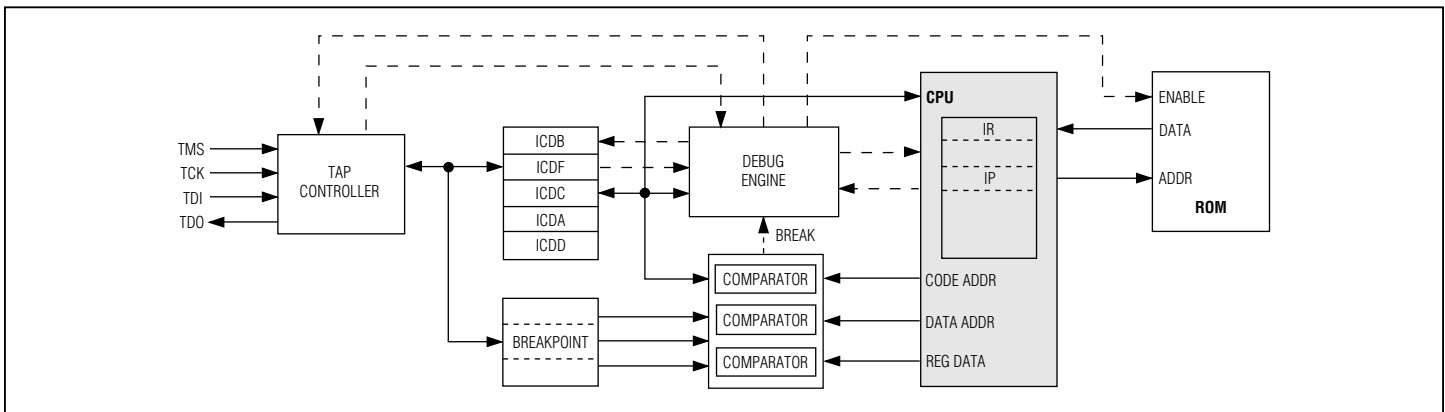
Figure 11-1. In-Circuit Debugger

The embedded debug engine is a state machine that takes commands from the host device and performs the necessary tasks to complete the debug function. While the TAP is running at the TCK clock frequency, the debug engine and all its associated hardware are clocked by the system clock. The debug engine is not operated in stop mode.

All debug engine activities are originated by the external host through 8-bit commands. The debug engine decodes the command in the ICDB register directly, and provides the following functions for use in debugging application software:

- Single-step (trace) execution
- Four program address breakpoints
- Two breakpoints configurable as data address or register address break points
- Register read and write
- Program stack read
- Data memory read and write
- Optional password protection

The debug engine is supported by five functional registers:

- **ICDB:** The ICDB register is an 8-bit data register that supports exchanging command/data between the host system and the in-circuit debugger. The register functions as an 8-bit parallel buffer for the debug shift register in the TAP. The ICDB register is mapped to the peripheral register space and is read/write accessible by the CPU and the debug engine.

- **ICDC:** The ICDC register is an 8-bit control register for the in-circuit debugger. All bits in this register are set/reset by the debug engine. It is mapped to the peripheral register space and is read only by the CPU.

- **ICDF:** The ICDF register is an 8-bit register and is used to provide system status to the host system, the debug engine, and the CPU during debug operation. This register is mapped to the peripheral register space and read/write accessible by the CPU and the debug engine.

- **ICDA:** The ICDA register is a 16-bit register that is primarily used to specify an address for ROM assisted operations. The ICDA is mapped to the peripheral register space and is read only by the CPU. It is read/write accessible by the debug engine. The ICDA may also be used as a bit mask for register access breakpoints (REGE = 1).

- **ICDD:** The ICDD register is a 16-bit register that is used to store data for ROM assisted operations. The ICDD is mapped to the peripheral register space and is read only by the CPU. It is read/write accessible by the debug engine. The ICDD may also be used as the bit compare match data for register access breakpoints (REGE = 1).

11.2 In-Circuit Debug Peripheral Registers

The MAXQ7665/MAXQ7666 in-circuit debug peripheral registers are described here. All the in-circuit debug peripheral registers are directly accessible by the microcontroller through the module/index address.

11.2.1 In-Circuit Debug Temporary 0 Register (ICDT0)

The ICDT0 register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routine as temporary storage to save registers that might otherwise have to be placed in the stack. This register is cleared after a power-on reset or by a test-logic-reset TAP state.

Register Description: **In-Circuit Debug Temporary 0 Register**
Register Name: **ICDT0**
Register Address: **Module 02h, Index 18h**

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	ICDT0.15	ICDT0.14	ICDT0.13	ICDT0.12	ICDT0.11	ICDT0.10	ICDT0.9	ICDT0.8
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	ICDT0.7	ICDT0.6	ICDT0.5	ICDT0.4	ICDT0.3	ICDT0.2	ICDT0.1	ICDT0.0
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

s = special (read/write access only in background or debug mode)

Bits 15 to 0: In-Circuit Debug Temporary 0 Register Bits 15 to 0 (ICDT0.15 to ICDT0.0)

11.2.2 In-Circuit Debug Temporary 1 Register (ICDT1)

The ICDT1 register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack. This register is cleared after a power-on reset or by a test-logic-reset TAP state.

Register Description: **In-Circuit Debug Temporary 1 Register**
Register Name: **ICDT1**
Register Address: **Module 02h, Index 19h**

Bit #	15	14	13	12	11	10	9	8
Name	ICDT1.15	ICDT1.14	ICDT1.13	ICDT1.12	ICDT1.11	ICDT1.10	ICDT1.9	ICDT1.8
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	ICDT1.7	ICDT1.6	ICDT1.5	ICDT1.4	ICDT1.3	ICDT1.2	ICDT1.1	ICDT1.0
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

s = special (read/write access only in background or debug mode)

Bits 15 to 0: In-Circuit Debug Temporary 1 Register Bits 15 to 0 (ICDT1.15 to ICDT1.0)

11.2.3 In-Circuit Debug Control Register (ICDC)

The ICDC register is read/write accessible by the debug engine and is read only by the CPU. This register is cleared after a power-on reset or by a test-logic-reset TAP state.

Register Description: **In-Circuit Debug Control Register**
Register Name: **ICDC**
Register Address: **Module 02h, Index 1Ah**

Bit #	7	6	5	4	3	2	1	0
Name	DME	—	REGE	—	CMD.3	CMD.2	CMD.1	CMD.0
Reset	0	0	0	0	0	0	0	0
Access	rs	r	rs	r	rs	rs	rs	rs

r = read, s = special (write only by debug engine)

Bit 7: Debug Mode Enable (DME). When this bit is cleared to 0, background mode commands can be executed but breakpoints are disabled. When this bit is set to 1, breakpoints are enabled while background mode commands can still be entered. This bit can only be set or cleared from background debug mode. This bit has no meaning for the ROM code.

Bits 6 and 4: Reserved. Read 0, write ignored.

Bit 5: Break-On Register Enable (REGE). The REGE bit is used to enable the break-on register function. When the REGE bit is set to 1, BP4 and BP5 are used as register breakpoints. A break occurs when the content of BP4 is matched with the destination address of the current instruction. For BP5, a break occurs only on a selected data pattern for a selected destination register addressed by BP5. The data pattern is determined by the contents in the ICDA and ICDD register. The REGE bit alone does not enable register breakpoints, but simply changes the manner in which BP4 and BP5 are used. The DME bit still must be set to logic 1 for any breakpoint to occur. This bit has no meaning for the ROM code.

Bits 3 to 0: Command Bits 3 to 0 (CMD.3 to CMD.0). These bits reflect the current host command in debug mode. These bits are set by the debug engine and allow the ROM code to determine the course of action.

CMD.3	CMD.2	CMD.1	CMD.0	ACTION
0	0	0	0	No Operation
0	0	0	1	Read Register Map
0	0	1	0	Read Data Memory
0	0	1	1	Read Stack Memory
0	1	0	0	Write Register
0	1	0	1	Write Data Memory
0	1	1	0	Trace, Single-Step the CPU
1	0	0	0	Unlock Password
1	0	0	1	Read Register
X	X	X	X	Reserved

11.2.4 In-Circuit Debug Flag Register (ICDF)

Register Description: **In-Circuit Debug Flag Register**
Register Name: **ICDF**
Register Address: **Module 02h, Index 1Bh**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	PSS1	PSS0	SPE	TXC
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

r = read, w = write

Bits 7 to 4: Reserved. Read 0, write ignored.

Bits 3 and 2: Programming Source Select Bits 1 and 0 (PSS1 and PSS0). See *Section 12* for information on these bits.

Bit 1: System Program Enable (SPE). See *Section 12* for information on this bit.

Bit 0: Serial Transfer Enable (TXC). This bit is set by hardware at the end of a transfer cycle at the TAP communication link. The TXC bit helps the debug engine to recognize host requests, either command or data. This bit is normally set by ROM code to signify or request the sending or receiving of data. Once set, the debug engine clears the TXC bit. CPU writes to the TXC bit result in the clearing of the JTAG PSS1 and PSS0 bits.

11.2.5 In-Circuit Debug Buffer Register (ICDB)

The ICDB register serves as the parallel holding buffer for the debug shift register of the TAP. Data is read from or written to ICDB for serial communication between the debug routines and the external host. This register is cleared to 00h after a power-on reset or a test-logic-reset TAP state.

Register Description: **In-Circuit Debug Buffer Register**
Register Name: **ICDB**
Register Address: **Module 02, Index 1Ch**

Bit #	7	6	5	4	3	2	1	0
Name	ICDB.7	ICDB.6	ICDB.5	ICDB.4	ICDB.3	ICDB.2	ICDB.1	ICDB.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: In-Circuit Debug Buffer Register Bits 7 to 0 (ICDB.7 to ICDB.0)

11.2.6 In-Circuit Debug Address Register (ICDA)

The debug engine uses the ICDA register to store addresses so that ROM code may view that information. This register is also used by the debug engine as a mask register to mask out don't care bits in the ICDD register when BP5 is used as a register breakpoint. When a bit in this register is set to 1, the corresponding bit location in the ICDD register is compared to the data being written to the destination register to determine if a break should be generated. When a bit in this register is cleared, the corresponding bit in the ICDD register are don't cares and are not compared against the data being written. When all bits in this register are cleared, any updated data pattern causes a break when the BP5 register matches the destination register address of the current instruction. This register is cleared to 0000h after a power-on reset or a test-logic-reset TAP state.

Register Description: **In-Circuit Debug Address Register**
Register Name: **ICDA**
Register Address: **Module 02h, Index 1Dh**

Bit #	15	14	13	12	11	10	9	8
Name	ICDA.15	ICDA.14	ICDA.13	ICDA.12	ICDA.11	ICDA.10	ICDA.9	ICDA.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ICDA.7	ICDA.6	ICDA.5	ICDA.4	ICDA.3	ICDA.2	ICDA.1	ICDA.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: In-Circuit Debug Address Register Bits 15 to 0 (ICDA.15 to ICDA.0)

11.2.7 In-Circuit Debug Data Register (ICDD)

The debug engines uses the ICDD register to store data/read count so that ROM code can view that information. The debug engine also uses this register as a data register for content matching when BP5 is used as a register breakpoint. In this case, only data bits in this register with their corresponding mask bits in the ICDA register set are compared with the updated destination data to determine if a break should be generated. This register is cleared to 0000h after a power-on reset and or a test-logic-reset sequence TAP state.

Register Description: **In-Circuit Debug Data Register**
Register Name: **ICDD**
Register Address: **Module 02h, Index 1Eh**

Bit #	15	14	13	12	11	10	9	8
Name	ICDD.15	ICDD.14	ICDD.13	ICDD.12	ICDD.11	ICDD.10	ICDD.9	ICDD.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ICDD.7	ICDD.6	ICDD.5	ICDD.4	ICDD.3	ICDD.2	ICDD.1	ICDD.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: In-Circuit Debug Data Register Bits 15 to 0 (ICDD.15 to ICDD.0)

MAXQ7665/MAXQ7666 User's Guide

11.2.8 System Control Register (SC)

Register Description: **System Control Register**
Register Name: **SC**
Register Address: **Module 08h, Index 08h**

Bit #	7	6	5	4	3	2	1	0
Name	TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
Reset	1	0	0	0	0	0	1*	0
Access	rw	r	rw	rw	rw	rw	rw	r

r = read, w = write
*This register defaults to 80h on all forms of reset except after power-on reset. After power-on reset, the PWL bit is also set and this register defaults to 82h.

Bit 7: Test Access (JTAG) Port Enable (TAP). This bit controls whether the TAP special function pins are enabled. The TAP defaults to being enabled. See *Section 10* for more information on this bit.

   0 = JTAG/TAP functions are disabled and P0.0–P0.3 can be used as general-purpose I/O pins
   1 = TAP special function pins P0.0–P0.3 are enabled to act as JTAG inputs and outputs

Bits 6 and 0: Reserved

Bits 5 and 4: Code Data Access Bits 1 and 0 (CDA1:CDA0). See *Section 1* for more information on these bits.

Bit 3: Upper Program Access (UPA). See *Section 1* for more information on this bit.

Bit 2: ROM Operation Done (ROD). This bit is used to signify completion of a ROM operation sequence to the control units. This allows the debug engine to determine the status of a ROM sequence. Setting this bit to logic 1 causes an internal system reset if the SPE bit is also set. Setting the ROD bit will clear the SPE bit if it is set and the ROD bit will be automatically cleared by hardware once the control unit acknowledges the done indication.

Bit 1: Password Lock (PWL). This bit defaults to 1 on a power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the program space before allowing access to the password protected in-circuit debug or bootstrap loader ROM routines. Clearing this bit to 0 disables the password protection for these ROM routines. See *Section 12* for more information on this bit.

11.3 Debug Engine Operation

To enable a communication link between the host and the MAXQ7665/MAXQ7666 debug engine, the debug instruction (010b) must be loaded into the TAP instruction register using the IR-scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:IR0) and is recognized by the TAP controller in the update-IR state, the 10-bit data shift register is activated as the communication channel for DR-scan sequences. The TAP instruction register retains the debug instruction until a new instruction is shifted via an IR-scan or the TAP controller returns to the test-logic-reset state.

The host now can transmit and receive serial data through the 10-bit data shift register that exists between the TDI input and TDO output during DR-scan sequences. All background and debug mode communication (commands, data input/output, and status) occurs via this serial channel. Each 10-bit exchange of data between the host and the MAXQ7665/MAXQ7666 internal hardware is composed of two status bits and a single byte of command or data. The 10-bit word is always transmitted least significant bit first according to the following format.

The data byte portion of the 10-bit shift register is interfaced directly to the ICDB parallel register. The ICDB register functions as the holding data register for both transmit and receive operations. On the falling edge of TCK in the update-DR state, the outgoing data is loaded from the ICDB parallel register to the debug shift register, and the incoming shift register data is latched in the ICDB parallel register.

s1:s0	STATUS/CONDITION
00	**NONDEBUG.** DEFAULT CONDITION, BACKGROUND MODE, OR DEBUG ENGINE INACTIVE.
01	**DEBUG IDLE.** DEBUG ENGINE IS READY TO RECEIVE DATA FROM THE HOST (COMMAND, DATA).
10	**DEBUG BUSY.** DEBUG ENGINE IS BUSY WITHOUT VALID DATA (i.e., ROM CODE EXECUTION, TRACE OPERATIONS).
11	**DEBUG VALID.** DEBUG ENGINE IS BUSY WITH VALID DATA.
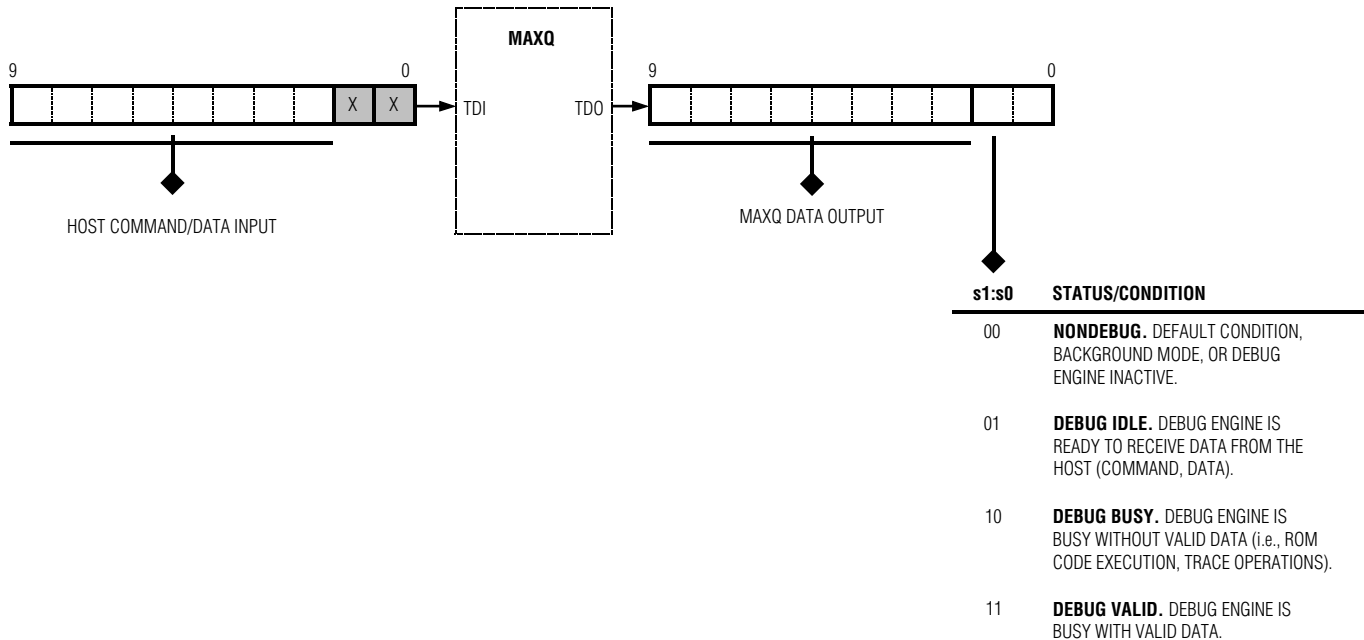
11.3.1 Background Mode Operation

When the instruction register is loaded with the debug instruction (IR2:IR0 = 010b), the host can communicate with the MAXQ7665/MAXQ7666 in a background mode using TAP DR-scan sequences without disturbing CPU operation. Note, however, that JTAG in-system programming also requires use of the 10-bit debug shift register and, if enabled (SPE, PSS1:PSS0 = 100b), takes precedence over background mode communication. When operating in background mode, the status bits are always cleared to 00b (nondebug), which indicates that the MAXQ7665/MAXQ7666 are ready to receive background mode commands. The host can perform the following operations from background mode:

- Read/write internal breakpoint registers (BP0–BP5).
- Read/write internal in-circuit debug registers (ICDC, ICDF, ICDA, ICDD).
- Monitor to determine when a breakpoint match has occurred.
- Directly invoke debug mode.

MAXQ7665/MAXQ7666 User's Guide

Table 11-1 shows the background mode commands supported by the MAXQ7665/MAXQ7666. Encodings not listed in this table are not supported in background mode and are treated as no operations.

A command can consist of multiple-byte transactions between the external host and the debug engine via the TAP. However, a command code is always 8 bits and is always transmitted first, followed by address and/or data when needed.

