



XAPP1233 (v1.1.1) September 23, 2016

SPI Configuration and Flash Programming in UltraScale FPGAs

Authors: Matt Nielson and Ryan Rumsey

Summary

This application note describes the UltraScale™ FPGAs master serial peripheral interface (SPI), 4-bit datapath (x4 or quad) configuration mode. The x4 mode is recommended, but the 1-bit datapath (x1) and 2-bit datapath (x2) modes are easily adapted from the x4 mode if needed. The application note reviews the basics of master SPI configuration that can aid in successful SPI configuration and debugging of configuration during initial design bring-up.

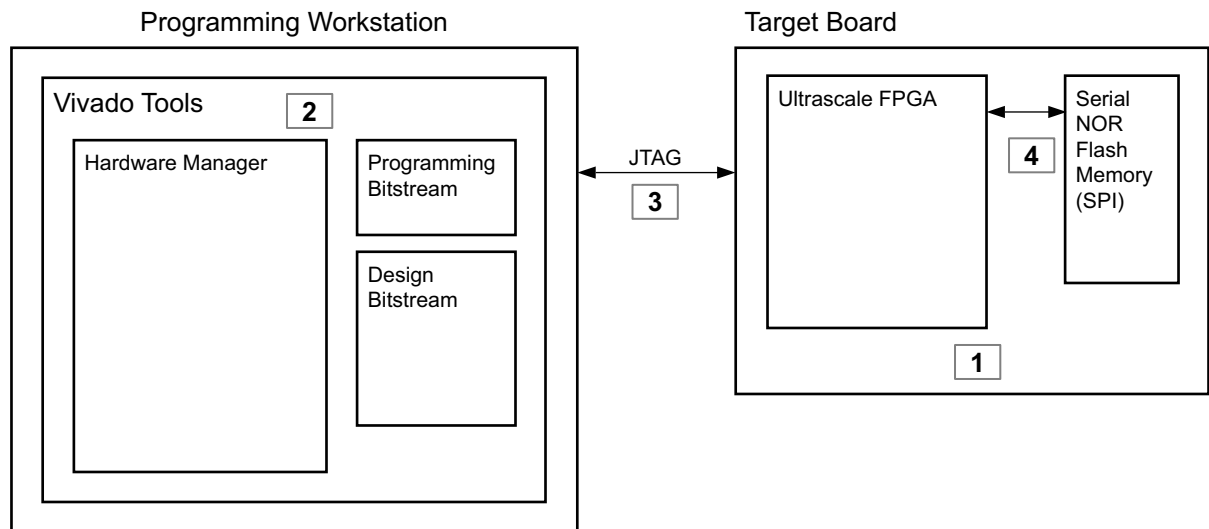
This document also includes instructions for generating a bitstream for SPI configuration and programming this bitstream into the SPI memory device using the Vivado® Design Suite Integrated Design Environment (IDE) as well as a Tcl flow that can be used from the command line. A sample set of connections between the SPI memory device and the FPGA required for master SPI x4 are also provided.

Introduction

The master SPI configuration mode of UltraScale FPGAs enables a low pin count configuration option. UltraScale FPGAs have four dedicated data pins in the configuration bank (bank 0) allowing for medium configuration speed via quad SPI serial NOR flash devices that adhere to the quad SPI flash standards.

Xilinx recommends laying out the board for this x4 width to achieve a fourfold speed-up in configuration time over x1 master SPI. Because all SPI x4 pins are in the dedicated bank 0, no additional user I/O pins are lost as a result of using x4. The x1 and x2 widths are supported and easily adapted from the x4 case presented in this application note. The UltraScale FPGA also introduces a master SPI dual quad mode that allows for higher configuration rates by using two quad SPI in parallel. Dual quad SPI configuration is not covered in detail in this application note. Refer to *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] for details on the x8 mode.

In-system programming of the flash is possible using the Vivado Design Suite. Referred to as *indirect programming*, the Vivado Design Suite is able to program the UltraScale FPGA configuration bitstream into the SPI flash using JTAG. The Vivado Design Suite supports a range of SPI flash densities with some memories large enough to store multiple bitstreams of even the largest UltraScale FPGAs. The extra storage can be accessed after configuration via an embedded SPI controller added to the interconnect logic. [Figure 1](#) represents both the indirect programming flow and the configuration sequences at a high level.



x1233_01_051515

Figure 1: Vivado SPI Flash Programming and Configuration Flow

Process flow overview:

1. The SPI flash is on-board and is connected directly to the target FPGA's configuration interface. The board is connected to a local workstation via a programming cable such as the Xilinx® Platform Cable USB II or Digilent programming module.
2. The Vivado Design Suite, which is installed on the local workstation, contains an indirect programming bitstream. It also has access to the final target bitstream to be programmed into the SPI flash.
3. Using Hardware Manager from the Vivado Design Suite, the target UltraScale FPGA is configured using the indirect programming bitstream. The Vivado Design Suite is then used to program the SPI flash with the target bitstream over the JTAG connection through the indirect programming bitstream.
4. Now that the SPI flash is programmed with the target bitstream, the UltraScale FPGA configures directly from the SPI flash after a PROGRAM_B assertion or power cycle if the mode pins are set for master SPI configuration mode.

The UltraScale FPGA is capable of configuring itself from serial NOR flash memories that support the SPI fast read and SPI quad output fast read commands. This configuration mode offers a simple and reliable configuration solution while not requiring any user I/O pins. Being a x4 wide configuration interface, it can offer medium configuration speeds. If shorter configuration times are required, other configuration options such as Slave SelectMAP (x8, x16, x32), Master BPI (x16), or Master SPI x8 (which uses two SPI flash devices in parallel) should be considered. See *UltraScale FPGA BPI Configuration and Flash Programming* (XAPP1220) [Ref 2] or *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] for details on these configuration modes. See the [Appendix](#) for details on configuration time calculation.

The required steps for configuring an UltraScale FPGA from SPI flash can be described at a high level as:

1. Understand the basics of SPI and master SPI quad configuration: [SPI Basics](#).
2. Lay out the board for FPGA configuration via SPI flash: [Board Layout for Master SPI Configuration](#).
3. Prepare target bitstream (as a .bin file) from the Vivado Design Suite: [Master SPI Configuration Files](#).
4. Program the NOR flash in-system using the Vivado Design Suite hardware manager: [Program the SPI Flash](#).
5. Configure the target FPGA from the serial NOR flash: [Configure the Target FPGA](#).

These steps make up the major sections in this application note. Several additional sections are provided in the [Appendix](#):

1. [Programming Time](#)
2. [Configuration Time](#)
3. [Selecting an SPI Flash](#)
4. [Master SPI x1, x2, and x8](#)
5. [Generating SPI Configuration Constraints: IDE Flow](#)
6. [Debugging and Troubleshooting Guidelines](#)

SPI Basics

A basic understanding of the subset of the SPI and quad SPI standard used for master SPI configuration in UltraScale FPGAs can help ensure successful configuration and assist in troubleshooting possible configuration failures.

The actual SPI standard is a simple chip-to-chip data link often used in embedded systems to communicate with memories and sensors. In its original form, referred to as x1 width in this application note, SPI operates in a full duplex mode sending commands and addresses from the master over the master-output, slave-input (MOSI) line and receiving data over the master-input, slave-output (MISO) line. Data is synchronous with an SPI clock (SCLK) signal, and the slave device is selected via an SPI select (SS#) signal. [Figure 2](#) shows the x1 interface between an SPI master and an SPI slave.

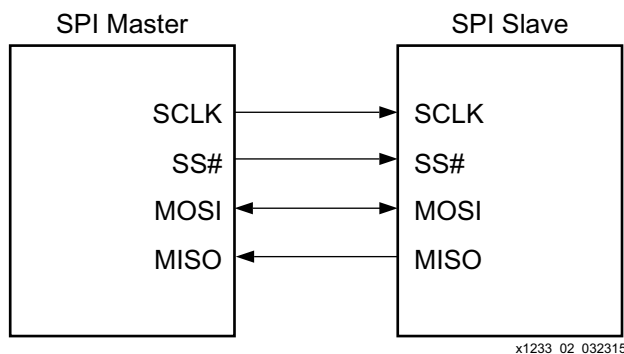


Figure 2: SPI Master/Slave Interface

In addition to the pins described above, the SPI flash can have additional pins that can be used to control other special functions. These additional pins can vary with the SPI flash vendor, however, two common special function pins are hold and write protect. The hold and write protect pins serve as additional data pins in SPI x4 mode as described in Table 1. This table summarizes the data pins as well as different names commonly used by different vendors.

Table 1: SPI Flash Data Pins

SPI Signal Name	UltraScale FPGA Pin Name (UltraScale FPGA Signal Name)	N25Q256A ⁽¹⁾ Pin Name	Other Common Names	Pin Function
SCLK	CCLK (CCLK)	C	SCK, CLK	Synchronous clock driven by master to slave.
SS#	RDWR_FCS_B (FCS_B)	S#	CS#	Active-Low slave select.
MOSI	D00_MOSI (D00_MOSI)	DQ0	SI, IO0, SIO0, DI	Master-output, slave-input. Used as LSB data bit in x2 and x4 output modes.
MISO	D01_DIN (D01_DIN)	DQ1	SI, IO1, SIO1, DO	Master-input, slave-output.
W#	D02 (D02)	DQ2	WP#, IO2, SIO2	Write protect. Used as data pin in x4 mode.
HOLD#	D03 (D03)	DQ3	HOLD#, IO3, SIO3	Hold or pause without deselecting the device. Used as the MSB data bit in x4 mode.

Notes:

1. Micron serial NOR flash memory [Ref 3].

The waveform in [Figure 3](#) shows a fast read command (0Bh) up to the first few bits of the synchronization word returned by the slave if that slave contains a valid UltraScale FPGA configuration bitstream at address 0. The read command continues until the slave select line is deasserted by the master. Data is always transmitted as most significant bit first.

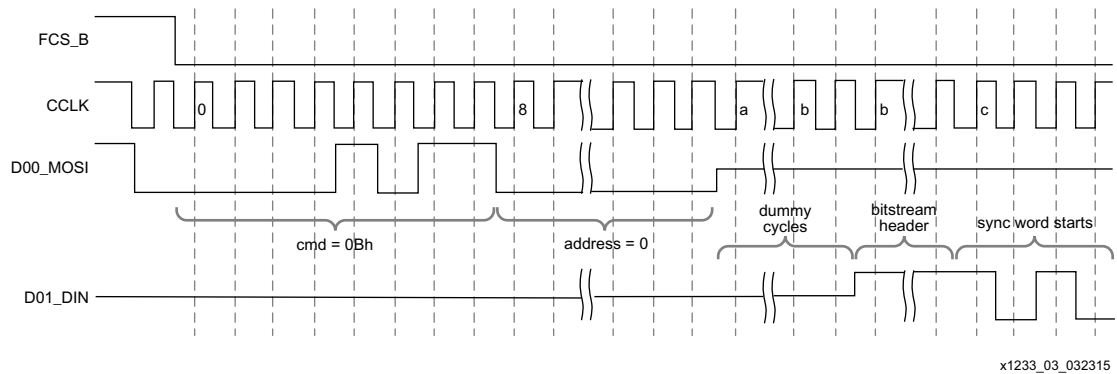


Figure 3: Master SPI x1 0Bh Command

Notes relevant to [Figure 3](#):

- Clock a = 32 cycles for 24-bit addressing or 40 cycles for 32-bit addressing fast read commands.
- Clock b = Clock a + number of dummy cycles output by SPI flash for fast read commands.
- Clock c = Clock b + number of bitstream header bytes. The bitstream header is bytes of 0xFF and a 32-bit address width detection word inserted into the bitstream before the synchronization word by write_bitstream.

Quad SPI Flash

To increase throughput over the x1 width, memory vendors offer a quad SPI mode in which two additional lines are used for data. The UltraScale FPGA configuration logic can issue commands and data over the MOSI line in x1 width. It then receives data in the x4 width using the MOSI and MISO lines and the two additional data lines. This interface is presented in [Figure 4](#). In this figure, the master and slave pin names are changed to match the UltraScale FPGA and the N25Q256A names.

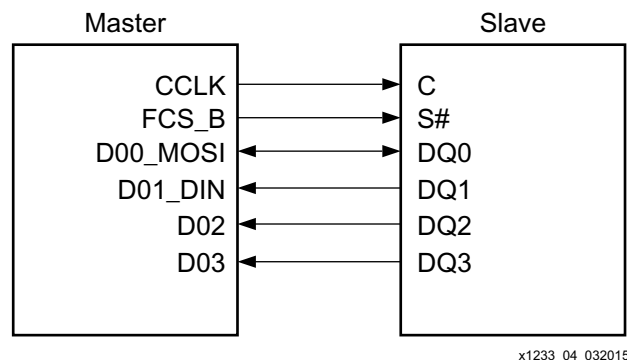


Figure 4: Quad (x4) SPI Master/Slave Interface

Figure 5 shows the quad output fast read command (6Bh) and quad output fast read 32-bit address command (6Ch). As in the fast read command, the command and address are sent over the D00_MOSI (D[00]) line but now data is received over the D[03:00] lines. Data is always transmitted as MSB first on D[03].

