AP-758

APPLICATION NOTE

Flash Memory PCI Add-In Card for Embedded Systems

September, 1997

Order Number: 273121-001

Flash Memory PCI Add-In Card for Embedded Systems

FIGURES

TABLES

1.0 Introduction

Embedded systems are typically designed to startup, operate, and store data to and from a type of non-volatile storage media. There are many types of non-volatile media, such as disc and tape drives. Silicon based, battery-backed volatile memory systems and non-volatile memory systems are also available. A typical embedded computing application does not require the vast amounts of storage space available from disk and tape drives. It is often impractical to use these media because of environmental limitations. Battery-backed semiconductor memory systems can provide suitable storage while the system is not operating, but reliability may be sacrificed because of limited battery lifetimes. Long term maintenance may also be an issue. For embedded systems, non-volatile semiconductor memory solutions can provide significant storage and robustness at an attractive price.

High performance embedded processors with their wide data buses and differing bus cycle protocols present a significant interface challenge to Flash memories. Three possible locations for the Flash memory interface can be considered when designing an embedded PC: the ISA bus, the PCI bus and the host bus.

Typically, the host bus cannot be heavily loaded due to speed considerations; this makes it impractical to add another host-to-Flash device to the host. Using current chipset technologies, placing the Flash on the DRAM bus would significantly reduce the bus speed and slow system performance.

Placing the Flash on the ISA bus would require dedicated logic to decode addresses, and to provide buffers and data steering. The ISA solution would be relatively slow and would limit the total amount of Flash memory to less than 16 Mbytes.

The PCI bus solution presents an easy to implement interface that can use current chipset technologies from the host processor bus. The PCI bus has good performance, industry standard protocols and provisions for system boot control.

Since typical chipsets do not directly support Flash memory interfaces, but commonly support host-to-PCI interfaces, a PCI-to-Flash solution was chosen for the proof of concept design.

The Flash Memory PCI Add-In Card was created to demonstrate the feasibility of interfacing Flash memory to the PCI bus in an embedded system. In such a system, Flash memory devices are used for high density and high performance mass storage for user code and data.

This application note describes a Flash Memory PCI Add-In Card design. This design is a proof-of-concept only. It is not a production product, nor is it being offered by Intel as such. This application note discusses design and software considerations specific to a prototype card assembled by Intel to demonstrate the feasibility of the PCI-to-Flash interface for various memory densities. This design and prototype can serve as a basis for customer designs. Several software utilities were developed for the prototype; these are described in this application note. Schematics of the prototype design and field programmable gate array (FPGA) are available for customers desiring to “copy exactly” the design presented here.

The PCI bus provides the interface between the microprocessor and the Flash memory. Since the PCI bus is widely accepted in the industry, the Flash Memory PCI Add-In Card can be easily integrated into existing systems that support the PCI bus, or it can be used in new PCI-bus-based designs. This easy integration can decrease time to market for new and retrofit designs.

An example of creating a new system with a shortened development time would be to use the Intel Embedded Processor Module (which supports PCI) and the Flash Memory PCI Add-In Card design. Designing a system using these components and appropriate baseboard support electronics saves considerable design time by eliminating the need to custom design the processor to Flash interface. For space sensitive systems without PCI connectors, the PCI-to-Flash interface can be designed to fit the system’s form factor.

