Keywords: I2C, MAX2990, powerline

APPLICATION NOTE 4649

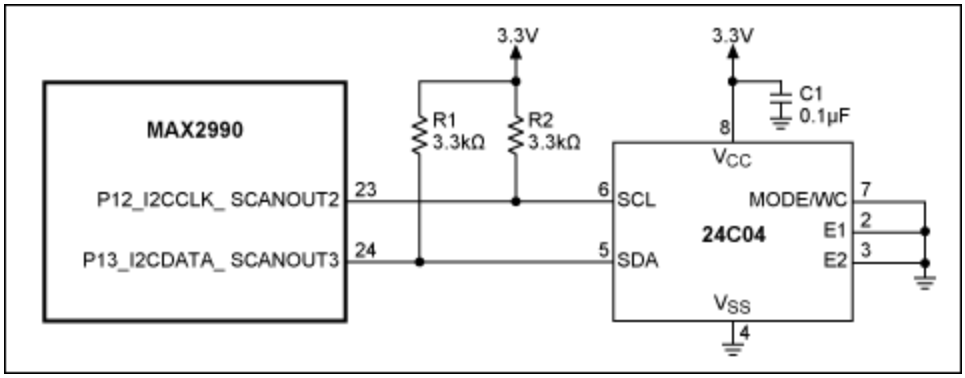
How to Write an Industry-Standard EEPROM (24C04) Using the MAX2990 I²C Interface

Aug 11, 2010

Abstract: Article and sample firmware code describe how to use the I²C interface on the MAX2990 power-line communications modem to interface with an external EEPROM 24C04.

Introduction

This application note and the sample firmware code describe how the I²C interface on MAX2990 power-line communications modem can be used to interface with an external EEPROM 24C04. The I²C bus is controlled by the MAX2990 (master) and the 24C04 EEPROM is the slave. The schematic below shows the hardware configuration used in this example.



Firmware Description

I²C Interface Initialization

Whenever the I²C module is enabled, SCL and SDA must be configured as open-drain. This configuration is necessary for the I²C communication to operate properly. Since the I²C is an alternate function for a GPIO port, firmware must ensure that the pullup on the SCL and SDA inputs is disabled (by writing a zero to that port controller output bit) during initialization.

The example has the 250kHz clock frequency. First you need to set up the I²C interface in the MAX2990 as shown below:

```
PO1_bit.Bit2 = 0;                // Disables the GPIO function of the
```

```
PO1_bit.Bit3 = 0;               // I2C pins

I2CCN_bit.I2CEN = 0;    // Makes sure that I2C is disabled
                        // to allow the changing of the I2C settings

I2CCN_bit.I2CMST = 1;           // Sets the I2C engine to master mode
I2CCN_bit.I2CEA = 0;            // 7-bit address mode
I2CCK_bit.I2CCKL = 0x40;        // 2µs CLK-low, to define I2C frequency
I2CCK_bit.I2CCKH = 0x40;        // 2µs CLK-high, to define I2C frequency

I2CTO = 200;            // I2C_TIMEOUT
I2CST = 0x400;          // Resets I2C status register

I2CCN_bit.I2CEN = 1;            // Enables the I2C engine
```

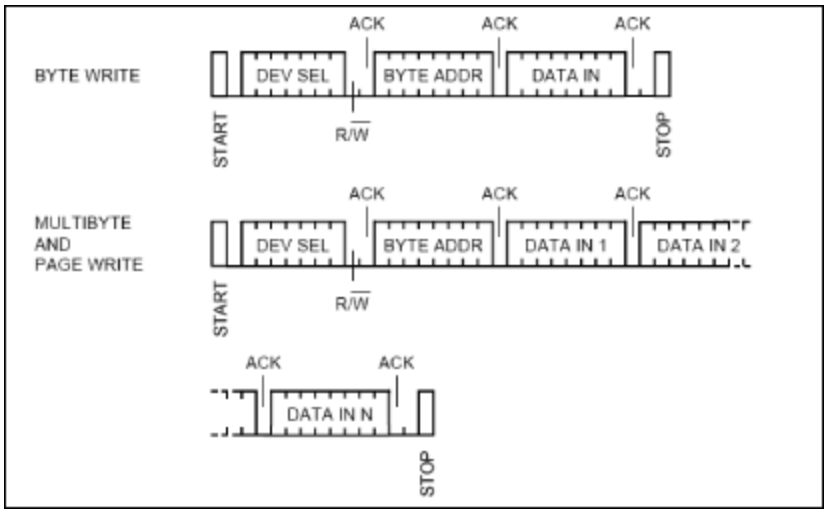
Write Mode

To write to a 24C04 EEPROM, you have to write the following bytes through the I²C interface:

1. Address of the I²C EEPROM (0xA0 in this example)
2. Address of the memory location in the EEPROM
3. Data bytes (address will increase automatically)

In this example we try to write the following bytes, starting from location 0x00, into the EEPROM: 0x12, 0x34, 0x56, 0x78, and 0x90.

```
i2c_init_write();       // Sets the MAX2990 I2C Engine into write mode
i2c_write(0x50);        // 24C04 write (adr = 0b1010 000 0) = 0xA0
                        // The MAX2990 I2C engine shifts the I2C address by
                        // 1 bit, because it will generate the R/W bit
                        // automatically

i2c_write(0x00);        // word address location
i2c_write(0x12);        // data1
i2c_write(0x34);        // data2
i2c_write(0x56);        // data3
i2c_write(0x78);        // data4
i2c_write(0x90);        // data5
I2C_STOP;               // Sends I2C stop-condition
```



