AN 006: Configuring Trion® FPGAs

AN006-v5.9
September 2023
www.efinixinc.com

Contents

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:

Configuration time = bitstream size / (configuration clock frequency * 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 Mbits * 100 ns / 8 = 17.25 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.

Efinix® 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.

Pins	Direction	Description	Use External Weak Pull-Up
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. If you hold CDONE low, the device will not enter user mode.	✓
CRESET_N	Input	Initiates FPGA re-configuration (active low). Pulse CRESET_N low for a duration of t_{creset_N} before asserting CRESET_N from low to high to initiate FPGA re-configuration. This pin does not perform a system reset.	✓
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.	✓
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.	Recommended
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.	Recommended
TDO	Output	JTAG test data output (TDO). This serial output from the test logic is fed from the instruction register or from a test data register depending on the sequence previously applied at TMS. During shifting, data applied at TDI appears at TDO after a number of cycles of TCK determined by the length of the register included in the serial path. 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).	✓

Table 3: Dual-Purpose Configuration Pins

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

Pins	Direction	Description	Use External Weak Pull-Up
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 according to the Table 5: SPI Hardware Settings on page 10.	✓[1]
CBSEL[1:0]	Input	Optional multi-image selection input (if external multi-image configuration mode is enabled).	✓[2]
CCK	I/O	Passive SPI input configuration clock or active SPI output configuration clock (active low). Includes an internal weak pull-up. **Important:** 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[3]
CDI*n*	I/O	*n* 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. *n*: Parallel I/O. In multi-bit daisy chain connection, the CDI*n* (31:0) connects to the data bus in parallel.	Optional[3]
CSI	Input	Chip select. 0: The FPGA is not selected or enabled and will not be configured. 1: Selects the FPGA for configuration (SPI and JTAG configuration).	✓
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 an ID mismatch, the bitstream CRC check has failed, or remote update has failed. For Trion® T4, T8 BGA49, and T8 BGA81 FPGAs, logic low indicates a configuration error due to ID mismatch.	N/A
SS_N	Input	SPI slave select (active low). Includes an internal weak pull-up resistor to VCCIO during configuration. During configuration, the logic level samples on this pin determine the configuration mode. This pin is an input when sampled at the start of configuration (SS is low); an output in active SPI flash configuration mode. The FPGA senses the value of SS_N when it comes out of reset (pulse CRESET_N low to high). 0: Passive mode 1: Active mode	Optional[3]
TEST_N	Input	Active-low test mode enable signal. Set to 1 to disable test mode. During configuration, rely on the external weak pull-up or drive this pin high.	✓

[1] Optional for x1 mode.
[2] Not applicable to single-image or remote update.
[3] Optional unless pull-up is required by external load.

Pins	Direction	Description	Use External Weak Pull-Up
RESERVED_OUT	Output	Reserved pin during user configuration. This pin drives high during user configuration. BGA49 and BGA81 packages only.	N/A

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	TEST_N	SS_N	CBUS2, CBUS1, CBUS0	Width
SPI Active	Serial	1	1	3'b111	x1
	Parallel	1	1	3'b110	x2
	Parallel	1	1	3'b101	x4
SPI Passive	Serial	1	0	3'b111	x1
	Parallel	1	0	3'b110	x2
	Parallel	1	0	3'b101	x4
	Parallel	1	0	3'b100	x8
	Parallel	1	0	3'b011	x16
	Parallel	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 4 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

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.[4]

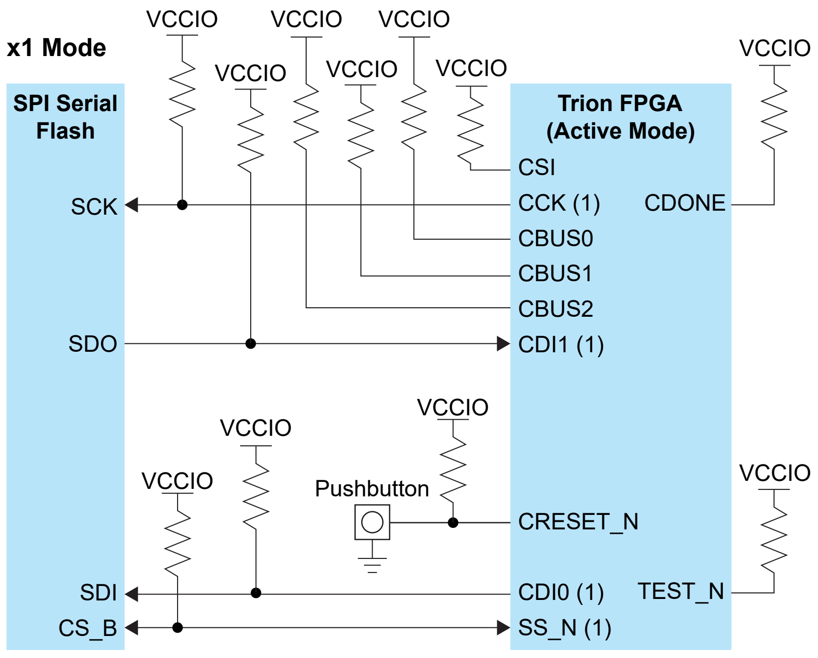
[4] Smaller packages may not have `CBUS2` bonded out. In this case, `CBUS2` is held high in the package.

Connection Examples

Note: A circuitry is required to control the `CRESET_N` pin to meet the t_{CRESET_N} requirement.

Figure 1: Active (x1)

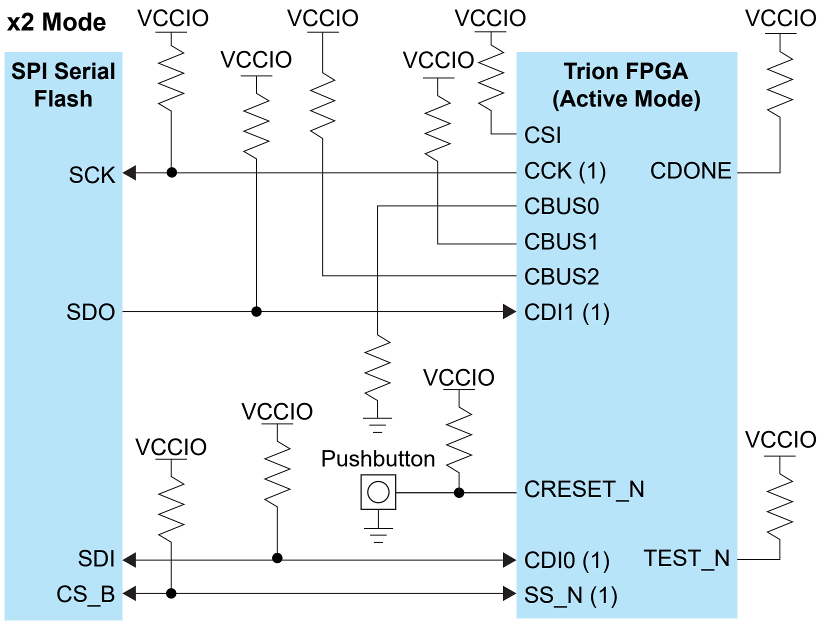
See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



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

Figure 2: Active (x2)

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.

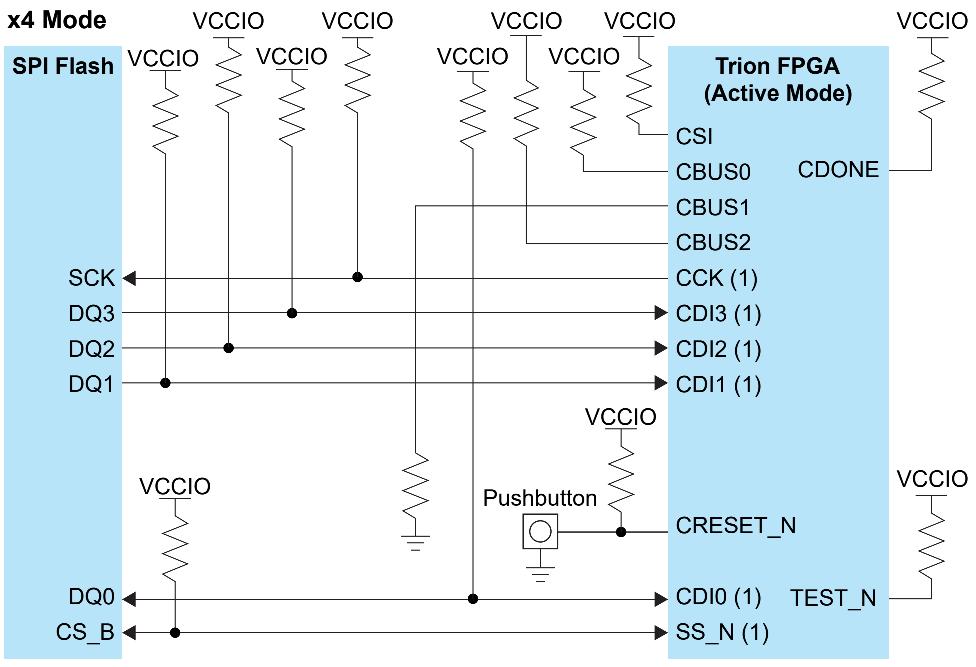


Note:
1. The external pull-up is optional unless required by an external load.

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 3: Active Quad (x4)

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



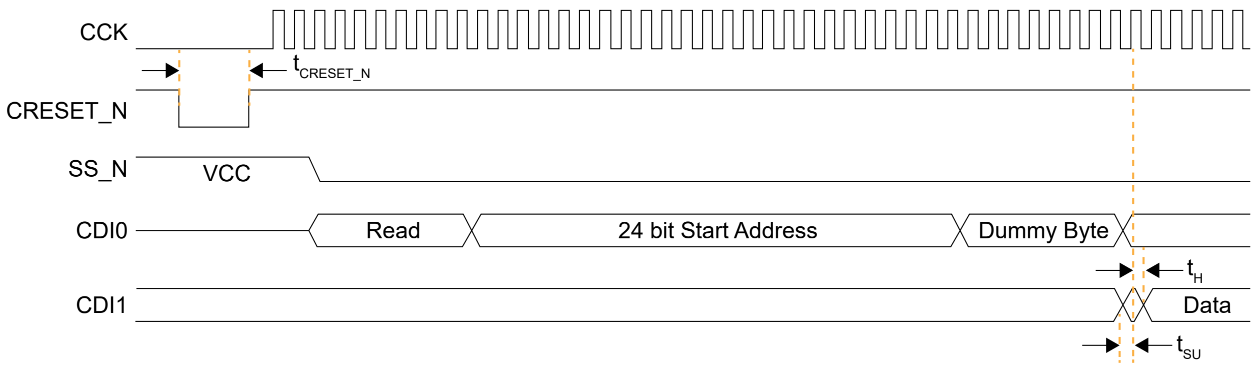
Note:
1. The external pull-up is optional unless required by an external load.

