XAPP800 (v1.0) April 20, 2004

Configuring Xilinx FPGAs with SPI Flash Memories Using CoolRunner-II CPLDs

Summary

This application note describes a method to configure Xilinx FPGAs, such as Spartan-IIE^{TM} and Spartan-3^{TM} FPGAs, using inexpensive small Serial Peripheral Interface (SPI) flash memories. These devices are physically small with low-pin count, small outline packages. Using a CoolRunner-II^{TM} CPLD, the SPI bus and protocol are converted to those which are used by the FPGA for configuration, specifically Master Serial configuration.

Introduction

SPI Flash based memories are inexpensive and are available in footprint-compatible, low-pin count, small outline packages that could reduce system cost if there was a way to interface the SPI bus and protocol to the configuration pins of the FPGA. Using a CoolRunner-II CPLD, this is now possible. The CPLD implements a simple state machine to read the SPI Flash memory using the SPI protocol, and then routes the serial data stream to the DIN pin of the FPGA. The cost of the CPLD is offset by the cost savings realized by using the SPI memory. Furthermore, the SPI Flash memory is available after configuration, offering cost-effective, random accessible, non-volatile data storage to the FPGA.

This application note discusses the design and provides reference source code, found at the end of this document. The CPLD source code contains commented VHDL generics that allow the designer to easily configure the design to implement the specific instruction set for several manufacturers of this type of memory, such as STMicroelectronics^{TM}, Atmel^{TM}, NexFlash^{TM}, PMC^{TM} and SST^{TM}. By default, the CPLD code interfaces to an STMicroelectronics M25P20 memory and assumes only one memory device is used. The source code will need modification to support multiple memory devices. The CPLD source code has been tested with the M25P20 implemented in hardware.

Software utilities are included in the downloadable zip file that allow the designer to perform Program, Erase, and Verify operations to the SPI Flash memory after it has been soldered to the PCB by using a cable and a PC. Currently, these software utilities only support the STMicroelectronics M25Pxx and compatible devices as described later.

SPI Basics

The SPI bus is a full duplex, synchronous serial data link. In other words, data is sent and received on the same clock edge, in a serial fashion. As such, there are 4 signals associated with SPI.

- SCK - Serial Clock
- MOSI - Master Out Slave In
- MISO - Master In Slave Out
- SS_n - Slave Select, active Low

Figure 1 displays these signals and a typical arrangement. This system is set up in a Master/Slave configuration where the Master clocks data out of the Master and into the Slave on the MOSI pin using SCK as the clock. During the same clock cycle, data is clocked out of the Slave and into the Master on the MISO pin, although this is done on the opposite clock edge than MOSI. Several Masters and Slaves can coexist on the same bus, and therefore a method of arbitration is necessary when more than one Master attempts to gain access to the bus. However, this reference design assumes that there is only one Master on the bus and therefore no arbitration is implemented. Since more than one Slave can exist on the bus, the Master needs a method to select which Slave is to be addressed during the transaction. The signal SS_n selects the specified Slave. This reference design assumes only one Slave is present and therefore must be modified to select additional Slave devices.

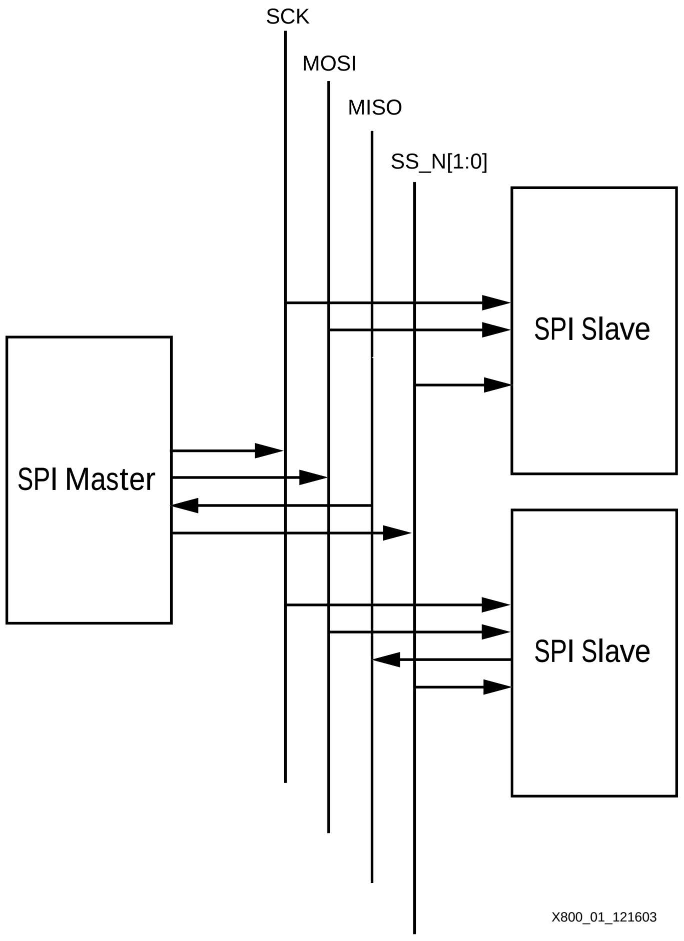

Figure 1: **Typical SPI System with a Single Master**

Clocking

SCK supports a variety of clock speeds, where this reference design assumes 20MHz in the simulation test bench. There are 4 types of clocking relationships the clock maintains with respect to the data. These are defined in the SPI Specification with two bits, CPOL and CPHA.