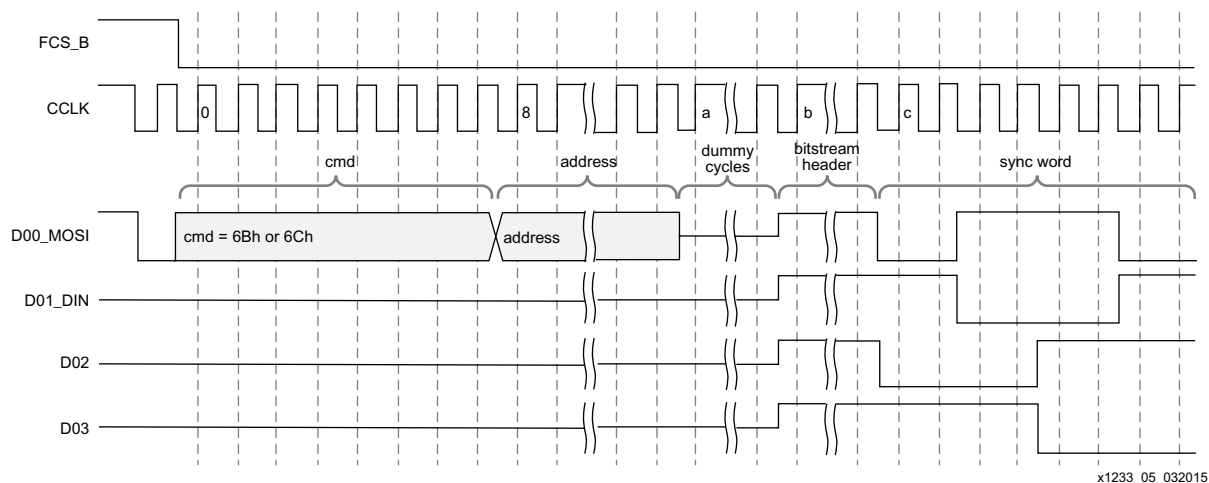


Figure 5: Master SPI x4 6Bh/6Ch Command

UltraScale FPGA Master SPI Configuration

The UltraScale FPGA can configure itself from an attached SPI flash device when set up for master SPI configuration mode. The UltraScale FPGA only issues SPI fast read and SPI output fast read commands as listed in Table 2.

Table 2: SPI Commands

SPI Command	Opcode
Fast read	0Bh
Dual output fast read	3Bh
Quad output fast read	6Bh
Fast read, 32-bit address	0Ch
Dual output fast read, 32-bit address	3Ch
Quad output fast read, 32-bit address	6Ch

After UltraScale FPGA power-up or following a PROGRAM_B pulse and release of INIT_B, if the FPGA configuration mode pins are set to the master SPI configuration mode (M[2:0] = 001), the FPGA asserts the flash chip select pin (FCS_B) and drives out the SPI fast read command (0Bh) onto D00_MOSI. The waveform is represented in Figure 3.

In all SPI commands used for UltraScale FPGA configuration, the command and address portions of the SPI commands are always sent over MOSI in the x1 width. The data is read back in the x1, x2, or x4 widths as appropriate for the command. The MOSI line is deasserted and allowed to float after sending the last address bit. For the x2 and x4, widths, the MOSI line is then used as the least significant bit D00 for the read commands.

UltraScale FPGA Master SPI Configuration Sequence

If the configuration mode pins are set for master SPI configuration mode ($M[2:0] = 001$), after FPGA power-up or following the deassertion of PROGRAM_B, the UltraScale FPGA starts the master SPI configuration sequence. This sequence always begins with the configuration logic driving out the fast read command as shown in [Figure 3](#):

1. Drive out the fast read command (0Bh)
2. Send three address bytes of zeros
3. Stop driving the D00_MOSI pin.
4. Keep FCS_B asserted and continue clocking CCLK
5. Scan data on D01_DIN for the synchronization word, the 32-bit pattern of 0xAA995566
6. Align to subsequent data to the synchronization word as 32-bit words
7. Start processing each 32-bit word as FPGA configuration packets and commands
8. Keep FCS_B asserted until one of the following occurs:
 - Configuration commands instruct the configuration logic to issue a new SPI command
 - A device ID error occurs
 - A cyclic redundancy check (CRC) error occurs
 - Configuration completes

For example, after the synchronization word is read, embedded FPGA configuration commands can cause the FPGA configuration logic to issue the 4-byte quad output fast read command (6Ch) at an address specified in the bitstream. If this address was 16 MB (0x01000000), the FPGA deasserts FCS_B and then reasserts it, sending the 0x6C followed by 0x01000000 on D00_MOSI. The configuration logic continues to keep FCS_B asserted while reading on D[03:00]. The FPGA configuration logic again scans the incoming data for the synchronization word and, after finding it, the behavior is the same as noted above except that the embedded SPI command is skipped.

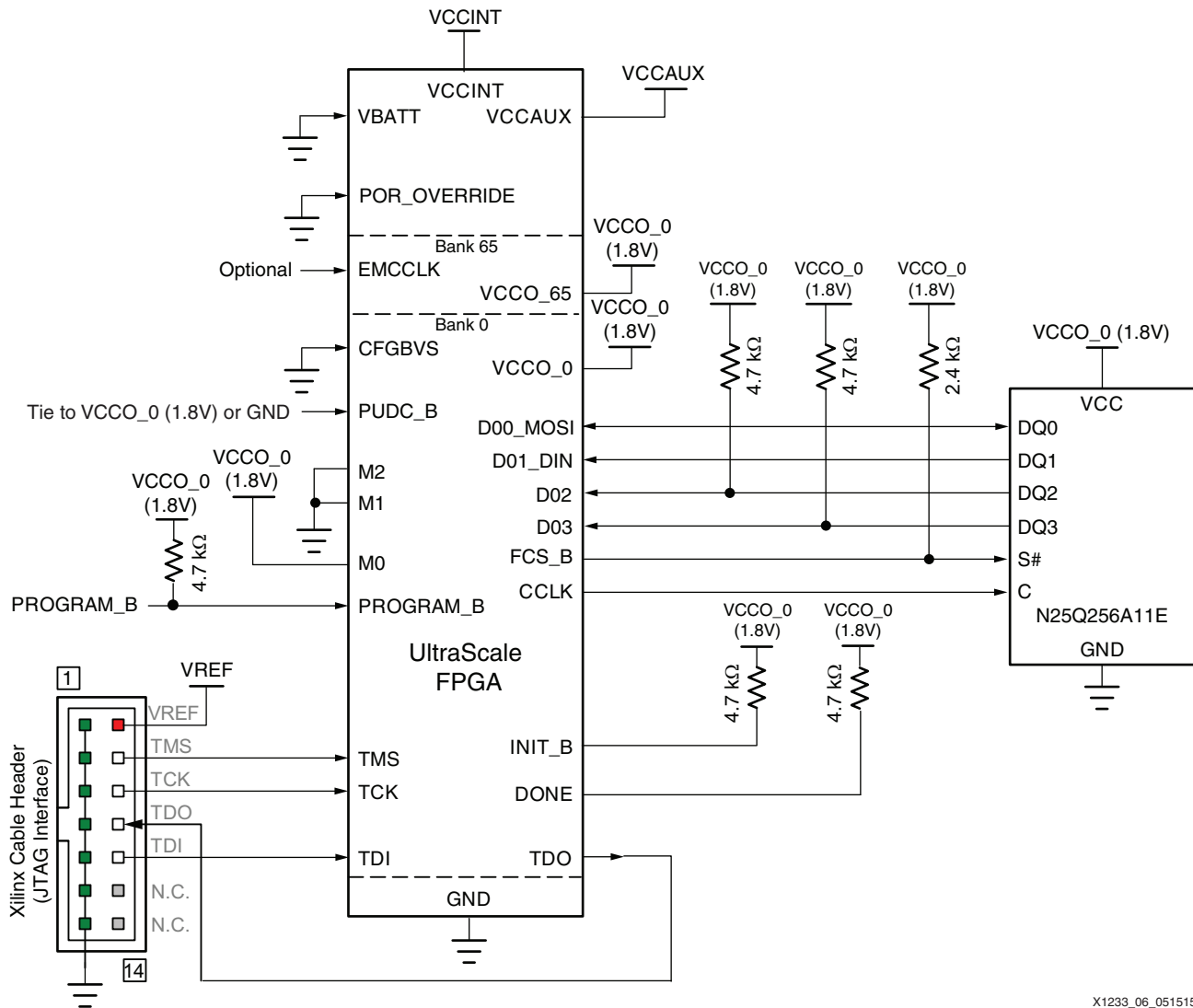
Note: Because the UltraScale FPGA configuration logic scans for the synchronization word to align data along a 32-bit boundary, the number of dummy cycles used by the SPI flash are not of consequence for master SPI configuration. The configuration logic does not have to count dummy bits to align the data but rather relies on latching in the synchronization word to do so.

Board Layout for Master SPI Configuration

While the SPI interface is not a high-performance interface, care must be taken to ensure signal integrity issues or board layout decisions do not limit the clock speeds or lead to signaling issues like reflection on the CCLK line.

The Kintex UltraScale FPGA KCU105 evaluation kit [\[Ref 5\]](#) provides good guidance for master SPI x4 configuration layout. This board has two N25Q256A devices to provide master SPI x8 mode. However, this application note only covers a single SPI flash, U35, used in master SPI x4 mode. This same setup can also be used for x1 or x2 modes if desired.

Board layout for master SPI x4 is very similar to previous generation master x1/x2 layout requirements. The x4 width requires two additional signals that might have been tied High on the board and not connected to the FPGA in 7 series FPGA configuration layouts. These additional signals (connecting D[03:02] to the SPI flash) as well as the rest of the connections required for master SPI x4 configuration are shown in [Figure 6](#). [Table 3](#) provides descriptions of these signals specific to SPI configuration.



X1233_06_051515

Figure 6: Master SPI x4 Configuration Interface

Table 3: Master SPI x4 Configuration Pins


Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
POR_OVERRIDE	N/A Input (Dedicated)	Power on reset delay override	Generally recommended to connect to GND for x4 configuration. The time saved by this option is small compared with the total time required to configure an entire UltraScale FPGA bitstream.	<p>Reduces T_{POR} (from power-up to INIT_B rise) as specified in the data sheets [Ref 6] [Ref 7]. Connect directly to V_{CCINT} for a shorter T_{POR} if required and if supported by the power-up timing of the configuration data source. Connect to GND for standard longer POR delay.</p> <p> CAUTION! Do not allow this pin to float before and during configuration. Must be tied to V_{CCINT} or GND. Do not connect to V_{CCO_0}.</p>
VBATT	N/A N/A (Supply voltage)	Battery backup supply	If there is no requirement to use a decryptor key stored in the volatile key memory area, connect this pin to GND or V _{CCAUX} . This application does not cover encryption and ties V _{BATT} to GND in Figure 6.	Battery backup supply for the FPGA internal volatile memory that stores the key for the AES decryptor. For encrypted bitstreams that require the decryptor key from the volatile key memory area, connect this pin to a battery to preserve the key when the FPGA is unpowered.
M[2:0]	0 Input (Dedicated)	Configuration mode	Set M[2:0] pins to 001 for master SPI configuration via 1 k Ω (or stronger) resistors to GND or V _{CCO_0} .	Determine the configuration mode. Connect each mode pin either directly or via a ≤ 1 k Ω resistor to V _{CCO_0} or GND.
TCK	0 Input (Dedicated)	IEEE Standard for Test Access Port and Boundary-Scan Architecture (IEEE Std 1149.1) (JTAG) test clock	Connect to JTAG cable header and pull up to 10 k Ω resistor. Required for Vivado tools indirect programming of the SPI flash.	Clock for all devices on a JTAG chain. Connect to Xilinx cable header's TCK pin. Treat as a critical clock signal and buffer the cable header TCK signal as necessary for multiple device JTAG chains. If the TCK signal is buffered, connect the buffer input to an external weak pull-up resistor (e.g., 10 k Ω) to maintain a valid High when no cable is connected.

