AN 033: Configuring Titanium FPGAs

AN033-v2.9
March 2025
www.efinixinc.com

Contents

About Configuring Titanium FPGAs

This document describes how to configure Titanium 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.

Table 1: Titanium FPGA Bitstream Size

FPGA	Maximum Supported Configuration Bits (Single Image)	Packages
Ti35	13,735,488	All
Ti60	13,735,488	All
Ti85	38,440,704	All
Ti90	50,096,640	All
Ti120	50,096,640	All
Ti135	38,440,704	All
Ti165	105,994,048	All
Ti180	50,096,640	All
Ti240	105,994,048	All
Ti375	105,994,048	All

The Efinity software automatically compresses the bitstream; therefore, the bitstream size for a typical design should be about 50% of the maximum size shown.

Note: If you are using any of the Titanium security features, the software cannot compress the bitstream.

Bitstream Size for Multiple Images

The following table shows the approximate flash size required for one to four images with and without compression.

> ⚠ **Important:** These numbers are estimates based on typical designs; the actual size varies slightly depending on your design and could be larger than the values shown here.

Table 2: Bitstream Size for Multiple Images

Rounded to nearest Mbit

Uncomp. = Uncompressed

Comp. = Compressed

Number of Images	Ti35, Ti60		Ti85, Ti135		Ti90, Ti120, Ti180		Ti165, Ti240, Ti375	
	Uncomp. Maximum (Mbits)	Comp. Typical (Mbits)	Uncomp. Maximum (Mbits)	Comp. Typical (Mbits)	Uncomp. Maximum (Mbits)	Comp. Typical (Mbits)	Uncomp. Maximum (Mbits)	Comp. Typical (Mbits)
1	14	8	38	19	50	25	106	64
2	28	16	76	38	100	50	212	128
3	42	24	114	57	150	75	318	192
4	56	32	152	76	200	100	424	256

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)

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

For example:
- *Ti60 FPGA*—Assuming 8.85 Mbits of *compressed* configuration data:
- *Configuration clock frequency*—10 MHz
- *Ti60 configuration data bus width*—8 bits[1]

Configuration time: 8.85 Mbits ÷ (10MHz × 8 bits) = 110.625 ms

---

[1] Not all Ti60 packages support x8 bus widths.

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.

> (i) **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 3: 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 mode except for TCK and TDO.

Calculate the resistor value as described in **Resistors in Configuration Circuitry** on page 46.

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_{creset_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 is typically a 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 is typically a 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

---

[2] CDONE has a drive strength of 12 mA at 1.8 V.

Pins	Direction	Description	External Weak Pull Up/ Pull Down Requirement
JTAG_VCCIO_SEL	Input	JTAG voltage select pin. This pin affects the BR4 bank voltage, or any banks merged with the BR4 bank. Supply VCCIO33_BR4 with 1.8 V and connect a 1 kΩ external resistor between this pin and ground to use JTAG at 1.8 V. Leave this pin floating to use the default JTAG at 3.3 V or 2.5 V. Available on Ti85, Ti135, Ti165, Ti240, and Ti375 FPGAs only.	Floating or pull down

Table 4: Dual-Purpose Configuration Pins

In user mode (after configuration), you can use these dual-purpose pins as general I/O.
Calculate the resistor value as described in **Resistors in Configuration Circuitry** on page 46.

Configuration Functions	Direction	Description	External Weak Pull Up/ Pull Down Requirement
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
CCK	I/O	Passive SPI input configuration clock or active SPI output configuration clock.	Optional pull up if required by external load
CDI*n*	I/O	Data input for SPI configuration. *n* is a number from 0 to 31 depending on the SPI configuration data width. CDI0 is an output in x1 active configuration mode and is a bidirectional pin in all other active configuration modes. CDI4 is a bidirectional pin in x8 active configuration mode. In a multi-bit daisy chain connection, CDI[31:0] connects to the data bus in parallel.	Optional pull up if required by external load

Configuration Functions	Direction	Description	External Weak Pull Up/ Pull Down Requirement
CSI	Input	Chip select. 0: The FPGA is not selected or enabled and will not be configured. 1: Select the FPGA for all configuration modes. Ti35, Ti60 FPGAs require this setting for JTAG configuration mode. This pin is not bonded out in some of the smaller packages, such as the F100S3F2 and F100. This pin is not bonded out in some of the smaller packages, such as the F100S3F2 and F100. This pin is not bonded out in some of the smaller packages, such as the F100. CSI must remain high throughout configuration.	Pull up
CSO	Output	Chip select output. Asserted after configuration is complete. Connect this pin to the chip select pin of the next FPGA for daisy chain configuration.[3] This pin is not bonded out in some of the smaller packages, such as the F100S3F2 and F100. This pin is not bonded out in some of the smaller packages, such as the F100S3F2 and F100. This pin is not bonded out in some of the smaller packages, such as the F100.	–
NSTATUS	Output	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.	–
SSL_N	I/O	SPI configuration mode select. The FPGA senses the value of SSL_N when it comes out of reset (i.e., CRESET_N transitions from low to high). 0: Passive mode; connect to external weak pull down. 1: Active mode; connect to external weak pull up. In active configuration mode, SSL_N is an active-low chip select to the flash device (CDI0 - CDI3).	Pull up or pull down
SSU_N	Output	Active-low chip select to the upper flash device (CDI4 - CDI17) in active x8 configuration mode (dual quad mode). Not available in W64, F100S3F2, and F100 packages. Not available in W64, F100S3F2, and F100 packages. Not available in F100 packages.	Optional pull up if required by external load
EXT_CONFIG_CLK	Input	Alternative clock in active configuration mode.	Optional pull up if required by external load
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

ⓘ **Note:** Refer to the column Configuration Functions in the pinout file.

---

[3] Cascaded configuration is not supported in the F100S3F2 and F100 packages.

FPGA Configuration Modes

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

Table 5: 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 `SSL_N` and `TEST_N` input pins. (The `SSU_N` pin can be any value.)
- Do not toggle the mode pins before the FPGA enters user mode.

Table 6: SPI Hardware Settings

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

Configuration Mode	CSI	TEST_N	SSL_N
SPI Active	1	1	1
SPI Passive	1	1	0
JTAG	1	1	N/A

The JTAG/boundary-scan configuration interface is always available regardless of pin settings. If you send configuration instructions to the JTAG interface, the Titanium 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 specific to the FPGA and package. Refer to the data sheet for information on the supported configuration modes.

About SPI Clocking and Sampling

Table 7: 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	User configurable, the default is positive. You can change this setting in the Efinity **Project Editor** > **Bitstream Generation** tab.

> **!** **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 8: 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 Titanium FPGA devices.

Figure 1: SPI Clock Polarity and Phase Modes Diagram

(CPOL Bit, CPHA Bit) = Mode #
(0,0) = Mode 0
(0,1) = Mode 1
(1,0) = Mode 2
(1,1) = Mode 3
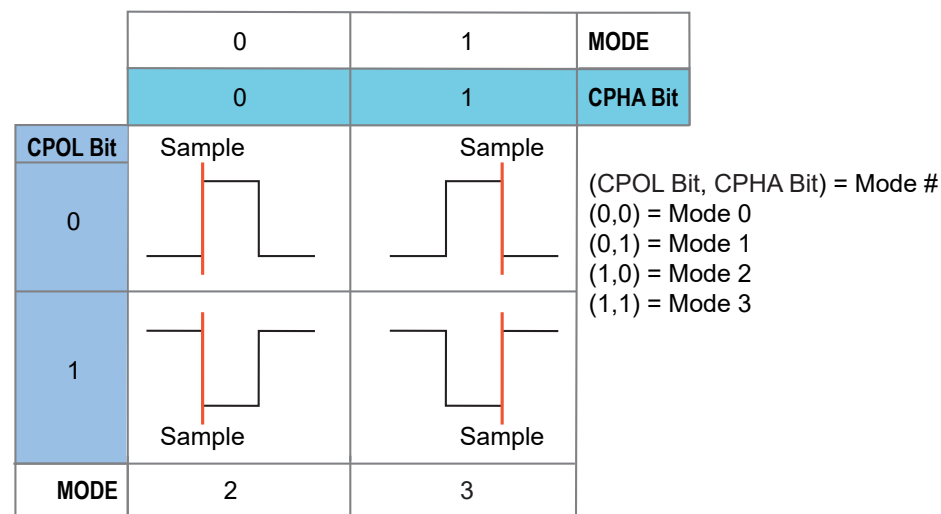
SPI Active Mode

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:** Titanium Ti35 and Ti60 FPGAs only support SPI flash memory with 3-byte addressing mode for configuration.

For even faster loading times, Titanium FPGAs also support a x8 mode that uses two identical SPI flash devices.

Table 9: Active Mode Instructions

Instruction	Description	SPI Data Width
0BH	Fast read	x1
3BH	Dual output fast read	x2
6BH	Quad output fast read, single flash device	x4
	Quad output fast read, two flash devices	x8

Connection Examples

> **Note:** Circuitry is required to control the CRESET_N pin to meet the t_{CRESET_N} requirement.