Table 11-1. Background Mode Commands

OP CODE	COMMAND	OPERATION
0000-0000	No Operation	**No Operation.** Default state for Debug Shift register.
0000-0001	Read ICDC	**Read Control Data from the ICDC.** The contents of the ICDC register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0010	Read ICDF	**Read Flags from the ICDF.** The contents of the ICDF register (one byte) are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0011	Read ICDA	**Read Data from the ICDA.** The contents of the ICDA register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0100	Read ICDD	**Read Data from the ICDD.** The contents of the ICDD register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0101	Read BP0	**Read Data from the BP0.** The contents of the BP0 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0110	Read BP1	**Read Data from the BP1.** The contents of the BP1 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0111	Read BP2	**Read Data from the BP2.** The contents of the BP2 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1000	Read BP3	**Read Data from the BP3.** The contents of the BP3 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1001	Read BP4	**Read Data from the BP4.** The contents of the BP4 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1010	Read BP5	**Read Data from the BP5.** The contents of the BP5 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0001-0001	Write ICDC	**Write Control Data to the ICDC.** The contents of ICDB are loaded into the ICDC register by the debug engine at the end of the data transfer cycle.
0001-0011	Write ICDA	**Write Data to the ICDA.** The contents of ICDB are loaded into the ICDA register by the debug engine at the end of the data transfer cycles. Data is transferred with the least significant byte first.
0001-0100	Write ICDD	**Write Data to the ICDD.** The contents of ICDB are loaded into the ICDD register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0101	Write BP0	**Write Data to the BP0.** The contents of ICDB are loaded into the BP0 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0110	Write BP1	**Write Data to the BP1.** The contents of ICDB are loaded into the BP1 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0111	Write BP2	**Write Data to the BP2.** The contents of ICDB are loaded into the BP2 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1000	Write BP3	**Write Data to the BP3.** The contents of ICDB are loaded into the BP3 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1001	Write BP4	**Write Data to the BP4.** The contents of ICDB are loaded into the BP4 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1010	Write BP5	**Write Data to the BP5.** The contents of ICDB are loaded into the BP5 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1111	Debug	**Debug Command.** This command forces the debug engine into debug mode and halts the CPU operation at the completion of the current instruction after the debug engine recognizes the debug command.

11.3.2 Breakpoint Registers

The MAXQ7665/MAXQ7666 incorporate six host-configurable breakpoint registers (BP0–BP5) for establishing different types of break-point mechanisms. The first four breakpoint registers (BP0–BP3) are 16-bit registers that are configurable as program memory address breakpoints. When enabled, the debug engine forces a break when a match between the breakpoint register and the program memory execution address occurs. The final two 16-bit breakpoint registers (BP4 and BP5) are configurable in one of two ways. They can be configured as data memory address breakpoints or can be configured to support register-access breakpoints. In either case, if breakpoints are enabled and the defined breakpoint match occurs, the debug engine generates a break condition. The six breakpoint registers are detailed in the following sections.

11.3.2.1 Breakpoint Registers 0 to 3 (BP0 to BP3)

The BP0 to BP3 registers are accessible only via background mode read/write debug commands. These four registers serve as program memory address breakpoints. When the DME bit is set, the debug engine monitors the program address bus activity while the CPU is executing the user program. A break occurs when the address pattern matches with the contents of these registers, allowing the debug engine to take control of the CPU and enter debug mode.

Register Description: **Breakpoint Register x (where x = 0, 1, 2, 3)**
Register Name: **BPx**

Bit #	15	14	13	12	11	10	9	8
Name	BPx.15	BPx.14	BPx.13	BPx.12	BPx.11	BPx.10	BPx.9	BPx.8
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	BPx.7	BPx.6	BPx.5	BPx.4	BPx.3	BPx.2	BPx.1	BPx.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

s = special (accessible only by background mode read/write commands)

Bits 15 to 0: Breakpoint Register x Bits 15 to 0 (BPx.15 to BPx.0). These registers default to FFFFh after a power-on reset or test-logic-reset TAP state.

11.3.2.2 Breakpoint Register 4 (BP4)

Register Description: **Breakpoint Register 4**
Register Name: **BP4**

This register is accessible only via background mode read/write commands.

When (REGE = 0): This register serves as one of the two data memory address breakpoints. When DME is set in background mode, the debug engine monitors the data memory address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take over control of the CPU and enter debug mode.

When (REGE = 1): This register serves as one of the two register breakpoints. A break occurs when the destination register address for the executed instruction matches with the specified module and index. When used as register breakpoint, the bits BP4.3:BP4.0 are recognized as module specifier and bits BP4.8:BP4.4 are recognized as the register index within the module. The bits BP4.15:BP4.9 are ignored.

This register defaults to FFFFh after a power-on reset or test-logic-reset TAP state.

Bit #	15	14	13	12	11	10	9	8
Name	BP4.15	BP4.14	BP4.13	BP4.12	BP4.11	BP4.10	BP4.9	BP4.8
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	BP4.7	BP4.6	BP4. 5	BP4.4	BP4.3	BP4.2	BP4.1	BP4.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

s = special (accessible only by background mode read/write commands)

Bits 15 to 0: Breakpoint Register 4 Bits 15 to 0 (BP4.15 to BP4.0)

11.3.2.3 Breakpoint Register 5 (BP5)

This register is accessible only through background mode read/write commands.

When (REGE = 0): This register serves as one of the two data memory address breakpoints. When DME is set in background mode, the debug engine monitors the data memory address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take control of the CPU and enter debug mode.

When (REGE = 1): This register serves as one of the two register breakpoints. A break occurs when the following two conditions are met:

- Condition 1: The destination register address for the executed instruction matches with the specified module and index. When used as register breakpoint, the bits BP5.3:BP5.0 are recognized as module specifier and bits BP5.8:BP5.4 are recognized as the register index within the module. Bits BP5.15:BP5.9 are ignored.

- Condition 2: The bit pattern written to the destination register matches those bits specified for comparison by the ICDD data register and ICDA mask register. Only those ICDD data bits with their corresponding ICDA mask bits will be compared. When all bits in the ICDA register are cleared, Condition 2 becomes a don't care.

This register defaults to FFFFh after a power-on reset or test-logic-reset TAP state.

Register Description: **Breakpoint Register 5**
Register Name: **BP5**

Bit #	15	14	13	12	11	10	9	8
Name	BP5.15	BP5.14	BP5.13	BP5.12	BP5.11	BP5.10	BP5.9	BP5.8
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	BP5.7	BP5.6	BP5.5	BP5.4	BP5.3	BP5.2	BP5.1	BP5.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s

s = special (accessible only by background mode read/write commands)

Bits 15 to 0: Breakpoint Register 5 Bits 15 to 0 (BP5.15 to BP5.0)

11.3.3 Using Breakpoints

All breakpoint registers (BP0–BP5) default to the FFFFh state on power-on reset or when the test-logic-reset TAP state is entered. The breakpoint registers are accessible only with background mode read/write commands issued over the TAP communication link. The breakpoint registers are not read/write accessible to the CPU.

Setting the debug-mode enable (DME) bit in the ICDC register to logic 1 enables all six breakpoint registers for breakpoint match comparison. The state of the break-on register enable (REGE) bit in the ICDC register determines whether the BP4 and BP5 breakpoints should be used as data memory address breakpoints (REGE = 0) or as register breakpoints (REGE = 1).

When using the register matching breakpoints, it is important to realize that debug mode operations (e.g., read data memory, write data memory, etc.) require the use of ICDA and ICDD for passing information between the host and MAXQ7665/MAXQ7666 ROM routines. It is advised that these registers be saved and restored, or be reconfigured before returning to the background mode if register breakpoints are to remain enabled.

When a breakpoint match occurs, the debug engine forces a break and the MAXQ7665/MAXQ7666 enter debug mode. If a breakpoint match occurs on an instruction that activates the PFX register, the break is held off until the prefixed operation completes. The host can assess whether debug mode has been entered by monitoring the status bits of the 10-bit word shifted out of the TDO pin. The status bits change from the nondebug (00b) state associated with background mode to the debug-idle (01b) state when debug mode is entered. Debug mode can also be manually invoked by host issuance of the debug background command.

11.3.4 Debug Mode

There are two ways to enter debug mode from background mode: 1) issuance of the debug command directly by the host through the TAP communication port, or 2) the breakpoint matching mechanism.

The host can issue the debug background command to the debug engine. This direct debug mode entry is unstable. The response time varies dependent on system conditions when the command is issued. The breakpoint mechanism provides a more controllable response, but requires that the breakpoints be initially configured in background mode. No matter the method of entry, the debug engine takes control of the CPU in the same manner. Debug mode entry is similar to the state machine flow of an interrupt except that the target execution address is 8010h, which resides in the utility ROM instead of the address specified by the IV register that is used for interrupts. On debug mode entry, the following actions occur:

1) Block the next instruction fetch from program memory.

2) Push the return address onto the stack.

3) Set the contents of IP to 8010h.

4) Clear the IGE bit to 0 to disable interrupt handler if it is not already clear.

5) Halt CPU operation.

Once in debug mode, further breakpoint matches or host issuance of the debug command are treated as no operations and do not disturb debug engine operation. Entering debug mode also stops the clocks to all timers, including the watchdog timer. Temporarily disabling these functions allows debug mode operations without disrupting the relationship between the original user program code and hardware-timed functions. No interrupt request can be granted because the interrupt handler is also halted as a result of IGE = 0.

11.3.5 Debug Mode Commands

The debug engine sets the data shift-register status bits to 01b (debug-idle) to indicate that it is ready to accept debug commands from the host. The host can perform the following operations from debug mode:

- Read register map
- Read program stack
- Read/write register
- Read/write data memory
- Single step of CPU (trace)
- Return to background mode
- Unlock password

The only operations directly controlled by the debug engine are single step and return. All other operations are assisted by debug service routines contained in the utility ROM. These operations require that multiple bytes be transmitted and/or received by the host; however, each operation always begins with host transmission of a command byte. The debug engine decodes the command byte to determine the quantity, sequence, and destination for follow-on bytes received from the host. Even though there is no timing window specified for receiving the complete command and follow-on data, the debug engine must receive the correct number of bytes for a particular command before executing that command. If command and follow-on data are transmitted out of byte order or proper sequence, the only way to resolve this situation is to disable the debug engine by changing the instruction register (IR2:IR0) and reloading the debug instruction. Once the debug engine has received the proper number of command and follow-on bytes for a given ROM assisted operation, it responds with the following actions:

- Update the command bits (CMD3:CMD0) in the ICDC register to reflect the host request.
- Enable the ROM if it is not been enabled.
- Force a jump to ROM address 8010h.
- Set the data shift register status bits to 10b (debug-busy).

The ROM code performs a read to the ICDC register CMD3:CMD0 bits to determine its course of action. The ROM can process some commands without receiving data from the host beyond the initially supplied follow-on bytes, while others (e.g., unlock password) require additional data from the host. Some commands need only to provide an indication of completion to the host, while others (read register map) need to supply multiple bytes of output data. To accomplish data flow control between the host and ROM, the status bits should be used by the host to assess when the ROM is ready for additional data and/or when the ROM is providing valid data output.

Internally, the ROM can ascertain when new data is available or when it can output the next data byte via the TXC flag. The TXC flag is an important indicator between the debug engine and the utility ROM debug routines. The utility ROM firmware sets the TXC flag to 1 to indicate that valid data has been loaded to the ICDB register. The debug engine clears the TXC flag to 0 to indicate completion of a data shift cycle, thus allowing the ROM to continue execution of a requested task that is still in progress. The utility ROM signals that it has completed a requested task by setting the ROM operation done (ROD) bit of the SC register to logic 1. The ROD bit is reset by the debug engine when it recognizes the done condition. Table 11-2 shows the debug mode commands supported by the MAXQ7665/MAXQ7666. Note that background mode commands are supported inside debug mode. Encodings not listed in this table are not supported in debug mode and are treated as no operations.

Table 11-2. Background Mode Debug Commands

OP CODE	COMMAND	OPERATION
0010-0000	No Operation	**No Operation**
0010-0001	Read Register Map	**Read Data from Internal Registers.** This command forces the debug engine to update the CMD3:0 bits in the ICDC to 0001b and perform a jump to ROM code at 8010h. The ROM debug service routine will load register data to ICDB for host capture/read, starting at the lowest register location in module 0, one byte at a time in a successive order until all internal registers are read and output to the host.
0010-0010	Read Data Memory	**Read Data from Data Memory.** This command requires four follow-on transfer cycles, two for the starting address and two for the word read count, starting with the LSB address and ending with the MSB read count. The address is moved to the ICDA register and the word read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:CMD0 bits to 0010b and performs a jump to ROM code at 8010h. The ROM debug service routine will load ICDB from data memory according to address and count information provided by the host.
0010-0011	Read Program Stack	**Read Data from Program Stack.** This command requires four follow-on transfer cycles, two for the starting address and two for the read count, starting with the LSB address and ending with the MSB read count. The address is moved to the ICDA register and the read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:CMD0 bits to 0011b and performs a jump to ROM code at 8010h. The ROM Debug service routine will pop data out from the stack according to the information received in the ICDA and ICDD register. The stack pointer is pre-decremented for each pop operation.
0010-0100	Write Register	**Write Data to a Selected Register.** This command requires four follow-on transfer cycles, two for the register address and two for the data, starting with the LSB address and ending with the MSB data. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:CMD0 bits to 0100b and performs a jump to ROM code at 8010h. The ROM Debug service routine will update the select register according to the information received in the ICDA and ICDD registers.
0010-0101	Write Data Memory	**Write Data to a Selected Data Memory Location.** This command requires four follow-on transfer cycles, two for the memory address and two for the data, starting with the LSB address and ending with the MSB data. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:CMD0 bits to 0101b and performs a jump to ROM code at 8010h. The ROM Debug service routine will update the selected data memory location according to the information received in the ICDA and ICDD registers.
0010-0110	Trace	**Trace Command.** This command allows single stepping the CPU and requires no follow-on transfer cycle. The trace operation is a 'debug mode exit, one cycle CPU execution, debug mode entry' sequence.
0010-0111	Return	**Return Command.** This command terminates the debug mode and returns the debug engine to background mode. This allows the CPU to resume its normal operation at the point where it has been last interrupted.
0010-1000	Unlock Password	**Unlock the Password Lock.** This command requires 32 follow-on transfer cycles each containing a byte value to be compared with the program memory password for the purpose of clearing the PWL bit and granting access to protected debug and loader functions. When this command is received, the debug engine updates the CMD3:CMD0 bits to 1000b and performs a jump to ROM code at 8010h. Data is loaded to the ICDB register when each byte of data is received, beginning with the LSB of the least significant word first and end with the MSB of the most significant word.
0010-1001	Read Register	**Read from a Selected Internal Register.** This command requires two follow-on transfer cycles, starting with the LSB address and ending with the MSB address. The address is moved to ICDA register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:CMD0 bits to 1001b and performs a jump to ROM code at 8010h. The ROM Debug service routine will always assume a 16-bit register length and return the requested data LSB first.

11.3.6 Read-Register Map Command Host-ROM Instruction

A read-register map command reads out data contents for all implemented system and peripheral registers. The host does not specify a target register but instead should expect register data output in successive order, starting with the lowest order register in register module 0. Data is loaded by the ROM to the 8-bit ICDB register and is output one byte per transfer cycle. Thus, for a 16-bit register, two transfer cycles are necessary. The host initiates each transfer cycle to shift out the data bytes and will find valid data output tagged with a debug-valid (status = 11b). At the end of each transfer cycle, the debug engine clears the TXC flag to signal the ROM service routine that another byte can be loaded to ICDB. The ROM service routine sets the TXC flag each time after loading data to the ICDB register. This process is repeated until all registers have been read and output to the host. The host system recognizes the completion of the register read when the status debug-idle is presented. This indicates that the debug engine is ready for another operation.

11.3.7 Single-Step (Trace) Operation

The debug engine supports single-step operation in debug mode by executing a trace command from the host. The debug engine allows the CPU to return to its normal program execution for one cycle and then forces a debug mode re-entry:

1) Set status to 10b (debug-busy).
2) Pop the return address from the stack.
3) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
4) Supply the CPU with an instruction addressed by the return address.
5) Stall the CPU at the end of the instruction execution.
6) Block the next instruction fetch from program memory.
7) Push the return address onto the stack.
8) Set the contents of IP to 8010h.
9) Clear the IGE bit to 0 to disable the interrupt handler.
10) Halt CPU operation.
11) Set the status to debug-idle.

Note that the trace operation uses a return address from the stack as a legitimate address for program fetching. The host must maintain consistency of program flow during the debug process. The instruction pointer is automatically incremented after each trace operation, thus a new return address is pushed onto the stack before returning the control to the debug engine. Also, note that the interrupt handler is an essential part of the CPU and a pending interrupt could be granted during single-step operation since the IGE bit state present on debug mode entry is restored for the single step.

11.3.8 Return

To terminate the debug mode and return the debug engine to background mode, the host must issue a return command to the debug engine. This command causes the following actions:

1) Pop the return address from the stack.
2) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
3) Supply the CPU with an instruction addressed by the return address.
4) Allow the CPU to execute the normal user program.
5) Set the status to 00b (nondebug).

To prevent a possible endless-breakpoint matching loop, no break occurs for a breakpoint match on the first instruction after returning from debug mode to background mode. Returning to background mode also enables all internal timer functions.

11.3.9 Debug Mode Special Considerations

The following are special considerations when using debug mode.

The debug engine cannot be operated reliably when the CPU is configured in the power management mode (divide-by-256 system clock mode). To allow for proper execution of debug mode commands when invoked during PMM, the switchback enable (SWB) bit should be configured to logic 1. With SWB = 1, entering active debug mode (whether by breakpoint match or issuance of the debug command) forces a switchback to the divide-by-1 system clock mode and allows the debug engine to function correctly. This allows user code to configure breakpoints that occur inside PMM, thus providing reliable use of debug commands. However, it does not allow a good means for re-entering PMM.

- Special caution should be exercised when using the write-register command on register bits that globally affect system operation (e.g., IGE, STOP). If the write-register command is used to invoke STOP mode (setting STOP = 1), the RST pin can be asserted to reset the debug engine and return to the background mode of operation.

- Single stepping (trace) through any IGE bit change operation results in the debug engine overriding the bit change since it retains the IGE bit setting captured when active debug mode was entered.

- Single stepping (trace) into an operation that sets STOP = 1 when IGE = 1 effectively allows enabled interrupts normally capable of causing exit from STOP mode to do so.

- Single stepping (trace) through any memory read instruction that reads from the utility ROM (such as "move Acc," @DP[0] with DP[0] set to 8000h) causes the memory read to return an incorrect value.

- Single stepping (trace) cannot be used when executing code from the utility ROM.

- Data memory allocation is important during system development if in-circuit debug is planned. The top 32-byte memory location can be used by the debug service routine during debug mode. The data contents in these locations can be altered and cannot be recovered.

- One available stack location is needed for debug mode. If the stack is full when entering debug mode, the oldest data in the stack will be overwritten.

- The crystal warmup counter is the only counter not disabled when active debug mode is entered. If the crystal warmup counter completes while in active debug mode, a glitchless switch will be made to selected clock source, which was being counted. It is important the user recognize that this action will occur as the TAP clock should be run no faster than 1/8th the system clock frequency.

- Any signal sampling that relies upon the internal system clock (e.g., counter inputs) can be unreliable since the system clock is turned off inside active debug mode between debug mode commands.

- Power management mode cannot be invoked in the first instruction executed when returning from active debug mode. The PMME bit is not set if such an attempt is made.

11.3.10 Debug Command Operation

The following sections provide specific notes on the MAXQ7665/MAXQ7666's operation in debugging mode.

11.3.10.1 Register Read and Write Commands

Any register location can be read or written using these commands, including reserved locations and those used for op code support. No protection is provided by the debugging interface, and avoiding side effects is the responsibility of the host system communicating with the MAXQ7665/MAXQ7666.

Writing to the IP register alters the address that execution resumes at once the debugging engine exits.

In general, reading a register through the debug interface returns the value that was in that register before the debugging engine was invoked. An exception to this rule is the SP register. Reading the SP register through the debug interface actually returns the value (SP + 1).

