XAPP502 (v1.4) November 13, 2002

Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode

Author: Mark Ng and Mike Peattie

Summary

With embedded systems becoming more popular, many designers want to reduce their component count and increase flexibility. To accomplish both of these goals, a microprocessor can be used to configure an FPGA. This application note provides a thorough discussion of FPGA configuration, covering Virtex™ Series, Virtex-II Series Platform FPGAs, and Spartan™-II Series devices. Also, this application note presents a system-level model using a Xilinx Complex Programmable Logic Device (CPLD) to implement an interface to the FPGA configuration pins. C code is included to illustrate an example application using either Slave Serial or SelectMAP mode. CPLD design files are included to illustrate a possible synchronous interface between the processor and the FPGA.

System Overview

Today's systems are becoming more complex and at the same time are becoming cheaper and smaller. While the Xilinx Virtex Series, Virtex-II Series Platform FPGAs, and Spartan-II Series FPGAs continue to deliver the performance and reliability that is now expected of programmable logic, designers have found it necessary to configure their FPGAs without the use of dedicated configuration devices. These configuration devices often times consume unnecessary board real estate and increase the system cost.

This application note describes a technique for achieving low-cost, efficient configuration of Xilinx FPGAs through the use of three common components: an embedded microprocessor, some non-volatile memory, and a CPLD. Cost, as well as real estate, can be reduced if the function of a dedicated configuration device, such as a PROM, can be integrated within these three components. A system diagram is shown in Figure 1.

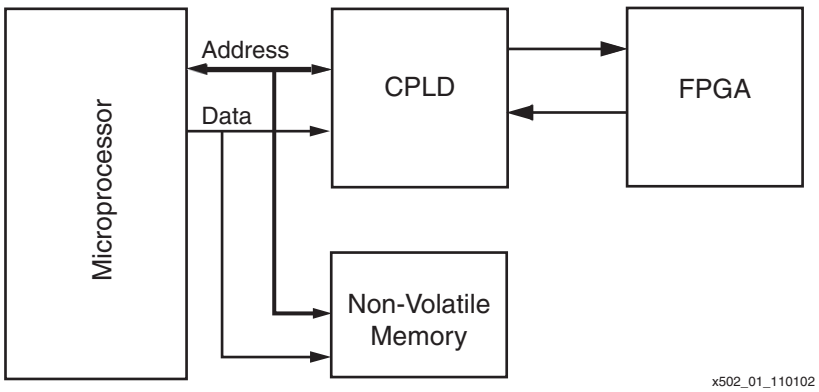

Figure 1: **System Diagram**

Note:

1. Some systems may not require a CPLD if the microprocessor has a sufficient number of general-purpose I/O (GPIO) pins available. For these systems, the Xilinx FPGA can be configured directly by the microprocessor. For this type of configuration, this application note still applies. The overall configuration flow will remain the same, but the user will have to modify the source code so that the microprocessor strobes its GPIO pins instead of its address and data bus.

Instead of using a dedicated PROM, the configuration bitstream can be loaded into unused portions of non-volatile system memory. Furthermore, a microprocessor, whose primary purpose may be to perform other tasks, can also be used to coordinate the loading of configuration data into a Xilinx FPGA device. Unused register bits of a CPLD accessible to the microprocessor can then be used to monitor and instrument the FPGA's control, data, and status bits. Microprocessors typically have a limited number of control signals, and a simple CPLD design can map a portion of the microprocessor's address space to control FPGA configuration. Using this method, the CPLD establishes a synchronous interface between the microprocessor and the Xilinx FPGA. Such an interface can also allow the microprocessor to do more advanced functions, like partial reconfiguration and readback.

Configuration Background

Microprocessor programming of Virtex Series, Virtex-II Series Platform FPGAs, and Spartan-II Series FPGAs can be accomplished in either Slave Serial and SelectMAP mode. There are several similarities between Slave Serial and SelectMAP modes. Most importantly, the overall configuration flow is identical for both modes. See Figure 2.

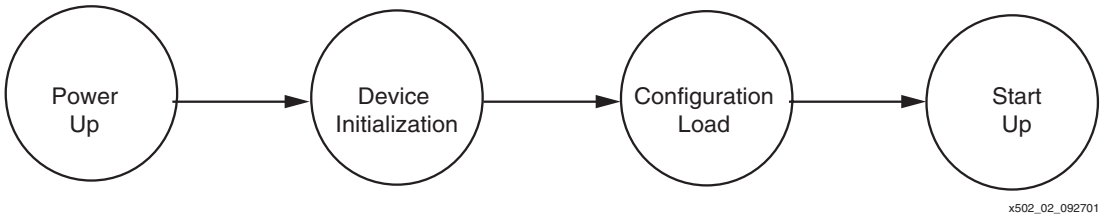

Figure 2: **Configuration Flow**

- **Power Up**

  Power Up is when power is first applied to the FPGA. Internal state machines are reset, and the device begins to wake up. At this point, the $\overline{\text{PROGRAM}}$ and $\overline{\text{INIT}}$ pins are both driven Low by the FPGA.

- **Device Initialization**

  The device has properly powered up, but the internal configuration memory needs to be reset. This portion of the configuration flow is signaled by $\overline{\text{PROGRAM}}$ going High and $\overline{\text{INIT}}$ going High a short time later (see the appropriate data sheet for this delay). The device can remain permanently in this state if the user holds either $\overline{\text{PROGRAM}}$ or $\overline{\text{INIT}}$ Low externally.

- **Configuration Load**

  The start of the configuration load phase is signaled by the $\overline{\text{INIT}}$ signal going high. After $\overline{\text{INIT}}$ has gone High, the mode pins (M2:M0) are sampled. The mode pins indicate which configuration mode is desired. Consult the data sheet for the mode pin settings required for the mode/device combination. In this phase, the device receives configuration data. All configuration events occur on the rising edge of CCLK.

- **Start Up**

  After the device has received all the configuration data, it proceeds to the startup sequence. This sequence governs when the global reset signals (GTS, GWE, and for Virtex/E/Spartan-II, GSR) are toggled and when the DONE pin goes High. By default, configuration is not complete when DONE goes High, that is, a four more CCLK cycles are needed to complete the startup sequence. The best practice is to load all the data in the configuration file, and then check for DONE assertion. Consult the *BitGen* section of the *Development Systems Reference Guide* for additional BitGen options and details.