CPOL

CPOL dictates the idle state of the SCK. When CPOL is Low, the idle state of SCK is Low. Similarly, when CPOL is High, the idle state of SCK is High.

CPHA

CPHA indicates which clock edge the data is valid. When CPHA is Low, data is valid on the first edge of SCK, rising or falling. When CPHA is High, data is valid on the second edge of SCK. Whether the data is valid on a rising or falling edge is dependent upon the state of CPOL.

To illustrate the relationship between CPOL and CPHA, Figure 2 and Figure 3 are provided for reference, where Figure 2 displays the data transfer when CPHA is Low, and Figure 3 when CPHA is High.

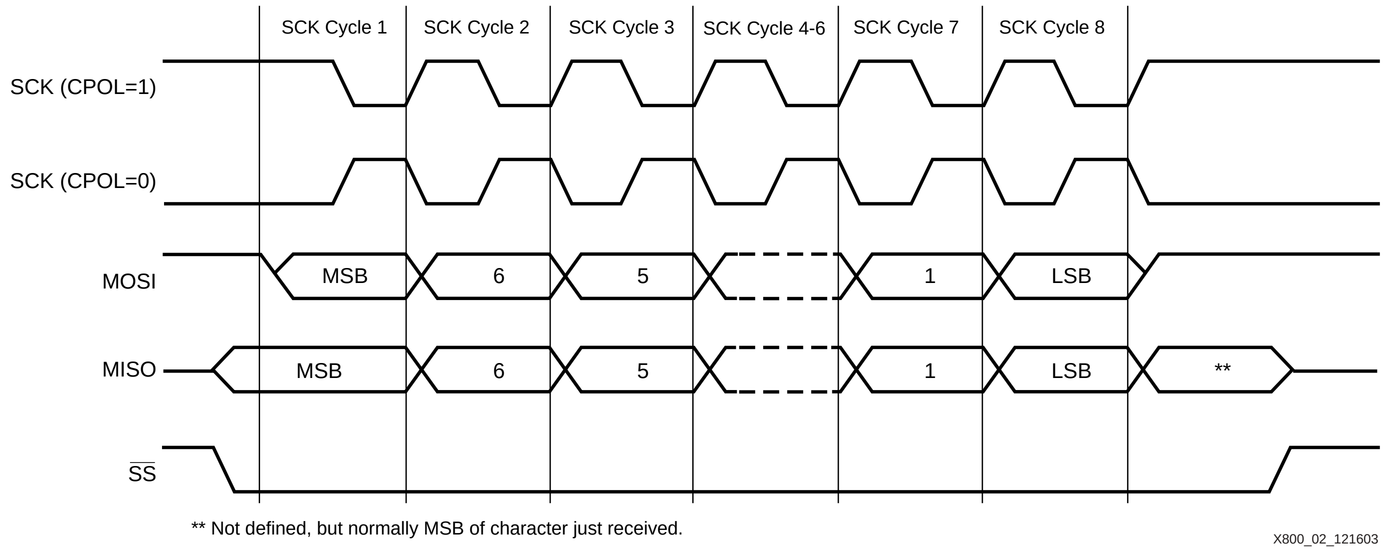

Figure 2: **Data Transfer on the SPI Bus with CPHA=0**

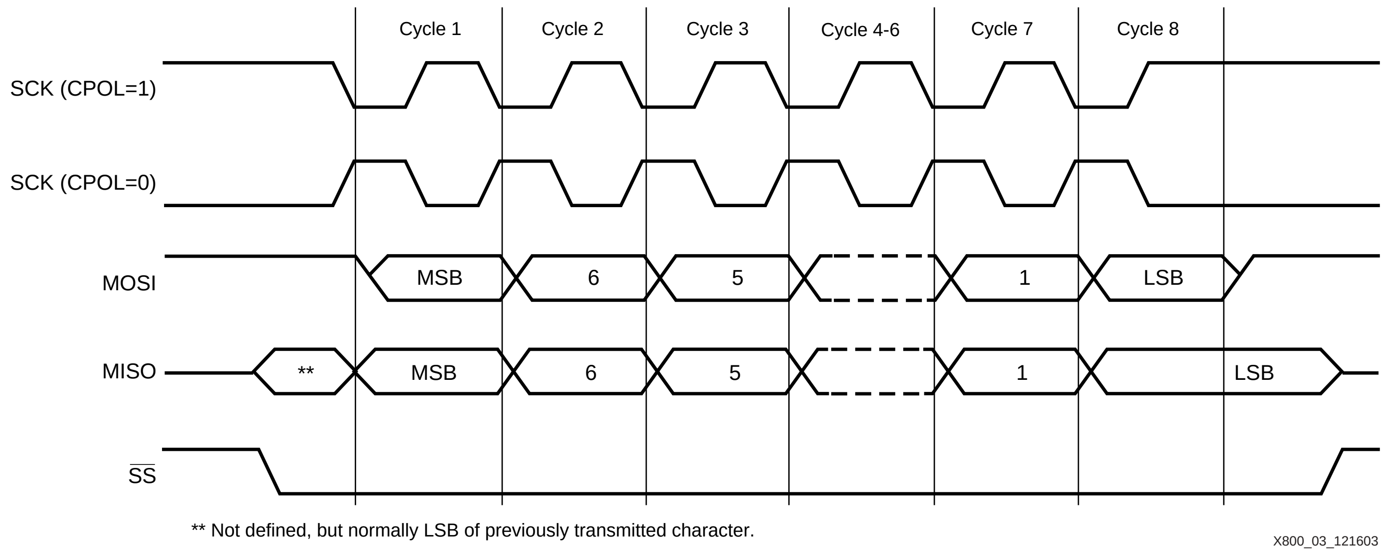

Figure 3: **Data Transfer on the SPI Bus with CPHA=1**

Data is clocked out of the device (Master or Slave) on one edge of the period, whereas data is clocked in on the next opposite edge of the period. For example, if data is clocked out of the Slave on the rising edge of SCK, then that data is clocked into the Master on the next falling edge of SCK. Simultaneously, the same sequence of data transfer occurs from the Master to the Slave. This full duplex data transfer can be viewed as a circular 16 bit shift register with 8 bits in the Master and 8 bits in the slave. The data simply trades places between the devices.

