



# AN 006: Configuring Trion<sup>®</sup> FPGAs

---

AN006-v6.6  
November 2025  
[www.efinixinc.com](http://www.efinixinc.com)



# Contents

<b>About Configuring Trion® FPGAs.....</b>	<b>4</b>
Bitstream Size.....	4
Configuration Time.....	5
Planning Your Device Pinout.....	5
Other Factors to Consider.....	6
<b>Configuration Pins.....</b>	<b>7</b>
<b>FPGA Configuration Modes.....</b>	<b>10</b>
Selecting the Configuration Mode.....	10
About SPI Clocking and Sampling.....	11
SPI Active Mode.....	12
SPI Active Mode for SIP Packages.....	17
SPI Active Mode without CSI.....	20
Clocking.....	20
SPI Passive Mode.....	21
SPI Passive Mode for SIP Packages.....	27
SPI Passive Mode without CSI or CBUS2.....	28
JTAG Mode.....	30
<b>Using FPGA, MCU, and SPI Flash Devices Together.....</b>	<b>35</b>
<b>Flash Programming Modes.....</b>	<b>38</b>
<b>Power Up.....</b>	<b>41</b>
Power Up Sequence.....	41
Power Supply Current Transient.....	42
Power Up Configuration Circuitry Recommendation.....	43
Unused Resources and Features.....	44
<b>Configuration Sequence.....</b>	<b>45</b>
<b>Support for Multiple Images.....</b>	<b>46</b>
<b>Configuring Multiple FPGAs.....</b>	<b>48</b>
Daisy Chaining with a SPI Flash Device.....	48
Daisy Chaining with a Microcontroller or Microprocessor.....	50
<b>Resistors in Configuration Circuitry.....</b>	<b>51</b>
<b>Configuration Timing.....</b>	<b>52</b>
<b>Selecting the Right SPI Flash Device.....</b>	<b>53</b>
<b>Supported Flash Devices.....</b>	<b>53</b>
<b>Connecting Programming Hardware.....</b>	<b>54</b>
SPI Programming Connections.....	54
JTAG Programming Connections.....	56
Supported Download Cables.....	59
<b>Using the Efinity Programmer.....</b>	<b>60</b>
Generate a Bitstream (Programming) File.....	60
Working with Bitstreams.....	61
Edit the Bitstream Header.....	61
Export to Raw Binary Format.....	61
Export to .svf Format.....	61
Convert to Intel Hex Format at the Command Line.....	62
Combine Bitstreams and Other Files.....	63

SPI Programming.....	64
Program a Single Image.....	64
Program Multiple Images (CBSEL).....	64
Program Multiple Images (Internal Reconfiguration).....	65
Program Multiple Images (Bitstream and Data).....	65
Program a Daisy Chain.....	66
JTAG Programming.....	67
Trion Family JTAG Device IDs.....	67
Program a Single Image.....	67
Program Using a JTAG Chain.....	68
Program using a JTAG Bridge.....	69
JTAG Programming with FTDI Chip Hardware.....	70
FTDI Programming at the Command Line.....	70
Identifying FTDI URLs.....	73
Using the Command-Line Programmer.....	74
Project-Based Programming Options.....	75
<b>Verifying Configuration.....</b>	<b>77</b>
<b>Installing USB Drivers.....</b>	<b>80</b>
Installing the Linux USB Driver.....	80
Installing the Windows USB Driver.....	81
<b>Appendix: Programming the Flash Using JTAG Bridge (Legacy).....</b>	<b>82</b>
<b>Revision History.....</b>	<b>83</b>

# About Configuring Trion® FPGAs

This document describes how to configure Trion® FPGAs. These FPGAs contain volatile Configuration RAM (CRAM) that you must configure with the desired logic function (via a bitstream) upon power-up and before the core enters normal operation. The Efinity® software generates the bitstream, which is design dependent.



**Learn more:** Refer to the [Efinity Software User Guide](#) for information on how to generate the bitstream.

## Bitstream Size

The bitstream size is dependent on the FPGA you choose and the configuration parameters you set in the Efinity software.



**Note:** In the Efinity software, you can optionally add header information to the bitstream file. This header information can add up to 1k bytes to the configuration size. The following maximums include the optional header size.

*Table 1: Trion® FPGA Bitstream Size*

FPGA	Maximum Supported Configuration Bits (Single Image)	Packages
T4	1,348,184	All
T8	1,394,584	BGA49, BGA81
	5,255,968	QFP144
T13	5,261,920	All
T20	5,255,968	QFP144
	5,445,600	WLCSP80, QFP100F3, BGA169, BGA256
	8,003,744	BGA324, BGA400
T35	8,139,168	All
T55	27,675,040	All
T85	28,042,400	All
T120	28,409,760	All

## Configuration Time

The FPGA configuration time depends on the frequency and data bus width. To estimate the configuration time for a given FPGA, use the following equation:

$$\text{Configuration Time} = \text{Bitstream Size} \div (\text{Configuration Clock Frequency} \times \text{Data Bus Width})$$



**Note:** The maximum configuration clock frequency depends on the configuration mode and implementation.

For example:

- *T8 FPGA*—approximately 1.38 Mbits of configuration data:
- *Configuration clock frequency*—10 MHz
- *T8 Configuration data bus width*—8 bits

Configuration time:  $1.38 \text{ Mbits} \times 100 \text{ ns} \div 8 = 17.25 \text{ ms}$

## Planning Your Device Pinout

The configuration mode you choose affects your design's pinout. You should decide which mode you will use and plan for it before performing floorplanning or pin selection for your logic design.

Active and passive configuration modes use multi-function pins during configuration. When configuration completes, these multi-function pins are available for general use. JTAG configuration uses dedicated configuration pins that cannot be used for other functions. Additionally, the configuration mode you choose can affect the voltage restrictions for the I/O bank that contains the configuration pins.

Efnix<sup>®</sup> recommends that you:

- Choose the configuration mode(s). Consider the primary configuration mode as well as configuration modes you may need for debugging or future updates.
- Find the pin and the bank locations for the configuration mode(s).
- Understand how you use these pins and any restrictions when using multi-function configuration pins as standard I/O pins. For example, consider internal and external pull-ups or pull-downs, connections to external devices, etc.



**Note:** In some situations, you may want to use a multi-function configuration pin as an output pin in user mode. If the pin is driven by an external device during configuration, the source that drives this pin during configuration must be tri-stated before the device enters user mode and user logic begins driving it. Otherwise, the drivers can be in contention, and can damage the pin.

- For each set of configuration pins, determine the common required I/O voltage support for the required configuration bank. You can only use compatible I/O standards elsewhere in that bank.

## Other Factors to Consider

Although configuration is typically a one-time event independent of device operation, your configuration choices can affect your design options. Make configuration decisions early in the design cycle to eliminate challenges later:

- Do you need to support JTAG configuration for debugging purposes?
- How can you provide easy access to the configuration control and status pins for debugging?
- What multi-function pins are you using in your logic design and are they active during configuration? If they are, check for conflicts with other uses of these pins.

Additionally, you should:

- Provide quality signal integrity for key signals during PCB layout, including the configuration clock (even though configuration can operate at a low frequency).
- Understand the configuration sequence to reduce configuration time.
- Generate the configuration bitstream for your FPGA using Efinity tools.

# Configuration Pins

Some configuration pins are dedicated, and some are dual-purpose.

- Dedicated pins cannot be used as general purpose I/O.
- During configuration, use dual-purpose pins as described in this document for the configuration mode you are using. After configuration (in user mode), you can use these pins as general-purpose I/O.

**Table 2: Dedicated Configuration Pins**


These pins cannot be used as general-purpose I/O after configuration.

All the pins are in internal weak pull-up during configuration except for TCK and TDO.

Pins	Direction	Description	External Weak Pull-Up/ Pull Down Requirement
CDONE	I/O	Configuration done status pin. CDONE is an open drain output; connect it to an external pull-up resistor to VCCIO. When CDONE = 1, the configuration is complete and the FPGA enters user mode. You can hold CDONE low and release it to synchronize the FPGAs entering user mode.	Pull up
CRESET_N	Input	Active-low FPGA reset and re-configuration trigger. Pulse CRESET_N low for a duration of $t_{\text{reset\_N}}$ before releasing CRESET_N from low to high to initiate FPGA re-configuration. This pin does not perform a system reset.	Pull up
TCK	Input	JTAG test clock input (TCK). The rising edge loads signals applied at the TAP input pins (TMS and TDI). The falling edge clocks out signals through the TAP TDO pin.	Pull up
TMS	Input	JTAG test mode select input (TMS). The I/O sequence on this input controls the test logic operation. The signal value typically changes on the falling edge of TCK. TMS has an internal weak pull-up; when it is not driven by an external source, the test logic perceives a logic 1.	Pull up
TDI	Input	JTAG test data input (TDI). Data applied at this serial input is fed into the instruction register or into a test data register depending on the sequence previously applied at TMS. Typically, the signal applied at TDI changes state following the falling edge of TCK while the registers shift in the value received on the rising edge. Like TMS, TDI has an internal weak pull-up; when it is not driven from an external source, the test logic perceives a logic 1.	Pull up
TDO	Output	JTAG test data output (TDO). This serial output from the test logic is fed from the instruction register or a test data register depending on the sequence previously applied at TMS. The shift out content is based on the issued instruction. The signal driven through TDO changes state following the falling edge of TCK. When data is not being shifted through the device, TDO is set to an inactive drive state (e.g., high-impedance).	Pull up

**Table 3: Dual-Purpose Configuration Pins**

In user mode (after configuration), you can use these dual-purpose pins as general I/O.

Configuration Functions	Direction	Description	External Weak Pull-Up/ Pull Down Requirement
CBUS[2:0]	Input	Configuration bus width select. CBUS has an internal weak pull-up. However, Efinix recommends that you use an external pull-up accordingly. See <i>Selecting the Configuration Mode</i> in <a href="#">AN 006: Configuring Trion FPGAs</a> .	Pull up or pull down <sup>(1)</sup>
CBSEL[1:0]	Input	Multi-image configuration selection pin. This function is not applicable to single-image bitstream configuration or internal reconfiguration (remote update). Connect CBSEL[1:0] to the external resistors for the image you want to use: 00 for image 1 01 for image 2 10 for image 3 11 for image 4 0: Connect to an external weak pull down. 1: Connect to an external weak pull up.	Pull up or pull down <sup>(2)</sup>
CCK	I/O	Passive SPI input configuration clock or active SPI output configuration clock (active low). Includes an internal weak pull-up.   <b>Important:</b> The CCK pin in Q100F3 packages are only available in user mode when the LVDS TX resources are not in use. The CCK pin should not be toggled when any LVDS TX is used.	Optional pull up if required by external load
CDIn	I/O	<i>n</i> is a number from 0 to 31 depending on the SPI configuration. 0: Passive serial data input or active serial output. 1: Passive serial data output or active serial input. <i>n</i> : Parallel I/O. In multi-bit daisy chain connection, the CDI (31:0) connects to the data bus in parallel.	Optional pull up if required by external load
CSI	Input	Chip select. 0: The FPGA is not selected or enabled and will not be configured. 1: Selects the FPGA for all configuration modes. CSI must remain high throughout all configuration modes.	Pull up
CSO	Output	Chip select output. Selects the next device for cascading configuration.	N/A
NSTATUS	Output	Status (active low). Indicates a configuration error. When the FPGA drives this pin low, it indicates either a device mismatch or a failed bitstream CRC check. For Trion® T4, T8 F49, and T8 F81 FPGAs, logic low indicates a configuration error due to ID mismatch.	N/A

<sup>(1)</sup> Optional for x1 mode.

<sup>(2)</sup> Not applicable to single-image or remote update.

Configuration Functions	Direction	Description	External Weak Pull-Up/ Pull Down Requirement
SS_N	I/O	SPI configuration mode select. The FPGA senses the value of SS_N when it comes out of reset (i.e., CRESET_N transitions from low to high). 0: SPI Passive mode; connect to external weak pull down. 1: SPI Active mode; connect to external weak pull up. In active configuration mode, SS_N is an active-low chip select to the flash device (CDI0 - CDI3).	Pull up or pull down
TEST_N	Input	Active-low test mode enable signal. Set to 1 to disable test mode. During all configuration modes, rely on the external weak pull-up or drive this pin high.	Pull up
RESERVED_OUT	Output	Reserved pin during user configuration. This pin drives high during user configuration. F49 and F81 packages only.	N/A
SPI_CS_N	Input	Active-low internal SPI flash memory chip select. Available in QFP100F3 packages only.	Pull up



**Note:** Refer to the column Configuration Functions in `device_pinout.xlsx`.

# FPGA Configuration Modes

Trion® FPGAs have dedicated configuration pins. You select the configuration mode by setting the appropriate condition on the input configuration pins. Trion® FPGAs support the following configuration modes.

**Table 4: FPGA Configuration Modes**

Mode	Description
SPI Active (serial/parallel)	The FPGA loads the bitstream itself from non-volatile SPI flash memory.
SPI Passive (serial/parallel)	An external microprocessor or microcontroller sends the bitstream to the FPGA using the SPI interface.
JTAG	A host computer sends instructions through a download cable to the FPGA's JTAG interface using JTAG instructions.

## Selecting the Configuration Mode

Each configuration interface corresponds to one or more configuration modes and bus widths.

- Select the configuration mode by setting the appropriate condition on the CBUS[2:0], SS\_N, and TEST\_N input pins.
- Set CBUS2, CBUS1, CBUS0, SS\_N, and TEST\_N using a pull-up or pull-down resistor, or drive them with an external active device.
- Do not toggle the mode pins before the FPGA enters user mode.

**Table 5: SPI Hardware Settings**

If you do not make any connections, the default mode is x1 SPI active.

Configuration Mode	Parallel/Serial	CSI	TEST_N	SS_N	CBUS2, CBUS1, CBUS0	Width
SPI Active	Serial	1	1	1	3'b111	x1
	Parallel	1	1	1	3'b110	x2
	Parallel	1	1	1	3'b101	x4
SPI Passive	Serial	1	1	0	3'b111	x1
	Parallel	1	1	0	3'b110	x2
	Parallel	1	1	0	3'b101	x4
	Parallel	1	1	0	3'b100	x8
	Parallel	1	1	0	3'b011	x16
	Parallel	1	1	0	3'b010	x32

The JTAG/boundary-scan configuration interface is always available regardless of pin settings. If you send configuration instructions to the JTAG interface, the Trion® FPGA overwrites the previous configuration.



**Note:** You must set the configuration mode in the Efinity® software; the software includes the mode and other configuration options in the bitstream.

The supported configuration modes are FPGA specific. Refer to your FPGA's data sheet for information on the configuration modes it supports.

## About SPI Clocking and Sampling

*Table 6: SPI Interface Clocking and Sampling*

Mode	Clock	Sampling Edge
Passive	The CCK clock comes from an external device	Positive
Active	The FPGA generates the CCK clock	Positive (not configurable)



**Important:** Refer to the setup and hold times in the data sheet to ensure system timing closure based on your timing budget.

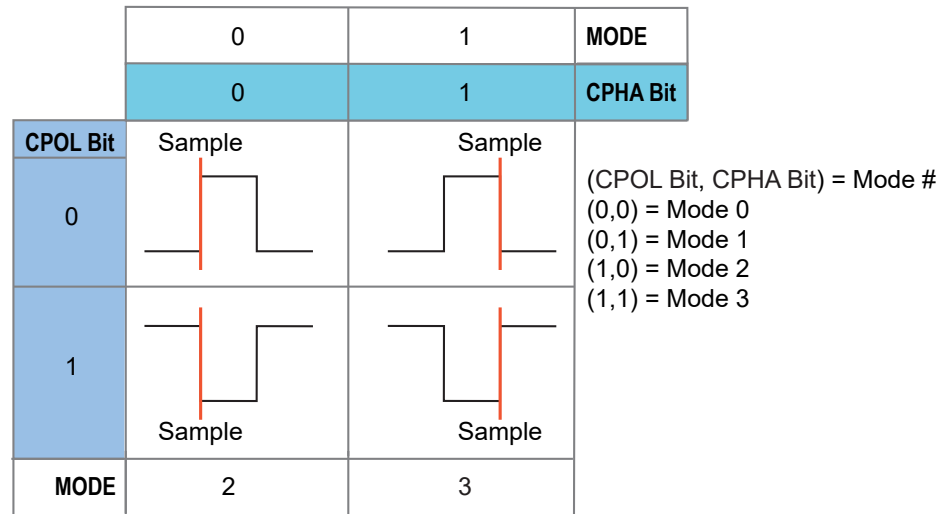
The microprocessor or microcontroller can set the SPI clock polarity (CPOL bit) and the clock phase (CPHA bit) when the interface is idle, which results in four modes, depending on how you set these bits. Use Mode 3 in your microprocessor or microcontroller when programming the FPGA.

*Table 7: SPI Clock Polarity and Phase Modes*

Mode	Clock Polarity when Idle	Data Sampled On	Data Shifted On
0	Low	Rising edge	Falling edge
1	Low	Falling edge	Rising edge
2	High	Falling edge	Rising edge
3	High	Rising edge	Falling edge

Efinix uses Mode 3 for SPI passive mode, which is CPOL bit = 1 and CPHA bit = 1 for all Trion FPGA devices.

**Figure 1: SPI Clock Polarity and Phase Modes Diagram**



## SPI Active Mode

In active mode, the FPGA loads configuration data itself from a configuration bitstream that typically resides in non-volatile memory on the same board. Active modes can be serial or parallel. The FPGA internally generates the configuration clock signal (CCK) and controls configuration by sending a clock or addresses to the flash memory.

The active SPI configuration mode supports low pin count, industry-standard external SPI flash devices to store the bitstream. The FPGA supports a direct connection to the flash device's four-pin SPI interface. Active SPI configuration mode can read from standard 1-bit serial SPI flash devices as well as from flash devices that support x2 and x4 fast output read operations. These modes are proportionally faster than the standard 1-bit SPI interface.



**Note:** Trion® FPGAs only support SPI flash memory with 3-byte addressing mode for configuration.

**Table 8: Active Mode Instructions**

Instruction	Description	SPI Data Width
0BH	Fast Read	x1
3BH	Dual Output Fast Read	x2
6BH	Quad Output Fast Read	x4

The FPGA samples CBUS0, CBUS1, and CBUS2 after power-up or reconfiguration; therefore, you must drive these signals to the correct value.<sup>(3)</sup>

<sup>(3)</sup> Smaller packages may not have CBUS2 bonded out. In this case, CBUS2 is held high in the package.

## Connection Examples



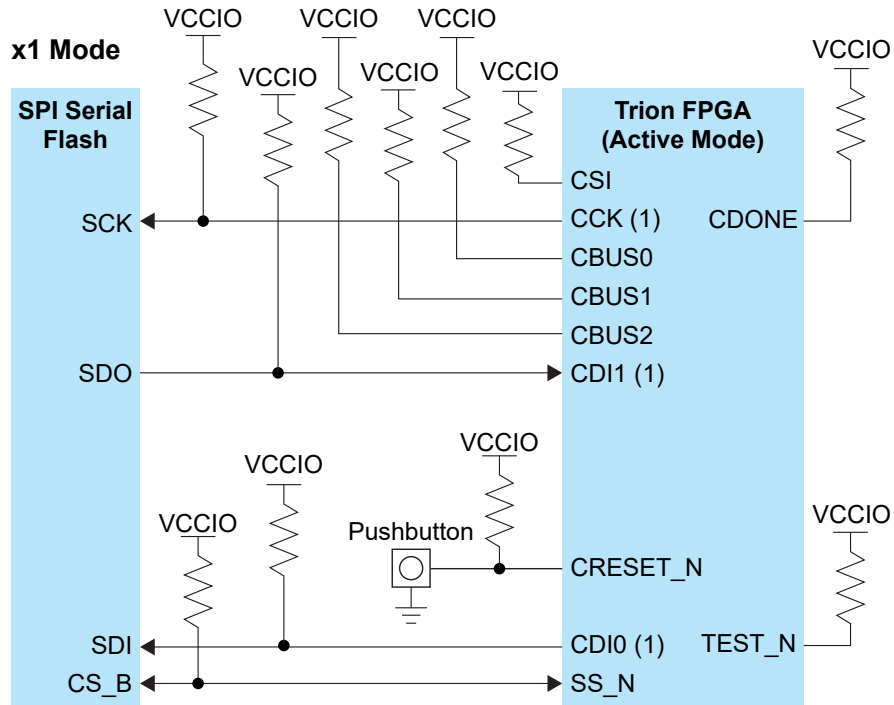
**Note:** Circuitry is required to control the CRESET\_N pin to meet the  $t_{CRESET\_N}$  requirement.



**Learn more:** Refer to [Trion Hardware Design Checklist and Guidelines](#) for the detailed connection requirements.

**Figure 2: Active (x1)**

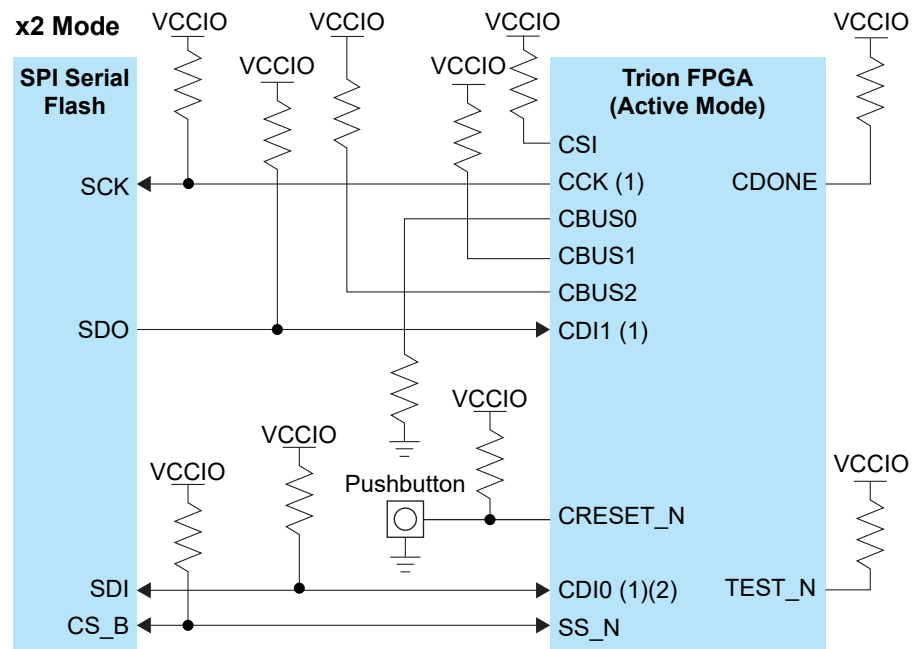
See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



**Note:** (1) The external pull-up is optional unless required by an external load.

Figure 3: Active (x2)

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.

**Notes:**

1. The external pull-up is optional unless required by an external load.
2. In x2, the CDI0 pin is a bidirectional data I/O pin.