Table 3: Master SPI x4 Configuration Pins (Cont'd)

Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
TMS	0 Input (Dedicated)	JTAG test mode select	Connect to JTAG cable header and pull up to 10 k Ω resistor. Required for Vivado tools indirect programming of the SPI flash.	Mode select for all devices on a JTAG chain. Connect to Xilinx cable header's TMS pin. Buffer the cable header TMS signal as necessary for multiple device JTAG chains. If the TMS signal is buffered, connect the buffer input to an external weak pull-up resistor (e.g. 10 k Ω) to maintain a valid High when no cable is connected.
TDI	0 Input (Dedicated)	JTAG test data input	Connect to JTAG cable header and pull up to 10 k Ω resistor. Required for Vivado tools indirect programming of the SPI flash.	JTAG chain serialized data input. For an isolated device or for the first device in a JTAG chain, connect to Xilinx cable header's TDI pin. Otherwise, when the FPGA is not the first device in a JTAG chain, connect to the TDO pin of the upstream JTAG device in the JTAG scan chain.
TDO	0 Output (Dedicated)	JTAG test data output	Connect to JTAG cable header. Required for Vivado tools indirect programming of the SPI flash.	JTAG chain serialized data output. For an isolated device or for the last device in a JTAG chain, connect to Xilinx cable header's TDO pin. Otherwise, when the FPGA is not the last device in a JTAG chain, connect to the TDI pin of the downstream JTAG device in the JTAG scan chain.
PROGRAM_B	0 Input (Dedicated)	Program (bar)	Pull-up to V _{CCO_0} via ≤ 4.7 k Ω . Connect a normally open pushbutton to GND to enable manual configuration reset to assist during configuration bring-up.	Active-Low reset to configuration logic. When PROGRAM_B is pulsed Low, the FPGA configuration is cleared and a new configuration sequence is initiated. Configuration reset is initiated upon the falling edge, and configuration (i.e., programming) sequence begins upon the following rising edge. Connect PROGRAM_B to an external ≤ 4.7 k Ω pull-up resistor to V _{CCO_0} to ensure a stable High input.

Table 3: Master SPI x4 Configuration Pins (Cont'd)

Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
INIT_B	0 Bidirectional, open-drain (Dedicated)	Initialization (bar)	Connect to V _{CCO_0} via 4.7 k Ω resistor. Adding dual color LEDs such as the ones used on the KCU105 board provide a quick visual indication of the configuration status along with the DONE pin. An LED is not required, but if not added, be sure the pin can be probed easily for configuration bring-up.	Active-Low FPGA initialization pin or configuration error signal. The FPGA drives this pin Low when the FPGA is in a configuration reset state, when the FPGA is initializing (clearing) its configuration memory, or when the FPGA has detected a configuration error. INIT_B does not drive Low when V _{CCINT} is powered down. Upon completing the FPGA initialization process, INIT_B is released to high impedance at which time an external resistor is expected to pull INIT_B High. INIT_B can externally be held Low during power-up to stall the power-on configuration sequence at the end of the initialization process. When a High is detected at the INIT_B input after the initialization process, the FPGA proceeds with the remainder of the configuration sequence dictated by the M[2:0] pin settings. After configuration, INIT_B can optionally be leveraged to indicate when the FPGA has detected a configuration error. Connect INIT_B to a 4.7 k Ω pull-up resistor to V _{CCO_0} to ensure clean Low-to-High transitions.
DONE	0 Bidirectional (Dedicated)	Configuration done	Do not actively pull up or down. Add an LED that is activated when DONE is asserted or an easily accessible test point to assist in configuration bring-up.	A High signal on the DONE pin indicates completion of the configuration sequence. By default, the DONE output is open-drain. DONE has a default internal pull-up resistor of approximately 10k Ω . An external resistor circuit is not required. If used, a 4.7 Ω resistor is recommended.


Table 3: Master SPI x4 Configuration Pins (Cont'd)

Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
CCLK	0 Input or Output (Dedicated)	Configuration clock	Required for SPI configuration. Poor signal quality on this connection to the SPI flash from the FPGA can negatively impact configuration speeds or prevent successful configuration altogether. Xilinx recommends placing the SPI flash close to the FPGA and matching the length fairly closely with the data pins. Xilinx recommends being able to probe this signal close to the flash device for configuration bring-up.	Runs the synchronous FPGA configuration sequence by default. The FPGA sources the configuration clock and drives CCLK as an output. Treat CCLK as a critical clock signal to ensure good signal integrity.
PUDC_B	0 Input (Dedicated)	Pull-up during configuration (bar)	Tie to V_{CCO_0} or GND either directly or via a $\leq 1\text{ k}\Omega$ resistor according to your design requirements for the state of the SelectIO™ interface pins during configuration.	Active-Low input enables internal pull-up resistors on the SelectIO interface pins after power-up and during configuration. When PUDC_B is Low, internal pull-up resistors are enabled on each SelectIO interface pin. When PUDC_B is High, internal pull-up resistors are disabled on each SelectIO interface pin. PUDC_B must be tied either directly, or via a $\leq 1\text{ k}\Omega$ resistor, to V_{CCO_0} , or GND.
RDWR_FCS_B	0 Output (Dedicated)	Active-Low flash chip select (FCS_B)	Required for SPI configuration. Connect to the SPI flash chip select pin and pull-up via a $\leq 4.7\text{ k}\Omega$ resistor. Make sure there is a point on this signal that can be easily probed for configuration bring-up.	Active-Low chip select output that enables SPI flash devices for configuration. Connect to the flash device chip select input and connect to an external $\leq 4.7\text{ k}\Omega$ pull-up resistor to V_{CCO_0} (2.4 k Ω recommended).

Table 3: Master SPI x4 Configuration Pins (Cont'd)

Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
D00_MOSI	0 Bidirectional (Dedicated)	SPI x1 serial data output, SPI x2/x4/x8 data pin 0 (D00_MOSI)	Required for SPI configuration. Connect to the SPI flash serial data input (DQ0/D/SI/IO0) pin. Make sure there is a point on this signal that can be easily probed for configuration bring-up.	Output for sending commands to the SPI (slave) flash device for all SPI widths. SPI commands and addresses are always sent over this pin in a x1 width to the slave during configuration. In the x2, x4, and x8 widths, this is a dual-purpose pin used as the least significant data bit.
D01_DIN	0 Input (Dedicated)	SPI x1 serial data input (DIN), SPI x2/x4/x8 data pin 1 (D01)	Required for SPI configuration. Connect to the SPI flash serial data output (DQ1/Q/SO/IO1) pin. Make sure there is a point on this signal that can be easily probed for configuration bring-up.	Data input that receives serial data from the slave in x1 mode. Data input bit 1 in the x2, x4, and x8 modes. By default, data from DIN is captured on the rising edge of CCLK.
D02	0 Input (Dedicated)	SPI x4/x8 data bit 2	Not required, but highly recommended. Pull-up to V _{CCO_0} via a 4.7 kΩ resistor. Connect this to the SPI flash data bit 2 pin (DQ2/W#/WP#/IO2).	Data input that receives SPI data in the x4 and x8 modes. This pin is commonly used as a write protect pin on the SPI flash devices and should be pulled up to ensure correct SPI functionality when not operating in the x4 or x8 modes.
D03	0 Input (Dedicated)	SPI x4/x8 data bit 3	Not required, but highly recommended. Pull-up to V _{CCO_0} via a 4.7 kΩ resistor. Connect to the SPI flash quad data bit 3 output (DQ3/HOLD#/IO3) pin.	Data input that receives SPI data in the x4 and x8 modes. This pin is commonly used as a hold or reset pin on SPI flash devices and should be pulled up to ensure correct SPI functionality when not operating in the x4 or x8 modes.

Table 3: Master SPI x4 Configuration Pins (Cont'd)

Pin Name	Bank/Direction (Type)	Configuration Function	Recommendations for SPIx4	Description
CFGBVS	0 Input (Dedicated)	Configuration banks voltage select	Connect directly to GND if using 1.8V SPI flash as in this application note.	<p>Determines the I/O voltage operating range and voltage tolerance for the dedicated configuration bank 0, and for the configuration pins in bank 65. CFGBVS selects the operating voltage for the dedicated bank 0 at all times in all UltraScale FPGAs, and for the multi-purpose bank 65 during configuration. Connect CFGBVS High or Low per the bank voltage requirements. If the V_{CCO_0} supply for bank 0 is supplied with 2.5V or 3.3V, this pin must be tied High (connected to V_{CCO_0}). Tie CFGBVS to Low (connect to GND) only if the V_{CCO_0} for bank 0 is less than or equal to 1.8V. When bank 65 is used for configuration, it should have the same voltage as bank 0.</p> <div>  <p>CAUTION! To avoid device damage, this pin must be connected correctly to either V_{CCO_0} or GND.</p> </div>
EMCCLK	65 Input (Multi-function)	External master configuration clock	Recommended if top configuration clock speeds are required. See Maximum Configuration Clock Frequency for details. Otherwise, this input can be left unconnected as in this application note.	Optional external clock input for running the configuration logic in a master mode (versus the internal configuration oscillator). The FPGA can optionally switch to EMCCLK as the clock source instead of the internal oscillator for driving the internal configuration engine. The EMCCLK frequency can optionally be divided via a bitstream setting and is forwarded for output as the master CCLK signal.

Master SPI Configuration Files

This section briefly overviews the various configuration files used for UltraScale FPGA master SPI configuration and summarizes the tools used to generate these files. Instructions for creating these files is provided in the following sections.

SPI x4 Programming Files

UltraScale FPGA Bitstream (.bit)

The default binary bitstream used for FPGA configuration. This version of the bitstream has a header that includes metadata used by the tools when working with the bitstream. This information is stripped off by the Vivado tools when configuring the FPGA via JTAG.

UltraScale FPGA Bitstream (.bin)

A version of the FPGA bitstream containing only the configuration data and with the metadata present in the .bit file removed. This file can be used to program the SPI flash for the x1, x2, and x4 widths by the Vivado tools.

UltraScale FPGA Intel HEX (.mcs)

An ASCII text file format that contains hexadecimal representation of the programming bitstream with addressing and data size information added in. An MCS is required for master SPI x8 and BPI programming, but is optional for SPI x1, x2, and x4 and is not covered in this application note. If required, an MCS file can be generated with the write_cfgmem Tcl command. For more information, run the following command from the Vivado tools Tcl console:

```
write_cfgmem -help
```

Before generating a bitstream for master SPI configuration, some properties must be enabled specific to SPI configuration. These settings include:

```
BITSTREAM.CONFIG.CONFIGRATE  
BITSTREAM.CONFIG.SPI_32BIT_ADDR  
BITSTREAM.CONFIG.SPI_BUSWIDTH  
BITSTREAM.CONFIG.SPI_FALL_EDGE  
CONFIG_VOLTAGE  
CFGBVS  
BITSTREAM.GENERAL.COMPRESS  
BITSTREAM.CONFIG.EXTMASTERCLK_EN
```


BITSTREAM.CONFIG.CONFIGRATE

(Configuration rate (MHz): 3, 6, 9, 12, 22, 33, ...)

Sets the nominal frequency for CCLK when EMCCLK is not enabled when configuring in master mode. To get the list of possible values, run the following command in the Vivado tools with an open synthesized or implemented design for the target FPGA:

```
list_property_value BITSTREAM.CONFIG.CONFIGRATE [current_design]
```

Recommended setting: 33

Note: This application note chooses 33 as a generally safe configuration rate. Most designs that have given proper consideration to SPI layout can run at much higher configuration rates. See [Maximum Configuration Clock Frequency](#) for instructions on how to calculate the maximum acceptable configuration rate. Also note that signal integrity problems on the CCLK line could cause even the lowest frequency to fail. For example, if CCLK routing leads to reflections at the SPI flash C pin, double clocking could occur.

BITSTREAM.CONFIG.SPI_32BIT_ADDR

(Enable SPI 32-bit address style: NO, YES)

Instructs the SPI configuration logic to issue 4-byte addressing commands as listed in [Table 2](#). Set this to Yes for SPI flash densities of 256 Mb or larger.

Recommended setting:

- YES for flash densities over 128 Mb
- NO for 128 Mb or smaller densities.

BITSTREAM.CONFIG.SPI_BUSWIDTH

(Bus width: None, 1, 2, 4, 8)

Sets the data width that SPI data is read from the SPI flash. This should be set to 4 for UltraScale FPGA SPI configuration as recommended in this application note. Setting to 1 can be useful for configuration bring-up.

Recommended setting: 4

BITSTREAM.CONFIG.SPI_FALL_EDGE

(Enables the FPGA to use a falling edge clock for the SPI data capture: NO, YES)

The SPI flash devices output data on the falling edge. Normally, the FPGA clocks this data on the rising edge of CCLK. Setting this to Yes causes the read data to be clocked on the following edge, improving the timing margins, and allows faster CCLK rates.

Recommended setting: YES

CONFIG_VOLTAGE

(Configuration voltage: 1.5, 1.8, 2.5, 3.3)

Informs the tools of the voltage used for configuration. Set this to the voltage being used on the configuration banks (bank 0 and bank 65 if using EMCCLK). This application note assumes a voltage of 1.8V and sets this value 1.8.

Recommended setting: This setting must match the configuration voltage on the target board.

CFGBVS

(Configuration bank voltage selection: GND, VCCO)

Informs the tools of the voltage driving the CFGBVS pin. If using 1.8V as in this application note and bank 65 is also at 1.8V or none of the configuration pins from the bank (such as EMCCLK) are being used, this can be set to GND. See the "Configuration Banks Voltage Select" section in *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] for more details.

Recommended setting: This setting must match the actual connection on the CFGBVS pin.