Slave Serial Specific Topics

After $\overline{\text{INIT}}$ goes High, one bit of Slave Serial configuration data (presented on the DIN pin) is loaded into the configuration logic on each rising CCLK edge. Refer to the appropriate data sheet for setup and hold time specifications. Table 1 describes the pins that are used during Slave Serial configuration.

Table 1: **Slave Serial Pin Descriptions**

Signal Name	Direction	Description
CCLK	Input	Configuration clock
PROG	Input/Output	Asynchronous reset to configuration logic. Indicates when device has cleared configuration memory.
INIT	Input/Output	Input to delay configuration. Indicates when device is ready to receive configuration data; also indicates configuration errors.
DONE	Input/Output	Input to delay device startup. Indicates when configuration is in the startup sequence.
M[2:0]	Input	Configuration Mode selection
DIN	Input	Serial configuration data input
DOUT	Output	Data output for serial daisy chains

SelectMAP Specific Topics

SelectMAP configuration data is loaded one byte at a time presented on the D[0:7] bus on each rising CCLK edge. Notice that the most significant bit (MSB) of each configuration byte is on the D0 pin (see the **Data Formatting and Byte-Swapping** section for more details). Two extra control signals are present for SelectMAP, $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$. These signals must both be asserted Low for a configuration byte to be transferred to the FPGA. A third signal, BUSY, is an output from the FPGA. When SelectMAP configuration is run very fast (> 50 MHz for Virtex, Virtex-E, and Spartan-II and > 66 MHz for Virtex-II), the BUSY line must be monitored to ensure that data was transferred. If BUSY is High, this indicates that the last data byte was not transferred and must remain on the data bus. Refer to the appropriate data sheet for setup and hold specifications for all signals. Table 2 shows the SelectMAP pin descriptions.

Table 2: **SelectMAP Pin Descriptions**

Signal Name	Direction	Description
CCLK	Input	Configuration clock
PROG	Input/Output	Asynchronous reset to configuration logic. Indicates when device has cleared configuration memory.
INIT	Input/Output	Input to delay configuration. Indicates when device is ready to receive configuration data; also indicates configuration errors.
DONE	Input/Output	Input to delay device startup. Indicates when configuration is in the startup sequence.
M[2:0]	Input	Configuration Mode selection
D[7:0]	Input	Byte-wide configuration data input

Table 2: **SelectMAP Pin Descriptions**

Signal Name	Direction	Description
$\overline{CS}$	Input	Active Low Chip Select input
$\overline{WRITE}$	Input	Active Low Write Select/Read Select
BUSY	Output	Handshaking signal to indicate successful data transfer (same pin as DOUT in Serial mode).

Data Formatting and Byte-Swapping

Since the configuration bitstream will be loaded into some memory connected to the processor, most likely it will have to be formatted in a way that the processor (or whatever device programs the memory) can use. There are a number of different formats that the Xilinx tools can produce. Table 3 lists these formats with a short description.

Table 3: **Xilinx Tool Formats**

File Extension	Description
.bit	Binary file that contains header information that should not be downloaded to the FPGA.
.rbt	ASCII file that contains a text header and ASCII 1's and 0's
.bin	Binary file that contains no header information (only produced by 4.1i and later versions of Xilinx software)
.mcs, .exo, .tek	ASCII PROM formats that contain address as well as checksum information
.hex	ASCII PROM format that only contains data

Generally, either the .**bin** or .**hex** files are the most useful data source for programming non-volatile memory. Some additional formatting may be required, and the Perl script included with this application note (**bitformat.pl**) can provide some assistance. This Perl script uses the .**bit** or .**rbt** file as a source and produces a user-customizable ASCII file that can be helpful in programming the on-board memory. Xilinx implementation software must be installed on the machine running this Perl script because the PROMGen program is called from within the script (For details, see the **bitformat.readme** file that is included).

In terms of data ordering, Slave Serial configuration is very simple. Start with the first bit in the bitstream and load one bit at a time until the end of the file is reached.

Data ordering for SelectMAP configuration is slightly more complex. Configuration data is loaded one byte on each CCLK, and the MSB of each byte is presented on the D0 pin, not the D7 pin. Because of this non-conventional ordering, presenting the data "as is" from the .**bin** file is generally incorrect. The reason is that most processors interpret D7 (not D0) as the most significant bit in each byte. Connecting D7 on the processor to the D7 on the FPGA SelectMAP data bus effectively loads the data "backwards." Thus, the configuration will not be successful.

For this reason, the source data stream may need to be "byte-swapped." This means that bits in each byte in the data stream are reversed. Figure 3 shows two bytes (0xABCD) being reversed.

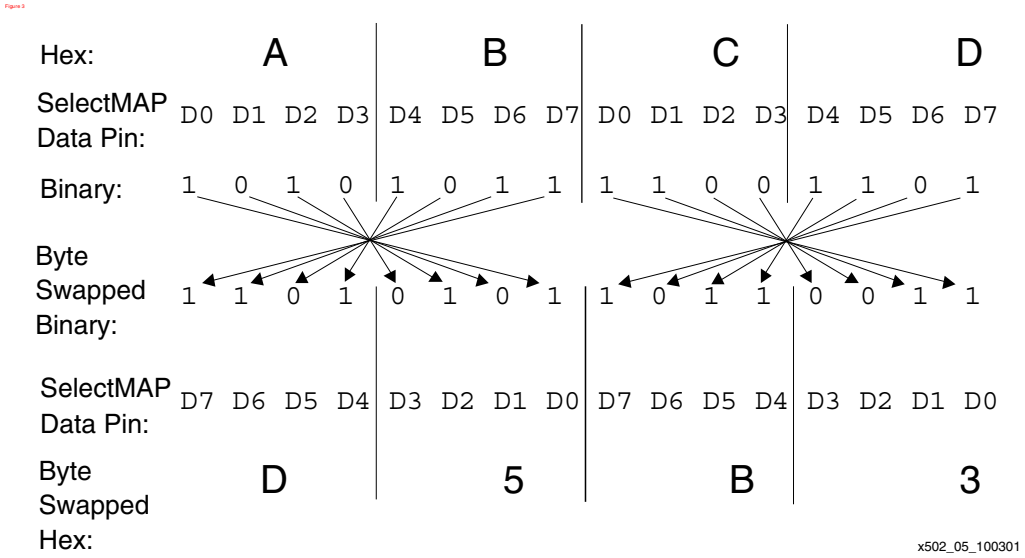

Figure 3: **Byte-Swapping Example**

Notice that the MSB of each byte of the data needs to go to D0 regardless of the orientation of the data. However, in the byte-swapped version of the data, the bit that needs to go to D0 is the rightmost bit and, in the non-byte-swapped data, the D0 bit is the leftmost bit. The Perl script included in this application note can be customized to produce byte-swapped files or not, as needed (see the **bitformat.readme** file for details). In the Xilinx software, the .**mcs**, .**exo**, and .**tek** files are always byte-swapped, and the .**bit**, .**rbt**, and .**bin** files are never byte-swapped. Hex files may be produced byte-swapped or not according to command-line options.