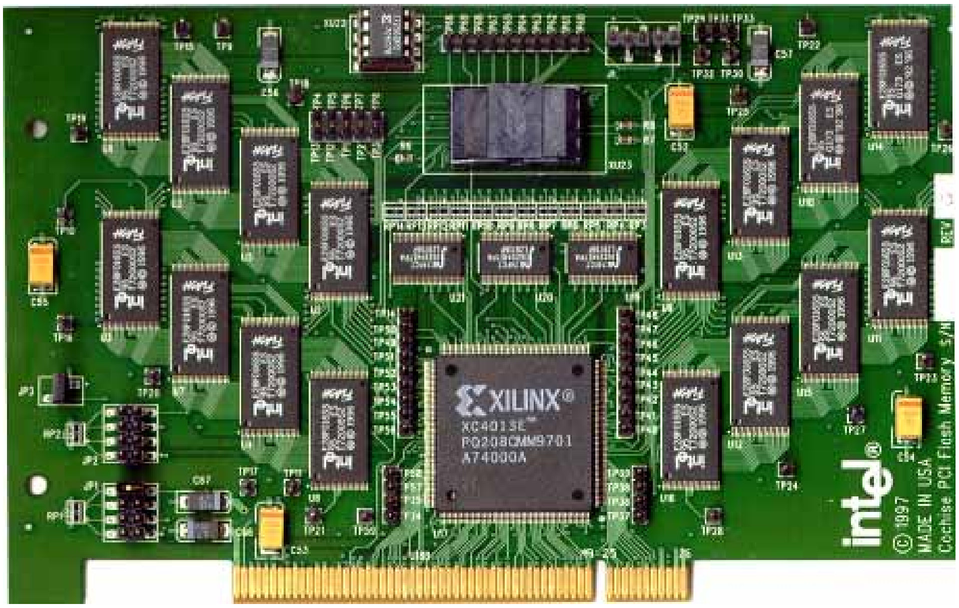

Figure 1. Flash Memory PCI Add-In Card

2.0 System Overview

The Flash Memory PCI Add-In Card Block diagram is shown in Figure 2.

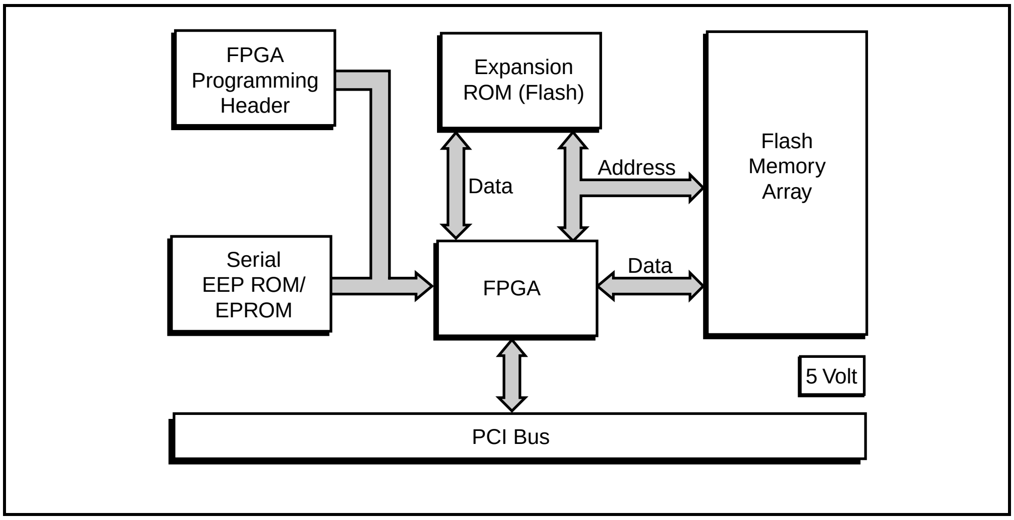

Figure 2. Flash Memory PCI Add-In Card Block Diagram

The Flash Memory PCI Add-In Card supports the PCI Local Bus Specification, revision 2.1, in both electrical characteristics and logical bus cycles. In accordance with allowable PCI protocols, it operates at 5 V on a standard 32-bit, 33 MHz bus. The card conforms to the specifications for a PCI Raw Variable Height Short Card as shown in Figure 5-3 of the PCI specification. This allows a maximum height of 4.2" and maximum length of 6.6". The card has a standard 32-bit PCI connector and operates on 5 V-only supply.

The card is designed around the Xilinx XC4000* family field programmable gate array (FPGA) with the PCI LogiCORE* interface. The PCI LogiCORE interfaces to the PCI bus and responds to all commands on the bus. The remaining FPGA logic responds to the commands from the PCI LogiCORE and generates appropriate address, data and control signals and sends them to the Flash array.

A programming header is available for FPGA programming alteration. You can write the program to the FPGA via a PC serial port. The serial ROM stores the FPGA program for configuration at power up. Once the ROM is downloaded to the FPGA, the card is ready to function.

The Flash Memory PCI Add-In Card supports the expansion ROM function. This function facilitates dead-start loads from the card at power up. The physical ROM is an Intel 2 Mbit (128 Kbit × 16 configuration) Boot Block Flash memory. This memory is set up in a ×16 mode to speed up data transfers.

The Flash Memory PCI Add-In Card supports plug-and-play operation and non-plug-and-play operation. The FPGA contains all necessary header information.