11.3.10.2 Data Memory Read Command

When invoking this command, ICDA should be set to the word address of the starting location to read from, and ICDD should be set to the number of words. The input address must be based on the utility ROM memory map, as shown in Section 1. Data memory words returned by this command are output LSB first.

11.3.10.3 Data Memory Write Command

When invoking this command, ICDA should be set to the word address of the location to write to, and ICDD should be set to the data word to write. The input address must be based on the utility ROM memory map, as shown in Section 1.

11.3.10.4 Program Stack Read Command

When invoking this command, ICDA should be set to the address of the starting stack location (value of SP) to read from, and ICDD should be set to the number of words. The address given in ICDA is the highest value that will be used, as words are popped off the stack and returned in descending order. Stack words returned by this command are output LSB first.

11.3.10.5 Read Register Map Command

This command outputs all peripheral registers in the range M0[00h] to M5[0Dh], along with a fixed set of system registers. The following formatting rules apply to the returned data.

- System registers are output as 8-bit or 16-bit, least significant byte first.
- All peripheral registers are output as 16-bit, least significant byte first. The top byte of 8-bit registers is returned as 00h.
- Non-implemented peripheral registers in the range M0[00h] to M5[0Dh] are returned as 0000h.
- The value of SBUF0, SPIB, C0S, C0DB, C0RMS, C0TMA, and ASR are not read, and this register is returned as 0000h.

The first byte output by this command is the value 174 (AEh), which represents the number of peripheral registers output. Table 11-3 lists the remaining 412 bytes output by this command.

Table 11-3. Output from DebugReadMap Command

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	PO0		00	00	00	00	EIF0		00	00	00	00	00	00	00	00
1x	PI0		00	00	00	00	EIE0		00	00	00	00	00	00	00	00
2x	PD0		00	00	00	00	EIES0		00	00	00	00	00	00	00	00
3x	00	00	00	00	00	00	00	00	00	00	SCON0		SMD0		PR0	
4x	MCNT		MA		MB		MC2		MC1		MC0		00	00	SPICN	
5x	SPICF		SPICK		FCNTL		FDATA		MC1R		MC0R		00	00	00	00
6x	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7x	00	00	00	00	00	00	00	00	FADDR		00	00	00	00	00	00
8x	T2CNA0		T2H0		T2RH0		T2CH0		T2CNA1		T2H1		T2RH1		T2CH1	
9x	T2CNB0		T2V0		T2R0		T2C0		T2CNB1		T2V1		T2R1		T2C1	
Ax	T2CFG0		T2CFG1		00	00	00	00	00	00	00	00	00	00	00	00
Bx	ICDT0		ICDT1		00	ICDC	00	ICDF	00	ICDB	ICDA		ICDD		TM	
Cx	T2CNA2		T2H2		T2RH2		T2CH2		00	00	00	00	00	00	00	00
Dx	T2CNB2		T2V2		T2R2		T2C2		00	00	00	00	00	00	00	00
Ex	T2CFG2		00	00	00	00	00	00	00	00	00	00	00	00	00	00
Fx	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10x	C0C		00	00	C0IR		C0TE		C0RE		COR		C0DP		00	00
11x	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
12x	00	00	C0M1C		C0M2C		C0M3C		C0M4C		C0M5C		C0M6C		C0M7C	
13x	C0M8C		C0M9C		C0M10C		C0M11C		C0M12C		C0M13C		C0M14C		C0M15C	
14x	VMC		APE		ACNT		DCNT		DACI		00	00	DACO		00	00
15x	ADCD		TSO		AIE		00	00	OSCC		OTP		AP	APC	PSF	IC
16x	IMR	SC	IIR	CKCN	WDCN	00	A[0]		A[1]		A[2]		A[3]		A[4]	
17x	A[5]		A[6]		A[7]		A[8]		A[9]		A[10]		A[11]		A[12]	
18x	A[13]		A[14]		A[15]		IP		SP+1		IV		LC[0]		LC[1]	
19x	OFFS		DPC		GR		BP		DP[0]		DP[1]					

SECTION 12: IN-SYSTEM PROGRAMMING

This section contains the following information:

LIST OF TABLES

SECTION 12: IN-SYSTEM PROGRAMMING

The MAXQ7665/MAXQ7666 are equipped with a bootstrap loader as part of the utility ROM firmware. The main function of the boot-strap loader is to provide in-system programming capability to the user application. The MAXQ7665/MAXQ7666 in-system program-ming features include:

- Standard JTAG/TAP interface based communication
- Built-in JTAG bootstrap loader for flash programming and verifying
- Password lock protection to access bootstrap loader operations

12.1 Bootstrap-Loader Mode

Internal flash memory for the MAXQ7665/MAXQ7666 can be initialized through bootstrap-loader mode. To enable the bootstrap loader and establish a desired communication channel, the system programming instruction (100b) must be loaded into the TAP instruction register using the IR-scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:IR0) and is recognized by the TAP controller in the update-IR state, a 3-bit data shift register is activated as the communication channel for DR-scan sequences. The TAP retains the system programming instruction until a new instruction is shifted in or the TAP controller returns to the test-logic-reset state. This 3-bit shift register formed between the TDI and TDO pins is directly interfaced to the 3-bit serial programming buffer (SPB). The SPB contains three bits with the following functions:

- **SPB.0: System Programming Enable (SPE).** Setting this bit to logic 1 denotes that system programming is desired upon exit-ing reset. When it is cleared to logic 0, no system programming is needed. The reset vector examines the logic state of SPE in the utility ROM to determine the program flow after a reset. When SPE = 1, the bootstrap loader selected by the PSS1:PSS0 bits are activated to perform a bootstrap-loader function. When SPE = 0, the utility ROM transfers execution control to the normal user program.

- **SPB.2 and SPB.1: Programming Source Select (PSS1:PSS0).** These bits allow the host to select programming interface sources. The PSS bits have no functions when the SPE bit is cleared.

  Note: The MAXQ7665/MAXQ7666 utility ROM bootstrap loader supports only JTAG as the programming source (see Table 12-1).

The DR-scan sequence is used to configure the SPB bits. The data content of the SPB register is reflected in the ICDF register and allows read/write access by the CPU. These bits are cleared by power-on reset or test-logic-reset of the TAP controller.

Table 12-1. Programming Source Select Decode

PSS1	PSS0	PROGRAMMING SOURCE
0	0	JTAG
0	1	Reserved
1	0	Reserved
1	1	Reserved

12.2 In-System Programming Peripheral Registers

The MAXQ7665/MAXQ7666 in-system programming peripheral registers are described here. All the in-system programming peripheral registers are directly accessible by the microcontroller through the module/index address.

12.2.1 In-Circuit Debug Flag Register (ICDF)

Register Description: **In-Circuit Debug Flag Register**
Register Name: **ICDF**
Register Address: **Module 02h, Index 1Bh**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	PSS1	PSS0	SPE	TXC
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

r = read, w = write

Bits 7 to 4: Reserved.

Bits 3 and 2: Programming Source Select Bits 1 and 0 (PSS1:PSS0). These bits are used to select a programming interface during in-system programming when SPE is set to logic 1. Otherwise, the logic values of these bits have no meaning. The logical states of these bits, when read by the CPU, reflect the logical-OR of the PSS bits that are write accessible by the CPU and those in the system programming buffer register (SPB) of the TAP module (which are accessible via JTAG). These bits are read/write accessible for the CPU and are cleared to 0 by a power-on reset or test-logic-reset. CPU writes to the PSS bits result in clearing of the JTAG PSS1:PSS0 bits. See Table 12-1.

Bit 1: System Program Enable (SPE). This bit controls the behavior of the MAXQ7665/MAXQ7666 following a reset. The SPE bit is used for in-system programming support, and its logical state, when read by the CPU, always reflects the logical-OR of the SPE bit that is write accessible by the CPU and the SPE bit of the SPB register in the TAP module, which is accessible via JTAG. The logical state of this bit determines the program flow after a reset.

    0 = The MAXQ7665/MAXQ7666 jump to application code in flash at 0000h following a reset.
    1 = The MAXQ7665/MAXQ7666 execute the in-system programming boot loader following a reset.

This bit allows read/write access by the CPU and is cleared to 0 only on a power-on reset or test-logic-reset. The JTAG SPE bit is cleared by hardware when the ROD bit is set. CPU writes to the SPE bit result in clearing of the JTAG PSS1:PSS0 bits.

Bit 0: Serial Transfer Complete (TXC). See *Section 11* for more information on this bit.

12.2.2 System Control Register (SC)

Register Description: **System Control Register**
Register Name: **SC**
Register Address: **Module 08h, Index 08h**

Bit #	7	6	5	4	3	2	1	0
Name	TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
Reset	1	0	0	0	0	0	1*	0
Access	rw	r	rw	rw	rw	rw	rw	r

r = read, w = write
*This register defaults to 80h on all forms of reset except after power-on reset. After power-on resert, the PWL bit is also set and this register defaults to 82h.

Bit 7: Test Access (JTAG) Port Enable (TAP). This bit controls whether the TAP special function pins are enabled. The TAP defaults to being enabled. See *Section 10* for more information on this bit.

   0 = JTAG/TAP functions are disabled and P0.0–P0.3 can be used as general-purpose I/O pins
   1 = TAP special function pins P0.0–P0.3 are enabled to act as JTAG inputs and outputs

Bits 6 and 0: Reserved.

Bits 5 and 4: Code Data Access Bits 1 and 0 (CDA1:CDA0). See *Section 1* for more information on these bits.

Bit 3: Upper Program Access (UPA). See *Section 1* for more information on this bit.

Bit 2: ROM Operation Done (ROD). This bit is used to signify completion of a ROM operation sequence to the control units. This allows the debug engine to determine the status of a ROM sequence. Setting this bit to logic 1 causes an internal system reset if the SPE bit is also set. Setting the ROD bit will clear the SPE bit if it is set and the ROD bit will be automatically cleared by hardware once the control unit acknowledges the done indication. See *Section 11* for more information on this bit.

Bit 1: Password Lock (PWL). This bit defaults to 1 on a power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the program space before allowing access to the password protected in-circuit debug or bootstrap loader ROM routines. Clearing this bit to 0 disables the password protection for these ROM routines.

The password is defined as the 16 words of physical program memory at addresses 0010h to 001Fh. A password value of all ones or all zeros for all 16 words at addresses 0010h to 001Fh will also unlock the password lock, regardless of the state of the PWL bit.

12.3 JTAG Bootloader Operation

The MAXQ7665/MAXQ7666 JTAG bootloader uses the same status bit handshaking hardware as is used for in-circuit debugging. When the SPE bit of the system programming buffer (SPB) is set to 1 and JTAG is selected as the programming source (PSS1:PSS0 = 00b), the background and active-debug-mode state machines are disabled. Once the host loads the debug instruction into the TAP instruction register (IR2:IR0), the 10-bit shift register interfaces to ICDB and the status bits becomes available for host-to-ROM boot-loader communication. The status bits should be interpreted as noted in Table 12-2 for a JTAG bootloader operation.

When the using the JTAG bootloader option (SPE = 1, PSS1:PSS0 = 00b), the sole purpose of the debug hardware is to simultaneously transfer the data byte shifted in from the host into the ICDB register and transfer the contents of an internal holding register (loaded by ROM code writes of ICDB) into the shift register for output to the host. This transfer takes place on the falling edge of TCK at the update-DR state. The debug hardware additionally clears the TXC bit at this point in the state diagram. The ROM-loader code controls the status bit output to the host by asserting TXC = 1 when it has valid data to be shifted out. The ROM code can flexibly implement whatever communication protocol and command set it wishes within the data byte portion of the shifted 10-bit word. The communication protocol implemented as part of the MAXQ7665/MAXQ7666 utility ROM JTAG bootloader is described in *Section 12.5*.

Table 12-2. JTAG Status Decode

BITS (1:0)		STATUS	CONDITION
0	0	Reserved	Invalid condition
0	1	Reserved	Invalid condition
1	0	Loader-Busy	ROM loader is busy executing code or processing the current command.
1	1	Loader-Valid	ROM loader is supplying valid output data to the host in current shift operation.

12.4 Password-Protected Access

Some applications require preventive measures to protect against simple access and viewing of program code memory. To address this need for code protection, the MAXQ7665/MAXQ7666 utility ROM that manages in-system programming, in-application programming, or in-circuit debugging grants full access to those utilities only after a password has been supplied. The password is defined as the 16 words of physical program memory at addresses 0010h to 001Fh. Note that using these memory locations as a password does not exclude their usage for general code space if a unique password is not needed. A single password-lock bit (PWL) is implemented in the SC register. When the PWL is set to 1, a password is required to access the in-circuit debug and in-system programming ROM routines that allow reading or writing of internal memory. When PWL is cleared to 0, these utilities are fully accessible through the utility ROM without a password.

The PWL bit defaults to 1 by a power-on reset. To access the ROM utilities, a correct password is needed; otherwise, access to the ROM utilities is denied. Once the user supplies the correct password, the ROM clears the password lock. The PWL remains clear until either a power-on reset occurs or it is set to logic 1 by user software.

For the MAXQ7665/MAXQ7666, the password is always known for a fully erased device since the unprogrammed state of these memories is all ones. Password data set to all ones or all zeros for all 16 words at addresses 0010h to 001Fh will remove the password lock, regardless of the state of the PWL bit. Once the memory has been programmed, a password is established and can be used for access protection. The utility ROM code denies access to the protected routines when PWL indicates a locked state.

12.4.1 Entering a Password

A password can be entered via the TAP interface directly by issuing the unlock-password debug-mode command. The unlock-password command requires 32 follow-on transfer cycles, each containing a byte value to be compared with the program memory password.

12.5 JTAG Bootloader Protocol

When communicating with the bootloader using the JTAG interface, the clock rate (TCK) must be kept below 1/8 the system clock rate.

All bootloader commands begin with a single command byte. The high four bits of this command byte define the command family (from 0 to 15), while the low four bits define the specific command within that family. All commands (except for those in Family 0) follow this format:

BYTE 1	BYTE 2	BYTE 3	BYTE 4	(LENGTH) BYTES/WORDS
Command	Length	Param 1	Param 2	Data

After each command has completed, the loader outputs a "prompt" byte to indicate that it has finished the operation. The prompt byte is the single chararacter ">".

Bootloader commands that fail for any reason set the bootloader status byte to an error code value describing the reason for the failure. See Table 12-3. This status byte can be read by means of the Get Status command (04h).

Table 12-3. Bootloader Status Codes

STATUS VALUE	FUNCTION
00	**No Error.** The last command completed successfully.
01	**Family Not Supported.** An attempt was made to use a command from a family the bootloader does not support.
02	**Invalid Command.** An attempt was made to use a nonexistent command within a supported command family.
03	**No Password Match.** An attempt was made to use a password-protected command without first matching a valid password. Or, the Password Match command was called with an incorrect password value.
04	**Bad Parameter.** The parameter (address or otherwise) passed to the command was out of range or otherwise invalid.
05	**Verify Failed.** The verification step failed on a Load/Verify or Verify command.
06	**Unknown Register.** An attempt was made to read from or write to a nonexistent register.
07	**Word Mode Not Supported.** An attempt was made to set word mode access, but the bootloader supports byte mode access only.
08	**Master Erase Failed.** The bootloader was unable to perform master erase.

All commands in Family 0 can be executed without first matching the password. All other commands (in Families 1x through Fx) are password protected; the password must first be matched before these commands can be executed.

A special case exists when the program memory has not been initialized (following master erase). If the password (stored in word locations 0010h to 001Fh in program memory) is all 0000h words or all FFFFh words, the bootloader treats the password as having been matched. This allows access to password-protected commands following master erase (when no password has been set in program memory).

When providing addresses for code or data read or write to bootloader commands, all addresses run from 0000h to (memory size–1).

12.5.1 Family 0 Commands (Not Password Protected)

Command 00h—No Operation

I/O	Byte 1
Input	00h
Output	

Command 01h—Exit Loader

This command causes the bootloader command loop to exit, and execution jumps to the beginning of application code.

I/O	Byte 1
Input	01h
Output	

Command 02h—Master Erase

This command clears (programs to FFFFh) all words in the program flash memory.

I/O	Byte 1
Input	02h
Output	

Command 03h—Password Match

This command accepts a 32-byte password value, which is matched against the password in program memory (in byte mode) from addresses 0020h to 003Fh. If the value matches, the password lock is cleared.

I/O	Byte 1	32 Bytes
Input	03h	Password value
Output		

Command 04h—Get Status

The status code returned by this command is defined in Table 12-3. The flags byte contains the following bit status flags.

I/O	Byte 1	Byte 2
Input	04h	
Output	Flags	Status Code

Table 12-4. Bootloader Status Flags

FLAG BIT	FUNCTION
0	**Password Lock** 0 = The password is unlocked or had a default value; password-protected commands can be used. 1 = The password is locked. Password-protected commands cannot be used.
1	**Word/Byte Mode** 0 = The bootloader is currently in byte mode for memory reads/writes. 1 = The bootloader is currently in word mode for memory reads/writes.
2	**Word/Byte Mode Supported** 0 = The bootloader supports byte mode only. 1 = The bootloader supports word mode as well as byte mode.
3 to 8	Reserved

Command 05h—Get Supported Commands

The SupportL (LSB) and SupportH (MSB) bytes form a 16-bit value that indicates which command families this bootloader supports. If bit 0 is set to 1, it indicates that Family 0 is supported. If bit 1 is set to 1, it indicates that Family 1 is supported, and so on.

The CodeLen and DataLen bytes return the fixed block lengths used by the Load/Dump/Verify Fixed Length commands for code and data space, respectively.

I/O	Byte 1	Byte 2	Byte 3	Byte 4
Input	05h			
Output	SupportL	SupportH	CodeLen	DataLen

Command 06h—Get Code Size

This command returns SizeH:SizeL, which represents the size of available code memory in words minus 1. If this command is unsupported, the return value will be 0000h meaning "unknown amount of memory."

I/O	Byte 1	Byte 2
Input	06h	
Output	SizeL	SizeH

Command 07h—Get Data Size

This command returns SizeH:SizeL, which represents the size of available data memory in words minus 1. If this command is unsupported, the return value will be 0000h meaning "unknown amount of memory."

I/O	Byte 1	Byte 2
Input	07h	
Output	SizeL	SizeH

Command 08h—Get Loader Version

I/O	Byte 1	Byte 2
Input	08h	
Output	VersionL	VersionH

MAXQ7665/MAXQ7666 User's Guide

Command 09h—Get Utility ROM Version

I/O	Byte 1	Byte 2
Input	09h	
Output	VersionL	VersionH

Command 0Ah—Set Word/Byte Mode Access

The Mode byte should be 0 to set byte access mode or 1 to set word access mode. The current access mode is returned in the status flag byte by command 04h, as well as a flag to indicate whether word access mode is supported by this particular bootloader.

I/O	Byte 1	Byte 2
Input	0Ah	Mode
Output		

Command 0Dh—Get ID Information

For the MAXQ7665/MAXQ7666, the information returned by this command is a zero-terminated ROM banner string.

I/O	Byte 1	(Variable)
Input	0Dh	
Output		Device dependent information

12.5.2 Family 1 Commands: Load Variable Length (Password Protected)

Command 10h—Load Code Variable Length

This command programs (Length) bytes/words of data into the program flash starting at address (AddressH:AddressL), with the following restrictions.