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 µs.
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 external the SPI flash to enter deep power-down state.

Figure 4: SPI Active Mode (x1) Timing Sequence



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

SPI Active Mode for SIP Packages

Trion® FPGAs in QFP100F3 packages are a system-in-package (SIP) that includes an internal SPI flash which 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:

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 enable or disable the FPGA from another host such as a microprocessor. Additionally, you cannot 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.

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.

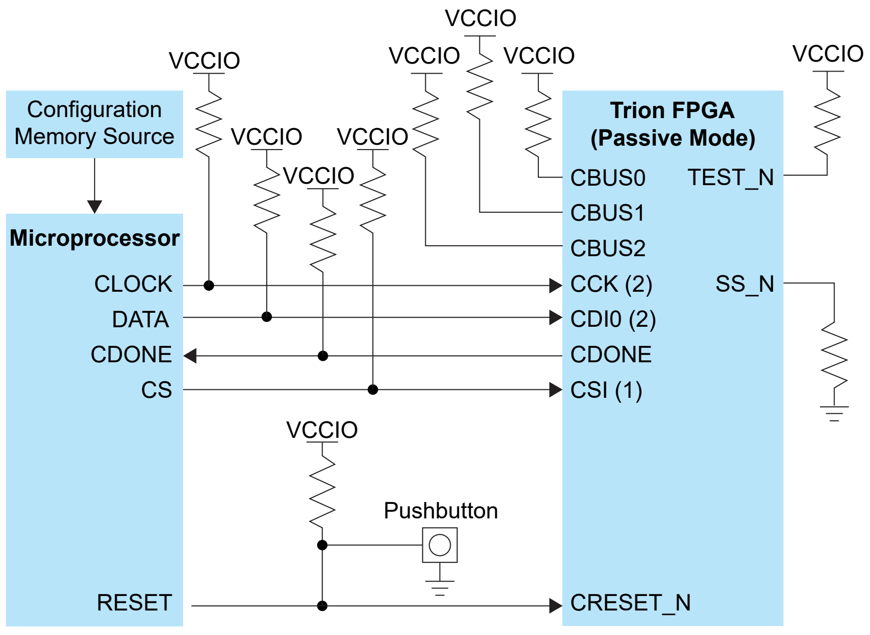
Note: Efinix recommends that you use the same VCCIO on the banks of all configuration pins.

Connection Examples

These examples show SPI passive x1 and x32 modes. For other modes, set the value of `CBUS[2:0]` according to Table 5: SPI Hardware Settings on page 10.

Figure 5: Passive (x1)

See Resistors in Configuration Circuitry on page 37 for the resistor values.



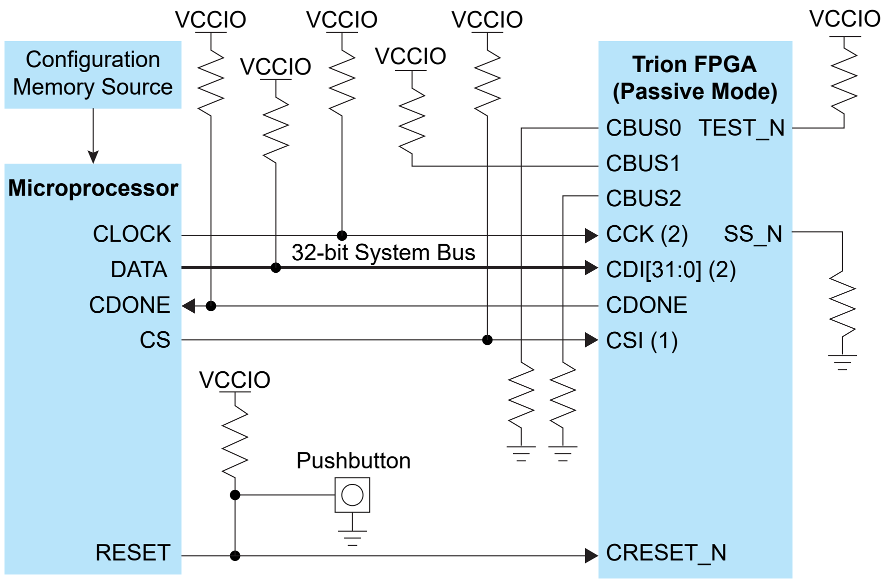
Note:
1. You can connect the CSI pin to VCCIO as well as driving it from the microprocessor.
2. The external pull-up is optional unless required by an external load.

Table 11: Bitstream Bits in Series

Bus Bit	0
Cycle 1	Bit 7
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
Cycle 9	Bit 15
Cycle 10	Bit 14
Cycle 11	Bit 13
Cycle 12	Bit 12
Cycle 13	Bit 11
Cycle 14	Bit 10
Cycle 15	Bit 9
Cycle 16	Bit 8

Figure 6: Passive (x32)

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



Note:
1. You can connect the CSI pin to VCCIO as well as driving it from the microprocessor.
2. The external pull-up is optional unless required by an external load.

Table 12: Bitstream Bytes Packed into 32 bit Parallel Bus

Bus Bit	**31**		**24**	**23**		**16**	**15**		**8**	**7**		**0**
Cycle 1		Byte 0			Byte 1			Byte 2			Byte 3	
Cycle 2		Byte 4			Byte 5			Byte 6			Byte 7	

Table 13: Bitstream Bytes Packed into 16 bit Parallel Bus

Bus Bit	**15**		**8**	**7**		**0**
Cycle 1		Byte 0			Byte 1	
Cycle 2		Byte 2			Byte 3	
Cycle 3		Byte 4			Byte 5	
Cycle 4		Byte 6			Byte 7	

Table 14: Bitstream Bytes Packed into 8 bit Parallel Bus

Bus Bit	**7**		**0**
Cycle 1		Byte 0	
Cycle 2		Byte 1	
Cycle 3		Byte 2	
Cycle 4		Byte 3	

Table 15: Bitstream Bits Packed into 4 bit Parallel Bus

Bus Bit	**3**	**2**	**1**	**0**
Cycle 1	Bit 7	Bit 6	Bit 5	Bit 4
Cycle 2	Bit 3	Bit 2	Bit 1	Bit 0
Cycle 3	Bit 15	Bit 14	Bit 13	Bit 12
Cycle 4	Bit 11	Bit 10	Bit 9	Bit 8

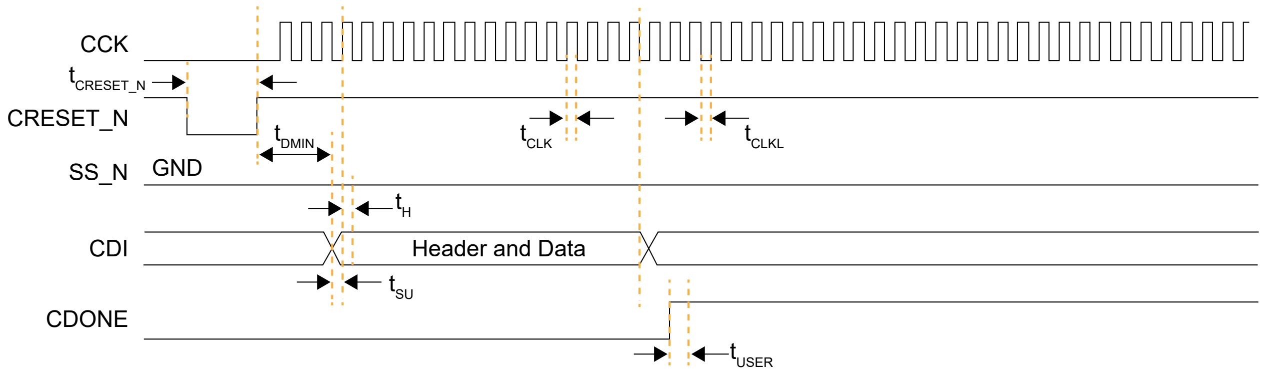
Table 16: Bitstream Bits Packed into 2 bit Parallel Bus

Bus Bit	**1**	**0**
Cycle 1	Bit 7	Bit 6
Cycle 2	Bit 5	Bit 4
Cycle 3	Bit 3	Bit 2
Cycle 4	Bit 1	Bit 0
Cycle 5	Bit 15	Bit 14
Cycle 6	Bit 13	Bit 12
Cycle 7	Bit 11	Bit 10
Cycle 8	Bit 9	Bit 8

Timing

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

Figure 7: SPI Passive Mode (x1) Timing Sequence



Note: Efinix does not recommend you to pause the Trion® FPGA configuration.

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

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

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 enable or disable the FPGA from another host such as a microprocessor. Additionally, you cannot 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 8: Passive (x1) without CSI or CBUS2

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.

