ANALOG DEVICES
Technical Notes on using Analog Devices' DSP components and development tools
Contact our technical support by phone: (800) ANALOG-D or e-mail: dsp.support@analog.com
Or visit our on-line resources http://www.analog.com/dsp and http://www.analog.com/dsp/EZAnswers

Using the PCI Interface On The ADSP-BF535 Blackfin® Processor in Host Mode

Contributed by Jorge Manguane

April 17, 2003

Introduction

The purpose of this document is to familiarize the user with the workings of the PCI interface on the ADSP-BF535 Blackfin® Processor.

The ADSP-BF535 Processor includes a Revision 2.2, 3.3V-compliant, 33 MHz/32-bit Peripheral Component Interconnect (PCI) bus interface. The interface can act as either a Host or a Device. In Host mode, an external arbiter is required. The ADSP-BF535 Processor can master the PCI bus in both Host and Device modes. In addition, a dedicated bus is available on chip to allow an external bus master to transfer data directly to internal (L2 memory) or external memory spaces of the ADSP-BF535.

This document describes how to program the PCI interface to function in Host mode.

Host mode

In host mode, the ADSP-BF535 PCI peripheral acts as a system controller of the PCI bus and, as such, it is responsible for configuring PCI devices attached to the bus as well as arbitrating mastership of the bus. Most PCI agents (host and/or devices) can master the PCI bus; a device can initiate a PCI transaction in the same way that a host can, which allows for peer to peer communications between different PCI agents. Essentially, what differentiates a device from a host in a system is who performs the configuration of the various PCI agents. Configuration entails allocating resources (memory ranges, interrupt lines) to the various devices. Once configured, devices will respond to transactions that fall within their allocated memory ranges.

Note: PCI bus arbitration is not built into the ADSP-BF535 PCI peripheral. An external PCI bus arbiter is required (See Momentum Data Systems Eagle-35 User Manual, Appendix B for example PLD equations).

Configuration Accesses

Before it can configure other devices, the PCI core must initialize its PCI space registers appropriately for host operation. At a minimum, it must do the following:

1. Set bit 0 of the *PCI Control Register (PCI_CTL)* register to enable host mode
2. Write to the *PCI Configuration Command Register (PCI_CFG_CMD)* to enable response to memory accesses, bus mastership
3. Write to the *PCI Host Memory Control Register(PCI_HMCTL)* to tell PCI agents what ADSP-BF535 resources can be accessed
4. Set bit 1 of the *PCI_CTL* register to enable PCI

To perform configuration accesses, the ADSP-BF535 PCI interface uses two resources:

1. *Configuration Base Address Pointer Register (PCI_CBAP)* is used to address the configuration space of the devices. The structure of the contents of this register depends on whether a Type 0 or Type 1 configuration access is being performed. If the device to be configured sits behind a PCI-to-PCI Bridge, Type 1 configuration accesses will be performed, otherwise, Type 0 accesses will be performed.
2. *Configuration Data Port* is a memory location (0xEEFF FFFC) that holds the data to be written to devices or returns data read from devices during configuration accesses.

When performing configuration cycles, the Host accesses a register set predefined by the PCI specification. The specification currently defines three Header formats:

- **Header Type Two:** defined for PCI-to-CardBus bridges
- **Header Type One:** defined for PCI-to-PCI bridges
- **Header Type Zero:** defined for all other devices.

The host must traverse some or all of the registers defined in the headers above. In this application note, we'll only touch on the last two header types (header type 0, and header type 1) since we are only concerned with configuring PCI devices that sit directly on the PCI bus (primary bus: Type 0 configuration) or devices that sit behind a PCI-to-PCI bridge (secondary bus: Type 1 configuration). Note when more than one PCI-to-PCI bridge is present on the system, the "secondary" bus can be anywhere from bus #1 to bus #255 in the bus hierarchy. Each PCI-to-PCI bridge adds a bus number to the system. For example, if there's only one bridge, devices on the other side of the bridge will reside on secondary bus #1, if there are two bridges, devices on the other side of the second bridge will reside on secondary bus #2, and so on. The bridges themselves may be on the same "primary" bus.

Alternatively, the primary bus of the second bridge may be the secondary bus of the first bridge, as shown in the figure below.