The Flash memory array is divided into four banks, with four devices in each bank. This banked array is supported by four memory density modes within the FPGA: 2 Mbyte, 8 Mbyte, 16 Mbyte and 32 Mbyte. Figure 3 shows the Flash memory array block diagram.

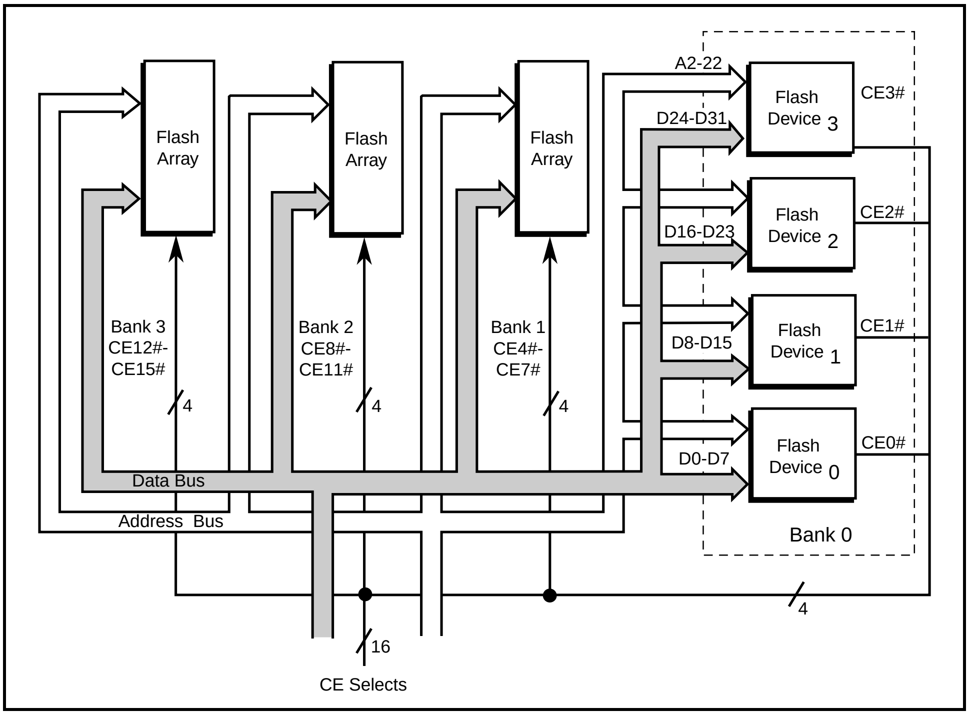

Figure 3. Flash Memory Array Block Diagram

The arrangement of the Flash devices represents a double word aligned memory array. This means that for a given 4 byte read/write starting on a double word boundary (address x00H) within bank 0, the 4 bytes reside in the Flash devices selected by CE0# through CE3# respectively. This is an important system consideration when defining software utilities and when programming and erasing the Flash. Refer to “Software Design Considerations” on page 9 for more information.

Intel’s High-Density FlashFile™ Memory components are recommended for the memory array. These are symmetrically-blocked FlashFile architecture devices. They allow selective block locking and are pin compatible with other families of Intel Flash memory. This compatibility allows for upgrades to higher performance devices.

The card is operated using standard software routines that write and read Flash devices. Standard software may need to be modified to account for the following:

- The FPGA does *not* contain algorithms that automatically perform the multiple operations necessary to program and erase the Flash devices.
- The card does *not* support burst PCI transfers.

Table 1 describes the commands used to read, write, and erase the Flash, and the commands used to determine Flash status. Consult the applicable Flash device specification and user guide to fully understand the operation of Flash devices.

Table 1. Flash Command User Interface

Command[1]	Bus Cycles Required	Notes	First Bus Cycle			Second Bus Cycle		
			Oper[2]	Addr[3]	Data[4]	Oper[2]	Addr[3]	Data[4]
Read Array/Reset	1		Write	X	FFH			
Read Identifier Codes	≥2	5	Write	X	90H	Read	IA	ID
Read Status Register	2		Write	X	70H	Read	X	SRD
Clear Status Register	1		Write	X	50H			
Block Erase	2	6	Write	BA	20H	Write	BA	D0H
Program	2	6,7	Write	PA	40H or 10H	Write	PA	PD
Block Erase and Program Suspend	1	5	Write	X	B0H			
Block Erase and Program Resume	1	5	Write	X	D0H			
Set Block Lock Bit	2	7	Write	BA	60H	Write	BA	01H
Set Master Lock Bit	2	7	Write	X	60H	Write	X	F1H
Clear Block Lock Bit	2	8	Write	X	60H	Write	X	D0H