- In byte mode, if the starting address is on an odd word boundary (such as 0001), the low bit will be changed to zero to make it an even word address.
- In byte mode, if an odd number of bytes is input, the data will be padded out with a 00 to make it an even number.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	10h	Length	AddressL	AddressH	Data to load
Output					

Command 11h—Load Data Variable Length

This command writes (Length) bytes/words of data into the data SRAM starting at address (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	11h	Length	AddressL	AddressH	Data to load
Output					

12.5.3 Family 2 Commands: Dump Variable Length (Password Protected)

Command 20h—Dump Code Variable Length

This command has a slightly different format depending on the length of the dump requested. It returns the contents of the application flash/ROM—(LengthL) or (LengthH:LengthL) bytes/words starting at (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Input (to dump < 256 bytes/words)	20h	1	AddressL	AddressH	LengthL	
Input (to dump 256+ bytes/words)	20h	2	AddressL	AddressH	LengthL	LengthH
Output	CodeByte 1	CodeByte 2	• • •		CodeByte N, where N = dump length	

Command 21h—Dump Data Variable Length

This command has a slightly different format depending on the length of the dump requested. It returns the contents of the data SRAM—(LengthL) or (LengthH:LengthL) bytes/words starting at (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Input (to dump < 256 bytes/words)	21h	1	AddressL	AddressH	LengthL	
Input (to dump 256+ bytes/words)	21h	2	AddressL	AddressH	LengthL	LengthH
Output	DataByte 1	DataByte 2	• • •		DataByte N, where N = dump length	

12.5.4 Family 3 Commands: CRC Variable Length (Password Protected)

Command 30h—CRC Code Variable Length

This command has a slightly different format depending on the length of the CRC requested. It returns the CRC-16 value (CrcH:CrcL) of the application flash/ROM—(LengthL) or (LengthH:LengthL) bytes/words starting at (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Input (to CRC < 256 bytes/words)	30h	1	AddressL	AddressH	LengthL	
Input (to CRC 256+ bytes/words)	30h	2	AddressL	AddressH	LengthL	LengthH
Output	CrcH	CrcL				

Command 31h—CRC Data Variable Length

This command has a slightly different format depending on the length of the CRC requested. It returns the CRC-16 value (CrcH:CrcL) of the data SRAM – (LengthL) or (LengthH:LengthL) bytes/words starting at (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Input (to CRC < 256 bytes/words)	31h	1	AddressL	AddressH	LengthL	
Input (to CRC 256+ bytes/words)	31h	2	AddressL	AddressH	LengthL	LengthH
Output	CrcH	CrcL				

12.5.5 Family 4 Commands: Verify Variable Length (Password Protected)

Command 40h—Verify Code Variable Length

This command operates in the same manner as the "Load Code Variable Length" command, except that instead of programming the input data into code flash, it verifies that the input data matches the data already in code space. If the data does not match, the status code is set to reflect this failure.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	40h	Length	AddressL	AddressH	Data to verify
Output					

Command 41h—Verify Data Variable Length

This command operates in the same manner as the "Load Data Variable Length" command, except that instead of writing the input data into data SRAM, it verifies that the input data matches the data already in data space. If the data does not match, the status code is set to reflect this failure.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	41h	Length	AddressL	AddressH	Data to verify
Output					

12.5.6 Family 5 Commands: Load and Verify Variable Length (Password Protected)

Command 50h—Load and Verify Code Variable Length

This command combines the functionality of the "Load Code Variable Length" and "Verify Code Variable Length" commands.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	50h	Length	AddressL	AddressH	Data to load/verify
Output					

Command 51h—Load and Verify Data Variable Length

This command combines the functionality of the "Load Data Variable Length" and "Verify Data Variable Length" commands.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes/Words
Input	51h	Length	AddressL	AddressH	Data to load/verify
Output					

12.5.7 Family 6 Commands: Erase Variable Length (Password Protected)

Command 60h—Erase Data Variable Length

This command has a slightly different format depending on the length of the erase requested. It clears (LengthL) or (LengthH:LengthL) bytes/words in the data SRAM to zero starting at (AddressH:AddressL).

I/O	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Input (to erase < 256 bytes/words)	60h	1	AddressL	AddressH	LengthL	
Input (to erase 256+ bytes/words)	60h	2	AddressL	AddressH	LengthL	LengthH

12.5.8 Family 9 Commands: Load Fixed Length (Password Protected)

Command 90h—Load Code Fixed Length

This command loads a block of 128 bytes into the program memory (SRAM) starting at the specified address. The address is rounded down to the nearest block boundary (multiple of 64) before the data is loaded.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	128 Bytes
Input	90h	80h	AddressL	AddressH	Data to load
Output					

Command 91h—Load Data Fixed Length

This command loads a block of 2 bytes into the data memory (SRAM) starting at the specified address.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	2 Bytes
Input	91h	2h	AddressL	AddressH	Data to load
Output					

12.5.9 Family D Commands: Load/Verify Fixed Length (Password Protected)

Command D0h—Load/Verify Code Fixed Length

This command loads a block of 128 bytes into the program memory starting at the specified address and immediately verifies to make sure the correct data was written. The address is rounded down to the nearest block boundary (multiple of 64) before the data is loaded.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	128 Bytes
Input	D0h	80h	AddressL	AddressH	Data to load
Output					

Command D1h—Load/Verify Data Fixed Length

This command loads a block of 2 bytes into the data memory starting at the specified address and immediately verifies to make sure the correct data was written.

I/O	Byte 1	Byte 2	Byte 3	Byte 4	2 Bytes
Input	D1h	2h	AddressL	AddressH	Data to load
Output					

12.5.10 Family E Commands: Erase Fixed Length (Password Protected)

Command E0h—Erase Code Fixed Length

This command erases (programs to FFFFh) a 64-byte block of the program flash memory. The address given should be located in the block to be erased.

I/O	Byte 1	Byte 2	Byte 3	Byte 4
Input	E0h	0	AddressL	AddressH
Output				

Command E1h—Erase Data Fixed Length

This command erases a single word/byte in data SRAM to zero at (AddressH:AddressL) to zero.

I/O	Byte 1	Byte 2	Byte 3	Byte 4
Input	E1h	0	AddressL	AddressH
Output				

SECTION 13: HARDWARE MULTIPLIER MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 13: HARDWARE MULTIPLIER MODULE

The MAXQ7665/MAXQ7666 microcontrollers include a hardware multiplier module to support high-speed multiplications. The hardware multiplier module is equipped with two 16-bit operand registers, a 32-bit read-only result register, and an accumulator of 48-bit width. The multiplier can complete a 16-bit x 16-bit multiply-and-accumulate/subtract operation in a single cycle. The hardware multiplier module supports the following operations without interfering with the normal core functions:

- Signed or Unsigned Multiply (16 bit x 16 bit)

- Signed or Unsigned Multiply-Accumulate (16 bit x 16 bit)

- Signed or Unsigned Multiply-Subtract (16 bit x 16 bit)

- Signed Multiply and Negate (16 bit x 16 bit)

13.1 Hardware Multiplier Organization

The hardware multiplier consists of two 16-bit, parallel-load operand registers (MA, MB); a read-only result register formed by two parallel 16-bit registers (MC1R and MC0R); an accumulator, which is formed by three 16-bit parallel registers (MC2, MC1, and MC0); and a status/control register (MCNT). Figure 13-1 shows a block diagram of the hardware multiplier.

The main arithmetic unit is the 16-bit x 16-bit multiplier that processes operands feeding from the MA and MB registers and generates a 32-bit final product. The multiplier unit includes an adder that can be used to perform a final accumulate/subtract operation of the multiplier output with the MC2:MC0 registers. The MCNT register must be configured to select the desired operation and operand count prior to loading the operand(s) to trigger the multiplier operation.
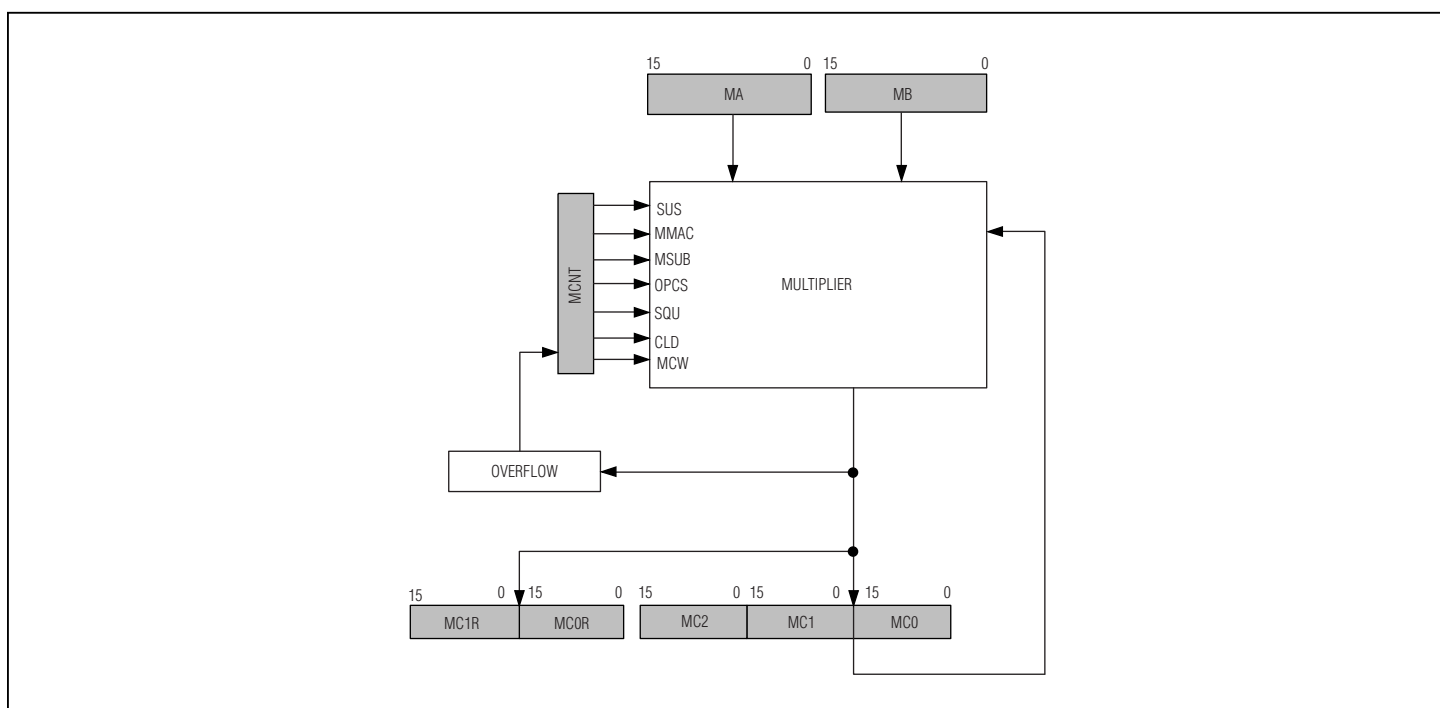
Figure 13-1. MAXQ7665/MAXQ7666 Multiplier Organization

13.2 Hardware Multiplier Peripheral Registers

13.2.1 Hardware Multiplier Control Register (MCNT)

Register Description: **Hardware Multiplier Control Register**
Register Name: **MCNT**
Register Address: **Module 001, Index 00h**

Bit #	15	14	13	12	11	10	9	8
Name	—	—	—	—	—	—	—	—
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	OF	MCW	CLD	SQU	OPCS	MSUB	MMAC	SUS
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 8: Reserved. Read returns 0, write ignored.

Bit 7: Overflow Flag (OF). This bit is set to logic 1 when an overflow occurred for the last operation. This bit can be set for accumulation/subtraction operations or unsigned multiply-negate attempts. This bit is automatically cleared to 0 following a reset, starting a multiplier operation, or setting of the CLD bit to 0.

Bit 6: MC Register Write Select (MCW). The state of the MCW bit determines if an operation result will be placed into the accumulator registers (MC).

    0 = The result is written to the MC registers.
    1 = The result is not written to the MC registers (MC register content is unchanged).

Bit 5: Clear Data Registers (CLD). This bit initializes the operand registers and the accumulator of the multiplier. When it is set to 1, the contents of all data registers and the OF bit are cleared to 0 and the operand load counter is reset immediately. This bit is cleared by hardware automatically. Writing a 0 to this bit has no effect.

Bit 4: Square Function Enable (SQU). This bit supports the hardware square function. When this bit is set to logic 1, a square operation is initiated after an operand is written to either the MA or the MB register. Writing data to either of the operand registers writes to both registers and triggers the specified square or square-accumulate/subtract operation. Setting this bit to 1 also overrides the OPCS bit setting. When SQU is cleared to logic 0, the hardware square function is disabled.

    0 = Square function disabled.
    1 = Square function enabled.

Bit 3: Operand Count Select (OPCS). This bit defines how many operands must be loaded to trigger a multiply or multiply-accumulate/subtract operation (except when SQU = 1 since this implicitly specifies a single operand). When this bit is cleared to logic 0, both operands (MA and MB) must be written to trigger the operation. When this bit is set to 1, the specified operation is triggered once either operand is written.

> 0 = Both operands (MA and MB) must be written to trigger the multiplier operation.
> 1 = Loading one operand (MA or MB) triggers the multiplier operation.

Bit 2: Multiply Negate (MSUB). Configuring this bit to logic 1 enables negation of the product for signed multiply operations and subtraction of the product from the accumulator (MC2:MC0) when MMAC = 1. When MSUB is configured to logic 0, the product of multiply operations will not be negated and accumulation is selected when MMAC = 1.

Bit 1: Multiply-Accumulate Enable (MMAC). This bit enables the accumulate or subtract operation (as per MSUB) for the hardware multiplier. When this bit is cleared to logic 0, the multiplier will perform only multiply operations. When this bit is set to logic 1, the multiplier will perform a multiply-accumulate or multiply-subtract operation based upon the MSUB bit.

> 0 = Accumulate/subtract operation disabled.
> 1 = Accumulate/subtract operation enabled.

Bit 0: Signed-Unsigned Select (SUS). This bit determines the data type of the operands. When this bit is cleared to logic 0, the operands are treated as two's complement values and the multiplier performs a signed operation. When this bit is set to logic 1, the operands are treated as absolute magnitudes and the multiplier performs an unsigned operation.

> 0 = Signed operands.
> 1 = Unsigned operands.

13.2.2 Multiplier Operand A Register (MA)

Register Description: **Multiplier Operand A Register**
Register Name: **MA**
Register Address: **Module 001, Index 01h**

Bit #	15	14	13	12	11	10	9	8
Name	MA.15	MA.14	MA.13	MA.12	MA.11	MA.10	MA.9	MA.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MA.7	MA.6	MA.5	MA.4	MA.3	MA.2	MA.1	MA.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Operand A Register Bits 15 to 0 (MA.15 to MA.0). This operand A register is used by the application code to load 16-bit values for multiplier operations.

13.2.3 Multiplier Operand B Register (MB)

Register Description: **Multiplier Operand B Register**
Register Name: **MB**
Register Address: **Module 001, Index 02h**

Bit #	15	14	13	12	11	10	9	8
Name	MB.15	MB.14	MB.13	MB.12	MB.11	MB.10	MB.9	MB.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MB.7	MB.6	MB.5	MB.4	MB.3	MB.2	MB.1	MB.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Operand B Register Bits 15 to 0 (MB.15 to MB.0). This operand B register is used by the application code to load 16-bit values for multiplier operations.

13.2.4 Multiplier Accumulator 2 Register (MC2)

Register Description: **Multiplier Accumulator 2 Register**
Register Name: **MC2**
Register Address: **Module 001, Index 03h**

Bit #	15	14	13	12	11	10	9	8
Name	MC2.15	MC2.14	MC2.13	MC2.12	MC2.11	MC2.10	MC2.9	MC2.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MC2.7	MC2.6	MC2.5	MC2.4	MC2.3	MC2.2	MC2.1	MC2.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Accumulator 2 Register Bits 15 to 0 (MC2.15 to MC2.0). The MC2 register represents the two most significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0. For a signed operation, the most significant bit of this register is the sign bit.

13.2.5 Multiplier Accumulator 1 Register (MC1)

Register Description: **Multiplier Accumulator 1 Register**
Register Name: **MC1**
Register Address: **Module 001, Index 04h**

Bit #	15	14	13	12	11	10	9	8
Name	MC1.15	MC1.14	MC1.13	MC1.12	MC1.11	MC1.10	MC1.9	MC1.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MC1.7	MC1.6	MC1.5	MC1.4	MC1.3	MC1.2	MC1.1	MC1.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Accumulator 1 Register Bits 15 to 0 (MC1.15 to MC1.0). The MC1 register represents bytes 3 and 2 of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

13.2.6 Multiplier Accumulator 0 Register (MC0)

Register Description: **Multiplier Accumulator 0 Register**
Register Name: **MC0**
Register Address: **Module 001, Index 05h**

Bit #	15	14	13	12	11	10	9	8
Name	MC0.15	MC0.14	MC0.13	MC0.12	MC0.11	MC0.10	MC0.9	MC0.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MC0.7	MC0.6	MC0.5	MC0.4	MC0.3	MC0.2	MC0.1	MC0.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Accumulator 0 Register Bits 15 to 0 (MC0.15 to MC0.0). The MC0 register represents the two least significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

13.2.7 Multiplier Read Register 1 (MC1R)

Register Description: **Multiplier Read Register 1**
Register Name: **MC1R**
Register Address: **Module 001, Index 0Ch**

Bit #	15	14	13	12	11	10	9	8
Name	MC1R.15	MC1R.14	MC1R.13	MC1R.12	MC1R.11	MC1R.10	MC1R.9	MC1R.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MC1R.7	MC1R.6	MC1R.5	MC1R.4	MC1R.3	MC1R.2	MC1R.1	MC1R.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Read Register 1 Bits 15 to 0 (MC1R.15 to MC1R.0). The MC1R register represents bytes' 3 and 2 result from the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC1 due to the combinatorial nature of the adder. The contents of this register may change if MCNT, MA, MB, or MC2:MC0 is changed.

13.2.8 Multiplier Read Register 0 (MC0R)

Register Description: **Multiplier Read Register 0**
Register Name: **MC0R**
Register Address: **Module 001, Index 0Dh**

Bit #	15	14	13	12	11	10	9	8
Name	MC0R.15	MC0R.14	MC0R.13	MC0R.12	MC0R.11	MC0R.10	MC0R.9	MC0R.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	MC0R.7	MC0R.6	MC0R.5	MC0R.4	MC0R.3	MC0R.2	MC0R.1	MC0R.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read
Note: *This register is cleared to 0000h on all forms of reset.*

Bits 15 to 0: Multiplier Read Register 0 Bits 15 to 0 (MC0R.15 to MC0R.0). The MC0R register represents bytes' 1 and 0 result from the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC0 due to the combinatorial nature of the adder. The contents of this register may change if MCNT, MA, MB, or MC2:MC0 is changed.

13.3 Hardware Multiplier Controls

The selection of operation to be performed by the multiplier is determined by four control bits in the MCNT register: SUS, MSUB, MMAC, and SQU. The number of operands that must be loaded to trigger the specified operation is dictated by the OPCS bit setting, except when the square function is enabled (SQU = 1). Enabling the square function implicitly defines that only a single operand (either MA or MB) needs to be loaded to trigger the square operation, independent of the OPCS bit setting. The MCNT register bits must be configured to select the desired operation and operand count prior to loading the operand(s) to trigger the multiplier operation. Any write to MCNT automatically resets the operand load counter of the multiplier, but does not affect the operand registers, unless such action is requested using the clear data registers (CLD) control bit. Once the desired operation has been specified via the MCNT register bits, loading the prescribed number of operands triggers the respective multiply, multiply-accumulate/subtract, or multiply-negate operation.

13.4 Register Output Selection

The hardware multiplier implements the MC register write select (MCW) control bit so that writing of the result to the MC2:MC0 registers can be blocked to preserve the MC registers (accumulator). When the MCW bit is configured to logic 1, the result for the given operation is not written to the MC registers. When the MCW bit is configured to logic 0, the MC registers are updated with the result of the operation. The MC1R, MC0R read-only register pair is updated independent of the MCW bit setting. This register pair always reflects the output that would normally be placed in MC1:MC0, given that MCW = 1 or MMAC = 0. When MCW = 0 and MMAC = 1, the MC1R:MC0R content may not match the MC1:MC0 register content, but it will be predictable and may be useful in certain situations. See Table 13-1 for details.