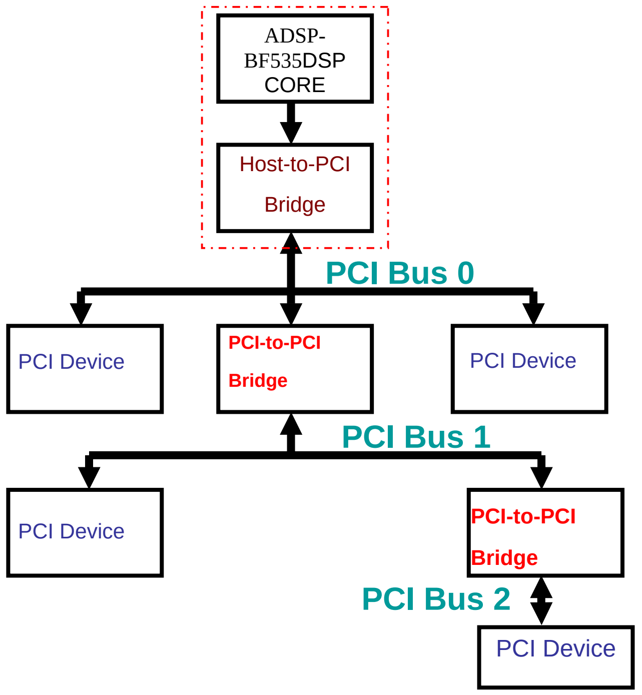

Figure 1

Type 0 Configuration Accesses

These accesses are directed to all PCI agents that reside on the primary bus, including devices and PCI-to-PCI bridges. The format of the data must be programmed into the PCI_CBAP register (Configuration Base Address Pointer register) as shown below.

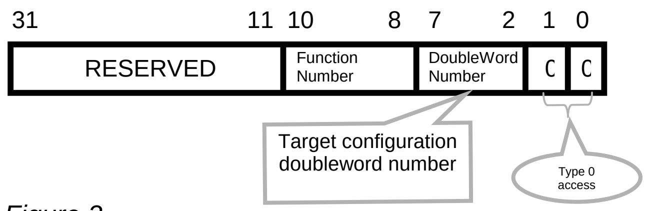

Figure 2

From the above figure,

AD[1:0] = 00 to indicate a type zero access

AD[7:2] indicate the target word to be accessed

AD[10:8] indicate the function number to be accessed within the targeted device.

AD[31:11] are reserved and can be used to implement IDSELs. IDSEL lines are used to select each of the various devices that may reside on the bus for configuration accesses. As an example, IDSEL of device 0 may be connected to AD11, IDSEL of device 1 may be connected to AD12, and so on. IDSELs are only meaningful during configuration. After configuration, during normal operation, devices decode and respond to PCI transactions that fall within their memory and I/O ranges as determined by what's in their respective Base Address Registers (BARs). In the case of the ADSP-BF535, these are PCI_CFG_MBAR and PCI_CFG_IBAR for memory and I/O respectively.

Here's an example that shows how to retrieve the device's Device and Vendor ID information:

```
P5.H = (PCI_CBAP >> 16);
P5.L = (PCI_CBAP & 0xFFFF);
R0.H = PCI_SLOT ;  R0.L=FUNCTION_NUMBER| 0x00;
 [P5] = r0; //load in PCI_CBAP the address of the 1st Dword of
            //the configuration space
 ssync;
 P3.H = (PCI_CONFIG_SPACE_PORT >> 16);
 P3.L = (PCI_CONFIG_SPACE_PORT & 0xFFFF);
 R0 = [P3]; // read Vendor ID and Device ID (1st Dword)
 ssync;
```

To access the next word (32 bits), increment the address in *PCI_CBAP* by 4 bytes.

To find out how much memory and/or I/O a particular device requires, the host writes *0xFFFFFFFF* to the device's BARx register and then reads this register back. Only the bits that the device unmasked will be written to. All others will return a value of zero when read back. The one's complement of the result read back will give the size in bytes of the given resource (memory or I/O).

```
R0.H = PCI_SLOT;
R0.L = FUNCTION_NUMBER| 0x10;
[P5] = r0; // access BAR0 – Memory BAR in the BF535
ssync;
r4 = 0xFFFFFFFF;
[P3] = R4; //write all F's to the Memory BAR register
SSYNC;
[P5] = r0;
SSYNC;
R4 = [P3]; // read back the unmasked (enabled) bits
ssync;
R4 = ~R4; // memory size requested by device [in bytes]
```