NOTES:

1. Commands not listed in this table are reserved by Intel for future device implementations; use only the signals listed.
2. Bus operations are defined in the Flash datasheet.
3. X= Any valid address within the device
   IA= Identifier Code Address
   BA= Address within the block being erased or locked
   PA= Address of memory location to be programmed
4. SRD= Data read from status register.
   PD= Data to be programmed at location PA. Data is latched on the rising edge of WE# or CE#, whichever goes high first
   ID= Data read from identifier codes
5. Following the Read Identifier codes command, read operations access manufacturer, device, lock block, and master lock codes.
6. When the block is locked, RP# must be at V_{HH} to enable block erase or program operations. Attempts to issue a block erase or program to a locked block while RP# is V_{IH} will fail.
7. Either 40H or 10H are recognized by the WSM as the program setup.
8. When the master lock bit is set, RP# must be at V_{HH} to clear lock-bits. The clear lock-bits operation simultaneously clears all block lock bits. When the master lock bit is not set, the Clear Lock Bits command can be done while RP# is V_{IH}.

3.0 System Operations and Performance

The next few sections provide a general description of reading, writing and erasing the Flash. See the datasheet for your Flash devices for specific instructions and programming flow diagrams.

3.1 Reading the Flash

PCI-to-Flash read transactions are handled by the FPGA. DEVSEL# is asserted back to the bus master during the fourth PCI clock cycle (slow DEVSEL# timing). Wait-state timing occurs during clock cycles 5-9; on cycle 10 the Flash data is valid and TRDY# and STOP# are asserted. This forces a target disconnect because burst transfers are not supported in this design. With this read timing, assuming 4 byte transfers, this design supports up to 12 Mbyte/s transfers. The host must recognize the PCI disconnect and keep track of addressing during what normally would be a burst transfer.

3.2 Writing the Flash

PCI-to-Flash write operations are similar to read operations (slow DEVSEL# timing) but complete in five PCI clock cycles; the assertion of STOP# signals that burst transactions are not supported. The maximum write rate to the Flash is 11 Mbyte/s, assuming 4 byte writes that are double-word aligned. The write performance is slightly less than the read performance because it takes two write cycles to complete the write operation (12 PCI cycles total, including the turnaround cycles per write transaction).

Execute a read command to return to reading the array.

3.3 Erasing the Flash

Erasing the Flash is accomplished by sending appropriate commands to the Flash array. This is no different than writing to the array, except that the first write to the array indicates a command to erase; the second write command confirms the erase command and indicates the block address to be erased. Similarly, commands to the array to determine Flash status are sent via writes, and status is read through normal read operations. Table 1 lists the transactions necessary to perform all Flash operations.

Execute a read command to return to reading the array.

3.4 Scaling Memory Size

The Flash Memory PCI Add-In Card supports four different memory sizes. Table 2 describes the devices and banks used in each of the memory size types.

Table 2. Memory Size versus Layout and Device Selection

Total Size	Device Layout	Flash Density	Product number	Access Time
2 Mbyte	4×1	4 Mbit	28F004S5	85 ns
8 Mbyte	4×1	16 Mbit	28F016S5	95 ns
16 Mbyte	4×2	16 Mbit	28F016S5	95 ns
32 Mbyte	4×4	16 Mbit	28F016S5	95 ns

For size-sensitive applications, the 4 × 1 bank option provides memory densities of 2 Mbyte and 8 Mbyte simply by changing out memory types. This can result in considerable cost savings over the life of a product, since no redesign is necessary for the memory upgrade. In applications for which larger amounts of memory are required and physical size limitations are not an issue, the designer may opt for the 2 or 4 bank solution.

The proof of concept design and FPGA support up to the full 4x4 memory configuration. In a custom design based on the proof of concept, the designer may opt to use only 1 or 2 banks when there are physical size limitations in the design.

3.5 Banked Memory Configuration Considerations

The memory system employed uses four separately selectable Flash memory devices connected to a 32-bit data bus. This presents a number of considerations that must be accounted for in system development, especially software.

It is possible to write single bytes to the Flash array, and there are no restrictions on addressing. However, when writing words or double words, the double-word boundaries must not be crossed in a single transaction. This causes unpredictable results.