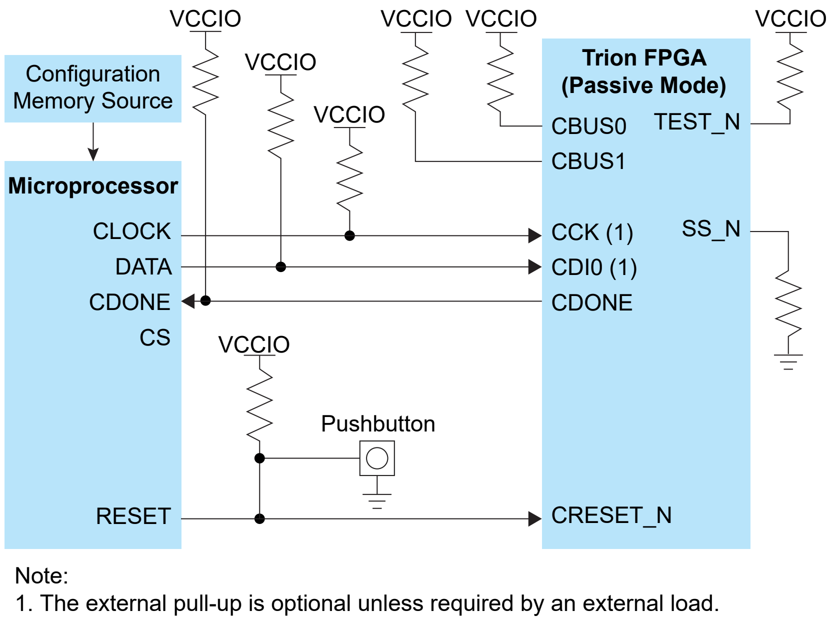
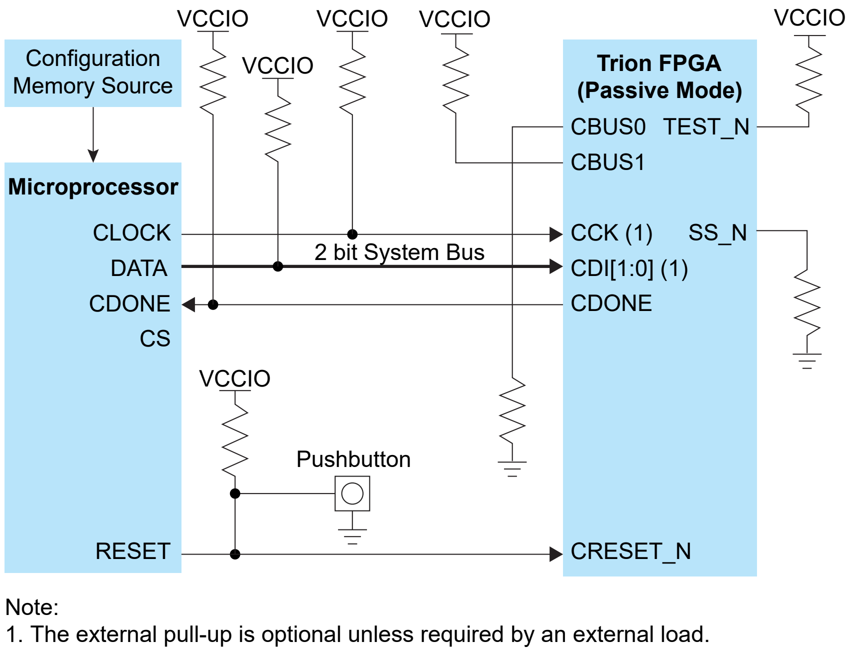


Figure 9: Passive (x2) without CSI or CBUS2

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



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 in 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 JTAG Instructions

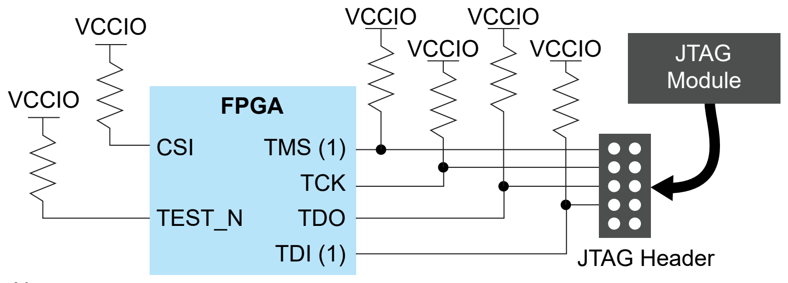
Instruction	Binary Code [3:0]	Description
SAMPLE/PRELOAD	0010	Enables the boundary-scan SAMPLE/PRELOAD operation
EXTEST	0000	Enables the boundary-scan EXTEST operation
BYPASS	1111	Enables BYPASS
IDCODE	0011	Enables shifting out the IDCODE
PROGRAM	0100	JTAG configuration
ENTERUSER	0111	Changes the FPGA 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 the 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 10: JTAG Programming

See Resistors in Configuration Circuitry on page 37 for the resistor values.



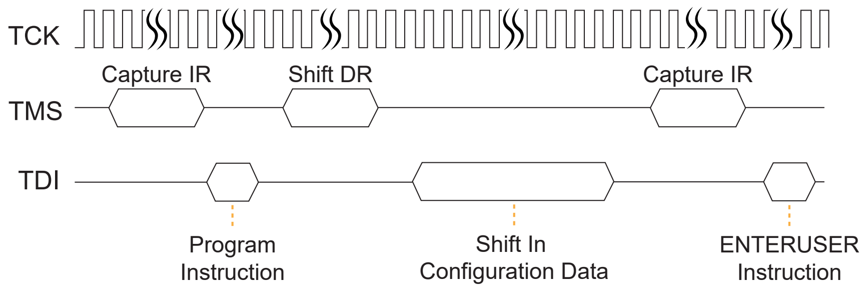
Note:
1. The external pull-up is optional unless required by an external load.

The `CRESET_N` signal needs to be deasserted before JTAG configuration begins. 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_{USER} 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.

Figure 11: JTAG Programming Waveform



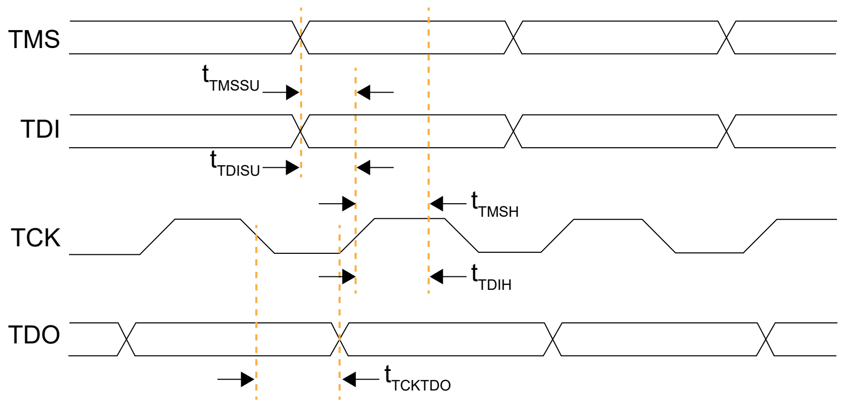
Design Considerations

- 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 diffferent 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:
  - Assertion and deassertion of CRESET_N is required 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. However, they are not required when performing SPI flash programming through the JTAG bridge, or when performing other non-configuration JTAG functions.

Learn more: Refer to **JTAG Programming Connections** on page 41 for JTAG configuration connection examples and **JTAG Programming** on page 49 for details about JTAG programming using the Efinity Programmer.

Timing Parameters

Figure 12: Boundary-Scan Timing Waveform



Learn more: Refer to the FPGA data sheet for timing specifications.
Refer to the Virtual I/O Debug Core section in the **Efinity Software User Guide** for more information about JTAG User TAP interface.

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 anyway.

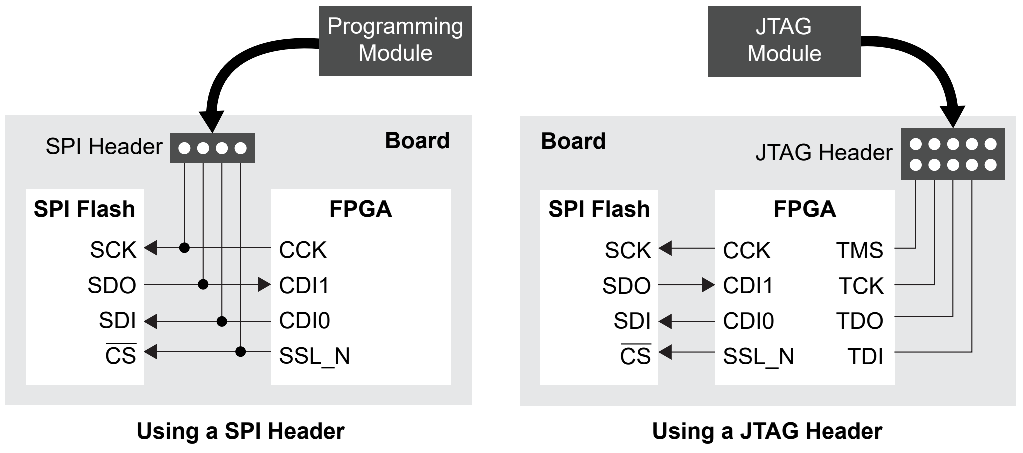
The Efinity software includes the JTAG SPI Flash Loader IP core that gives you full control over a SPI flash device and lets you perform actions comparable to an FTDI flash controller chip. With this IP core you can turn the FPGA into a flash programmer and use it to program the flash device.

Learn more: Refer to the JTAG SPI Flash Loader Core User Guide for more information.

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	Program a single flash device. First, program the FPGA with a design that turns it into a flash programmer. Then, program the flash.

Figure 13: Flash Programming Board Setup



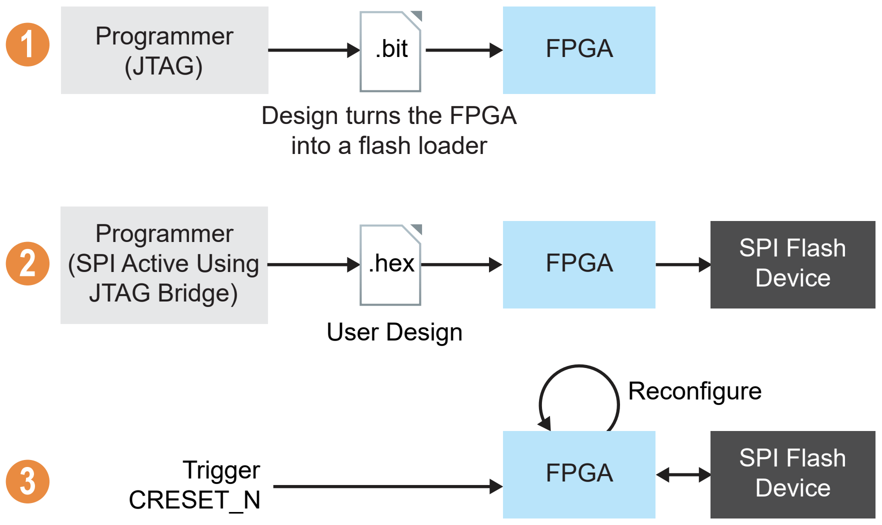
Programming the Flash Using a JTAG Bridge

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 14: 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 41 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.