Figure 2: Active (x1)

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
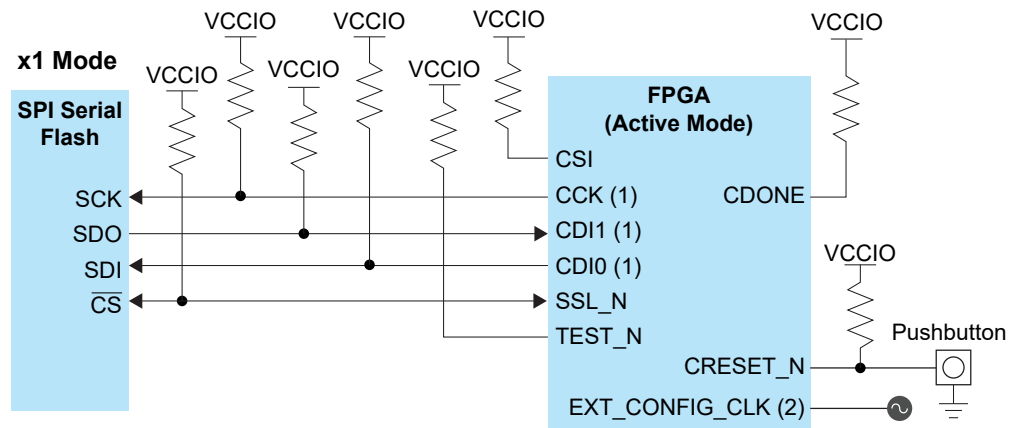
Notes:
1. The external pull-up is optional unless required by an external load.
2. Connect the EXT_CONFIG_CLK to an external clock source ONLY if Clock Source is selected as external clock in Efinity Project Editor bit stream generation.

Figure 3: Active (x2)

See **Resistors in Configuration Circuitry** on page 46 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.
3. Connect the EXT_CONFIG_CLK to an external clock source ONLY if Clock Source is selected as external clock in Efinity Project Editor bit stream generation.

Figure 4: Active (x4)

See **Resistors in Configuration Circuitry** on page 46 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.
3. Connect the EXT_CONFIG_CLK to an external clock source ONLY if Clock Source is selected as External clock in Efinity Project Editor bit stream generation.

Figure 5: Active (x8)

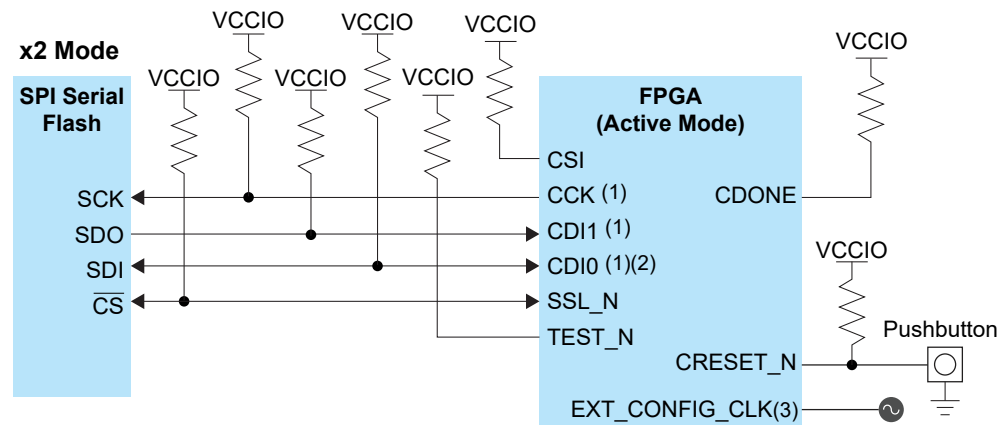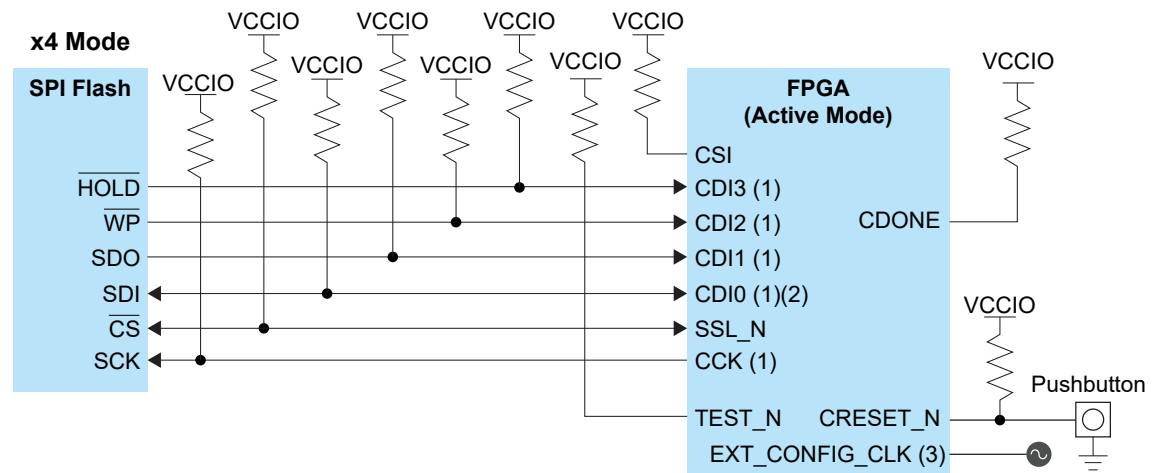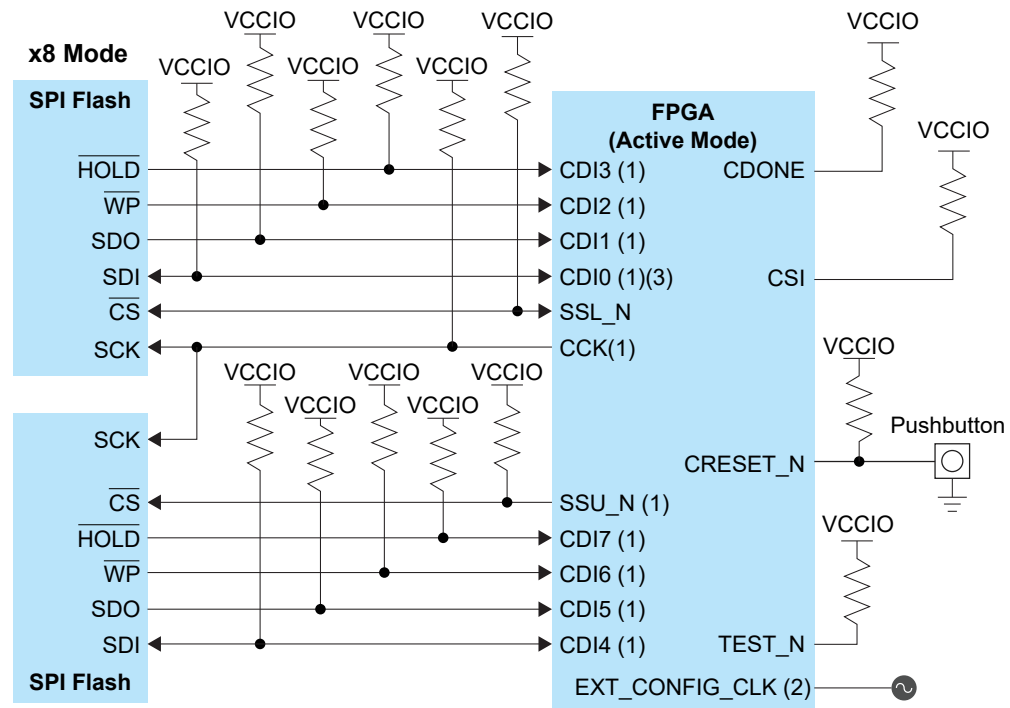See **Resistors in Configuration Circuitry** on page 46 for the resistor values.

Notes:
1. The external pull-up is optional unless required by an external load.
2. Connect the EXT_CONFIG_CLK to an external clock source ONLY if Clock Source is selected as External clock in Efinity Project Editor bit stream generation.
3. In x8, 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 SSL_N is externally pulled up, the FPGA enters active SPI configuration mode. Then, the FPGA:

1. Starts configuration by driving SSL_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 or 32h'00000000. See **SPI Flash Address Width** on page 19.
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.

Refer to **SPI Active Clocking** on page 18 and **Using an External Configuration Clock** on page 18 for more details on clocking.