For contiguous address read and writes, the user must be aware that the bytes are being stored across four separate Flash devices and not in one device. For instance, four byte writes to address (x)00H are written to Flash devices 0-3 at address represented by the (x) decoded value. Therefore, from a system standpoint, the Flash memory “block” is not 64 Kbytes as defined by each device, but rather 256 Kbytes as defined for the bank of memory. Similarly when it is desired to lock the first “block” of 256 Kbytes, the command must be available across all 32 bits of the write data to indicate to each Flash device that a block lock is requested.

Table 3. Flash Programming Code Example

```
MOV     EAX, 77777777H      ;Place lock block command across all 4
                            ;bytes to signal all 4 devices to lock the
                            ;block
MOV     [address], EAX      ;Write lock block command
MOV     EAX, D0D0D0D0H      ;Place confirm command across all 4
                            ;bytes to signal all 4 devices to confirm
                            ;lock block command
MOV     [address], EAX      ;Confirm lock block
MOV     EAX, FFFFFFFFH      ;Place read array command across all 4
                            ;bytes to signal all 4 devices to go to
                            ;read mode
MOV     [address], EAX      ;return to read mode for all of bank
```

The system programmer can overcome this addressing system and use a single Flash block (64 Kbytes) for code or data by offsetting each of the byte writes/reads by 4. This ensures that the data is assigned to a single Flash device as long as the bank boundary is not crossed. The user must decide which system suits the application best. The penalty for such a system is decreased throughput performance, since only 1 byte at a time can be read or written per PCI cycle. This reduces the performance to 3 Mbyte/s for reads and 2.75 Mbyte/s for writes.

The system designer/programmer should consider Flash wear leveling. If sections of the Flash are going to be erased and programmed many times over the life of the product, and others not used at all, some provision should be made to spread the program/erase cycles evenly between devices. This increases system data retention reliability and life.

3.6 Expansion ROM Operation

An Intel Boot Block Flash memory device is recommended as the PCI expansion ROM. This device operates in ×16 bit mode to speed boot load operations. The expansion ROM is accessed at power-up through the system BIOS. The ROM PCI cycle can provide the 4 bytes in 12 PCI clock cycles for a maximum read performance of 11 Mbytes/s.

During initialization the system BIOS checks the Expansion ROM header. The BIOS verifies that the header is correct (using signatures and checksums). Depending upon the capabilities of the BIOS and the data contained within the ROM header, the BIOS shadows the expansion ROM contents to shadow RAM and calls the initialization routine. The initialization routine is responsible for card specific initialization functions. Once the initialization is complete, the ROM initialization code returns to the BIOS for completion of the system initialization.

4.0 Hardware Considerations

4.1 Address Line Buffering

Since the address lines on the Flash side of the FPGA connect to as many as 17 devices, additional buffering is needed to accommodate the load. Each address line from the FPGA goes to two buffers and is distributed to memory. Address lines are routed through the address pins to minimize load reflections. Series resistors are employed to reduce reflections in the long lines connected to the memory arrays.

4.2 Choosing Memory Size

The Flash Memory to PCI Add-In Card has two sets of jumpers that select the configuration.

JP1: This block selects the Flash memory array size. This size is selectable from 64 Kbytes to 2 Gbytes. Only four sizes are currently supported by the chip-enable code within the FPGA: 2 Mbyte, 8 Mbyte, 16 Mbyte, and 32 Mbyte. Refer to the schematics for these jumper settings. (Schematics are available from the Intel World Wide Web site; see “Downloadable Files” on page 13 for more information.)

JP2: This configures the expansion ROM size. By default (all jumpers open) the expansion ROM is set at 64 Kbytes. The FPGA supports all sizes up to 2 Gbytes, although the PCI specification limits the Expansion ROM size to 16 Mbytes. Since there is only one socket for the expansion ROM, this size is really limited by the size of available Flash memory.

Jumpers or hardwired through-holes provide a cost effective configuration change mechanism for designs that support more than one memory size.

4.3 FPGA Options

The FPGA is programmed by a bitstream from either the serial ROM or from a PC. This bitstream was created by compiling the VHDL code and the LogiCORE netlist with the FPGA software tools from Xilinx. Once the FPGA is programmed with this bitstream it acts as the bridge between the PCI bus and the Flash memory array by converting the PCI bus transactions into Flash memory control signals.

The FPGA supports master and slave modes. The mode is selectable through jumper JP3. When the jumper block is open, the FPGA is in slave mode. In slave mode, the FPGA must be programmed through the header connection on the board using a PC and the appropriate tools. When the jumper is installed, the FPGA is in master mode and automatically downloads its configuration from the serial ROM.