13.5 Signed-Unsigned Operand Selection

The operands can be either signed or unsigned numbers, but the data type must be defined by the user software via the signed-unsigned (SUS) bit prior to triggering the operation. For an unsigned operation, the SUS bit in the MCNT register must be set to 1; for a signed operation, the SUS bit must be cleared to 0. The multiplier treats unsigned numbers as absolute magnitude. For a 16-bit positional binary number, this represents a value in the range 0 to 2^{16} - 1 (FFFFh). The signed number representation is a two's complement value, where the most significant bit is defined as a sign bit. The range of a 16-bit two's complement number is $-2^{(16-1)}$ (8000h) to $+2^{(16-1)}$ - 1 (7FFFh). The product of any signed operation will be sign extended before being stored or accumulated/subtracted into the MC registers. The SUS bit should always be configured to logic 0 (i.e., signed operands) for the multiply-negate operation. Attempting an unsigned multiply-negate operation results in incorrect results and setting of the OF bit. Modifying the operand data type selection via the SUS bit does not alter the contents of the MC registers. The MC registers are read/write accessible and can be modified by user code when necessary.

13.6 Operand Count Selection

The OPCS bit allows selection of single operand or two operands operation for the multiply and multiply-accumulate/subtract operations. When the OPCS bit is cleared to 0, the multiply or multiply-accumulate/subtract operation established by the SUS, MSUB, and MMAC bits is triggered once two operands are loaded, one to each of the MA and MB registers. When OPCS is set to 1, the operation commences once data is loaded to either MA or MB. The OPCS bit is ignored when the square operation is enabled (SQU), since loading of data to the MA or MB register actually writes to both registers.

13.7 Hardware Multiplier Operations

The control bits, which specify data type (SUS), operand count (OPCS or SQU), and destination control (MCW), have already been described. However, there are two additional MCNT register bits that serve to define the hardware multiplier operation. The multiply-accumulate/subtract and multiply-negate operations are enabled by the multiply-accumulate enable (MMAC) and multiply negate (MSUB) bits in the MCNT register. When the MMAC bit is set to 1, the multiplier performs a multiply-accumulate (if MSUB = 0) or a multiply-subtract (if MSUB = 1). If MMAC is configured to 0, the multiplier result is not accumulated or subtracted, but can be stored directly (if MSUB = 0) or negated (if MSUB = 1) before storage. The multiply-negate operation (MMAC = 0, MSUB = 1) is only allowable for signed data operands (SUS = 0). For unsigned multiply-accumulate/subtract operations, the OF bit is set when a carry-out/borrow-in from the most significant bit of the MC register occurs. For a signed two's complement multiply-accumulate/subtract operations, the OF bit is set when the carry-out/borrow-in from the most significant magnitude position of the MC register is different from the carry-out/borrow-in of the sign position of the MC register. Since there is no overflow condition for multiply and multiply-negate operations, the OF bit is always cleared for these operations with one exception. The OF bit will be set to logic 1 if an unsigned multiply-negate (invalid operation) is requested. Table 13-1 shows the operations supported by the multiplier and associated MCNT control bit settings.

13.8 Accessing the Multiplier

There are no restrictions on how quickly data is entered into the operand registers or on the order of data entry. The only requirement to do a calculation is to perform the loading of MA and/or MB registers having specified data type and operation in the MCNT register. The multiplier keeps track of the writes to the MA and MB registers, and starts calculation immediately after the prescribed number of operands is loaded. If two operands are specified for the operation, the multiplier waits for the second operand to be loaded into the other operand register before starting the actual calculation. If for any reason software needs to reload the first operand, it should either reload that same operand register or use the CLD bit in the MCNT register to reinitialize the multiplier; otherwise, loading data to another operand register triggers the calculation. The CLD bit is a self-clearing bit that can be used for multiplier initialization. When it is set, it clears all data registers and the OF bit to zero and resets the multiplier operand write counter.

The specified hardware multiplier operation begins when the final operand(s) is loaded and will complete in a single cycle. The read-only MC1R, MC0R result registers can be accessed in the very next cycle unless accumulation/subtraction with MC2:MC0 is request-ed (MCW = 0 and MMAC = 1), in which case, one cycle is required so that stable data can be read. When MCW = 0, the MC2:MC0 registers always require one wait cycle before the operation result is accessible. The single wait cycle needed for updating the MC2:MC0 registers with a calculated result does not prevent initiating another calculation. Back-to-back operations can be triggered (independent of data type and operand count) without the need of wait state between loading of operands.

Table 13-1. MAXQ7665/MAXQ7666 Hardware Multiplier Operations

MCW:MSUB:MMAC	OPERATION	MC2	MC1	MC0	MC1R:MC0R	OF STATUS
000	Multiply	MA x MB			MA x MB	No
001	Multiply-Accumulate	MC + (MA x MB)			32 LSb of [MC + 2 x (MA x MB)]	Yes
010	Multiply-Negate (SUS = 0 Only)	-(MA x MB)			-(MA x MB)	No
011	Multiply-Subtract	MC - (MA x MB)			32 LSb of [MC - 2 x (MA x MB)]	Yes
100	Multiply	MC2	MC1	MC0	MA x MB	No
101	Multiply-Accumulate	MC2	MC1	MC0	32 LSb of [MC + (MA x MB)]	No
110	Multiply-Negate (SUS = 0 Only)	MC2	MC1	MC0	-(MA x MB)	No
111	Multiply-Subtract	MC2	MC1	MC0	32 LSb of [MC - (MA x MB)]	No

13.9 MAXQ7665/MAXQ7666 Hardware Multiplier Examples

The following are code examples of multiplier operations.

```
;Unsigned Multiply 16-bit x 16-bit
    move  MCNT, #21h          ; CLD=1, SUS=1 (unsigned)
    move  MA, #0FFFh          ; MC2:0=0000_0000_0000h
    move  MB, #1001h          ; MC1R:MC0R= 00FF_FFFFh
                              ; MC2:0=0000_00FF_FFFFh

;Signed Multiply 16-bit x 16-bit
    move  MCNT, #20h          ; CLD=1, SUS=0 (signed)
    move  MA, #F001h          ; MC2:0=0000_0000_0000h
    move  MB, #1001h          ; MC1R:MC0R= FF00_0001h
                              ; MC2:0=FFFF_FF00_0001h

;Unsigned Multiply-Accumulate 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
    move  MCNT, #03h          ; MMAC=1, SUS=1 (unsigned)
    move  MA, #0FFFh          ;
    move  MB, #1001h          ;
                              ; MC1R:MC0R=02FF_FFFFh
                              ; MC2:0=0000_0200_0000h

;Signed Multiply-Accumulate 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
    move  MCNT, #02h          ; SUS=0 (signed)
    move  MA, #F001h          ;
    move  MB, #1001h          ;
                              ; MC1R:MC0R= FF00_0003h
                              ; MC2:0=0000_0000_0002h

;Unsigned Multiply-Subtract 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
    move  MCNT, #07h          ; MMAC=1, MSUB=1, SUS=1 (unsigned)
    move  MA, #0FFFh          ;
    move  MB, #1001h          ;
                              ; MC1R:MC0R=FF00_0003h
                              ; MC2:0=0000_0000_0002h

;Signed Multiply-Subtract 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
    move  MCNT, #06h          ; MMAC=1, MSUB=1, SUS=0 (signed)
    move  MA, #F001h          ;
    move  MB, #1001h          ;
                              ; MC1R:MC0R= 02FF_FFFFh
                              ; MC2:0=0000_0200_0000h

;Signed Multiply Negate 16-bit x 16-bit
    move  MCNT, #24h          ; CLD=1, MSUB=1, SUS=0 (signed)
    move  MA, #F001h          ; MC2:0=0000_0000_0000h
    move  MB, #1001h          ; MC1R:MC0R =00FF_FFFFh
                              ; MC2:0=0000_00FF_FFFFh
```

SECTION 14: MAXQ7665/MAXQ7666 INSTRUCTION SET SUMMARY

This section contains the following information:

LIST OF TABLES

SECTION 14: MAXQ7665/MAXQ7666 INSTRUCTION SET SUMMARY

Table 14-1. MAXQ7665/MAXQ7666 Instruction Set Summary

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	NOTES
LOGICAL OPERATIONS	AND src	Acc ← Acc AND src	f001 1010 ssss ssss	S, Z	Y	1
	OR src	Acc ← Acc OR src	f010 1010 ssss ssss	S, Z	Y	1
	XOR src	Acc ← Acc XOR src	f011 1010 ssss ssss	S, Z	Y	1
	CPL	Acc ← ~Acc	1000 1010 0001 1010	S, Z	Y	
	NEG	Acc ← ~Acc + 1	1000 1010 1001 1010	S, Z	Y	
	SLA	Shift Acc left arithmetically	1000 1010 0010 1010	C, S, Z	Y	
	SLA2	Shift Acc left arithmetically twice	1000 1010 0011 1010	C, S, Z	Y	
	SLA4	Shift Acc left arithmetically four times	1000 1010 0110 1010	C, S, Z	Y	
	RL	Rotate Acc left (w/o C)	1000 1010 0100 1010	S	Y	
	RLC	Rotate Acc left (through C)	1000 1010 0101 1010	C, S, Z	Y	
	SRA	Shift Acc right arithmetically	1000 1010 1111 1010	C, Z	Y	
	SRA2	Shift Acc right arithmetically twice	1000 1010 1110 1010	C, Z	Y	
	SRA4	Shift Acc right arithmetically four times	1000 1010 1011 1010	C, Z	Y	
	SR	Shift Acc right (0 → msbit)	1000 1010 1010 1010	C, S, Z	Y	
	RR	Rotate Acc right (w/o C)	1000 1010 1100 1010	S	Y	
	RRC	Rotate Acc right (though C)	1000 1010 1101 1010	C, S, Z	Y	
BIT OPERATIONS	MOVE C, Acc.<b>	C ← Acc.<b>	1110 1010 bbbb 1010	C		
	MOVE C, #0	C ← 0	1101 1010 0000 1010	C		
	MOVE C, #1	C ← 1	1101 1010 0001 1010	C		
	CPL C	C ← ~C	1101 1010 0010 1010	C		
	MOVE Acc.<b>, C	Acc.<b> ← C	1111 1010 bbbb 1010	S, Z		
	AND Acc.<b>	C ← C AND Acc.<b>	1001 1010 bbbb 1010	C		
	OR Acc.<b>	C ← C OR Acc.<b>	1010 1010 bbbb 1010	C		
	XOR Acc.<b>	C ← C XOR Acc.<b>	1011 1010 bbbb 1010	C		
	MOVE dst.<b>, #1	dst.<b> ← 1	1ddd dddd 1bbb 0111	C, S, E, Z		2
	MOVE dst.<b>, #0	dst.<b> ← 0	1ddd dddd 0bbb 0111	C, S, E, Z		2
	MOVE C, src.<b>	C ← src.<b>	fbbb 0111 ssss ssss	C		
MATH	ADD src	Acc ← Acc + src	f100 1010 ssss ssss	C, S, Z, OV	Y	1
	ADDC src	Acc ← Acc + (src + C)	f110 1010 ssss ssss	C, S, Z, OV	Y	1
	SUB src	Acc ← Acc – src	f101 1010 ssss ssss	C, S, Z, OV	Y	1
	SUBB src	Acc ← Acc – (src + C)	f111 1010 ssss ssss	C, S, Z, OV	Y	1

Table 14-1. MAXQ7665/MAXQ7666 Instruction Set Summary (continued)

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	NOTES
BRANCHING	{L/S}JUMP src	IP ← IP + src or src	f000 1100 ssss ssss			6
	{L/S}JUMP C, src	If C=1, IP ← (IP + src) or src	f010 1100 ssss ssss			6
	{L/S}JUMP NC, src	If C=0, IP ← (IP + src) or src	f110 1100 ssss ssss			6
	{L/S}JUMP Z, src	If Z=1, IP ← (IP + src) or src	f001 1100 ssss ssss			6
	{L/S}JUMP NZ, src	If Z=0, IP ← (IP + src) or src	f101 1100 ssss ssss			6
	{L/S}JUMP E, src	If E=1, IP ← (IP + src) or src	0011 1100 ssss ssss			6
	{L/S}JUMP NE, src	If E=0, IP ← (IP + src) or src	0111 1100 ssss ssss			6
	{L/S}JUMP S, src	If S=1, IP ← (IP + src) or src	f100 1100 ssss ssss			6
	{L/S}DJNZ LC[n], src	If --LC[n] <> 0, IP← (IP + src) or src	f10n 1101 ssss ssss			6
	{L/S}CALL src	@++SP ← IP+1; IP ← (IP+src) or src	f011 1101 ssss ssss			6,7
	RET	IP ← @SP--	1000 1100 0000 1101			
	RET C	If C=1, IP ← @SP--	1010 1100 0000 1101			
	RET NC	If C=0, IP ← @SP--	1110 1100 0000 1101			
	RET Z	If Z=1, IP ← @SP--	1001 1100 0000 1101			
	RET NZ	If Z=0, IP ← @SP--	1101 1100 0000 1101			
	RET S	If S=1, IP ← @SP--	1100 1100 0000 1101			
	RETI	IP ← @SP-- ; INS← 0	1000 1100 1000 1101			
	RETI C	If C=1, IP ← @SP-- ; INS← 0	1010 1100 1000 1101			
	RETI NC	If C=0, IP ← @SP-- ; INS← 0	1110 1100 1000 1101			
	RETI Z	If Z=1, IP ← @SP-- ; INS← 0	1001 1100 1000 1101			
	RETI NZ	If Z=0, IP ← @SP-- ; INS← 0	1101 1100 1000 1101			
	RETI S	If S=1, IP ← @SP-- ; INS← 0	1100 1100 1000 1101			
DATA TRANSFER	XCH	Swap Acc bytes	1000 1010 1000 1010	S	Y	
	XCHN	Swap nibbles in each Acc byte	1000 1010 0111 1010	S	Y	
	MOVE dst, src	dst ← src	fddd dddd ssss ssss	C, S, Z, E	(Note 8)	7, 8
	PUSH src	@++SP ← src	f000 1101 ssss ssss			7
	POP dst	dst ← @SP--	1ddd dddd 0000 1101	C, S, Z, E		7
	POPI dst	dst ← @SP-- ; INS ← 0	1ddd dddd 1000 1101	C, S, Z, E		7
	CMP src	E ← (Acc = src)	f111 1000 ssss ssss	E		
	NOP	No operation	1101 1010 0011 1010			

Note 1: The active accumulator (Acc) is not allowed as the src in operations where it is the implicit destination.

Note 2: Only module 8 and modules 0-5 (when implemented for a given product) are supported by these single-cycle bit operations. Potentially affects C or E if PSF register is the destination. Potentially affects S and/or Z if AP or APC is the destination.

Note 3: The terms Acc and A[AP] can be used interchangeably to denote the active accumulator.

Note 4: Any index represented by <b> or found inside [ ] brackets is considered variable, but required.

Note 5: The active accumulator (Acc) is not allowed as the dst if A[AP] is specified as the src.

Note 6: The '{L/S}' prefix is optional.

Note 7: Instructions that attempt to simultaneously push/pop the stack (e.g. PUSH @SP--, PUSH @SPI--, POP @++SP, POPI @++SP) or modify SP in a conflicting manner (e.g., MOVE SP, @SP--) are invalid.

Note 8: Special cases: If 'MOVE APC, Acc' sets the APC.CLR bit, AP will be cleared, overriding any auto-inc/dec/modulo operation specified for AP. If 'MOVE AP, Acc' causes an auto-inc/dec/modulo operation on AP, this overrides the specified data transfer (i.e., Acc will not be transferred to AP).

ADD/ADDC *src*		**Add/Add with Carry**

Description: The ADD instruction sums the active accumulator (Acc or A[AP]) and the specified src data and stores the result back to the active accumulator. The ADDC instruction additionally includes the Carry (C) Status Flag in the summation. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: C, S, Z, OV

ADD Operation: Acc ← Acc + src

Encoding:

15			0
f100	1010	ssss	ssss

Example(s):

```
                    ;Acc = 2345h for each example
ADD A[3]     ; A[3]=FF0Fh
             ; → Acc =2254h,C=1, Z=0, S=0, OV=0
ADD #0C0h    ; → Acc =2405h,C=0, Z=0, S=0, OV=0
ADD A[4]     ; A[4]=C000h
             ; → Acc = E345h, C=0, Z=0, S=1, OV=0
ADD A[5]     ; A[5]=6789h
             ; → Acc = 8ACEh, C=0, Z=0, S=1, OV=0
```

ADDC Operation: Acc ← Acc + C + src

Encoding:

15			0
f110	1010	ssss	ssss

Example(s):

```
                    ; Acc = 2345h for each example
ADDC A[3]    ; A[3] = DCBAh, C=1
             ; → Acc = 0000h, C=1, Z=1, S=0, OV=0
ADDC @DP[0]-- ; @DP[0] = 00EEh, C=1
             ; → Acc = 2434h, C=0, Z=0, S=0, OV=0
```

Special Notes: The active accumulator (Acc) is not allowed as the src for these operations.

AND *src*	**Logical AND**

Description:	Performs a logical-AND between the active accumulator (Acc) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.
Status Flags:	S, Z
Operation:	Acc ← Acc AND src

Encoding:

15   0

f001	1010	ssss	ssss

Example(s):

```
                      ; Acc = 2345h for each example
AND A[3]      ; A[3]=0F0Fh
              ; → Acc = 0305h, S=0, Z=0
AND  #33h     ; → Acc = 0001h
AND #2233h    ; generates object code below
              ; MOVE PFX[0], #22h (smart-prefixing)
              ; AND #33h
              ; → Acc = 2201h
MOVE PFX[0], #0Fh
AND M0[8]     ; M0[8]=0Fh (assume M0[8] is an 8-bit register)
              ; → Acc = 0305h
```

Special Notes:	The active accumulator (Acc) is not allowed as the src for this operation.

AND *Acc.<b>*	**Logical AND Carry Flag with Accumulator Bit**

Description:	Performs a logical-AND between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.
Status Flags:	C
Operation:	C ← C AND Acc. <b>

Encoding:

15   0

1001	1010	bbbb	1010

Example(s):

```
                      ; Acc = 2345h, C=1 at start
AND Acc.0     ; Acc.0=1   → C=1
AND Acc.1     ; Acc.1=0   → C=0
AND C, Acc.8  ; Acc.8=1   → C=0
```

{L/S}CALL *src*	{Long/Short} Call to Subroutine

Description: Performs a call to the subroutine destination specified by src. The CALL instruction uses an 8-bit immediate src to perform a relative short call (IP +127/-128 words). The CALL instruction uses a 16-bit immediate src to perform an absolute long CALL to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute long CALL. Using the optional 'L' prefix (i.e., LCALL) results in an absolute long call and use of the PFX[0] register. Using the optional 'S' prefix (i.e., SCALL) attempts to generate a relative short call, but is flagged by the assembler if the destination is out of range. Specifying an internal register src (no matter whether 8-bit or 16-bit) always produces an absolute CALL to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register src is specified.

