XAPP1188 (v1.0) September 23, 2014

FPGA Configuration from SPI Flash Memory using a Microprocessor

Authors: Simon Tam, Matt Nielson, and Mattics Phi

Summary

This application note describes a simple and efficient FPGA configuration method that utilizes a microprocessor to configure an FPGA device from a Serial Peripheral Interface (SPI) flash memory. This method reduces hardware components, board space, and costs. Reference hardware design and firmware are included to illustrate the methodology. You can download the Reference Design Files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Introduction

Systems containing a discrete embedded microprocessor and an FPGA are common. These kinds of systems comprise two typical subsystems, as shown in Figure 1.

A recommended method to utilize the microprocessor to configure an FPGA is described in XAPP583 [Ref 1]. This method is to store the user firmware as well as the configuration bitfile on a flash memory device attached to a microprocessor. The microprocessor reads the bitfile through an SPI interface and in turn sends out the bitstream to the FPGA via a slave Serial or Slave SelectMAP interface. This eliminates the need for an extra PROM for FPGA configuration. The block diagram is shown in Figure 2.

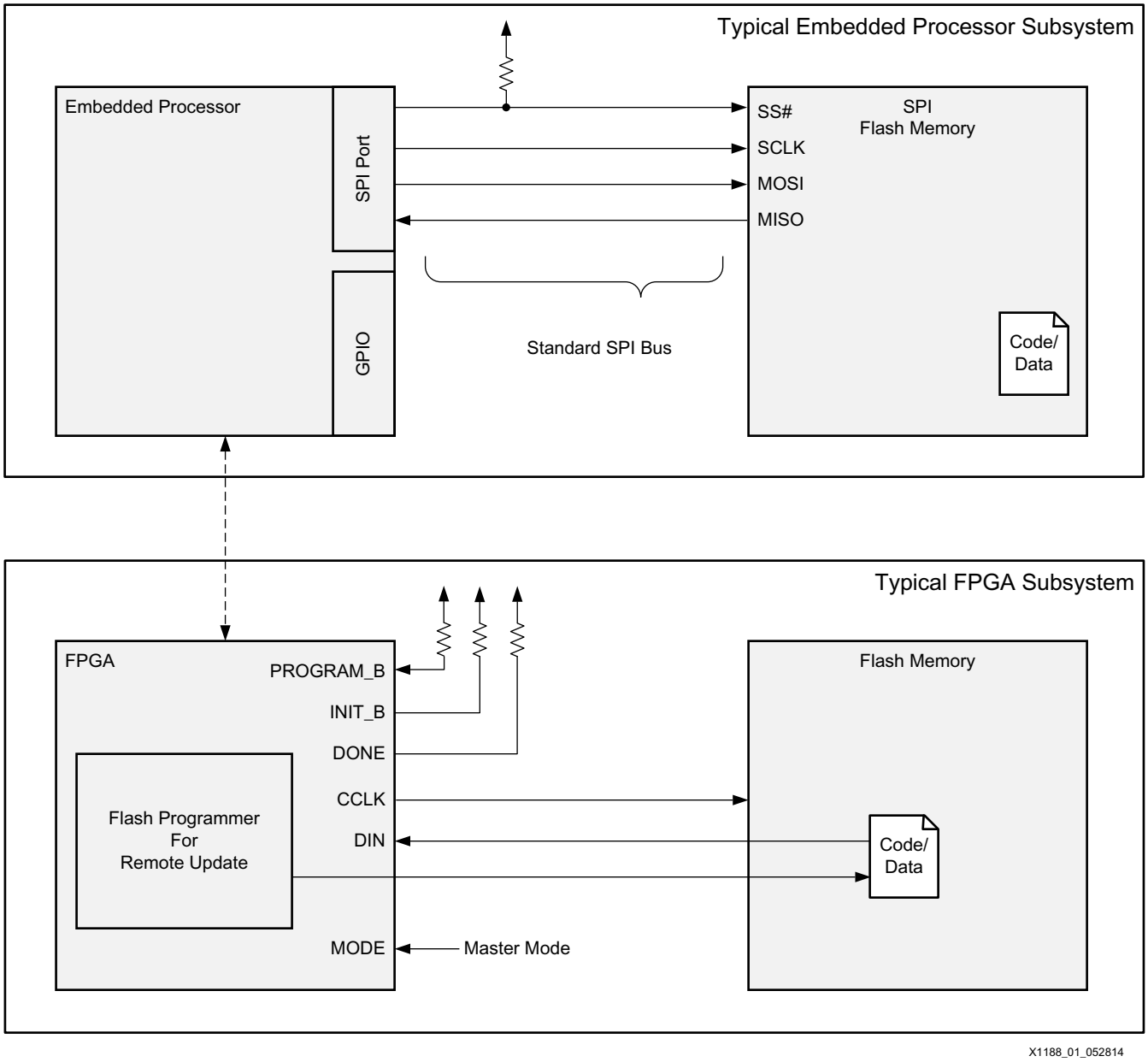

Figure 1: **Typical Embedded Microprocessor System with an FPGA**

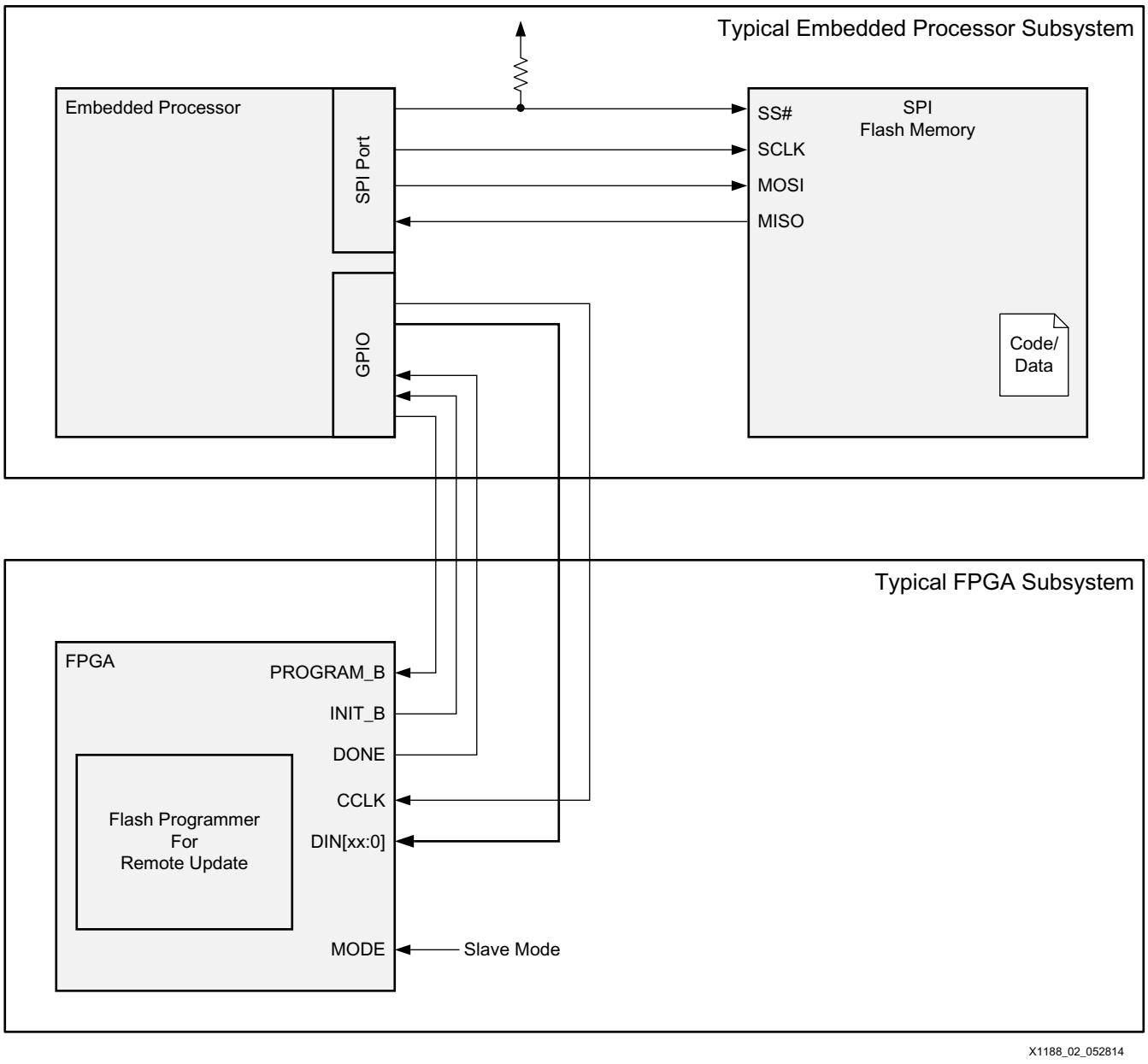

Figure 2: **XAPP583 Block Diagram**

The method described in this application note simplifies the configuration scheme further. It takes advantage of compatibility between the FPGA's slave serial configuration pins and the standard SPI bus signals. Figure 3 illustrates the connections between the microprocessor, flash memory, and the FPGA. The subsequent sections explain its design and operation.