Figure 6: SPI Active (x1) Configuration
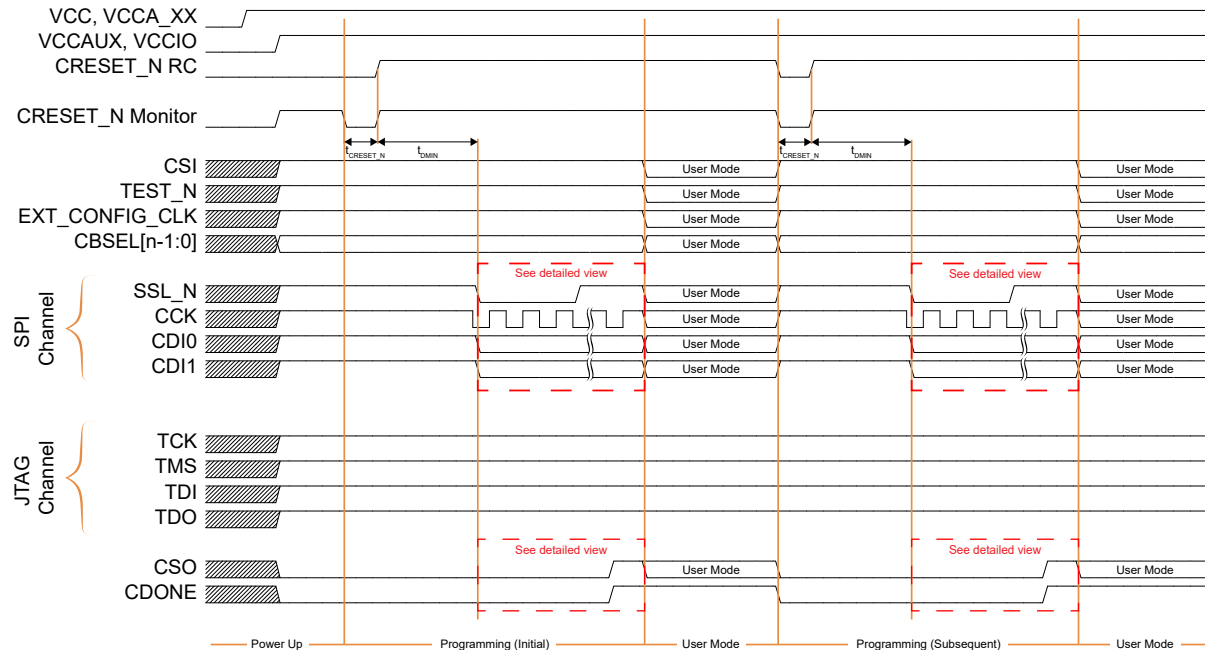
NOTE:
CRESET_N RC refers to the CRESET_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
CRESET Monitor refers to the CRESET_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 7: SPI Active (x1) Configuration (Detailed View)
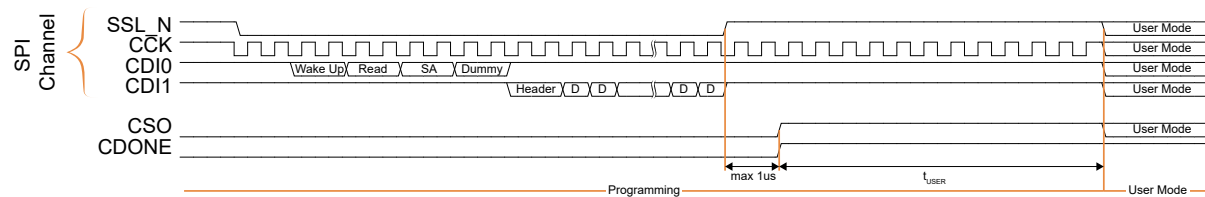
LEGEND:
SA: Start Address   D: Data

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

❗ **Important:** JTAG pins must be inactive during configuration.

❗ **Important:** The external configuration clock pin must be inactive during configuration if the FPGA is using its internal oscillator as the configuration clock source (default).

📖 **Learn more:** Refer to **Using an External Configuration Clock** on page 18 for information on how to use an external clock instead.

📖 **Learn more:** Refer to "Configuration Timing" in the Titanium data sheet for timing specifications.

SPI Active Mode for SIP Packages

Titanium FPGAs in F100F3S2 packages are a system-in-package (SIP) that includes an internal SPI flash; as such, you can ONLY start SPI active configuration with this internal SPI flash. However, you can still cascade an external SPI flash by sharing the SPI bus to increase the storage capacity for user data in your application.

Depending on the setup, you must observe the following pin connection requirements in addition to the connections shown in Connection Examples on page 12.

Figure 8: Connections between FPGA and SPI Flash Inside the Package
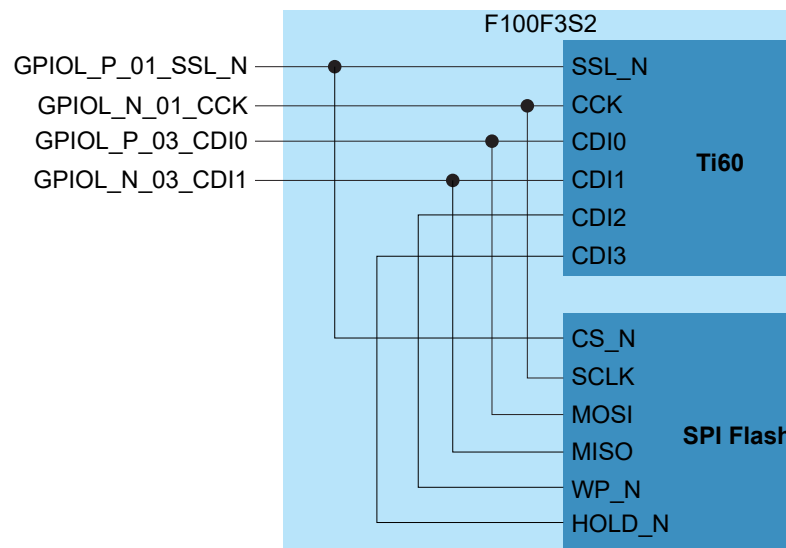
Figure 9: Configuration with Internal Flash
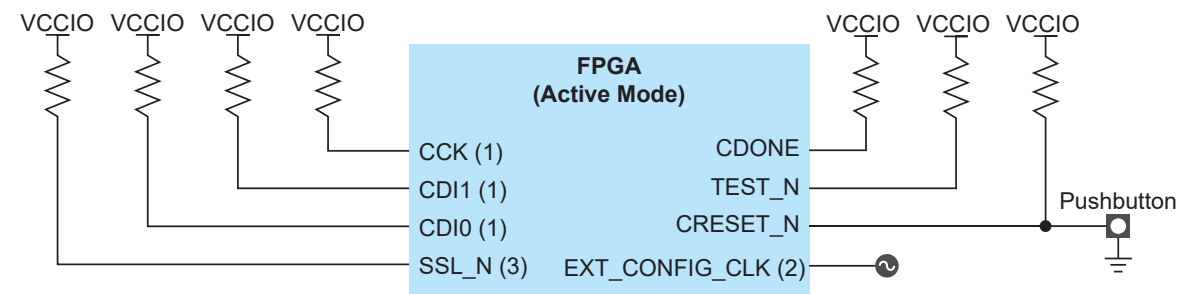
Notes:
1.   The external pull-up is optional unless required by an external load.
2.   Connect the EXT_CONFIG_CLK to an external clock source if the clock source is selected as an external clock in the Efinity **Project Editor > Bitstream Generation** tab.
3.   Only use the SSL_N pin as the GPIO to access the internal flash.

Figure 10: Cascading an External SPI Flash for User Data Storage

¹ The external pull up is optional unless
required by an external load.
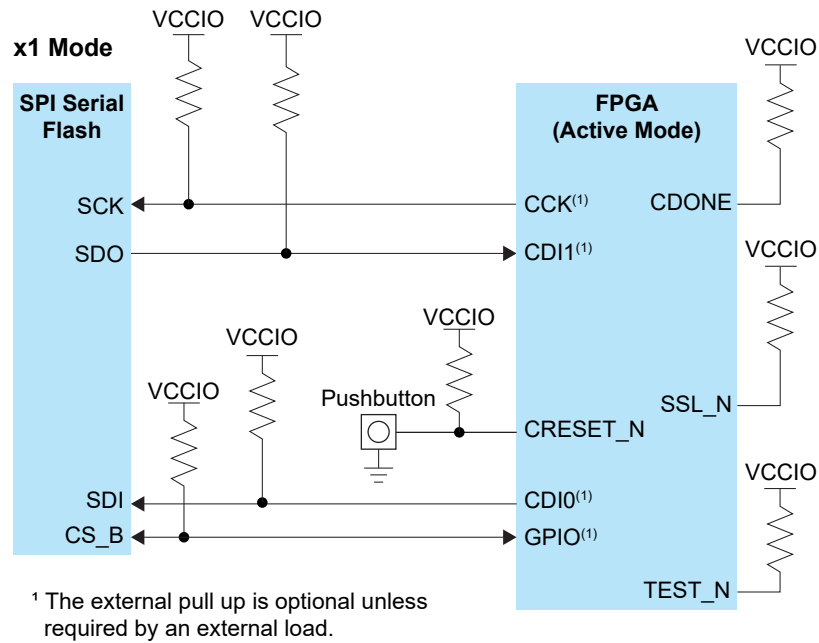
Note: You can access to the internal and external SPI flash by instantiating either the SPI flash block or GPIO blocks of `GPIOL_P_01(SSL_N)`, `GPIOL_N_01(CCK)`, `GPIOL_P_03(CDI0)` and `GPIOL_N_03(CDI1)` in the Interface Designer. Drive the `SSL_N` pin to low to access the internal flash and drive the GPIO connected to the external flash's CS_B to access the external flash. Only enable one flash a time.