BITSTREAM.GENERAL.COMPRESS

(Enable bitstream compression: TRUE, FALSE)

Informs the tools to compress the generated bitstream in a manner that can be uncompressed during configuration by the FPGA. Xilinx recommends enabling this to allow faster configuration and SPI programming times. However, do not rely on configuration bitstream sizes when calculating required storage space because small design changes can result in significantly different compression ratios. For calculating storage needs, refer to the "Bitstream Length for UltraScale FPGAs" table in *UltraScale Architecture Configuration User Guide* (UG570).

Recommended setting: YES

BITSTREAM.CONFIG.EXTMASTERCLK_EN

(Enable external configuration clock and set divide value: DISABLE, DIV-1, DIV-2, DIV-3, DIV-4, ...)

Instructs the configuration to switch to using the EMCCLK pin as the source for the CCLK early in the configuration process. This can enable significantly higher configuration rates but requires using the EMCCLK pin from bank 65. For information on the configuration rate improvements, see [Configuration Time](#) and [Maximum Configuration Clock Frequency](#). Additional DIV values are available. To get the full list for your targeted FPGA, run the following command in the Vivado tools with an open synthesized or implemented design for the target FPGA:

```
list_property_value BITSTREAM.CONFIG.EXTMASTERCLK_EN [current_design]
```

Recommended setting: DIV-1 (or appropriate divide value) if the fastest possible configuration rates are required, otherwise DISABLE.

These properties are set via constraints in the design. They can be set in an XDC constraints file or specified at run time via Tcl commands. The Vivado IDE automatically sets constraints based on selections made in the Bitstream Settings dialog. The recommended method is to add the settings via the constraints file by adding the following recommended constraints to the design constraints file:

```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 1.8 [current_design]
set_property CFGBVS GND [current_design]
set_property BITSTREAM.CONFIG.SPI_32BIT_ADDR YES [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property BITSTREAM.CONFIG.SPI_FALL_EDGE YES [current_design]
```

Generate Bitstream: Vivado Design Suite IDE Example

Use the following IDE flow in your Vivado tools design based in Project Mode to generate a bitstream for master SPI x4 configuration:

1. Add the recommended constraints from [SPI x4 Programming Files](#) to the design's constraints file. This step should ideally be done before synthesis and implementation. Adding constraints after synthesis or implementation causes these runs to be marked as out-of-date upon saving the constraints file. If no other constraints (or design files) are otherwise changed, the runs can safely be forced up-to-date using the Force Up-to-Date option as described in [Generating SPI Configuration Constraints: IDE Flow](#).
2. In the Flow Navigator, locate the Program and Debug tab and click the **Bitstream Settings** button as shown in [Figure 7](#).

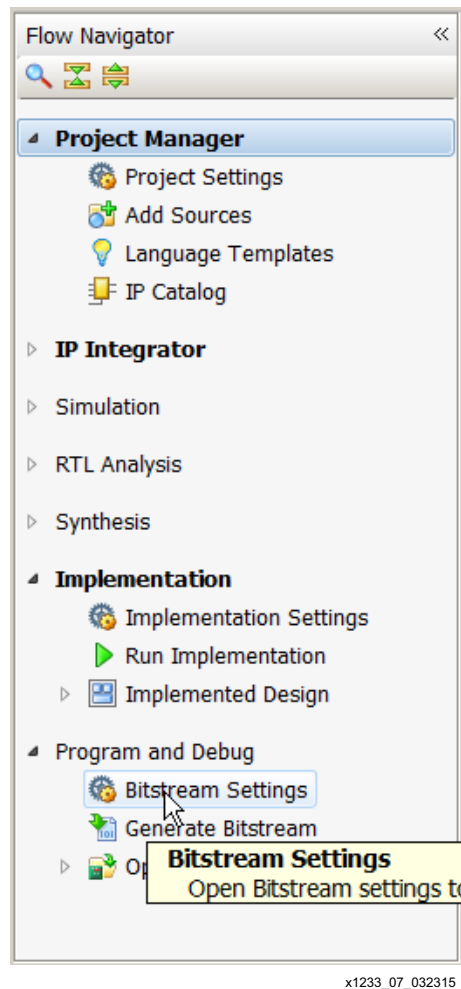
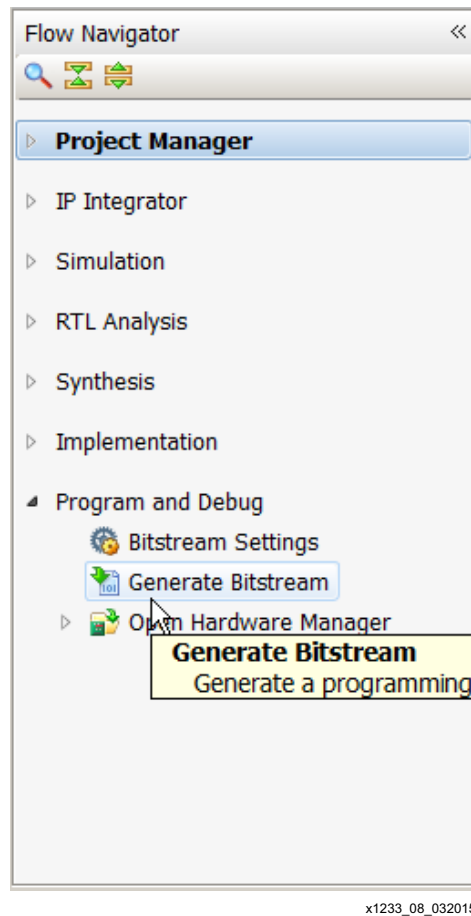


Figure 7: Flow Navigator

3. In the Project Settings dialog, select `-bin_file` which instructs the `write_bitstream` command to generate a headerless bitstream for programming the SPI flash. Click **OK** to accept this change.
4. In the Flow Navigator, click **Generate Bitstream** under the Program and Debug flow or select **Generate Bitstream** from the Flow menu.



x1233_08_032015

Figure 8: Flow Navigator

This starts the Generate Bitstream flow. Upon completion, if no errors are encountered, a bitstream that can be used for SPI programming is found at:

```
<Project_Dir>\<Project_Name>.runs\impl_1\
```

Because the `-bin_file` option was selected earlier, the bitstream was generated as a BIN file with a `.bin` extension in addition to the standard `.bit` file. The `.bin` file is used for programming the SPI flash in [SPI Programming File Generation](#).

Generate Bitstream: Vivado Tools Tcl Batch File Example

As an alternative to the IDE flow presented in the previous section, a Tcl batch file is presented here that can be used with a non-project Vivado tools flow. For this flow to succeed, there must have been a checkpoint created previously. This script assumes the checkpoint is saved in a subdirectory of the current directory from which the script is run:

```
## Non-project Tcl flow to generate a BIN file used for SPI flash
# programming
# NOTE: write_checkpoint must have been run prior to running this script
open_checkpoint {post_route.dcp}
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 1.8 [current_design]
```

```
set_property CFGBVS GND [current_design]
set_property BITSTREAM.CONFIG.SPI_32BIT_ADDR YES [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property BITSTREAM.CONFIG.SPI_FALL_EDGE YES [current_design]
write_bitstream -verbose -force -bin_file spi_programming.bit
```

SPI Programming File Generation

The Vivado Design Suite can optionally be used to generate an MCS file for programming the SPI flash. This is a requirement for SPI x8 because two separate programming files are needed. However, for SPI x1, x2, or x4, an MCS file is not required and this application note uses the .bin file created in the previous section to program the SPI flash.

Program the SPI Flash

The Vivado Design Suite is capable of programming the SPI flash attached to the FPGA by downloading an indirect programming bitstream to the target FPGA that contains an SPI controller. The Vivado tools communicate with the SPI controller via a JTAG cable to transfer the programming file into the SPI flash. This in-system programming feature can enable the testing and debugging of multiple design iterations in the prototyping phase and for debugging. This feature is not intended for high-volume production programming. For production programming, consider solutions from BPM Microsystems or Data I/O.

Programming the SPI Flash: Vivado Design Suite IDE Example

Programming of the SPI flash via the Vivado IDE can happen as part of the bitstream generation flow covered previously or can be run without opening a project. If the Vivado tools are not currently loaded with an implemented design, they can be opened in the Hardware Manager flow. When opened directly in the Hardware Manager flow without opening a project, the Flow Navigator window is not available, but otherwise should match these steps:

1. Ensure the board is connected and a programming cable is connected. Turn on the power to the target board.

2. Open the Vivado tools Hardware Manager by selecting **Open Hardware Manager** under the Program and Debug flow as shown in Figure 9. Alternatively, select **Flow > Open Hardware Manager**.

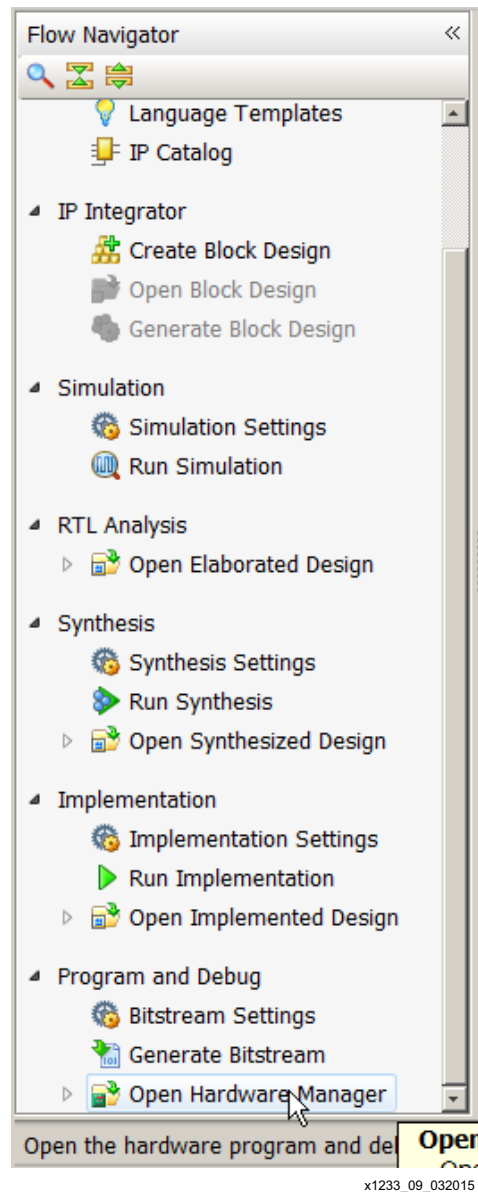


Figure 9: **Open Hardware Manager**

- Open and connect to the hardware target by clicking on the **Open Target** link in the Hardware Manager pane and then selecting **Auto Connect** as shown in Figure 10.

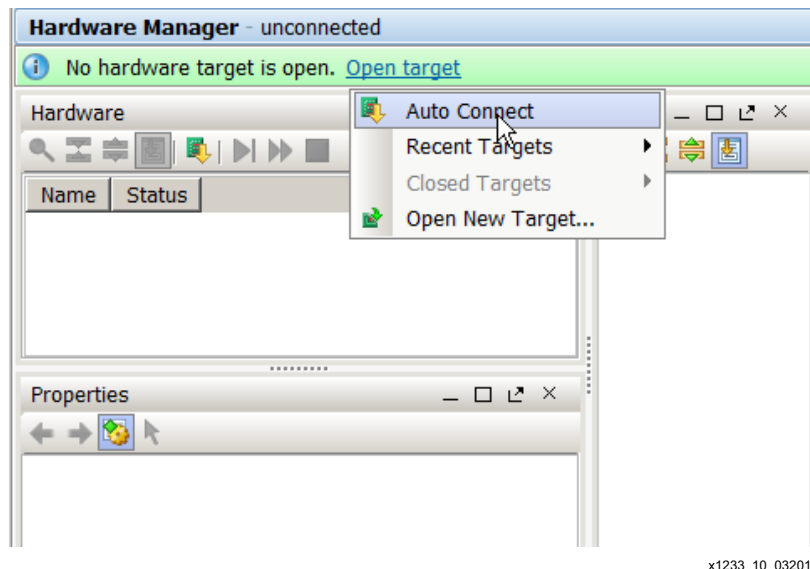


Figure 10: Hardware Manager

Note: This step should automatically connect to the UltraScale FPGA target if it is connected to the workstation on which you are running the Vivado Design Suite, the cable is properly connected, and the board is powered. However, it is also possible to connect remotely to another workstation that is running hw_server and is physically connected to the target board via a programming cable. See the "Opening a New Hardware Target" section in *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 8] for more details if this flow is needed or auto connection fails.

- Launch the Add Configuration Memory Device dialog by right clicking the target FPGA in the Hardware Manager as shown in Figure 11.