Power Up

Power Up Sequence

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

Figure 15: Trion® FPGAs without MIPI Power Up Sequence

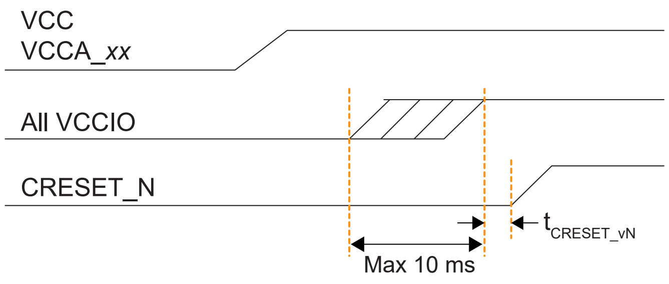
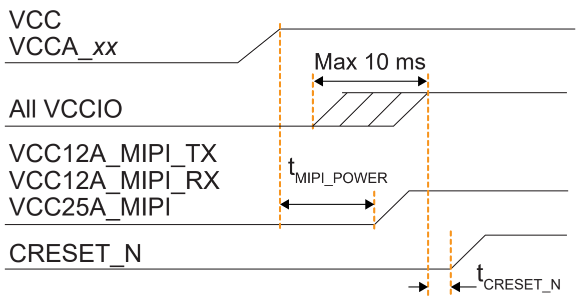


Figure 16: Trion® FPGAs with MIPI Power Up Sequence



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_{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).

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

Table 19: 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 (1.2 V).
MIPI	VCC12A_MIPI_TX	Connect to VCC (1.2 V).
	VCC12A_MIPI_RX	Connect to VCC (1.2 V).

Unused Resource	Pin	Note
	VCC25A_MIPI	Connect to VCC (1.2 V).
DDR	VCCIO_DDR	Floating. Leave unconnected.
	DDR_VREF	Connect to ground.

Learn more: Refer to Trion Hardware Design Checklist and Guidelines for connection requirements for unused resources.

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[5][6] (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

(5) Inrush current for other power rails are not significant in Trion® FPGAs.
(6) Measured at room temperature.

Power Up Circuitry Recommendation

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

- Supervisor integrated circuit (IC)
- Microprocessor or microcontroller

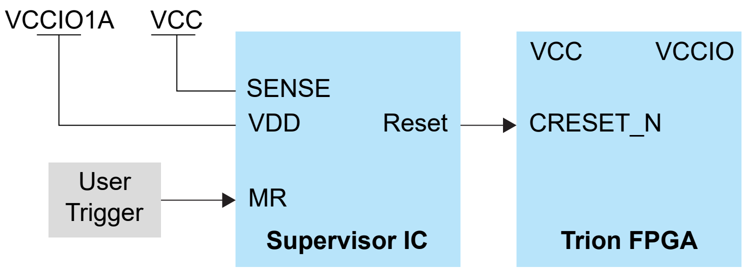
Important: Do not drive a signal to any Trion® I/O pins before the Trion® FPGA is powered up. Most FPGAs have electrostatic discharge (ESD) circuits to protect the devices from ESD events. Driving the I/O pins before VCCIO will result in an in-rush current driving the I/Os to a specific voltage through the ESD circuit to the VCCIO rail. Trion® 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 17: Supervisor IC Power Up Circuitry

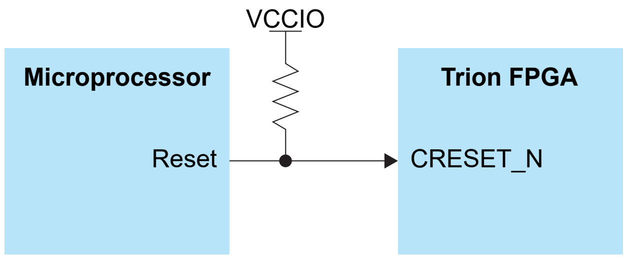


Note: The user trigger (pushbutton, FTDI module) must be connected to the `MR` pin of the supervisor IC.

Microprocessor or Microcontroller Circuitry Example

Figure 18: Microprocessor Power Up Circuitry

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



The microprocessor or microcontroller must hold the `CRESET_N` pin low more than the required t_{CRESET_N} duration.

Configuration Sequence

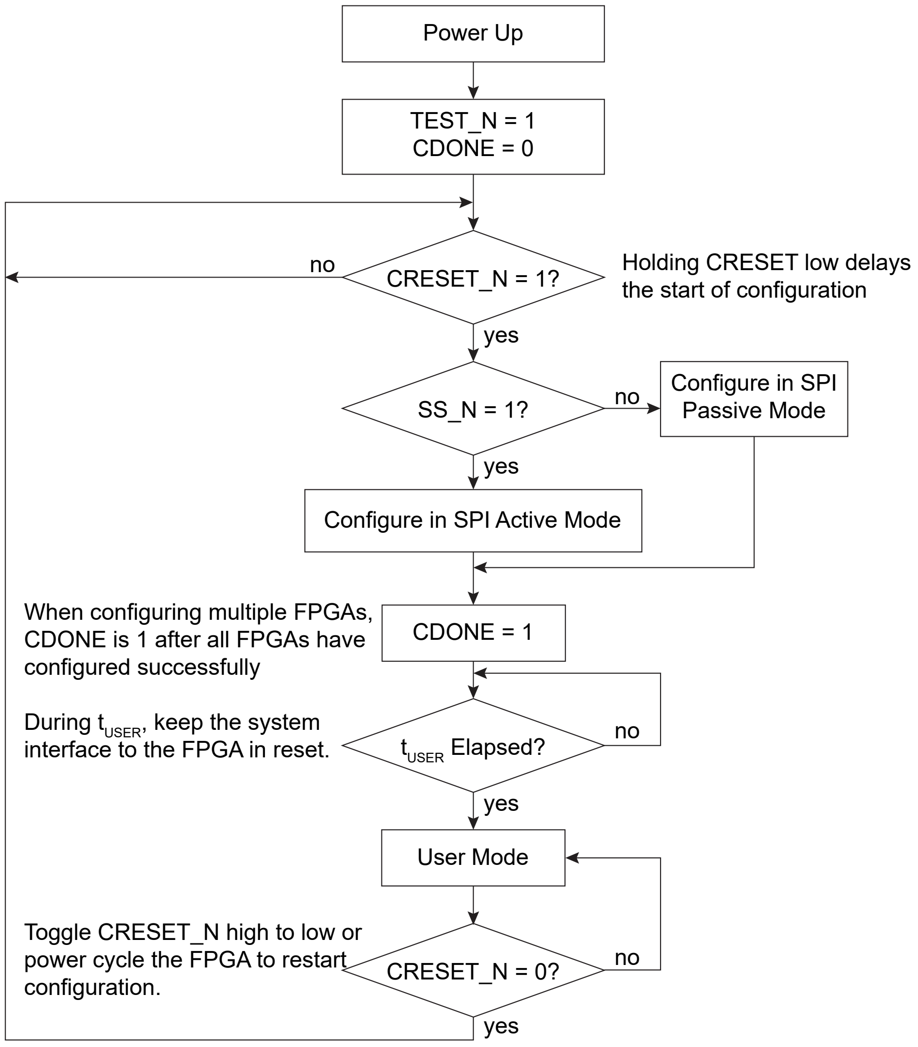
The Trion® FPGA configuration logic uses the following sequence during configuration:

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 19: Configuration Flow Diagram


Power Up
TEST_N = 1
CDONE = 0
no
CRESET_N = 1?
Holding CRESET low delays the start of configuration
yes
SS_N = 1?
no
Configure in SPI Passive Mode
yes
Configure in SPI Active Mode
When configuring multiple FPGAs, CDONE is 1 after all FPGAs have configured successfully
CDONE = 1
During t_{USER}, keep the system interface to the FPGA in reset.
t_{USER} Elapsed?
no
yes
User Mode
Toggle CRESET_N high to low or power cycle the FPGA to restart configuration.
CRESET_N = 0?
no
yes

Support for Multiple Images

When powered up in SPI active mode, the default action is for an image stored at address `0` to configure the Trion® FPGA. If you enable the multi-image feature, you can optionally choose from three other images.

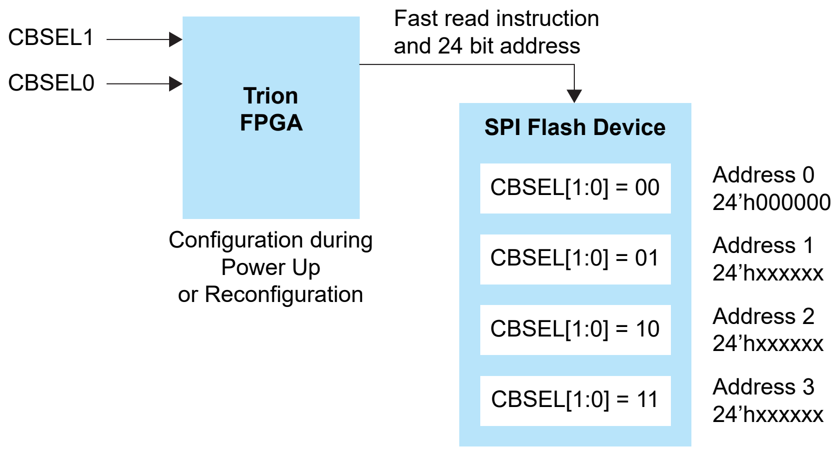
Learn more: To enable the multi-image feature, use the Efinity Programmer to combine multiple images into a single hex file. See **Program Multiple Images (CBSEL)** on page 46 for more information.

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.

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 20: Configuration Setup for Multiple Images



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 address for image `00` must be `0x0000_0000`. The following table describes valid and invalid images.