SPI Active Mode without CSI

Titanium FPGAs in smaller pin count packages, such as the F100 and F100S3F2, 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.

SPI Active Clocking

An internal oscillator or external clock generates the 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: Clock Setting Recommendations

Clock Source	Divider	Frequency (MHz) (Typical)	Sample On (Default)
Internal Oscillator	DIV8	10	Positive edge
Internal Oscillator	DIV4	20	Positive edge
Internal Oscillator	DIV2	40	Negative edge
Internal Oscillator	DIV1	80	Negative edge
External Clock	–	<40	Positive edge
		40 - 125	Negative edge

> **Note:** A free running external clock is required during bitstream generation if External Clock is selected as the Clock Source. Toggling a non-continuous external clock after warm boot will result in configuration failure, and CDONE does not go high.

> **Note:** These sampling recommendations are for typical cases. It is *likely* that high-speed clocking requires negative edge sampling at the FPGA. In practice, the sampling edge you need to use depends on the round-trip delay, that is, the total delay from the clock edge sent from the FPGA to the flash, the t_{CO} of the flash memory (which can be quite high), and the return data trip back to the FPGA. Therefore, you should consider your board design and choice of flash device when determining whether to sample the positive or negative edge.

Using an External Configuration Clock

For SPI active configuration, the FPGA internally generates the configuration clock signal (CCK) by default. The FPGA can also use an external clock, that you send to the FPGA via the external configuration clock pin. In the pinout file, this pin is listed as EXT_CONFIG_CLK in the Configuration Functions column.

> **Important:** If you are using the internally generated clock, the external configuration clock pin must be inactive during configuration.

To use an external configuration clock:

- In the Efinity Project Editor, choose **Bitstream tab > Clock Source > External Clock**. This change affects the bitstream; therefore, you should re-generate the bitstream when you change this setting.
- Provide a free running clock to the external configurartion clock pin before triggering the reconfiguration.
- When the FPGA enters user mode, you can stop toggling the external configurartion clock pin.

(!) **Important:** If the bitstream **Clock Source** setting is **External Clock**, you **must** provide a clock on the external configuration clock pin during configuration. If you do not provide the clock, the FPGA remains stuck in configuration mode. You can restore the system with the following procedure:

1. Change the Project Editor **Clock Source** option to **Internal Oscillator** and re-generate the bitstream.
2. Pull the FPGA's SSN pin low to boot the device in SPI passive mode.
3. Use the Efinity Programmer to program the FPGA using the SPI Active using JTAG Bridge mode (see **Program using a JTAG Bridge (New)** on page 60).

SPI Flash Address Width

Table 11: Supported SPI Flash Address Width for Configuration

Device	SPI Flash Address Width	
	24 bit (3-byte Addressing)	32 bit (4-byte Adressing)
Ti35, Ti60	✓	-
Ti85, Ti90, Ti120, Ti135, Ti165, Ti180, Ti240, Ti375	✓	✓

Refer to **Project-Based Programming Options** on page 65 for information about enabling 4-byte addressing in the Efinity software.

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 Titanium 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 Titanium 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. .

Figure 11: Passive (x1)

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
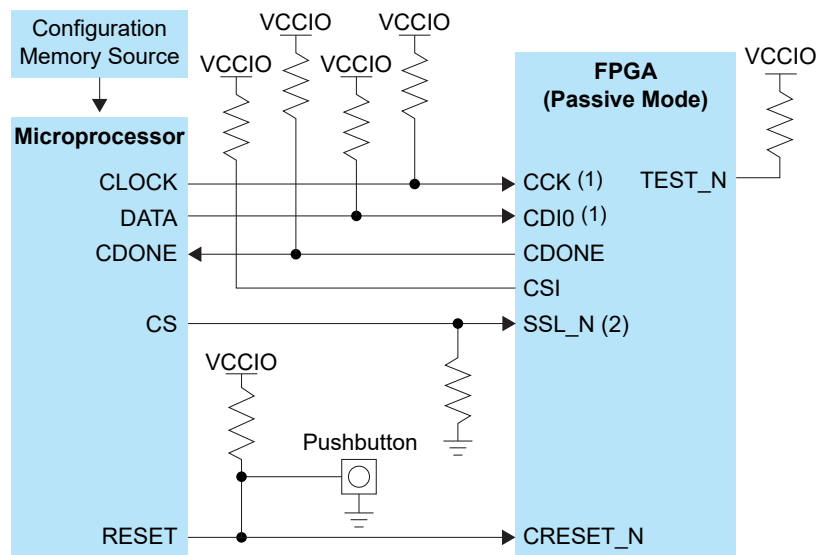
Notes:
1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.

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

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

You cannot disable the configuration by driving either SSL_N to high or CSI to low. You have to complete the bit stream transmission before communicating with other devices on the same SPI bus. A waveform, such as that shown in **Figure 12: Supported SPI Waveform** on page 21, should result in successful configuration. While a waveform, such

as that shown in **Figure 13: Unsupported SPI Waveform** on page 21, will result in configuration failure.

Figure 12: Supported SPI Waveform

Complete bitstream transmission before activating other devices on the same SPI bus.
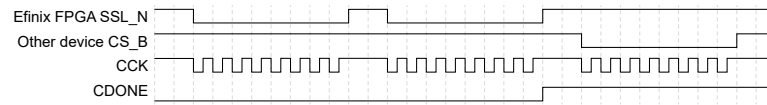
Figure 13: Unsupported SPI Waveform

Interleaving the bitstream transmission and communication with other devices on the same SPI bus.
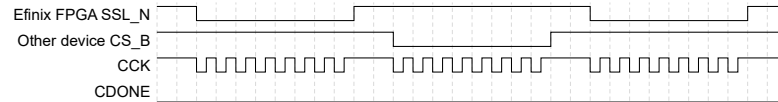
Table 12: Bitstream Bits in Series

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 14: Passive (x32)

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
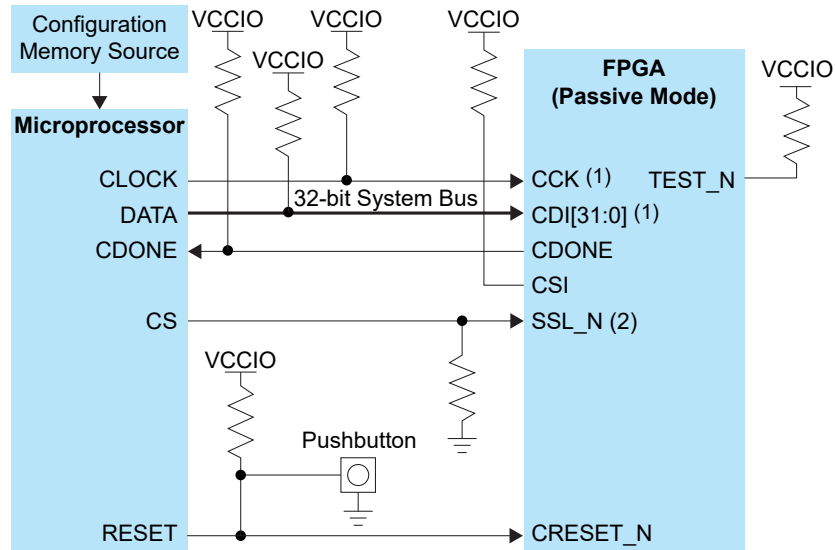
Notes:
1. The external pull-up is optional unless required by an external load.
2. The external pull-down is optional.