Status Flags: None

Operation:

@++SP ← IP + 1	PUSH
IP ← src	Absolute CALL
IP ← IP + src	Relative CALL

Encoding:

15   0

f011	1101	ssss	ssss

Example(s):

CALL label1	; relative call to label1 (must be within IP +127/ -
	; 128 address range)
CALL label1	; absolute call to label1 = 0120h
	; MOVE PFX[0], #01h
	; CALL #20h.
CALL DP[0]	; DP[0] holds 16-bit address of subroutine
CALL M0[0]	; assume M0[0] is an 8-bit register
	; absolute call to addr16
	; high(addr16)=00h (PFX[0])
	; low (addr16)=M0[0]
MOVE PFX[0], #22h	;
CALL M0[0]	; assume M0[0] is an 8-bit register
	; high(addr16)=22h (PFX[0])
	; low (addr16)=M0[0]
LCALL label1	; label=0120h and is relative to this instruction
	; absolute call is forced by use of 'L' prefix
	; MOVE PFX[0], #01h
	; CALL #20h
SCALL label1	; relative offset for label1 calculated and used
	; if label1 is not relative, assembler will generate an error
SCALL #10h	; relative offset of #10h is used directly by the CALL

CMP *src*   **Compare Accumulator**

Description:	Compare for equality between the active accumulator and the least significant byte of the specified src. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: E

Operation: Acc = src: E ← 1

Acc <> src: E ← 0

Encoding: 15   0

f111	1000	ssss	ssss

Example(s):   CMP #45h   ; Acc = 0145h, E=0

CMP #145h   ; PFX[0] register used

; MOVE PFX[0], #01h (smart-prefixing)

; CMP #45h E=1

CPL   **Complement Acc**

Description:	Performs a logical bitwise complement (1's complement) on the active accumulator (Acc or A[AP]) and returns the result to the active accumulator.

Status Flags: S, Z

Operation: Acc ← ~Acc

Encoding: 15   0

1000	1010	0001	1010

Example(s):   ; Acc = FFFFh, S=1, Z=0

CPL   ; Acc ← 0000h, S=0, Z=1

; Acc = 0990h, S=0, Z=0

CPL   ; Acc ← F66Fh, S=1, Z=0

CPL C   **Complement Carry Flag**

Description:	Logically complements the Carry (C) Flag.
Status Flags:	C
Operation:	$C \leftarrow \sim C$

Encoding: 15   0

1101	1010	0010	1010

Example(s):   ; C = 0

CPL C   ; C $\leftarrow$ 1

{L/S}DJNZ LC[n], *src*   **Decrement Counter, {Long/Short} Jump Not Zero**

Description:	The DJNZ LC[n], src instruction performs a conditional branch based upon the associated Loop Counter (LC[n]) register. The DJNZ LC[n], src instruction decrements the LC[n] loop counter and branches to the address defined by src if the decremented counter has not reached 0000h. Program branches can be relative or absolute depending upon the src specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the JUMP src op code.
Status Flags:	None
Operation:	$LC[n] \leftarrow LC[n] -1$
	LC[n] <> 0: IP $\leftarrow$ IP + src (relative) -or- src (absolute)
	LC[n] = 0: IP $\leftarrow$ IP + 1

Encoding: 15   0

f10n	1101	ssss	ssss

Example(s):   MOVE LC[1], #10h   ; counter = 10h

Loop:

ADD   @DP[0]++   ; add data memory contents to Acc, post-inc DP[0]

DJNZ LC[1], Loop   ; 16 times before falling through

{L/S}JUMP *src*   **Unconditional {Long/Short} Jump**

Description:	Performs an unconditional jump as determined by the src specifier. The JUMP instruction uses an 8-bit immediate src to perform a relative jump (IP +127/-128 words). The JUMP instruction uses a 16-bit immediate src to perform an absolute JUMP to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute JUMP. Using the optional 'L' prefix (i.e., LJUMP) results in an absolute long jump and use of the PFX[0] register. Using the optional 'S' prefix (i.e., SJUMP) attempts to generate a relative short jump, but is flagged by the assembler if the destination is out of range. Specifying an internal register src (no matter whether 8-bit or 16-bit) always produces an absolute JUMP to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register src is specified.
Status Flags:	None

Operation:

IP ← src	Absolute JUMP
IP ← IP + src	Relative JUMP

Encoding: 15   0

f000	1100	ssss	ssss

Example(s):

JUMP label1	; relative jump to label1 (must be within range
	; IP +127/-128 words)
JUMP label1	; absolute jump to label1= 0400h
	; MOVE PFX[0], #04h
	; JUMP #00h
JUMP DP[0]	; absolute jump to addr16 DP[0]
JUMP M0[0]	; assume M0[0] is an 8-bit register
	; absolute jump to addr16
	; high(addr16)=00h (PFX[0])
	; low (addr16)=M0[0]
LJUMP label1	; label=0120h and is relative to this instruction
	; absolute jump is forced by use of 'L' prefix
	; MOVE PFX[0], #01h
	; JUMP #20h
SJUMP label1	; relative offset for label1 calculated and used
	; if label1 is not relative, assembler will generate an error
SJUMP #10h	; relative offset of #10h is used directly by the JUMP

MAXQ7665/MAXQ7666 User's Guide

{L/S}JUMP C/{L/S}JUMP NC, *src*
L/S}JUMP Z/{L/S}JUMP NZ, *src*
{{L/S}JUMP E/{L/S}JUMP NE, *src*
{L/S}JUMP S, *src*

<div align="right">

Conditional {Long/Short} Jump on Status Flag

</div>

Description:	Performs conditional branching based upon the state of a specific processor status flag. JUMP C results in a branch if the Carry flag is set while JUMP NC branches if the Carry flag is clear. JUMP Z results in a branch if the Zero flag is set while JUMP NZ branches if the Zero flag is clear. JUMP E results in a branch if the Equal flag is set while JUMP NE branches if the Equal flag is clear. JUMP S results in a branch if the Sign flag is set. Program branches can be relative or absolute depending upon the src specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the JUMP src op code. Special src restrictions apply to JUMP E and JUMP NE.
Status Flags:	None

JUMP C
Operation:

C=1: IP ← IP + src (relative) -or- src (absolute)
C=0: IP ← IP + 1

Encoding: 15   0

f010	1100	ssss	ssss

Example(s):   JUMP C, label1   ; C=0, branch not taken

JUMP NC
Operation:

C=0: IP ← IP + src (relative) -or- src (absolute)
C=1: IP ← IP +1

Encoding: 15   0

f110	1100	ssss	ssss

Example(s):   JUMP NC, label1   ; C=0, branch taken

JUMP Z
Operation:

Z=1: IP ← IP + src
Z=0: IP ← IP + 1

Encoding: 15   0

f001	1100	ssss	ssss

Example(s):   JUMP Z, label1   ; Z=1, branch taken

JUMP NZ	Z=0: IP ← IP + src (relative) -or- src (absolute)
Operation:	Z=1: IP ← IP + 1
Encoding:	15 0

f101	1100	ssss	ssss

Example(s):	JUMP NZ, label1 ; Z=1, branch taken

JUMP E	E=1: IP ← IP + src (relative) -or- src (absolute)
Operation:	E=0: IP ← IP + 1
Encoding:	15 0

0011	1100	ssss	ssss

Example(s):	JUMP E, label1 ; E=1, branch taken
Special Notes:	The src specifier must be immediate data.

JUMP NE	
Operation:	E=0: IP ← IP + src (relative) -or- src (absolute)
	E=1: IP ← IP + 1
Encoding:	15 0

0111	1100	ssss	ssss

Example(s):	JUMP NE, label1 ; E=0, branch taken
Special Notes:	The src specifier must be immediate data.

JUMP S	S=1: IP ← IP + src (relative) -or- src (absolute)
Operation:	S=0: IP ← IP + 1
Encoding:	15 0

f100	1100	ssss	ssss

Example(s):	JUMP S, label1 ; S=0, branch not taken

MOVE dst, *src* **Move Data**

Description:	Moves data from a specified source (src) to a specified destination (dst). A list of defined source, destination specifiers is given in the table below. Also, since src can be either 8-bit (byte) or 16-bit (word) data, the rules governing data transfer are also explained below in the encoding section.
Status Flags:	S, Z (if dst is Acc or AP or APC)
	C, E (if dst is PSF)
Operation:	dst ← src
Encoding:	15 0

fddd	dddd	ssss	ssss
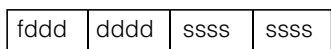
Table 14-2. Source Specifier Codes

src	src BIT ENCODING (f sssssss)	WIDTH (16 or 8)	DESCRIPTION
#k	0 kkkk kkkk	8	kkkkkkkk = Immediate (Literal) Data
MN[n]	1 nnnn 0NNN	8/16	nnnn Selects One of First 16 Registers in Module NNN; where NNN= 0 to 5. Access to Second 16 Using PFX[n].
AP	1 0000 1000	8	Accumulator Pointer
APC	1 0001 1000	8	Accumulator Pointer Control
PSF	1 0100 1000	8	Processor Status Flag Register
IC	1 0101 1000	8	Interrupt and Control Register
IMR	1 0110 1000	8	Interrupt Mask Register
SC	1 1000 1000	8	System Control Register
IIR	1 1011 1000	8	Interrupt Identification Register
CKCN	1 1110 1000	8	Clock Control Register
WDCN	1 1111 1000	8	Watchdog Control Register
A[n]	1 nnnn 1001	8/16	nnnn Selects One of 16 Accumulators
Acc	1 0000 1010	8/16	Active Accumulator = A[AP]. Update AP per APC
A[AP]	1 0001 1010	8/16	Active Accumulator = A[AP]. No change to AP
IP	1 0000 1100	16	Instruction Pointer
@SP--	1 0000 1101	16	16-Bit Word @SP, Post-Decrement SP
SP	1 0001 1101	16	Stack Pointer
IV	1 0010 1101	16	Interrupt Vector
LC[n]	1 011n 1101	16	n Selects 1 of 2 Loop Counter Registers
@SPI--	1 1000 1101	16	16-bit word @SP, Post-Decrement SP, INS=0
@BP[OFFS]	1 0000 1110	8/16	Data Memory @BP[OFFS]
@BP[OFFS++]	1 0001 1110	8/16	Data memory @BP[OFFS]; Post Increment OFFS
@BP[OFFS--]	1 0010 1110	8/16	Data Memory @BP[OFFS]; Post Decrement OFFS
OFFS	1 0011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	1 0100 1110	16	Data Pointer Control Register
GR	1 0101 1110	16	General Register
GRL	1 0110 1110	8	Low Byte of GR Register
BP	1 0111 1110	16	Frame Pointer Base Pointer (BP)
GRS	1 1000 1110	16	Byte-Swapped GR Register
GRH	1 1001 1110	8	High Byte of GR Register
GRXL	1 1010 1110	16	Sign Extended Low Byte of GR Register
FP	1 1011 1110	16	Frame Pointer (BP[OFFS])
@DP[n]	1 0n00 1111	8/16	Data Memory @DP[n]
@DP[n]++	1 0n01 1111	8/16	Data Memory @DP[n], Post-Increment DP[n]
@DP[n]--	1 0n10 1111	8/16	Data Memory @DP[n], Post-Decrement DP[n]
DP[n]	1 0n11 1111	16	n Selects 1 of 2 Data Pointers

Table 14-3. Destination Specifier Codes

dst	dst BIT ENCODING (ddd dddd)	WIDTH (16 OR 8)	DESCRIPTION
NUL	111 0110	8/16	Null (Virtual) Destination. Intended as a bit bucket to assist software with pointer increments/decrements.
MN[n]	nnn 0NNN	8/16	nnnn Selects One of First 8 Registers in Module NNN; where NNN= 0 to 5. Access to Next 24 Using PFX[n].
AP	000 1000	8	Accumulator Pointer
APC	001 1000	8	Accumulator Pointer Control
PSF	100 1000	8	Processor Status Flag Register
IC	101 1000	8	Interrupt and Control Register
IMR	110 1000	8	Interrupt Mask Register
A[n]	nnn 1001	8/16	nnn Selects 1 of First 8 Accumulators: A[0]..A[7]
Acc	000 1010	8/16	Active Accumulator = A[AP]
PFX[n]	nnn 1011	8	nnn Selects One of 8 Prefix Registers
@++SP	000 1101	16	16-Bit Word @SP, Pre-Increment SP
SP	001 1101	16	Stack Pointer
IV	010 1101	16	Interrupt Vector
LC[n]	11n 1101	16	n Selects 1 of 2 Loop Counter Registers
@BP[OFFS]	000 1110	8/16	Data Memory @BP[OFFS]
@BP[++OFFS]	001 1110	8/16	Data Memory @BP[OFFS]; Pre-Increment OFFS
@BP[--OFFS]	010 1110	8/16	Data Memory @BP[OFFS]; Pre-Decrement OFFS
OFFS	011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	100 1110	16	Data Pointer Control Register
GR	101 1110	16	General Register
GRL	110 1110	8	Low Byte of GR Register
BP	111 1110	16	Frame Pointer Base Pointer (BP)
@DP[n]	n00 1111	8/16	Data Memory @DP[n]
@++DP[n]	n01 1111	8/16	Data Memory @DP[n], Pre-Increment DP[n]
@--DP[n]	n10 1111	8/16	Data Memory @DP[n], Pre-Decrement DP[n]
DP[n]	n11 1111	16	n Selects 1 of 2 Data Pointers
2-CYCLE DESTINATION ACCESS USING PFX[n] REGISTER (See Special Notes)			
SC	000 1000	8	System Control Register
CKCN	110 1000	8	Clock Control Register
WDCN	111 1000	8	Watchdog Control Register
A[n]	nnn 1001	16	nnn Selects 1 of Second 8 Accumulators A[8]..A[15]
GRH	001 1110	8	High Byte of GR Register

Data Transfer	dst (16-bit) ← src (16-bit):	dst[15:0] ← src[15:0]
Rules	dst (8-bit) ← src (8-bit):	dst[7:0] ← src[7:0]
	dst (16-bit) ← src (8-bit):	dst[15:8] ← 00h *
		dst[7:0] ← src[7:0]
	dst (8-bit) ← src (16-bit):	dst[7:0] ← src[7:0]

__Note:__ The PFX[0] register may be used to supply a separate high-order data byte for this type of transfer.

Example(s):	MOVE A[0], A[3]	; A[0] ← A[3]
	MOVE DP[0], #110h	; DP[0] ← #0110h (PFX[0] register used)
		; MOVE PFX[0], #01h (smart-prefixing)
		; MOVE DP[0], #10h
	MOVE DP[0], #80h	; DP[0] ← #0080h (PFX[0] register not needed)

Special Notes: Proper loading of the PFX[n] registers, when for the purpose of supplying 16-bit immediate data or accessing 2-cycle destinations, is handled automatically by the assembler and is therefore an optional step for the user when writing assembly source code. Examples of the automatic PFX[n] code insertion by the assembler are demonstrated below.

Initial Assembly Code	**Assembler Output**
MOVE DP[0], #0100h	MOVE PFX[0], #01h
MOVE A[15], A[7]	MOVE PFX[2], anysrc
	MOVE A[7], A[7]
MOVE A[8], #3040h	
MOVE PFX[2], #30h	MOVE A[0], #40h

MOVE Acc.\<b\>, C   **Move Carry Flag to Accumulator Bit**

Description: Replaces the specified bit of the active accumulator with the Carry bit.

Status Flags: S, Z

Operation: Acc.\<b\> ← C

Encoding:

15   0

1111	1010	bbbb	1010

| **Example(s):** | | ; Acc = 8000h, S=1, Z=0, C=0 |
| | MOVE Acc.15, C | ; Acc = 0000h, S=0, Z=1 |

MOVE C, Acc.<b>   **Move Accumulator Bit to Carry Flag**

Description:	Replaces the Carry (C) status flag with the specified active accumulator bit.
Status Flags:	C
Operation:	C ← Acc.<b>
Encoding:	15 0

1110	1010	bbbb	1010

Example(s):	; Acc = 01C0h, C=0
	MOVE C, Acc.8 ; C =1

MOVE C, src.<b>   **Move Bit to Carry Flag**

Description:	Replaces the Carry (C) status flag with the specified source bit src.<b>.
Status Flags:	C
Operation:	C ← src.<b>
Encoding:	15 0

fbbb	0111	ssss	ssss

Example(s):	; M0[0] = FEh; C=1 (assume M0[0] is an 8-bit register)
	MOVE C, M0[0].0 ; C=0

MOVE C, #0   **Clear Carry Flag**

Description:	Clears the Carry (C) processor status flag.
Status Flag:	C ← 0
Operation:	C ← 0
Encoding:	15 0

1101	1010	0000	1010

Example(s):	; C = 1
	MOVE C, #0 ; C ← 0

MOVE C, #1	**Set Carry Flag**

Description: Sets the Carry (C) processor status flag.

Status Flag: $C \leftarrow 1$

Operation: $C \leftarrow 1$

Encoding:

15   0

1101	1010	0001	1010

Example(s):   ; C = 0

MOVE C, #1   ; $C \leftarrow 1$

MOVE dst.<b>, #0	**Clear Bit**

Description: Clears the bit specified by dst.<b>.

Status Flags: C, E (if dst is PSF), S, Z

Operation: dst.<b> $\leftarrow 0$

Encoding:

15   0

1ddd	dddd	0bbb	0111

Example(s):   ; M0[0] = FEh

MOVE M0[0].1, #0   ; M0[0] = FCh

MOVE M0[0].7, #0   ; M0[0] = 7Ch

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by MOVE dst.<b>, #0.

MOVE dst.<b>, #1	**Set Bit**

Description: Sets the bit specified by dst.<b>.

Status Flags: C, E (if dst is PSF), S, Z

Operation: dst.<b> $\leftarrow 1$

Encoding:

15   0

1ddd	dddd	1bbb	0111

Example(s):   ; M0[0] = 00h

MOVE M0[0].1, #1   ; M0[0] = 02h

MOVE M0[0].7, #1   ; M0[0] = 82h

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by MOVE dst.<b>, #1.

NEG   **Negate Accumulator**

Description:	Performs a negation (2's complement) of the active accumulator and returns the result back to the active accumulator.
Status Flags:	S, Z
Operation:	Acc ← ~Acc + 1
Encoding:	15 0

1000	1010	1001	1010

Example(s):	; Acc = FEEDh, S=1, Z=0
NEG	; Acc = 0113h, S=0, Z=0

OR *src*   **Logical OR**

Description:	Performs a logical-OR between the active accumulator (Acc or A[AP]) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.
Status Flags:	S, Z
Operation:	Acc ← Acc OR src
Encoding:	15 0

f010	1010	ssss	ssss

Example(s):	; Acc = 2345h for each example
OR A[3]	; A[3]= 0F0Fh → Acc = 2F4Fh
OR #1133h	; MOVE PFX[0], #11h (smart-prefixing)
	; OR #33h → Acc = 3377h
Special Notes:	The active accumulator (Acc) is not allowed as the src for this operation.

OR Acc.<i>\<b></i>   **Logical OR Carry Flag with Accumulator Bit**

Description:	Performs a logical-OR between the Carry (C) status flag and a specified bit of the active accumulator (Acc.\<b>) and returns the result to the Carry.
Status Flags:	C
Operation:	C ← C OR Acc.\<b>
Encoding:	15 0

1010	1010	bbbb	1010

Example(s):	; Acc = 2345h, C=0 at start
OR Acc.1	; Acc.1=0 → C=0
OR Acc.2	; Acc.2=1 → C=1

POP <i>dst</i>   **Pop Word from the Stack**

Description:	Pops a single word from the stack (@SP) to the specified dst and decrements the stack pointer (SP).
Status Flags:	S, Z (if dst = Acc or AP or APC)
	C, E (if dst = PSF)
Operation:	dst ← @ SP--
Encoding:	15 0

1ddd	dddd	0000	1101

Example(s):	; GR ← 1234h
POP GR	; @DP[0] ← 76h (WBS0=0)
POP @DP[0]	; @DP[0] ← 0876h (WBS0=1)

Stack Data:

xxxxh
1234h
0876h
xxxxh
xxxxh

← SP (initial)
← SP (after POP GR)
← SP (after POP @DP[0])

POPI dst   **Pop Word from the Stack Enable Interrupts**

Description:	Pops a single word from the stack (@SP) to the specified dst and decrements the stack pointer (SP). Additionally, POPI returns the interrupt logic to a state in which it can acknowledge additional interrupts.
Status Flags:	S, Z (if dst = Acc or AP or APC)
	C, E (if dst = PSF)
Operation:	dst ← @ SP--
	INS ← 0
Encoding:	15 0

1ddd	dddd	1000	1101

Example(s):	See POP

PUSH *src*   **Push Word to the Stack**

Description:	Increments the stack pointer (SP) and pushes a single word specified by src to the stack (@SP).
Status Flags:	None
Operation:	SP ← ++SP
Encoding:	15 0

f000	1101	ssss	ssss

Example(s):	PUSH GR ; GR=0F3Fh
	PUSH #40h

Stack Data:

xxxxh
0040h
0F3Fh
xxxxh
xxxxh

0040h ← SP (after PUSH #40h)
0F3Fh ← SP (after PUSH GR)
xxxxh ← SP (initial)

RET   **Return from Subroutine**

Description: RET pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). The decremented SP is saved as the new stack pointer (SP).