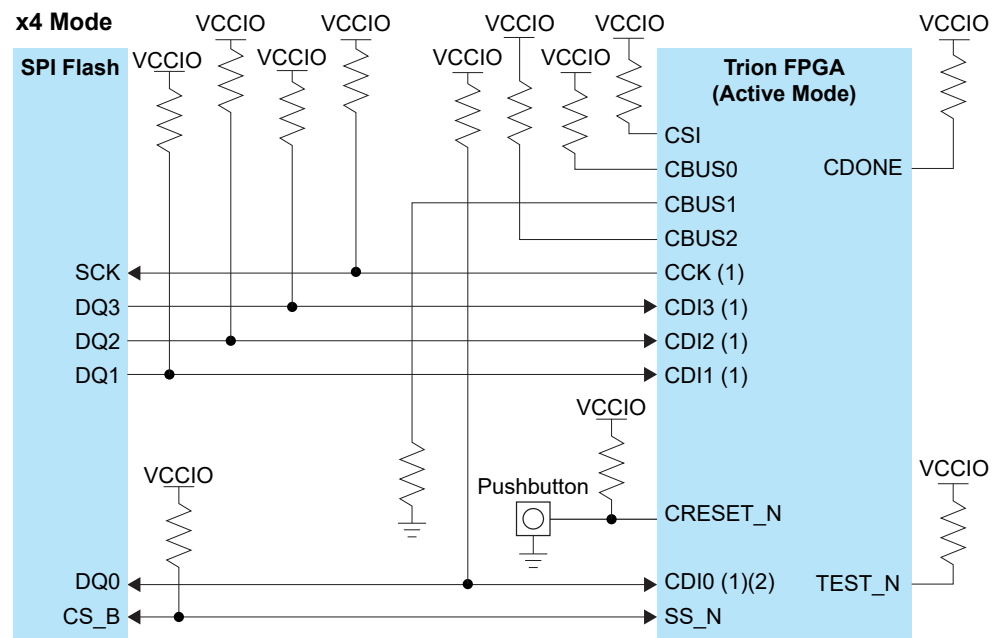
**Note:** The connections for x2 are the same as x1. However:

The modes use different CBUS values (see [Table 5: SPI Hardware Settings](#) on page 10).

In x2 the CDI0 pin is a bidirectional data I/O pin.

Figure 4: Active Quad (x4)

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.

**Notes:**

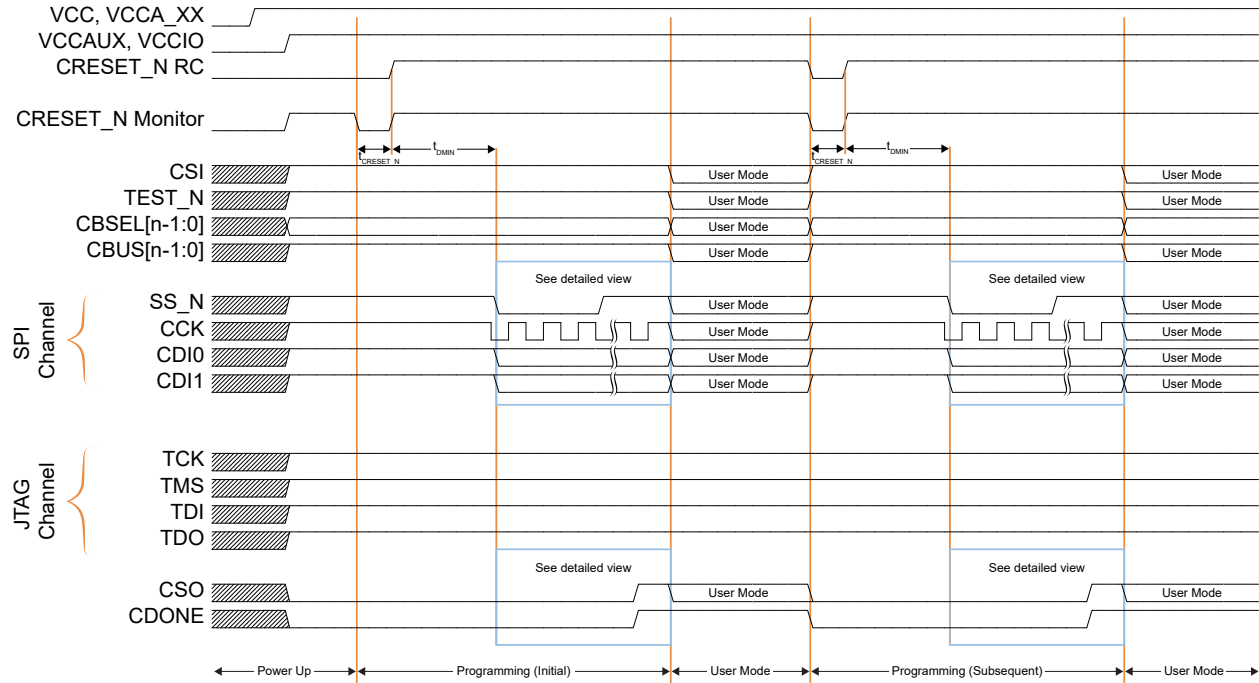
1. The external pull-up is optional unless required by an external load.
2. In x4, the CDI0 pin is a bidirectional data I/O pin.

## Timing

The FPGA supplies the configuration clock and issues instructions to interact with an external flash through the SPI pins. When the FPGA powers up and enters active mode, SS\_N is a weak pullup. Then, the FPGA:

1. Starts configuration by driving SS\_N low to wake up the external SPI flash.
2. Issues a release from power-down instruction to wake up the external SPI flash by driving the CDI0 pin.
3. Waits for at least 30  $\mu$ s.
4. Issues a fast read command to read the content of SPI flash from address 24h'000000. The maximum SPI flash address width for configuration is 24 bits.
5. Optional: when configuration completes, the FPGA issues a deep power-down instruction to force the external SPI flash to enter into deep power-down state.

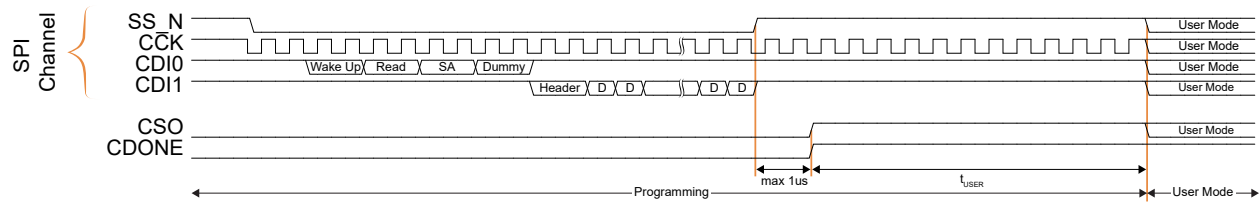
Figure 5: SPI Active (x1) Configuration



**Notes:**

1. CRESET\_N RC refers to the CRESET\_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
2. CRESET Monitor refers to the CRESET\_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 6: SPI Active (x1) Configuration (Detailed View)



**Legend:**

SA: Start Address    D: Data

**Note:** The waveforms are in control block perspective without any optional external pull-up or pull-down resistor connected.

**Note:** Refer to "Power-up Sequence" in the Trion® data sheet for power-up details.

**Learn more:** Refer to the Trion® FPGA data sheet for timing specifications.

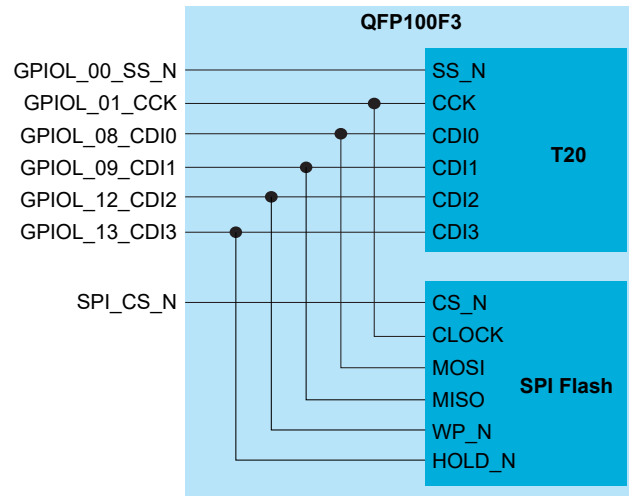
**Important:** Only a single configuration channel can be activated as running either SPI active and JTAG at the same time can result in configuration failure. Therefore, JTAG pins must be inactive during SPI active configuration.

## SPI Active Mode for SIP Packages

Trion® FPGAs in QFP100F3 packages are a system-in-package (SIP) that includes an internal SPI flash that you can use to store configuration bitstreams. However, you can still use an external SPI flash to store the configuration bitstreams.

Depending on the setup, you must observe the following pin connection requirements in addition to the connections shown in [Connection Examples](#) on page 13.

**Figure 7: Connections between FPGA and SPI Flash Inside the QFP100F3 Package**





## Configuration with External Flash

You will need to configure external flash if you have three or more configuration images that exceeds the capacity of the internal SPI flash. You can remove the connection between `SPI_CS_N` and `GPIO` if the external SPI flash capacity is large enough for your application to free up the `GPIO`.

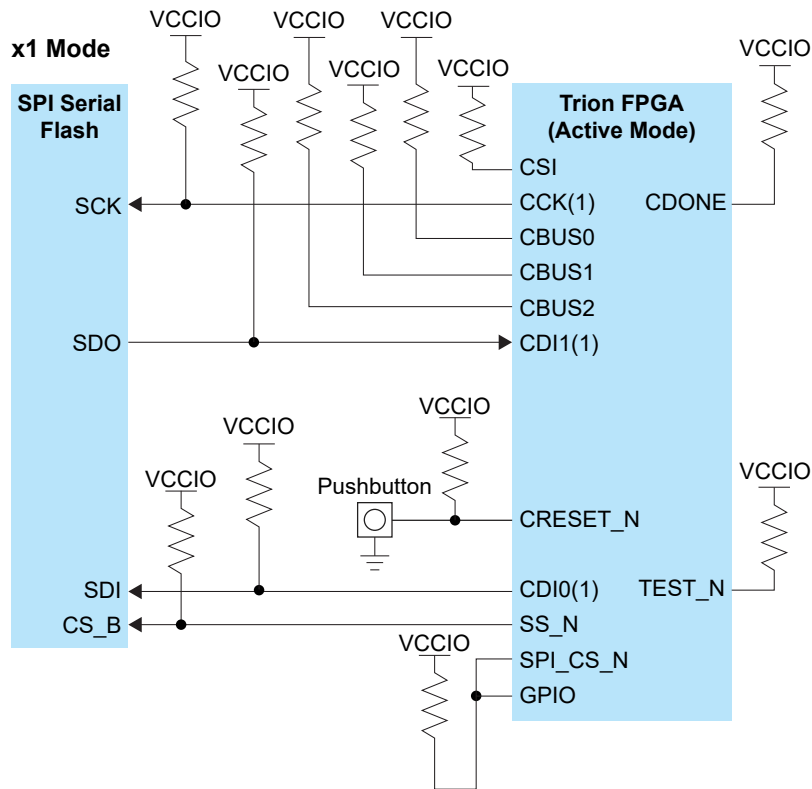
Connecting:

- `SS_N` to SPI Flash `CS_B`
- `GPIO` to `SPI_CS_N`

to allow:

- configuration with external SPI flash
- user data storage in internal SPI flash.

Figure 9: Configuration with External Flash



**Note:** (1) External pull-up is optional unless required by an external load.

Table 9: Additional Connection Requirements for SIP Packages

Configuration Setup	SPI_CS_N Pin	External Flash Chip Select Pin
Configure with internal flash only	Connect to Trion® SS_N pin	Not applicable
Configure with internal flash	Connect to Trion® SS_N pin	Connect any Trion® GPIO pin
Configure with external flash	Connect any Trion® GPIO pin	Connect to Trion® SS_N pin

## SPI Active Mode without CSI

Trion® FPGAs in smaller pin count packages, such as the WLCSP80 and BGA169, may not have the CSI signals bonded out. This pinout limits your programming options. Without CSI, you cannot use cascade configuration.

The schematics for programming without CSI are the same as the regular SPI active schematics except that you do not connect the CSI signal.



**Important:** Daisy-chain configuration is not supported in packages without CSI.

## Clocking

An internal oscillator generates the internal clocks the FPGA uses during configuration. In SPI active configuration mode, configuration starts operating at the default frequency (10 MHz) and then switches to the user-selected clock to minimize configuration time (assuming the SPI flash device supports the faster  $f_{MAX}$ ).

You set the configuration clock frequency in the Efinity® software.

**Table 10: Internal Oscillator Clock Settings**

SPI Clock Divider	Frequency (MHz)
DIV4	20
DIV8	10

## SPI Passive Mode

In passive mode, the FPGA receives the configuration clock and data from an external active module, such as an external microprocessor or microcontroller. This mode supports a data width of up to 32 bits.



**Learn more:** Refer to the Trion® device data sheet for the widths your device supports.

Design considerations are similar to active configuration except CCK must be driven from an external clock source. Each configuration image contains a synchronization pattern. When the Trion® FPGA detect the synchronization pattern, it begins configuration. The external active device must supply data continuously on every clock until configuration ends.



**Note:** Efinix recommends that you use the same VCCIO on the banks of all configuration pins.

### Connection Examples

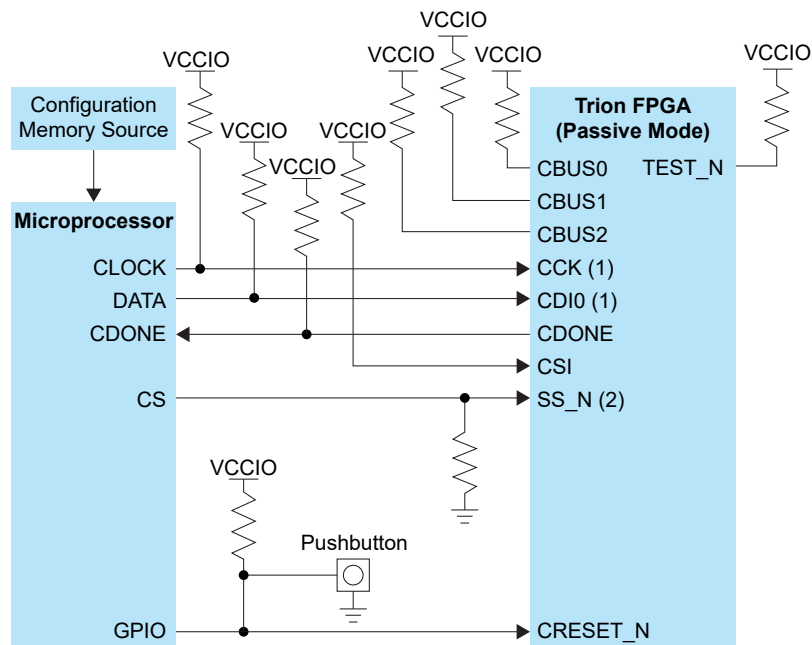
These examples show SPI passive x1 and x32 modes. .



**Learn more:** Refer to [Trion Hardware Design Checklist and Guidelines](#) for the detailed connection requirements.

**Figure 10: Passive (x1)**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



**Notes:**

1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.



**Note:** The microprocessor might disable either CS or CLOCK due to firmware latency.

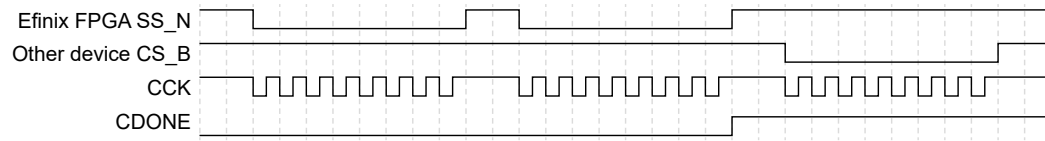


**Important:** You must complete the bitstream transmission before enabling the CS of other devices sharing the same SPI bus.

You cannot disable the configuration by driving either SS\_N to high or CS\_I to low. You have to complete the bit stream transmission before communicating with other devices on the same SPI bus. The waveform as shown in **Figure 11: Supported SPI Waveform** on page 22 results in successful configuration. While the waveform as shown in **Figure 12: Unsupported SPI Waveform** on page 22 results in configuration failure.

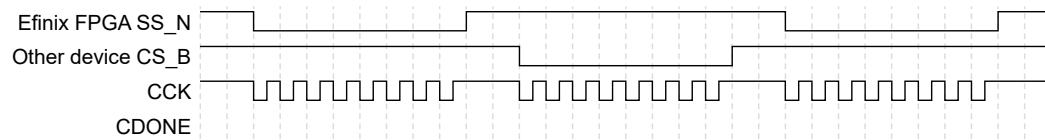
**Figure 11: Supported SPI Waveform**

Complete bitstream transmission before activating other devices on the same SPI bus.



**Figure 12: Unsupported SPI Waveform**

Interleaving the bitstream transmission and communication with other devices on the same SPI bus.

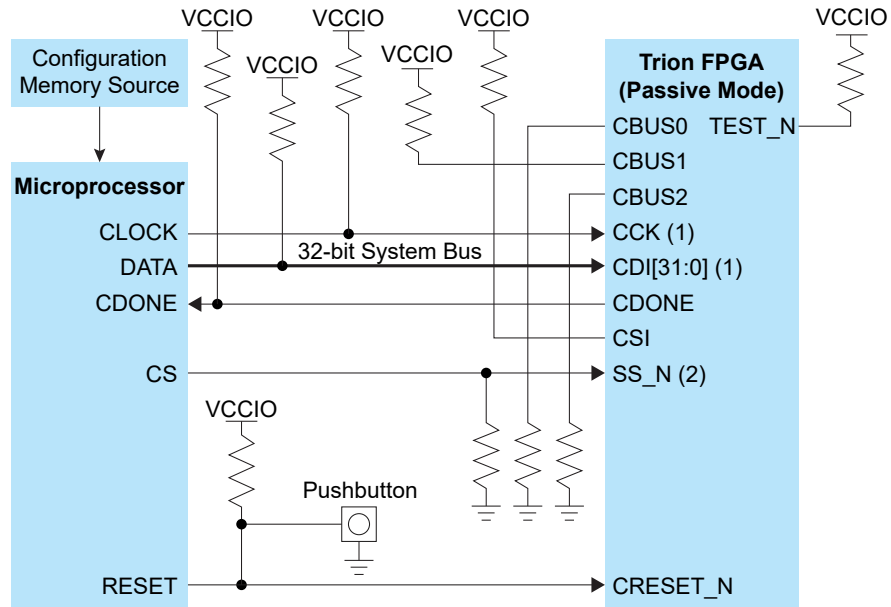


**Table 11: Bitstream Bits in Series**

CBUS	Byte Order	Bit Order
Cycle 1	Byte 0	Bit 7 (MSB)
Cycle 2		Bit 6
Cycle 3		Bit 5
Cycle 4		Bit 4
Cycle 5		Bit 3
Cycle 6		Bit 2
Cycle 7		Bit 1
Cycle 8		Bit 0 (LSB)
Cycle 9	Byte 1	Bit 7 (MSB)
Cycle 10		Bit 7
Cycle 11		Bit 5
Cycle 12		Bit 4
Cycle 13		Bit 3
Cycle 14		Bit 2
Cycle 15		Bit 1
Cycle 16		Bit 0 (LSB)

Figure 13: Passive (x32)

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



**Notes:**

1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.

Table 12: Bitstream Bytes Packed into 32 bit Parallel Bus

CBUS	31	24	23	16	15	8	7	0
Cycle 1	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 0		Byte 1		Byte 2		Byte 3	
Cycle 2	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 4		Byte 5		Byte 6		Byte 7	

Table 13: Bitstream Bytes Packed into 16 bit Parallel Bus

CBUS	15	8	7	0
Cycle 1	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 0		Byte 1	
Cycle 2	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 2		Byte 3	
Cycle 3	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 4		Byte 5	
Cycle 4	Bit 7 (MSB)	Bit 0 (LSB)	Bit 7 (MSB)	Bit 0 (LSB)
	Byte 6		Byte 7	

**Table 14: Bitstream Bytes Packed into 8 bit Parallel Bus**

<b>CBUS</b>	<b>7</b>		<b>0</b>
Cycle 1	Bit 7 (MSB)		Bit 0 (LSB)
	Byte 0		
Cycle 2	Bit 7 (MSB)		Bit 0 (LSB)
	Byte 1		
Cycle 3	Bit 7 (MSB)		Bit 0 (LSB)
	Byte 2		
Cycle 4	Bit 7 (MSB)		Bit 0 (LSB)
	Byte 3		

**Table 15: Bitstream Bits Packed into 4 bit Parallel Bus**

<b>CBUS</b>	<b>Byte Order</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
Cycle 1	Byte 0	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4
Cycle 2		Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
Cycle 3	Byte 1	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4
Cycle 4		Bit 3	Bit 2	Bit 1	Bit 0 (LSB)

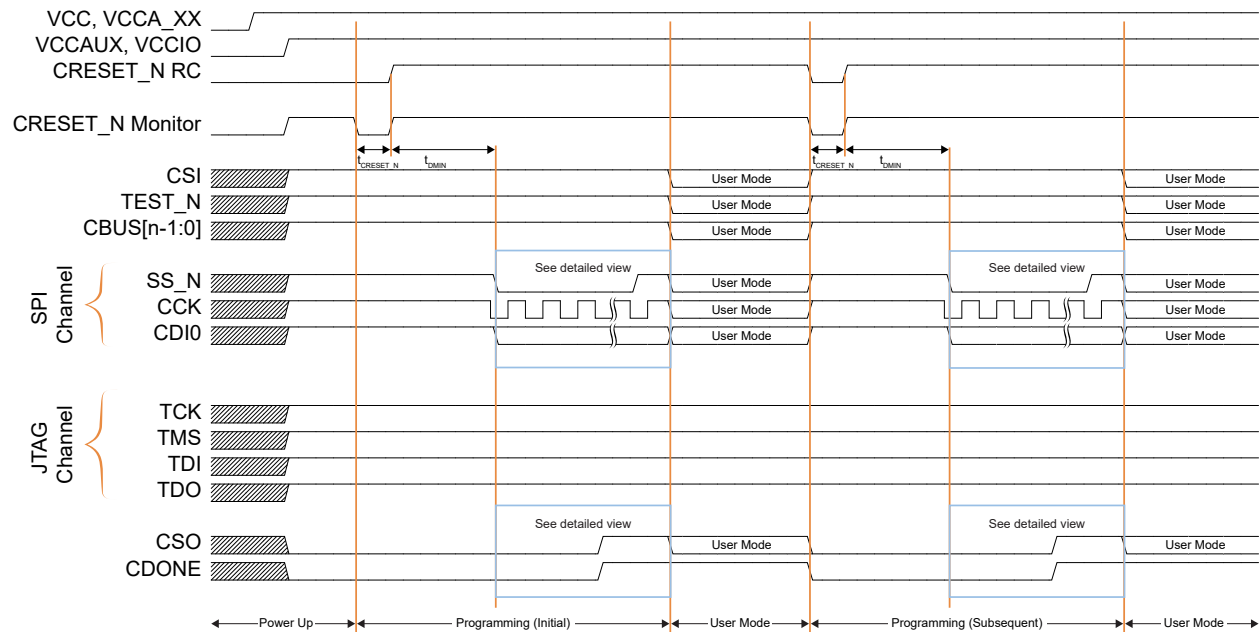
**Table 16: Bitstream Bits Packed into 2 bit Parallel Bus**

<b>CBUS</b>	<b>Byte Order</b>	<b>1</b>	<b>0</b>
Cycle 1	Byte 0	Bit 7 (MSB)	Bit 6
Cycle 2		Bit 5	Bit 4
Cycle 3		Bit 3	Bit 2
Cycle 4		Bit 1	Bit 0 (LSB)
Cycle 5	Byte 1	Bit 7 (MSB)	Bit 6
Cycle 6		Bit 5	Bit 4
Cycle 7		Bit 3	Bit 2
Cycle 8		Bit 1	Bit 0 (LSB)

## Timing

The microprocessor or microcontroller supplies the configuration clock and controls the reset signal. The microprocessor or microcontroller must hold CRESET<sub>N</sub> low for a duration of  $t_{CRESET\_N}$  and then release it to start the SPI passive configuration. After  $t_{DMIN}$ , the Trion<sup>®</sup> FPGA samples the synchronization pattern and begins configuration.