Table 13: 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 14: 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 15: Bitstream Bytes Packed into 8 bit Parallel Bus

CBUS	7		0
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 16: Bitstream Bits Packed into 4 bit Parallel Bus

CBUS	Byte Order	3	2	1	0
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 17: Bitstream Bits Packed into 2 bit Parallel Bus

CBUS	Byte Order	1	0
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_N low for a duration of t_{CRESET_N} and then release it to start the SPI passive configuration. After t_{DMIN}, the Titanium FPGA samples the synchronization pattern and begins configuration.
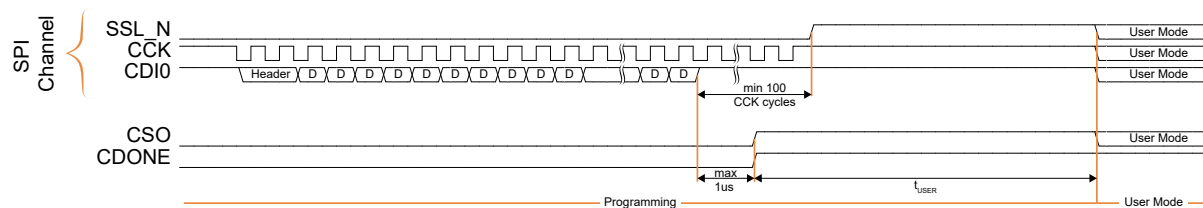
Figure 15: SPI Passive (x1, Mode 3) Configuration
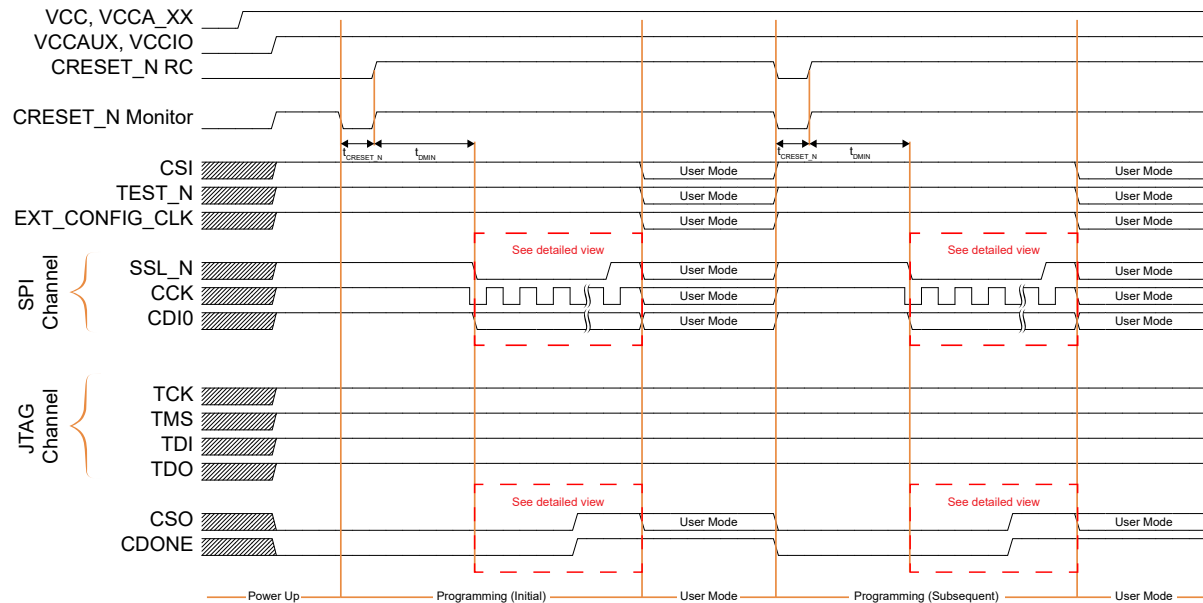
NOTE:
CRESET_N RC refers to the CRESET_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
CRESET Monitor refers to the CRESET_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 16: SPI Passive (x1, Mode 3) Configuration (Detailed View)

LEGEND:
D: Data

> **Note:** Refer to "Power-up Sequence" in the Titanium 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 12: Supported SPI Waveform** on page 21 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.
- CSI must stay high during configuration.

- It is recommended that SSL_N remain low during configuration. Sometimes, however, an SPI Master might release the SSL_N to high during firmware latency. Refer to **Figure 12: Supported SPI Waveform** on page 21 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 Titanium FPGA for at least 100 cycles after sending the last configuration data.

Learn more: Refer to the Titanium 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 without CSI

Titanium FPGAs in smaller pin count packages, such as the F100 and F100S3F2, may not have the CSI signal 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.

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.

Efinix FPGAs support IEEE standard 1149.1 - 2001.

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 18: Supported Topaz and Titanium JTAG Instructions

Instruction	Binary Code [4:0]	Description
BYPASS	11111	Enables BYPASS.
DEVICE_STATUS	01100	Lets you read the device configuration status.
EFUSE_PREWRITE	11000	Loads user data for fuse operations.
EFUSE_USER_WRITE	11010	Blows fuses as defined in EFUSE_PREWRITE.
EFUSE_WRITE_STATUS	11011	Returns status of EFUSE_USER_WRITE operation.
ENTERUSER	00111	Changes the FPGA into user mode.
EXTEST	00000	Enables the boundary-scan EXTEST operation.
IDCODE	00011	Enables shifting out the IDCODE.
INTEST	00001	Enables the boundary-scan INTEST operation.
JTAG_USER1	01000	Connects the JTAG User TAP 1.
JTAG_USER2	01001	Connects the JTAG User TAP 2.
JTAG_USER3	01010	Connects the JTAG User TAP 3.
JTAG_USER4	01011	Connects the JTAG User TAP 4.
PROGRAM	00100	JTAG configuration.
SAMPLE/PRELOAD	00010	Enables the boundary-scan SAMPLE/PRELOAD operation.
USERCODE	01101	Use this instruction to program a 32-bit signature into the FPGA during programming.

Important: Once JTAG is disabled, the only supported instructions are: `BYPASS`, `DEVICES_STATUS`, `IDCODE`, and `USERCODE`.

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 17: JTAG Programming (Ti35 and Ti60 FPGAs with CSI)

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
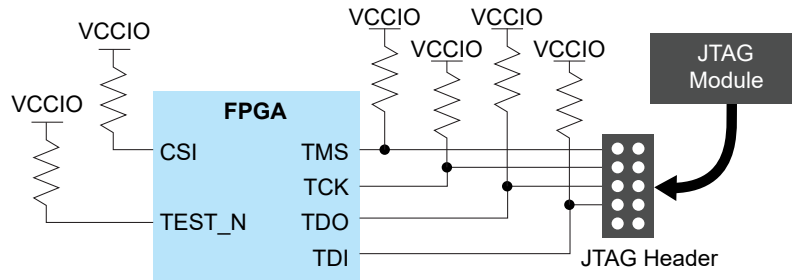
Figure 18: JTAG Programming (Other FPGAs)

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
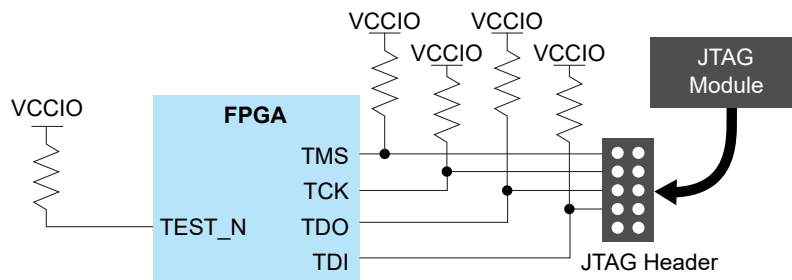
The `CRESET_N` signal needs to be deasserted before JTAG configuration begins. When configuration ends, the JTAG host issues the `ENTERUSER` instruction to the FPGA. After `CDONE` goes high and the FPGA receives the `ENTERUSER` instruction, the FPGA waits for t_{USER} to elapse, and then it goes into 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 19: JTAG Programming Waveform
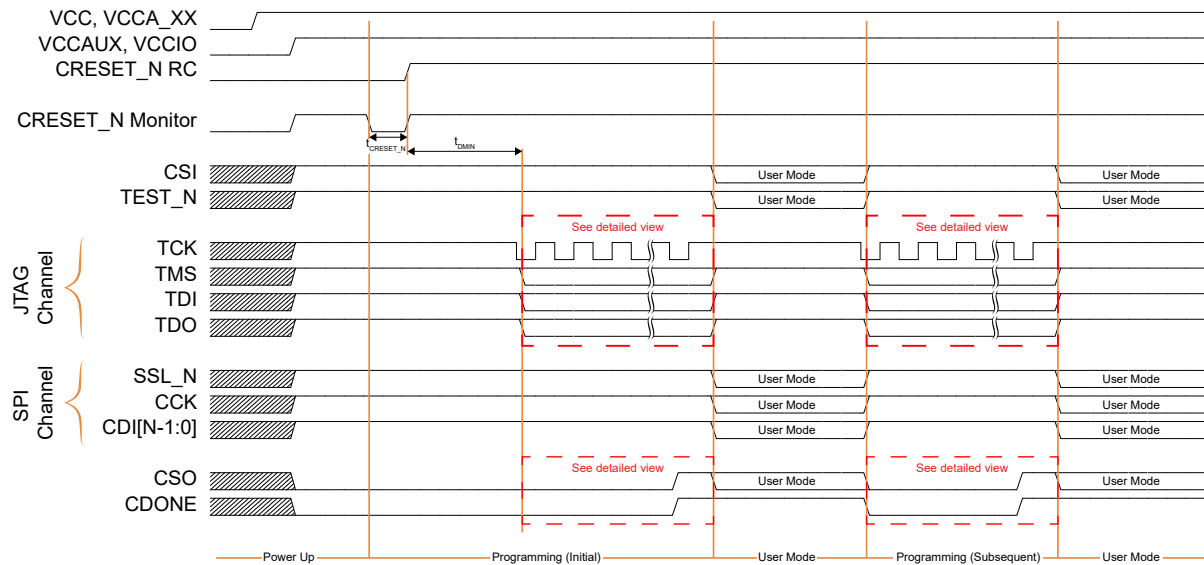
NOTE:
CRESET_N RC refers to the CRESET_N pin connected to an RC circuit that delays the voltage supplied to the pin after power-up.
CRESET Monitor refers to the CRESET_N pin connected to a pull-up resistor, where the supplied voltage is the same as VCCIO.