Image Details	Note
Correct synchronization pattern and CRC	Image is valid and configuration performs as expected.
Corrupted synchronization pattern	Image is invalid. The FPGA continues to search in ascending order until it locates an image with a valid synchronization pattern. The FPGA is then configured with that image.
Correct synchronization pattern but corrupted data resulting in a CRC error	Image is invalid. The FPGA configuration will fail.

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 Remotely Update Trion® and Titanium FPGAs** 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 daisy chain packages that do not have the `CSI` signal bonded out (such as the WLCSP80 and BGA169).

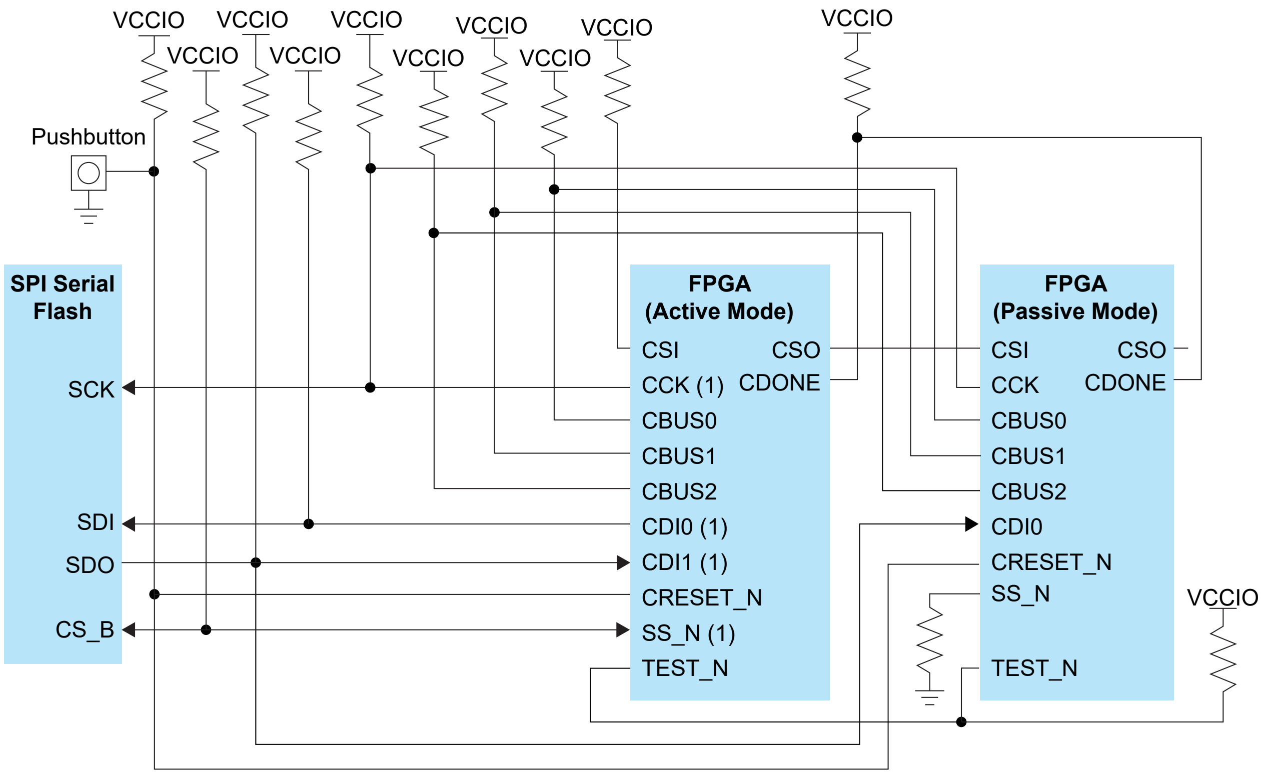
Daisy Chaining with a SPI Flash Device

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 21: Serial Daisy Chain Configuration Interface Example

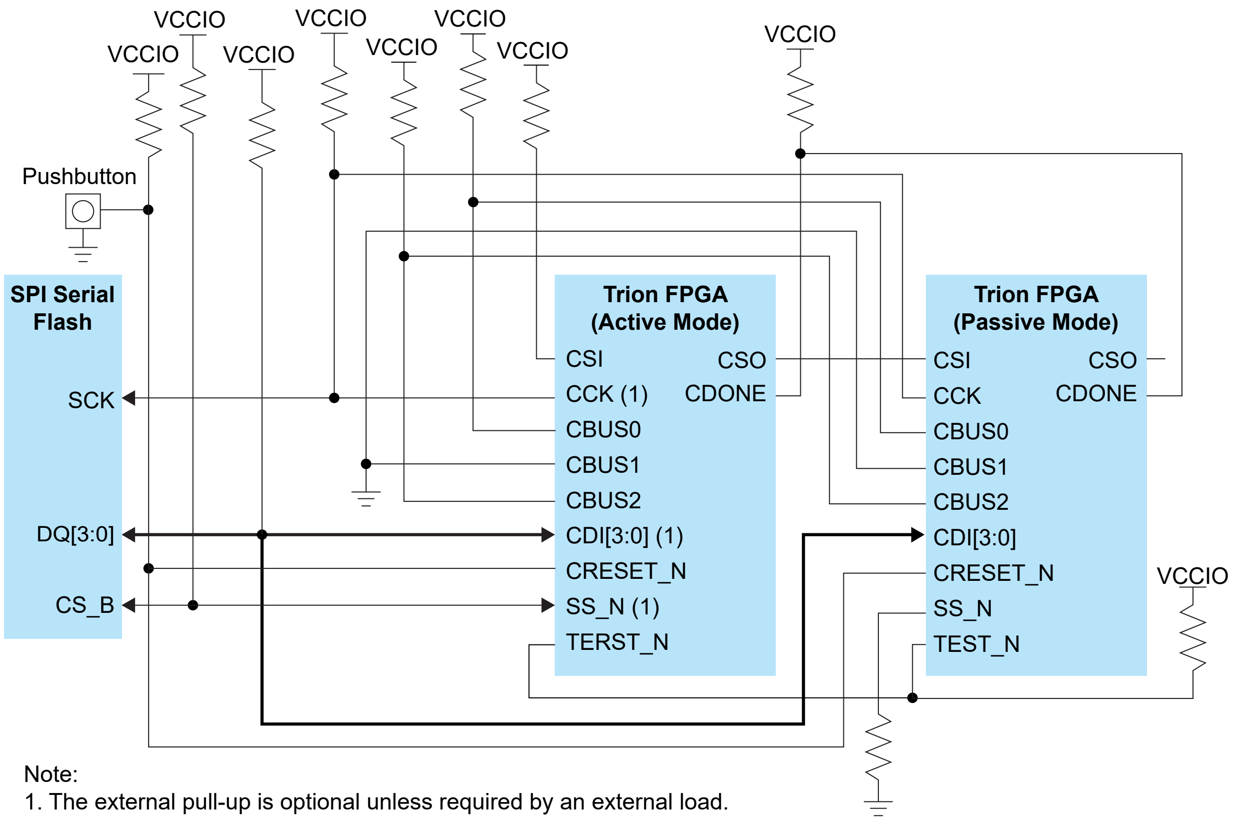
See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



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

Figure 22: Parallel Daisy Chain Configuration (x4) Interface Example

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



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

Important: For parallel daisy chain x2 and x4, you are required to set the active configuration clock frequency, f_{MAX_M}, to DIV4 (20 MHz).

Daisy Chaining with a Microcontroller or Microprocessor

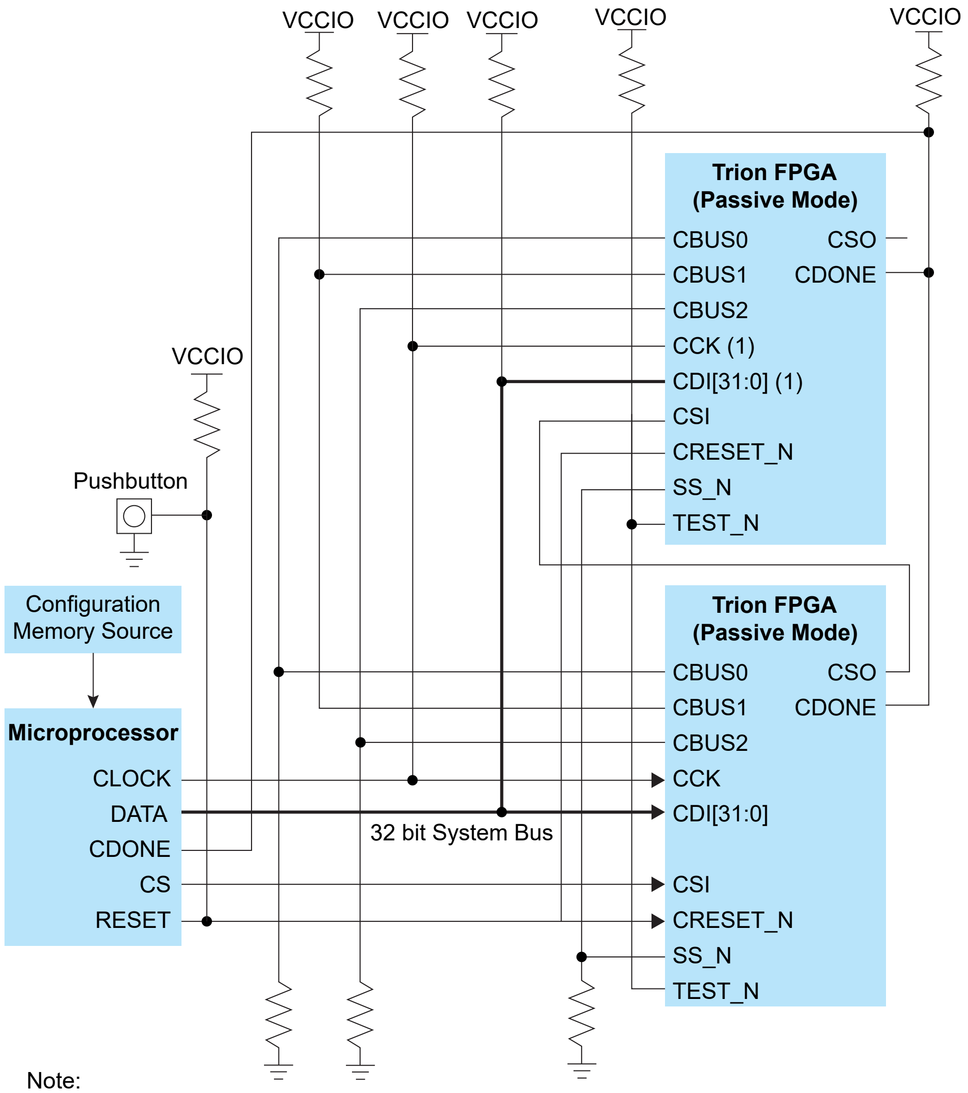
A microcontroller or microprocesser can configure FPGAs in a daisy chain with a single cascaded bitstream file. All FPGAs must be in passive mode.