Notes:

1. Whether or not data is byte-swapped is entirely processor/application dependent and is generally only applicable for SelectMAP applications. Non-byte-swapped data should be used for Slave Serial downloads.

Errors and Troubleshooting

If the configuration is not successful, the DONE pin does not go High after all the data is loaded. There are many different reasons why DONE does not go High. Instead of discussing them herein, the reader is encouraged to search the Xilinx Answers Database and go through the Configuration Problem Solver at: http://support.xilinx.com.

Hardware Implementation

Microprocessor

A Motorola Dragonball MC68VZ328 processor is used in this design. This specific design uses a Handspring Visor Handheld computer to configure a Xilinx Virtex FPGA. To complete the system, an Insight Springboard Development kit provides 32 Megabits (Mb) of on-board Flash memory, as well as a 256-macrocell CoolRunner™ CPLD.

Figure 4 shows the memory map for the Dragonball processor in this design. The Handspring Visor Springboard expansion slot provides two Chip Select regions available to the user, called Chip Select 0 and Chip Select 1. The Flash memory is connected to the Chip Select 0 region. This is defined as the address space between 0x28000000 and 0x28FFFFFF. The CPLD is connected to the Chip Select 1 region, defined by the address space between 0x29000000 and 0x29FFFFFF. Each address location references 8 bits of data. However, the Dragonball processor does not support bytewide addressing. As a consequence, the microprocessor's address line 0 is not used. Therefore, data can only be accessed on even address locations (i.e.,
0x28000000, 0x28000002, etc.). Each access reads or writes 16 bits of data. Accessing address locations on odd boundaries is not allowed.

The shaded area on the bottom represents portions of memory that are used specifically by the processor for internal functions. This memory space is not utilized in this reference design.

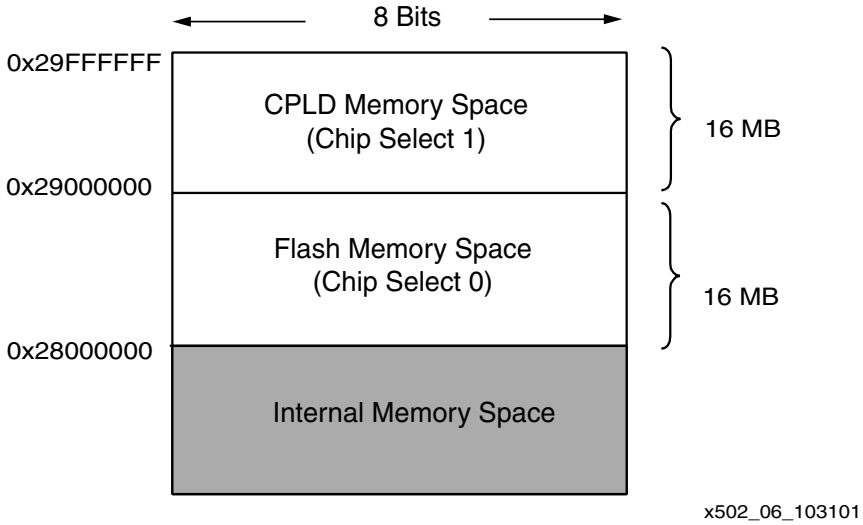

Figure 4: **Microprocessor Memory Map**

Notes:

1. Even though a Motorola Dragonball processor is used in this reference design, any microprocessor can be used. In addition, the microprocessor code in this reference design can be easily ported to different processors. This requires an understanding of how to retarget the memory mapped Dragonball I/O structure to match the memory map of another processor.

Flash Memory

In this design, the 16-bit wide Flash memory begins at address 0x28000000 and ends at address 0x28FFFFFF. As shown in Figure 5, a Virtex XCV50 configuration bitstream consists of 559,200 bits. Since each address location contains 16 bits of data, a total of 34,950 addresses are needed. The first 16 bits of configuration data reside at address 0x28000200 and the final 16 bits are located at address 0x2801130A. This design arbitrarily chose Flash address 0x28000200 to be the starting address in order to simulate actual customer scenarios in which the Flash is also used to hold other system data.

The data within the Flash memory may need to be byte-swapped, depending upon the application. For this design, byte-swapped data is used for SelectMAP configuration but not for Slave-Serial configuration. Byte-swapping reformats the FPGA configuration data so that the order of the bits within each byte is reversed. For a complete description of byte-swapping, refer to the description in the *Configuration Background* section.

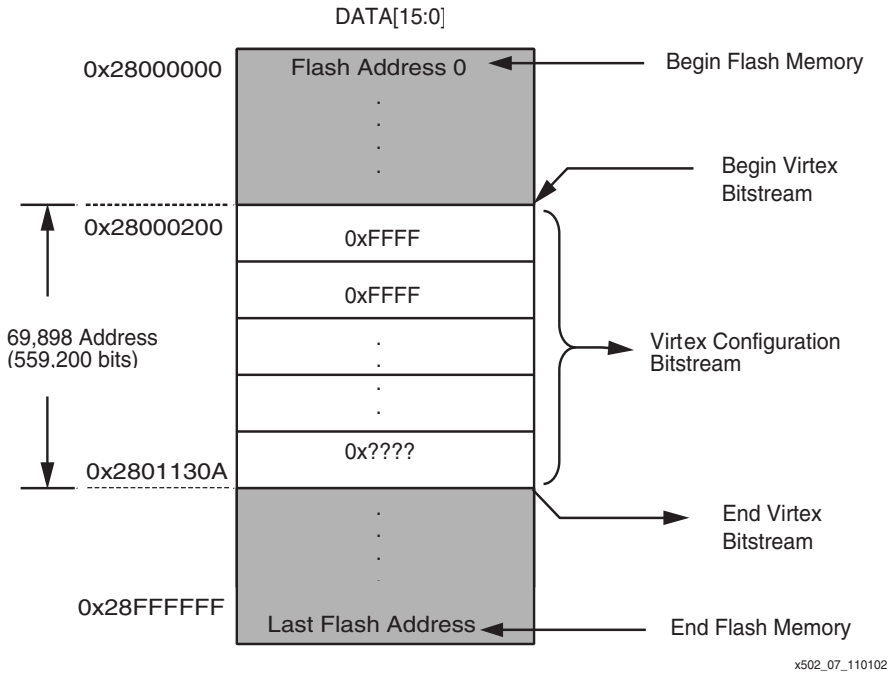

Figure 5: **Flash Memory Map**

SelectMAP Hardware

The SelectMAP configuration mode is the fastest configuration option in which data is presented to the FPGA in a byte-wide fashion. This section discusses a reference design that allows a microprocessor to configure an FPGA device via the SelectMAP mode. In order to accomplish SelectMAP configuration, this design uses a combination of a microprocessor, CPLD, and Flash memory. See Figure 6.