The SPI Specification does not describe a protocol for the data on the bus. Therefore, it is up to the designer to specify the protocol of the data transfer. SPI Flash memory devices define their particular protocol in the respective data sheets. Please consult the manufacturers data sheet for the appropriate protocol. Note that in particular, manufacturers implement different addressing schemes.

Timing relationships must follow those specified in both the SPI Specification and the SPI Flash memory data sheet.

System Design

This reference design implements an SPI interface between a Spartan-IIE FPGA and a STMicroelectronics M25P20, 2 Mbit SPI Flash memory. A working prototype has been built to test the reference design and uses a CoolRunner-II CPLD (XC2C32-4-VQ44) to perform the data transfer between the SPI bus and the FPGA configuration signals. Other Xilinx FPGAs may be used with this reference design, and the subsections below describe the unique details found using the Spartan-IIE and the Spartan-3 FPGAs.

Note: This design was tested on the STMicroelectronics M25P20, but will work with other M25Pxx parts, including M25P05, M25P10, M25P40, and M25P80.

Figure 4 shows how this reference design should be implemented in a system with an SPI Flash memory. This implementation provides a method of configuring the FPGA using an SPI based memory upon power up, or after the FPGA requests a reconfiguration. Once the FPGA has been configured, the FPGA may access the SPI memory as user space. This reference design provides for three methods of programming the SPI memory while mounted to the PCB. All of these scenarios are discussed in the subsections below.

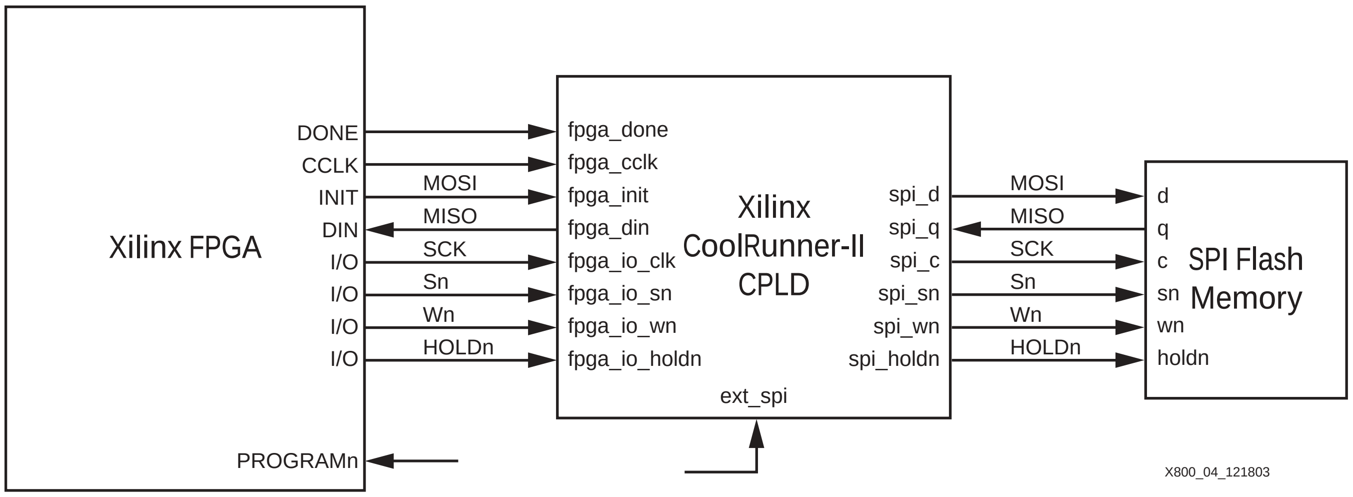

Figure 4: **SPI Memory Interface to a Xilinx FPGA**

FPGA Configuration

Upon power up, the FPGA requests configuration data using Master Serial mode in the usual manner. Both the DONE and INIT pins go Low, followed by the INIT pin going High indicating the start of configuration is requested. The state machine in the CPLD recognizes this event and begins to set up the SPI memory for a read operation.

The operation begins by bringing spi_sn Low, followed by passing CCLK through to spi_c and issuing the FAST_READ instruction. Once the instruction has been sent, the starting address is immediately transmitted followed by dummy bits. When the prescribed number of dummy bits has been sent to the memory, configuration data appears on the q pin of the memory. The state machine in the CPLD recognizes this event and passes the data through to the DIN pin of the FPGA. This data transfer continues until the DONE pin goes High. At this point, the CPLD does one of two things to its I/Os depending upon a VHDL generic setting in the source code. These two options are described in the next subsection.

After the FPGA has been successfully configured, the CPLD monitors the DONE pin to determine when the FPGA requests a reconfiguration cycle. When the DONE pin goes Low and the INIT pin cycles Low then High, the CPLD begins the SPI transaction again and presents the data to the FPGA in a Master Serial mode. FPGA reconfiguration is initiated by applying a Low value to the PROGRAMn pin of the FPGA.