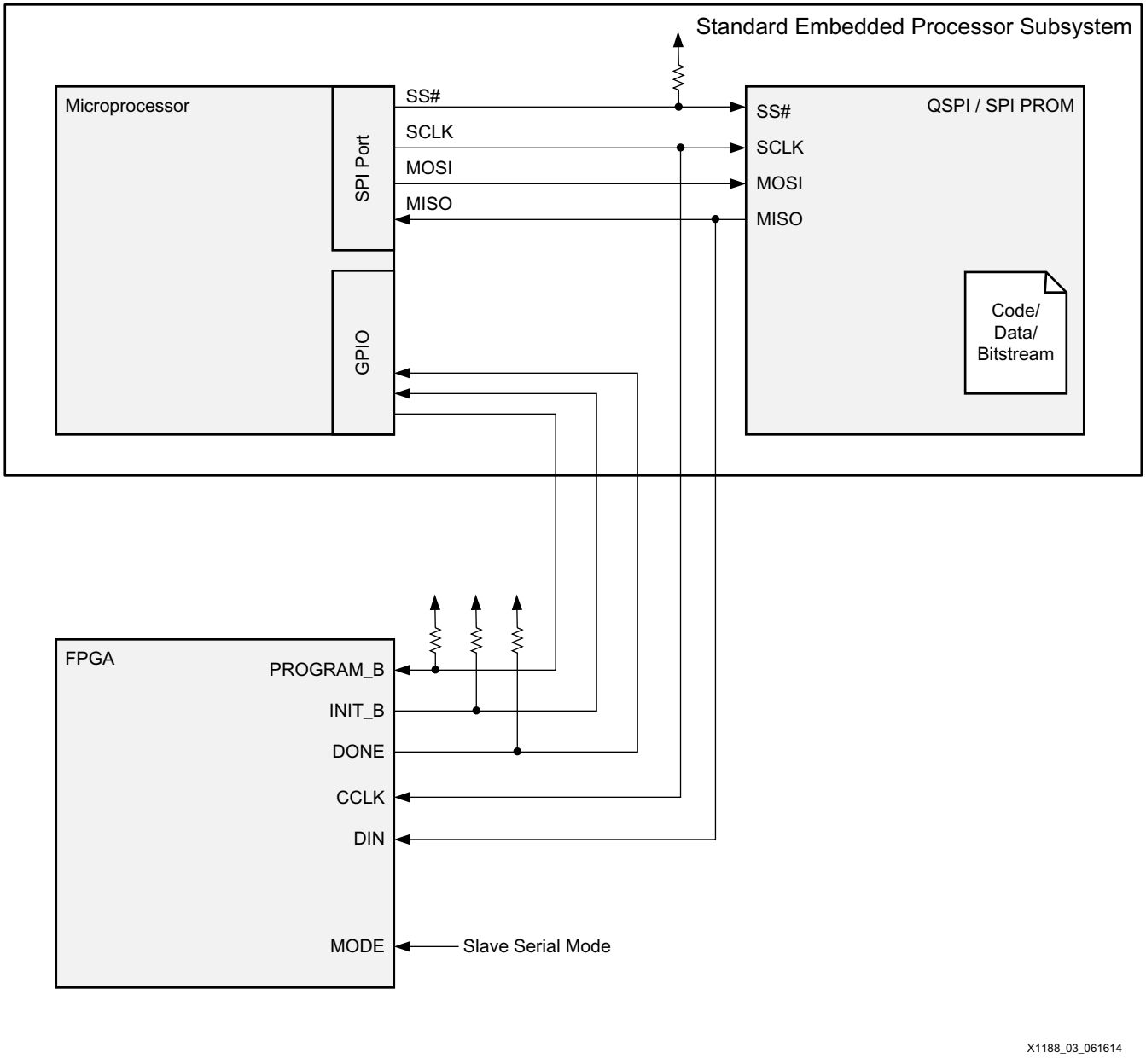

Figure 3: **XAPP1188 Block Diagram**

Similar to the method described in XAPP583 [Ref 1], the flash memory attached to the microprocessor stores the user firmware and FPGA bitfiles. However the microprocessor does not configure the FPGA through the FPGA configuration port directly. Instead, the FPGA slave serial DIN and CCLK pins connect to the SPI bus between the flash memory and the microprocessor. Configuration by this method is possible because the slave serial interface and configuration sequence are compatible with SPI protocol coincidentally. The Operation and Implementation Details section explains the compatibility in more detail.

Depending on your requirements, this method can use as few as two connections. You just need to connect the following pins to deliver the configuration bitstream to the target FPGA:

- The FPGA CCLK pin connects to the SPI bus SCLK pin.
- The FPGA DIN pin connects to the SPI bus MISO pin.

Additional signals can be included for configuration control or monitoring. The following FPGA pins should connect to the microprocessor GPIO pins to enable the features:

- PROGRAM_B – resets the FPGA configuration sequence.
- INIT_B – checks for configuration initialization or error status.
- DONE – monitors configuration success.

This solution offers multiple benefits compared to traditional systems depicted in Figure 1 with an embedded microprocessor and flash:

- Reduced component count – the system requires only one flash memory.
- Fewer microprocessor GPIO and less code space – improves configuration time compared to other embedded microprocessor-based configuration solutions that iterate between reading a bitstream from a memory source and delivering a bitstream to the FPGA.
- Microprocessor-based FPGA configuration control and monitoring capabilities - exceeds the native FPGA configuration functions.
- In-system PROM updates – This solution supports the use of standard flash programming libraries on a microprocessor to update stored bitstreams.

To demonstrate the solution, subsequent sections use the processing subsystem in the Zynq-7000 AP SoC as a model to perform all microprocessor tasks.

Operation and Implementation Details

This section describes the FPGA slave serial configuration interface and sequence, and the method by which an Zynq-7000 AP SoC processing subsystem is used to configure the FPGA. Note that the solution does not apply to the Zynq-7000 AP SoC as the target because its programmable logic must be configured from its processing subsystem.

FPGA Slave Serial Configuration Mode

Slave Serial mode (one of the FPGA configuration modes) is efficient because of its simplicity. Some of its key attributes are listed below. The configuration method described in this application note takes advantage of these attributes.

- After the completion of the FPGA's internal configuration memory clearing and configuration mode sampling stages, the FPGA is put into its bitstream loading state where it monitors its DIN pin.
- A configuration data bit is clocked into the DIN pin on each rising edge of CCLK. However, the FPGA discards all data received prior to a valid sync word (`0xAA995566`).
- The sync word marks the beginning of a bitstream. After receiving a valid sync word through its DIN pin, the FPGA regards the subsequent bitstream as valid configuration data.
- When the FPGA receives a startup command (near the end of the bitstream), the FPGA begins its startup process, during which the FPGA starts the user design and stops monitoring its DIN pin.

The *Clocking Serial Configuration Data* and *Configuration Sequence* sections in the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 2], describe the serial configuration pins and relevant configuration sequence from power-on.

Simple Configuration Method

If the FPGA is only required to be configured once after the system is powered on then the configuration method can be as simple as connecting the following pins:

- The FPGA CCLK pin connects to the SPI bus SCLK pin.
- The FPGA DIN pin connects to the SPI bus MISO pin.