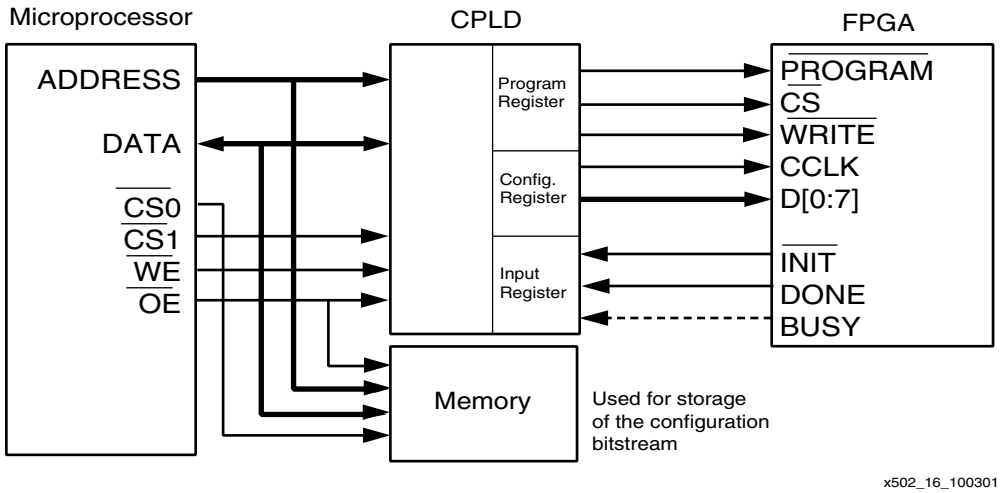

Figure 6: **SelectMAP Configuration**

A 16-bit wide Flash memory device is used to store the FPGA configuration data. Since each Flash address stores 16-bits of data, the microprocessor reads each address to retrieve the 16-bit wide data and delivers the data in a byte-wide fashion to the FPGA. During this process, the microprocessor provides the FPGA with a configuration clock and monitors the DONE and BUSY pins. A CPLD is used to form the glue logic between the microprocessor and the FPGA device.

This section discusses the functional microprocessor C code, as well as the functional VHDL code that can be readily targeted to a Xilinx 9500XL/XV, as well as a Xilinx CoolRunner XPLA3 CPLD.

CoolRunner CPLD

A Xilinx CoolRunner CPLD is used for glue logic between the microprocessor and the Xilinx Virtex FPGA. In the SelectMAP configuration mode, the CoolRunner device is responsible for driving the Virtex FPGA configuration pins, namely Program ($\overline{\text{PROGRAM}}$), Chip Select ($\overline{\text{CS}}$), Write ($\overline{\text{WRITE}}$), Configuration Clock (CCLK), and Data (D[0:7]). The Virtex FPGA's $\overline{\text{INIT}}$, DONE, and BUSY pins are also registered by the CPLD, so that the status of these pins can be monitored by the microprocessor.

The CoolRunner is used to establish a synchronous interface between the microprocessor and the Xilinx FPGA. The interface is comprised of three registers: the Configuration Register, Program Register, and Input Register. These registers store FPGA configuration signals each time the processor does a read or write to the port. They support the set of signals required for SelectMAP mode ($\overline{\text{PROGRAM}}$, CCLK, DATA[0:7], $\overline{\text{INIT}}$, $\overline{\text{CS}}$, $\overline{\text{WRITE}}$, DONE, and BUSY). The Configuration Register and Program Register are write-only registers, while the Input Register is read-only.

A detailed block diagram of the CPLD is shown in Figure 7.

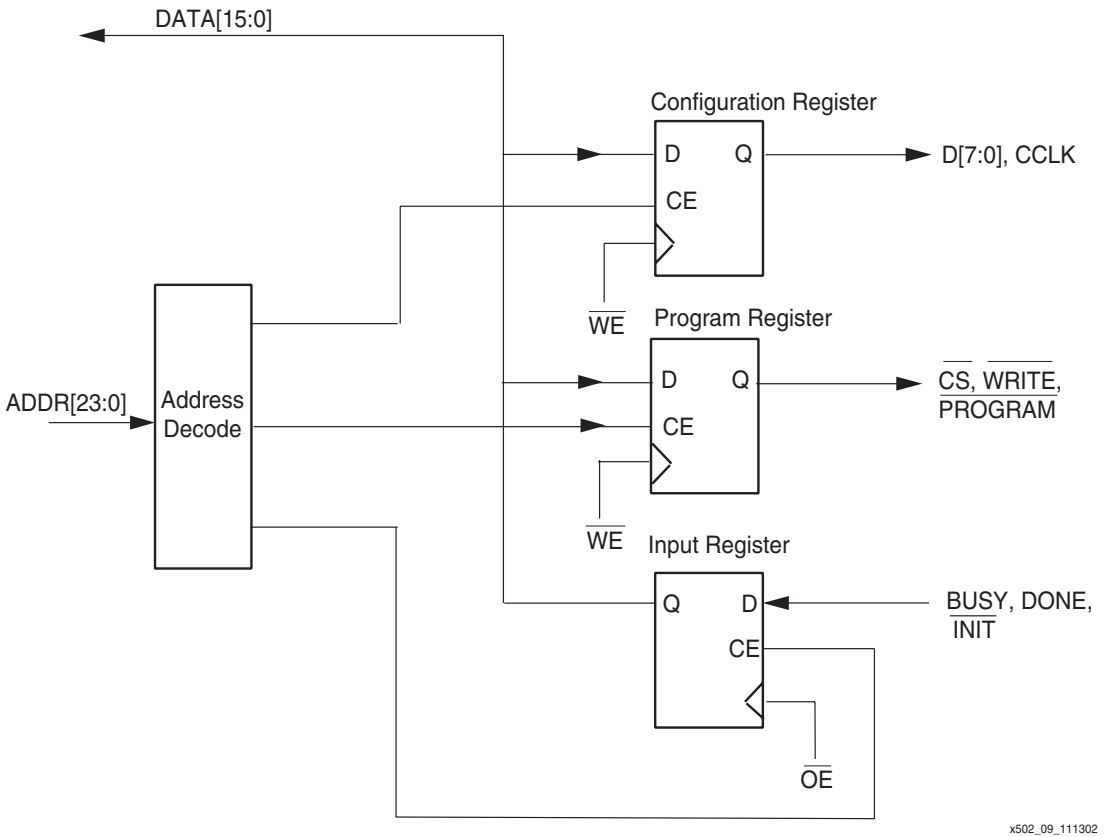

Figure 7: **CPLD Block Diagram**

The Register Address Map is shown in Table 4.