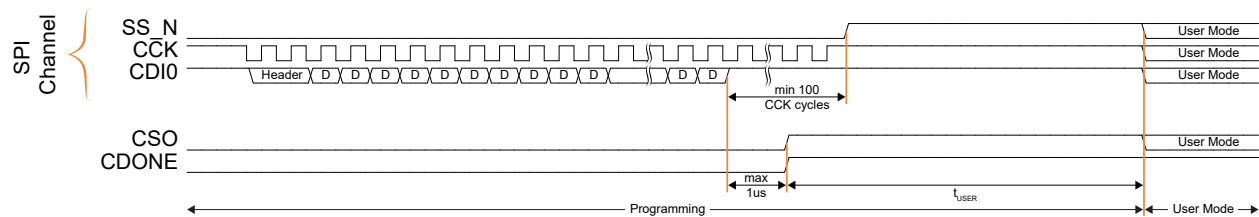
Figure 14: SPI Passive (x1, Mode 3) Configuration



**NOTES:**

1. CRESET\_N RC refers to the CRESET\_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
2. CRESET Monitor refers to the CRESET\_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 15: SPI Passive (x1, Mode 3) Configuration (Detailed View)



**LEGEND:**

D: Data



**Note:** Refer to "Power-up Sequence" in the Trion<sup>®</sup> data sheet for power-up details.



**Important:** You can only use a single configuration channel (SPI or JTAG) at the same time. Using both at the same time can result in configuration failure. Therefore, JTAG pins must be inactive during SPI passive configuration.

- Refer to **Figure 11: Supported SPI Waveform** on page 22 for the supported waveform.
- The waveform shows the perspective from the control block without any optional external pull-up or pull-down resistors connected.
- CDI input data is clocked by CCK. To prevent configuration failure, CCK must stop toggling if the bitstream data becomes invalid. You must resume with the next bitstream data before stopping to continue the configuration.
- CS<sub>I</sub> must stay high during configuration.
- It is recommended that SS<sub>N</sub> remain low during configuration. Sometimes, however, an SPI Master might release the SS<sub>N</sub> to high during firmware latency. Refer to **Figure 11: Supported SPI Waveform** on page 22 for the supported waveform.
- Efinix does not recommend connecting multiple slaves on the same SPI bus to prevent signal contention.
- Refer to the data sheet for timing specifications.



**Important:** To ensure successful configuration, the microprocessor must continue to supply the configuration clock to the Trion<sup>®</sup> FPGA for at least 100 cycles after sending the last configuration data.

---



**Learn more:** Refer to the Trion<sup>®</sup> FPGA data sheet for timing specifications.

Refer to the **AN 035: SPI Passive Programming with Raspberry Pi** for a SPI passive programming example design.

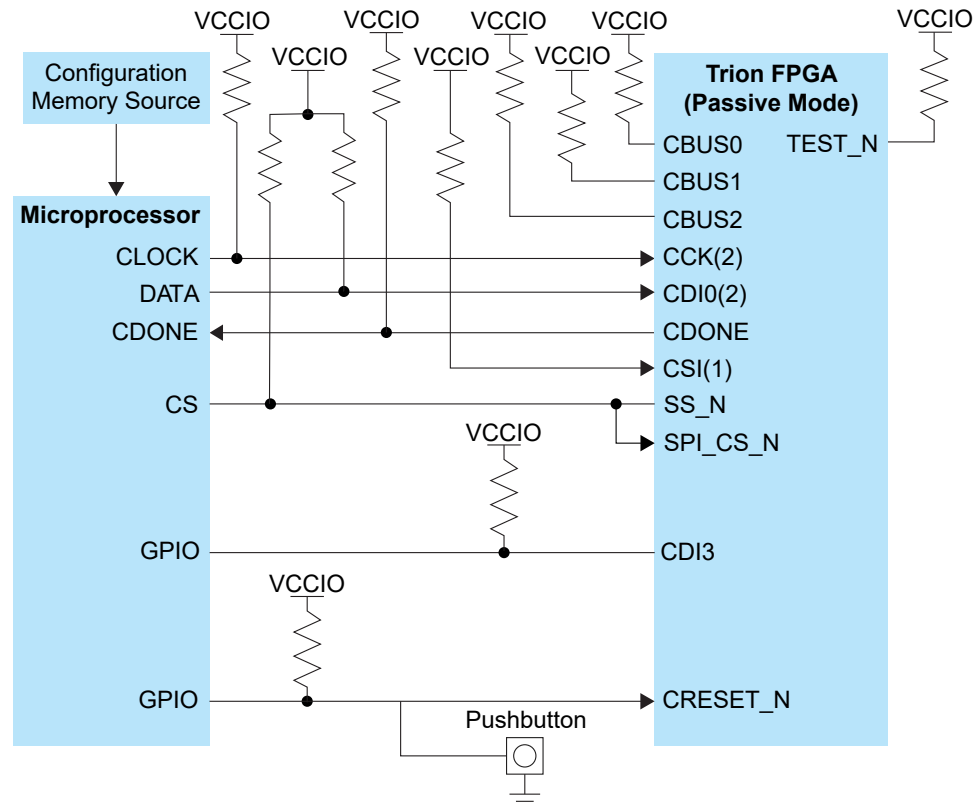
---

## SPI Passive Mode for SIP Packages

You can start SPI Passive configuration in QFP100F3 packages by pulling `CDI3` low to prevent unexpected programming of the internal SPI Flash during SPI Passive configuration.

**Figure 16: Configuration with Microprocessor in SPI Passive with Direct Access to Internal Flash**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



### Notes:

1. Connect the CSI of a downstream device to the CSO of an upstream device only in daisy chain configuration.
2. The external pull-up is optional unless required by an external load.



**Important:** You are strongly advised against sharing the same SPI bus with other devices if you also intend to use the internal SPI flash for user data storage. Sharing the same SPI bus in this manner can result in signal contention due to the microprocessor and FPGA accessing the bus simultaneously.

You need to drive `CRESET_N` low and `CDI3` high with the microprocessor if you intend to update the SPI flash data through the microprocessor in user mode.



**Note:** Refer to [Using FPGA, MCU, and SPI Flash Devices Together](#) on page 35 for more information.

## SPI Passive Mode without CSI or CBUS2

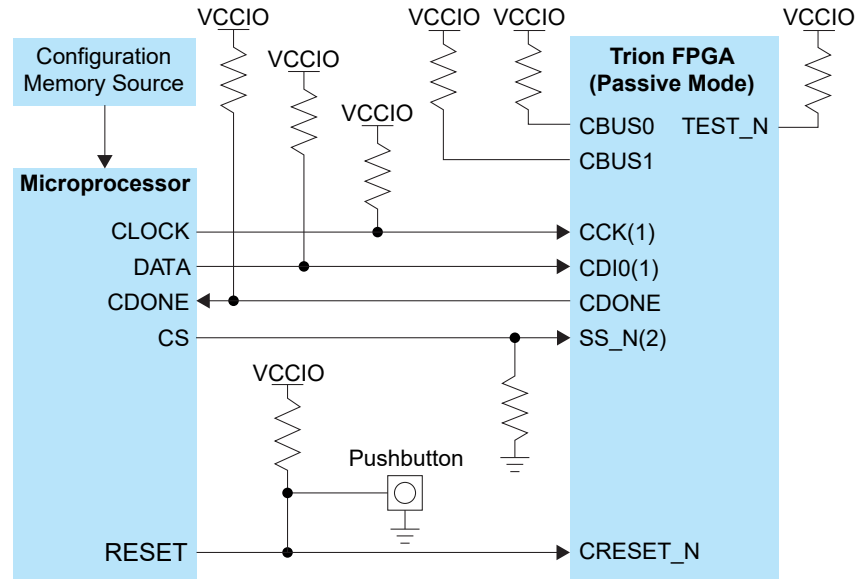
Trion® FPGAs in smaller pin count packages, such as the WLCSP80 and BGA169, may not have the CSI or CBUS2 signals bonded out. This pinout limits your programming options.

- Without CSI, you cannot use cascade configuration.
- Without CBUS2 you can still use CBUS0 and CBUS1 to program using the passive serial x1, x2, and x4 modes.

The following figures show the schematics for programming without CSI or CBUS2.

**Figure 17: Passive (x1) without CSI or CBUS2**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.

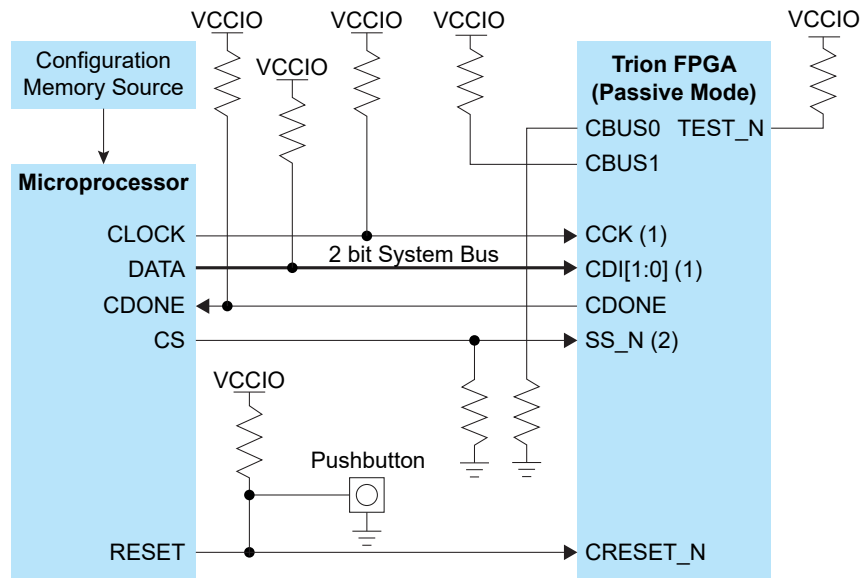


**Notes:**

1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.

Figure 18: Passive (x2) without CSI or CBUS2

See **Resistors in Configuration Circuitry** on page 51 for the resistor values.



**Notes:**

1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.

## JTAG Mode

The JTAG serial configuration mode is popular for prototyping and board testing. The four-pin JTAG boundary-scan interface is commonly available on board testers and debugging hardware.

This section describes the JTAG configuration mode, for JTAG boundary-scan testing, refer to [AN 021: Performing Boundary-Scan Testing on Trion FPGAs](#).



**Learn more:** Refer to the following web sites for more information about the JTAG interface:

<http://ieeexplore.ieee.org/document/6515989/>

<https://en.wikipedia.org/wiki/JTAG>

*Table 17: Supported Trion JTAG Instructions*

Instruction	Binary Code [3:0]	Description
SAMPLE/PRELOAD	0010	Enables the boundary-scan SAMPLE/PRELOAD operation
EXTEST	0000	Enables the boundary-scan EXTEST operation
BYPASS	1111	Enables BYPASS
IDCODE	0011	Enables shifting out the IDCODE
PROGRAM	0100	JTAG configuration
ENTERUSER	0111	Changes the FPGA into user mode.
JTAG_USER1	1000	Connects the JTAG User TAP 1.
JTAG_USER2	1001	Connects the JTAG User TAP 2.
JTAG_USER3	1010	Connects the JTAG User TAP 3.
JTAG_USER4	1011	Connects the JTAG User TAP 4.

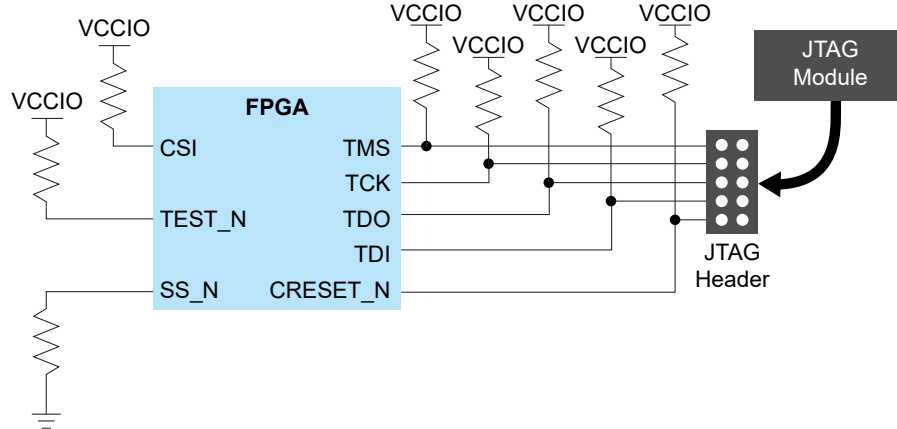


**Learn more:** Refer to [AN 038: Programming with an MCU and the JTAG Interface](#) for more information about programming Efinix® FPGAs with a microcontroller using JTAG mode.

Connect the FPGA pins as shown in the following diagrams.

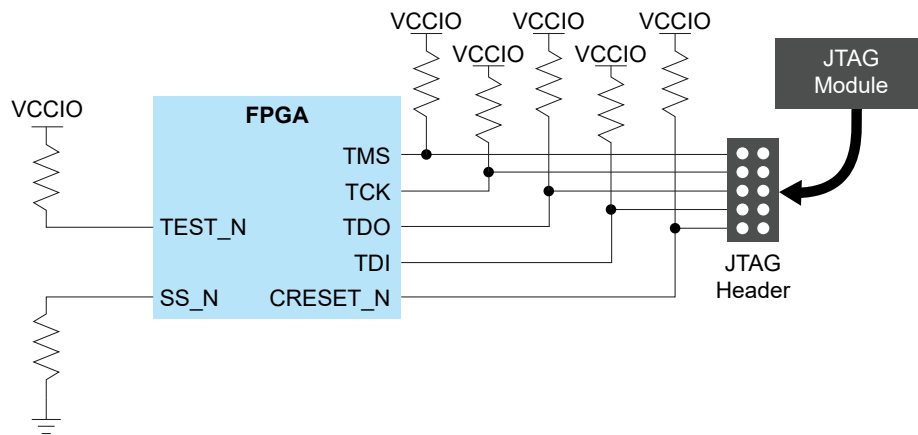
**Figure 19: JTAG Programming for T4, T8, T13QFP100F3, T13BGA256, T20QFP100F3, T20QFP144, and T20BGA256 FPGAs**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



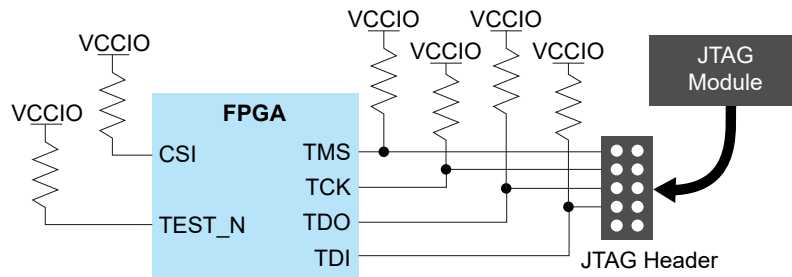
**Figure 20: JTAG Programming for T13BGA169, T20WLCSP80, and T20BGA169 FPGAs**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



**Figure 21: JTAG Programming for T20BGA324, T20BGA400, T35, T55, T85, and T120 FPGAs**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.

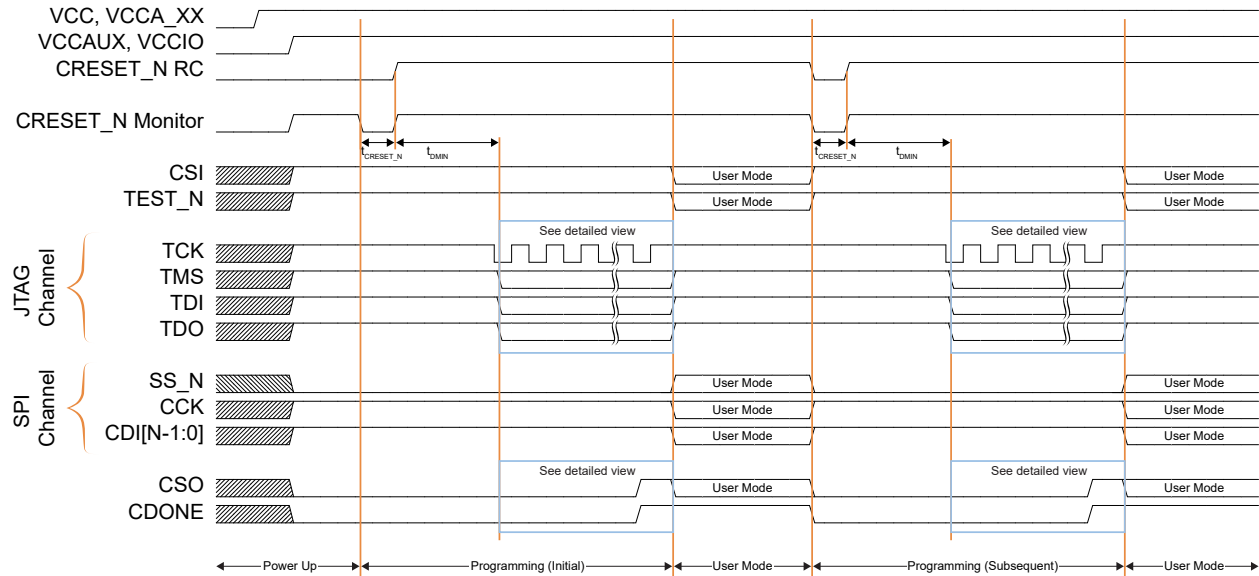


The CRESET\_N signal needs to be deasserted before JTAG configuration begins. Only for T4, T8, T13, T20WLCSP80, T20QFP100F3, T20QFP144, T20BGA256, and T20BGA169 FPGAs, drive CRESET\_N low, and then high prior to JTAG configuration. When configuration ends, the JTAG host issues the ENTERUSER instruction to the FPGA. After CDONE goes high and the FPGA receives the ENTERUSER instruction, the FPGA waits for t<sub>USER</sub> to elapse, and then it goes into user mode.

**Note:** The FPGA may go into user mode before  $t_{USER}$  has elapsed. Therefore, you should keep the system interface with the FPGA in reset until  $t_{USER}$  has elapsed.

**Important:** JTAG configuration is not supported in the F49 package.

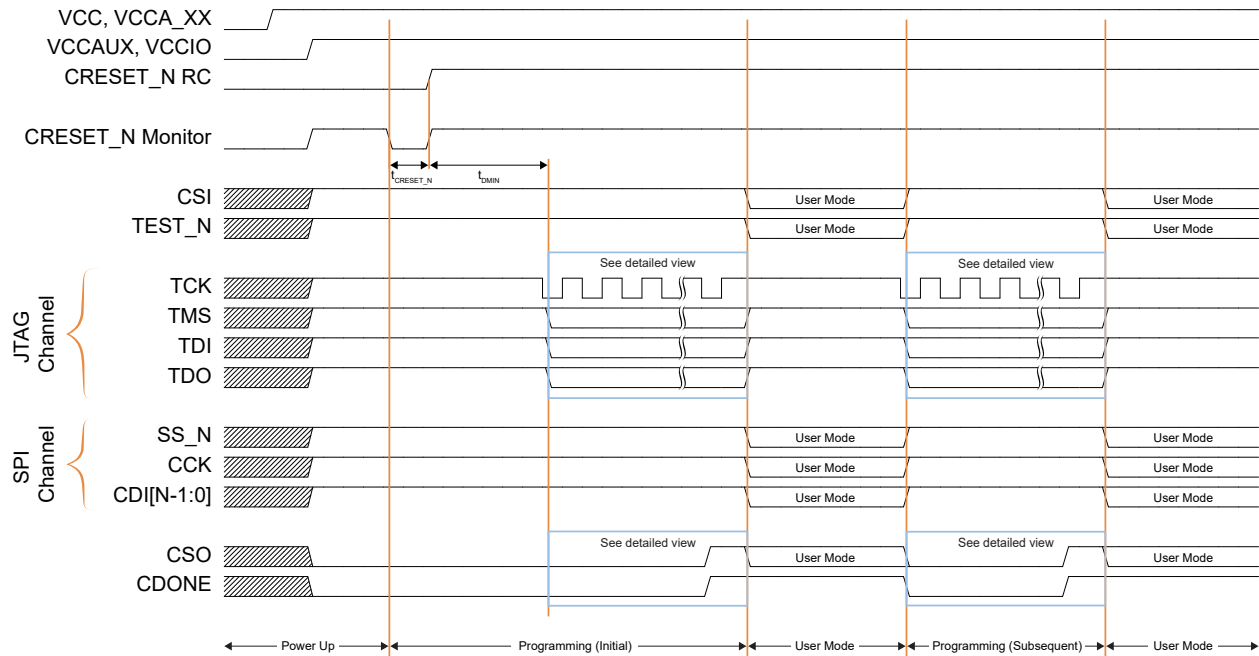
Figure 22: JTAG Programming Waveform (T4, T8, T13, T20W80, T20Q100F3, T20Q144, T20F256, and T20F169 FPGAs)



**NOTES:**

1. CRESET\_N RC refers to the CRESET\_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
2. CRESET Monitor refers to the CRESET\_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

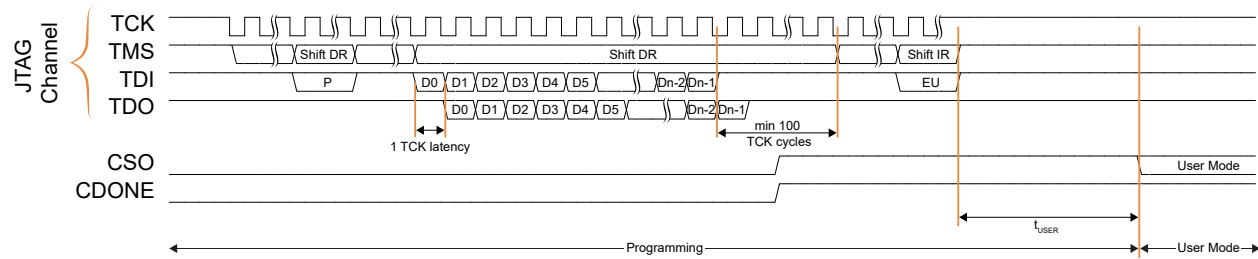
Figure 23: JTAG Programming Waveform (T20F324, T20F400, T35, T55, T85, and T120 FPGAs)



**NOTES:**

1. CRESET\_N RC refers to the CRESET\_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
2. CRESET Monitor refers to the CRESET\_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 24: JTAG Programming (All Trion FPGAs) (Detailed View)

**Legend:**

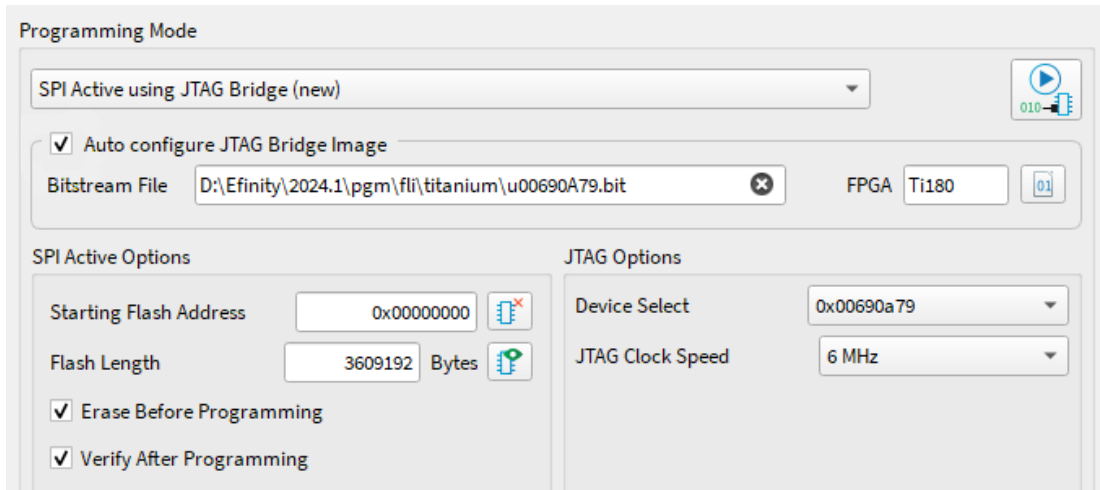
P: PROGRAM EU: ENTERUSER

**Note:** Refer to "Power-up Sequence" in the Trion® data sheet for power-up details.**Note:** Refer to the data sheet for timing specifications.**Important:** You can only use a single configuration channel (SPI or JTAG) at the same time. Using both at the same time can result in configuration failure. Therefore, the SPI bus must be inactive during JTAG configuration.**Note:** The waveform is shown from the control block perspective and it is required to connect to weak internal pull-up resistors.