Now that we know the memory size this device requires, we can assign it its base address in the PCI Memory Space.

```
P0.H = PCI_Memory_Window >> 16;
P0.L = PCI_Memory_Window & 0xFFFF;
[P3] = P0; // assign Device the base of PCI MEM space
ssync;
```

In this case, the label *PCI_Memory_Window* corresponds to address 0xE000 0000, the base address of the PCI Memory Space. As can be seen from the PCI Memory Map in Figure 3, the PCI Memory Space is 128 Mbytes, and it can be partitioned among the various devices on the bus so that each device has its own unique PCI memory region.

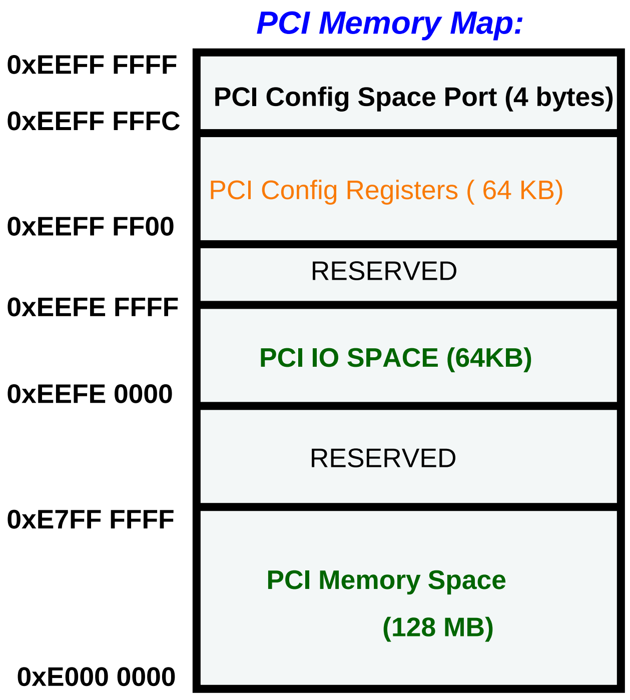

Figure 3

For the next device, we will have to assign it a memory base address offset by the memory size requested by the first device, and so on.

Type 1 Configuration accesses

If a device resides on a secondary bus (for example, behind a PCI-to-PCI bridge) Type 1 accesses will be required to configure the device.

In general, when a Type 1 transaction is seen on the bus, only PCI-to-PCI bridges claim the transaction, which then convert it to a Type 0 transaction and pass it on to devices that reside on their secondary buses. It is, of course, understood that only one bridge will claim a transaction at a given time, depending on whether the targeted device resides on its secondary bus (i.e., there is a Bus Number match). Before we get into more detail, let's review the data format of a Type 1 address:

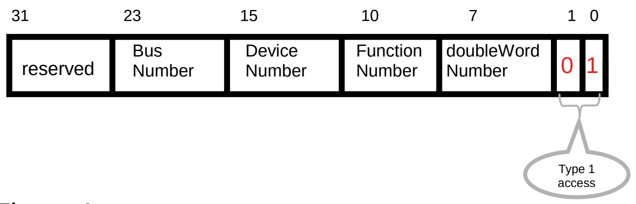

Figure 4

Lets focus on the case where the Bus Number does not correspond to the bridge's secondary bus number but is between the secondary bus number and the subordinate bus number (the very last bus number on the system). In this case, the bridge still claims the transaction and passes it on as a Type 1 transaction to be claimed by other bridges.

Before the devices behind a bridge are configured, the bridge itself must be configured with the Primary, Secondary and Subordinate bus number information, as well with the base address and ranges of the memory and I/O spaces. Bridges themselves are configured using Type 0 accesses.

As an example of how to configure a bridge, refer to Intel Corp's **21152 PCI-to-PCI Bridge Configuration Application Note (**Order Number: 278068-001).