Figure 20: JTAG Programming Waveform (Detailed View)
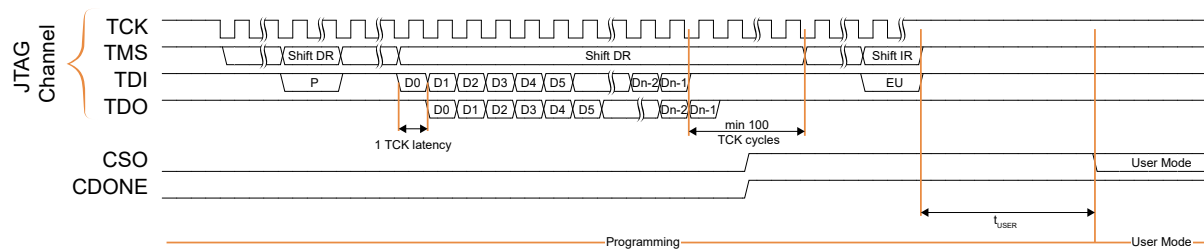
Legend:
P: PROGRAM   EU: ENTERUSER

> **Note:** Refer to "Power-up Sequence" in the Titanium 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.

1.8 V JTAG Support

The JTAG_VCCIO_SEL function is a dedicated JTAG voltage setting pin. This setting also affects the BR4 bank voltage or any banks that merged with the BR4 bank.

Table 19: Supported Titanium FPGAs

Ti35	Ti60	Ti85	Ti90	Ti120	Ti135	Ti165	Ti180	Ti240	Ti375
–	–	✓	–	–	✓	✓	–	✓	✓

(!) Important: For the device without the JTAG_VCCIO_SEL, TCK runs at a lower maximum frequency at VCCIO 1.8V. Refer to the **Titanium data sheets** for the JTAG timing.

Figure 21: 1.8 V JTAG Connection
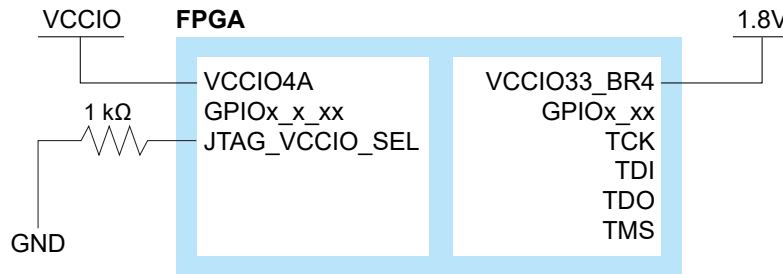
Supply VCCIO33_BR4 with 1.8 V and connect an external resistor between the JTAG_VCCIO_SEL pin and ground to use JTAG at 1.8 V.

Figure 22: 2.5, 3.0, 3.3 V JTAG Connection
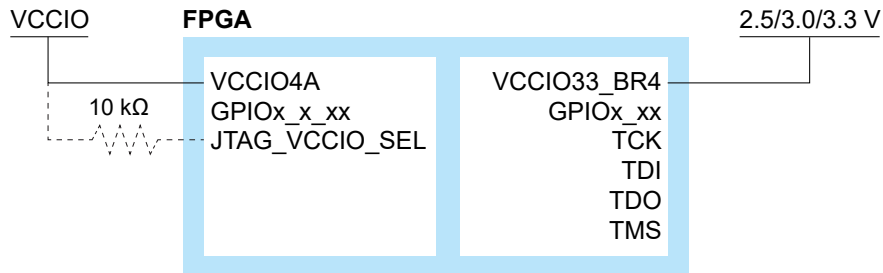
Supply VCCIO33_BR4 with 2.5, 3.0, or 3.3 V and leave the JTAG_VCCIO_SEL pin floating or connect it to an external pull-up resistor to use the default JTAG at 2.5, 3.0, or 3.3 V.

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.

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 23: MCU with Separate SPI Bus Connections to SPI Flash and FPGA
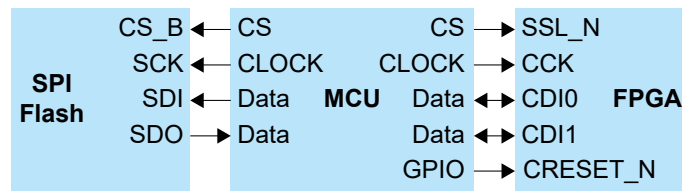
FPGA with Separate SPI Bus Connections

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

- `SSL_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 24: FPGA Configured with SPI Bus Connections to SPI Flash and GPIO Emulated MCU SPI Bus Connections to MCU

- `SSL_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.

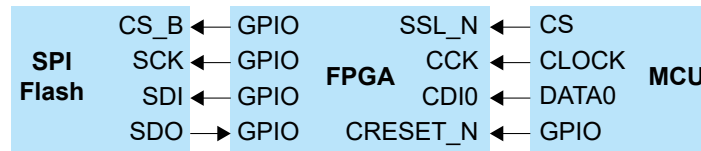*Figure 25: FPGA Configured with SPI Bus Connections to MCU and GPIO Emulated MCU SPI Bus Connections to SPI Flash*

Shared SPI Bus Connections

- Per the SPI protocol, the SPI master sets the `CS_B`/`SSL_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 `SSL_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 `SSL_N` to low
  — Drive `CRESET_N` to low.

Figure 26: Shared SPI Bus Connections
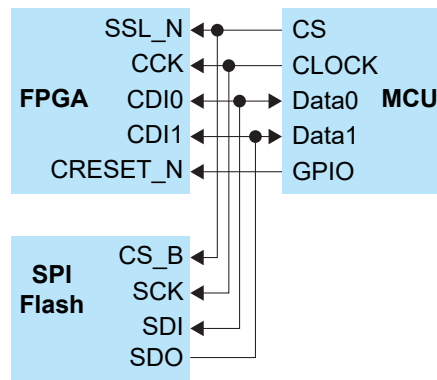
To program the SPI flash or to enter SPI passive configuration, the off-chip MCU should:
- Drive `SSL_N` to low
- Drive `CRESET_N` to low.

> **Note:** You can treat the Ti60F100S3F2 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.

> **Important:** If you want to access the internal SPI Flash with the external MCU, you need to: **Turn on Configuration** > **External Flash Control** > **Enable external controller access to flash memory** in the Interface Designer if you instantiate the SPI Flash block.

> **Note:** Refer to Support for Multiple Images on page 41 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 69 for more information.

Note: If you have an off-chip MCU, the configuration advice is the same as **Figure 28: Flash Programming Board Setup** on page 35 with an SPI header.

Figure 27: Shared SPI Bus Connections with Off-Chip MCU through SPI Header
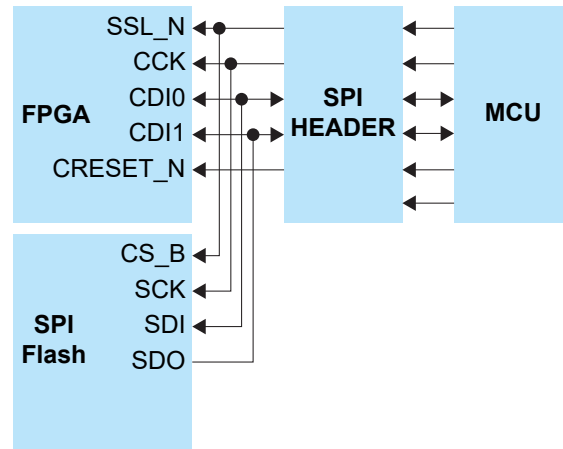
Learn more: Refer to **Flash Programming Modes** on page 34 for further advice.

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 anyway.

> **Important:** Once JTAG access has been disabled, you can no longer use the SPI Active using JTAG Bridge (New) or SPI Active x8 using JTAG Bridge (New) to program the configuration bitstream into the flash. After disabling JTAG, you will need to use either SPI Active (serial/parallel) or Flash Programming using Microcontroller to program the flash. See Flash Programming Using Microcontroller on page 36 for more information.

Table 20: 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.
SPI Active x8 using JTAG Bridge (New)	A improved version of the SPI Active x8 using JTAG Bridge (Legacy) mode with a faster flash programming time.

> **Learn more:** Refer to Program using a JTAG Bridge (New) on page 60 for more information.