Table 4: **Register Address Map**

REGISTER ADDRESS A[23:0]	REG. NAME	REGISTER DATA									
		D[15:9]	D8	D7	D6	D5	D4	D3	D2	D1	D0
0x000000	Configuration Register	D[7:0]	NA	NA	NA	NA	NA	NA	NA	NA	CCLK
0x000002	Program Register	NA	NA	NA	NA	NA	NA	NA	CS	WRITE	PROG
0x000004	Input Register	NA	NA	NA	NA	NA	NA	NA	BUSY	DONE	INIT

SelectMAP.c

The C code discussed in this section allows a microprocessor to do the following:

- Read FPGA configuration data from Flash memory
- Generate a CCLK
- Deliver byte-wide data to the FPGA
- Check the status of the BUSY pin (optional)

Before continuing, it is important to note that different microprocessors have different memory mapped I/O structures, depending upon the system configuration. Therefore, while the majority of the C code listed here should be easily transportable, the user will have to modify the sections of code that commands the microprocessor to perform a write operation on its data bus to match the memory map of the new microprocessor. This modification should not be difficult.

To allow for maximum readability, all C code shown in this reference design accesses the microprocessor address and data bus uniformly. This is accomplished through two functions, called IOWrite() and IORead(). IOWrite() writes a 16-bit word to a specified address location. IORead() retrieves a 16-bit result from a specified location. Refer to the **Microprocessor** section for this reference design's memory map. Refer to the **Appendix: Processor Specific I/O Function Calls** for the syntax of the IOWrite() and IORead() functions.

The SelectMAP.c source file contains three important functions: SelectMAP(), SelectMAP_output(), and Busy_Check(). The SelectMAP() function is called from within the main function to begin configuration. SelectMAP() pulses the FPGA's $\overline{\text{PROGRAM}}$ pin and checks for $\overline{\text{INIT}}$ deassertion. After $\overline{\text{INIT}}$ is deasserted, the FPGA is ready to receive configuration data, the SelectMAP_output() function is called. A software flow diagram is shown in Figure 8.

First, the processor writes to the Program Register to assert the $\overline{\text{PROGRAM}}$ pin, which will reset the FPGA. A loop may be required to assert the FPGA's $\overline{\text{PROGRAM}}$ pin for the minimum required time (refer to the appropriate data sheet). Note that if the FPGA has just been powered up or reset through other means, this is not necessary. After the $\overline{\text{PROGRAM}}$ bit has been asserted, the SelectMAP() function checks the CPLD Input Register until $\overline{\text{INIT}}$ is High.

Then, the function enters a *for* loop that cycles through the Flash memory address range that contains the FPGA configuration data. In this example, using a V50 device the Flash address range is between 0x28000200 and 0x2801130A. For each address location, the 16-bit data is retrieved, and the SelectMAP_output() function is called to perform presentation of byte-wide data to the FPGA.

Notice that in the SelectMAP_output() function, two write cycles are needed for each byte in the configuration file. One cycle is needed for driving CCLK Low and to present the next configuration byte. A second cycle is needed to drive CCLK High. Immediately after driving the CCLK pin High, Busy_Check() is called to ensure that the data has been properly received by the FPGA.

