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 ADSP-BF535 Blackfin® processor's PCI interface in Device Mode

Contributed by Jorge Manguane

September 10, 2003

Introduction

The purpose of this document is to familiarize the user with the ADSP-BF535 Blackfin® Processor's Peripheral Component Interconnect (PCI) interface in Device mode. This document describes how to program the PCI interface to function in Device mode.

For the remainder of this document, "processor" will refer to "ADSP-BF535 Blackfin® processor."

The BF535 includes a 3.3V Revision 2.2-compliant, 33 MHz, 32-bit PCI bus interface. The interface can act as either a PCI Host or as a PCI Device. The BF535 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 processor.

Device Mode:

The processor's PCI interface supports a single-function PCI device and can be interfaced gluelessly to 3.3 Volt systems.

When a PCI device is plugged into a system, the PCI host needs to gather some information about the device that will allow the host to configure the device and load the proper device driver. Since this information exchange happens at power up or after system reset, the device needs to have this information stored in its boot code. The PCI specification rev. 2.2 stipulates that a device has up to 2^{25} PCI clock cycles to initialize its PCI configuration register space. The processor's PCI core interface will issue retries if a system host tries to access the device's configuration space before the processor has enabled the PCI interface (which is done by writing a 1 to the **"PCI Enable"** bit in the PCI Control Register (*PCI_CTL*)). It is, therefore, imperative not to set the **"PCI Enable"** bit until all configuration registers have been initialized.

An external PCI clock must be supplied to the BF535 PCI module before attempting to access the PCI Configuration registers since these reside in the PCI clock domain.

The ADSP-BF535 EZ-Kit Lite evaluation board does not bring out the PCI signals, and PCI Clock is not provided.

BF535 based PCI boards for both host and device applications are available from Momentum Data Systems at: http://www.mds.com

Figure 1. Type 0 PCI Configuration space Header region

Device ID		Vendor ID	
Status Register		Command Register	
Class Code			Revision ID
BIST	Header Type	Latency Timer	Cache Line Size
BAR 0 (Memory BAR)			
BAR 1			
BAR 2 (I/O BAR)			
BAR 3			
BAR 4			
BAR 5			
CardBus CIS Pointer			
Subsystem ID		Subsystem Vendor ID	
Expansion ROM Base Address			
Reserved			Capabilities Pointer
Reserved			
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line

Figure 1 shows the PCI configuration registers implemented on the BF535 PCI interface as specified by the PCI Local Bus Specification, Revision 2.2. This document should be consulted for a detailed description of these registers.

Cache Line Size and *Built-In Self Test (BIST) registers* are implemented but not supported by the processor. The remaining registers, shown shaded in gray, are not implemented.

This register space is referred to as the device's configuration header region or space. The format of the BF535 PCI header region is Header Type Zero, which is used for all devices other than PCI-to-PCI and CardBus bridges.

Listing 1, in the Appendix, shows an example program of how to initialize the PCI interface in device mode.

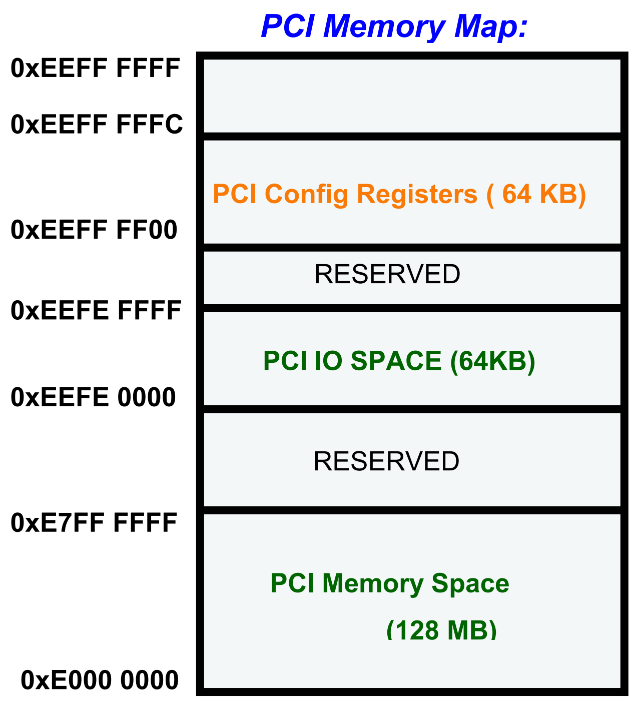
PCI Transaction Types:

There are three types of PCI transactions:

1. Configuration Accesses
2. Memory Accesses
3. I/O Accesses

In device mode the BF535 PCI core responds to all of the transaction types listed above. Note, however, that the processor's architecture does not define separate Configuration and I/O address spaces. The three PCI memory spaces are mapped into the processor's unified memory region. Refer to *Figure 2*, which shows a subset of the processor's external memory map.