SPI Memory as FPGA User Space

This reference design allows the designer to specify if the CPLD will be used as a simple buffer after FPGA configuration, or if the CPLD becomes electrically invisible to the FPGA I/Os after configuration has completed. The way the designer specifies either mode is by modifying the VHDL generic cpld_buffer found in the VHDL file spi_cpld.vhd. If this generic constant is set to a '1', the CPLD will become a buffer between the FPGA I/Os and the SPI memory I/Os. This allows the FPGA to use the SPI memory as a user space post-configuration. Alternately, if the generic cpld_buffer is set to a '0', the CPLD will become invisible to the FPGA I/Os so that it does not interfere with future transactions on these I/Os. The CPLD I/Os in this case go to a High-Z mode. It is up to the user do specify weak pull-ups on the CPLD I/Os as needed in the particular system.

Although the SPI Flash memory is mainly used to store FPGA configuration data, the memory can also be used as working memory space by the FPGA one it has been configured. As discussed earlier, the CPLD becomes a simple buffer and passes all SPI bus signals between the FPGA and the SPI memory post FPGA configuration. These signals can be then driven by the FPGA as needed to conduct memory transactions. The SPI bus includes the typical SCK, MOSI, MISO and SSn signals, but provided in this scenario are two other signals: Wn and HOLDn. Some memories contain these signals which provide additional control of the memory. This reference design is based on the STMicroelectronics M25P20 where Wn is a write protect signal, active Low. HOLDn is an active Low hold signal used to pause any serial communications with the device without actually deselecting the device on the SSn pin.

The FPGA can be set up with a design that uses this memory as working space. This may be in the form of a state machine or a microcontroller, etc. This allows the FPGA to not only use the unused memory as user storage, but can also be used to overwrite the configuration data in the SPI memory. For example, if SPI Flash stored both the FPGA configuration data and the application code for an embedded controller, then it would be possible to update both hardware and software by writing a new image to the Flash. Alternatively, the configuration bits in the memory could be overwritten or erased by the FPGA in a security type application.

Programming the SPI Memory

There are 3 primary methods of programming the SPI memory as described below.

3rd Party Programmer

The most obvious method of programming the SPI memory is using a 3rd party programmer as would be done with PROMs. The advantage with this method is that mass production programming is possible. But the disadvantage is that it will be difficult to reprogram the device on the PCB unless the FPGA or some other controller on the SPI bus is set up to perform this operation.

JTAG Chain

The CoolRunner-II CPLD has JTAG test capabilities which include the standard PRELOAD and EXTEST commands. Using these commands, it is possible to drive and sample the pins of the CPLD with the JTAG chain and thereby stimulate the pins of the SPI memory via the traces routed on the PCB. This method is shown in Figure 5. Using a tool that understands the JTAG protocol as well as converts the applied data to SPI bus relationships, the SPI memory can be programmed via the JTAG chain of the CPLD

The advantage with this is by adding minimal hardware to the board, the memory can be quickly programmed. The added benefit is that the user now has access to the CPLD for configuration as well.

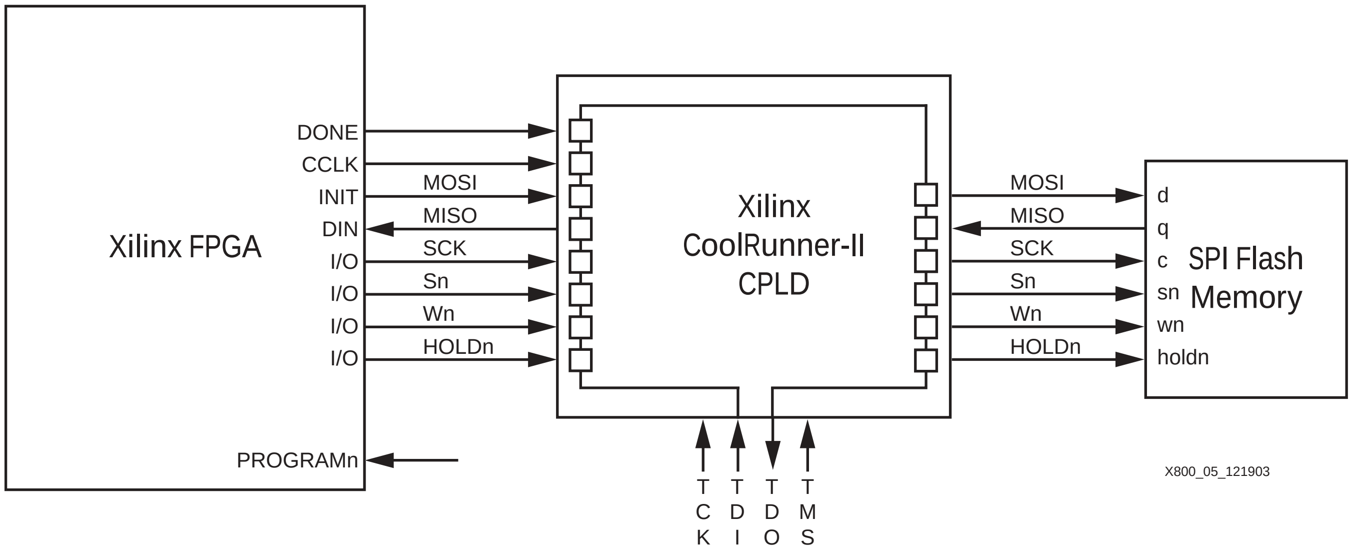

Figure 5: **Programming SPI Flash with the JTAG Chain**

Directly

It is also possible to program the SPI Flash Memory directly on the PCB by adding a cable connector to the PCB which taps into the SPI bus as shown in Figure 6. This method requires a cable and minimal hardware to connect to the bus. In addition, software must be used on a PC that can supply the data via an SPI protocol to the board via the PC Parallel Port. This software is available as a downloadable zip file from a link at the end of this application note. These tools are described in a following section.