This example shows serial daisy chain configuration with SPI passive x32 mode. For other modes, set the value of `CBUS[2:0]` according to **Table 5: SPI Hardware Settings** on page 10.

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

Figure 23: Serial Daisy Chain Configuration (x32 Passive Mode) Interface Example

See **Resistors in Configuration Circuitry** on page 37 for the resistor values.



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

Resistors in Configuration Circuitry

Efinix recommends that you use 10 kΩ for all unspecified pull-up and pull-down resistors in configuration circuitries. However, because of different board setups and environment noises, the configuration may fail when using 10 kΩ resistors. You can decrease the resistor values to improve signal integrity. Typically, you can also use 1 kΩ resistors.

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 Data Sheet in **Support Center**.

User-Defined Pull-Up Resistor Values

$R_{USER} = (R_{CPU} * R_{IPU}) / (R_{IPU} - R_{CPU})$

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:

$VT+ \leq VCCIO * (R_{IPD} / (R_{CPU} + R_{IPD}))$

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

$R_{USER} = (R_{CPD} * R_{IPD}) / (R_{IPD} - R_{CPD})$

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:

$VT- \geq VCCIO * (R_{CPD} / (R_{CPD} + R_{IPU}))$

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 21: All Modes

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

Table 22: Active Mode

Symbol	Parameter
f_{MAX_M}	Active mode configuration clock frequency.
t_{SU}	Setup time.
t_H	Hold time.

Table 23: Passive Mode

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

Table 24: JTAG Mode

Symbol	Parameter
f_{TCK}	TCK frequency.
t_{TDISU}	TDI setup time.
t_{TDIH}	TDI hold time.
t_{TMSSU}	TMS setup time.
t_{TMSH}	TMS hold time.
t_{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: Trion FPGA Bitstream Size on page 4). 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 x b) + (2048 x (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 x 1,386,584) + (2,048 x (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 16.
- **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 25: 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

Note: Efinix recommends using SPI NOR flash memories.

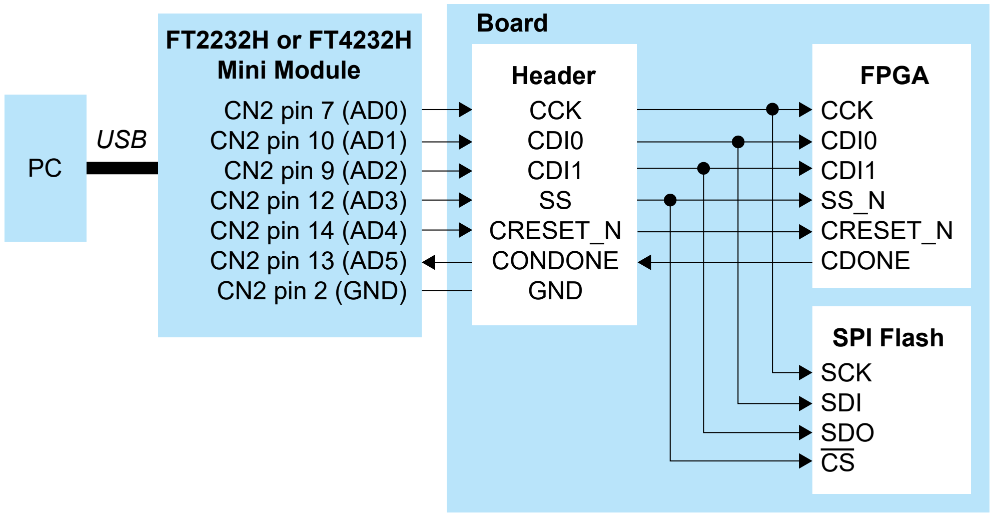
Connecting Programming Hardware

You can program Efinix FPGA or the SPI flash using FTDI Mini Modules. This section describes the hardware connections required. See **About the Programmer GUI** 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 24: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections



JTAG Programming Connections

Connecting a JTAG Cable

Efinix 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 25: Connect FT2232 Mini Module to JTAG Pins plus CRESET_N and SS_N

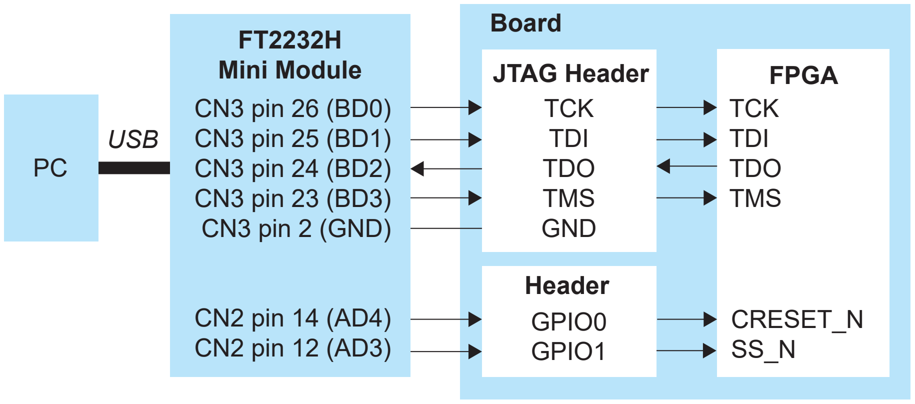
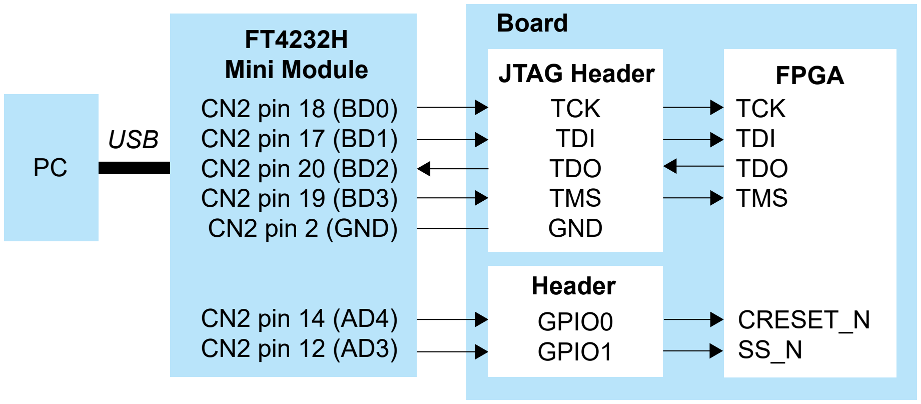


Figure 26: Connect FT4232 Mini Module to JTAG Pins plus CRESET_N and SS_N



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 27: Connect FT2232 Mini Module to JTAG Pins

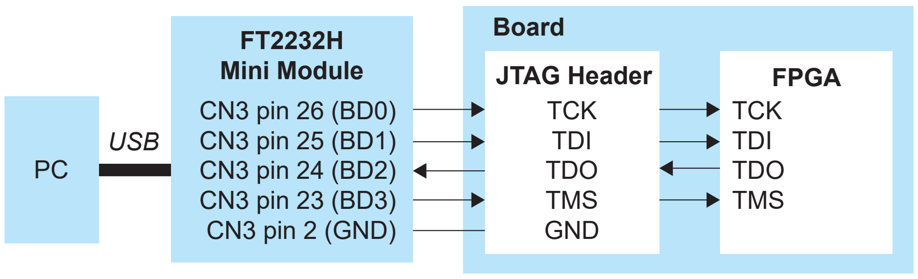
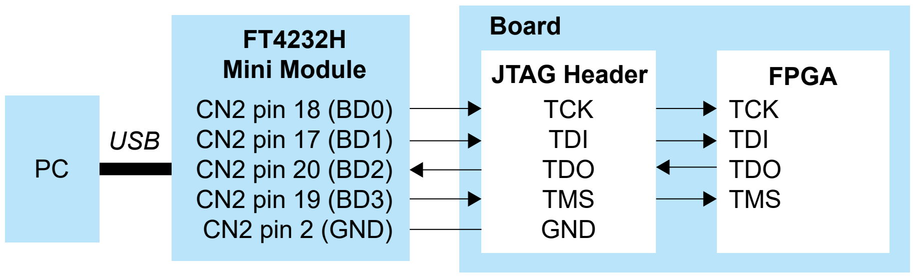


Figure 28: Connect FT4232 Mini-Module to JTAG Pins



Using the Efinity Programmer

The Efinity® software has a Programmer you use to configure Trion® 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 54.

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 45.

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.

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 45.