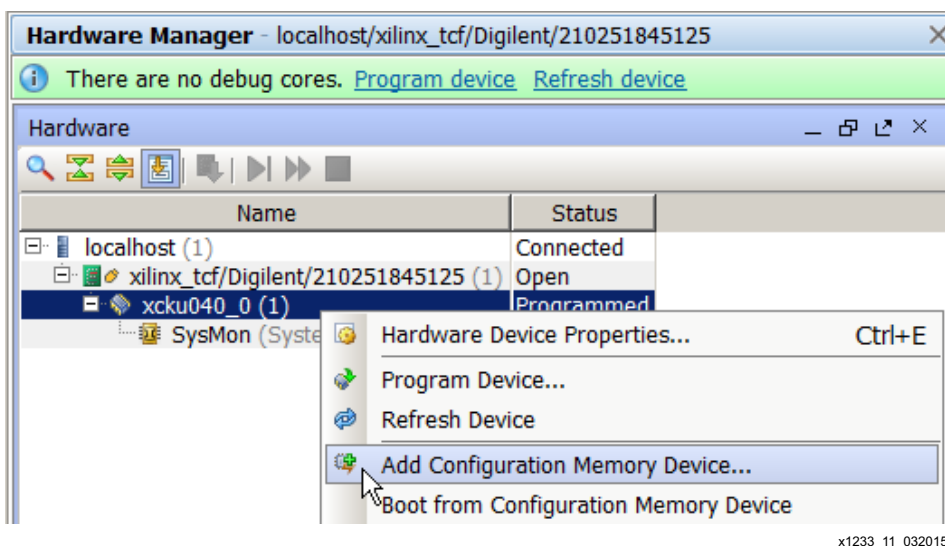
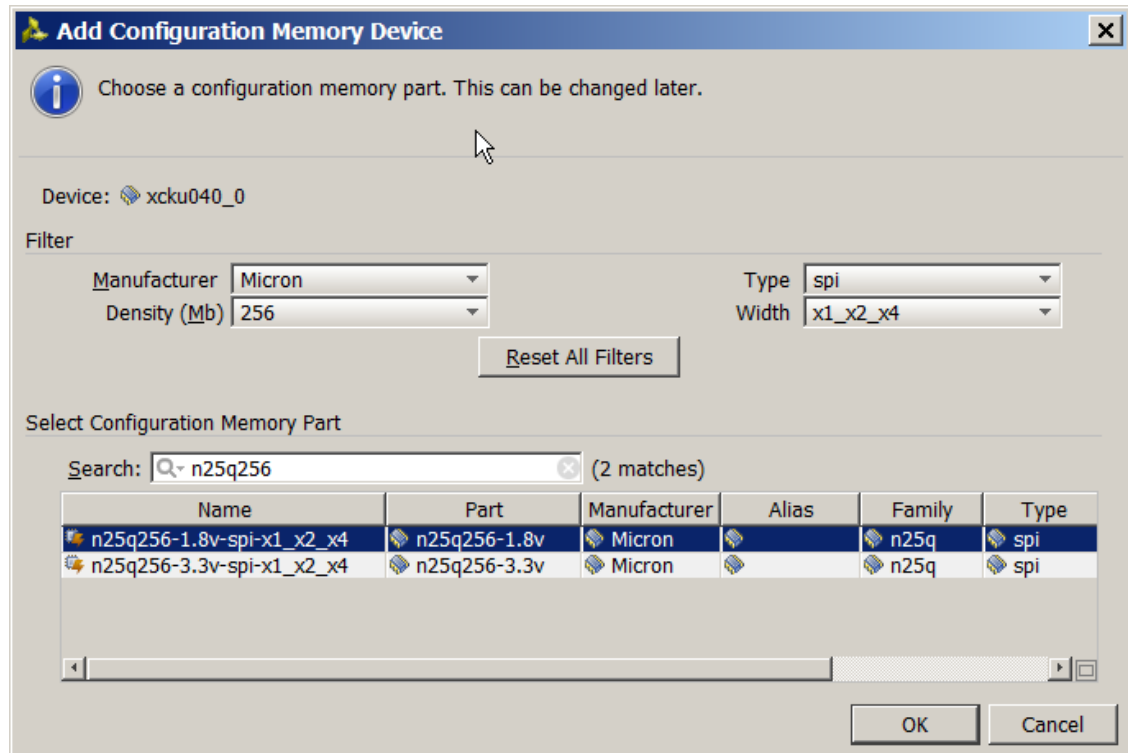


Figure 11: Add Configuration Memory Device

- In the Add Configuration Memory Device window, select the proper SPI flash from the list. The filter options and the search field can both be used to greatly simplify the search and narrow down the options as shown in Figure 12. Click **OK**.



x1233_12_032015

Figure 12: Add Configuration Memory Device

Note: For SPI widths x1, x2, or x4, use SPI flash devices that end with "-x1_x2_x4" (or possibly "-x1" or "-x1_x2") and not ending in "-x1_x2_x4_x8". The latter is only for use with master SPI x8 configuration mode.

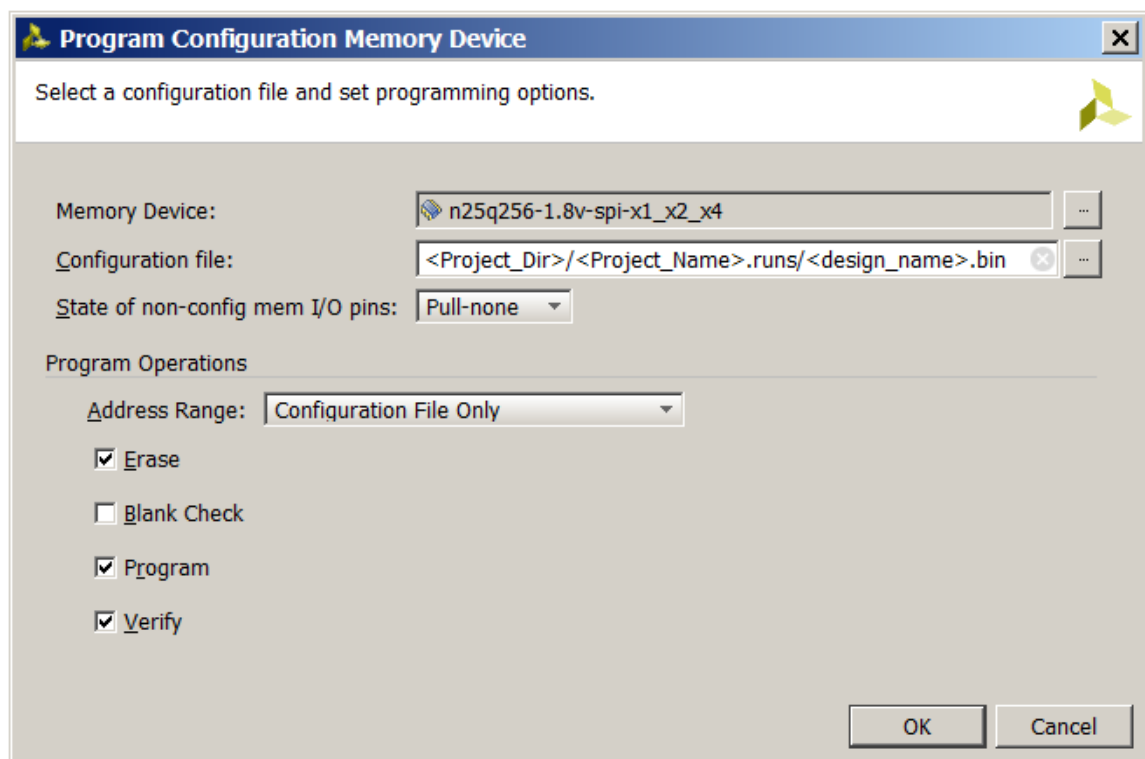
- Click **OK** in the Add Configuration Memory Device Completed dialog box that appears next to program the configuration memory devices at this time.
- In the Program Configuration Memory Device dialog, click on the browse button to the right of the Configuration file field and select the .bin created previously. By default, the .bin file is located in the impl_1 project directory in the following subdirectory:

```
<Project_Dir>\<Project_Name>.runs\impl_1\
```

Note: The location of the Vivado tools project subdirectory is referred to as <Project_Dir> in this application note and the project name is referred to as <Project_Name>.

- Change the **State of non-config mem I/O pins** if required for your board. This specifies an internal termination on the configuration pins while the programming bitstream is loaded. This option instructs the Vivado tools to use a programming file that has the appropriate BITSTREAM.CONFIG.UNUSEDPIN setting to be used to program the SPI flash. For example, if your board had a peripheral connected to non-configuration pins that must not be pulled up when the board is powered (and the normal design has pull-downs on these signals or always actively drives these pins), you can select **Pull-down** and the temporary programming bitstream will have internal pull-downs on all SelectIO interface pins.

9. Set the **Program Operations** as needed for your design and click **OK** when finished (Figure 13):
- **Address Range:** Specify to erase just enough sectors to cover the size of the programming file (**Configuration File Only**) or the entire SPI flash device (**Entire Configuration Memory Device**).
 - **Erase:** Erase the sectors or entire device as indicated by **Address Range**.
 - **Blank Check:** Check that the sectors as indicated by **Address Range** are blank.
 - **Program:** Program the SPI device with the **Configuration** file.
 - **Verify:** Read back the programming contents and verify they match the **Configuration** file.



x1233_13_032015

Figure 13: Program Configuration Memory Device

Programming should begin as soon as **OK** is clicked. A Programming Configuration Memory Device window is loaded showing the progress of the various stages selected in the Program Operations dialog. The erase operation status is not estimated because erase time can vary greatly depending on the existing contents of the SPI flash. Programming is a slow process and takes some time to complete depending on several factors. For an estimate, see [Programming Time](#).

Programming the SPI Flash: Vivado Tools Tcl Batch File Example

As an alternative to the IDE flow presented in the previous section, a Tcl batch file is presented here that can be used with a non-project Vivado tools flow. For this flow to succeed, there must have been a .bin file created previously. Save the following Tcl script as `program_spi.tcl` or adjust the instructions on sourcing with the Vivado tools as appropriate:

```
# Simple Vivado script to program a SPI flash
#
# The board should be connected via a programming cable and powered prior to
# running. The programming file is specified by the variable
# "programming_files" in this example
#
# Run this script from a Vivado command prompt:
# vivado -mode batch -source program_spi.tcl

open_hw
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets]
open_hw_target

# Set the current Xilinx FPGA device. If more than one FPGA is in the JTAG
# chain, you may need to use the get_hw_devices command to help set the proper
# one with the current_hw_target command current_hw_device [lindex [get_hw_devices] 0]

# Set my_mem_device variable for the SPI flash device get_cfgmem_parts can be
# used to find the supported flash. See "help get_cfgmem_parts" in the Vivado
# Tcl Console when in the Hardware Manager for options which can help narrow
# the search. UG908 also lists supported SPI flash devices. Be sure to use the
# parts ending with _x1_x2_x4 for width x1, x2, and x4 and parts ending with
# _x1_x2_x4_x8 for Dual Quad SPI (x8 width).
set my_mem_device [lindex [get_cfgmem_parts {n25q256-1.8v-spi-x1_x2_x4}] 0]

# Set a variable to point the to BIN file to program
set programming_files {top.bin}

# Create a hardware configuration memory object and associate it with the
# hardware device. Also, set a variable with which to point to this object
set my_hw_cfgmem [create_hw_cfgmem -hw_device \
[lindex [get_hw_devices] 0] -mem_dev $my_mem_device]

# Set the address range used for erasing to the size of the programming file
set_property PROGRAM.ADDRESS_RANGE {use_file} $my_hw_cfgmem

# Set the programming file to program into the SPI flash
set_property PROGRAM.FILES $programming_files $my_hw_cfgmem

# Set the termination of unused pins when programming the SPI flash
set_property PROGRAM.UNUSED_PIN_TERMINATION {pull-none} $my_hw_cfgmem

# Configure the hardware device with the programming bitstream
program_hw_devices [lindex [get_hw_devices] 0]
# Set programming options
# Do not perform a blank check, but erase, program and verify
set_property PROGRAM.BLANK_CHECK 0 $my_hw_cfgmem
set_property PROGRAM.ERASE 1 $my_hw_cfgmem
set_property PROGRAM.CFG_PROGRAM 1 $my_hw_cfgmem
set_property PROGRAM.VERIFY 1 $my_hw_cfgmem

# Now program the part
program_hw_cfgmem -hw_cfgmem $my_hw_cfgmem
```

Configure the Target FPGA

After SPI flash programming has completed successfully, initiate FPGA configuration by either power cycling the board or asserting and deasserting PROGRAM_B. Configuration should start and complete in the amount of time that can be determined with the calc_config_time option in the Vivado tools. See [Configuration Time](#) for more details.

Appendix

Programming Time

The Vivado Design Suite 2014.4 was used to measure the time to erase, program, and verify the contents of the N25QU256A on the KCU105 board. These times are provided for an uncompressed XCKU040 bitstream (~128 Mb) using the board's embedded USB cable at 15 MHz ([Table 4](#)). These values are for reference only and not guaranteed timing.

Table 4: Programming Time

Operation	Time (s)
Erase ⁽¹⁾	163
Program	223
Verify	62

Notes:

1. Erase time might vary significantly based upon the pre-programmed content to be erased.

For other UltraScale FPGA configuration sizes, see the "Device Resources and Configuration Bitstream Lengths" section in *UltraScale Architecture Configuration User Guide* (UG570) [[Ref 1](#)].