4.4 Socketing Versus Soldering

This proof of concept design uses Flash devices soldered to the board to improve signal reliability and to reduce cost. Socketing provides the advantage of quick replacement, which might be required when one Flash device is erased or programmed more often than the other devices. The Expansion ROM device is socketed so that it can be easily swapped, since the device is not rewritable in-circuit.

4.5 PCB Layout

The critical signals on the PCB are the traces between the PCI connector and the FPGA, and between the FPGA and the Flash memory devices. The traces between the FPGA and the PCI bus connector must be less than 1.5” long, except the PCI clock signal which must be 2.5” (+/- 0.1”) long. This is a requirement of the PCI Local Bus Specification, revision 2.1.

To minimize the length of the address line traces, they are routed *through* the pins (rather than having stubs connecting the pins to the trace) of the Flash memory array devices which were laid out in an angular fashion.

Decoupling capacitors are located next to power pins on the devices and between the power and ground planes of the PCB to act as filters and reduce noise in the power signals. Refer to the schematics for information on the placement of decoupling capacitors. Decoupling consistent with published datasheets should always be employed.

The resistors placed on the address lines between the outputs of the buffers and the address lines of the Flash devices are used to suppress the effects of reflections in the address lines.

5.0 Software Overview

Few embedded applications require a full file system implementation. In most cases, designers do not want to provide the storage space required for a file system. Typical embedded systems do not run a full-size, hardware-protected operating system, such as Windows NT. For these reasons it was decided that a file system and driver would not be used for the Flash Memory PCI Add-in Card. Instead, proprietary applications were used to demonstrate card functionality.

It is advisable for the card to begin executing after system start. An expansion ROM serves this function.

5.1 Software Design Considerations

The memory system uses four separately selectable Flash memory devices connected to a 32-bit data bus. This presents several software design considerations when programming the card.

5.1.1 Erasing/Programming Algorithms

Standard Flash implementations provide contiguous address space for each chip implemented. Since the card implements a 32-bit data bus, the algorithms used to erase and program devices must change.

Figure 4 shows the address map for a 16-byte block within the 32-bit data bus implementation.

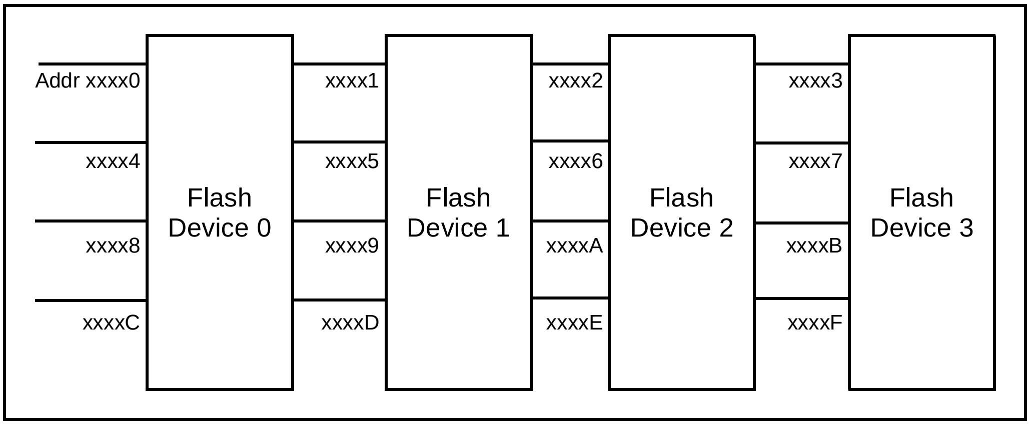

Figure 4. Memory Block Address Map

The erase/programming algorithms must take this into account since the contiguous address space spans multiple devices.

Two methods can be used to address this issue.

- The first method is to create an algorithm that accounts for the fact that consecutive addresses within the same device differ by 4 instead of 1. This method is useful only when the programmer is running diagnostic routines on a specific device.
- The second method involves taking advantage of the 32-bit data bus. When programming/erasing is initiated, the command code byte sent to the devices is replicated four times within a 32-bit command/data word. This invokes the program/erase operation on all four devices within that address line. When using this method, the programmer must verify that accesses are done on double-word boundaries. Once the command code is written, double-word accesses must then be performed to retrieve status from all four devices. A ready condition should not be acknowledged until ready is returned from all devices.