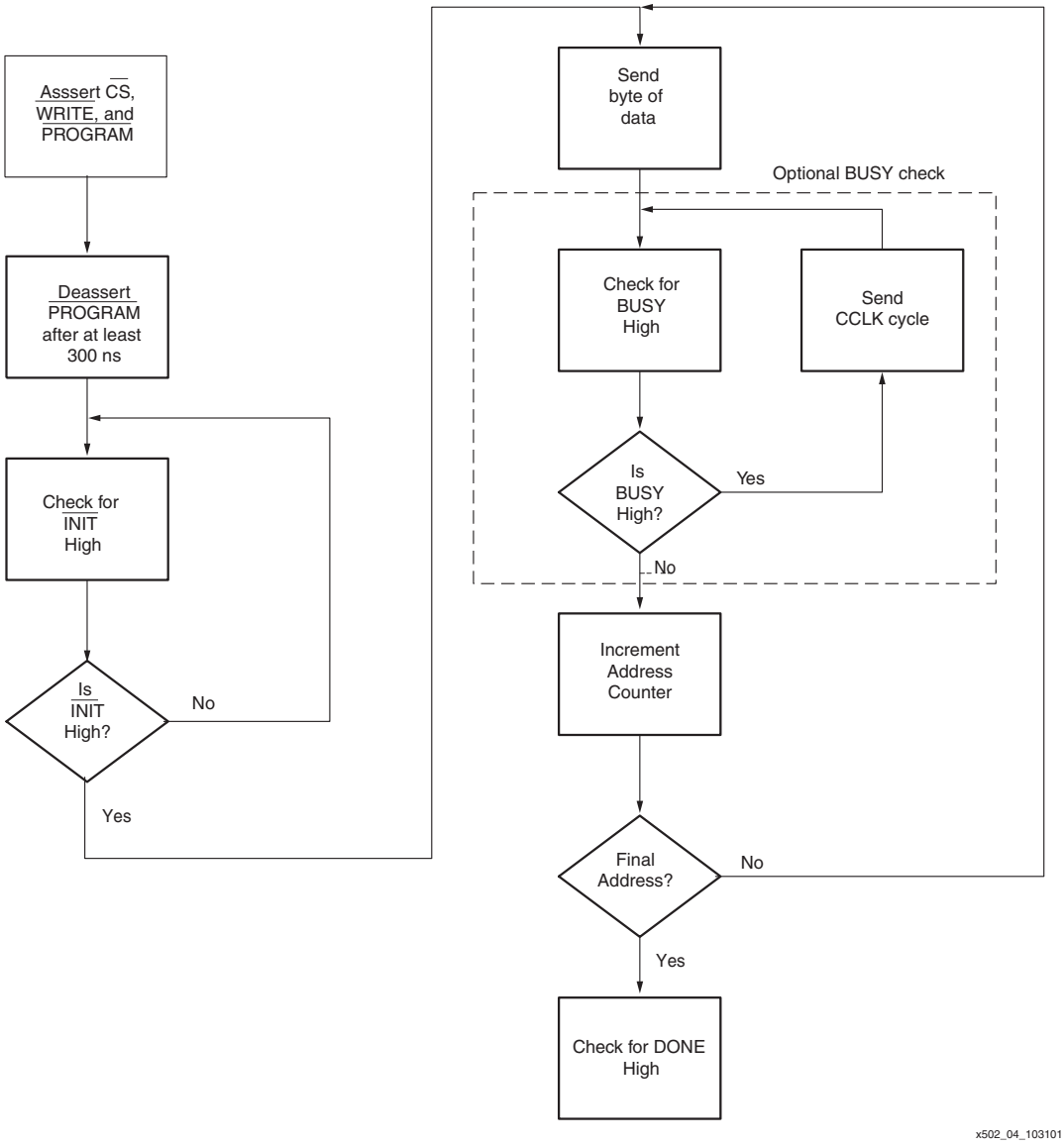

Figure 8: **Select MAP Configuration Flow Diagram**

If BUSY is asserted (High), the BUSY check function continues to provide additional CCLK cycles until BUSY is deasserted.

Monitoring the FPGA Busy pin via the Busy_Check() function is optional. The maximum speed of SelectMAP configuration is different for different FPGA families. Check the appropriate data sheet for this specification. After all configuration data has been loaded, the *for* loop is terminated and the SelectMAP() function checks the CPLD Input Register for DONE assertion.

Slave Serial Hardware

Slave Serial configuration is accomplished by providing the Virtex FPGA with a Serial Clock and delivering a single data bit at every rising edge of the Serial clock until the final configuration bit has been sent. This section discusses a reference design that allows a Virtex FPGA to be configured in the Slave Serial mode through a combination of a microprocessor and a CPLD.

This design uses a 16-bit Flash memory to store the FPGA configuration data. Since each Flash address stores 16-bits of data, the microprocessor reads each address, retrieves the 16-bit data, and serializes it. This process continues until the final 16 bits of configuration data has been read and serialized. During this process, the microprocessor is also responsible for providing the FPGA with a configuration clock. A CPLD is used to decode both the serialized bitstream, as well as the configuration clock, and is ultimately responsible for toggling the Virtex FPGA's configuration pins. Refer to Figure 9 for a simple schematic.

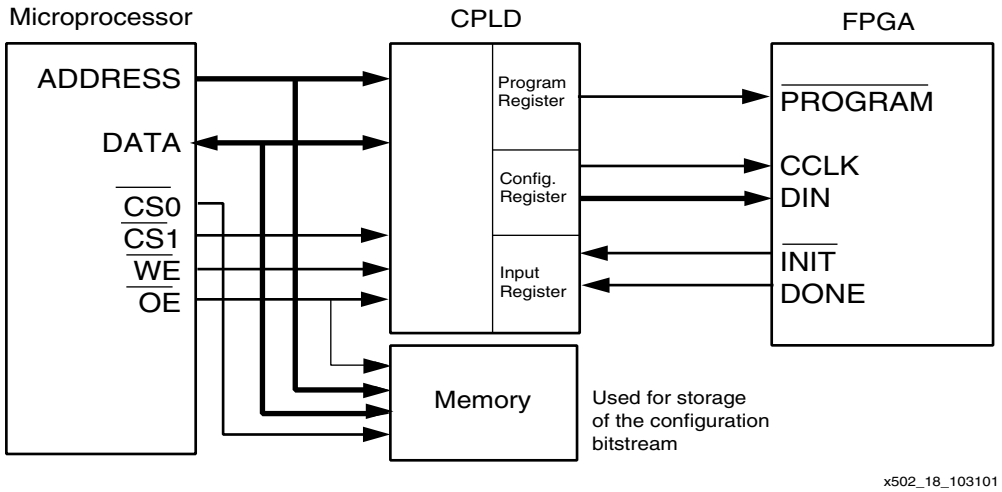

Figure 9: **Slave Serial Configuration**

This section discusses the functional microprocessor C code, as well as the functional VHDL code that can readily be targeted to a Xilinx 9500XL/XV, as well as a Xilinx CoolRunner XPLA3 CPLD.

CoolRunner CPLD

A Xilinx CoolRunner CPLD is used for glue logic between the microprocessor and the Xilinx Virtex FPGA. In the Slave-Serial mode of configuration, the CoolRunner device is responsible for driving the Virtex FPGA configuration pins, namely PROGRAM, CCLK, and DIN. The Virtex device's INIT and DONE pins are also registered by the CPLD, so that the status of these pins can be read by the microprocessor.

The CoolRunner is used to establish a synchronous interface between the microprocessor and the Xilinx FPGA. The interface is comprised of three registers: the Configuration Register, Program Register, and Input Register. These registers store FPGA configuration signals each time the processor does a read or write to the port. They support the set of signals required for Slave Serial mode (PROGRAM, DIN, CCLK, INIT and DONE). The Configuration Register and Program Register are write-only registers, while the Input Register is read-only. A detailed block diagram of the CPLD design is shown in Figure 10. The Register Address Map is shown in Table 5.