To facilitate this method, the CPLD must have its I/Os in a high impedance state with no weak pullup so that it does not interfere with SPI bus transactions during the programming operation. A signal has been added to one CPLD I/O which allows for the CPLD to go into a High-Z condition when needed. The pin that controls this function is called ext_spi and is shown in Figure 4. When ext_spi is brought High by an external signal, such as the cable used to program the memory, the CPLD I/Os that interface to the memory as well as those to the FPGA will go into a High-Z condition. Note that since the software utility bundled with this reference design does not drive spi_holdn or spi_wn, the CPLD will drive these pins High when ext_spi is

High. Placing a Low value on the ext_spi pin allows the CPLD state machine to drive the I/Os when required by the FPGA.

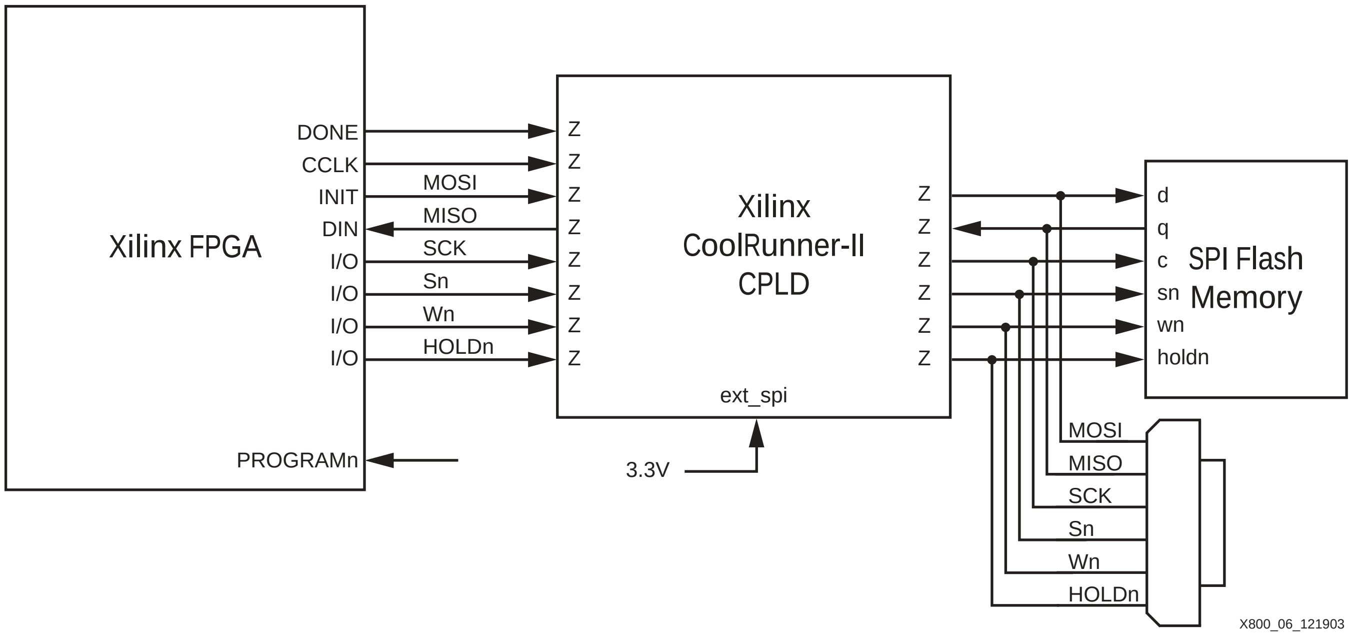

Figure 6: **Programming SPI Flash with Direct Cable**

Spartan-IIE

Although this reference design targets a CoolRunner-II CPLD (XC2C32-4-VQ44), the device selection can be substituted for any other Xilnx CPLD. In addition, this reference design targeted a Spartan-IIE FPGA (XC2S200E PQ208) for testing purposes. Other Xilinx FPGAs may be used with this reference design.

The most important thing to consider when using the Spartan-IIE is the I/O standard selected on the I/Os. SPI memories are typically 3.3V driven devices. Although the CoolRunner-II CPLDs have I/O banking, the 32 macrocell device (XC2C32) only has one bank. Therefore, the FPGA and the memory must match I/O voltage capabilities. The CPLD, in this case, must be configured with LVCMOS33 I/O functionality to properly interface to the SPI bus. Consequently, the Spartan-IIE must be configured with 3.3V LVTTL I/Os where the Vcco pins of Bank 2 and Bank 3 are supplied with 3.3V.

Spartan-3

As an example of another Xilinx FPGA that can be configured using this method, the Spartan-3 FPGA can be set up with the CoolRunner-II CPLD for SPI transactions. The Spartan-3 dual purpose configuration pins, in this case DIN and INITn, should be powered to 3.3V via V_{CCO_4} to interface to the 3.3V I/Os of the CPLD. Spartan-3 dedicated configuration pins, in this case DONE and CCLK, are set up for 2.5V signaling and therefore require 2.5V supplied to V_{CCAUX}. To successfully interface to the 3.3V I/Os of the CPLD, modifications are necessary to the interface between the FPGA and the CPLD as described in the Spartan-3 Functional Description Data Sheet, DS099-2. According to this data sheet, it is advisable to add a pullup resistors to both DONE and CCLK which are terminated to 3.3V. Both of these pins are set up as outputs in the Master Serial configuration mode. The pullup resistors must be selected to provide proper edge rates to the CPLD as described in the CoolRunner-II CPLD Data Sheet. In addition, the power supply for V_{CCAUX} must be able to tolerate reverse currents.