Status Flags: None

Operation: IP ← @ SP--

Encoding: 15   0

1000	1100	0000	1101

Example(s):   RET

Code Execution:

Addr (IP)	Op Code
0311h	…
0312h	RET
0103h	…

Stack Data:

xxxxh
xxxxh
0103h	← SP (before RET)
xxxxh	← SP (after RET)
xxxxh
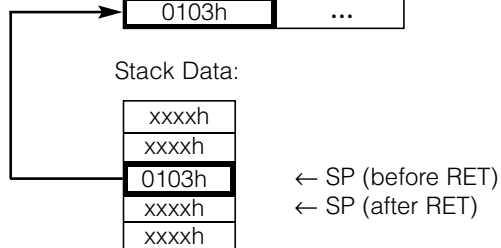
RET C/RET NC   **Conditional Return on Status Flag**
RET Z/RET NZ
RET S

Description: Performs conditional return (RET) based upon the state of a specific processor status flag. RET C returns if the Carry flag is set while RET NC returns if the Carry flag is clear. RET Z returns if the Zero flag is set while RET NZ returns if the Zero flag is clear. RET S returns if the Sign flag is set. See RET for additional information on the return operation.

Status Flags: None

RET C   C=1: IP ← @SP--

Operation: C=0: IP ← IP + 1

Encoding: 15   0

1010	1100	0000	1101

Example(s):   RET C   ; C=1, return (RET) is performed

RET NC

Operation:	C=0: IP ← @SP--
	C=1: IP ← IP +1
Encoding:	15 0

1110	1100	0000	1101

Example(s):   RET NC   ; C=1, return (RET) does not occur

RET Z

Operation:	Z=1: IP ← @SP--
	Z=0: IP ← IP + 1
Encoding:	15 0

1001	1100	0000	1101

Example(s):   RET Z   ; Z=0, return (RET) does not occur

RET NZ

Operation:	Z=0: IP ← @SP--
	Z=1: IP ← IP +1
Encoding:	15 0

1101	1100	0000	1101

Example(s):   RET NZ   ; Z=0, return (RET) is performed

RET S

Operation:	S=1: IP ← @SP--
	S=0: IP ← IP + 1
Encoding:	15 0

1100	1100	0000	1101

Example(s):   RET S   ; S=0, return (RET) does not occur

MAXQ7665/MAXQ7666 User's Guide

RETI	Return from Interrupt

Description: RETI pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). Additionally, RETI returns the interrupt logic to a state in which it can acknowledge additional interrupts.

Status Flags: None

Operation: IP ← @SP--

INS ← 0

Encoding:

15   0

1000	1100	1000	1101

Example(s): See RETI

RETI C/RETI NC	Conditional Return from Interrupt on Status Flag
RETI Z/RETI NZ	
RETI S	

Description: Performs conditional return (RETI) based upon the state of a specific processor status flag. RETI C returns if the Carry flag is set while RETI NC returns if the Carry flag is clear. RETI Z returns if the Zero flag is set while RETI NZ returns if the Zero flag is clear. RETI S returns if the Sign flag is set. See RETI for additional information on the return operation.

Status Flags: None

RETI C

Operation: C=1: IP ← @SP--

INS ← 0

C=0: IP ← IP + 1

Encoding:

15   0

1010	1100	1000	1101

Example(s): RETI C   ; C=1, return from interrupt (RETI) is performed

---

RETI NC

Operation: C=0: IP ← @SP--

INS ← 0

C=1: IP ← IP +1

Encoding:

15   0

1110	1100	1000	1101

Example(s): RETI NC   ; C=1, return from interrupt (RETI) does not occur

RETI Z

Operation:	Z=1: IP ← @SP--
	INS ← 0
	Z=0: IP ← IP + 1

Encoding: 15   0

1001	1100	1000	1101

Example(s):   RETI Z   ; Z=0, return from interrupt (RETI) does not occur

---

RETI NZ

Operation:	Z=0: IP ← @SP--
	INS ← 0
	Z=1: IP ← IP +1

Encoding: 15   0

1101	1100	1000	1101

Example(s):   RETI NZ   ; Z=0, return from interrupt (RETI) is performed

---

RETI S

Operation:	S=1: IP ← @SP--
	INS ← 0
	S=0: IP ← IP + 1

Encoding: 15   0

1100	1100	1000	1101

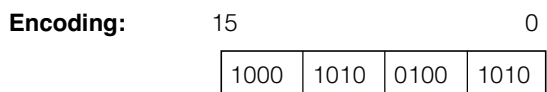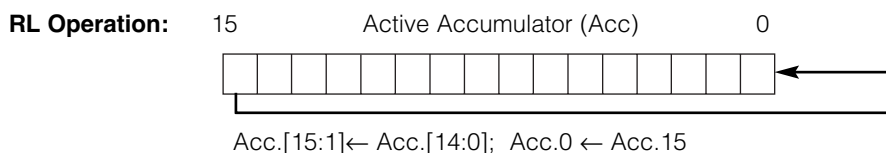**Example(s):**   RETI S   ; S=0, return from interrupt (RETI) does not occur
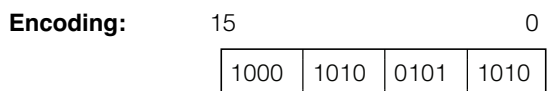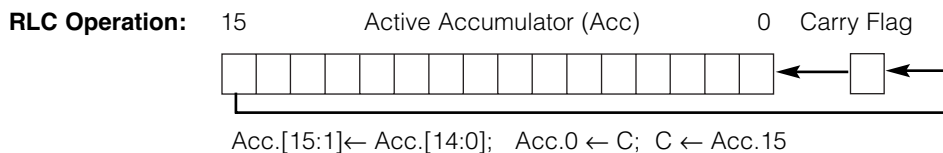
RL/RLC	**Rotate Left Accumulator**
	Carry Flag (Ex/In)clusive

Description: Rotates the active accumulator left by a single bit position. The RL instruction circulates the msb of the accumulator (bit 15) back to the lsb (bit 0) while the RLC instruction includes the Carry (C) flag in the circular left shift.

Status Flags: C (for RLC only), S, Z (for RLC only)

RL Operation: 15   Active Accumulator (Acc)   0

Acc.[15:1]← Acc.[14:0]; Acc.0 ← Acc.15

Encoding: 15   0

1000	1010	0100	1010

Example(s):

```
                              ; Acc = A345h, S=1, Z=0
RL                            ; Acc = 468Bh, S=0, Z=0
RL                            ; Acc = 8D16h, S=1, Z=0
```

RLC Operation: 15   Active Accumulator (Acc)   0   Carry Flag

Acc.[15:1]← Acc.[14:0]; Acc.0 ← C; C ← Acc.15

Encoding: 15   0

1000	1010	0101	1010

Example(s):

```
                              ; Acc = A345h, C=1, S=1, Z=0
RLC                           ; Acc = 468Bh, C=1, S=0, Z=0
RLC                           ; Acc = 8D17h, C=0, S=1, Z=0
```

RR/RRC	**Rotate Right Accumulator** **Carry Flag (Ex/In)clusive**

Description: Rotates the active accumulator right by a single bit position. The RR instruction circulates the lsb of the accumulator (bit 0) back to the msb (bit 15) while the RRC instruction includes the Carry (C) flag in the circular right shift.

Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation: 15   Active Accumulator (Acc)   0

Acc.[14:0]← Acc.[15:1]; Acc.15 ← Acc.0

Encoding: 15   0

1000	1010	1100	1010

Example(s):   ;Acc = A345h, S=1, Z=0

RR   ; Acc = D1A2h, S=1, Z=0

RR   ; Acc = 68D1h, S=0, Z=0

RRC Operation: 15   Active Acc (Acc)   0   Carry Flag

Acc.[14:0]← Acc.[15:1];   Acc.15 ← C; C ← Acc.0

Encoding: 15   0

1000	1010	1101	1010

Example(s):   ; Acc = A345h, C=1, S=1, Z=0

RRC   ; Acc = D1A2h, C=1, S=1, Z=0

RRC   ; Acc = E8D1h, C=0, S=1, Z=0

MAXQ7665/MAXQ7666 User's Guide

SLA/SLA2/SLA4	**Shift Accumulator Left Arithmetically** **One, Two, or Four Times**

Description:	Shifts the active accumulator left once, twice, or four times respectively for SLA, SLA2, and SLA4. For each shift iteration, a 0 is shifted into the lsb, and the msb is shifted into the Carry (C) flag. For signed data, this shifting process effectively retains the sign orientation of the data to the point at which overflow/underflow would occur.
Status Flags:	C, S, Z

SLA Operation: Carry Flag 15   Active Accumulator (Acc)   0

$C \leftarrow Acc.15$; $Acc.[15:1] \leftarrow Acc.[14:0]$; $Acc.0 \leftarrow 0$

Encoding: 15   0

1000	1010	0010	1010

Example(s):   ; Acc = E345h, C=0, S=1, Z=0

SLA   ; Acc = C68h, C=1, S=1, Z=0

SLA   ; Acc = 8D14h, C=1, S=1, Z=0

SLA2 Operation: Carry Flag 15   Active Accumulator (Acc)   0

$C \leftarrow Acc.14$; $Acc.[15:2] \leftarrow Acc.[13:0]$; $Acc.[1:0] \leftarrow 0$

Encoding: 15   0

1000	1010	0011	1010

Example(s):   ; Acc = E345h, C=0, S=1, Z=0

SLA2   ; Acc = 8D14h, C=1, S=1, Z=0

SLA4 Operation: Carry Flag 15   Active Accumulator (Acc)   0

$C \leftarrow Acc.12$; $Acc.[15:4] \leftarrow Acc.[11:0]$; $Acc.[3:0] \leftarrow 0$

Encoding: 15   0

1000	1010	0110	1010

Example(s):   ; Acc = E345h, C=0, S=1, Z=0

SLA4   ; Acc = 3450h, C=0, S=0, Z=0

SR/SRA/SRA2/SRA4	**Shift Accumulator Right/** **Shift Accumulator Right Arithmetically** **One, Two, or Four Times**

Description: Shifts the active accumulator right once for the SR, SRA instructions and 2 or 4 times, respectively, for the SRA2, SRA4 instructions. The SR instruction shifts a 0 into the accumulator msb while the SRA, SRA2, and SRA4 instructions effectively shift a copy of the current msb into the accumulator, thereby preserving any sign orientation. For each shift iteration, the accumulator lsb is shifted into the Carry (C) flag.

Status Flags: C, S (changes for SR only), Z

SR Operation:

Acc.15 ← 0; Acc.[14:0]← Acc.[15:1]; C ← Acc.0

Encoding:

15			0
1000	1010	1010	1010

Example(s):

	; Acc = A345h, C=1, S=1, Z=0
SR	; Acc = 51A2h, C=1, S=0, Z=0
SR	; Acc = 28D1h, C=0, S=0, Z=0

SRA Operation:
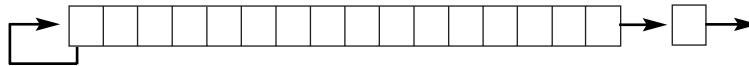
Acc.[14:0]← Acc.[15:1]

Acc.15 ← Acc.15

C ← Acc.0

Encoding:

15			0
1000	1010	1111	1010

Example(s):

	; Acc = 0003h, C=0, Z=0
SRA	; Acc = 0001h, C=1, Z=0
SRA	; Acc = 0000h, C=1, Z=1

SRA2 Operation:
15   Active Accumulator (Acc)   0   Carry Flag
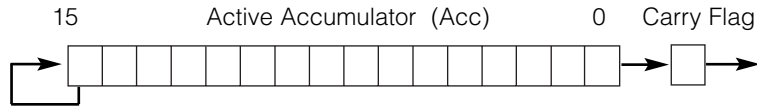
Acc.[13:0] ← Acc.[15:2]

Acc.[15:14] ← Acc.15

C ← Acc.1

Encoding: 15   0

| 1000 | 1010 | 1110 | 1010 |
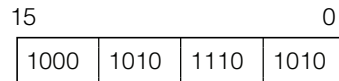
Example(s):
```
                            ; Acc = 0003h, C=0, Z=0
SRA2                        ; Acc = 0000h, C=1, Z=1
```

SRA4 Operation:
15   Active Accumulator (Acc)   0   Carry Flag
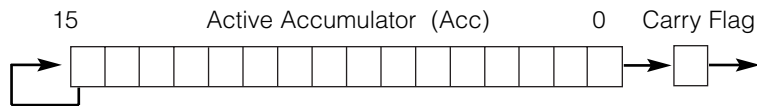
Acc.[11:0] ← Acc.[15:4]

Acc.[15:12] ← Acc.15

C ← Acc.3

Encoding: 15   0

| 1000 | 1010 | 1011 | 1010 |
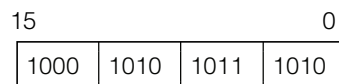
Example(s):
```
                            ; Acc = 9878h, C=0, Z=0
SRA4                        ; Acc = F987h, C=1, Z=0
SRA4                        ; Acc = FF98h, C=0, Z=0
```

SUB/SUBB *src*	Subtract /Subtract with Borrow

Description: Subtracts the specified src from the active accumulator (Acc) and returns the result back to the active accumulator. The SUBB additionally subtracts the borrow (Carry Flag), which may have resulted from previous subtraction. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: C, S, Z, OV

SUB Operation: Acc ← Acc - src

Encoding:

15   0

f101	1010	ssss	ssss

Example(s):

```
                              ; Acc = 2345h to start, A[1]= 1250h
SUB    A[1]                   ; Acc = 10F5h, C=0, S=0, Z=0, OV=0
SUB    A[1]                   ; Acc = FEA5h, C=1, S=1, Z=0, OV=0
SUB    A[2]                   ; A[2] =7FFFh
                              ; → Acc = 7EA6h; C=0, S=0, Z=0, OV=1
```

SUBB Operation: Acc ← Acc - (src + C)

Encoding:

15   0

f111	1010	ssss	ssss

Example(s):

```
                              ; Acc = 2345h, A[1]= 1250h, C=1
SUBB   A[1]                   ; Acc = 10F4h, C=0, S=0, Z=0
SUBB   A[1]                   ; Acc = FEA4h, C=1, S=1, Z=0
```

Special Notes: The active accumulator (Acc) is not allowed as the src for these operations.

XCH	**Exchange Accumulator Bytes**

Description:	Exchanges the upper and lower bytes of the active accumulator.
Status Flags:	S
Operation:	Acc.[15:8] ← Acc.[7:0]
	Acc.[7:0] ← Acc.[15:8]

Encoding:

```
15                          0
```

1000	1010	1000	1010

Example(s):		; Acc = 2345h
	XCHN	; Acc = 4523h

XCHN	**Exchange Accumulator Nibbles**

Description:	Exchanges the upper and lower nibbles in the active accumulator byte(s).
Status Flags:	S
Operation:	Acc.[7:4] ← Acc.[3:0]
	Acc.[3:0] ← Acc.[7:4]
	Acc.[15:12] ← Acc.[11:8]
	Acc.[11:8] ← Acc.[15:12]

Encoding:

```
15                          0
```

1000	1010	0111	1010

Example(s):		; Acc = 2345h
	XCHN	; Acc = 3254h

XOR src	**Logical XOR**

Description: Performs a logical-XOR between the active accumulator (Acc or A[AP]) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ7665/MAXQ7666 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: S, Z

Operation: Acc ← Acc XOR src

Encoding:
15   0

f011	1010	ssss	ssss

Example(s):
```
                          ; Acc = 2345h
XOR A[2]                  ; A[2]=0F0Fh; Acc ← 2C4Ah
```

Special Notes: The active accumulator (Acc) is not allowed as the src for this operation.

XOR Acc.<b>	**Logical XOR Carry Flag with Accumulator Bit**

Description: Performs a logical-XOR between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.

Status Flags: C

Operation: C ← C XOR Acc.<b>

Encoding:
15   0

1011	1010	bbbb	1010

Example(s):
```
                          ; Acc = 2345h, C=1 at start
XOR Acc.1                 ; Acc.1=0 → C=1
XOR Acc.2                 ; Acc.2=1 → C=0
```

SECTION 15: UTILITY ROM (SPECIFIC TO MAXQ7665A–MAXQ7665D WITH TYPE A FLASH)

This section contains the following information:

LIST OF TABLES

SECTION 15: UTILITY ROM (SPECIFIC TO MAXQ7665A–MAXQ7665D WITH TYPE A FLASH)

The MAXQ7665 utility ROM includes routines that provide the following functions to application software:

- In-application programming routines for Type A flash memory (program, erase, mass erase)
- Single word/byte copy and buffer copy routines for use with lookup tables
- Temperature conversion routine to perform internal/remote diode-connected transistor based temperature measurement

The MAXQ7665 flash type is identified by bit 4 of the read-only OTP register (Module 5, Index 0Dh). For devices with Type A flash, OTP register bit 4 (OTP.4) = 1. In the MAXQ7665x family, only MAXQ7665A–MAXQ7665D devices support Type A flash.

To provide backward compatibility among different versions of the utility ROM, a function address table is included that contains the entry points for all user-callable functions. With this table, user code can determine the entry point for a given function as follows:

1) Read the location of the function address table from address 0800Dh in the utility ROM.

2) The entry points for each function listed in Table 15-1 are contained in the function address table, one word per function, in the order given by their function numbers.

For example, the entry point for the **flashEraseAll** function can be accessed and called by the following procedure.

```
get_urom_table_entry:
 move dpc, #1Ch            //all data pointers in word mode
 move dp[ 0] , #0800Dh     //initialize dp[ 0]
 move bp, @dp[ 0]          //load function address table location in bp
 move offs, #2             //load function number in offs
 call @bp[ offs]           //call flashEraseAll
```

It is also possible to call utility ROM functions directly, using the entry points given in Table 15-1. Standard include files are provided for this purpose with the MAXQ7665 development tool set. This method calls functions more quickly, but the application may need to be recompiled in order to run properly with a different version of the utility ROM.

Table 15-1. Utility ROM User Functions (for Utility ROM Version 1.01)

FUNCTION NUMBER	FUNCTION NAME	ENTRY POINT	SUMMARY
0	—	—	Reserved.
1	flashEraseSector	08836h	Erases (programs to FFFFh) a sector of flash memory.
2	flashEraseAll	0885Dh	Erases (programs to FFFFh) all flash memory.
3	moveDP0	0886Bh	Reads a byte/word at DP[0].
4	moveDP0inc	0886Eh	Reads a byte/word at DP[0], then increments DP[0].
5	moveDP0dec	08871h	Reads a byte/word at DP[0], then decrements DP[0].
6	moveDP1	08874h	Reads a byte/word at DP[1].
7	moveDP1inc	08877h	Reads a byte/word at DP[1], then increments DP[1].
8	moveDP1dec	0887Ah	Reads a byte/word at DP[1], then decrements DP[1].
9	moveFP	0887Dh	Reads a byte/word at BP[OFFS].
10	moveFPinc	08880h	Reads a byte/word at BP[OFFS], then increments OFFS.
11	moveFPdec	08883h	Reads a byte/word at BP[OFFS], then decrements OFFS.
12	copyBuffer	08886h	Copies LC[0] values from DP[0] to BP[OFFS].
13	tempConv	0888Ch	Performs temperature conversion.
14	flashWriteA	088DBh	Writes a word to the flash memory.