Configuration Time

Configuration time is a function of configuration bitstream size divided by the configuration rate times the configuration width. However, there are several other factors such as setting the start-up options MATCH_CYCLE or LCK_CYCLE which increase the number of configuration clock cycles by a variable amount to wait for clock management circuits in the design to achieve lock. For the following formulas, these settings are assumed to be unset. All other factors are minor and can be ignored for a first order estimate.

T = Configuration time (s)

S = Bitstream size (bits)

CR = Configuration clock frequency (Hz)

W = Data bus width (bits)

$$T = \frac{S}{(CR \times W)} \quad \text{Equation 1}$$

If using the internal configuration clock source to drive CCLK, the clock tolerance needs to be considered. For UltraScale FPGAs, this tolerance parameter $F_{MCCKTOL}$ can be found in the UltraScale data sheet. So the range of configuration time is:

$$\frac{S}{(1 + F_{MCCKTOL})CR \times W} < T < \frac{S}{(1 - F_{MCCKTOL})CR \times W} \quad \text{Equation 2}$$

Maximum Configuration Clock Frequency

The following parameters are defined to help describe the calculations for finding the maximum possible configuration clock for master SPI configuration.

F_{CCLK,MAX}: Maximum Configuration Clock Frequency

In master SPI mode, the FPGA delivers the configuration clock through the CCLK pin. The source of this clock can be an internal clock source or the EMCCLK pin. Using EMCCLK has a significant advantage in that it can be driven by a low tolerance source but does require the use of an additional pin from bank 65. If using the external clock source via the EMCCLK clock pin, set the property to some value other than DISABLED such as DIV-1 or DIV-2. The second option divides the clock at the EMCCLK by 2. The other available divide options that enable EMCCLK can be found with the following Tcl command run with an open implemented or synthesized design:

```
list_property_value BITSTREAM.CONFIG.EXTMASTERCCLK_EN [current_design]
```

CR: Configuration Rate Setting

If using the internal clock source, the possible configuration rates can be found by opening the implemented design in the Vivado tools and running this command:

```
list_property_value BITSTREAM.CONFIG.CONFIGRATE [current_design]
```

T_{SPI_TCO}: SPI Clock Low to Output Valid

This is the time the SPI flash requires for data to be valid after the negative edge of CCLK. The timing parameter name might have the symbol name of t_{CLQV} or t_V in the vendor's data sheet. For this example, the N25Q256A11ESF40x t_{CLQV} value of 5 ns is used [Ref 3] because it is assumed there is only one load (the FPGA's D[03:00] pins) on the D[3:0] pins of the SPI flash. If there are additional loads that would increase the load above 10 pF, a T_{SPI_TCO} value of 7 ns would need to be used as described in the N25Q256A11ESF40x data sheet.

The load presented by the FPGA can be determined from the IBIS models by summing the package capacitance and the I/O pad capacitance. For example, the package capacitance for the

FFVA1156 package can be found in the [Package] section of the Kintex UltraScale IBIS model as shown in [Table 5](#).

Table 5: Package Capacitance for FFVA1156 Package

FFVA1156 Variable	Typical (pF)	Minimum (pF)	Maximum (pF)
C_pkg	2.52	1.87	3.16

In this example, the FPGA I/O is configured for LVCMOS18_F_12 in a high-range bank. The I/O pad capacitance can be found in the [Model] HR_LVCMOS18_F_12 as shown in [Table 6](#).

Table 6: I/O Pad Capacitance

[Model] HR_LVCMOS18_F_12	Typical (pF)	Minimum (pF)	Maximum (pF)
C_comp	5.9136	5.8211	5.9685

This example therefore assumes a capacitive load of 9.13 pF with no other significant loads on the data nets. If other loads are present, they should be considered when selecting which T_{SPITCO} value to use [\[Ref 4\]](#).

T_{SPIDCC} : FPGA Data Setup Time

This is the setup time required for the data pins relative to the clock edge. The rising clock edge is used by default, but this greatly limits the working period as the data is output from the SPI flash on the falling edge. The UltraScale FPGA has the option to switch to the falling edge (BITSTREAM.CONFIG.SPI_FALL_EDGE). The equations and example in this section assume this has been set to YES.

T_{TPD} : Board Trace Propagation Delay

This is the board trace propagation delay of both the CCLK to C pin and the longest propagation delay of any of the data pins. For the following example, a rule of thumb of 165 ps per inch is used. For more accurate results, other techniques such as IBIS simulation are recommended. In this example, we assume trace lengths of 6 inches from the FPGA to the SPI flash.

$F_{CCLK,MAX}$ = Maximum configuration clock frequency

T_{SPITCO} = Flash clock Low to data valid

T_{SPIDCC} = FPGA data setup

T_{TPD} = Board trace propagation delay

$$F_{CCLK,MAX} \leq \frac{1}{(T_{SPITCO} + T_{SPIDCC} + T_{TPD})} \quad \text{Equation 3}$$

T_{SPITCO} = 5 ns (for a 10 pF load, from N25Q256A11ESF40x data sheet [\[Ref 3\]](#))

T_{SPIDCC} = 3 ns (from *Kintex UltraScale FPGAs Data Sheet* (DS892) [\[Ref 6\]](#))

T_{TPD} = 2 ns (Total trace propagation delay assuming 6 inch traces and using 165 ps/inch trace propagation delays)

$$F_{\text{CCLK,MAX}} \leq 100 \text{ MHz}$$

However, if using the internal oscillator to drive CCLK, the FMCKTOL tolerance must be accounted for. So the maximum configuration rate must not exceed $F_{\text{CCLK,MAX}}$ at the highest possible frequency given the tolerance.

$$CR \leq \frac{F_{\text{CCLK,MAX}}}{(1 + \text{FMCKTOL})} \quad \text{Equation 4}$$

Assuming FMCKTOL is $\pm 35\%$, the maximum configuration rate is:

$$CR \leq 100 \text{ MHz}/1.35$$

$$CR \leq 74.2 \text{ MHz}$$

To find the maximum BITSTREAM.CONFIG.CONFIGRATE property run this command in an open synthesized or implemented design:

```
list_property_value BITSTREAM.CONFIG.CONFIGRATE [current_design]
```

Vivado Design Suite User Guide: Programming and Debugging (UG908) [Ref 8] also lists the possible values for BITSTREAM.CONFIG.CONFIGRATE:

3, 6, 9, 12, 22, 33, 40, 50, 57, 69, 82, 87, 90, 110, 115, 130, 148.

Based off these possible values and a CR maximum of 74 MHz, BITSTREAM.CONFIG.CONFIGRATE should be set to 69.

Note: Equation 4 should be used to determine the maximum frequency that the SPI flash can safely operate and still deliver the bitstream reliably. The SPI flash delivers data on the falling edge of the clock. The default for UltraScale FPGAs is to capture data on the rising edge of the clock. For Equation 4 to be true, the SPI flash configuration setting to enable capturing data on the falling edge must be set to YES (BITSTREAM.CONFIG.SPI_FALL_EDGE YES). See [Generate Bitstream: Vivado Design Suite IDE Example](#) or [Generate Bitstream: Vivado Tools Tcl Batch File Example](#) for instructions on making this setting.

Selecting an SPI Flash

Several factors should be considered when selecting an SPI flash device:

- **Storage capacity:** The minimum density required is always the size of the FPGA configuration bitstream. See the "Device Resources and Configuration Bitstream Lengths" section in *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] for a list of bitstream sizes. If the design requires multiple bitstreams, multiply the size of the bitstream accordingly. Although the Vivado Design Suite allows bitstream compression, Xilinx recommends not relying on observed compression ratios in determining final storage needs because compression can potentially vary significantly between builds.
- **Data bus width:** For UltraScale FPGAs, always select devices that support x4 configuration. This is because the D[03:00] data lines required for master SPI configuration are dedicated in bank 0 and are not available as user I/O. Most, if not all, SPI flash devices supported for UltraScale FPGAs do support quad SPI, but double check that the quad output fast read (6Bh) and the quad output fast read 32-bit address (6Ch) commands are supported.
- **Performance:** Xilinx recommends using x4 for master SPI configuration because all the required data pins D[03:00] are dedicated. If higher frequencies are also required, pay

attention to the Clock Low to Data Valid timing parameter of the SPI flash data sheet. This, combined with the D[03:00] setup times on the UltraScale FPGA are likely to be the limiting factor in upper CCLK frequencies as opposed to the maximum frequency supported by the SPI flash. Using EMCCLK as a clock input is also critical if the highest possible configuration rates are required. Be sure to set BITSTREAM.CONFIG.SPI_FALL_EDGE to "YES" as mentioned in [SPI x4 Programming Files](#). See [Configuration Time](#) for more details.

Note: For higher configuration rates, EMCCLK is strongly recommended.

- I/O voltage: Almost all UltraScale FPGAs are capable of configuring in SPI x4 at 3.3V. However, other board needs might require 1.8V on the configuration bank. Most supported SPI flash devices are available in 1.8V variants. Some SPI flash devices might come with an option that allows the SPI flash to interface with the FPGA at 1.8V but still operate at a V_{CC} of 3.3V. These devices likely have an increased Clock Low to Data Valid time when operated in this mode. This should be taken into consideration if higher CCLK rates are required.
- Package type: There are a variety of potential packages to meet your needs. The Vivado Design Suite does not distinguish between packages, and verification is typically done on the SOIC16 package. This package is convenient if debugging is required because the pins are easily probed. If your board requirements or the flash availability dictate a different package, make sure the CCLK, FCS_B, D00_MOSI, and D01_DIN pins are accessible for probing in case debugging is required. If BGA packaging is used, vias under the packages can likely serve as probe points.

Master SPI x1, x2, and x8

Xilinx recommends using master SPI in the x4 width because the four data pins D[03:00] required for SPI x4 are in the dedicated bank 0. UltraScale FPGAs also provide x1 or x2 SPI data bus widths that can be used, if preferred. However, the unused D01 and D02 pins are not available for user I/O, so it is recommended to still connected them to the SPI flash.

The dual quad SPI mode (x8) might be required if higher throughput is required. Also, EMCCLK might be required to drive CCLK for higher configuration clock rates.

Master SPI x1 and x2

The x1 and x2 would enable reducing two signals on the board but would not increase the number of I/O pins available to the design as the data pins D[03:00] are located in bank 0 and are dedicated for configuration use. If x1 and x2 is still preferred over x4, the steps in this application note can easily be modified for these widths. The D[03:02] signals do not need to be connected. However, it might be good to pull these signals up to V_{CC0_0} as shown in the "UltraScale FPGA SPI x1/x2 Configuration Interface" figure in *UltraScale Architecture Configuration User Guide* (UG570) [\[Ref 1\]](#). If the board is built for x4, the x1 and x2 modes are still available.

Master SPI x8

The x8 master SPI configuration mode available in UltraScale FPGAs can halve the configuration time over the x4 width. This requires five additional pins from bank 65 as shown in the "Master SPI Configuration Mode" chapter of *UltraScale Architecture Configuration User Guide* (UG570).

Generating SPI Configuration Constraints: IDE Flow

Use the following IDE flow in your Vivado tools design based in Project Mode to generate a bitstream for master SPI x4 configuration. This mode sets the constraints automatically via the IDE options selected as described here:

Note: Before running these steps, be sure to open the implemented design by selecting **Flow Navigator > Implementation > Open Implemented Design**.

1. In the Flow Navigator, locate the Program and Debug tab and click the **Bitstream Settings** button as shown in Figure 14.