Before the Zynq-7000 AP SoC begins configuration, the FPGA should be ready to receive configuration data. In other words the FPGA should have already completed the configuration house cleaning stage from initial power-up. See the *Device Power-Up Timing* description in UG470 [Ref 2] for the FPGA pre-configuration timing diagram.

Next the Zynq-7000 AP SoC sends a single read command to SPI flash memory and reads the entire bitstream from SPI flash memory. During the SPI flash serial read operation, the bitstream is also serially transmitted across the SPI bus MISO signal to the FPGA DIN pin. The SPI flash outputs each serial data bit on the falling edge of SCLK/CCLK. The FPGA captures the serial data bit on the following rising edge of SCLK/CCLK. Upon reading the entire bitstream from SPI flash memory the FPGA is effectively configured.

Three attributes of the FPGA configuration interface enable the FPGA DIN and CCLK pins to be connected directly to the SPI bus for FPGA configuration:

- The SPI bus serial data (MISO) and clock (SCLK) signals are compatible with the FPGA slave serial data (DIN) and clock (CCLK) pins, respectively.
- The FPGA configuration interface ignores all SPI bus activity prior to the bitstream read operation. Because the FPGA discards all incoming data until it receives a valid 32-bit sync word, effectively ignores all SPI bus activity.
- The FPGA configuration interface ignores all SPI bus activity after the configuration operation completes. Because the FPGA stops monitoring its DIN input after the

completion of configuration, the FPGA effectively ignores all SPI bus activity after the bitstream read operation.

Figure 4 illustrates the SPI bus and FPGA configuration transactions relationship.

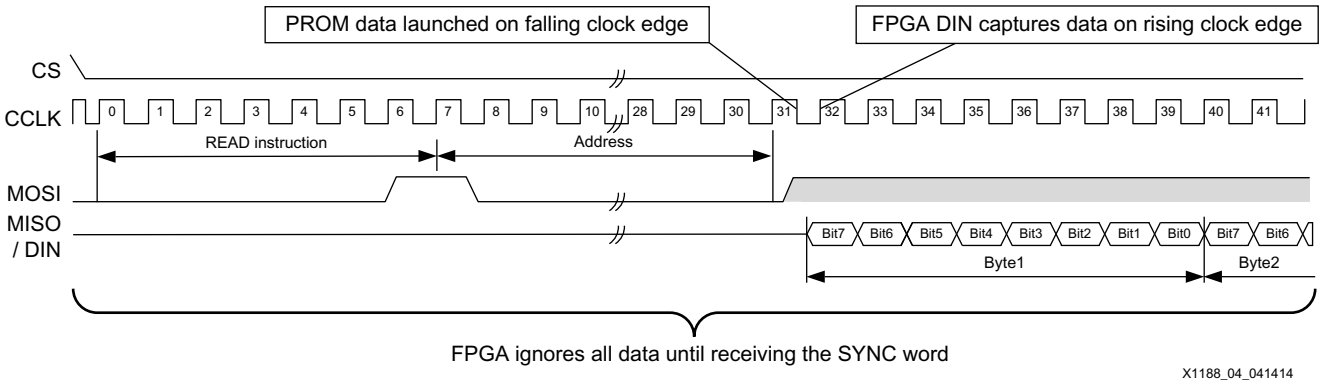

Figure 4: **SPI to FPGA Timing Relationship**

Configuration Method with Additional Controls

Another method to enhance the configuration operation and capabilities is to use the Zynq-7000 AP SoC to control FPGA configuration pins. For example, the Zynq-7000 AP SoC can use its GPIO pin to drive the FPGA PROGRAM_B pin to clear FPGA configuration memory and restart configuration. This allows reconfiguration of the user design based on the system operating conditions. It can offer great advantage to certain user designs which benefit from functionality changes during operation.

In addition, the Zynq-7000 AP SoC can monitor the FPGA configuration readiness and results by connecting its GPIO pins to the FPGA DONE and INIT_B pins. The INIT_B is a configuration error or ready signal. The DONE signal is a configuration completion indicator. These status signals provide information for the Zynq-7000 AP SoC to make effective decisions to ensure reliable configuration.

For example one popular scheme to recover from configuration failure is called configuration fallback. If the target bitfile is corrupted in the flash memory the user design does not configure and the FPGA does not function. In this case firmware detects the situation by checking the INIT_B and DONE signals. It then mitigates the problem by re-configuring the FPGA with a known good (golden) bitfile from the flash memory. This brings the FPGA to a known functional state to avoid a catastrophic system failure.

SPI Bus SCLK Maximum Frequency

Data on a SPI bus are launched and captured on opposite clock edges. In the case the SCLK base value is zero (CPOL=0) and the clock phase is zero (CPHA=0) the data is launched on the falling edge and captured on the rising edge. Therefore the limiting factor of the SCLK frequency is the minimum SCLK/CCLK Low time. It is dictated by the sum of the SPI flash MISO output data valid time from the falling edge of the SCLK and the FPGA DIN input setup time to the rising edge of CCLK shown in Figure 5.