## Design Considerations

- Because the TCK and TMS signals connect devices in the JTAG chain, they must have good signal quality.
- TCK should transition monotonically at the receiving devices and should be terminated correctly. Poor TCK quality can limit the maximum frequency you can use for configuration.
- Buffer TMS and TCK so they have sufficient drive strength at all receiving devices.
- Ensure that the logic high voltage is compatible with all devices in the JTAG chain.
- If your chain contains devices from different vendors, you might need to drive optional JTAG signals, such as TRST and enables.
- For Trion T4, T8, T13, T20 (WLCSP80, QFP100F3, QFP144, BGA256, and BGA169 packages) FPGAs:
  - Drive CRESET\_N low and then high prior to JTAG configuration.
  - When using the Efinity programmer to perform JTAG configuration, the CRESET\_N and SS\_N pins are used in addition to the standard JTAG pins. If the JTAG-SPI bridge image has already been configured, neither CRESET\_N or SS\_N are required. However, you will need to establish both CRESET\_N and SS\_N connections if using "Auto configure JTAG Bridge Image" in SPI flash programming through the JTAG bridge (see figure below).

Figure 25: CREST\_N and SS\_N are Required if Using Auto Configure JTAG Bridge Image



**Learn more:** Refer to [JTAG Programming Connections](#) on page 56 for JTAG configuration connection examples and [JTAG Programming](#) on page 67 for details about JTAG programming using the Efinity Programmer.



**Learn more:** Refer to the Virtual I/O Debug Core section in the [Efinity Software User Guide](#) for more information about JTAG User TAP interface.

# Using FPGA, MCU, and SPI Flash Devices Together

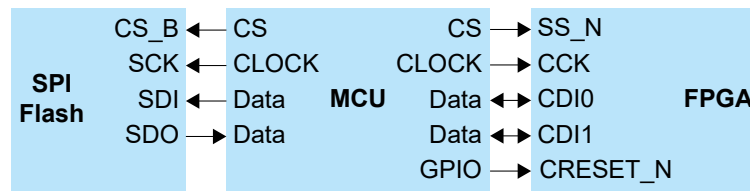
Most applications include some combination of FPGA, MCU, and SPI flash in the system design. How the FPGA, MCU, and SPI flash connected affect what configuration is supported and the configuration sequence.

## MCU with Separate SPI Bus Connections

In a system where the MCU relies on separate SPI buses connected to the SPI flash and FPGA, respectively:

- FPGA images can be stored in the SPI flash so that the MCU can configure the FPGA through SPI passive mode.
- MCU can transmit the user data stored in SPI flash to the FPGA in user mode.

Figure 26: MCU with Separate SPI Bus Connections to SPI Flash and FPGA

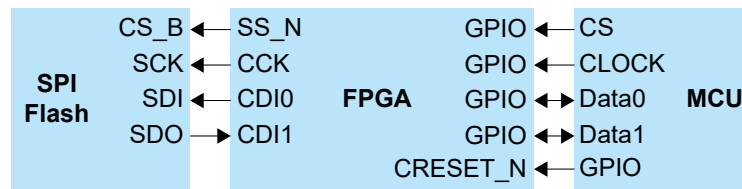


## FPGA with Separate SPI Bus Connections

In a system where the FPGA relies on separate SPI buses connected to the SPI flash and MCU, respectively, note the following guidance:

- SS\_N, CCK, CDI0, and CDI1 are connected to the SPI flash with GPIO pins connecting to the MCU:
  - SPI active applies.
  - SPI flash can be used for both FPGA images and user data storage.

Figure 27: FPGA Configured with SPI Bus Connections to SPI Flash and GPIO Emulated MCU SPI Bus Connections to MCU



- SS\_N, CCK, and CDI0 are connected to MCU with GPIOs connecting to SPI flash:
  - Only SPI passive through MCU applies.
  - SPI flash can ONLY be used for data storage.

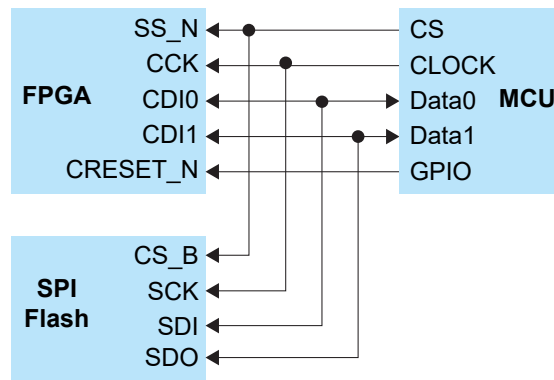
**Figure 28: FPGA Configured with SPI Bus Connections to MCU and GPIO Emulated MCU SPI Bus Connections to SPI Flash**



## Shared SPI Bus Connections

- Per the SPI protocol, the SPI master sets the CS\_B/SS\_N pin to a tri-state with external pull up to disable the SPI bus. In this configuration, the FPGA shares the SPI bus between the SPI flash and the MCU; therefore, the system defaults to SPI active configuration after power up.
- Whether the system is in SPI active or SPI passive mode depends upon the state of the SS\_N pin when CRESET\_N is driven low.
- If there are no valid FPGA images stored in the SPI flash (e.g., first power up of a fresh system, SPI flash for user data storage only, etc.), the FPGA keeps driving the SPI bus. Therefore, to ensure SPI bus access from the MCU, either to program the SPI flash or to enter SPI passive configuration, the MCU should:
  - Drive SS\_N to low
  - Drive CRESET\_N to low.

**Figure 29: Shared SPI Bus Connections**



To program the SPI flash or to enter SPI passive configuration, the off-chip MCU should:

- Drive SS\_N to low
- Drive CRESET\_N to low.



**Note:** You can treat the T20Q100F3 SIP with SPI bus connected to the MCU as a system in which the SPI bus is shared between the FPGA, SPI flash, and MCU.



**Note:** Refer to [Support for Multiple Images](#) on page 46 for valid bitstream searching behavior.

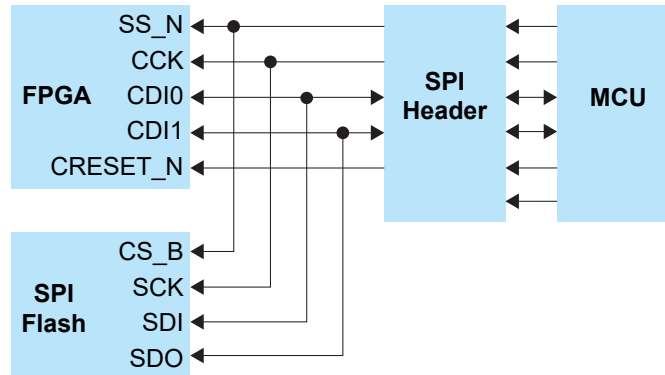


**Note:** You can confirm the status of the FPGA by connecting the MCU to CDONE and NSTATUS. This connection helps you check whether the SPI bus can reliably determine when it needs to drive CRESET\_N to prevent unexpected system interrupts. Refer to [Verifying Configuration](#) on page 77 for more information.



**Note:** If you have an off-chip MCU, the configuration advice is the same as [Figure 31: Flash Programming Board Setup](#) on page 39 with an SPI header.

*Figure 30: Shared SPI Bus Connections with Off-Chip MCU through SPI Header*



**Learn more:** Refer to [Flash Programming Modes](#) on page 38 for further advice.

# Flash Programming Modes

The following table shows the methods you can use to program the configuration bitstream into the flash device on your board. Although you can program the flash directly using the SPI interface, this method requires that you have a SPI header on your board or use an FDTI chip. Therefore, Efinix recommends that you use a JTAG bridge, because that method only requires a JTAG header, which you would typically have on your board for other purposes.

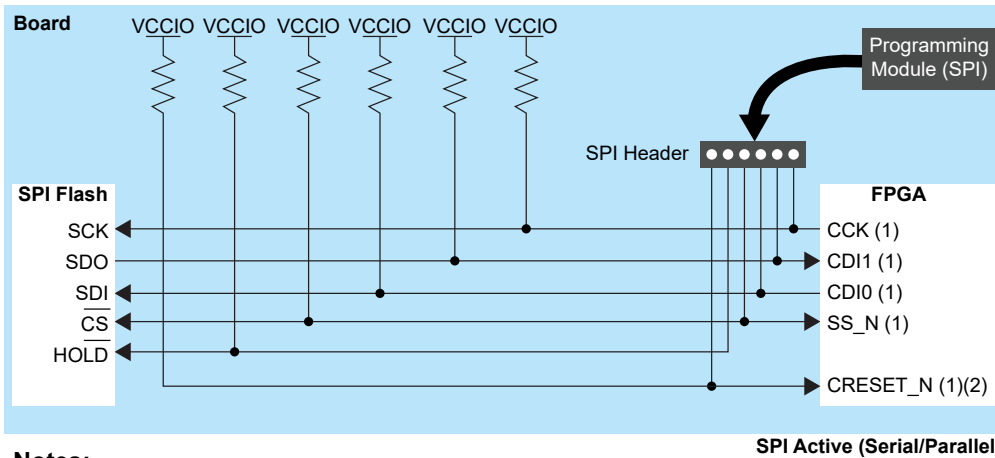
*Table 18: Flash Programming Modes*

Mode	Description
SPI Active (serial/parallel)	Use the Efinity Programmer and a cable connected to a SPI header on the board.
SPI Active using JTAG Bridge (New)	A improved version of the SPI Active using JTAG Bridge (Legacy) mode with a faster flash programming time.



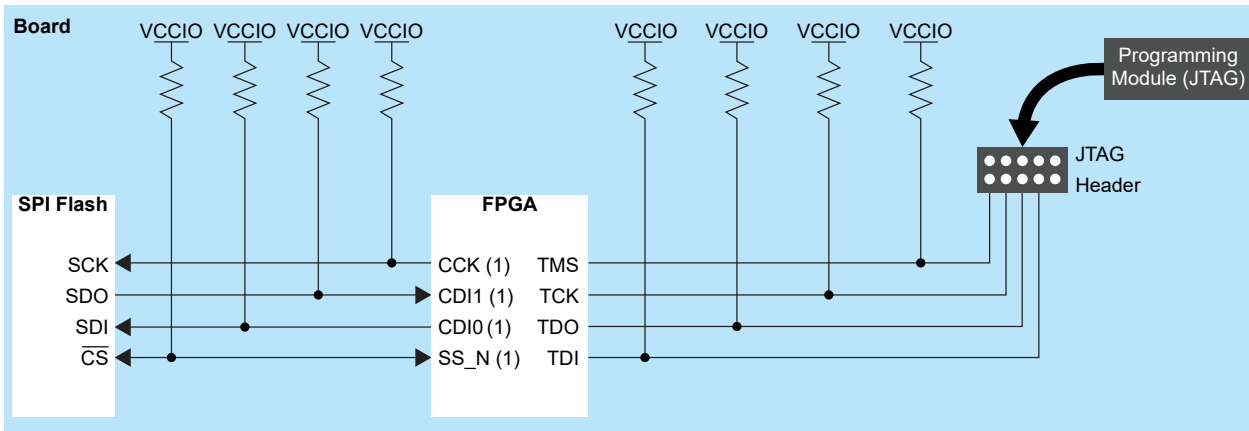
**Learn more:** Refer to [Program using a JTAG Bridge](#) on page 69 for more information.

Figure 31: Flash Programming Board Setup



**Notes:**

- 1) The external pull-up is optional unless required by an external load.
- 2) Be sure to hold CRESET\_N low to prevent signal contention during SPI flash programming.



**Note:**

- 1) The external pull-up is optional unless required by an external load.



**Important:** Depending upon your vendor, your flash device may have been programmed in the factory. If this is the case, your vendor may have set the Status Register Protect (SRP) bits to protect the software and/or hardware, to prevent power supply lock-down, or to otherwise make the device one-time programmable. To overcome this issue, you will need to check your device flash specifications and status to ensure that it is quad enabled before attempting to re-program the device. The following table outlines the various states of the SRP bits.

**Table 19: Status Register Protect Bits**

SRP1	SRP2	/WP	Status Register	Description
0	0	X	Software Protection	/WP pin has no control. Following a Write Enable instruction, the status register is writable, WEL=1.
0	1	0	Hardware Protected	When /WP is low, the status register is locked and is unwritable.
0	1	1	Hardware Unprotected	When /WP is high, the status register is unlocked. A Write Enable instruction will make the status register writable, WEL=1.
1	0	X	Power Supply Lock-Down	The status register is locked and cannot be written to again until the next power-up cycle.
1	1	X	One Time Program	The status register is permanently protected.

# Power Up

## Power Up Sequence

Efinix® recommends the following power up sequence when powering Trion® FPGAs:

Figure 32: Trion® FPGAs without MIPI Power Up Sequence

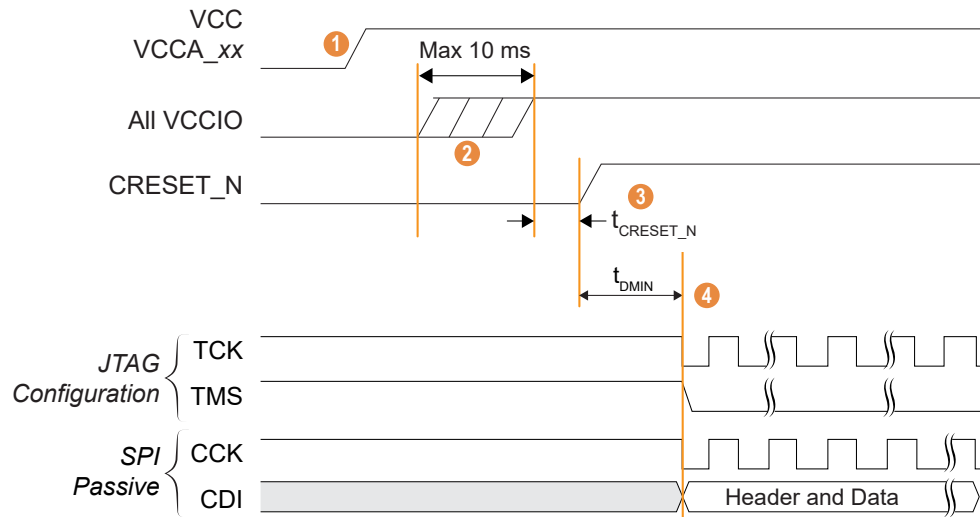
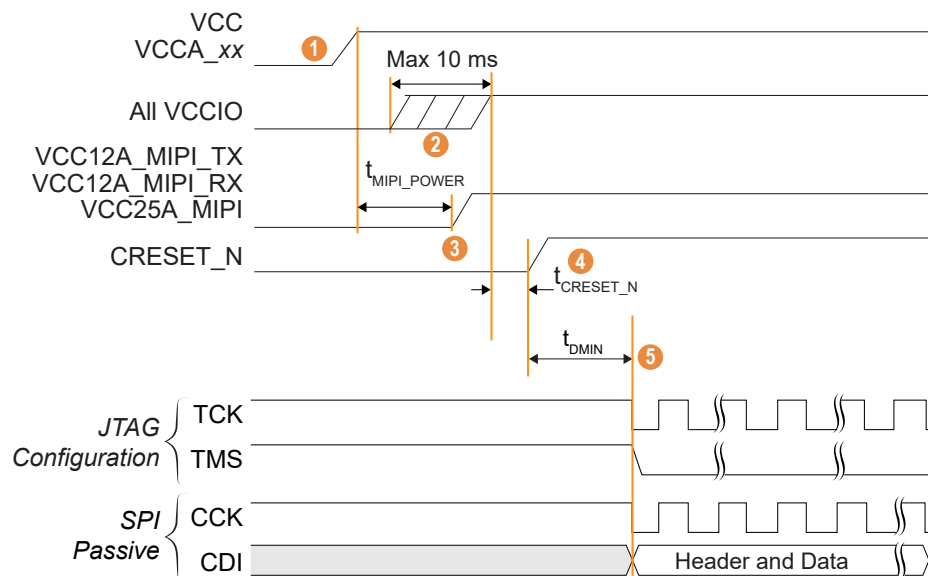


Figure 33: Trion® FPGAs with MIPI Power Up Sequence



1. Power up VCC and VCCA\_xx first.
2. When VCC and VCCA\_xx are stable, power up all VCCIO pins. There is no specific timing delay between the VCCIO pins.
3. For FPGAs with MIPI: Apply power to VCC12A\_MIPI\_TX, VCC12A\_MIPI\_RX, and VCC25A\_MIPI at least  $t_{MIPI\_POWER}$  after VCC is stable.



**Important:** Ensure the power ramp rate is within VCCIO/10 V/ms to 10 V/ms.

4. After all power supplies are stable, hold CRESET\_N low for a duration of  $t_{\text{CRESET\_N}}$  before asserting CRESET\_N from low to high to trigger active SPI programming (the FPGA loads the configuration data from an external flash device).
5. FPGA configuration can begin after there has been a  $t_{\text{DMIN}}$  minimum delay after CRESET\_N goes high (see **SPI Passive** and **JTAG** for the delay specification).

When you are not using the GPIO, MIPI, DDR or PLL resources, connect the pins as shown in the following table.



**Note:** Refer to Configuration Timing and MIPI Power Up Timing sections in the Trion® FPGA data sheets for timing information.

## Power Supply Current Transient

You may observe an inrush current on the dedicated power rail during power-up. You must ensure that the power supplies selected in your board meets the current requirement during power-up and the estimated current during user mode. Use the Power Estimator to calculate the estimated current during user mode.

*Table 20: Maximum Power Supply Current Transient for VCC*

FPGA	Package	Maximum Power Supply Current Transient <sup>(4)(5)</sup> (mA)
T4	All	18
T8	BGA49, BGA81	18
	QFP144	35
T13	All	35
T20	WLCSP80, QFP100F3, QFP144, BGA169, BGA256	35
	BGA324, BGA400	57
T35	All	57
T55	All	200
T85	All	200
T120	All	200

<sup>(4)</sup> Inrush current for other power rails are not significant in Trion® FPGAs.

<sup>(5)</sup> Measured at room temperature.

## Power Up Configuration Circuitry Recommendation

You can use one of the following methods to hold the `CRESET_N` pin of the Trion<sup>®</sup> FPGA low after the power supplies are stable:

- Supervisor integrated circuit (IC)
- Microprocessor or microcontroller



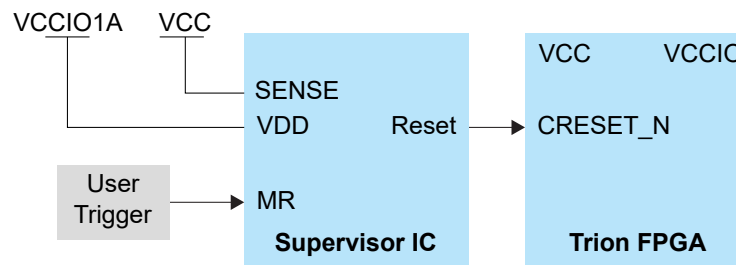
**Important:** Do not drive a signal to any Trion<sup>®</sup> I/O pins before the Trion<sup>®</sup> FPGA is powered up. Most FPGAs have electrostatic discharge (ESD) circuits to protect the devices from ESD events. Driving the I/O pins before `VCCIO` will result in an in-rush current driving the I/Os to a specific voltage through the ESD circuit to the `VCCIO` rail. Trion<sup>®</sup> FPGAs will remain in configuration mode after power-up if this unexpected voltage exists on the `CRESET_N` due to the improper power-up sequence.

### Supervisor IC Circuitry Example

Assuming that the `VCCIO1A` is the last power supply to be stable in the system, the supervisor IC must hold the `CRESET_N` pin low for a duration of  $t_{RP}$  (reset timeout period) after the `VCCIO1A` reaches the stable threshold.

Ensure that the  $t_{RP}$  of the selected supervisor IC is more than the required  $t_{CRESET\_N}$ . Refer to the supervisor IC vendor for the recommended operating circuitry.

*Figure 34: Supervisor IC Power Up Circuitry*

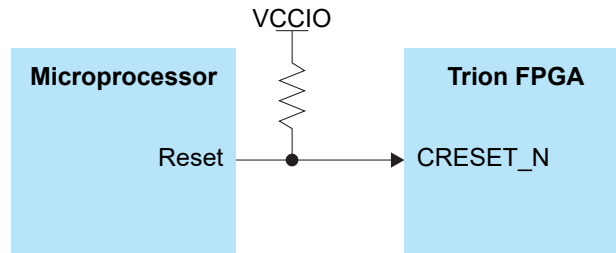


**Note:** The user trigger (pushbutton, FTDI module) must be connected to the `MR` pin of the supervisor IC.

## Microprocessor or Microcontroller Circuitry Example

**Figure 35: Microprocessor Power Up Circuitry**

See [Resistors in Configuration Circuitry](#) on page 51 for the resistor values.



The microprocessor or microcontroller must hold the CRESET\_N pin low more than the required  $t_{\text{CRESET\_N}}$  duration.