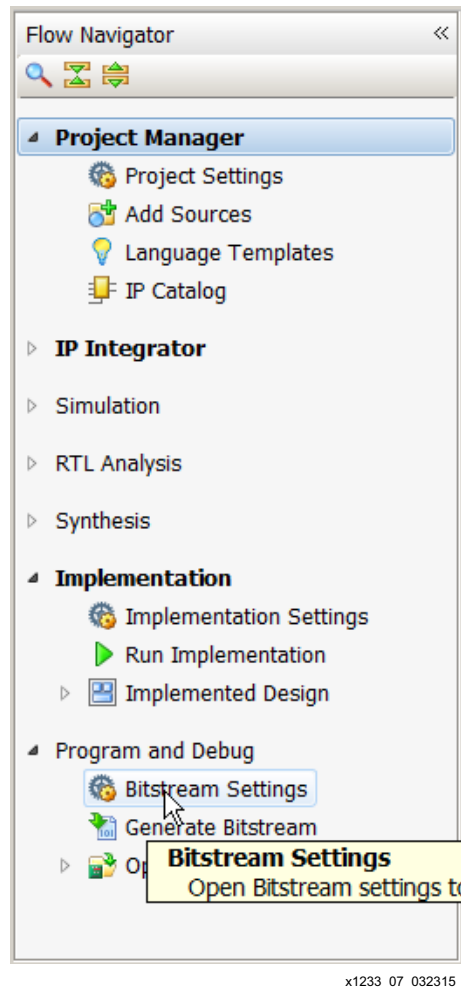
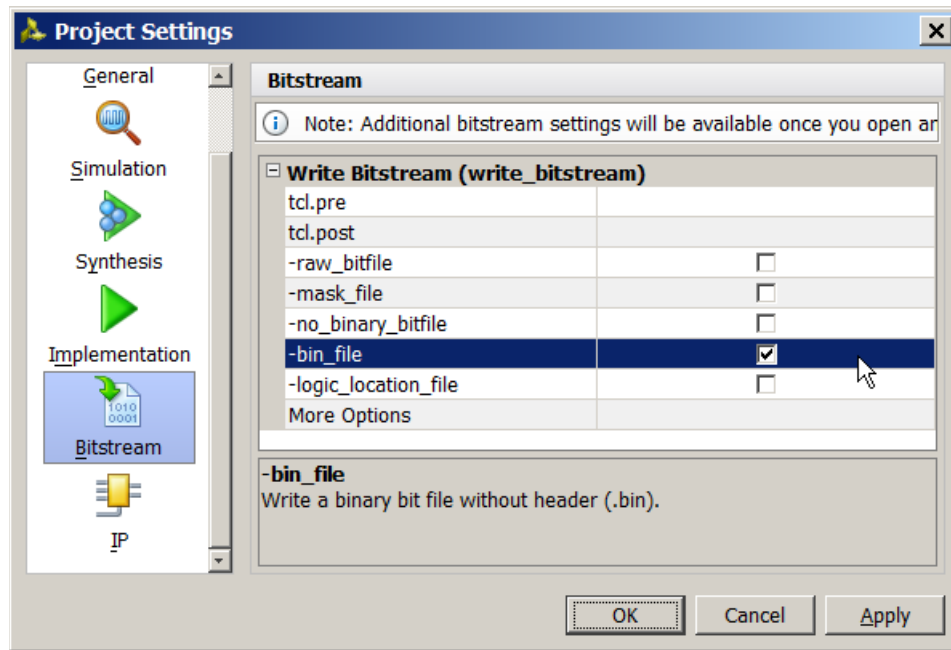


Figure 14: Flow Navigator

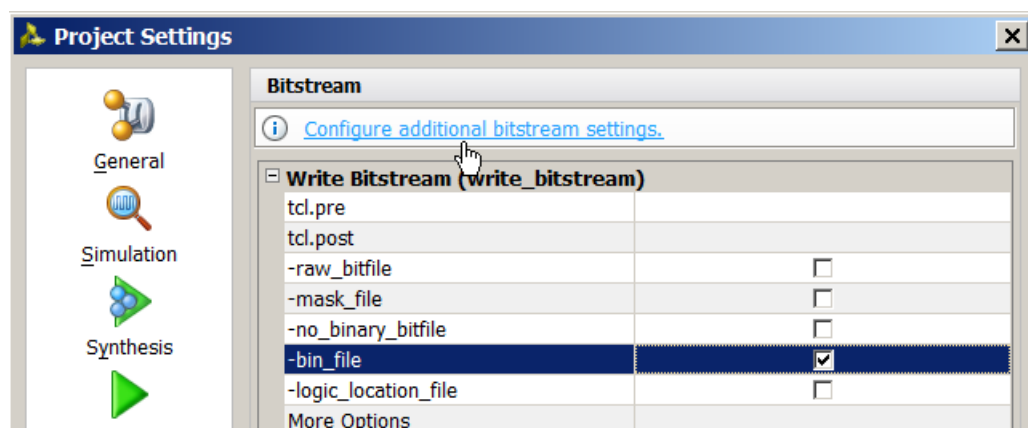
2. In the Project Settings window under the Bitstream settings page, enable the `-bin_file` option as shown in Figure 15.



x1233_15_032415

Figure 15: Enabling `-bin_file`

3. From the same Project Settings window, click on the **Configure additional bitstream settings** link as shown in Figure 16. (This link is only available when the project has an opened synthesized or implemented design.)



x1233_16_032415

Figure 16: Configure Additional Bitstream Settings

4. In the Edit Device Properties window that opens, enable the following property in the General page (Figure 17):
 - Enable bitstream compression: TRUE

Note: Bitstream compression is optional but recommended because it might shorten programming times. However, compression ratios should not be relied upon when calculating SPI densities or configuration times because compression ratios can vary significantly between builds from one design implementation run to the next.

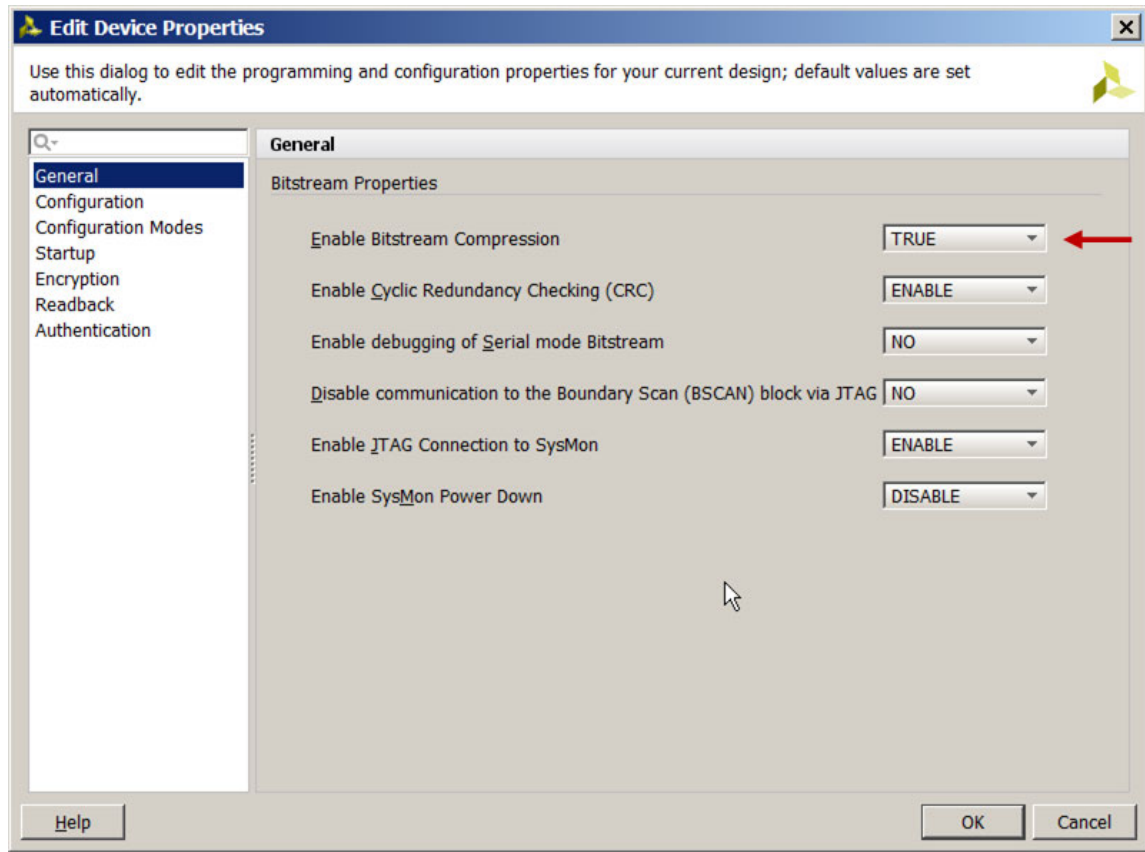
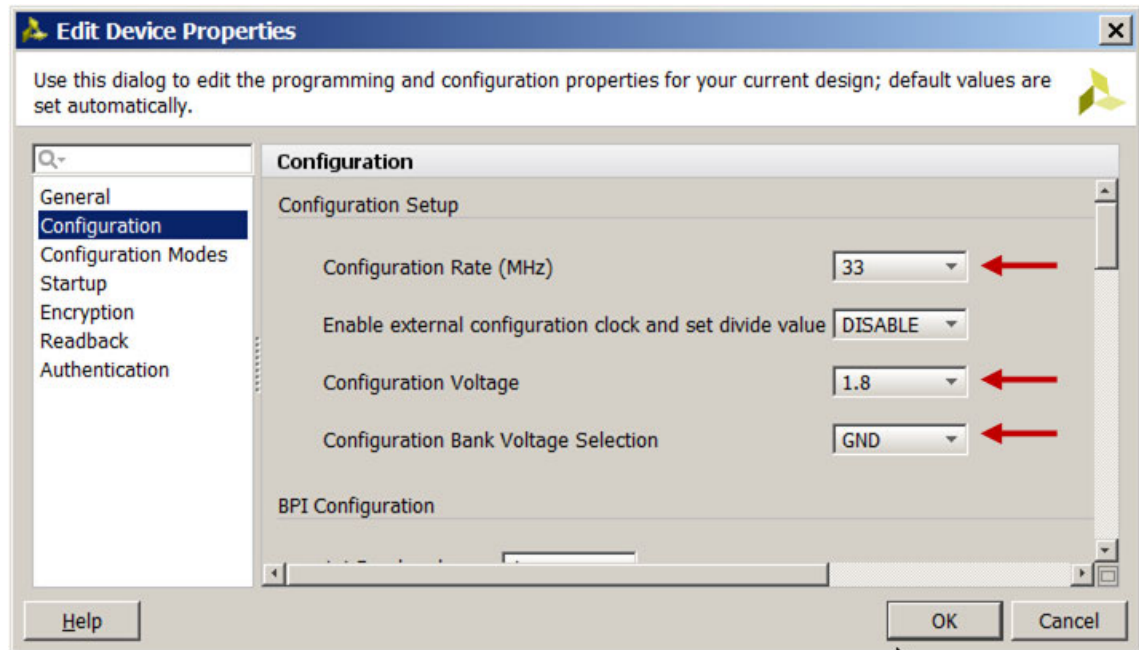


Figure 17: Edit Device Properties (General)

5. Select the Configuration page in the Edit Device Properties window and select these options under the Configuration Setup section (Figure 18):
 - Configuration rate (MHz): 33
 - Configuration voltage: 1.8
 - Configuration bank voltage selection: GND

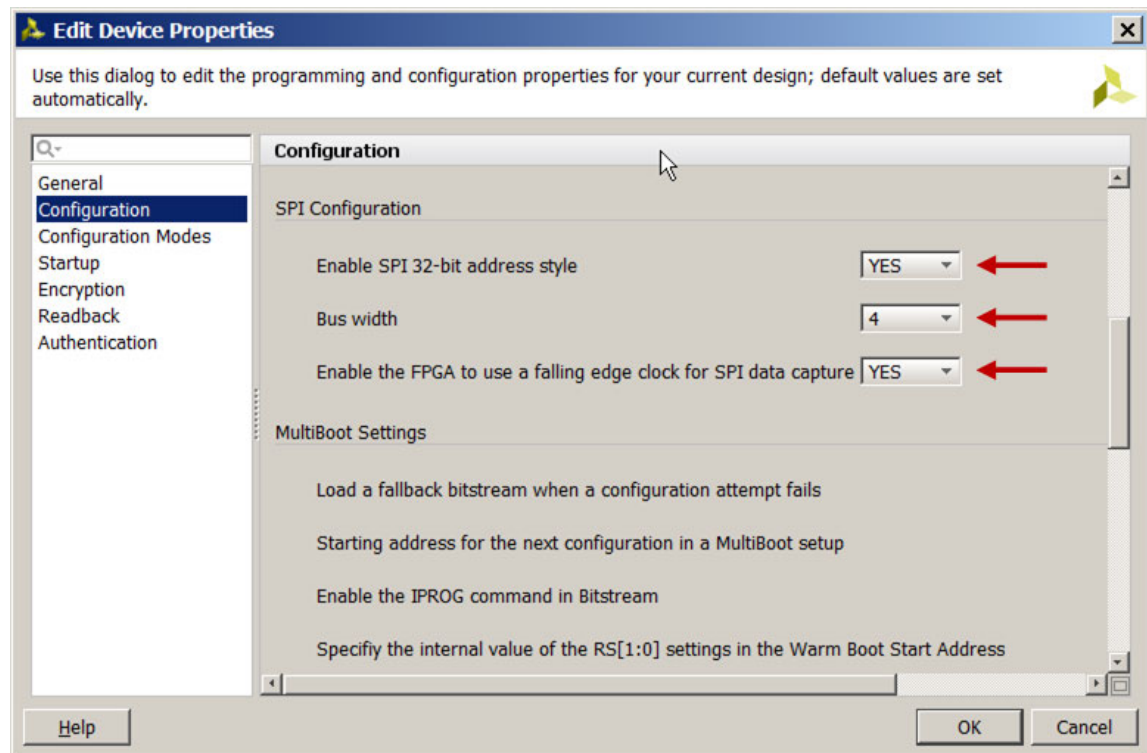
Note: Set configuration voltage to match your board's configuration voltage (the voltage on bank 0) as shown in Figure 18. The above settings are valid for the KCU105 board and might need to be adjusted for different board configurations.