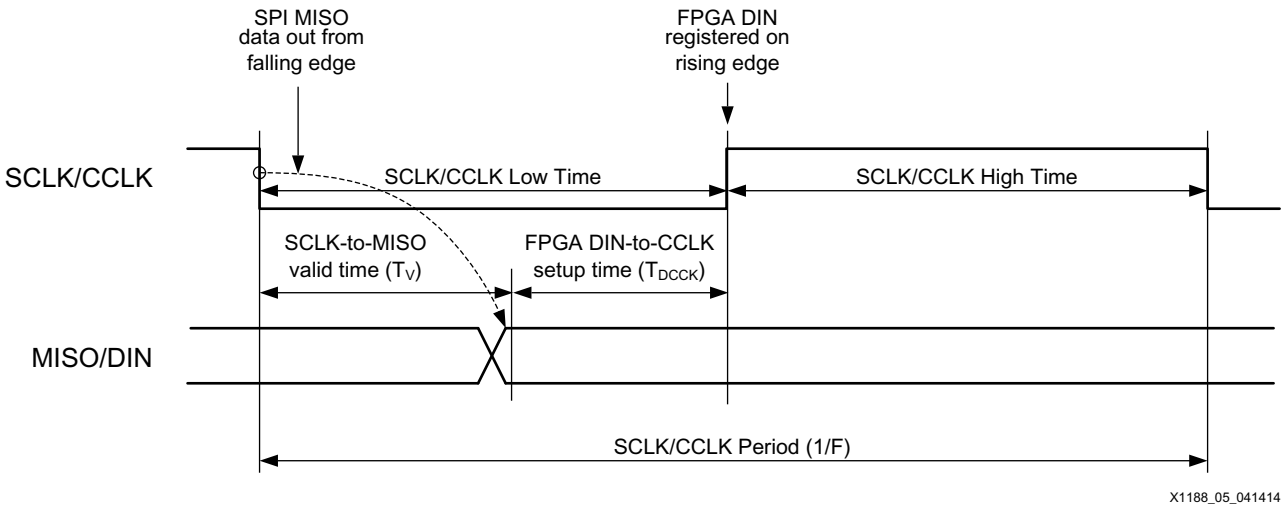

Figure 5: **SPI Bus to FPGA Timing**

The following equation estimates the maximum SCLK frequency for the SPI bus to support FPGA configuration:

$$SCLK_{MaximumFrequency} \leq 1/(T_V + T_{DCCK}) \times SCLK_{LowDutyCycle\ \%\ Minimum} \qquad \textit{Equation 1}$$

Where:

- T_V = SPI flash SCLK to MISO data output valid time.
- T_{DCCK} = FPGA DIN input setup to CCLK setup time.
- SCLK Low duty cycle %, minimum = Minimum percent for SCLK Low within the clock period.

For a more accurate estimate, the trace paths and applicable components of the corresponding propagation delays for the SCLK/CCLK and MISO/DIN signals should be incorporated into the above equation.

In addition the signal integrity of the SCLK signal (FPGA CCLK pin) is critical. To achieve the maximum possible clock frequency, use best practices to design and route this clock signal from the Zynq-7000 AP SoC to the SPI flash memory and FPGA end-points.

SCLK/CCLK Signal Integrity

The signal integrity of the SCLK signal (FPGA CCLK pin) is critical. All SPI bus activity is triggered from a falling or rising edge of SCLK. Signal trace issues that can cause additional glitches (i.e., additional edges) at the SPI flash memory SCLK pin or FPGA CCLK pin must be avoided. Use best practices to design and route this clock signal from the Zynq-7000 AP SoC to the SPI flash memory and FPGA end-points. For example, use a dual buffer to implement point-to-point routing for each destination of the SCLK signal or use fly-by routing with appropriate termination. Avoid traces with separate and long stubs to each destination pin.

Reference Design

A demonstration reference design is provided with this application note. It implements the solution shown in Figure 2 using a Zynq®-7000 All Programmable SoC (AP SoC) and an FPGA on a target board. You can download the Reference Design Files for this application note from the Xilinx® website. Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Xilinx
Target Devices	7 Series FPGAs UltraScale™ Architecture Devices Zynq-7000 AP SoC
Source code provided	Yes
Source code format	C
Design uses code and IP from existing Xilinx Application note and reference designs, CORE Generator software, or third party	Yes
Simulation	
Functional Simulation Performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version used	Vivado® Design Suite version 2013.4
Implementation software tools/versions used	Vivado® Design Suite version 2013.4
Static timing analysis performed	No
Hardware verification	
Hardware verified	Yes
Hardware platform used for verification	Zynq-7000 AP SoC and Kintex®-7 FPGA evaluation boards

Reference Hardware

The hardware design is based on a Zynq-7000 AP SoC 7Z045 CLG484 device. The AP SoC Quad-SPI controller is connected to a multiple I/O memory device that supports SPI x1, x2, or x4 width (e.g. Micron/Numonyx N25Q128A13ESF40F). Although the Zynq Quad-SPI controller can support x1, x2, x4, and x8 width as well, this reference design works in legacy SPI protocol using single bit bus width. Figure 6 shows the hardware connections between the Zynq-7000 AP SoC, the SPI PROM and the Kintex-7 FPGA.