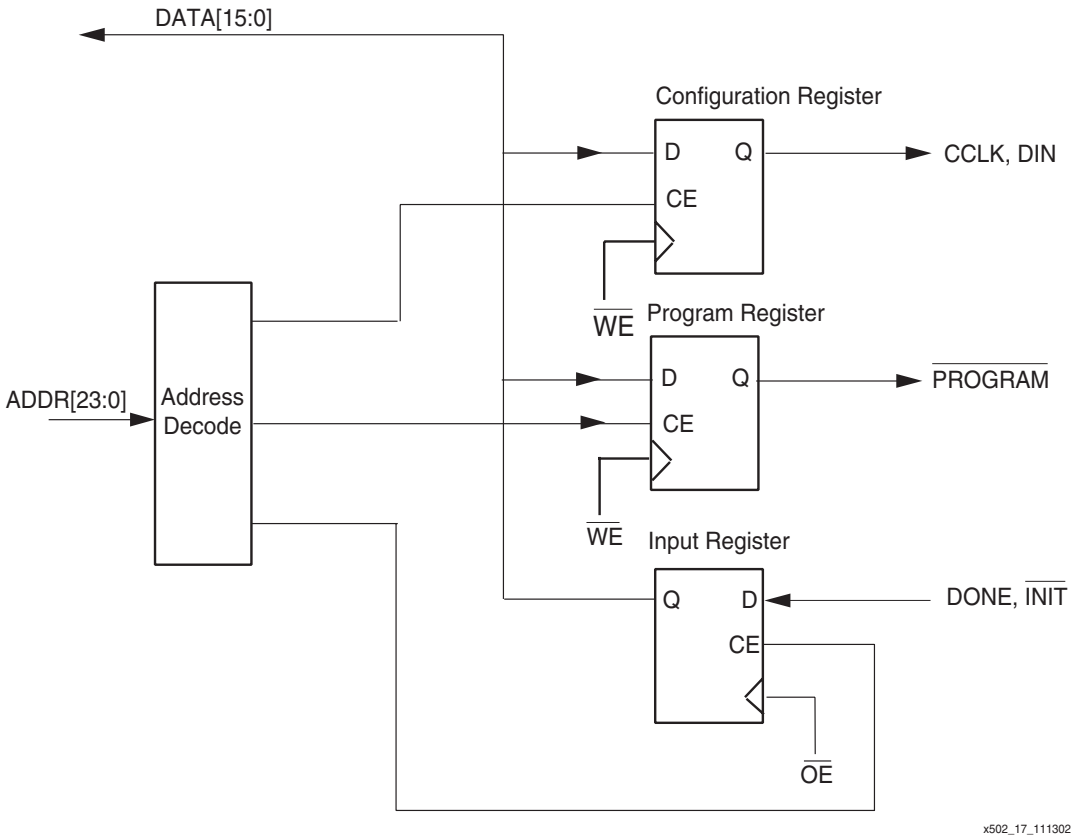

Figure 10: **CPLD Block Diagram**

Table 5: **Register Address Map**

Register Address A[23:0]	Register Name	Register Data							
		D7	D6	D5	D4	D3	D2	D1	D0
0x000000	Configuration Register	NA	NA	NA	NA	NA	NA	CCLK	DATA
0x000002	Program Register	NA	NA	NA	NA	NA	NA	NA	PROG
0x000004	Input Register	NA	NA	NA	NA	NA	NA	DONE	INIT

SlaveSerial.c

The C code discussed in this section allows a microprocessor to do the following:

- Read FPGA configuration data from Flash memory
- Provide a CCLK
- Serialize the configuration bitstream

Before continuing, it is important to note that different microprocessors have different memory mapped I/O structures, depending upon the system configuration. Therefore, while the majority of the C code listed here should be easily transportable, the user will have to modify the sections of code that command the microprocessor to perform a write operation on its data bus in order to match the memory map of the new microprocessor. This modification should not be difficult.

To allow for maximum readability, all C code shown in this reference design accesses the microprocessor address and data bus uniformly. This is accomplished through two functions, called IOWrite() and IORead(). IOWrite() writes a 16-bit word to a specified address location. IORead() retrieves a 16-bit result from a specified location. Refer to the **Microprocessor** section for this reference design's memory map. Refer to the **Appendix: Processor Specific I/O Function Calls** for the syntax of these two functions.

Within the **SlaveSerial.c** source file, two functions, SlaveSerial() and ShiftDataOut() accomplish configuration. To begin configuration, SlaveSerial() is called from within the main() function. SlaveSerial() is responsible for pulsing the program pin, checking for $\overline{\text{INIT}}$ deassertion, then calling the ShiftDataOut() function. A software flow diagram is shown in Figure 11.

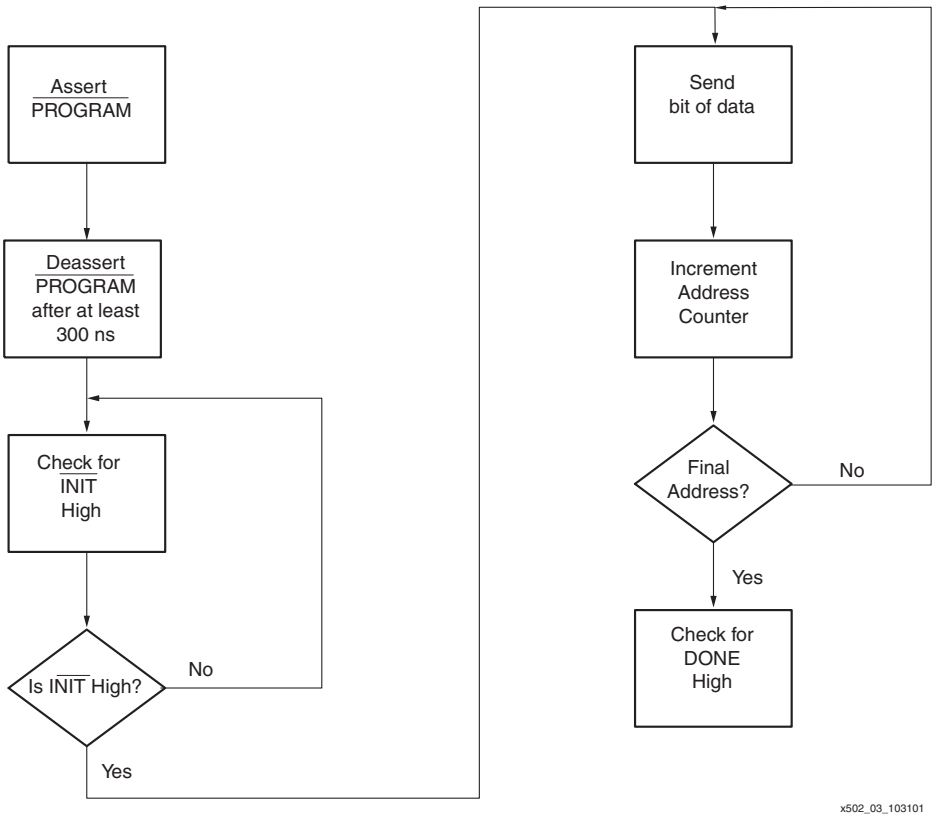

Figure 11: **Slave Serial Configuration Flow Diagram**

First, the processor writes to the Program Register to assert the $\overline{\text{PROGRAM}}$ pin, which will reset the FPGA. A loop may be required to assert the $\overline{\text{PROGRAM}}$ pin for the minimum required time (refer to the appropriate data sheet). Note that if the FPGA has just been powered up or reset through other means, this is not necessary. Once the $\overline{\text{PROGRAM}}$ bit has been asserted, the SlaveSerial() function checks the CPLD Input Register until $\overline{\text{INIT}}$ is High.