x1233_18_032415

Figure 18: Edit Device Properties (Configuration 1)

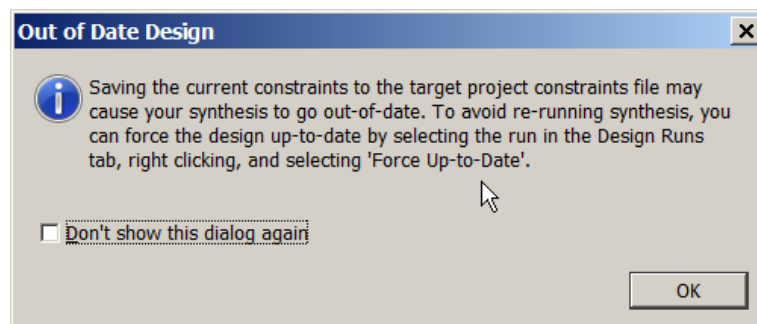
6. On the same page, scroll down to the SPI Configuration section and make the following selections (Figure 19):
 - Enable SPI 32-bit address style: YES
 - Bus width: 4
 - Enable the FPGA to use a falling edge clock for SPI data capture: YES



x1233_19_032415

Figure 19: Edit Device Properties (Configuration 2)

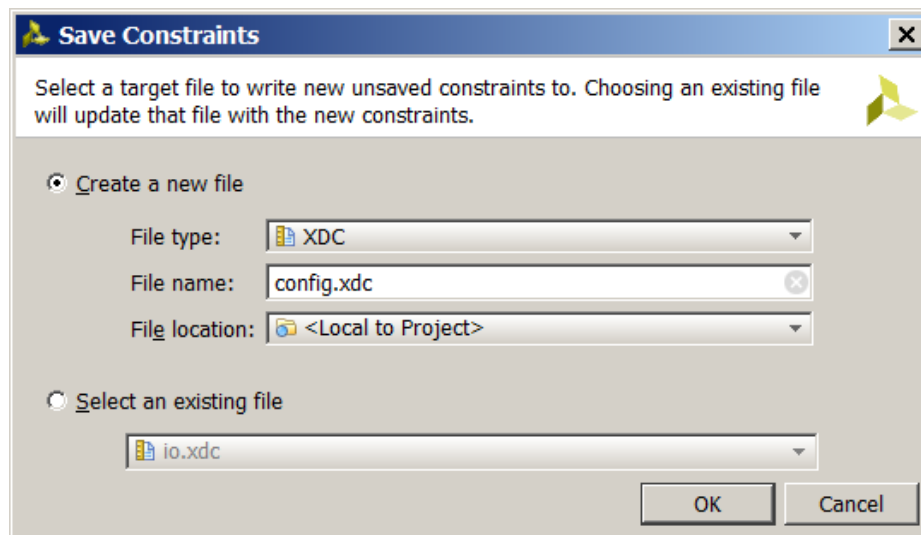
- Click **OK** on the Edit Device Properties dialog to accept the modified bitstream settings. Be sure to save the constraints generated by clicking the **Save** button on the tool bar, typing CTRL + S, or selecting **Save Constraints** from the File menu.
- Saving the constraints might cause two dialogs to appear. The first one will be an Out of Date Design dialog indicating that saving the constraints will cause the design to be out of date and that implementation or synthesis will need to be re-run or the design runs will need to be forced up-to-date. The Force Up-to-Date option is a good choice at this time if no constraints other than those needed for bitstream generation have been changed. This is shown in Figure 20. Click **OK** to close this first window.



x1233_20_032415

Figure 20: Out of Date Design

9. The second window might be a Save Constraints dialog (Figure 21). Either save the constraints just generated in an existing file or create a new file. In this example, a new file called `config.xdc` is created. After creating the constraints file or selecting an existing file, click **OK**.



x1233_21_032415

Figure 21: Save Constraints

Making these settings in the Vivado IDE generates the necessary constraints for master SPI configuration bitstream generation. Saving the constraints writes them to the specified constraints file. The generated constraints can be seen in the Tcl console:

```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [get_designs synth_1]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [get_designs synth_1]
set_property CONFIG_VOLTAGE 1.8 [get_designs synth_1]
set_property CFGBVS GND [get_designs synth_1]
set_property BITSTREAM.CONFIG.SPI_32BIT_ADDR YES [get_designs synth_1]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [get_designs synth_1]
set_property BITSTREAM.CONFIG.SPI_FALL_EDGE YES [get_designs synth_1]
```

10. Force synthesis and implementation up-to-date because the constraints changed above only impact bitstream generation (Figure 22). Select **Design Runs** in the Results Window Area. If the synthesis run has a status indicating it is out of date, right-click the synthesis run and select **Force Up-to-Date**. If only the implementation status shows that it is out of date, right-click the implementation run and select **Force Up-to-Date**.

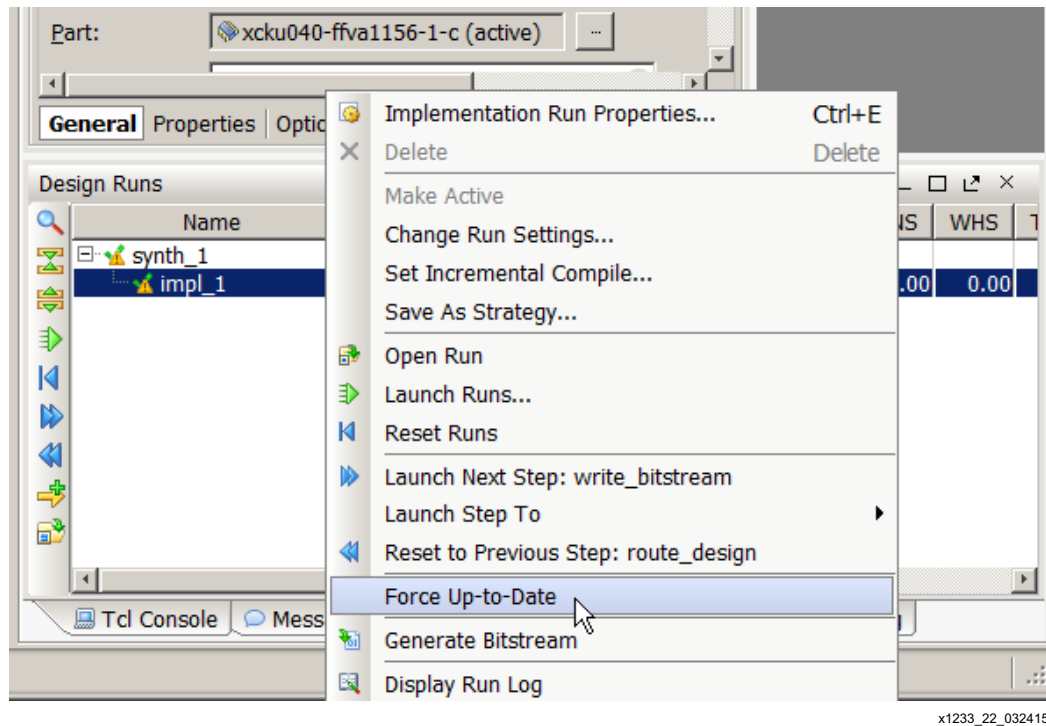


Figure 22: Force Up-to-Date

Debugging and Troubleshooting Guidelines

Configuration issues can have many different causes that are not readily differentiated at the outset. For troubleshooting master SPI configuration failures, the following sections summarize some of the common debugging steps that can help narrow in on the root cause and possible solutions.

Verify Layout

- It is very important that the FPGA signal integrity of the JTAG TCK and FPGA CCLK signals are maintained. Avoid using lengthy connections where possible. Using long connections can result in unwanted noise or voltage waveform reflections that degrade the integrity of FPGA signals. Refer to the “Unidirectional Topographies and Termination” section of *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 9] for additional guidance.
- Review the master SPI x4 schematic for the correct connectivity.
- Scope the CCLK, FCS_B, and D00_MOSI lines. After power-up or following a PROGRAM_B pulse, trigger on the falling edge of FCS_B and verify that the waveform matches the master SPI x1 0Bh command.

Ensure Correct File Generation

- When the EMCCLK option is enabled in bitstream settings, ensure that the I/O standard is defined. EMCCLK is a multi-purpose pin, so by default its I/O standard is undefined. There are two options for allowing the use of EMCCLK:

- Include the following property in the design XDC constraint file:

```
set_property CONFIG_VOLTAGE 1.8 [current_design]
```

- Instantiate EMCCLK into the design by including it as an input, using it in the design, and defining its I/O standard in the constraints file.
- For faster configuration and programming times, use the bitstream compression setting. For troubleshooting purposes, a simple bitstream can be used (that simply toggles a single LED or example) to get higher compression ratios and shorter programming times.
- Ensure the ConfigRate option does not exceed the maximum frequency supported by the target flash and FPGA. Refer to the $F_{MCCKTOL}$ timing parameter in the appropriate UltraScale device data sheet for the FPGA CCLK tolerance. For debugging, consider dropping the configuration rate lower than the maximum calculated frequency or even leaving it at the default. See [Maximum Configuration Clock Frequency](#) for instructions on calculating the maximum configuration rate.
- Prepare bitstream (as a .bin file) from Vivado Design Suite:
 - Test the bitstream (.bit) file via loading with JTAG to verify that configuration works as expected.
 - Check that the binary bitstream (.bin) file was generated at the same time as the bitstream if it is behaving differently.

Vivado Tools Indirect Programming

- Check that the Vivado Design Suite was able to read the ID of the SPI flash. Two common failures indicate a connectivity or power problem:

```
Mfg ID : 00 Memory Type : 00 Memory Capacity : 00  
Mfg ID : FF Memory Type : FF Memory Capacity : FF
```

- Enable the Erase, Program, and Verify operations.
- Check that the correct bitstream was programmed.
- Ensure that the maximum cable speed for indirect programming is not exceeded.
- Ensure the JTAG chain integrity is good:
 - Perform a basic FPGA IDCODE operation to verify the connections.
 - Program a simple bitstream into the FPGA.

- Review the FPGA status register for additional insight. Figure 23 shows the CONFIG STATUS register after a successful configuration.

Name	Status
172.19.69.216 (1)	Connected
xilinx_tcf/xilinx/000015de4a0301 (1)	Open
xcku040_0 (2)	Programmed
SysMon (System Monitor)	
n25q256-1.8v-spi-x1_x2_x4_0	

Hardware Device Properties	
xcku040_0	
REGISTER	
BOOT STATUS	00000001
CONFIG STATUS	00000001
BIT00 CRC ERROR	0
BIT01 DECRYPTOR ENABLE	0
BIT02 PLL LOCK STATUS	1
BIT03 DCI MATCH STATUS	1
BIT04 END OF STARTUP (EOS) STATUS	1
BIT05 GTS CFG B STATUS	1
BIT06 GWE STATUS	1
BIT07 GHIGH STATUS	1
BIT08 MODE PIN M[0]	1
BIT09 MODE PIN M[1]	0
BIT10 MODE PIN M[2]	0
BIT11 INIT B INTERNAL SIGNAL STATUS	1
BIT12 INIT B PIN	1
BIT13 DONE INTERNAL SIGNAL STATUS	0
BIT14 DONE PIN	1
BIT15 IDCODE ERROR	0
BIT16 SECURITY ERROR	0
BIT17 SYSTEM MONITOR OVER-TEMP AL	0
BIT18 CFG STARTUP STATE MACHINE P	100
BIT21 SECURITY STATUS	100
BIT24 RESERVED	0
BIT25 CFG BUS WIDTH DETECTION	00
BIT27 SECURITY AUTH ERROR	0
BIT28 PUDC B PIN	0
BIT29 BAD PACKET ERROR	0
BIT30 CFGBVS PIN	0
BIT31 RESERVED	0

General **Properties**

x1233_23_032415

Figure 23: CONFIG STATUS Register

References

1. *UltraScale Architecture Configuration User Guide* ([UG570](#))
2. *UltraScale FPGA BPI Configuration and Flash Programming* ([XAPP1220](#))
3. *Micron Serial NOR Flash Memory*
www.micron.com/~media/documents/products/data-sheet/nor-flash/serial-nor/n25q/n25q_256mb_1_8v.pdf
4. IBIS Models Downloads
<http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/device-models.html>
5. Kintex UltraScale FPGA KCU105 Evaluation Kit
www.xilinx.com/products/boards-and-kits/kcu105.html
6. *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS892](#))
7. *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS893](#))
8. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
9. *UltraScale Architecture PCB Design User Guide* ([UG583](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/29/2015	1.0	Initial Xilinx release.
09/20/2016	1.1	Updated T_{SPITCO} : SPI Clock Low to Output Valid. Added T_{TPD} : Board Trace Propagation Delay.
09/23/2016	1.1.1	Corrected typographical errors.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe

performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY. AUTOMOTIVE APPLICATIONS DISCLAIMER

© Copyright 2015–2016 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.