Again, a different Xilinx CPLD may be selected for the interface to acquire a different voltage match between these components.

Software Utilities

Xilinx provides two command line utilities for storing an FPGA bitstream into an SPI Flash device:

- xmcsutil: reverse bytes in a Xilinx-generated bitstream file
- x_spi: erase/program/verify/read SPI device

These two utilities are described in this section. To obtain a copy of these tools, please follow the link at the end of this application note. These files must be extracted to the path specified by the XILINX environment variable.

These tools have been tested on Microsoft Windows^{TM} NT 4.0, Microsoft Windows^{TM} 2000, and Microsoft Windows^{TM} XP. These tools have not been tested for use in Microsoft Windows^{TM} 95, Microsoft Windows^{TM} 98, or Microsoft Windows^{TM} ME, but will likely work with these operating systems.

xmcsutil

The FPGA configuration data bytes in a Xilinx-generated MCS file must be reversed before programming into an SPI Flash. This is required for the FPGA configuration data bits to come out of the memory in the correct order.

To reverse the bytes, run xmcsutil with the following options:

```
xmcsutil -i <MCS file> -o <new MCS file> -8
```

where:

- -i <MCS file> = FPGA bitstream in MCS format
- -o <new MCS file> = output filename for new MCS file
- -8 = reverse bytes in the input MCS file

Once modified with xmcsutil, the new MCS file can be programmed into the SPI Flash device using a device programmer or with a PC using a Xilinx JTAG cable (Parallel Cable-III or Parallel Cable-IV). The latter method is described in the next section.

x_spi

As discussed earlier, the SPI Flash device may be programmed on the PCB directly by placing the CPLD into a High-Z state. This is done by asserting the ext_spi pin High and driving the SPI flash pins directly from a PC. The x_spi tool is a PC command line utility for programming these devices in this direct manner. Currently the x_spi tool supports the STMicroelectronics SPI Flash devices as listed below:

- M25P05
- M25P10
- M25P20
- M25P40
- M25P80

It can also be used to program other STMicroelectronics SPI Flash devices using the same erase, program, and read opcodes. Other supported vendors include NexFlash Technologies, Inc., Programmable Microelectronics Corporation, Inc. (PMC), Atmel and Silicon Storage Technology, Inc. (SST). For example, PMC and NexFlash supply the following devices supported by this reference design:

- Pm25LV512
- Pm25LV010
- NX25P10
- NX25P20
- NX25P40
- NX25P80

The x_spi tool works with the Parallel Cable-III (PC-III) or Parallel Cable-IV (PC-IV) JTAG cables with the following JTAG-to-SPI pin connections:

JTAG	SPI Flash
TCK	C
TMS	S
TDI	D
TDO	Q

The HOLDn and Wn signals are driven Low by the CPLD during the programming process when ext_spi is held High.

The following operations are supported by x_spi:

- Erase
- Program only
- Program and verify
- Verify only
- Read
- Blank Check

The below sections describe how to implement these operations in detail. Other options are available but are not described here. To learn more about these options, simply type *x_spi* with no options at the command prompt to view a complete list of all options.

Erase

An erase makes use of the -spi_e switch. At the command line, type the following to erase the SPI Flash device:

```
x_spi -spi_e
```

Program Only

A program only function requires the -spi_p switch. Using an erased device, type the following at a command prompt to perform a program only function on the SPI Flash device.

```
x_spi -spi_p -i system.mcs
```

Notice that the -i switch is used to specify the file that contains the FPGA bitstream to be programmed into the SPI Flash device. In this example, system.mcs is the file containing the FPGA bitstream.

Program and Verify

Program and verify operations are specified by using the -spi_pv switch. After an erase, type the following at the command line to program and verify the SPI Flash memory with the FPGA bitstream specified after the -i switch. In this case we specify the input file named system.mcs:

```
x_spi -spi_pv -i system.mcs -o verify.txt
```

Verify errors will be written to a file with a name as specified after the -o switch, in this case verify.txt is the specified output file.

Verify Only

A verify only operation uses the -spi_v switch. To verify the SPI Flash device, specify the file name for comparison using the -i switch as shown in the below example. In this case compare the contents of the SPI Flash device with the bitstream contained in the file named `system.mcs`.

```
x_spi -spi_v -i system.mcs -o verify.txt
```

If any errors occur, they will appear in a file named as specified after the -o switch. In this case, the output file is named verify.txt.

Read

The read function requires the -spi_r switch. To read N kilobytes of data from the SPI Flash device, type the following at the command prompt:

```
x_spi -spi_r 256 1 -o readback.mcs
```

Where:

- 256 = read 256 kilobytes (or 2 Mbits) starting from location 0
- 1 = write data to output file in MCS format

Note that the bits read out of the SPI Flash device will be contained in a file with a name as specified after the -o switch. In this case, the readback.mcs file will contain the bits read from the SPI Flash device.