MAXQ7665/MAXQ7666 User's Guide

15.1 In-Application Programming Functions

Function: flashEraseSector
Summary: Erases (programs to FFFFh) a sector of flash memory.
Inputs: A[0]: Word address located in the sector to be erased.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
  1: Failure due to software timeout.
  2: Failure reported by hardware (FERR).
  3: Failure due to trying to erase current page.
Destroys: PSF, LC[1], GR, ACC, AP, APC (AP = APC = 0)

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) When calling this function from flash, care should be taken that the return address is not in the sector that is being erased.

Function: flashEraseAll
Summary: Erases (programs to FFFFh) all locations in flash memory.
Inputs: None.
Outputs: Carry: Set on error and cleared on success.
Destroys: PSF, GR, LC[0], LC[1], APC, AP, A[0] (AP, APC set to 0)

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) This function can only be called by code running from the RAM. An attempt to call this function while running from the flash results in an error.

Function: flashWriteA
Summary: Writes a single word to the flash memory.
Inputs: A[0]: Word address in code flash memory to write to.
  A[1]: Word value to write to flash memory.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
  1: Failure due to software timeout.
  2: Failure reported by hardware (DQ5).
  4: Command not supported.
Destroys: LC[1], A[2], AP

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) If the flash location has already been programmed to a non-FFFF value, this function returns with an error (Carry set). To reprogram a flash sector, it must first be erased by calling flashEraseSector or flashEraseAll.

15.2 Data Transfer Functions

Function: moveDP0
Summary: Reads the byte/word value pointed to by DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP0inc
Summary: Reads the byte/word value pointed to by DP[0], then increments DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
DP[0] is incremented.
Destroys: Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP0dec
Summary: Reads the byte/word value pointed to by DP[0], then decrements DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
DP[0] is decremented.
Destroys: Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP1
Summary: Reads the byte/word value pointed to by DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: Selects DP[1] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP1inc
Summary: Reads the byte/word value pointed to by DP[1], then increments DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
DP[1] is incremented.
Destroys: Selects DP[1] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP1dec
Summary: Reads the byte/word value pointed to by DP[1], then decrements DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
DP[1] is decremented.
Destroys: Selects DP[1] in DPC.
Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFP
Summary: Reads the byte/word value pointed to by BP[OFFS].
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: Selects BP in DPC.
Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFPinc
Summary: Reads the byte/word value pointed to by BP[OFFS] then increments OFFS.
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
OFFS is incremented.
Destroys: Selects BP in DPC.
Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFPdec
Summary: Reads the byte/word value pointed to by BP[OFFS] then decrements OFFS.
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
OFFS is decremented.
Destroys: Selects BP in DPC.
Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function:	copyBuffer
Summary:	Copies LC[0] bytes/words from DP[0] to BP[OFFS].
Inputs:	DP[0]: Address to copy from.
	BP[OFFS]: Address to copy to.
	LC[0]: Number of bytes or words to copy.
Outputs:	OFFS is incremented by LC[0].
	DP[0] is incremented by LC[0].
Destroys:	LC[0]

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] and BP[OFFS] for byte or word mode. Both DP[0] and BP[OFFS] should be configured to the same mode (byte or word) for correct buffer copying.

2) The addresses passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointers before reading the byte/word values.

15.3 Temperature Conversion Function

Function:	tempConv
Summary:	Performs internal/remote diode-connected transistor-based temperature measurements.
Inputs:	GR: Sensor source (SS) code.
	A[0]: Single-ended or differential configuration, SEDIF. Set to 1 for differential or 0 for singled-ended.
Outputs:	GR: Temperature conversion result in degree Celsius.
	Carry: Set on error or illegal SS code.
Destroys:	A[0], A[1], PSW; AP, APC set to 0

Notes:

1) Valid sensor source (SS) code identifiers:

SS = 0x00 Internal diode-connected transistor-based temperature measurement.

SS = 0x01 Remote diode-connected transistor-based temperature measurement on AIN0.

SS = 0x02 Remote diode-connected transistor-based temperature measurement on AIN2.

MAXQ7665/MAXQ7666 User's Guide

15.4 ROM Example 1: Calling A MAXQ7665 Utility ROM Function Directly

This example shows the direct addressing method for calling MAXQ7665 utility functions, using the function **moveDP1inc** to read a static string from code space. Note the equate UROM_MOVEDP1INC.

```
UROM_MOVEDP1INC EQU 08877h

   Text:
      DB   "Hello World!",0          ; Define a string in code space.

      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ;; Function:      PrintText
      ;; Description:   Prints the string stored at the "Text" label.
      ;; Returns:       N/A
      ;; Destroys:      ACC, DP[1], DP[0], and GR.
      ;; Notes:         This function assumes that DP[0] is set to word mode, and
      ;;                DP[1] is in byte mode.
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   PrintText:
      move  DP[1], #Text             ; Point to the string to display.
      move  ACC, DP[1]               ; "Text" is a word address and we need a
      sla                            ; byte address, so shift left 1 bit.
      or    #08000h                  ; Code space is mapped to 8000h when running
      move  DP[1], ACC               ; from the ROM, so the address must be masked.

   PrintText_Loop:
      call  UROM_MOVEDP1INC          ; Fetch the byte from code space.
      move  ACC, GR
      jump  Z, PrintText_Done        ; Reached the null terminator.
      call  PrintChar                ; Call a routine to output the char in ACC
      jump  PrintText_Loop           ; Process the next byte.

   PrintText_Done:
      ret
```

15.5 ROM Example 2: Calling A MAXQ7665 Utility ROM Function Indirectly

The second example shows the indirect addressing method (lookup table) for calling MAXQ7665 utility functions. We use the same function (**UROM_MoveDP1Inc**) to read our static string, but this time we must figure out the address we want dynamically. Note the inserted code where we before had a direct call to the function. Also note that the function index of **moveDP1inc** is 7.

```
Text:
    DB  "Hello World!",0          ; Define a string in code space.

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Function:      PrintText
    ;; Description:   Prints the string stored at the "Text" label.
    ;; Returns:       N/A
    ;; Destroys:      ACC, DP[1], DP[0], and GR.
    ;; Notes:         This function assumes that DP[0] is set to word mode, and
    ;;                DP[1] is in byte mode.
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PrintText:
    move  DP[1], #Text            ; Point to the string to display.
    move  ACC, DP[1]             ; "Text" is a word address and we need a
    sla                          ; byte address, so shift left 1 bit.
    or    #08000h                ; Code space is mapped to 8000h when running
    move  DP[1], ACC             ; from the ROM, so the address must be masked.

PrintText_Loop:
    ;
    ; Fetch the byte from code space.
    ;
    move  DP[0], #0800Dh         ; This is where the address of the table is stored.
    move  ACC, @DP[0]            ; Get the location of the function table.
    add   #7                     ; Add the index to the moveDP1inc function.
    move  DP[0], ACC             ; Point to where the address of moveDP1 is stored.
    move  ACC, @DP[0]            ; Retrieve the address of the function.
    call  ACC                    ; Execute the function.

    move  ACC, GR
    jump  Z, PrintText_Done      ; Reached the null terminator.
    call  PrintChar              ; Call a routine to output the char in ACC
    jump  PrintText_Loop         ; Process the next byte.

PrintText_Done:
    ret
```

SECTION 16: UTILITY ROM (SPECIFIC TO MAXQ7666 WITH TYPE F FLASH)

This section contains the following information:

LIST OF TABLES

SECTION 16: UTILITY ROM (SPECIFIC TO MAXQ7666 WITH TYPE F FLASH)

The MAXQ7666 utility ROM includes routines that provide the following functions to application software:

- In-application programming routines for Type F flash memory (program, erase, mass erase)
- Single word/byte copy and buffer copy routines for use with lookup tables
- Temperature conversion routine to perform internal/remote diode-connected transistor based temperature measurement

The MAXQ7666 flash type is identified by bit 4 of the read-only OTP register (Module 5, Index 0Dh). For devices with Type F flash, OTP register bit 4 (OTP.4) = 0. Only the MAXQ7666 supports Type F flash.

To provide backward compatibility among different versions of the utility ROM, a function address table is included that contains the entry points for all user-callable functions. With this table, user code can determine the entry point for a given function as follows:

1) Read the location of the function address table from address 0800Dh in the utility ROM.
2) The entry points for each function listed in Table 16-1 are contained in the function address table, one word per function, in the order given by their function numbers.

For example, the entry point for the **programFlashEraseAll** function can be accessed and called by the following procedure.

```
get_urom_table_entry:
 move dpc, #1Ch          //all data pointers in word mode
 move dp[ 0] , #0800Dh   //initialize dp[ 0]
 move bp, @dp[ 0]        //load function address table location in bp
 move offs, #2           //load function number in offs
 call @bp[ offs]         //call flashEraseAll
```

It is also possible to call utility ROM functions directly, using the entry points given in Table 16-1. Standard include files are provided for this purpose with the MAXQ7666 development tool set. This method calls functions more quickly, but the application may need to be recompiled in order to run properly with a different version of the utility ROM.

Table 16-1. Utility ROM User Functions (for Utility ROM Version 1.01)

FUNCTION NUMBER	FUNCTION NAME	ENTRY POINT	SUMMARY
0	programFlashWritePage	0882Ah	Writes an entire 32-word/64-byte program flash page.
1	programFlashErasePage	0883Fh	Erases (programs to FFFFh) two pages of program flash.
2	programFlashEraseAll	08866h	Erases (programs to FFFFh) the entire program flash memory.
3	moveDP0	08874h	Reads a byte/word at DP[0]
4	moveDP0inc	08877h	Reads a byte/word at DP[0], then increments DP[0].
5	moveDP0dec	0887Ah	Reads a byte/word at DP[0], then decrements DP[0].
6	moveDP1	0887Dh	Reads a byte/word at DP[1].
7	moveDP1inc	08880h	Reads a byte/word at DP[1], then increments DP[1].
8	moveDP1dec	08883h	Reads a byte/word at DP[1], then decrements DP[1].
9	moveFP	08886h	Reads a byte/word at BP[OFFS].
10	moveFPinc	08889h	Reads a byte/word at BP[OFFS], then increments OFFS.
11	moveFPdec	0888Ch	Reads a byte/word at BP[OFFS], then decrements OFFS.
12	copyBuffer	0888Fh	Copies LC[0] values from DP[0] to BP[OFFS].
13	tempConv	08895h	Performs temperature conversion.
14	—	—	Reserved.
15	—	—	Reserved.

Table 16-1. Utility ROM User Functions (for Utility ROM Version 1.01) (continued)

FUNCTION NUMBER	FUNCTION NAME	ENTRY POINT	SUMMARY
16	dataFlashWrite	08906h	Writes a word to the data flash memory.
17	dataFlashWriteEven	08920h	Writes a word to the data flash memory even address.
18	dataFlashErasePage	08929h	Erases two pages of data flash.
19	—	—	Reserved.
20	dataFlashEraseAll	08937h	Erases the entire data flash.
21	dataFlashReadEven	0893Eh	Reads a word from the data flash memory even address.

16.1 In-Application Programming Functions

Function: programFlashWritePage
Summary: Writes an entire 32-word/64-byte program flash page.
Inputs: DP[0]: Word address in program flash page to write to.
DP[1]: Word address in data space pointing to the 32 words in SRAM that will be written into the program flash page. Word mode assumed.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
1: Failure due to software timeout.
2: Failure reported by hardware (FERR).
4: Command not supported.
Destroys: PSF, LC[1], ACC, AP

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) If the flash location has already been programmed to a non-FFFF value, this function returns with an error (Carry set). To reprogram a flash page, it must first be erased by calling programFlashErasePage or programFlashEraseAll.

Function: programFlashErasePage
Summary: Erases (programs to FFFFh) two pages (1 page = 32 words) of the program flash memory.
Inputs: A[0]: Word address located in the page to be erased. The specified even page and the next sequential odd page will be erased.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
1. Failure due to software timeout.
2. Failure reported by hardware (FERR).
3. Failure due to trying to erase current page.
Destroys: PSF, LC[1], GR, ACC, AP, APC (AP = APC = 0)

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) When calling this function from flash, care should be taken that the return address is not in the page that is being erased.

Function:	programFlashEraseAll
Summary:	Erases (programs to FFFFh) all locations in program flash memory.
Inputs:	None.
Outputs:	Carry: Set on error and cleared on success.
Destroys:	PSF, GR, LC[0], LC[1], APC, AP, A[0] (AP, APC set to 0)

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) This function can only be called by code running from the RAM. An attempt to call this function while running from the flash results in an error.

Function:	dataFlashWrite
Summary:	Write a word (2 bytes) to the data flash page (1 page = 1 word).
Inputs:	A[0]: Word address in data flash page to write to.
	A[1]: Word value (2 bytes) to write to data flash.
Outputs:	Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
	1. Failure due to software timeout.
	2. Failure reported by hardware (FERR).
	3. Failure due to trying to erase current page.
	4. Command not supported.
Destroys:	A[0], LC[1]

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) If the flash location has already been programmed to a non-FFFF value, this function returns with an error (Carry set). To reprogram a flash page, it must first be erased by calling dataFlashErasePage or dataFlashEraseAll.

3) The function "dataFlashWrite" has no entry in the utility ROM functions table and can be called only directly using the entry point (088F5h) given in Table 16-1.

Function:	dataFlashWriteEven
Summary:	Write a word (2 bytes) to an even address data flash page (1 page = 1 word).
Inputs:	A[0]: Word address in data flash page to write to. Addresses are translated ((A[0] << 1)	C000h) to map even address only. Only word addresses 0–3Fh are valid.
	A[1]: Word value (2 bytes) to write to data flash.
Outputs:	Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
	1. Failure due to software timeout.
	2. Failure reported by hardware (FERR).
	3. Failure due to invalid address.
	4. Command not supported.
Destroys:	A[0], LC[1], AP, APC

Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) If the flash location has already been programmed to a non-FFFF value, this function returns with an error (Carry set). To reprogram a flash page, it must first be erased by calling dataFlashErasePage or dataFlashEraseAll.

3) The even functions make it possible to work around the asymmetric "erase two, write one" page behavior by writing to or reading from only even addresses. By putting data only into even locations, the intrinsic two-page erase function is made to look like a single word erase at the cost of halving the available storage.

MAXQ7665/MAXQ7666 User's Guide

Function: dataFlashErasePage
Summary: Erases (programs to FFFFh) two pages (1 page = 1 word) of the data flash memory.
Inputs: A[0]: Word address located in the page to be erased. The specified even page and the next sequential odd page will be erased.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
    1. Failure due to software timeout.
    2. Failure reported by hardware (FERR).
    3. Failure due to invalid address.
    4. Command not supported.
Destroys: LC[1], DPC, ACC, APC, AP
Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) The location will be aligned to even address before erase.

Function: dataFlashEraseAll
Summary: Erases (programs to FFFFh) all locations in data flash memory.
Inputs: None.
Outputs: Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
    1. Failure due to software timeout.
    2. Failure reported by hardware (FERR).
    4. Command not supported.
Destroys: LC[0], LC[1], A[0]
Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

Function: dataFlashReadEven
Summary: Read a word (2 bytes) from an even address data flash page (1 page = 1 word).
Inputs: A[0]: Word address in data flash page to read from. Addresses are translated ((A[0] << 1) | C000h) to map even address only. Only word addresses 0–3Fh are valid.
Outputs: A[0]: Word value (2 bytes) read from data flash.
    Carry: Set on error and cleared on success. If set, then A[0] contains one of the following error codes:
    4. Command not supported.
Destroys: A[0], DP[0], DPC, APC
Notes:

1) If the watchdog reset is enabled, user code should disable it before calling this function. Also, disable interrupts globally (IGE = 0).

2) The even functions make it possible to work around the asymmetric "erase two, write one" page behavior by writing to or reading from only even addresses. By putting data only into even locations, the intrinsic two-page erase function is made to look like a single word erase at the cost of halving the available storage.

16.2 Data Transfer Functions

Function:	moveDP0
Summary:	Reads the byte/word value pointed to by DP[0].
Inputs:	DP[0]: Address to read from.
Outputs:	GR: Data byte/word read.
Destroys:	Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function:	moveDP0inc
Summary:	Reads the byte/word value pointed to by DP[0], then increments DP[0].
Inputs:	DP[0]: Address to read from.
Outputs:	GR: Data byte/word read.
	DP[0] is incremented.
Destroys:	Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function:	moveDP0dec
Summary:	Reads the byte/word value pointed to by DP[0], then decrements DP[0].
Inputs:	DP[0]: Address to read from.
Outputs:	GR: Data byte/word read.
	DP[0] is decremented.
Destroys:	Selects DP[0] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function:	moveDP1
Summary:	Reads the byte/word value pointed to by DP[1].
Inputs:	DP[1]: Address to read from.
Outputs:	GR: Data byte/word read.
Destroys:	Selects DP[1] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP1inc
Summary: Reads the byte/word value pointed to by DP[1], then increments DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
DP[1] is incremented.
Destroys: Selects DP[1] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveDP1dec
Summary: Reads the byte/word value pointed to by DP[1], then decrements DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
DP[1] is decremented.
Destroys: Selects DP[1] in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFP
Summary: Reads the byte/word value pointed to by BP[OFFS].
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: Selects BP in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFPinc
Summary: Reads the byte/word value pointed to by BP[OFFS] then increments OFFS.
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
OFFS is incremented.
Destroys: Selects BP in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: moveFPdec
Summary: Reads the byte/word value pointed to by BP[OFFS] then decrements OFFS.
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
OFFS is decremented.
Destroys: Selects BP in DPC.

Notes:

1) Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.

2) The address passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointer before reading the byte/word value.

Function: copyBuffer
Summary: Copies LC[0] bytes/words from DP[0] to BP[OFFS].
Inputs: DP[0]: Address to copy from.
BP[OFFS]: Address to copy to.
LC[0]: Number of bytes or words to copy.
Outputs: OFFS is incremented by LC[0].
DP[0] is incremented by LC[0].
Destroys: LC[0]

Notes:

1) Before calling this function, DPC should be set appropriately to configure DP[0] and BP[OFFS] for byte or word mode. Both DP[0] and BP[OFFS] should be configured to the same mode (byte or word) for correct buffer copying.

2) The addresses passed to this function should be based on the data memory mapping for the utility ROM, as explained in *Section 1*. The CDA0 and CDA1 bits must be set appropriately to access either the upper or lower half of program flash memory.

3) This function automatically refreshes the data pointers before reading the byte/word values.

16.3 Temperature Conversion Function

Function: tempConv
Summary: Performs internal/remote diode-connected transistor-based temperature measurements.
Inputs: GR: Sensor source (SS) code.
A[0]: Single-ended or differential configuration, SEDIF. Set to 1 for differential or 0 for singled-ended.
Outputs: GR: Temperature conversion result in degree Celsius.
Carry: Set on error or illegal SS code.
Destroys: A[0], A[1], PSW; AP, APC set to 0

Notes:

1) Valid sensor source (SS) code identifiers:

SS = 0x00 Internal diode-connected transistor-based temperature measurement.

SS = 0x01 Remote diode-connected transistor-based temperature measurement on AIN0.

SS = 0x02 Remote diode-connected transistor-based temperature measurement on AIN2.

MAXQ7665/MAXQ7666 User's Guide

REVISION HISTORY

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	12/07	Initial release.	—

maxim
integrated™

The Maxim logo and Maxim Integrated are trademarks of Maxim Integrated Products, Inc.