Figure 2. PCI Address spaces



I/O accesses can be used to access the processor's non-prefetchable Memory Mapped Register (MMR) space.

The processor can initiate Memory and I/O accesses, but never Configuration accesses, while in Device mode. Configuration accesses are always initiated by the system host.

To respond to Configuration accesses, the processor need only initialize its configuration registers and then enable the PCI interface. After this, the host system can configure the PCI device. The host assigns to the device the following parameters:

- **The capabilities of the device**, such as Bus Mastership, Parity Error Response, Special Cycles, etc… This information is written into the processor's *PCI_CFG_CMD* register . Please refer to the processor's hardware reference manual for a description of this register and of the remaining registers mentioned below.
- **Latency timer**, the minimum amount of time the PCI device master can retain ownership of the bus when it initiates a transaction. This information is written into the processor's *PCI_CFG_MLT* register
- **Unique Memory and I/O addresses**. This information is written into the processor's

  PCI_CFG_MBAR and PCI_CFG_IBAR registers. The values in these registers will allow the device to respond, by asserting its Device Select line (#DEVSEL), to any accesses that fall within the range specified by the value in *PCI_CFG_xBAR* offset by the value in the corresponding mask register, *PCI_CFG_MBARM* or *PCI_CFG_IBARM* for memory and I/O, respectively.

  For example, if

  PCI_CFG_MBAR = 0xE000 0000 and

  PCI_CFG_MBARM = 0xFFFC 0000

  (which corresponds to a size of

  0x0003 FFFF+1 = 0x0004 0000)

  then the ADSP-BF535 PCI core will claim all transactions that fall in the range:

  0xE000 0000 to *0xE004 0000*

  In the above example, note that the host calculates the memory size required by the device by taking a 1's complement of the value in *PCI_CFG_MBARM* and adding a 1 to the result. Likewise, for I/O size the operation described above is also performed on the value in *PCI_CFG_IBARM* register.

- **Interrupt line** information written into the processor's *PCI_CFG_IL* register

Device is now ready to participate in PCI transactions as either a *target* or as an *initiator.*

OUTBOUND TRANSACTIONS

Outbound transactions are those transactions for which the BF535 PCI interface is the bus master. The following registers are used to perform outbound transactions:

PCI Outbound Memory Base Address Register (PCI_MBAP) for memory accesses

and the

PCI Outbound I/O Base Address Register (PCI_IBAP) for I/O accesses.

MEMORY ACCESSES:

To initiate a memory access, the device 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 = HI(PCI_MBAP );
P4.L = LO (PCI_MBAP );
R0.H = 0xE000;
R0.L = 0x0000;
[P4] = R0;
ssync;
```

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

```
P2.H = 0xE000;
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:

000 0000 0000 0001 0010 0011 0100

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

1110 0000 0000 0000 0001 0010 0011 0100

which corresponds to: 0xE000 1234

I/O ACCESSES

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

INBOUND TRANSACTIONS

Inbound transactions are those transactions for which the BF535 PCI interface is the bus target/slave.

To participate in inbound transactions, the processor's PCI interface should have bits 0 and 1 of the Command register (*PCI_CFG_CMD*) set to 1. Typically, in device mode the PCI host system writes to this register. These bits enable the device to participate in I/O and Memory transactions, respectively. Also, the registers: *PCI Device Memory BAR Mask (PCI_DMBARM)* and *PCI Device I/O BAR Mask (PCI_DIBARM)* should contain non zero values, as a value of zero in these registers also disables access to the respective spaces.

To map the incoming traffic into its memory regions the processor uses the following registers:

PCI Inbound Memory Base Address Register (PCI_TMBAP)

and

PCI Inbound I/O Base Address Register (PCI_TIBAP)

PCI_TMBAP and PCI_TIBAP specify the base addresses in the BF535's memory space for memory and I/O transactions, respectively.

Example:

Let's say the PCI host has assigned the BF535 PCI device the following hypothetical Base Address Register (BAR) values:

BAR 0 = 0xE000 0000 (memory BAR)

BAR 1 = 0xFFFE 0000 (I/O BAR)

(Note that the above BAR 0 and BAR 1 addresses are part of the PCI host's memory address map).