Convert to Intel Hex Format at the Command Line

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

```
export_bitstream.py [-h] [--family <Trion or Titanium>] [--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 or off>]
    <format> <input filename> <output filename>
```

Where <*format*> is:

- hex_to_bin
- hex_to_intelhex
- bin_to_hex
- intelhex_to_hex
- hex_to_svf

For example:

```
C:\Efinity\2021.1\bin\setup.bat
python3 C:\Efinity\2021.1\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 26: Modes when Combining Images

Mode	Use For	Number of Images	Refer to
Selectable Flash Image	Multi-image configuration	Up to 4	Program Multiple Images (CBSEL) on page 46
	Internal reconfiguration	Up to 4	Program Multiple Images (Internal Reconfiguration) on page 47
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors	Program a Daisy Chain on page 48
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	Program Multiple Images (Bitstream and Data) on page 47

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 **External Flash 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 programming multiple images, refer to Example Design: Configuring a Trion Development Board with Multiple Images on the Downloads page in the Support center.

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 **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 Remotely Update Trion® and Titanium FPGAs**.

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.

JTAG Device IDs

The following table lists the Trion JTAG device IDs.

Table 27: 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**.

Program using a JTAG Bridge

Programming with a JTAG bridge is a 2-step process: first you configure the FPGA to turn it into a JTAG programmer (**.bit**) and second you use the FPGA to program the flash device with the bitstream (**.hex**).

With the Efinity software v2021.2.323.2.18 or higher, the Programmer has an option to auto-configure the FPGAwith the **.bit** file so that you can complete both steps with a single button push.

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** mode.
4. Turn on the **Auto configure JTAG Bridge Image** option.
5. Click the **Select Image File** button to browse for the **.bit** file to configure the FPGA. The Programmer stores the file location so you do not have to specify it the next time you program.
6. Click **Start Program**. The Programmer first configures the FPGA and then programs the flash device.

Learn more: Refer to the JTAG SPI Flash Loader Core User Guide for instructions on creating the **.bit** file.

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**.

FDTI Programming at the Command Line

The Efinity® includes a script, **ftdi_program.py**, which you can use for command-line programming with FTDI modules. The command is in the format:

```
ftdi_program.py <filename>.bit -m <mode> --url <url> --aurl
<url>
```

<mode> is the programming mode (`active`, `passive`, `jtag`, `jtag_chain`, `jtag_bridge`)

Note: To use the `jtag_bridge` mode, you must have already configured the FPGA with the JTAG SPI flash loader. Refer to the JTAG SPI Flash Loader Core User Guide for more information.

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

<url> is in the format:

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

where:

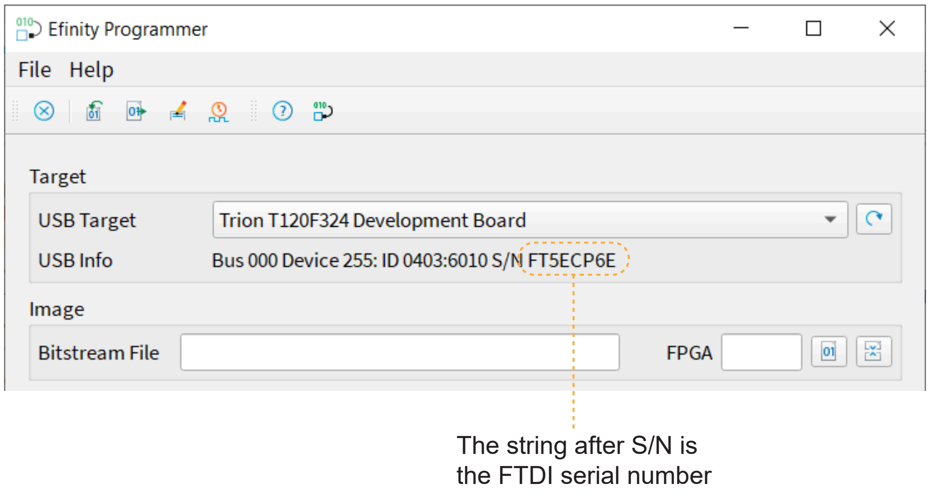
<product> is the USB product ID of the device

<product>	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® development board or FTDI device connected to your computer, you do not need to specify the serial number.

- In the Efinity® 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`.



Target

USB Target: Trion T120F324 Development Board

USB Info: Bus 000 Device 255: ID 0403:6010 S/N FT5ECP6E

Image

Bitstream File:

FPGA:

The string after S/N is the FTDI serial number

<*interface*> is the interface number. For Efinix® development boards, <*interface*> is always 1.

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, use:

```
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, use:

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

```
%EFINITY_HOME%\bin\python3 %EFINITY_HOME%\pgm\bin\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, use:

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

Windows:

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

(Optional) Use these options:

- `--pgm_opts mode` specifies the configuration mode. The available modes are:
  - `active`—SPI active configuration
  - `passive`—SPI passive configuration
  - `jtag`—JTAG programming
  - `jtag_bridge`—SPI active using JTAG bridge mode

  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.

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.

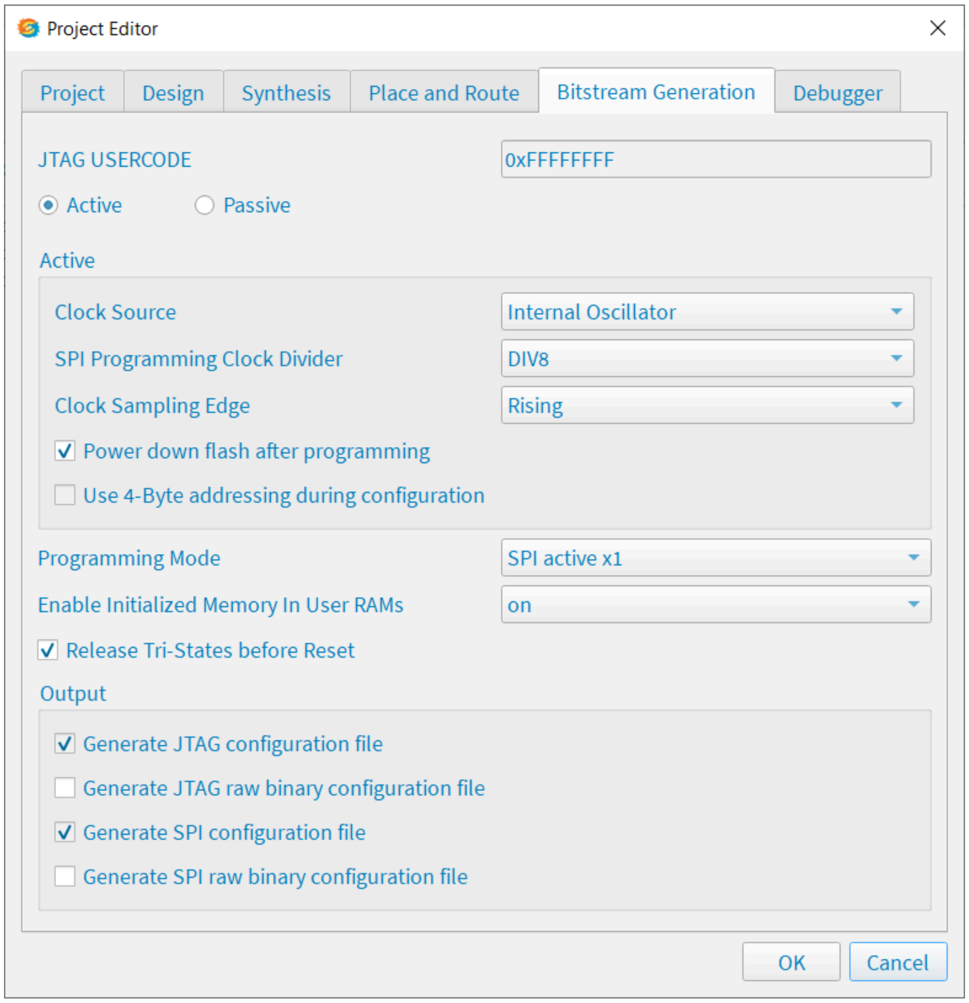
Table 28: Project-Specific Programming Options

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	Use this field to specify a 32-bit user electronic signature. The USERCODE is included in the bitstream. You can read it from the FPGA via the JTAG interface, and you can view the JTAG USERCODE in the Programmer's Advanced Device Status dialog box. Default: 0xFFFFFFFF
Clock Source	For Titanium FPGAs, choose whether you want to use the FPGA's internal oscillator or an external clock source as the configuration clock. For Trion FPGAs, this option is always **Internal Oscillator**.
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 Titanium FPGAs, choose whether the configuration clock should sample on the rising or falling edge. The default is **Rising**. For Trion FPGAs, this option is always **Rising**.
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
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. **on**: The bitstream has initialized memory. **off**: The bitstream does not have initialized memory. **smart**: For the Trion family, this option has the same effect as **on**. Default: smart
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).

Option	Notes
Generate JTAG configuration file	On (default): Generate a **.bit** file for JTAG configuration. Off: Do not generate a **.bit** file.
Generate JTAG raw binary configuration file	On: Generate a **.bin** file (raw binary) for JTAG configuration. Off (default): Do not generate a **.bin** file.
Generate SPI configuration file	On (default): Generate a **.hex** file for SPI programming. Off: Do not generate a **.hex** file.
Generate SPI raw binary configuration file	On: Generate a **.bin** file (raw binary) for SPI programming. Off (default): Do not generate a **.bin** 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 29: Setting Programming Options



JTAG USERCODE: 0xFFFFFFFF

● Active ○ Passive

Active

Clock Source: Internal Oscillator

SPI Programming Clock Divider: DIV8

Clock Sampling Edge: Rising

✓ Power down flash after programming

☐ Use 4-Byte addressing during configuration

Programming Mode: SPI active x1

Enable Initialized Memory In User RAMs: on

✓ Release Tri-States before Reset

Output

✓ Generate JTAG configuration file

☐ Generate JTAG raw binary configuration file

✓ Generate SPI configuration file

☐ Generate SPI raw binary configuration file

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

Verifying Configuration

Once you have configured your FPGA, you can confirm that the bitstream is loaded and the user design is running successfully:

- 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.
- For SPI active modes, if the **Programmer > SPI Active Options > Verify After Programming** option is turned on (which is the default), the Programmer confirms that the flash is programmed correctly. This option can help find errors such as a flash image that is 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).

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 check can be useful for debugging board problems such as signal integrity issues between the flash device and 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, for example, when the FPGA appears to be in user mode but is not functioning as expected. To simplify this debug process, add some logic to your RTL design that generates a signal you can monitor using an oscilloscope 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 for `CDONE` and `NSTATUS` and their meaning.

Table 29: Monitoring with the Efinity Programmer

CDONE	NSTATUS	Programmer Message	Description
0	0 or 1	Device is not in user mode	Configuration failed. If NSTATUS is 0, it indicates that the FPGA received a bitstream with the wrong device ID or a CRC error is detected.
1	1	Device is not in user mode	Bit stream transmission completed, however, it is corrupted with undetectable errors that is not wrong device ID or CRC error.
1	1	Device is in user mode	The FPGA is functioning correctly according to the user design.
			The FPGA is not functioning as expected. The most likely reason is that the FPGA is programmed with a bitstream that was targeted for a different configuration mode or width.

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 it's output value to a constant 0.

The following figures show example schematics connecting a microcontroller or LEDs to the FPGA's CDONE and NSTATUS pins:

Figure 30: Connect CDONE and NSTATUS to a Microcontroller

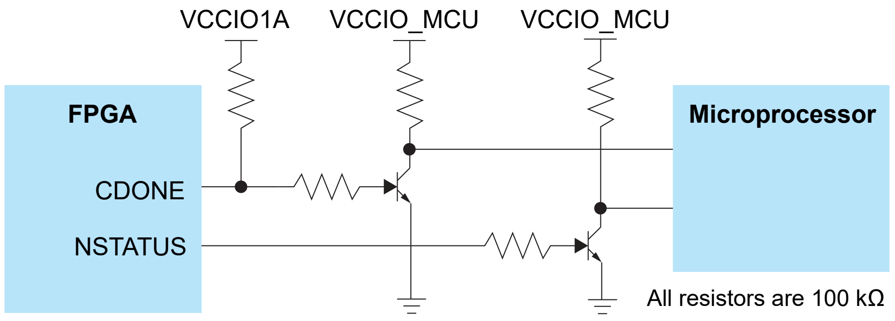
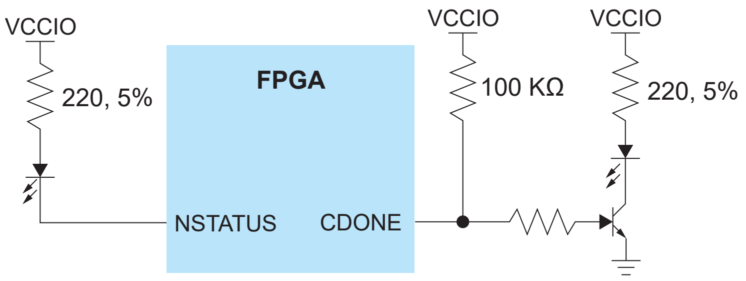


Figure 31: Connect CDONE and NSTATUS to LEDs



To add NSTATUS to your design as a GPIO:

1. In the Interface Designer, create a GPIO block for NSTATUS with the following settings:
   - **Instance Name**: NSTATUS
   - **Mode**: Output
   - **Constant Output**: 0
2. In the Instance View pane, assign the NSTATUS instance to the NSTATUS package pin (refer to the pinout file to find the package pin).
3. Recompile the design.
4. Download the bitstream to the flash memory on your board.
5. Reset the FPGA.

Observe the status of the CDONE and NSTATUS pins using the microcontroller or LEDs. The following table shows the possible conditions:

Table 30: Monitoring with a Microcontroller or LEDs

CRESET_N	CDONE	NSTATUS	Description
1	0	0	Configuration failed. The FPGA received a bitstream with the wrong device ID or it detected a CRC error.
1	0	1	The FPGA is in configuration mode.
1	1	0	The FPGA is in user mode.
1	1	1	The FPGA is in transition from configuration mode to user mode

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.

Learn more: 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 31: USB Programming Connections

Board	Connect to Computer with
Efinix development boards	USB cable
Your own board	FTDI *x*232H programming kit. For example: • FT2232H Mini Module • FT4232H Mini Module

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
```