Having crossed this bridge (configuring the PCI-to-PCI bridge), devices can be configured in much the same way as that discussed for Type 0 transactions, except that now the programmer must include the Bus Number and Device Number, as well as add a 1 to bit 0 of the address presented on the bus.

```
// Device ID | Vendor ID
 R0.H =  BUS_NUMBER;
 R0.L = DEVICE_NUMBER | FUNCTION_NUMBER| 0x01;
 [P5] = r0;
 ssync;
 P3.H = (PCI_CONFIG_SPACE_PORT >> 16);
 P3.L = (PCI_CONFIG_SPACE_PORT & 0xFFFF);
 R0 = [P3];
 ssync;
```

Now that the devices are configured, the host can access the devices by initiating transactions in the PCI memory or I/O regions that correspond to those regions that were assigned to the devices' BARs. To that end, the host uses *PCI Outbound Memory Base Address Register (PCI_MBAP)* for memory accesses and the *PCI Outbound I/0 Base Address Register (PCI_IBAP)* for I/O accesses.

Memory Accesses

To initiate a memory access, the host writes to the *PCI_MBAP* register the uppermost 5 bits (31:27) of the intended destination address. These 5 bits will be prefixed to the 27-bit offset of the actual address in the PCI memory space.

Example:

```
P4.H = (PCI_MBAP >> 16);
P4.L = (PCI_MBAP & 0xFFFF);
R0.H = 0xE000;
R0.L = 0x0000;
[P4] = R0;
ssync;
```

Now an access in the PCI memory space will initiate a PCI transaction:

```
P2.H = 0xEF00;
P2.L = 0x1234;
R0 = [P2];
ssync;
```

The upper 5 bits of the *PCI_MBAP* are: **11100**

The lower 27 bits of the address in P2 are:

111 0000 0000 0001 0010 0011 0100

The actual address that will be output on the bus will be:

1110 0111 0000 0000 0001 0010 0011 0100

which corresponds to: 0xE700 1234

This is not the intended address. To access 0xEF00 0000, *PCI_MBAP* should have been programmed with 0xE800 0000, which would produce the following upper 5 bits:

11101

Prefixing the above upper 5 bits to the lower 27 bits of the address in P2 would produce:

1110 1111 0000 0000 0001 0010 0011 0100

the intended 0xEF00 1234 address.

This address may correspond to different addresses within different devices. Each device has its own internal memory/(I/O) space mapping to the PCI space. As an example, the ADSP-BF535 acting as a device uses the *Target Inbound Memory Base Address Register (PCI_TMBAP)* to map the PCI memory space to its internal/external memory space.

If *PCI_TMBAP = 0xF000 0000*, the above address would correspond to *0xFF00 1234*, which falls within Reserved memory space. Note that the above example is for illustration purposes only.

I/O ACCESSES

The mechanism to perform I/O accesses is similar to that described for memory accesses except that now *PCI_IBAP* register is used. This register prefixes 16 bits to the actual address that will go on the bus.

INBOUND TRANSACTIONS

We just described transactions that are initiated by the host. As stated earlier, any PCI agent on the bus can be the bus master and the host can be the target, making it a slave for that particular transaction. In this case, the particular transaction is referred to as an ***inbound transaction.***

To participate in inbound transactions, the ADSP-BF535 as a host needs to enable accesses to its various resources, i.e., L2 memory, system MMR space, SDRAM space, and Asynchronous Memory space.

Note, however, that in host mode the ADSP-BF535 PCI core interface does not respond to I/O accesses, it only claims memory transactions. Therefore, the host's MMR space should be accessed as a *non-prefetchable* memory region. When PCI-to-PCI bridges are used, be sure to program their Non-Prefetchable Memory Base and Limit registers with a base address that corresponds to the ADSP-BF535's system MMR space.

To enable the various resources, the host uses the *PCI Host Memory Control Register (PCI_HMCTL). Refer to the ADSP-BF535 Blackfin Processor Hardware Reference Manual for bit definitions.*

```
HMCTL:
P4.H = PCI_HMCTL >> 16; P4.L = PCI_HMCTL & 0XFFFF;
R0.H = 0X0; R0.L = 0X3;
[P4] = R0; // allow access to L2 and system MMR spaces
SSYNC;
```

Now, the host can claim transactions that have the host's L2 memory and/or System MMR spaces as the target addresses.

References

ADSP-BF535 Blackfin Processor Hardware Reference , Rev. 1.0, November 2002, Analog Devices, Inc.

PCI SYSTEM ARCHITECTURE, 4th Edition, MindShare, Inc.

21152 PCI-to-PCI Bridge Configuration Application Note, INTEL

Document History

Version	Description
April 17, 2003 by J. Manguane	Initial Release