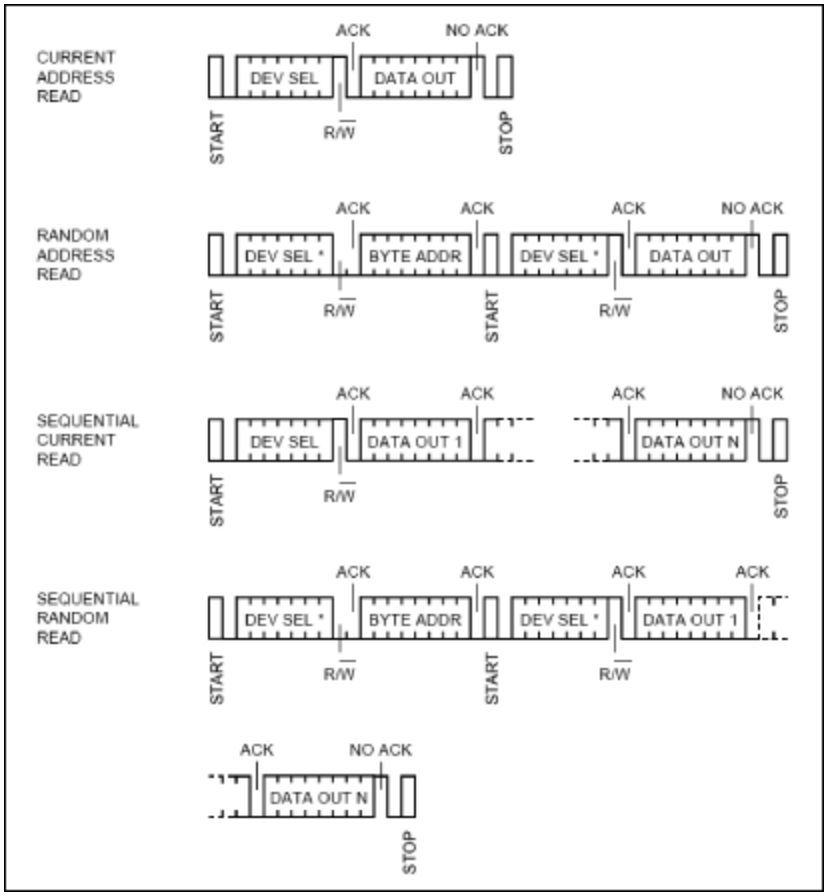
Read Mode

To read back the data, we wrote from the EEPROM. It is important that we give the 24C04 enough time to write. This typically takes several milliseconds after the "stop-condition." Consult the data sheet of your IC to make sure that you use the correct timing.

```
i2c_init_write();       // Sets the MAX2990 I2C engine into write mode
i2c_write(0x50);        // 24C04 write (adr = 0b1010 000 0) = 0xA0
                        // The MAX2990 I2C engine shifts the I2C address by
                        // 1 bit, because it will generate the R/W bit
                        // automatically

i2c_write(0x00);        // word address location

i2c_init_read();        // Sets the MAX2990 I2C engine into read mode

i2c_write(0x50);        // 24C04 read (adr = 0b1010 000 1) = 0xA1
                        // The MAX2990 I2C engine shifts the I2C address by
                        // 1 bit, because it will generate the R/W bit
                        // automatically

unsigned char data[5];              // Array to store the received data
i2c_read(data[0]);             // Reads 1 byte from I2C and writes it to the
array
i2c_read(data[1]);             // Reads 1 byte from I2C and writes it to the
array
i2c_read(data[2]);             // Reads 1 byte from I2C and writes it to the
array
i2c_read(data[3]);             // Reads 1 byte from I2C and writes it to the
array
i2c_read(data[4]);             // Reads 1 byte from I2C and writes it to the
array
I2C_STOP;                       // Sends I2C stop-condition
```

Now we examine the following functions which were used to read and write the EEPROM.

```
i2c_init_write(void)
i2c_init_read(void)
i2c_write(UINT8 data)
i2c_read(UINT8 *data)
```




```
void i2c_init_write(void)
{
I2CCN_bit.I2CMODE = 0;  // I2C transmit mode
I2CCN_bit.I2CACK = 1;   // Creates I2C NACK so that slave can create ACK
I2C_START;              // Generates I2C START condition
while( I2CCN_bit.I2CSTART == 1 );       // Waits until the START condition
                                        // was put to the I2C bus
I2CST_bit.I2CSRI = 0;                   // Resets the I2C interrupt flag
}

int i2c_init_read(void)
{
I2CCN_bit.I2CMODE = 1;  // I2C read-mode
I2CCN_bit.I2CACK = 0;   // Creates I2C ACK after receive
I2C_START;              // Generates I2C START condition

while( I2CCN_bit.I2CSTART == 1 );       // Waits until the START condition
I2CST_bit.I2CSRI = 0;                   // Resets the I2C interrupt flag
}

void i2c_write(UINT8 data)
{
I2CBUF = data;                          // Puts the data on the I2C bus
while( I2CST_bit.I2CTXI == 0 );         // Waits for transfer complete
I2CST_bit.I2CTXI = 0;                   // Resets the I2C transmit complete
                                        // interrupt flag
}

void i2c_read(UINT8 *data)
{
I2CBUF = 0xff;          // Puts "all ones" on the I2C bus so that slave can
```

```
pull
                        // the bus down to generate zeros

while( !I2CST_bit.I2CRXI );             // Waits for receive complete
I2CST_bit.I2CRXI=0;                     // Resets the I2C receive complete
                                        // interrupt flag

*data = I2CBUF;                         // Writes the data to the pointer
}
```

Related Parts		
MAX2990	10kHz to 490kHz OFDM-Based Power Line Communications Modem	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 4649: http://www.maximintegrated.com/an4649
APPLICATION NOTE 4649, AN4649, AN 4649, APP4649, Appnote4649, Appnote 4649