If *PCI_TMBAP = 0 (SDRAM space)* then all incoming memory transactions will be directed to the SDRAM memory space of the BF535 memory map.

For example, address *0xE000 0000* will map to address *0x0000 0000,* address *0xE000 1234* will map to address *0x0000 1234,* and so on.

Similarly, if *PCI_TIBAP = 0xFFC0 4000 (PCI MMR space)* then all incoming I/O transactions will be directed to this memory region.

The processor's PCI module cannot access L1 memory or the processor's core resources (core MMRs). Therefore, *PCI_TMBAP or PCI_TIBAP* should not point to L1 or to the core MMR space.

PCI INTERRUPTS

The processor's PCI module can generate PCI interrupts as well as respond to PCI interrupts. Of the four PCI interrupt lines (INTA – INTD), only INTA can be configured as either input or output. All others are inputs only.

The processor can also generate a PCI RESET interrupt. To generate either PCI RESET or INTA interrupts the processor first sets the pads as outputs by writing to the appropriate bits in the PCI Bridge Control Register (*PCI_CTL*). The processor then writes a 1 to the "**RST to PCI**" or the "**INTA to PCI**" bit of the PCI_CTL register.

To have the PCI module generate an interrupt to the processor's core, the PCI interrupt must be unmasked both at the core level (by writing to the *IMASK* register) and at the system level (by writing to the *SIC_IMASK* register). At the PCI level, the event that is going to cause the interrupt needs to be enabled in the PCI Interrupt Controller Register (*PCI_ICTL*) by setting the respective bit to a 1. Once the event occurs, the corresponding bit will be set in the PCI Status Register (*PCI_STAT*), and an interrupt to the core will be generated.

The bits in the PCI_STAT register are "sticky" and are Write 1 to Clear. To prevent the interrupt from being continuously generated, the particular bit(s) that caused the interrupt will need to be expressly cleared inside the interrupt service routine. This is done by writing a 1 to the specific bit, followed by issuing an SSYNC instruction.

References

ADSP-BF535 BLACKFIN® PROCESSOR Blackfin DSP Hardware Reference, Analog Devices, Inc.