Blank Check

The blank check function requires the -spi_b switch. This function simply verifies the device is truly erased by reading N kilobytes of data from the SPI Flash device. Type the following at the command prompt to perform a blank check function:

```
x_spi -spi_b 256 -o blank.txt
```

Where

- 256 = blank check 256 kilobytes (or 2Mbits) starting from location 0

Note that the results of the blank check are sent to an output file as specified by the -o switch. In this case, the output file is named blank.txt

CPLD Design

This reference design, as implemented in the XC2C32-4-PC44, can obtain speeds of over 300 MHz. However, the limiting factor with speed is the SPI Flash memory which is typically limited to 25 MHz. The user must specify FPGA configuration speed by selecting the configuration speed using BitGen found in Xilinx ISE software. Note that the fastest FPGA configuration speed for Xilinx FPGAs is faster than the maximum speed of most SPI Flash memories.

State Machine

The main functionality of this reference design focuses around one state machine, found in processes named statem_comb and statem_reg. The following discussion references Figure 7.

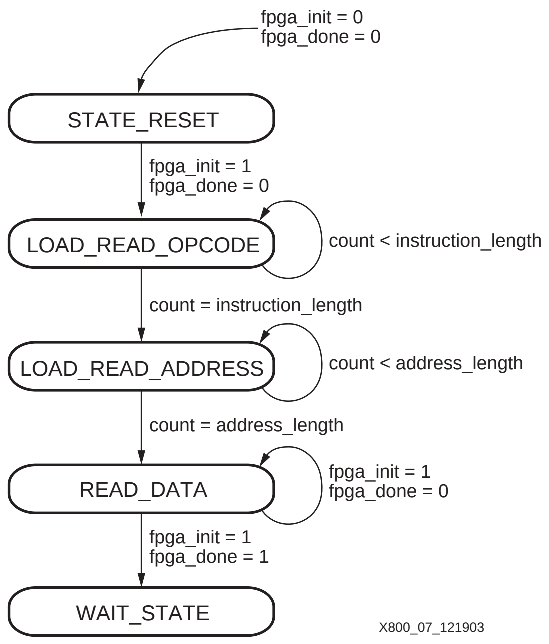

Figure 7: **State Machine**

STATE_RESET State

The state machine is clocked by fpga_cclk which arrives directly from the FPGA CCLK pin. When the fpga_init (INITn) and fpga_done (DONE) pins go Low, the state machine is reset to the STATE_RESET state.

While in this state, the variable bit width counter is reset. This counter tracks the number of instruction bits, address bits, and dummy bits sent to the SPI bus in subsequent states.

The STATE_RESET state also applies High values to fpga_din (DIN) to provide benign data to the FPGA DIN pin. This is necessary because there is a finite amount of time that the state machine must set up the SPI memory before data is read out of the memory. Meanwhile, the FPGA Master Serial configuration mode is expecting data to arrive on DIN once INITn goes High. The entire state machine waits to submit valid data to DIN until it reaches the READ_DATA state. To avoid erroneous data being applied to the DIN pin during this initial phase, harmless data in the form of 1's must be sent to DIN. During this time frame, the FPGA simply waits for the synchronization word to arrive, which indicates the beginning of valid configuration data.

Once in the STATE_RESET state, the state machine waits for the condition where fpga_init goes High, and fpga_done is still Low. When this occurs, the state machine progresses to the LOAD_READ_OPCODE state.

LOAD_READ_OPCODE State

This state sends the READ or FAST_READ instruction code to the SPI memory to set up the device for a read operation. The instruction is stored in the generic variable READ_INSTRUCTION and can be customized by the user for the specific SPI memory. Sample instruction codes are provided in the generic statement for some common memories.

In this state, spi_sn is brought Low to initiate communication with the SPI memory as shown in Figure 2 and Figure 3. At the same time, the counter is enabled and begins tracking the number of instruction bits sent to the SPI memory.

While in this state, instruction code data is placed on the spi_d pin and is shifted out on every clock edge, while the counter tracks the number of instruction code bits sent. The data and spi_sn signal occur on the opposite edge of fpga_cclk than the state machine to retain proper timing.

Simultaneously, 1's continue to be applied to the DIN pin of the FPGA via the fpga_din output signal.

When the counter reaches the bit length of the instruction as specified by the generic variable INSTRUCTION_LENGTH, the state machine transitions to the next state LOAD_READ_ADDRESS.

LOAD_READ_ADDRESS State

The purpose of this state is to shift out the starting address of the data located in the SPI memory followed by shifting out dummy bits as required by the particular memory. As such, spi_sn is held Low as it was in the previous state and the counter continues to increment without a reset.

The counter tracks the number of address bits shifted as well as the number of dummy bits. With this in mind, the terminal count is the sum of the number of address bits, dummy bits, and instruction bits (since the counter is not reset). Therefore, the user must set the size of variable width counter to a value which corresponds to the sum of these three values. This is done by adjusting the generic variable COUNTER_WIDTH.

Address bits are shifted out serially on the spi_d pin. This address is defined by the variable START_ADDRESS generic. The ADDRESS_LENGTH generic variable should be specified by the user to match the number of bits in the starting address. Once the starting address is shifted out, dummy bits are shifted out and are zeros in this reference design. The user needs to specify the number of dummy bits in the DUMMY_LENGTH generic variable.

During this process, the DIN pin of the FPGA continues to receive 1's as they are applied via the fpga_din output.

When the counter reaches terminal count, which again is defined as the sum of instruction bits, address bits and dummy bits, the state machine transitions to the READ_DATA state.

READ_DATA State

Once the SPI Memory is setup with the instruction, starting address, and any necessary dummy bits, data appears on the SPI Flash's q pin. At this point the state machine is in the READ_DATA state which simply applies a Low value on spi_sn and passes the data from the memory straight through the CPLD to the FPGA DIN pin via fpga_din. The pin spi_sn is held Low to continue receiving data from the memory.