Figure 28: Flash Programming Board Setup

SPI Active (Serial/Parallel)
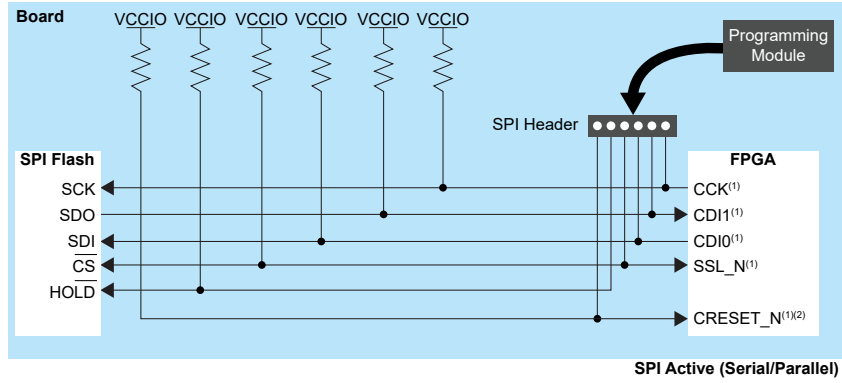
[1] The external pull-up is optional unless required by an external load.
[2] Be sure to hold CRESET_N low to prevent signal contention during SPI flash programming.
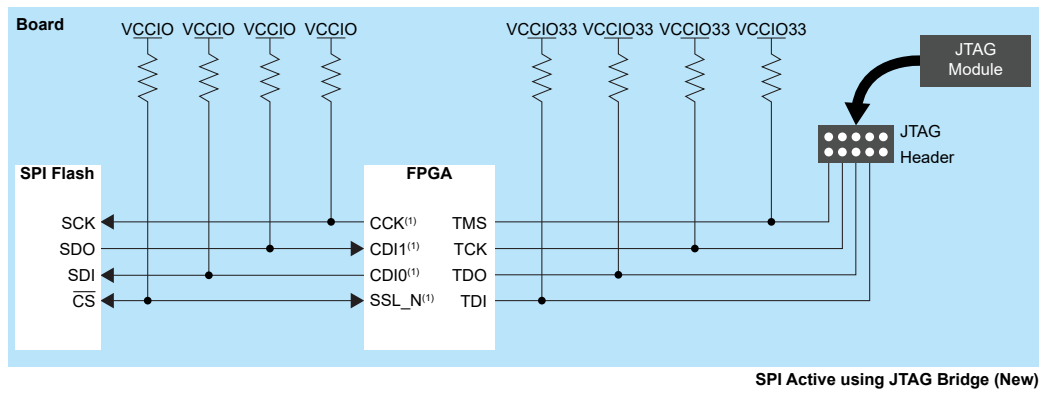
SPI Active using JTAG Bridge (New)

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

SPI Active x8 using JTAG Bridge (New)
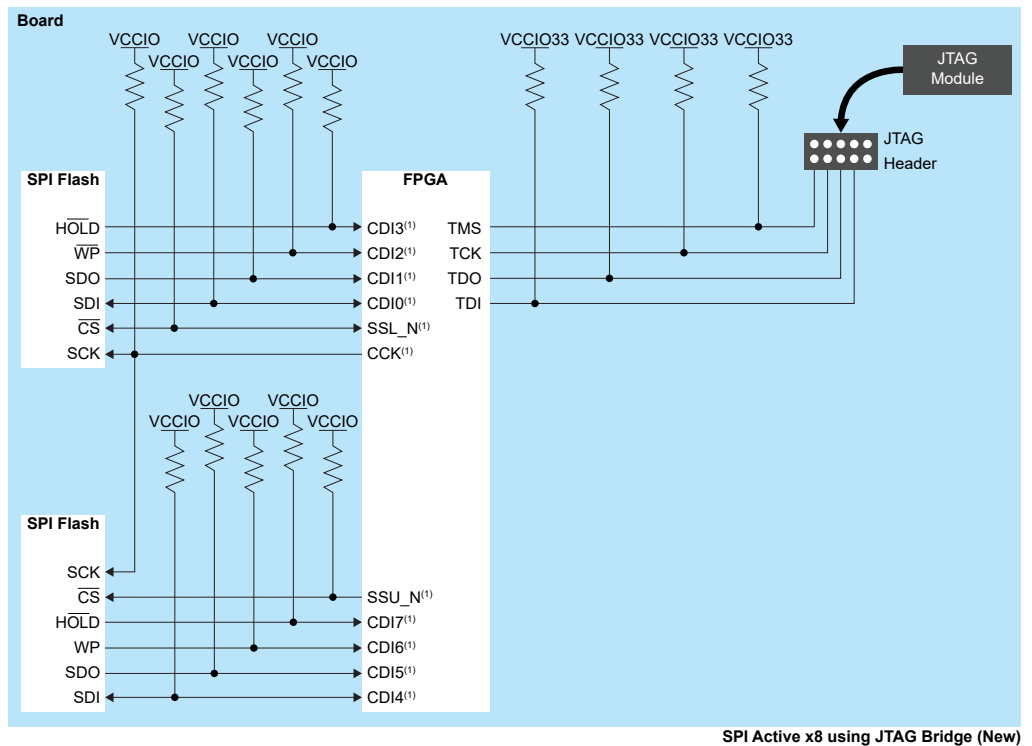
[1] The external pull-up is optional unless required by an external load.

! **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 21: 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 perminantly protected.

Flash Programming Using Microcontroller

After JTAG has been disabled, you can program the internal SPI flash through the SPI bus with a microprocessor.

i **Note:** The Microprocessor can be an off-chip device, such as a programming cable run through the header connection.

To start updating the internal flash, use the microprocessor to:

1. Drive `CRESET_N` low
2. Drive `CS/SSL_N` low

3. Begin programming the internal flash.

Figure 29: SPI Flash Programming with Microprocessor
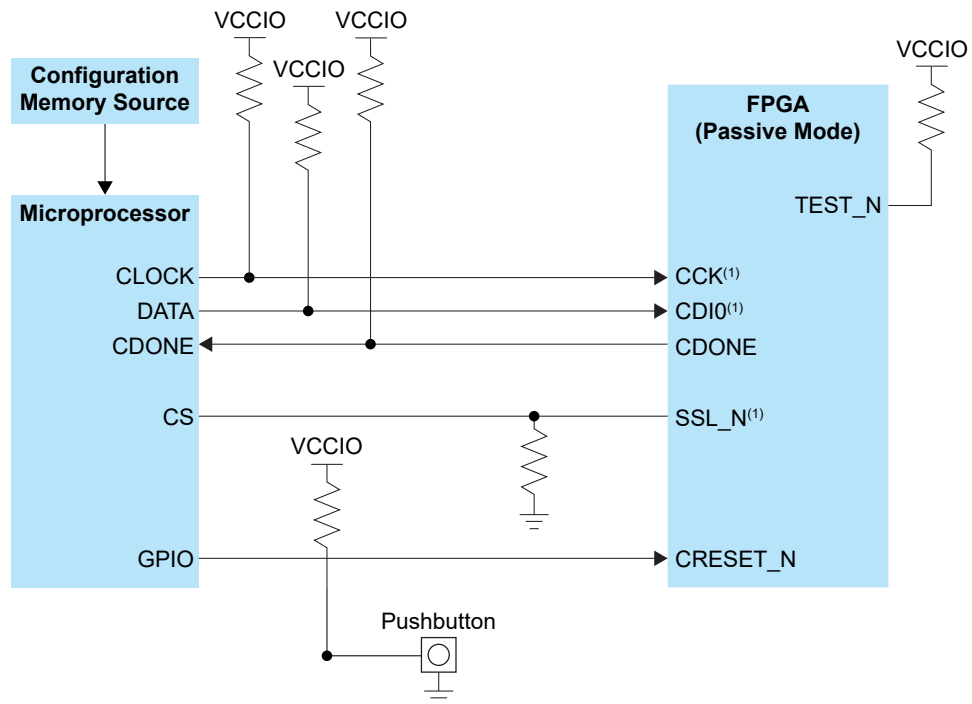
Note:
¹ The external pull-up is optional unless required by an external load.
² SSL_N must be dedicated to microprocessor only.

Note: Refer to Using FPGA, MCU, and SPI Flash Devices Together on page 31 for more information.

Power Sequence

> ℹ **Note:** For information regarding the power-up sequence, power-down sequence, and power supply current transient, refer to the **Titanium Data Sheets** in Support Center.

Power Up Configuration Circuitry Recommendation

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

- Supervisor integrated circuit (IC)
- Microprocessor or microcontroller

> ⚠ **Important:** Do not drive a signal to any Titanium I/O pins before the Titanium FPGA is powered up.

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 30: Supervisor IC Power Up Circuitry
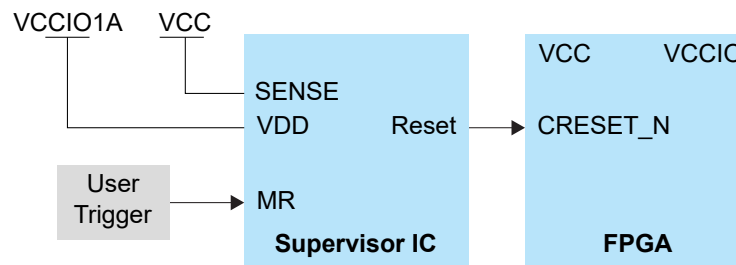
> ℹ **Note:** The user trigger (pushbutton, FTDI module) must be connected to the `MR` pin of the supervisor IC.