Then, the function enters a *for* loop that cycles through the Flash memory address range that contains the FPGA configuration data. In this example using a V50 device, the Flash address range is between 0x28000200 and 0x2801130A. For each address location, the 16-bit data is retrieved, and the ShiftDataOut() function is called to perform serialization and presentation of the data to the FPGA.

Notice that in the ShiftDataOut() function, two write cycles are needed each bit in the configuration file. One cycle is needed for driving CCLK Low and to present the next configuration bit. A second cycle is needed to drive CCLK High. After all configuration data has been loaded, the *for* loop is terminated, and the SlaveSerial() function checks the CPLD Input Register for DONE assertion.

Reference Design Files

The reference design files can be downloaded from the Xilinx ftp site at:

ftp://ftp.xilinx.com/pub/applications/xapp/xapp502.zip

Conclusion

This application note provided a configuration background as well as a description of two complete sets of reference designs that allow a Xilinx FPGA device to be configured through SelectMAP mode or through Slave Serial mode. While the microprocessor C code was targeted for a Motorola 68VZ328 Dragonball processor, it was written with portability in mind. Porting the code to another processor requires some effort, but all the design files have been documented extensively. Also, complete design files for a Xilinx CPLD design have been included to provide a synchronous interface between the microprocessor and the target Xilinx FPGA.

Appendix: Processor Specific I/O Function Calls

IOWrite(int Addr, int Data)

To use IOWrite(), an address and data must be passed to the function. In this reference design, the configuration, program, and input registers are located at 0x29000000, 0x29000002, and 0x29000004, respectively. These are the only three address values that are ever written to.

I/O Write Usage Example

Table 6: **I/O Write Usage Example**

IOWrite (0x29000000, 0x0011)	Write to CPLD Configuration Register, CCLK="1," DATA="1"
IOWrite (0x29000000, 0x0001)	Write to CPLD Configuration Register, CCLK="0," DATA="1"
IOWrite (0x29000002, 0x0001)	Write to CPLD Program Register, Program="1"

The timing diagram for a microprocessor write cycle is shown below in Figure 12

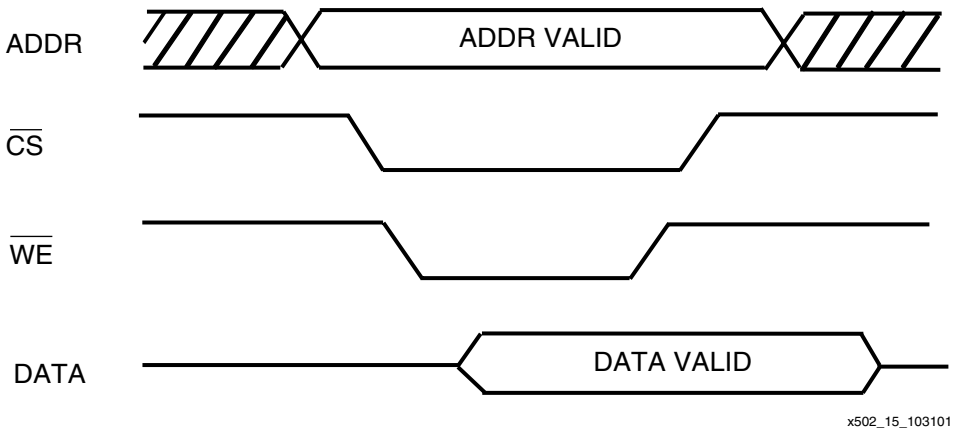

Figure 12: **Write Cycle Timing Diagram**

IORead(int Addr)

To use IORead(), an address must be provided. The 16-bit data value at that address is then returned. In this design, data is read from address ranges between 0x28000200 and 0x2801130A, as well as address location 0x29000004. The former is where the FPGA Configuration bitstream is stored, and the latter is the location of the Input Register.

I/O Read Usage Example

Table 7: **I/O Read Usage Example-**

IORead (0x28000200)	Retrieve first 16 bits of configuration data from Flash Memory
IOWrite (0x28000002)	Retrieve second set of configuration data from Flash Memory
IOWrite (0x2801130A)	Retrieve Final 16-bits of configuration data from Flash Memory
IOWrite (0x29000002)	Read from Input Register to determine values of DONE and $\overline{INIT}$

The timing diagram for a microprocessor read cycle is shown below in Figure 13

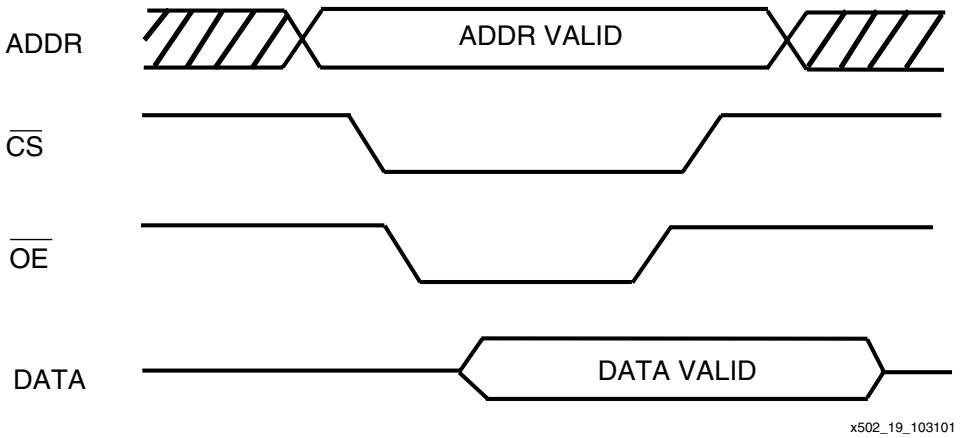

Figure 13: **Read Cycle Timing Diagram**

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/03/01	1.0	Initial Xilinx release.
01/08/02	1.1	Added Spartan-IIE to **Summary**.
06/10/02	1.2	Added "Virtex Series, Virtex-II Series Platform FPGAs, and Spartan-II Series" to **Summary**.
11/06/02	1.3	Updated Figure 5 and Figure 7.
11/13/02	1.4	Updated Figure 10.