## Unused Resources and Features

**Table 21: Connection Requirements for Unused Resources**

Unused Resource	Pin	Note
GPIO Bank	VCCIOxx	Connect to either 1.8 V, 2.5 V, or 3.3 V.
PLL	VCCA_PLL	Connect to VCC.
MIPI	VCC12A_MIPI_TX	Connect to VCC (1.2 V).
	VCC12A_MIPI_RX	Connect to VCC (1.2 V).
	VCC25A_MIPI	Connect to VCC (1.2 V).
DDR	VCCIO_DDR	Floating. Leave unconnected.
	DDR_VREF	Connect to ground.

# Configuration Sequence

The Trion® FPGA configuration logic uses the following sequence during configuration:

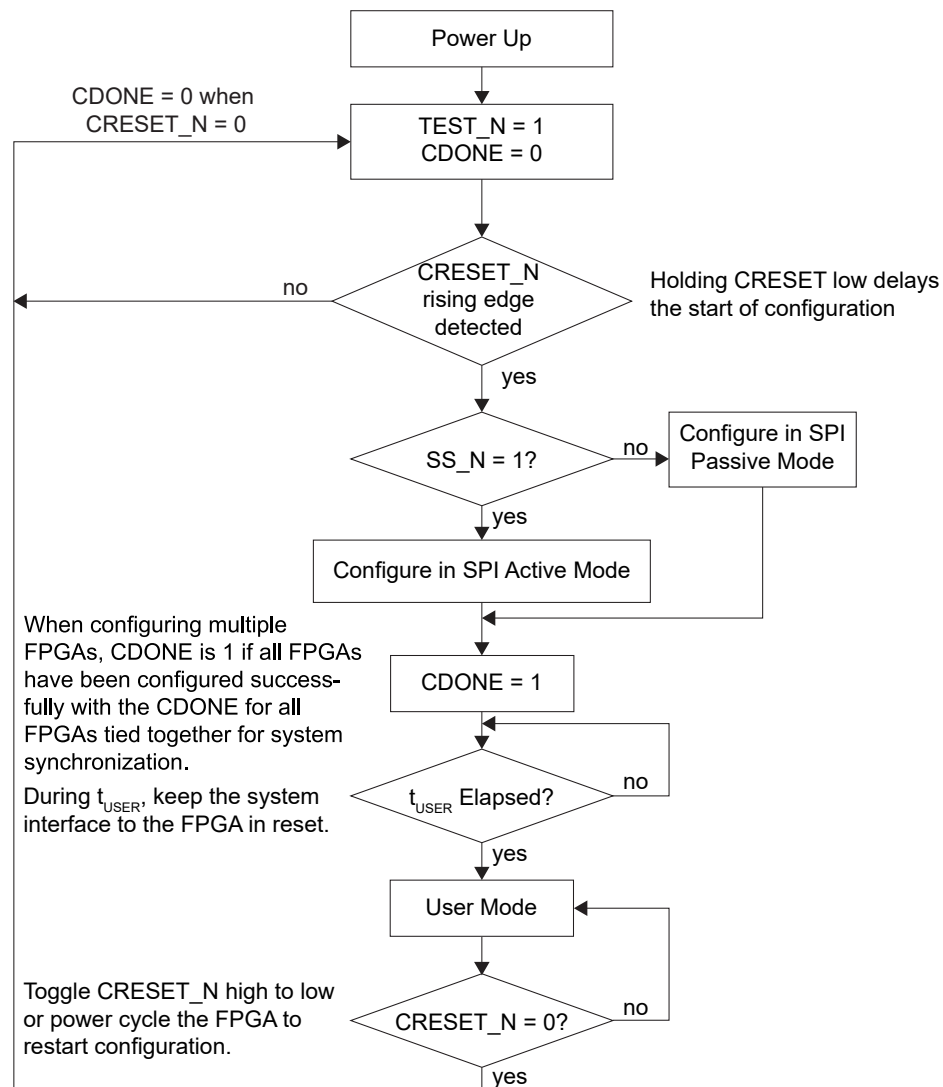
1. When CRESET\_N returns high (logic 1) after being held low (logic 0), the FPGA samples the logical value on its SS\_N pin. Like other programmable I/O pins, the SS\_N pin has an internal pull-up resistor.



**Learn more:** Refer to the Trion® data sheet for the pulse width requirements of CRESET\_N.

2. If the SS\_N pin is sampled as a logic 1 (high), the FPGA configures using the SPI active configuration interface.
3. If the SS\_N pin is sampled as a logic 0 (low), the FPGA waits to be configured from an external controller or from another FPGA in SPI active configuration mode using an SPI-like interface.

Figure 36: Configuration Flow Diagram



# Support for Multiple Images

When powered up in SPI active mode, the Trion® FPGA defaults to the first valid image it finds searching from address 0. If you enable the multi-image feature, you can optionally choose from three other images.



**Learn more:** To enable the multi-image feature, use the Efinity Programmer to combine multiple images into a single hex file. See [Program Multiple Images \(CBSEL\)](#) on page 64 for more information.

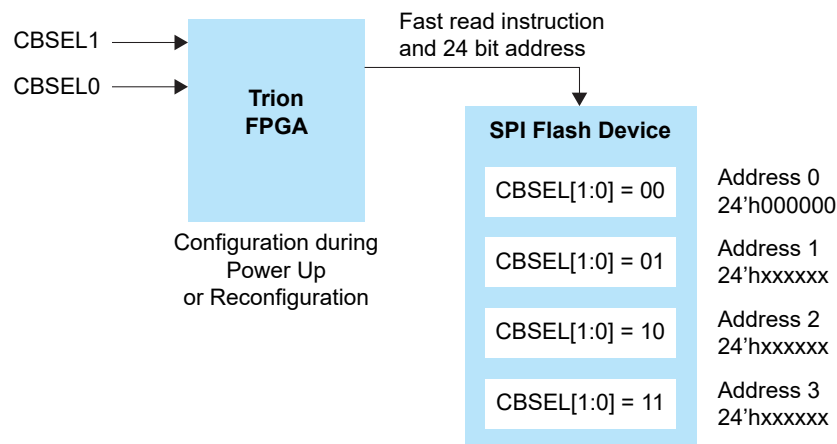
During multi-image configuration, the Trion® FPGA monitors the CBSEL [1:0] pin logic value when configuration or reconfiguration begins to determine which bitstream image to use. Then, it loads the corresponding image starting from the address specified in the bitstream option bits by sending out a fast read instruction followed by the address.

For multi-image configuration, the Efinity® software saves the images to the bitstream file with no configuration bits between images by default.



**Note:** Some Trion® FPGAs may not support multiple images for all configuration modes. The Supported Configuration Modes topic in your data sheet explains which modes the FPGA supports.

*Figure 37: Configuration Setup for Multiple Images*



**Note:** Efinix recommends you to store image 0 at the 24'h000000 address as Trion® FPGAs always start searching for a valid bit stream from the 24'h000000 address.

Connect CBSEL [1:0] for the image you want to use:

- 00 for image 1
- 01 for image 2
- 10 for image 3
- 11 for image 4

During configuration, the FPGA initially searches for a valid image starting at the memory location `0x0000_0000` in the SPI flash. It then proceeds to read the memory location based on the `CBSEL[1:0]` setting. If no valid image is found at that memory location, the FPGA continues to search in ascending order until it locates a valid image. For example, if `CBSEL[1:0]` is set to `11` and the SPI flash only contains valid images for `00` and `01`, the FPGA will load the image from `00`. The following table describes valid and invalid images.

Image Details	Note
Valid image	Configuration performs as expected.
Invalid image	FPGA The FPGA cannot recognize a valid image at the targeted SPI Flash address. It continues to search in ascending address and configure with the next valid image if any.
Corrupted image	Image is recognized, but the FPGA fails in configuration with <code>CDONE = 0</code> and <code>NSTATUS = 0</code> to indicate a device mismatch or CRC error (except T4, T8 F49, and F81).



**Learn more:** You can also use the internal reconfiguration feature to reconfigure the FPGA with a different image. This feature uses internal logic instead of the `CBSEL[1:0]` pins. Refer to [AN 010: Using the Internal Reconfiguration Feature to Update Efinix FPGAs Remotely](#) for details on this feature.

# Configuring Multiple FPGAs

If your application uses multiple Trion® FPGAs, you can configure all of them using a single configuration source.

- FPGAs that use the *same* configuration file can be loaded at the same time.
- FPGAs that use *different* configuration files (images) can be loaded sequentially, either through Trion® FPGAs in a daisy chain, or using external logic.

For daisy chain configurations, the Efinity® software includes 2,048 configuration bits between images in the bitstream file.



**Note:** You cannot utilize daisy-chain packages that do not have the CSI signal bonded out (such as the WLCSP80 and BGA169).

## Daisy Chaining with a SPI Flash Device

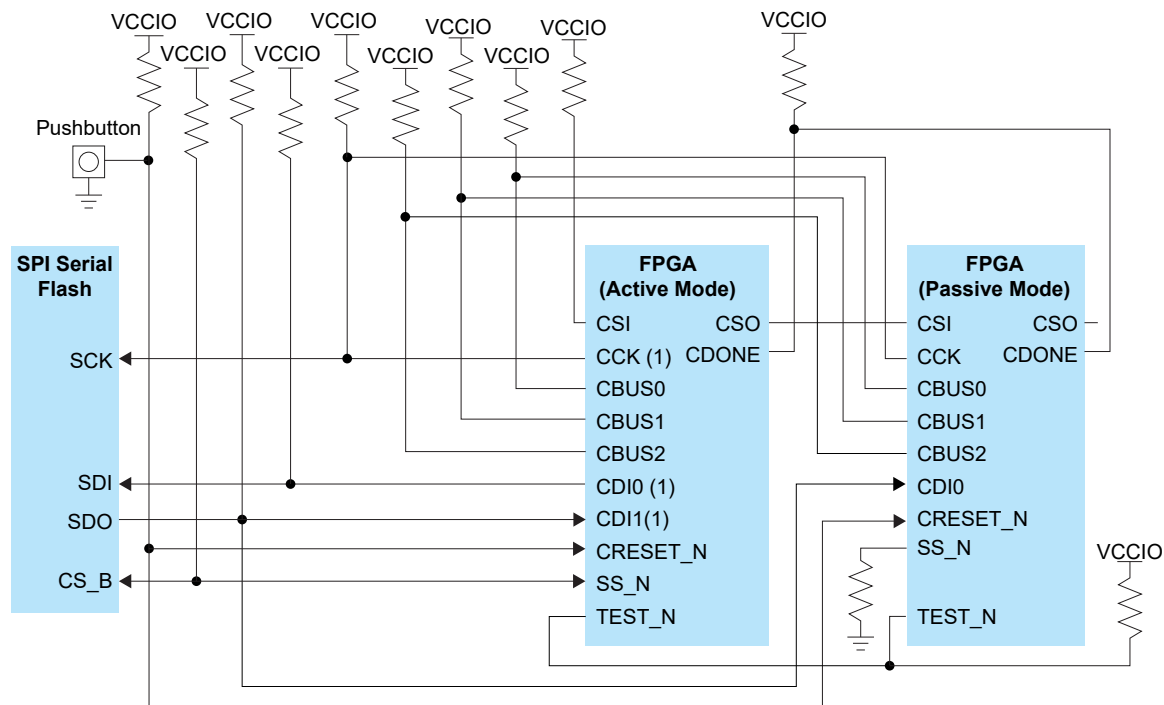
In a daisy chain, the FPGA closest to the configuration data source is the most upstream FPGA and the FPGA furthest from the source is the most downstream FPGA. The most upstream FPGA typically provides the configuration clock. All other FPGAs are in passive serial mode.



**Important:** Do not connect the NSTATUS pins of multiple FPGAs together when configuring in daisy chain configuration.

Figure 38: Serial Daisy Chain Configuration Interface Example

See **Resistors in Configuration Circuitry** on page 51 for the resistor values.



**Note:** 1) The external pull-up is optional unless required by an external load.





# Resistors in Configuration Circuitry

Efnix recommends that you use 10 k $\Omega$  for all unspecified pull-up and pull-down resistors in configuration circuitries.



**Important:** Perform an IBIS simulation to analyze the impact of pull-up and pull-down resistors on the signal integrity of dual-purpose configuration pins. Typically, 10 k $\Omega$  pull-up and pull-down resistors do not significantly affect either the single-ended or differential signals.

Alternatively, you can calculate your own pull-up or pull-down resistance,  $R_{USER}$ , shown in the following sections.



**Learn more:** The internal weak pull-up resistance, internal weak pull-down resistance, and Schmitt Trigger thresholds values used in the following formulas are included in the Trion<sup>®</sup> Data Sheet in the [Documentation page of the Support Center](#).

## User-Defined Pull-Up Resistor Values

$$R_{USER} = (R_{CPU} \times R_{IPU}) \div (R_{IPU} \cdot R_{CPU})$$

where:

- $R_{USER}$  = User-defined pull-up resistance
- $R_{CPU}$  = Combined pull-up resistance
- $R_{IPU}$  = Internal weak pull-up resistance

The combined pull-up resistance,  $R_{CPU}$ , can be derived using the following formula:

$$VT+ \leq VCCIO \times (R_{IPD} \div (R_{CPU} + R_{IPD}))$$

where:

- $VT+$  = Schmitt Trigger low-to-high threshold
- $VCCIO$  = I/O bank power supply
- $R_{IPD}$  = Internal weak pull-down resistance

## User-Defined Pull-Down Resistor Values

$$R_{USER} = (R_{CPD} \times R_{IPD}) \div (R_{IPD} \cdot R_{CPD})$$

where:

- $R_{USER}$  = User-defined pull-down resistance
- $R_{CPD}$  = Combined pull-down resistance
- $R_{IPD}$  = Internal weak pull-down resistance

The combined pull-down resistance,  $R_{CPD}$ , can be derived using the following formula:

$$VT- \geq VCCIO \times (R_{CPD} \div (R_{CPD} + R_{IPU}))$$

where:

- $VT-$  = Schmitt Trigger high-to-low threshold
- $VCCIO$  = I/O bank power supply
- $R_{IPU}$  = Internal weak pull-up resistance

# Configuration Timing

Trion® FPGA configuration timing is process dependent. The following tables show the timing parameters for the various configuration modes.



**Important:** Refer to the data sheet for your Trion® FPGA for the timing specifications for these parameters.

**Table 22: All Modes**

Symbol	Parameter
$t_{\text{CRESET\_N}}$	Minimum CRESET_N low pulse width required to trigger re-configuration.
$t_{\text{USER}}$	Minimum configuration duration after CDONE goes high before entering user mode.

**Table 23: Active Mode**

Symbol	Parameter
$f_{\text{MAX\_M}}$	Active mode configuration clock frequency.
$t_{\text{SU}}$	Setup time.
$t_{\text{H}}$	Hold time.

**Table 24: Passive Mode**

Symbol	Parameter
$f_{\text{MAX\_S}}$	Passive mode configuration clock frequency.
$t_{\text{CLKH}}$	Configuration clock pulse width high.
$t_{\text{CLKL}}$	Configuration clock pulse width low.
$t_{\text{SU}}$	Setup time.
$t_{\text{H}}$	Hold time.
$t_{\text{DMIN}}$	Minimum time between deassertion of CRESET_N to first valid configuration data.

**Table 25: JTAG Mode**

Symbol	Parameter
$f_{\text{TCK}}$	TCK frequency.
$t_{\text{TDISU}}$	TDI setup time.
$t_{\text{TDIH}}$	TDI hold time.
$t_{\text{TMSU}}$	TMS setup time.
$t_{\text{TMSH}}$	TMS hold time.
$t_{\text{TCKTDO}}$	TCK falling edge to TDO output.

## Selecting the Right SPI Flash Device

Trion® FPGAs support an SPI flash memory interface for active mode configuration. Use these guidelines to help choose the correct flash device for your Trion® FPGA.

- **Configuration Bits**—Ensure that your chosen flash device has enough bits to store the configuration bitstream.
  - *Single image*—Find the configuration bits a single image uses (refer to **Table 1: Trion FPGA Bitstream Size** on page 4).
  - *Multiple images*—Find the configuration bits a single image uses (refer to **Table 1**). Multiply the number of bits times the number of images to determine the total bits required to store the full bitstream.
  - *Daisy chain*—Use the formula  $(i \times b) + (2048 \times (i - 1))$  where  $i$  is the number of images and  $b$  is the configuration bits for each image. For example, a daisy chain of three T8 FPGAs uses  $(3 \times 1,386,584) + (2,048 \times (3-1)) = 4,159,752 + 4,096 = 4,163,848$  bits.
- **Configuration Bus Width**—Determine the supported configuration bus width for the SPI flash device in **Table 5: SPI Hardware Settings** on page 10 .
- **SPI Clock Frequency**—Ensure that your SPI flash device supports a clock frequency that is higher than the SPI active configuration clock frequency as described in **Table 10: Internal Oscillator Clock Settings** on page 20.
- **Required Voltage**—Make sure the voltage your SPI flash device requires is the same as the FPGA I/O bank voltage.
- **Temperature Range**—Check that the SPI flash device's temperature range is compatible with the operating temperature as described in the FPGA data sheet.

## Supported Flash Devices

Table 26: Supported Flash Devices

Manufacturer	Family Part Number
GigaDevice	GD25Q, GD25WQ, and GD25LQ
Macronix	MX25L, MX25U, MX25V, MX75L, and MX75U
Puya Semiconductor	P25Q
Winbond	W25Q
Micron	M25P and MT25Q
XTX	XT25F
Atmel (Adesto Technologies)	AT25SF
ISSI	IS25LP128 and IS25WP512M



**Note:** Efinix recommends using SPI NOR flash memories.

# Connecting Programming Hardware

You can program Efinix FPGA or the SPI flash using FTDI Mini Modules. This section describes the hardware connections required. See [Using the Efinity Programmer](#) on page 60 for instructions about SPI and JTAG programming using the Efinity® Programmer.

## SPI Programming Connections

The following figure illustrates the connection required when programming the SPI flash with FTDI FT2232H and FT4232H Mini Module.

Figure 41: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections

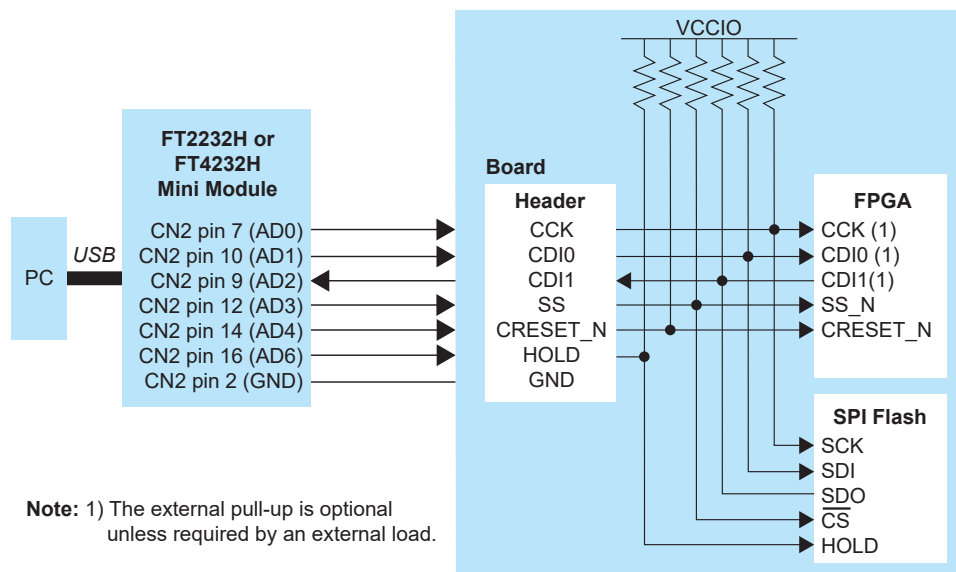


Figure 42: SPI Flash 1.8V Programming with FTDI FT2232H and FT4232H Mini-Module Connections

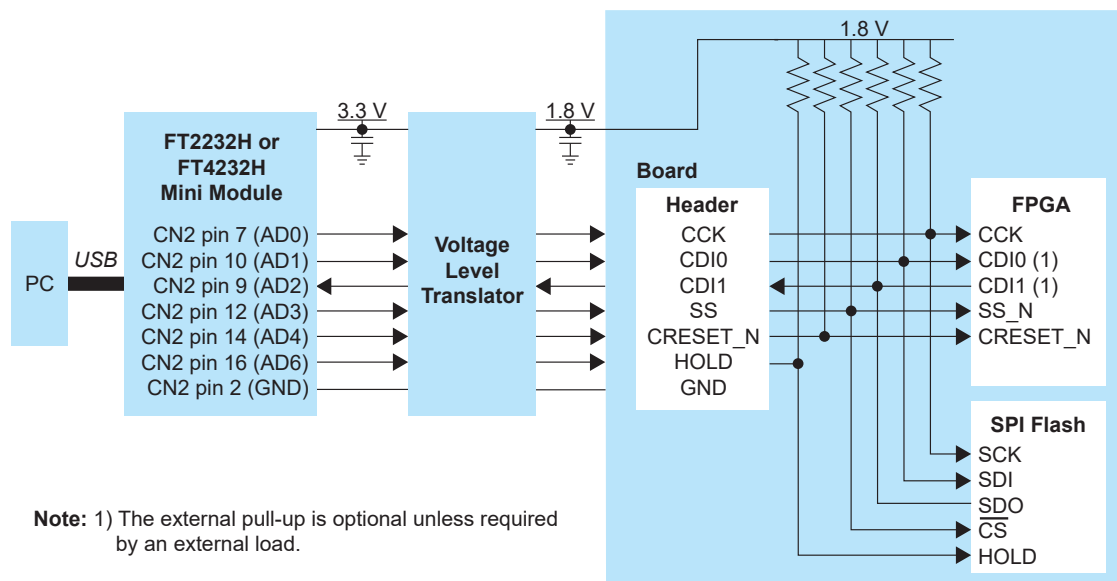
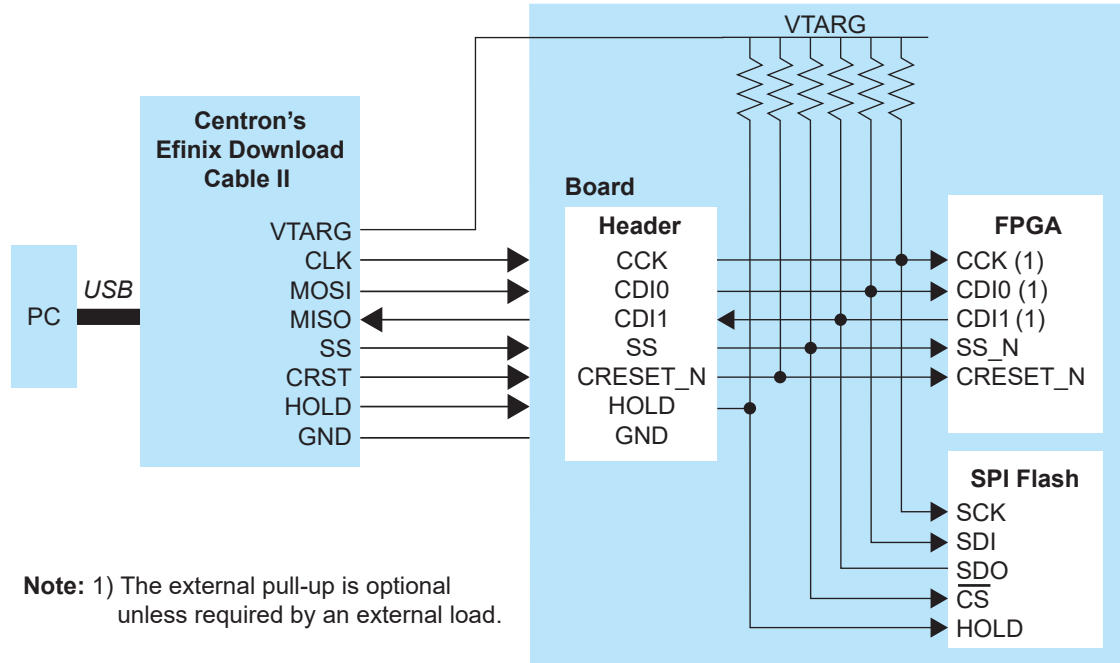


Figure 43: SPI Flash Programming with Centron's Efinix Download Cable II



# JTAG Programming Connections

## Connecting a JTAG Cable

Efnix does not recommend using the FTDI cable C232HM-DDHSL-0 for JTAG programming due to the possibility of the FPGA not being recognized or the potential for programming failures.