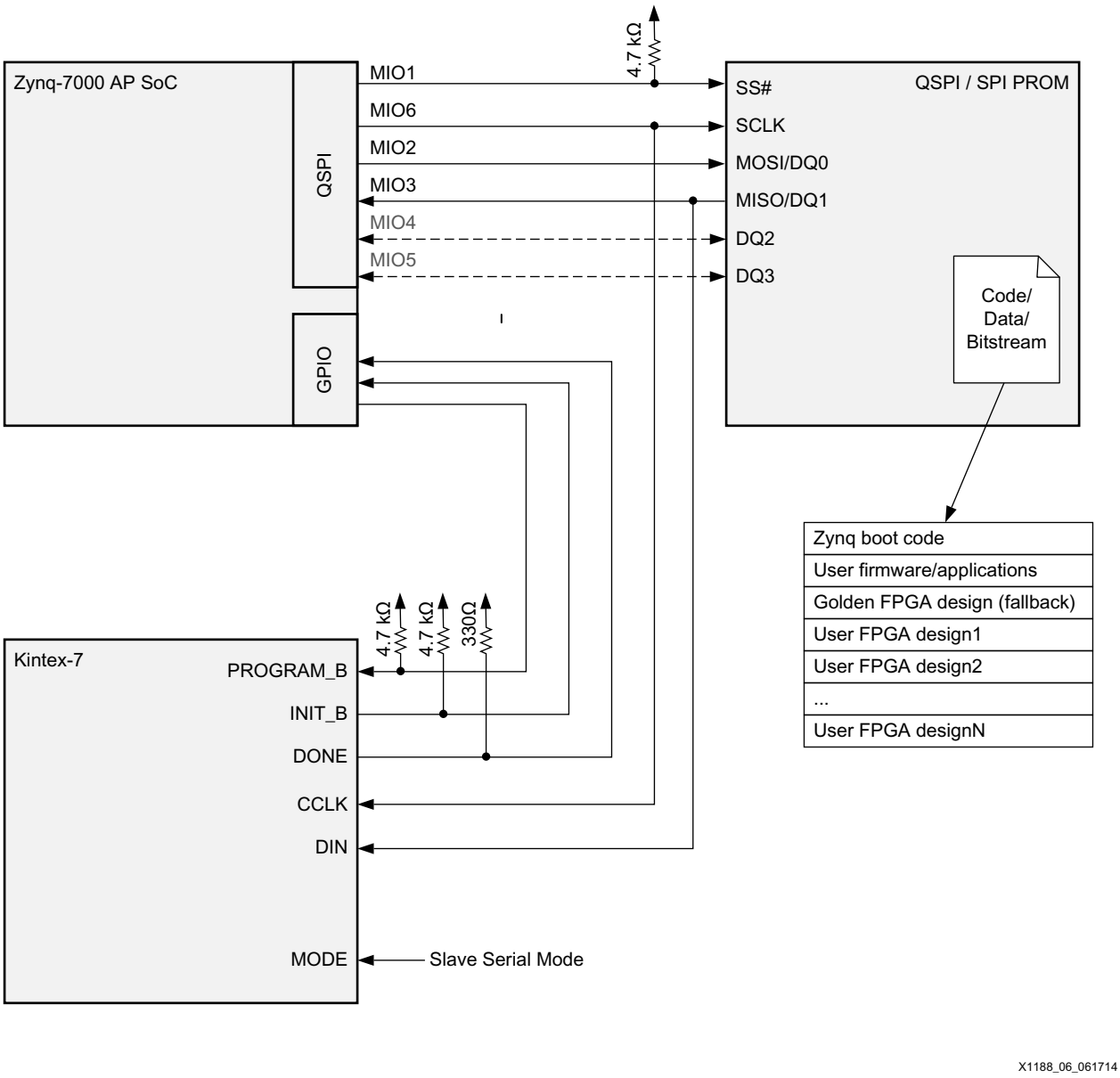

Figure 6: **Reference Hardware Connections**

The reference hardware design is provided in a Vivado® Design Suite project. The system block diagram within the AP SoC is shown in Figure 7.