Note: If your board was connected to your computer before you executed these commands, you need to disconnect and 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. (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.

Revision History

Table 32: Revision History

Date	Version	Description
September 2023	5.9	Updated CCK pin description. (DOC-1451)
June 2023	5.8	Updated Internal Flash Image option to Remote Update Flash Image in Programmer. (DOC-1302) Trion FPGAs only support SPI Active Using JTAG Bridge. (DOC-1319)
May 2023	5.7	Added IS25LP128 to list of supported flash devices. (DOC-1247)
April 2023	5.6	Added information about QFP100F3 packages and SPI active configuration for SIP packages. (DOC-1188) Updated JTAG configuration design considerations. (DOC-994)
February 2023	5.5	Added more description about valid and invalid image. (DOC-1118) Corrected user-defined pull-up/pull-down resistor formula. (DOC-1136) Updated power up sequence diagram. (DOC-954)
December 2022	5.4	Corrected SPI Clock Polarity and Phase Mode table. (DOC-946) Added note about not recommending user to pause FPGA configuration. (DOC-944) Added description about `CRESET_N` needs to be deasserted before JTAG configuration begins. (DOC-1069)
September 2022	5.3	Updated Verifying Configuration topic. **Enable CRC Check** option removed from the **Project Editor** (always on with Efinity v2022.1 and higher). (DOC-912) Removed support for C232HM-DDHSL-0 cable. (DOC-860) Removed JTAG Device ID for BGA49 packages. (DOC-899) Added note about Trion only supports SPI flash with 3-byte addressing mode for configuration. (DOC-910) Updated supported flash devices. (DOC-896) Updated Project-Based Programming Options topic for new options.
August 2022	5.2	Defined VCC is 1.2 V in the Connection Requirements for Unused Resources table. (DOC-770) Corrected SPI active x1 SDI to CDI0 example connection. (DOC-783) Added topic on SPI clocking and sampling. (DOC-625) Corrected SS_N connection to be bidirectional in active SPI mode connection diagrams. Updated configuration flow diagram. Added JTAG USER TAP instructions.
April 2022	5.1	Added user-defined pull-down resistance formula. (DOC-747) Added Program using a JTAG Bridge topic. Added topic on combining a bitstream and other data into a single file for programming. Re-organized topics about working with bitstreams.

Date	Version	Description
March 2022	5.0	Moved FTDI hardware connection diagrams into Programming Hardware Connections topic. Added topic about external pull-up resistors. (WEB-39) Removed optional pull-up resistors from SPI active circuitry diagrams. Updated power up sequence stating that all supplies must be powered up within 10 ms. (DOC-631)
January 2022	4.9	Improved connection diagrams to show pull-ups to point upwards. (DOC-612) Added Bitstream Bytes Packed into Parallel Bus for x16, x8, x4, x2, and x1. (DOC-626) Added reference to AN 035: SPI Passive Programming with Raspberry Pi. (DOC-569) With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG. (DOC-638) Added Exporting to .svf Format topic. (DOC-569) Corrected Passive (x2) without CSI or CBUS2 figure. 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) Updated active mode connection diagram.
November 2021	4.8	Corrected the power-up sequence waveform.
November 2021	4.7	Updated JTAG mode connection diagram. (DOC-546) Updated the Project-Based Programming Options topic. (DOC-550) Added Macronix MX75L and MX75U to supported flash devices. (DOC-573) Updated note about not driving any Trion® I/O pins before the Trion® FPGA is powered up. (DOC-587) Added support for FTDI FT4232H Mini Module. (DOC-597)
September 2021	4.6	Updated the Project-Based Programming Options topic. (DOC-523) Added XT25F family to list of supported flash devices. (DOC-529) Minor text corrections to the Programming the Flash Using a JTAG Bridge topic. (DOC-543) Updated Verifying Configuration topic. (DOC-539, DOC-486) Removed x4 from Passive (x2) without CSI or CBUS2 figure as the figure is showing x1 CBUS settings.
August 2021	4.5	Updated FPGA configuration mode topic. Added flash programming mode topic. Added topic on verifying configuration. (DOC-508) Corrected FPGA pin names for the SPI passive FTDI FT2232H Mini Module Connections figure. Updated for T20QFP144 package. (DOC-519) Added note about not connecting the `NSTATUS` pins of multiple FPGAs in daisy chain configuration. (DOC-518) Added note about FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495) Added note about using DIV4 in x2 and x4 parallel daisy chain configuration. (DOC-525)

Date	Version	Description
July 2021	4.4	Described more detail on the Enable Initialized Memory in User RAMs option in the **Project Editor** > **Bitstream Generation** tab. (DOC-458) Updated the Windows USB driver installation topic. Updated the FTDI command-line programming topic. Added the command-line programmer configuration mode options. (DOC-430) Corrected SPI flash daisy chain configuration example figures. (DOC-483) Updated CDI*n* pin description. (DOC-483)
May 2021	4.3	Added Macronix MX25U and Micron M25P16 to list of supported flash devices. Added the SENSE pin to Figure 17: Supervisor IC Power Up Circuitry on page 29. 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 anf SPI passive programming. Corrected configuration clock signal name from CCKO to CCK. Added JTAG device IDs for T20BGA324 and T20BGA400.

Date	Version	Description
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.
January 2020	3.0	Added schematics and information on active and passive configuration for packages that do not have the CSI or CBUS2 pins bonded out. Clarified usage of TEST_N pin in configuration and user modes. Clarified dedicated vs. dual-purpose configuration pins. Removed DIV1 and DIV2 SPI clock divider settings; currently they are not supported. Added information on programming with the SPI Active using JTAG Bridge mode. Added JTAG device IDs for T35, T55, T85, and T120 FPGAs. Added bitstream lengths for T20 (BGA324 and BGA400), T35, T55, T85, and T120 FPGAs. Clarified handling of multi-function configuration pins that are outputs in user mode. When installing USB drivers for Windows using the Zadig software. specify the **libusb-win32** driver instead of **libusbK**. Added a note to the Support for Multiple Images topic to refer readers to the data sheet for which modes support multiple images.
September 2019	2.1	Added information on JTAG chain programming. Added a note that adding optional header information to the bitstream file using the Efinity® software can add up to 1k bits to the bitstream length. Updated the maximum supported configuration bits. Updated CBUS[2:0] setting for SPI active mode.
March 2019	2.0	Added information about JTAG programming support. Updated information on installing the programming cable USB driver. Added additional information on using the Efinity® Programmer.
December 2018	1.2	Added sections on using the Efinity® Programmer. Added note for SPI passive mode that the `CSI` pin can also be connected to VCCIO. Fixed incorrect equation for calculating bitstream length. Changed `TCK` to have a recommended weak pull down. Fixed typo in Daisy Chaining with a SPI Flash Device on page 33, Parallel Daisy Chain Configuration (x4) Interface Example.
August 2018	1.1	Added section on choosing an SPI flash device.
June 2018	1.0	Initial release.