## Connecting a JTAG Mini Module

When programming T4, T8, T13, T20WLCSP80, T20QFP100F3, T20QFP144, T20BGA256, and T20BGA169 FPGAs, use this connection:

Figure 44: Connect FT2232H Mini Module to JTAG Pins plus CRESET\_N and SS\_N

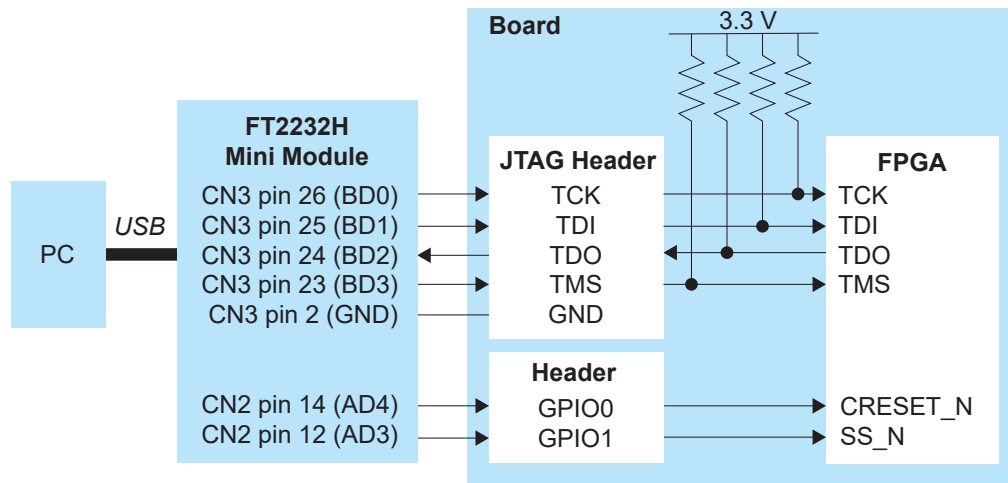


Figure 45: Connect FT4232H Mini Module to JTAG Pins plus CRESET\_N and SS\_N

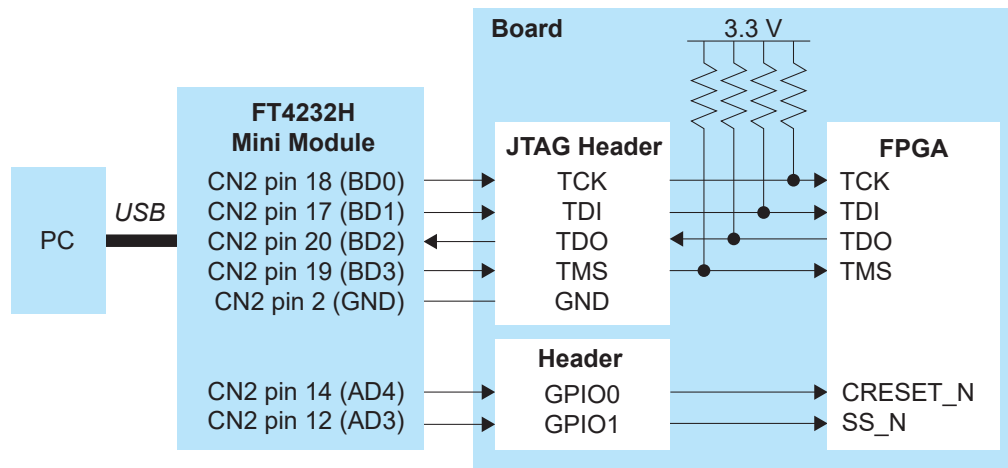
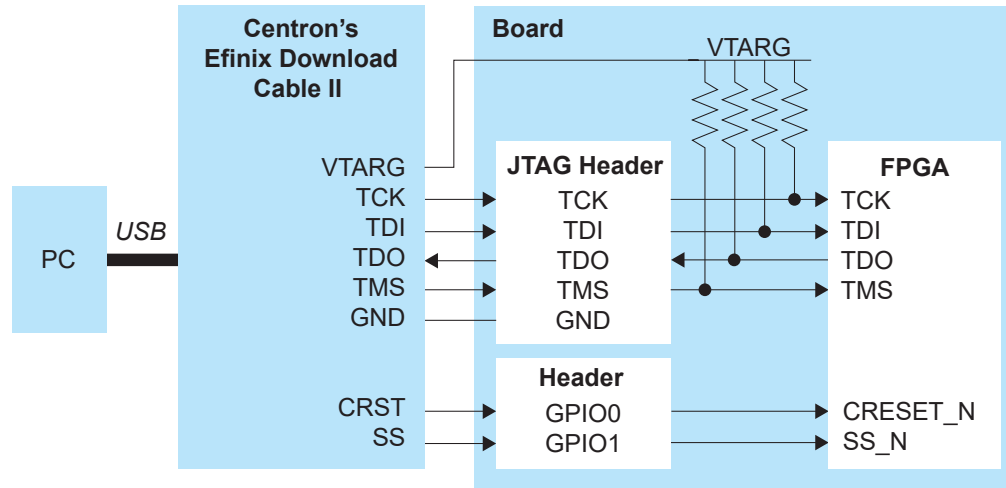


Figure 46: Connect Centron's Efinix Download Cable II to JTAG Pins



**Note:** This figure uses the `CRESET_N` and `SS_N` pins in addition to the standard JTAG pins. However, this setup is only needed for JTAG configuration. You can use the standard 4 JTAG pins and any cable for other JTAG functions.

When programming T20BGA324, T20BGA400, T35, T55, T85, and T120 FPGAs, use this connection:

Figure 47: Connect FT2232 Mini Module to JTAG Pins

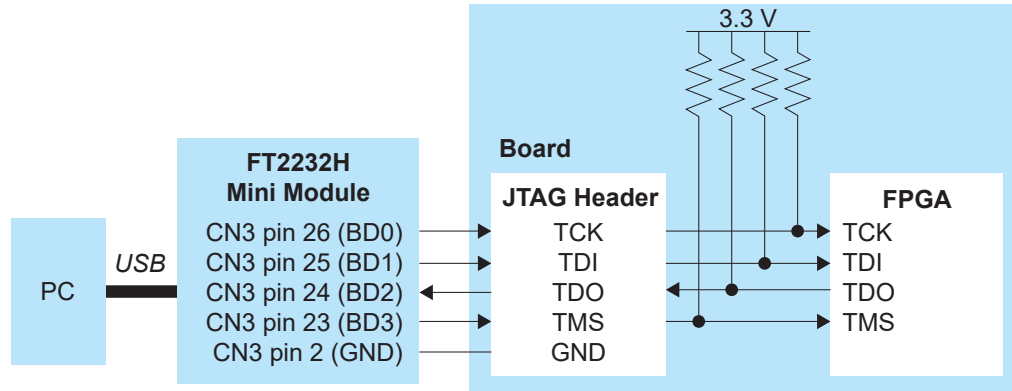


Figure 48: Connect FT4232 Mini-Module to JTAG Pins

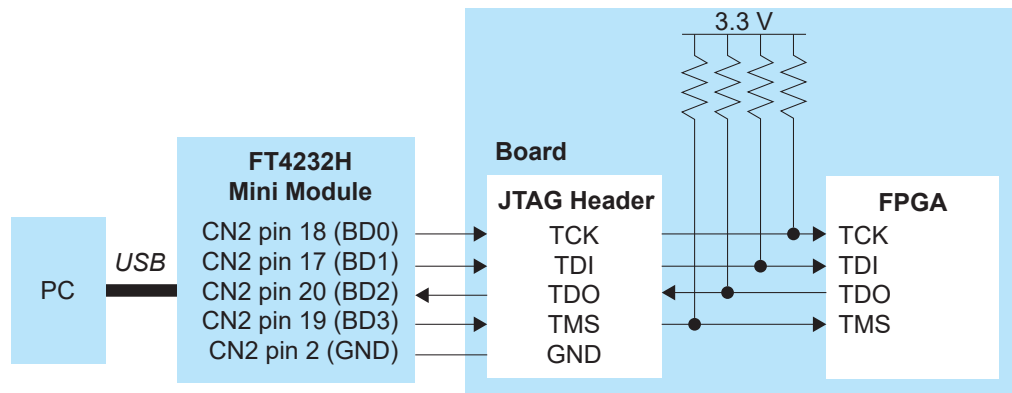
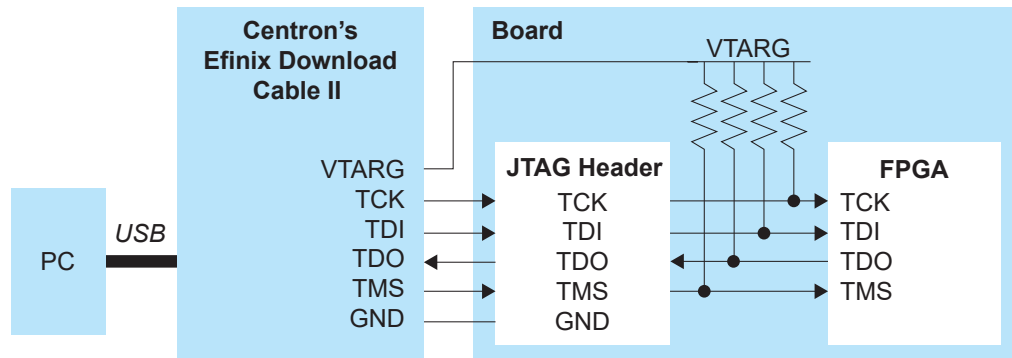


Figure 49: Connect Centron's Efinix Download Cable II to JTAG Pins



## Supported Download Cables

*Table 27: Verified Supported Download Cables*

Manufacturer	Type	Part Number
FTDI	Dual Channel	FT2232
	Quad Channel	FT4232
	Single Channel	FT232L
Centron	Single Channel	Efinix Download Cable II



**Note:** For more details on the specifications and setup of the Centron's Efinix Download Cable II, refer to the [Centron cable information sheet](#).

# Using the Efinity Programmer

The Efinity<sup>®</sup> software has a Programmer you use to configure Trion<sup>®</sup> FPGAs. You can run the Programmer using the GUI or with the command line.

## Generate a Bitstream (Programming) File

When you run the automated flow, the software automatically generates bitstream files that you can use to configure your target device. You can also generate the bitstream files manually. To generate bitstream files from the command line, use the following command:

### Example: Generate a Bitstream File from the Command Line

Linux:

```
> efx_run.py <project name>.xml --flow pgm
```

Windows:

```
> efx_run.bat <project name>.xml --flow pgm
```

The software generates these files in the **outflow** directory:

- **.hex** file as *<project name>.hex*. Use this file to program in SPI active or passive mode.
- **.bit** file as *<project name>.bit*. Use this file for JTAG programming.



**Important:** With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG.

The bitstream file includes programming options you set for your project (e.g., to initialize user memory or set configuration mode). If you change these options you must regenerate the bitstream file. See [Project-Based Programming Options](#) on page 75.



**Note:** The software does not generate bitstream files for preliminary devices.

## Working with Bitstreams

You can use the Efinity Programmer to manipulate a bitstream before programming an FPGA or flash device.

### *Edit the Bitstream Header*

You can use the Programmer to edit the bitstream header information, for example, to add project or revision information. To edit the header:

1. In the Programmer, choose **File > Edit Header...** or click the toolbar icon to open the **Edit Image Header** dialog box. The window shows the default header information.
2. Edit the header.
3. Click **Save**.



**Important:** When editing the bitstream header, if you remove any of the auto-generated information (such as `Device: <name>`), the Programmer may not be able to recognize the bitstream. Efinix recommends that you only append a small amount of information to the auto-generated data if you want to customize or annotate the header. The header can be a maximum of 256 characters, including the auto-generated text.

If you want to write your own program to detect which device the bitstream targets (e.g., using a microprocessor and SPI passive mode), be sure to keep all of the auto-generated header, specifically the `Device: <name>` string.

### *Export to Raw Binary Format*

The Efinity® software v2018.4 and later supports raw binary (**.bin**) format for use with third-party flash programmers. To export to this format:

1. Open the Programmer.
2. Select the bitstream file.
3. Click **Export**.
4. Specify the filename.
5. Click **Save**.

You can also convert the file to **.bin** at the command line as described in [Convert to Intel Hex Format at the Command Line](#) on page 62.

### *Export to .svf Format*

The Efinity® software v2021.1 and later supports serial vector format (**.svf**) files for use with third-party JTAG programmers. To export to this format:

1. Open the Programmer.
2. Select a bitstream file.
3. Click **Export**.
4. Specify the filename.
5. Choose **Serial Vector Format (\*.svf)** as the **Files of type**.
6. Click **Save**.



**Note:** For more information on using this bitstream format, refer to Working with JTAG .svf Files section of the Efinity® Programmer user Guide.

You can also convert the file to **.hex** at the command line as described in [Convert to Intel Hex Format at the Command Line](#) on page 62.

## Convert to Intel Hex Format at the Command Line

You can convert a bitstream file to Intel Hex and other formats at the command line using this command:

```
export_bitstream.py [--help] [--family FAMILY] [--idcode IDCODE] [--freq FREQ]
  [--sdr_size SDR_SIZE] [--tir_length TIR_LENGTH] [--hir_length HIR_LENGTH]
  [--tdr_length TDR_LENGTH] [--hdr_length HDR_LENGTH] [--enter_user_mode {on, off}]
  format input_file output_file
```

**Table 28: export\_bitstream.py Positional Arguments**

Argument	Input	Description
format	hex_to_bin, hex_to_intelhex, bin_to_hex, intelhex_to_hex, hex_to_svf	Conversion type.
input_file	Filename	Image file source.
output_file	Filename	Image file destination.

**Table 29: export\_bitstream.py Options**

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--family	N/A	Family name	Device family (SVF only)
--idcode	N/A	Identification code	JTAG IDCODE (SVF only).
--freq	N/A	Number	JTAG frequency (SVF only).
--sdr_size	N/A	Number	Approximate JTAG <code>shift_dr</code> size before cycling to idle state (SVF only).
--tir_length	N/A	Number	JTAG bypass trailer instruction register length (SVF only).
--hir_length	N/A	Number	JTAG bypass header instruction register length (SVF only).
--tdr_length	N/A	Number	JTAG bypass header data register length (SVF only).
--enter_user_mode	N/A	on, off	Enter user mode after JTAG configuration (SVF only).

The following example shows conversion of the bitstream **hex** file to **bin** format:

### Example: Converting Hex to Bin

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\export_bitstream.py hex_to_bin new_project.hex test2.bin
```

## Combine Bitstreams and Other Files

You may want to store multiple bitstreams or other data into the same flash device on your board. For example, you can combine files for:

- Multi-image configuration using the CBSEL pins
- Internal reconfiguration
- Programming FPGAs in a daisy chain
- Programming a bitstream and other files such as a RISC-V application binary

You use the **Combine Multiple Image Files** dialog box to choose files to combine into a single file for programming. Choose one of the following modes:

**Table 30: Modes when Combining Images**

Mode	Use For	Number of Images	Notes
Selectable Flash Image	Multi-image configuration	Up to 4	Use this mode if you want the CBSEL pins to control which image the FPGA loads. For this mode, you also need to choose <b>Image Type &gt; External Controller Flash Image</b> . See <a href="#">Program Multiple Images (CBSEL)</a> on page 64
	Internal reconfiguration	Up to 4	Use this mode if you want the internal reconfiguration pins to determine which image the FPGA loads. For this mode, you also need to choose <b>Image Type &gt; Remote Update Flash Image</b> . See <a href="#">Program Multiple Images (Internal Reconfiguration)</a> on page 65
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors <sup>(6)</sup> .	See <a href="#">Program a Daisy Chain</a> on page 66
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	See <a href="#">Program Multiple Images (Bitstream and Data)</a> on page 65



**Note:** When you combine images for an MCU-controlled system or SPI passive daisy chain, the Programmer adds padding between the images as needed. Therefore, you can send the entire bitstream continuously until all devices in the chain are configured.

<sup>(6)</sup> Efinity Programmer does not apply any constraints for combining multiple images. Efinix recommends that you run the IBIS simulation to check the signal integrity if you need to connect more than 4 devices in the same daisy chain.

## SPI Programming

You can program Efinix FPGAs using the SPI interface and a **.hex** file.

### *Program a Single Image*

In single image programming mode, you configure one FPGA with one image.

1. Click the **Select Image File** button.
2. Browse to the **outflow** directory and choose *<filename>.hex*.
3. Choose **SPI Active** or **SPI Passive** configuration mode.
4. Click **Start Program**. The console displays programming messages.

### *Program Multiple Images (CBSEL)*

In this programming mode, you specify up to four images that can configure one FPGA. You then use the FPGA's CBSEL pins to select which image to use. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Selectable Flash Image**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Choose **Image Type > External Flash Image**. This setting tells the FPGA to use the CBSEL pins.
6. Click in the table row corresponding to the position for which you want to add an image.
7. Click **Add Image**.
8. Select the image file to place in that location.
9. Click **OK**.
10. Repeat steps 6 through 9 as needed. You can add up to four images.
11. Click **Apply** to generate the combined image file.
12. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
13. Click **Start Program**.

## Program Multiple Images (Internal Reconfiguration)

In this programming mode, you specify up to four images that can configure one FPGA. You then use the FPGA's internal reconfiguration interface to select which image to use. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Selectable Flash Image**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Choose **Image Type > Remote Update Flash Image**.



**Note:** When using internal reconfiguration, you **must** choose **Remote Update Flash Image**. If you choose **External Flash Image**, the FPGA reconfigures with the first image as specified by the CBSEL pins instead of the golden image.

6. Click in the table row corresponding to the position for which you want to add an image.
7. Click **Add Image**.
8. Select the image file to place in that location.
9. Click **OK**.
10. Repeat steps 6 through 9 as needed. You can add up to four images.
11. Click **Apply** to generate the combined image file.
12. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
13. Click **Start Program**.



**Note:** For more information on using the internal reconfiguration feature, refer to [AN 010: Using the Internal Reconfiguration Feature to Update Efinix FPGAs Remotely](#).

## Program Multiple Images (Bitstream and Data)

In this programming mode, you specify one bitstream and one or more data files to combine into a single file for programming. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Generic Image Combination**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Click **Add Image**.
6. Select the image file to place in that location.
7. Click **Open**. The image file and flash length are displayed in the table.
8. Specify the flash address.
9. Repeat steps 5 through 8 as needed.



**Note:** If you want to combine a bitstream and a RISC-V binary, use 0x00000000 as the bitstream's flash address and 0x00380000 as the binary's flash address.

10. Click **Apply** to generate the combined image file.
11. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
12. Click **Start Program**.

## *Program a Daisy Chain*

In this programming mode, you specify any number of images to configure a daisy chain of FPGAs. You can choose active or passive configuration for first FPGA; the rest are in passive mode.

1. Click the **Combine Multiple Images** button.
2. Select **Daisy Chain** as the **Mode**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Click **Add Image** to add a file to the daisy chain.
6. Repeat step 5 to add as many files as you want to the chain. Use the up/down arrows to re-order the images if needed.
7. Click **Apply** to generate the combined image file.
8. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
9. Click **Start Program**.

## JTAG Programming

You can program Efinix FPGAs using the JTAG interface and a **.bit** file.

### *Trion Family JTAG Device IDs*

The following table lists the Trion JTAG device IDs.

**Table 31: Trion JTAG Device IDs**

FPGA	Package	JTAG Device ID
T4, T8	BGA81	0x0
T8	QFP144	0x00210A79
T13	All	0x00210A79
T20	WLCSP80, QFP100F3, QFP144, BGA169, BGA256	0x00210A79
T20	BGA324, BGA400	0x00240A79
T35	All	0x00240A79
T55, T85, T120	All	0x00220A79

### *Program a Single Image*

In single image programming mode, you configure one FPGA with one image.

1. Click the **Select Image File** button.
2. Browse to the **outflow** directory and choose *<filename>.bit*.
3. Choose the **JTAG** configuration mode.
4. Click **Start Program**. The console displays programming messages.

## Program Using a JTAG Chain

You can program an FPGA that is part of a JTAG chain. The chain can include Trion® FPGAs as well as other devices. You define your JTAG chain using a JTAG chain file. You import the JTAG chain file into the Programmer to perform programming. The JTAG chain file is an XML file (.xml) that includes all of the devices in the chain. For example:

```
<?xml version="1.0"?>
<chain>
  <device chip_num="1" id_code="0x00210a79" ir_width="4" istr_code="1100" />
  <device chip_num="2" id_code="0x00210a79" ir_width="4" istr_code="1100" />
  <device chip_num="3" id_code="0x00210a79" ir_width="4" istr_code="1100" />
</chain>
```

where:

- chip\_num is the device order starting from position 1.
- id\_code is the hexadecimal JEDEC device ID (all lowercase letters)
- ir\_width is the width of the instruction register in bits
- istr\_code is the binary IDCODE instruction



**Note:** For Trion FPGAs, use 1100 as the istr\_code.

To program using a JTAG chain:

1. Create a JTAG Chain File using a text editor.
2. Open the Programmer.
3. Choose your **USB Target and Image**.
4. Select **JTAG** as the **Programming Mode**.
5. Click the Import JCF toolbar button.
6. Browse to your JTAG Chain File and click **Open**.
7. Select which device you want to program in the drop-down list next to the **JTAG Programming Mode** option.
8. Click **Start Program**.



**Note:** If you implement both the daisy chain and JTAG chain together, ensure that the daisy chain is fully completed before executing the JTAG chain. Because the daisy chain requires CSIs to be connected to CSOs, the JTAG chain will only configure successfully when the CSIs are high.

## Program using a JTAG Bridge

Programming with a JTAG bridge is a two-step process: first you configure the FPGA to turn it into a flash programmer (**.bit**) and second you use the FPGA to program the flash device with the bitstream (**.hex**).

The SPI Active using JTAG Bridge mode (formerly named SPI Active using JTAG Bridge (New)) has pre-built flash loader (**.bit**) files that you can use. These **.bit** files do not require an external clock source. You can still use your own **.bit** file if you choose to do so.



**Note:** The JTAG bridge modes were changed in the Efinity software v2025.1. If you are using an older version of software and want to use the SPI Active using JTAG Bridge (Legacy) mode, refer to **Appendix: Program using a JTAG Bridge (Legacy)**.

The JTAG bridge bitstream files bundled with v2025.1 and higher can only be used with v2025.1 or higher. You cannot use older bundled JTAG bridge bitstream files with v2025.1 or higher, and you cannot use the v2025.1 or higher bundled files with older software versions. If you need to use the older bundled files, use the Programmer v2024.2.

**Tip:** If you would like to incorporate the RTL files for the new flash loader into your own design, use the JTAG to SPI Flash Bridge core in the IP Manager.

To program using a JTAG bridge:

1. Choose the **USB Target**.
2. In the **Image** box, click the **Select Image File** button to browse for the **.hex** file to program the flash device.
3. Choose the **SPI Active using JTAG Bridge** programming mode.
4. Turn on the **Auto configure JTAG Bridge Image** option.
5. Specify your own **.bit** file.
  - a) In the **Programming Mode** box, click **Select Image File**.
  - b) The **Open Image File** dialog box opens to a directory of available pre-built **.bit** files. Choose the file for your FPGA, or browse to find your own **.bit** file. The Programmer remembers which file you specify and uses it automatically the next time you run the Programmer.
6. Choose the **SPI Active Options** and **JTAG Options**.

Option	Description
<b>Select Flash</b>	x8 mode only. Choose whether to use the upper flash, lowre, flash, or both.
<b>Starting Flash Address</b>	Specify the address if other than the default.
<b>Flash Length</b>	Specify the length if other than the default.
<b>Erase Before Programming</b>	Default: on. When turned on, the Programmer erases the flash device before re-programming it.
<b>Select Verify Method</b>	<p><b>Normal verify</b>—The FPGA computes an on-chip hash from the read back flash data to perform verification. <b>Normal verify</b> is <b>significantly</b> faster than in the Programmer v2024.2 and lower (so much faster that you might think it did not do anything).</p> <p><b>Fast verify</b>—Similar to normal verify, but requires a SPI x4 width (quad mode). The Programmer cannot detect whether your board is using quad mode; if your board is not using it and you try to use fast verify, programming will fail.</p> <p><b>Skip verify</b>—Do not verify the flash.</p>

Option	Description
<b>Device Select</b>	Choose the JTAG device ID of the FPGA to program.
<b>JTAG Clock Speed</b>	Choose a speed or specify a custom one.

7. Click **Start Program**. The Programmer first configures the FPGA and then programs the flash device.

## JTAG Programming with FTDI Chip Hardware

These instructions describe how to program Trion® FPGAs using the FTDI Chip FT2232H and FT4232H Mini Modules. Efinix® has tested the hardware for use with Trion® FPGAs.



**Note:** Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures.

1. Open the Efinity® software.
2. Open the Efinity® Programmer.
3. Click the Select Bitstream Image button.
4. Browse to your image and click **OK**.
5. Choose one of the following in the **USB Target** drop-down list:
  - **Dual RS232 HS** for FT2232H Mini Module
  - **FT4232H\_MM** for FT4232H Mini Module
6. Choose **JTAG** from the **Programming Mode** drop-down list.
7. Click **Start Program**.

## FTDI Programming at the Command Line

The Efinity software includes a Python script you can use for programming FTDI modules at the command line.