5.2 Applications

Several applications were created for the Flash Memory PCI Add-in Card. These are available for download from Intel's World Wide Web site (http://www.intel.com; see "Downloadable Files" on page 13 for more information). The following sections describe these applications.

5.2.1 COCHISE.EXE

COCHISE.EXE is a DOS executable that provides the following high-level functions:

- Plug and Play identification of device
- Flash part detection and identification
- Erase/Write/Read testing of entire Flash space
- Device erase capability
- Download of DOS file to Flash
- Upload of Flash contents and storage in file

The application is a menu based system that allows the user to choose which actions should take place.

5.2.2 CUPLOAD.EXE

CUPLOAD.EXE is a DOS executable that transfers a file from the system (motherboard) to the Flash. The executable verifies that the card exists in the system using Plug and Play (PnP) BIOS calls, gets the configuration information for the PCI device and the Flash devices, and then copies the file from the system to the Flash. The executable supports other functions. To determine the available options, type /? on the command line.

5.2.3 CDNLOAD.EXE

CDNLOAD.EXE is a DOS executable that transfers the contents of the Flash devices on the card to a specified DOS binary file. The executable verifies the existence of the card within the system (through the Plug and Play BIOS), retrieves the Flash device information, and copies the data from the Flash device to the specified file. To determine available options for the executable, use the /? Option on the command line.

5.3 Software Alternatives

The card could be used by an operating system as a nonvolatile storage device. To do this, the user must write a device driver that enables the operating system to format, write and read the Flash devices on the card. Several Flash file system alternatives exist that could be used once the driver is written. This option is not addressed within this application note.

5.4 Boot Execution

Typical embedded applications do not require user interaction before they begin execution. Possible solutions to execute the code stored in the card upon system start are dependent upon the capabilities of the system BIOS. The following sections describe the options based upon the capabilities of the BIOS.

5.4.1 BIOS Boot Specification 1.01

The BIOS Boot Specification as defined by Compaq, Phoenix Technologies and Intel provides the means for any device in a system to be identified as an Initial Program Load (IPL) device. If the system BIOS supports this specification, the expansion ROM within the card can indicate that it is an available IPL device.

The BIOS maintains a list of available IPL devices and assigns a priority to the devices. The user can change to the IPL order/priority of the available devices. If the expansion ROM is used as an IPL device, the IPL code must be contained within the expansion ROM space of the card. The system BIOS calls the Bootstrap Entry Vector (BEV) to boot the operating system. The expansion ROM either copies the OS to DRAM for execution or jumps directly to the entry point of the image contained in Flash. Example ROM code is available which provides an example of how to implement a Plug and Play IPL device. Please see the BIOS Boot Specification for a complete description of all required tables and actions to support this capability.

5.4.2 Expansion ROM Initialization

If the system BIOS does not support the PnP Boot Process, the expansion ROM can take advantage of the expansion ROM initialization. During the POST process, the system BIOS shadows all valid expansion ROMs and calls the initialization routines. The card can determine if the BIOS supports the PnP installation and boot process by checking input parameters provided to the unitization routine.

If the system BIOS does not support the BIOS boot specification (as determined at run time by the expansion ROM), the expansion ROM can perform any necessary initialization, and prior to returning control to the system BIOS can perform one of the following actions:

- Provide a option screen for the user to select available boot options
- Automatically load the image to DRAM from Flash and execute
- Jump to the entry point in the Flash image to provide eXecute In Place (XIP) for the image

When this method is used, it is recommended to use the first option. If the other options are chosen, the flexibility of the system to boot from alternate devices upon request and recover from boot failure is limited.

6.0 VHDL Code

Figure 5 shows the functional block diagram of the VHDL code used for the Flash Memory PCI Add-In Card design. VHDL code is available from Intel’s World Wide Web site (http://www.intel.com; see “Downloadable Files” on page 13 for more information).