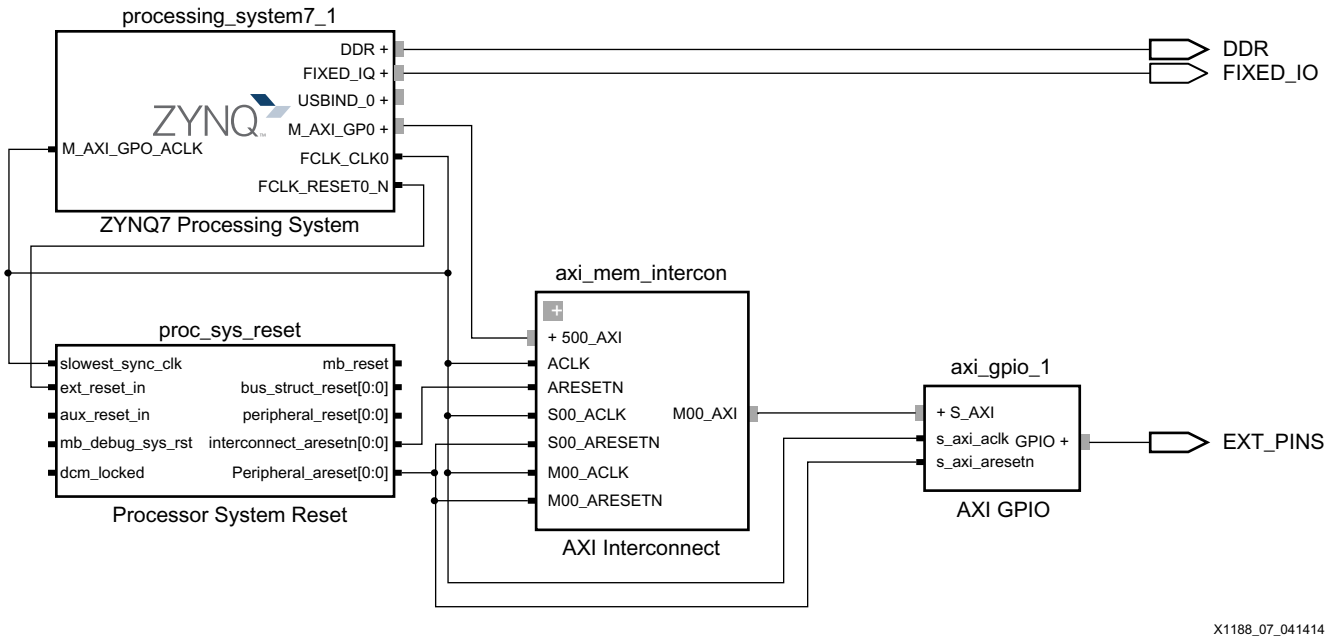

Figure 7: **Vivado Block Design Diagram**

Reference Firmware

The reference design implements a try-and-fallback configuration scheme with an updated (latest design revision) bitstream and golden (known good) bitstream stored in the SPI flash memory. The Zynq-7000 AP SoC firmware first tries to load the latest intended bitstream. Then, if the configuration is unsuccessful, the firmware restarts the configuration and loads the golden bitstream. This procedure is known as fallback configuration. The firmware flow diagram is shown in Figure 8.

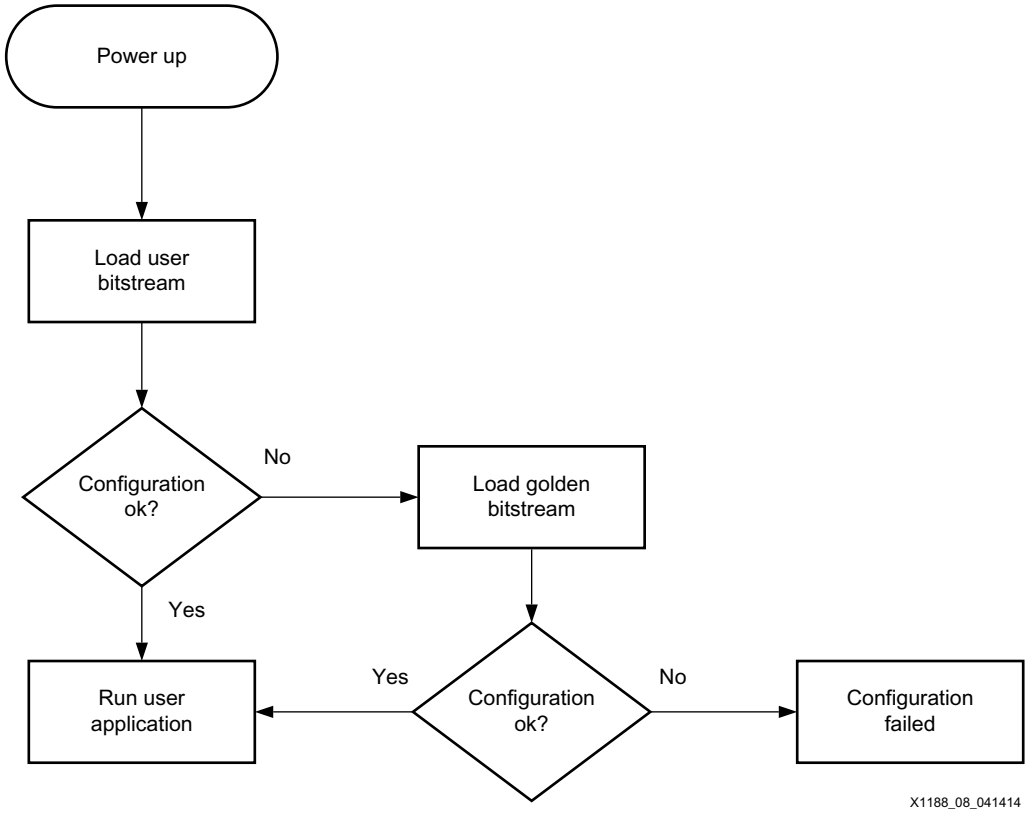

Figure 8: **Firmware Flow Diagram**

Details of the firmware configuration algorithm:

- Initialize the Zynq-7000AP SoC Quad-SPI interface configuration register to I/O mode.
  - Perform tasks such as setting up the clock divider, chip select mode, and enabling the controller.
- Clear FPGA configuration memory.
  - Assert the PROGRAM_B for time period at least longer than TPROGRAM and then de-assert it.
- Wait for INIT_B to be released.
  - Poll the INIT_B pin to make sure the FPGA has completed the house cleaning stage.
- Set up Quad-SPI controller transfer function.
  - The demo uses the polled data transfer function XQspiPs_PolledTransfer(). The function prototype is:

```
int XQspiPs_PolledTransfer(XQspiPs *InstancePtr, u8 *SendBufPtr, u8 *RecvBufPtr,
unsigned ByteCount);
```

  - InstancePtr is the pointer to the Quad-SPI instance
  - SendBufPtr and RecvBufPtr are the transmit and receive buffer pointers respectively. This function requires the send and receive buffer to have the same amount of bytes as the length of the bitstream that will reconfigure the FPGA.

- ByteCount is the number of bytes to be transferred.
- The first word of the send buffer contains command and address as shown in Table 2. The first byte contains the command for Quad-SPI memory device. In this case it will send the Read Byte command (0x03). Only one command is needed to complete the entire operation. The 24-bit start address of the target bitstream is split across three bytes. The content of the rest of the buffer is undefined since this is a read operation.