When the FPGA's DONE pin goes High indicating the FPGA has completed receiving needed configuration bits, the state machine transitions to the WAIT_STATE state.

WAIT_STATE State

The controller waits in this state indefinitely until the FPGA's INIT and DONE pins go Low as sensed on the fpga_init and fpga_done pins respectively. This indicates a reconfiguration request by the FPGA, and the state machine moves to the RESET state to begin the cycle again.

CoolRunner-II Implementation

This reference design has targeted the CoolRunner-II XC2C32-4-VQ44 device. Using this device, 26 macrocells are used, which corresponds to 81% utilization. This utilization level allows for minor changes to the design such as adding support for multiple SPI Memory devices without exceeding the device resource limits.

Other Xilinx CPLDs may be used in lieu of the XC2C32-4-VQ44 device, allowing for a flexible selection of system voltage ranges and feature choices.

VHDL Code and Software Download

VHDL Code Disclaimer

This application note supports the STMicroelectronics M25P-series SPI Flash memories and directly compatible versions from NexFlash and PMC. With small modifications, this same CPLD design likely supports SPI Flash memories from Atmel Corporation and Silicon Storage Technology, Inc.

Xilinx Software License

results for your own competing software development activities, without the prior written permission of Xilinx. You may not make any copies of the Software, except to the extent necessary to be used on separate non-simultaneous computers as permitted herein, and one copy of the Software in machine-readable form for backup purposes only. You must reproduce on each copy of the Software the copyright and any other proprietary legends that were on the original copy of the Software.

This Software may be time-based to permit you to evaluate certain products only for a predetermined time. After the predetermined time expires, the Software will be automatically disabled. Xilinx requests that you inform users that the Software will time-out after a predetermined time and that a full license can be obtained thereafter. XILINX MAKES NO REPRESENTATION OR WARRANTY THAT TIME-BASED ROUTINE CANNOT BE DISABLED; AND YOU ACKNOWLEDGE THAT YOU WILL BEAR ALL RISK AND LIABILITY IF THE TIME-BASED ROUTINE IS UNLAWFULLY DISABLED TO ALLOW THE UNLICENSED USE OF THE SOFTWARE.

3. LIMITED WARRANTY AND DISCLAIMER. SUBJECT TO APPLICABLE LAWS: (1) XILINX'S AND ITS LICENSORS' ENTIRE LIABILITY TO YOU AND YOUR EXCLUSIVE REMEDY UNDER THIS WARRANTY WILL BE FOR XILINX, AT ITS OPTION, AFTER RETURN OF THE DEFECTIVE SOFTWARE MEDIA, TO EITHER REPLACE SUCH MEDIA OR TO REFUND THE PURCHASE PRICE PAID THEREFOR AND TERMINATE THIS LICENSE; (2) EXCEPT FOR THE ABOVE EXPRESS LIMITED WARRANTY, THE SOFTWARE IS PROVIDED TO YOU AS IS; (3) XILINX AND ITS LICENSORS MAKE AND YOU RECEIVE NO OTHER WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX AND ITS LICENSORS SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. XILINX DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, XILINX DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY OR OTHERWISE. XILINX CANNOT PROVIDE TECHNICAL SUPPORT FOR THE SOFTWARE AND WILL NOT BE RESPONSIBLE FOR ANY CONSEQUENCES OF THE USE OF THE SOFTWARE.
4. LIMITATION OF LIABILITY. SUBJECT TO APPLICABLE LAWS: (1) IN NO EVENT WILL XILINX OR ITS LICENSORS BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, COST OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING FROM THE USE OR OPERATION OF THE SOFTWARE OR ACCOMPANYING DOCUMENTATION, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY; (2) THIS LIMITATION WILL APPLY EVEN IF XILINX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE; (3) THIS LIMITATION SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDIES HEREIN; AND (4) XILINX'S AND ITS LICENSORS' LIMITATIONS OF LIABILITIES ARE NOT CUMULATIVE. THE LIMITATIONS OF REMEDIES AND DAMAGES IN THIS SOFTWARE LICENSE SHALL NOT APPLY TO PERSONAL INJURY (INCLUDING DEATH) TO ANY PERSON CAUSED BY XILINX'S NEGLIGENCE AND ARE SUBJECT TO THE PROVISION SET OUT BELOW UNDER THE HEADING GOVERNING LAW.
5. Term & Termination. You may terminate this License at any time by destroying the Software and all copies thereof. This License will terminate immediately without notice from Xilinx if you fail to comply with any provision of this License. Upon termination you must destroy the Software and all copies thereof.

XAPP800 - http://www.xilinx.com/products/xaw/coolvhdlq.htm

STMicroelectronics Simulation Models

Included in the above reference design are the simulation models for the STMicroelectronics SPI Flash Memories. To obtain the latest simulation models as well as any assistance regarding these memories, please visit the STMicroelectronics website at:

http://www.st.com/stonline/products/families/memories/fl_ser/snvm_fo.htm

Conclusion

The CoolRunner-II CPLD is an excellent choice for configuring Xilinx FPGAs using SPI Flash memories since these CPLDs are low cost. Together with the low cost SPI Flash memory and the CoolRunner-II CPLD, overall configuration bit storage costs are reduced over conventional methods allowing for a more cost effective solution. Further, the low power characteristics of the CoolRunner-II CPLD provide a minimal impact to the power consumption of the total configuration bit storage solution.

Additional Information

CoolRunner-II Data Sheets, Application Notes, and White Papers

Device Packages

Online Store

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/20/04	1.0	Initial Xilinx release.