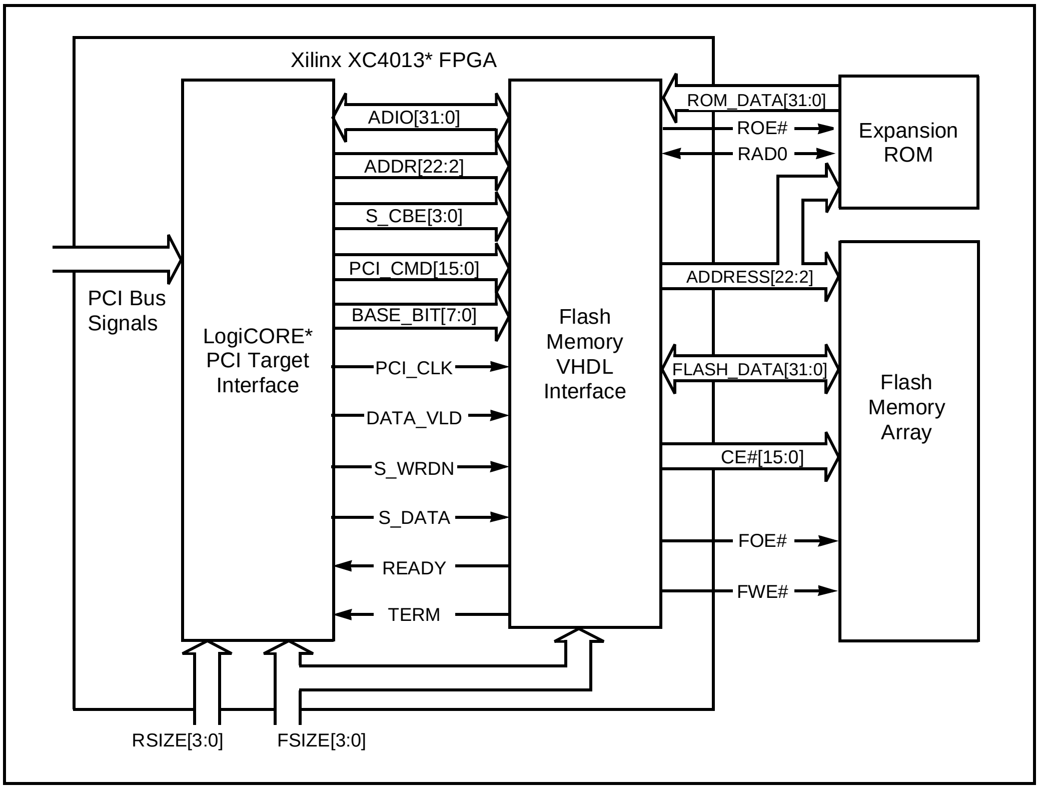

Figure 5. VHDL Code Block Diagram

7.0 Schematic and Material List

Schematics and the material list are available from Intel's World Wide Web site (http://www.intel.com; see "Downloadable Files" on page 13 for more information).

8.0 Vendor Contact List

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
(408)559-7778

Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
(408) 765-8080

9.0 Specification Summary

PCI compatibility:	Revision 2.1
Logical PCI Interface:	Xilinx LogiCORE PCI Slave Interface (LC-DI-PCIS-C)
Programmable Logic:	Xilinx XC4013E-2PQ208C
Configuration PROM:	XC17256DPD8C, 256 Kbit Serial PROM
Development Software:	Synopsys* VHDL FPGA-Compiler, Xilinx Foundation Series 6.0.2, XACT* 6.0 for PC
Flash Array Devices:	Intel FlashFile™ Memory Architecture 28F004S5, 28F016S5
Flash access time:	85 ns (28F004S5), 95 ns (28F016S5)
Expansion ROM:	Intel Boot Block Flash Memory (E28F200-B5-B60)
Electrical voltage:	5 V_{cc}, 5 V_{pp}
Operating Frequency:	33 MHz
Data Path:	32 bits

10.0 Related Documents

These documents are available for download from Intel's World Wide Web site at http://www.intel.com.

Table 4. Related Documents

Document	Order Number
Byte-Wide Smart 5 FlashFile™ Memory Family datasheet	290597
2-Mbit Boot Block Flash Memory Family datasheet	290448
Intel Embedded Processor Module datasheet	273105
Intel Embedded Processor Module Evaluation Board Developer's Manual	273122
PCI Local Bus Specification, Revision 2.1	This document can be ordered from the PCI Special Interest group at: http://www.pcisig.com

10.1 Downloadable Files

Files mentioned in this document are available for download from the Intel World Wide Web site at http://www.intel.com. Search for FL_AIC.EXE to find the following information related to the Flash Memory PCI Add-In Card proof of concept design:

- Bill of Materials for the card
- VHDL source code and description
- FPGA Serial ROM code used on the card
- Software executables discussed in this application note
- Schematics for the card