Table 2: **Quad-SPI Write Buffer Definition**

Byte 0	Byte 1	Byte 2	Byte 3	Remaining Bytes
SPI command	Address[23:16]	Address[15:8]	Address[7:0]	Transmit data

- Call polled transfer function
  - After calling the function the Quad-SPI controller sends content of the send buffer to the Quad-SPI memory. At the same time it reads in the same number of bytes the size of the bitstream. As the DIN and CCLK pins of the FPGA are connected to the SPI controller's MISO and SCK signals, the FPGA (set to Slave Serial configuration mode) will begin configuration when the SYNC word is detected. The FPGA is configured when the bitstream is fully read.
- Check for error
  - After the FPGA is configured, if INIT_B and DONE become asserted, then this signals the end of a successful read.
  - However, if the INIT_B and/or the DONE pin stays low the firmware will load the known good configuration (i.e., golden bitstream) and reconfigure the FPGA. This is known as configuration fallback.

Conclusion

This application note describes a unique FPGA configuration solution that reduces hardware and firmware requirements of a typical system that contains a microprocessor, flash memory, and FPGA. It leverages the compatibility between the FPGA serial configuration mode and SPI memory. Although the solution is applicable to most microprocessors, this application note and reference design use the processing subsystem of the Zynq-7000 AP SoC to demonstrate the operation and implementation details.

References

1. *Using a Microprocessor to Configure 7 Series FPGAs via Slave Serial or Slave SelectMAP Mode* (XAPP583)
2. *7 Series FPGAs Configuration User Guide* (UG470)
3. *Using SPI Flash with 7 Series FPGAs* (XAPP586)

4. *Zynq-7000 All Programmable SoC (Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics* (DS187)
5. *Zynq-7000 All Programmable SoC (Z-7030, Z-7045, and Z-7100): DC and AC Switching Characteristics* (DS191)
6. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)

Appendix A

Design Troubleshooting Tips

Follow this checklist in case the configuration is not successful:

- Check the following on the target board:
  - Check for basic FPGA connectivity and functionality via JTAG:
    - Use the Xilinx programming tool and JTAG cable to read the FPGA JTAG IDCODE.
    - Use the Xilinx programming tool and JTAG cable to configure the FPGA with a bitstream.
    - Check that DONE is High at the end of the JTAG configuration operation.
  - Check that the FPGA is ready to be configured before the microprocessor reads the bitstream from the SPI flash memory:
    - Use a digital oscilloscope to probe the FPGA PROGRAM_B and INIT_B signals. Monitor these signals from power-on or through the PROGRAM_B reset procedure. These signals should behave as shown in the *Device Power-Up Timing* figure in the 7 Series FPGAs Configuration User Guide (UG470) [Ref 2].
  - When the microprocessor reads the bitstream from SPI flash memory, check the SPI flash data bit/byte order:
    - Use a digital oscilloscope or logic analyzer to monitor the FPGA DIN (SPI MISO) signal and FPGA CCLK (SPI SCLK) signal for the first ~256 bits of the bitstream. The bitstream sync word (`0xAA995566`) should appear within the first 256 bits of the bitstream. The order of bits/bytes for the sync word arriving at the DIN pin should be (where the bit value on the left side is first):

```
1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0
```

  - After the microprocessor reads the bitstream from SPI flash memory, check the configuration state of the FPGA:
    - Use the Xilinx programming tool and a JTAG cable to read the FPGA configuration status. Review the status report for hints of completed and failed configuration steps.
    - Use the Xilinx programming tool and a JTAG cable to verify the configuration in the FPGA.

        - Use the Xilinx programming tool and a JTAG cable to readback the FPGA configuration to a file. Review the file with Xilinx Support for patterns of bitstream data that configured the device or did not configure the device.
    - Use a digital oscilloscope to probe as close as possible to the FPGA CCLK pin to check for glitches that appear on any edge.
- Check the schematic for the following:
    - FPGA is set up for compatible I/O voltages with microprocessor and SPI flash memory SPI bus.
    - SPI flash memory pins are properly terminated, including: write-protect#, hold#, and reset# pins.
    - The signal integrity of the SCLK signal (FPGA CCLK pin) is critical. Use best practices to design and route this clock signal from the microprocessor to the SPI flash memory and FPGA end-points. All SPI bus activity is triggered from a falling or rising edge of SCLK. Issues that can cause additional glitches (i.e., additional edges) at the SPI flash memory SCLK pin or FPGA CCLK pin must be avoided.
    - It is recommended to pulse the PROGRAM_B pin prior to configuration. This eliminates the issue of extraneous SPI bus activity from possibly moving the FPGA configuration logic to an unexpected state and assures that FPGA configuration logic is in a known state for receiving a bitstream, including for the first configuration after power-on.
- Check the microprocessor configuration or application code for the following:
    - Check that the GPIO configured correctly as outputs or inputs for controlling the FPGA PROGRAM_B signal or monitoring the FPGA INIT_B and DONE signals.
    - Check that the FPGA PROGRAM_B reset pulse meets the required timing and otherwise leaves the PROGRAM_B signal in a High state.
    - If the code pulses the FPGA PROGRAM_B, check that the code either waits for INIT_B to go High or waits for a minimum of TPL time. See the FPGA data sheet for TPL time.
    - Check that the SPI bus SCLK frequency is configured to operate no faster than the maximum frequency computed in SPI Bus SCLK Maximum Frequency.
    - Check that the SPI flash memory read command is paired with the appropriate starting address for the location of the bitstream in the SPI flash memory.
    - Check that the SPI command for reading the bitstream reads all bits of the bitstream. See the *Bitstream Length* table in the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 2] for the bitstream length.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/23/2014	1.0	Initial Xilinx release.

Please Read: Important Legal Notices