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

Appendix

PCI_Device_Init.asm

```
/* ***************************************************************************
 *
 * Copyright (c) 2003 Analog Devices Inc.  All rights reserved.
 *
 * **************************************************************************/
#include "defBF535.h"


/* Helper macros
 * usage:
 *   P0.H = HI(UART_THR);
 *   P0.L = LO(UART_THR);
 */

#define LO(con32) ((con32) & 0xFFFF)
#define lo(con32) ((con32) & 0xFFFF)
#define HI(con32) (((con32) >> 16) & 0xFFFF)
#define hi(con32) (((con32) >> 16) & 0xFFFF)


.section program;
/**********************  PCI module Configuration ***************************/
pci_cfg_init:

Initialize:
    P4.H = HI(PCI_STAT); P4.L = LO(PCI_STAT);
    R1 = [P4];
    [P4] = R1;  // Clear PCI status register
    SSYNC;

    P4.H = HI(PCI_CTL); P4.L = LO(PCI_CTL);
    R0.H = 0x0; R0.L = 0x0;
    [P4] = R0;     // disable PCI before writing config space
    SSYNC;

    P4.H = HI(PCI_ICTL); P4.L = LO(PCI_ICTL);
    R0.H = 0x0; R0.L = 0x0;
    [P4] = R0;     // clear PCI interrupt enables before writing config space
    SSYNC;

/*************** BEGIN INITIALIZATION ***************************/
Vendor_ID:
    P4.H = HI(PCI_CFG_VIC); P4.L = LO(PCI_CFG_VIC);
```

```
    R0.H = 0x0; R0.L = 0x11D4;
    [P4] =


Device_ID:
    P4.H = HI(PCI_CFG_DIC); P4.
    R0.H = 0x0; R0.L = 0x1535;
    [P4] =



// in Devi
Command:
    P4.H = HI(PCI_CFG_CMD);
    R0.H = 0x0; R0.L
    [P4] =


Revision_ID:
    P4.H = HI(PCI_CFG_RID);
    R0.H = 0x0; R0.L
    [P4] =


Class_Code:
    P4.H = HI(PCI_CFG_CC); P4.L = L
    R0.H = 0x0011; R0.L = 0x8000;




// This function
Cache_Line_Size:
    P4.H = HI(PCI_CFG_CLS);
    R0.H = 0x0; R0.L
    [P4] =


// in Device mo
Latency_Timer:
    P4.H = HI(P
    R0 = 0x0;
    [P4] =


Header_Type:
    P4.H = HI(PCI_CFG_HT); P
    R0.H = 0x0; R0.L = 0x0;
    [P4] =


// This function is
Built_In_Self_Test:
    P4.H = HI(PCI_CFG_CLS);
    R0.H= 0x0; R0.L
    SSYNC;


Subsystem_Vendor_ID:
    P4.H = HI(PCI_CFG_SVID); P4.L = LO(PCI_CFG_SVID);
    R0.H = 0x0; R0.L = 0x0;
    [P4] = R0;
    SSYNC;

Subsystem_ID:
    P4.H = HI(PCI_CFG_SID); P4.L = LO(PCI_CFG_SID);
    R0.H= 0x0; R0.L = 0x0;
    [P4] = R0;
    SSYNC;


Interrupt_Pin:
    P4.H = HI(PCI_CFG_IP); P4.L = LO(PCI_CFG_IP);
    R0.H = 0x0;   R0.L = 0x1;/* INTA is used by ADSP-BF535 in device mode  */
    [P4] = R0;
    SSYNC;

// the host system will provide this information
Interrupt_Line:
    P4.H = HI(PCI_CFG_IL); P4.L = LO(PCI_CFG_IL);
    R0.H = 0x0;   R0.L = 0x0;
    [P4] = R0;
    SSYNC;

Minimum_Grant:
    P4.H = HI(PCI_CFG_MING); P4.L = LO(PCI_CFG_MING);
    R0.H = 0x0; R0.L = 0x1; //Refer to the PCI Local Bus Speficication, Revision 2.2
    [P4] = R0;

    SSYNC;

Maximum_Latency:
    P4.H = HI(PCI_CFG_MAXL); P4.L = LO(PCI_CFG_MAXL);
    R0.H = 0x0; R0.L = 0x2; //Refer to the PCI Local Bus Speficication, Revision 2.2
    [P4] = R0;
    SSYNC;

// DMBARM & DIBARM specify the size of memory and I/O requested by the device
Device_Memory_BAR_Mask:
    P4.H = HI(PCI_DMBARM); P4.L = LO(PCI_DMBARM);
    R0.H = 0xF800; R0.L = 0x0000;
    [P4] = R0;     // specify the memory size required by the ADSP-BF535 PCI
    SSYNC;

Device_IO_BAR_Mask:
    P4.H = HI(PCI_DIBARM); P4.L = LO(PCI_DIBARM);
    R0.H = 0xFFFF;  R0.L = 0x0000;
    [P4] = R0;     // specify the I/O size required by the ADSP-BF535 PCI
    SSYNC;
```

```
//BF535 specific register to map PCI memory space to the processor's mem space
Target_Memory_Base_Address_Pointer:
    P4.H = HI(PCI_TMBAP); P4.L = LO(PCI_TMBAP);
    R0.H = 0xF000; R0.L = 0x0000;
    [P4] = R0;     //  map inbound memory accesses to the base of L2 memory
    SSYNC;

//BF535 specific register to map PCI IO space to the processor's MMR space
Target_IO_Base_Address_Pointer:
    P4.H = HI(PCI_TIBAP); P4.L = LO(PCI_TIBAP);
    R0.H = 0xFFC0; R0.L = 0x4000;
    [P4] = R0;     // map inbound I/O accesses to 0xFFC0 4000 (PCI MMR space)
    SSYNC;

Enable_PCI_Interrupts:
    P4.H = HI(PCI_ICTL); P4.L = LO(PCI_ICTL);
    R0.H= 0x0; R0.L = 0x0040;
    [P4] = R0;     // Enable PCI reset to generate an interrupt to the ADSP-BF535 core
    SSYNC;

Enable_PCI:
    P4.H = HI(PCI_CTL); P4.L = LO(PCI_CTL);
    R0.H = 0x0; R0.L = 0x0006 ;
    [P4] = R0;     /* enable PCI as a Device; enable Fast back-to-back*/
    SSYNC;

pci_cfg_init.end:  nop;
```

Listing 1 PCI_Device_Init.asm

Document History

Version	Description
September 9, 2003 by J. Manguane	Initial Release
August 7, 2003 by J. Manguane	Initial draft release