```
ftdi_pgm.py [--help] [--mode MODE] [--output_file OUTPUT_FILE] [--url URL] [--aurl AURL]
[--xml XML] [--num NUM] [--board_profile BOARD_PROFILE] [--address ADDRESS]
[--num_bytes NUM_BYTES] [--burst_size BURST_SIZE] [--jtag_bridge_mode JTAG_BRIDGE_MODE]
[--jtag_clock_freq JTAG_CLOCK_FREQ] [--verify_method VERIFY_METHOD]
[--check_flash_if_supported CHECK_FLASH_IF_SUPPORTED] [--spi_active_freq SPI_ACTIVE_FREQ]
[--spi_passive_freq SPI_PASSIVE_FREQ] [--list_usb] [input_file]
```

Table 32: `ftdi_pgm.py` Positional Arguments

Argument	Description
<code>input_file</code>	HEX file generated from <code>efx_pgm</code> .

Table 33: `ftdi_pgm.py` Options

Option (Long)	Option (Short)	Input	Description
<code>--help</code>	<code>-h</code>	None	Show help.

Option (Long)	Option (Short)	Input	Description
--mode	-m	passive, active, jtag, jtag_chain, erase_flash, read_flash, jtag_bridge, jtag_bridge_x8	<p>Programming mode.</p> <p>See the <a href="#">Efinity Programmer User Guide</a>.</p> <p>In Efinity software versions prior to v2025.1, the <code>jtag_bridge</code> and <code>jtag_bridge_new</code> options were named <code>jtag_bridge_new</code> and <code>jtag_bridge_x8_new</code>, respectively.<sup>(7)</sup></p> <p>To use the JTAG bridge modes, you must have already configured the with the JTAG SPI flash loader.</p> <p>The Efinity software v2023.2 and higher includes pre-built flash loader.bit files in <code>&lt;installation directory&gt;/pgm/fli/&lt;family&gt;</code>. Refer to the <a href="#">JTAG SPI Flash Loader Core User Guide</a> for information on using the legacy flash loader.</p>
--output_file	-o	Filename	Output file used for <code>read_flash</code> mode.
--url	-u	URL	FTDI URL (see <a href="#">Identifying FTDI URLs</a> on page 73).
--aurl	-a	URL	Alternative URL ( <i>Deprecated</i> ).
--xml	-x	Filename	XML file for JTAG programming.
--num	-n	Number	Chip target number for JTAG chain programming.
--board_profile	-b	Generic Board Profile Using FT232, Digilent JTAG-HS3, FireAnt Development Board, Generic Board Profile Using FT2232H, ISX Programming Cable, Titanium Ti180J484 Dev Board, Titanium Ti180M484 Development Kit, Generic Board Profile Using FT4232, JinChen Programming Cable, TJ180A484S Development Kit, Xyloni Development Board, Generic Board Profile Using FT4234HA	Name of the board profile used.
--address	N/A	Hex number	Starting flash address for flash read and write operations.
--num_bytes	N/A	Number	Number of bytes to erase or read. For modes <code>erase</code> and <code>read</code> only.
--burst_size	N/A	Number	Individual read or write burst size in multiples of 256 bytes. For legacy JTAG bridge modes only ( <code>jtag_bridge</code> and <code>jtag_bridge_x8</code> ).
--jtag_bridge_mode	N/A	Erase, write, erase_and_write, read, all, all_no_erase	JTAG bridge programming mode.

<sup>(7)</sup> The `jtag_bridge_x8` mode is only supported in some and FPGAs. Refer to the data sheet for the modes your FPGA supports.

Option (Long)	Option (Short)	Input	Description
--jtag_clock_freq	N/A	Number	JTAG clock frequency.
--verify_method	N/A	None, onchipx1, onchipx2, onchipx4	The method used to verify the downloaded bitstream. Default: onchipx2 (On-chip hash calculation with SPI x2 mode)
--check_flash_if_supported	N/A	Hex string	Check if flash is supported using the JEDEC ID hex string (e.g., C84012).
--spi_active_freq	N/A	Number	Set SPI active frequency. Default: 6000000 Hz
--spi_passive_freq	N/A	Number	Set SPI passive frequency. Default: 3000000 Hz
--list_usb	-l	None	List the available USB target's URL.

## Linux Examples

To program in Linux:

1. Open a terminal and change to the Efinity® installation directory.
2. Type: `source ./bin/setup.sh` and press enter.
3. Use the `ftdi_program.py` command.

**Example:** Xyloni Development Board as the only board attached to your computer:

```
ftdi_program.py <filename>.bit -m jtag
```

**Example:** Trion T120 BGA324 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
ftdi_program.py <filename>.bit -m jtag --url ftdi://ftdi:2232h:FT5ECP6E/1
--aur1 ftdi://ftdi:2232h:FT5ECP6E/1
```

## Windows Examples

To program in Windows:

1. Open a command prompt and change to the Efinity® installation directory.
2. Type: `.\bin\setup.bat` and press enter.
3. Use the `ftdi_program.py` command.

**Example:** Xyloni Development Board as the only board attached to your computer:

```
%EFINITY_HOME%\bin\python3
\ftdi_program.py <filename>.bit -m jtag
```

**Example:** Trion T120 BGA324 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
%EFINITY_HOME%\bin\python3
\ftdi_program.py <filename>.bit
-m jtag --url ftdi://ftdi:2232h:FT5ECP6E/1 --aur1 ftdi://ftdi:2232h:FT5ECP6E/1
```

## Identifying FTDI URLs

Certain Efinity<sup>®</sup> scripts contain the `--url` and `--aur1` options, which require the input of an FTDI URL.



**Important:** You only need to specify the `--url` and `--aur1` options if you have more than one board with an FTDI chip connected to your computer.

Only supported in T20 (BGA324 and BGA400), T35, T55, and T120 FPGAs.

The FTDI URL is in the format:

```
ftdi://ftdi:<product>:<serial>/<interface>
```

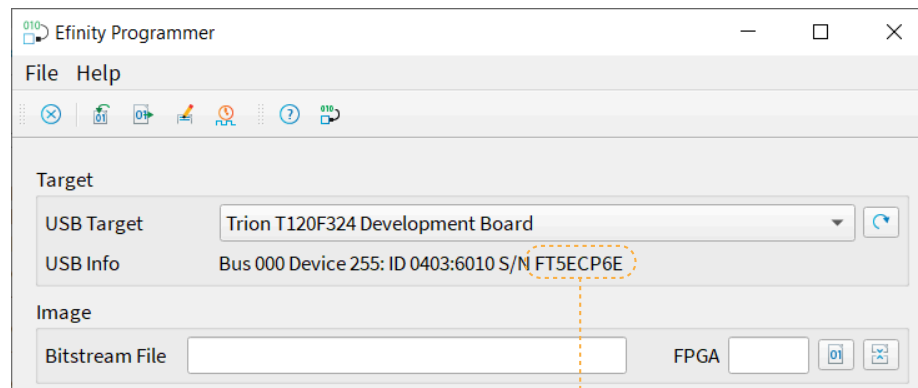
where:

`<product>` is the USB product ID of the device

<code>&lt;product&gt;</code>	Board
232h	Trion T8 Development Board
2232h	Trion T20 MIPI Development Board Trion T20 BGA256 Development Board Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board
4232h	Xyloni Development Board

`<serial>` is the serial number of the FTDI chip. (Optional)

- If you only have one Efinix<sup>®</sup> development board or FTDI device connected to your computer, you do not need to specify the serial number.
- In the Efinity<sup>®</sup> software v2020.2 and higher, the Programmer displays the serial number of the FTDI device in the **USB Info** string. The serial number is a string beginning with FT.



The string after S/N is the FTDI serial number

`<interface>` is the interface number. For Efinix<sup>®</sup> development boards, `<interface>` is always 1.

## Using the Command-Line Programmer

To run the Programmer using the command line, use the command:

### Example: Command-Line Programmer

Linux:

```
efx_run.py <project name>.xml --flow program [--pgm_opts [mode=MODE] [settings_file]]
```

Windows:

```
efx_run.bat <project name>.xml --flow program [--pgm_opts [mode=MODE] [settings_file]]
```

### Options

--pgm\_opts mode specifies the configuration mode. The available modes are:

**Table 34: --pgm\_opts Modes**

Mode	Description
active	SPI Active configuration.
passive	SPI Passive configuration.
jtag	JTAG programming. See the <a href="#">Efinity Programmer User Guide</a> for more information about programming with the JTAG interface.
jtag_bridge	SPI Active using JTAG bridge mode.
jtag-bridge_x8	SPI Active x8 using JTAG bridge mode (used with two flash devices). <sup>(8)</sup>

In active mode, the FPGA configures itself from flash memory; in passive mode, a CPU drives the configuration. If you do not specify the mode, it defaults to active. For example, to use JTAG mode, use the command:

```
efx_run.py <project name>.xml --flow program --pgm_opts mode=jtag
```

--pgm\_opts **settings\_file** specifies a file in which you have saved all of the programming options. A settings file is useful for performing batch programming of multiple devices.



**Note:** See [Programming Options](#) for more programming options.

<sup>(8)</sup> Used with two flash devices. Only supported in some FPGAs. Refer to the data sheet for the modes your FPGA supports.

## Project-Based Programming Options

You specify project-based programming options in the **Project Editor > Bitstream Generation** tab in the Efinity® software. Efinix FPGAs support active and passive configuration in a variety of modes.



**Note:** Some of these project settings affect bits in the bitstream. Therefore, when you program an FPGA with the Programmer, the setting you make in the Project Editor should match what you intend to use in the Programmer.

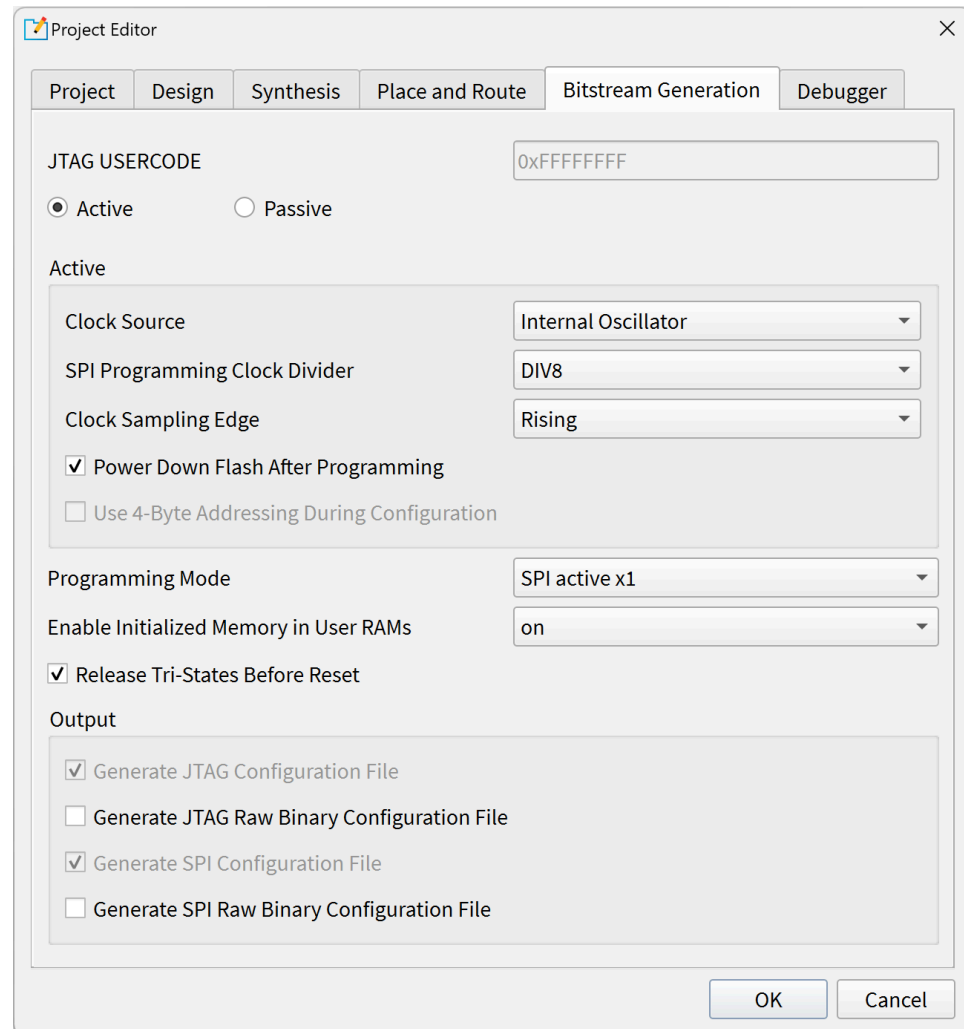
**Table 35: Project-Specific Programming Options**

Option	Notes
Active/Passive	Active: SPI active mode. Passive: SPI passive mode. Your choice of active or passive affects the pinout and determines which choices are available in the Programming Mode box.
JTAG USERCODE	Fixed at: 0xFFFFFFFF
Clock Source	For Trion FPGAs, this option is always <b>Internal Oscillator</b> .
SPI Programming Clock Divider	Choose the divider for the SPI clock. This setting is reflected in the bitstream file. Default: DIV8
Clock Sampling Edge	For Trion FPGAs, this option is always <b>Rising</b> .
Power down flash after programming	Enable this option to power down the flash device after the FPGA finishes programming. This setting is reflected in the bitstream file, and you can only set it here. Default: On
Use 4-byte addressing during configuration	This option is not supported for all Trion FPGAs.
Programming mode	Choose the programming mode and width; the choices depend on the FPGA and package you are targeting. This setting is reflected in the bitstream file, and you can only set it here. Default: SPI <active or passive> x1
Enable Initialized Memory in User RAMs	This setting is reflected in the bitstream file, and you can only set it here. <b>on:</b> The bitstream has initialized memory. <b>off:</b> The bitstream does not have initialized memory. <b>smart:</b> For the Trion family, this option has the same effect as <b>on</b> .
Release Tri-States before Reset	During configuration, core signals are held in reset and the I/O pins are tri-stated. These states are released when the FPGA enters user mode. On: (default) I/O pins are released from tri-state before the core is released from reset (use this option when the application is core sensitive). Off: Core signals are released from reset before the I/O pins are released from tri-state (use this option when the application is I/O sensitive).
Generate JTAG configuration file	On (always): Generate a <b>.bit</b> file for JTAG configuration.
Generate JTAG raw binary configuration file	On: Generate a <b>.bin</b> file (raw binary) for JTAG configuration. Off (default): Do not generate a <b>.bin</b> file.

Option	Notes
Generate SPI configuration file	On (always): Generate a <b>.hex</b> file for SPI programming.
Generate SPI raw binary configuration file	On: Generate a <b>.bin</b> file (raw binary) for SPI programming. Off (default): Do not generate a <b>.bin</b> file.

When you change one of these options, you can simply re-run the bitstream generation flow step. You do not need to recompile the design.

**Figure 50: Setting Programming Options**



**Notice:** Refer to the data sheet for your FPGA for information on which configuration options it supports.

# Verifying Configuration

You can confirm that the FPGA is configured by your bitstream successfully by checking on the following:

- Use the Efinity Programmer to check whether the FPGA is in user mode. In the Programmer, click the **Device Configuration Status > Refresh Device Configuration Status** button. The console displays the FPGA status.
- Monitor the `CDONE` and `NSTATUS` pins to determine the status of the FPGA. The status lets you know if there is a configuration error (`CDONE = 0` and `NSTATUS = 0`).
- You can verify the bit stream stored in the SPI flash after programming is completed in **SPI Active** mode. Turn on the option in **Project Editor > Programmer > SPI Active Options > Verify After Programming** to check for errors such as flash image being corrupted during a write, an improperly skipped erase step, etc.



**Note:** Generally, Efinix recommends that you keep the **Verify After Programming** option turned on. However, if you are using the FPGA's built-in CRC checking (enabled by default) with `NSTATUS` monitoring to verify configuration, you can use that method as a way of verifying the flash (that is, if the FPGA goes into user mode, the flash write is verified).



**Note:** The **Verify After Programming** confirms the correctness of the bit stream programmed into the SPI Flash. It doesn't confirm if the device is in use afterwards.

The Efinity software adds a CRC to the bitstream. During configuration, the FPGA generates another CRC as it reads the bitstream. Then, it compares the two CRCs to see if they match. If they do not, it indicates a configuration error. The CRC error is reflected by `CDONE = 0` and `NSTATUS = 0`. The CRC check can be useful for debugging board problems such as signal integrity issues between the flash device and the FPGA.



**Note:** The CRC check is not supported in T4 or T8 FPGAs in F49 and F81 packages. The CRC is only applicable to the core logic portion of the design (not the interface).

In some cases, you may need to debug possible configuration errors. To simplify this debugging process, add logic to your RTL design that generates a signal you can monitor to confirm that the FPGA has entered user mode.

## Monitoring with the Efinity Programmer

With the board connected to your computer, you can monitor the FPGA's status with the Programmer. The following table describes the values of `CDONE` and `NSTATUS` and their meaning.

*Table 36: Monitoring with the Efinity Programmer*

CDONE	NSTATUS	Programmer Message	Description
0	0	Failure to configure was detected	Configuration failed. This may be caused by: <ul style="list-style-type: none"> <li>• The configured bitstream is for a different configuration mode or width.</li> <li>• Wrong device ID.</li> <li>• CRC error is detected during configuration.</li> </ul>
0	1	Programming ...	The FPGA is in configuration mode.

CDONE	NSTATUS	Programmer Message	Description
1	0	-	The FPGA is in transition from configuration mode to user mode conditionally.
1	1	Device is in user mode!	The FPGA is functioning correctly according to the user design.



**Note:** As NSTATUS is a dual purpose GPIO, you may observe different behavior from Efinity Programmer on NSTATUS if NSTATUS is applied as a GPIO in your bit stream.

## Monitoring with a Microcontroller or LEDs

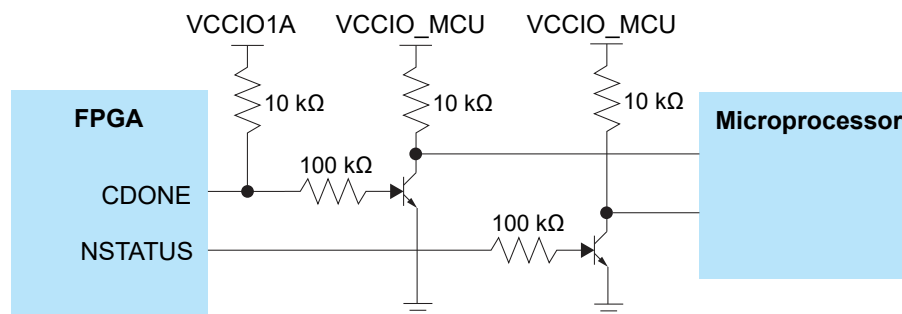
You can optionally monitor the status of CRESET\_N, CDONE and NSTATUS with a microcontroller or LEDs. CDONE is a dedicated configuration pin and you can monitor it directly. However, NSTATUS is a dual-purpose configuration pin. To use it to monitor configuration, you can connect it to a GPIO and set its output value to a constant 0.

To add NSTATUS to your design as a GPIO for monitoring:

- In the Interface Designer, create a GPIO block for NSTATUS with the following settings:
  - Instance Name:** NSTATUS
  - Mode:** Output
- In the Instance View pane, assign the NSTATUS instance to the NSTATUS package pin (refer to the pinout file to find the package pin).
- Follow these steps to set the external or internal configuration:
  - For single image (SPI active, SPI passive, and JTAG) and external controller flash image,
    - Set **Constant Output:** 0 for the NSTATUS
  - For internal configuration,
    - Assign the `cfg_error` to the NSTATUS in the RTL top module.
- Recompile the design.
- Download the bitstream to the flash memory on your board.

The following figure shows example schematics connecting a microcontroller to the FPGA's CDONE and NSTATUS pins:

*Figure 51: Connect CDONE and NSTATUS to a Microcontroller*



The MCU can verify the configuration with the following steps:

- Reset the FPGA.
- Poll CDONE for 1.
- Wait for  $t_{USER}$ .
- Sample NSTATUS.



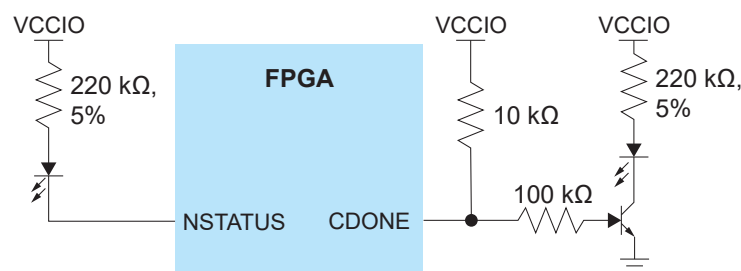
**Note:** You can add a watchdog timer in the MCU to time the system's configuration, thus preventing the system from hanging if any configuration issues arise.

**Table 37: Monitoring with a Microcontroller**

CRESET_N	CDONE	NSTATUS (After tUSER)	Description
1	0	0	Configuration failed. This may be caused by: <ul style="list-style-type: none"> <li>The configured bitstream is for a different configuration mode or width.</li> <li>Wrong device ID.</li> <li>CRC error is detected during configuration.</li> </ul>
1	0	1	The FPGA is in configuration mode.
1	1	0	The FPGA is functioning correctly according to the user design.
1	1	1	For internal reconfiguration only. The targeted application image FPGA cannot be configured successfully after 6 trials or a timeout with the golden image restored.

The following figure shows example schematics connecting LED's to the FPGA's CDONE and NSTATUS pins:

**Figure 52: Connect CDONE and NSTATUS to LEDs**



**Table 38: Observation through LEDs**

CRESET_N	CDONE LED	NSTATUS LED	Description
1	Off	Off	The FPGA is in configuration mode.
1	Off	On	Configuration fails. This may be caused by: <ul style="list-style-type: none"> <li>The configured bitstream is for a different configuration mode or width.</li> <li>Wrong device ID.</li> <li>CRC error is detected during configuration.</li> </ul>
1	On	On	The FPGA is functioning correctly according to the user design.
1	On	Off	The internal reconfiguration only, the targeted application image FPGA cannot be configured successfully after 6 trials or a timeout with the golden image restored.

# Installing USB Drivers

To program Trion® FPGAs using the Efinity® software and programming cables, you need to install drivers.

Efinix development boards have FTDI chips (FT232H, FT2232H, or FT4232H) to communicate with the USB port and other interfaces such as SPI, JTAG, or UART. Refer to the Efinix development kit user guide for details on installing drivers for the development board.



**Note:** If you are using more than one Efinix development board, you must manage drivers accordingly. Refer to [AN 050: Managing Windows Drivers](#) for more information.



**Note:** The Trion T8 BGA81 Development Boards do not have FTDI chip for USB communication. Refer to the T8 BGA81 Development Kit User Guide for more information about installing its Windows USB driver.

For your own development board, Efinix suggests using the FTDI Chip FT2232H or FT4232H Mini Modules for JTAG programming Trion® FPGAs. (You can use any JTAG cable for JTAG functions other than programming.)



**Note:** Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures.

*Table 39: USB Programming Connections*

Board	Connect to Computer with
Efinix development boards	USB cable
Your own board	FTDI x232H programming kit. For example: <ul style="list-style-type: none"> <li>• FT2232H Mini Module</li> <li>• FT4232H Mini Module</li> </ul>



**Note:** The FTDI Chip Mini Module supports 3.3 V I/O voltage only. Refer to the [FTDI Chip website](#) for more information about the modules.

## Installing the Linux USB Driver

The following instructions explain how to install a USB driver for Linux operating systems.

1. Disconnect your board from your computer.
2. In a terminal, use these commands:

```
> sudo <installation directory>/bin/install_usb_driver.sh
> sudo udevadm control --reload-rules
> sudo udevadm trigger
```



**Note:** If your board was connected to your computer before you executed these commands, you need to disconnect it, then re-connect it.

## Installing the Windows USB Driver

On Windows, you use software from Zadig to install drivers. Download the Zadig software (version 2.7 or later) from [zadig.akeo.ie](http://zadig.akeo.ie). (You do not need to install it; simply run the downloaded executable.)



**Important:** For some Efinix development boards, Windows automatically installs drivers for some interfaces when you connect the board to your computer. You do not need to install another driver for these interfaces. Refer to the user guide for your development board for specific driver installation requirements.

To install the driver:

1. Connect the board to your computer with the appropriate cable and power it up.
2. Run the Zadig software.



**Note:** To ensure that the USB driver is persistent across user sessions, run the Zadig software as administrator.

3. Choose **Options > List All Devices**.
4. Repeat the following steps for each interface. The interface names end with (*Interface N*), where *N* is the channel number.
  - Select **libusb-win32** in the **Driver** drop-down list.
  - Click **Replace Driver**.
5. Close the Zadig software.



**Note:** This section describes how to install the libusb-win32 driver for each interface separately. If you have previously installed a composite driver or installed using libusbK drivers, you do not need to update or reinstall the driver. They should continue to work correctly.

## Appendix: Programming the Flash Using JTAG Bridge (Legacy)

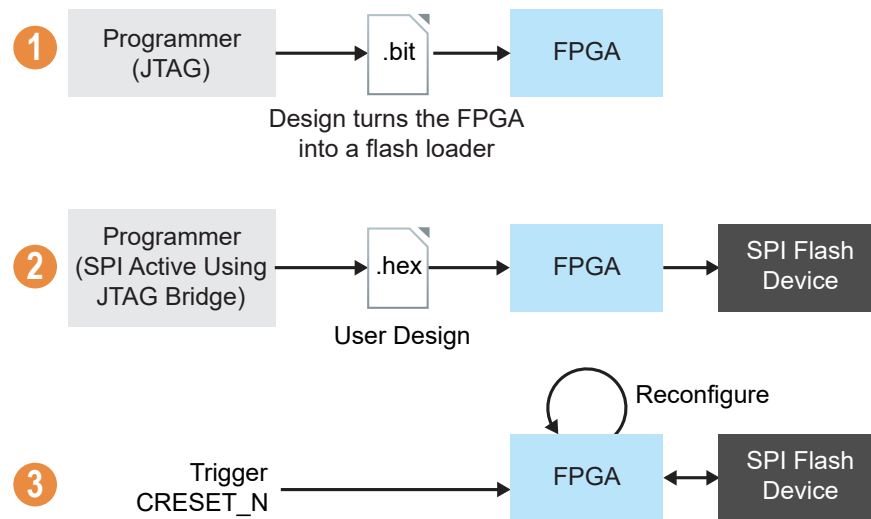
JTAG Bridge (Legacy) and JTAG SPI Flash Loader are EOL. The section has been kept to provide legacy support. For new designs, Efinix recommends using JTAG Bridge (New).

You can use the JTAG SPI Flash Loader to load a new user image into the SPI flash device on your board. The Trion® FPGA bridges the JTAG commands sent from the computer to the flash device. This mode lets you save board space because you can use the JTAG header on your board to program the flash instead of using a separate SPI header.

The flash programming flow consists of these steps:

1. Turn the Trion® FPGA into a flash programmer by configuring the FPGA via JTAG with the JTAG SPI Flash Loader IP core. You can configure the IP core using the Efinity IP Manager. You use a **.bit** bitstream file to configure the FPGA.
2. Use the Efinity Programmer and the **SPI Active using JTAG Bridge** mode to program the user image into the flash device. The Programmer sends the command through the Trion® FPGA, which in turn programs the flash. You use a **.hex** bitstream file for the user image.
3. After the flash is programmed, toggle the Trion® FPGA's **CRESET\_N** signal to trigger reconfiguration using the new flash image.

Figure 53: SPI Flash Programming Flow



When using this mode, you need to connect the JTAG pins. Refer to the diagrams in [Connecting a JTAG Mini Module](#) and [JTAG Programming Connections](#) on page 56 for the pins to connect.



**Learn more:** For more information on using the JTAG SPI Flash Loader and the **SPI Active using JTAG Bridge** programming mode, refer to the [JTAG SPI Flash Loader Core User Guide](#).

# Revision History

**Table 40: Revision History**

Date	Version	Description
November 2025	6.6	<p>Added Supported Download Cables section. (DOC-1303)</p> <p>Added IS25WP512M for ISSI in Supported Flash Devices.</p> <p>Corrected and Updated Verify Configuration topic including sub-topic Monitoring with a Microcontroller or LEDs.</p> <p>Added note about padding in combined bitstream images in <b>Combine Bitstreams and Other Files</b> on page 63.</p>
March 2025	6.5	<p>Added <b>Figure 5: SPI Active (x1) Configuration</b> on page 16.</p> <p>Added <b>Figure 6: SPI Active (x1) Configuration (Detailed View)</b> on page 16.</p> <p>Added <b>Figure 14: SPI Passive (x1, Mode 3) Configuration</b> on page 25.</p> <p>Added <b>Figure 15: SPI Passive (x1, Mode 3) Configuration (Detailed View)</b> on page 25.</p> <p>Update <b>Figure 22: JTAG Programming Waveform (T4, T8, T13, T20W80, T20Q100F3, T20Q144, T20F256, and T20F169 FPGAs)</b> on page 32.</p> <p>Update <b>Figure 23: JTAG Programming Waveform (T20F324, T20F400, T35, T55, T85, and T120 FPGAs)</b> on page 32.</p> <p>Added <b>Figure 24: JTAG Programming (All Trion FPGAs) (Detailed View)</b> on page 33.</p> <p>Fix typo in <b>Figure 31: Flash Programming Board Setup</b> on page 39.</p> <p>Added note to <b>Program Using a JTAG Chain</b> on page 68. (DOC-2310)</p>
February 2025	6.4	<p>Corrected timing waveforms. (DOC-2325)</p> <p>Move information about unused resources to <b>Unused Resources and Features</b> on page 44.</p>
January 2025	6.3	<p>Fixed typo in title of referenced document. (DOC-2302)</p>

Date	Version	Description
December 2024	6.2	<p>Added <b>Table 19: Status Register Protect Bits</b> on page 40. (DOC-2254)</p> <p>Updated <b>Figure 10: Passive (x1)</b> on page 21 and added important note. (DOC-2076)</p> <p>Added <b>Figure 11: Supported SPI Waveform</b> on page 22. (DOC-2176)</p> <p>Added <b>Figure 12: Unsupported SPI Waveform</b> on page 22. (DOC-2176)</p> <p>Added <b>SPI Passive Mode for SIP Packages</b> on page 27. (DOC-2077)</p> <p>Updated <b>Table 3: Dual-Purpose Configuration Pins</b> on page 8. (DOC-2077)</p> <p>Added <b>Figure 7: Connections between FPGA and SPI Flash Inside the QFP100F3 Package</b> on page 17. (DOC-2077)</p> <p>Added <b>Figure 9: Configuration with External Flash</b> on page 19. (DOC-2077)</p> <p>Added <b>Figure 8: Configuration with Internal Flash</b> on page 18. (DOC-2077)</p> <p>Updated <b>Figure 51: Connect CDONE and NSTATUS to a Microcontroller</b> on page 78 and <b>Figure 52: Connect CDONE and NSTATUS to LEDs</b> on page 79. (DOC-2165)</p> <p>Notes added to <b>SPI Active Mode</b> on page 12. (DOC-2046)</p> <p>Notes added to <b>SPI Passive Mode</b> on page 21. (DOC-2046)</p> <p>Notes added to <b>JTAG Mode</b> on page 30. (DOC-2046)</p> <p>Updated entry for SS_N configuration function in <b>Table 3: Dual-Purpose Configuration Pins</b> on page 8. (DOC-2077)</p> <p>Updated <b>Figure 31: Flash Programming Board Setup</b> on page 39. (DOC-1408)</p> <p>Fixed typo in <b>Table 3: Dual-Purpose Configuration Pins</b> on page 8. (DOC-2038)</p> <p>Changed column name from Pins to Configuration Functions in <b>Table 3: Dual-Purpose Configuration Pins</b> on page 8. (DOC-2038)</p> <p>Added note after <b>Table 3: Dual-Purpose Configuration Pins</b> on page 8 directing the reader to <code>device_pinout.xlsx</code>. (DOC-2038)</p>
August 2024	6.1	<p>Updated <b>Design Considerations</b> on page 33 for <code>CRESET_N</code> and <code>SS_N</code> requirements when using JTAG bridge programming. (DOC-2011)</p>
February 2024	6.0	<p>Updated figures Flash Programming Board Setup and SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections. (DOC-1256)</p> <p>Updated on table Dedicated Configuration Pins and Dual-Purpose Configuration Pins. (DOC-1490)</p> <p>Updated SPI Flash Programming with FTDI Mini Module Connections. (DOC-1497)</p> <p>Updated note about multiple FTDI connection to mention the supported FPGAs. (DOC-1512)</p> <p>Updated notes in figure SPI Passive Mode (x1) Timing Sequence. (DOC-1690)</p> <p>Updated the waveform for figures SPI Active Mode (x1) Timing Sequence, SPI Passive Mode (x1, Mode 3) Timing Sequence, and JTAG Programming Waveform.</p>
September 2023	5.9	<p>Updated CCK pin description. (DOC-1451)</p>
June 2023	5.8	<p>Updated Internal Flash Image option to Remote Update Flash Image in Programmer. (DOC-1302)</p> <p>Trion FPGAs only support SPI Active Using JTAG Bridge. (DOC-1319)</p>
May 2023	5.7	<p>Added IS25LP128 to list of supported flash devices. (DOC-1247)</p>

Date	Version	Description
April 2023	5.6	<p>Added information about QFP100F3 packages and SPI active configuration for SIP packages. (DOC-1188)</p> <p>Updated JTAG configuration design considerations. (DOC-994)</p>
February 2023	5.5	<p>Added more description about valid and invalid image. (DOC-1118)</p> <p>Corrected user-defined pull-up/pull-down resistor formula. (DOC-1136)</p> <p>Updated power up sequence diagram. (DOC-954)</p>
December 2022	5.4	<p>Corrected SPI Clock Polarity and Phase Mode table. (DOC-946)</p> <p>Added note about not recommending user to pause FPGA configuration. (DOC-944)</p> <p>Added description about <math>\overline{\text{CRESET\_N}}</math> needs to be deasserted before JTAG configuration begins. (DOC-1069)</p>
September 2022	5.3	<p>Updated Verifying Configuration topic.</p> <p><b>Enable CRC Check</b> option removed from the <b>Project Editor</b> (always on with Efinity v2022.1 and higher). (DOC-912)</p> <p>Removed support for C232HM-DDHSL-0 cable. (DOC-860)</p> <p>Removed JTAG Device ID for BGA49 packages. (DOC-899)</p> <p>Added note about Trion only supports SPI flash with 3-byte addressing mode for configuration. (DOC-910)</p> <p>Updated supported flash devices. (DOC-896)</p> <p>Updated Project-Based Programming Options topic for new options.</p>
August 2022	5.2	<p>Defined VCC is 1.2 V in the Connection Requirements for Unused Resources table. (DOC-770)</p> <p>Corrected SPI active x1 SDI to CDI0 example connection. (DOC-783)</p> <p>Added topic on SPI clocking and sampling. (DOC-625)</p> <p>Corrected SS_N connection to be bidirectional in active SPI mode connection diagrams.</p> <p>Updated configuration flow diagram.</p> <p>Added JTAG USER TAP instructions.</p>
April 2022	5.1	<p>Added user-defined pull-down resistance formula. (DOC-747)</p> <p>Added Program using a JTAG Bridge topic.</p> <p>Added topic on combining a bitstream and other data into a single file for programming.</p> <p>Re-organized topics about working with bitstreams.</p>
March 2022	5.0	<p>Moved FTDI hardware connection diagrams into Programming Hardware Connections topic.</p> <p>Added topic about external pull-up resistors. (WEB-39)</p> <p>Removed optional pull-up resistors from SPI active circuitry diagrams.</p> <p>Updated power up sequence stating that all supplies must be powered up within 10 ms. (DOC-631)</p>

Date	Version	Description
January 2022	4.9	<p>Improved connection diagrams to show pull-ups to point upwards. (DOC-612)</p> <p>Added Bitstream Bytes Packed into Parallel Bus for x16, x8, x4, x2, and x1. (DOC-626)</p> <p>Added reference to AN 035: SPI Passive Programming with Raspberry Pi. (DOC-569)</p> <p>With the Efinity software v2021.2 and higher, you <b>must</b> use <b>.hex</b> for SPI and <b>.bit</b> for JTAG. (DOC-638)</p> <p>Added Exporting to .svf Format topic. (DOC-569)</p> <p>Corrected Passive (x2) without CSI or CBUS2 figure.</p> <p>Added note about if the flash device does not have a valid image in the location the FPGA expects based on the CBSEL setting, the FPGA looks at the image locations in ascending order until it finds a valid image. (DOC-686)</p> <p>Updated active mode connection diagram.</p>
November 2021	4.8	Corrected the power-up sequence waveform.
November 2021	4.7	<p>Updated JTAG mode connection diagram. (DOC-546)</p> <p>Updated the Project-Based Programming Options topic. (DOC-550)</p> <p>Added Macronix MX75L and MX75U to supported flash devices. (DOC-573)</p> <p>Updated note about not driving any Trion® I/O pins before the Trion® FPGA is powered up. (DOC-587)</p> <p>Added support for FTDI FT4232H Mini Module. (DOC-597)</p>
September 2021	4.6	<p>Updated the Project-Based Programming Options topic. (DOC-523)</p> <p>Added XT25F family to list of supported flash devices. (DOC-529)</p> <p>Minor text corrections to the Programming the Flash Using a JTAG Bridge topic. (DOC-543)</p> <p>Updated Verifying Configuration topic. (DOC-539, DOC-486)</p> <p>Removed x4 from Passive (x2) without CSI or CBUS2 figure as the figure is showing x1 CBUS settings.</p>
August 2021	4.5	<p>Updated FPGA configuration mode topic.</p> <p>Added flash programming mode topic.</p> <p>Added topic on verifying configuration. (DOC-508)</p> <p>Corrected FPGA pin names for the SPI passive FTDI FT2232H Mini Module Connections figure.</p> <p>Updated for T20QFP144 package. (DOC-519)</p> <p>Added note about not connecting the <code>NSSTATUS</code> pins of multiple FPGAs in daisy chain configuration. (DOC-518)</p> <p>Added note about FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495)</p> <p>Added note about using DIV4 in x2 and x4 parallel daisy chain configuration. (DOC-525)</p>
July 2021	4.4	<p>Described more detail on the Enable Initialized Memory in User RAMs option in the <b>Project Editor &gt; Bitstream Generation</b> tab. (DOC-458)</p> <p>Updated the Windows USB driver installation topic.</p> <p>Updated the FTDI command-line programming topic. Added the command-line programmer configuration mode options. (DOC-430)</p> <p>Corrected SPI flash daisy chain configuration example figures. (DOC-483)</p> <p>Updated CDIn pin description. (DOC-483)</p>

Date	Version	Description
May 2021	4.3	Added Macronix MX25U and Micron M25P16 to list of supported flash devices. Added the SENSE pin to <a href="#">Figure 34: Supervisor IC Power Up Circuitry</a> on page 43. Removed T120S content. (DOC-445) Updated CRESET_N pin description. (DOC-450)
March 2021	4.2	SPI Active using JTAG Bridge mode only requires the 4 JTAG pins. (DOC-404) For Windows, plug in the board and power it up before installing USB drivers. (DOC-415)
February 2021	4.1	Updated Power Up Sequence topic. (DOC-396) Added bitstream length for T20W80. (DOC-393)
February 2021	4.0	Added note stating a circuitry is needed to control CRESET_N pin to meet timing requirement for SPI active mode. (DOC-380)
December 2020	3.9	Updated NSTATUS pin description. (DOC-335) Added MX25V to list of supported flash devices. Corrected JTAG Mini Module pin names for T4, T8, T13, T20BGA256, and T20BGA169 connection setup. (DOC-348) Removed instance pin connection notes and replaced with a table in the Power Up Sequence topic.
November 2020	3.8	Updated Added note to power-up sequence and DDR about DDR_VREF and VCCIO_DDR connection when not using DDR. (DOC-325) Updated About USB Drivers and subtopics to support separate interfaces driver installation and FTDI4232H. (DOC-334)
October 2020	3.7	Corrected serial and parallel daisy chain configuration interface example figures. (DOC-135)
August 2020	3.6	Corrected maximum power supply current transient table.
August 2020	3.5	Clarified settings for CBSEL. Added missing CBUS2 signal in SPI active mode schematics. Removed T165 and T200 device information.
July 2020	3.4	Updated timing parameter symbols in boundary scan timing waveform to reflect JTAG mode parameter symbols in datasheet. Added note to refer to AN021 for boundary-scan testing information. Removed Efinity Interface Designer JTAG User TAP Interface subsection and added note and link to Efinity® Software User Guide for more information about JTAG User TAP interface. Added note about sending additional 100 CCK cycles after sending the last configuration data for passive mode configuration. Added support for FTDI FT2232H module for JTAG and SPI passive programming. Corrected configuration clock signal name from CCKO to CCK. Added JTAG device IDs for T20BGA324 and T20BGA400.
April 2020	3.3	Corrected GigaDevice supported family (GD25 not GD32). Added T8 QFP144 bitstream length. Added Micron to table of supported flash devices.
February 2020	3.2	Added GigaDevice to table of supported flash devices.
February 2020	3.1	Added table of supported flash devices.

Date	Version	Description
January 2020	3.0	<p>Added schematics and information on active and passive configuration for packages that do not have the CSI or CBUS2 pins bonded out.</p> <p>Clarified usage of TEST_N pin in configuration and user modes.</p> <p>Clarified dedicated vs. dual-purpose configuration pins.</p> <p>Removed DIV1 and DIV2 SPI clock divider settings; currently they are not supported.</p> <p>Added information on programming with the SPI Active using JTAG Bridge mode.</p> <p>Added JTAG device IDs for T35, T55, T85, and T120 FPGAs.</p> <p>Added bitstream lengths for T20 (BGA324 and BGA400), T35, T55, T85, and T120 FPGAs.</p> <p>Clarified handling of multi-function configuration pins that are outputs in user mode.</p> <p>When installing USB drivers for Windows using the Zadig software, specify the <b>libusb-win32</b> driver instead of <b>libusbK</b>.</p> <p>Added a note to the Support for Multiple Images topic to refer readers to the data sheet for which modes support multiple images.</p>
September 2019	2.1	<p>Added information on JTAG chain programming.</p> <p>Added a note that adding optional header information to the bitstream file using the Efinity<sup>®</sup> software can add up to 1k bits to the bitstream length.</p> <p>Updated the maximum supported configuration bits.</p> <p>Updated CBUS[2:0] setting for SPI active mode.</p>
March 2019	2.0	<p>Added information about JTAG programming support.</p> <p>Updated information on installing the programming cable USB driver.</p> <p>Added additional information on using the Efinity<sup>®</sup> Programmer.</p>
December 2018	1.2	<p>Added sections on using the Efinity<sup>®</sup> Programmer.</p> <p>Added note for SPI passive mode that the CSI pin can also be connected to VCCIO.</p> <p>Fixed incorrect equation for calculating bitstream length.</p> <p>Changed TCK to have a recommended weak pull down.</p> <p>Fixed typo in <b>Daisy Chaining with a SPI Flash Device</b> on page 48, Parallel Daisy Chain Configuration (x4) Interface Example.</p>
August 2018	1.1	<p>Added section on choosing an SPI flash device.</p>
June 2018	1.0	<p>Initial release.</p>