Microprocessor or Microcontroller Circuitry Example

Figure 31: Microprocessor Power Up Circuitry

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
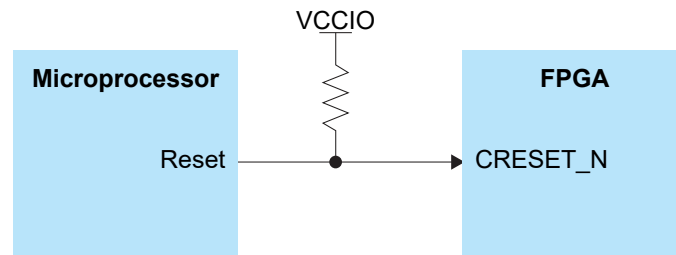
The microprocessor or microcontroller must hold the CRESET_N pin low more than the required t_{CRESET_N} duration.

Configuration Sequence

The Titanium 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 `SSL_N` pin. Like other programmable I/O pins, the `SSL_N` pin has an internal pull-up resistor.

> **Learn more:** Refer to the Titanium data sheet for the pulse width requirements of `CRESET_N`.

2. If the `SSL_N` pin is sampled as a logic 1 (high), the FPGA configures using the SPI active configuration interface.
3. If the `SSL_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 32: Configuration Flow Diagram
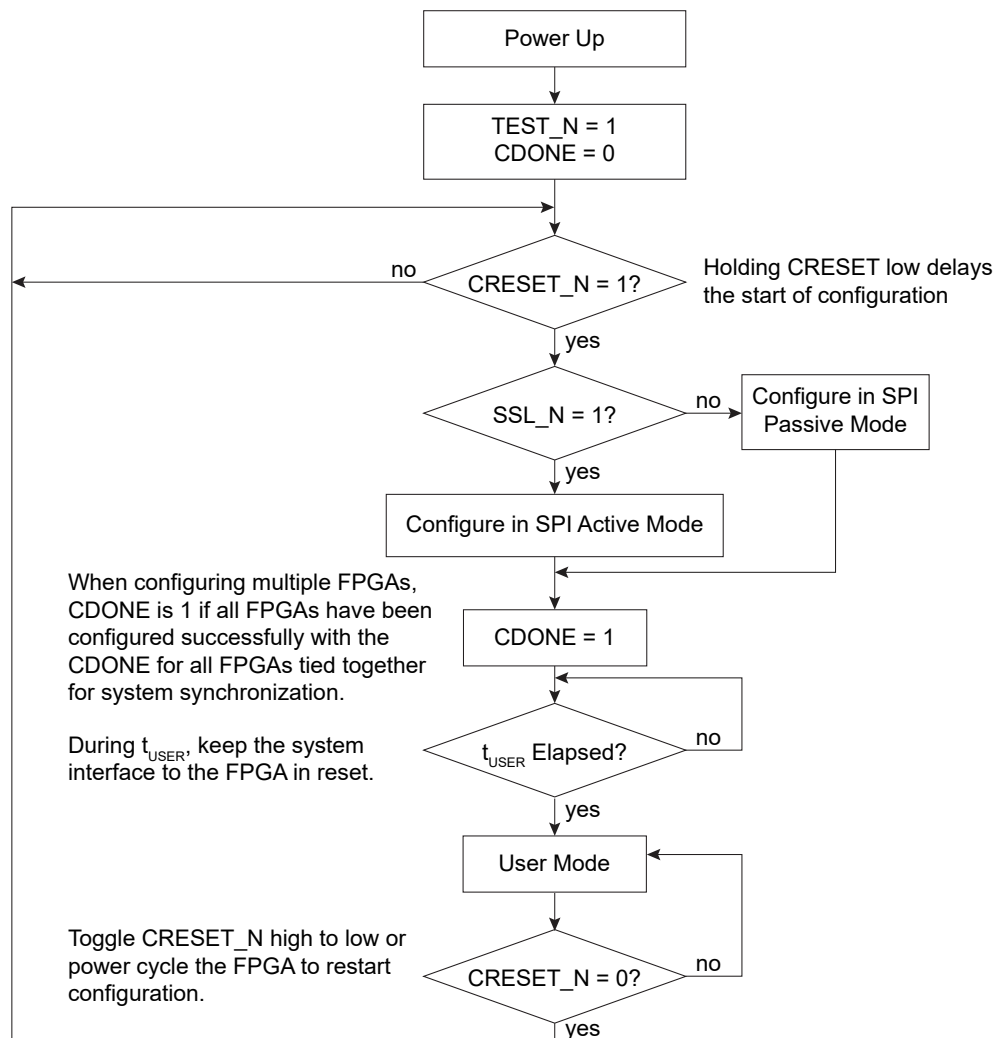
Support for Multiple Images

When powered up in SPI active mode, the Titanium 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 55 for more information.

During multi-image configuration, the Titanium 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 Titanium 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 33: Configuration Setup for Multiple Images
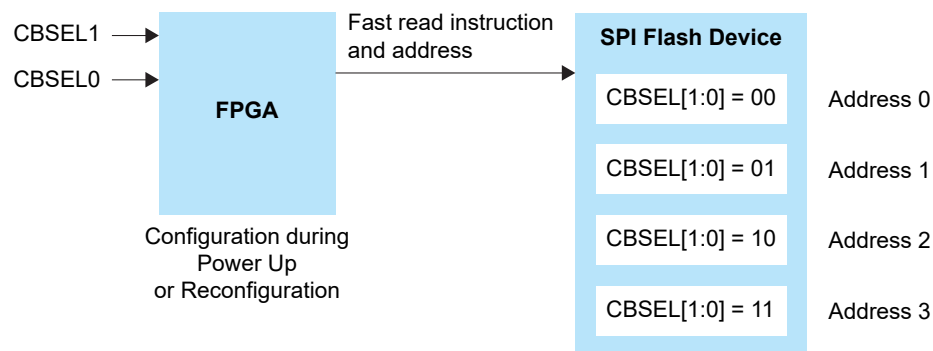
Note: Efinix recommends you to store image 0 at the `24'h000000` address as Titanium FPGAs always start searching for a valid bit stream from the `24'h000000` address.

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
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 valid, but the FPGA fails in configuration with `CDONE` = 1 and `NSTATUS` = 0.

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 Titanium 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 Titanium 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 F100 and F100S3F2).

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.

Figure 34: Serial Daisy Chain Configuration Interface Example

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
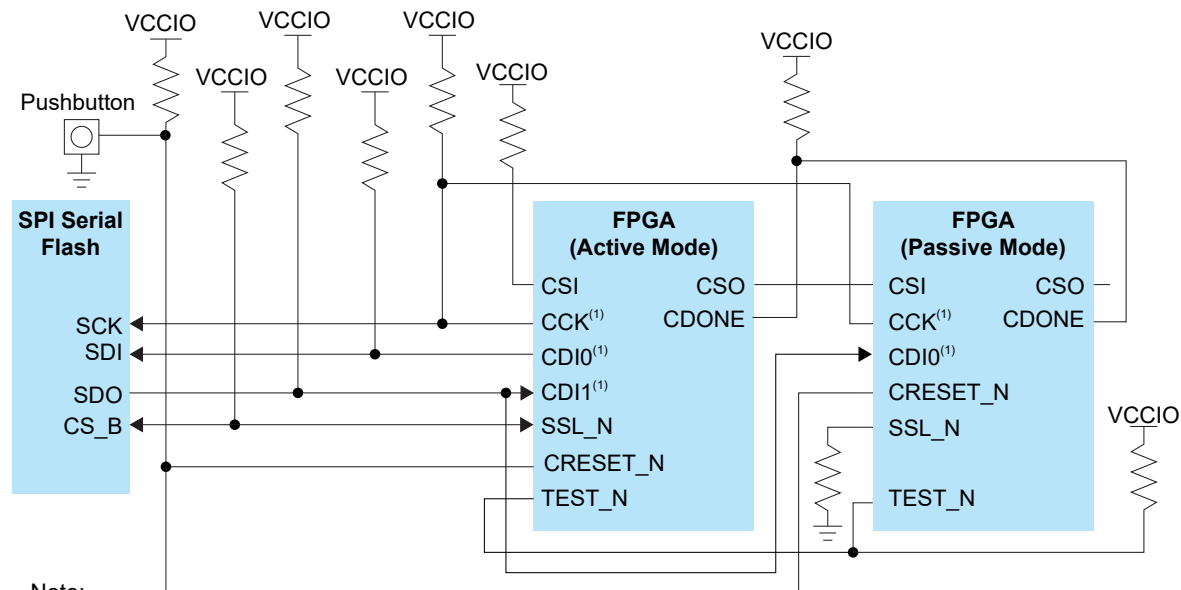
Note:
¹ The external pull-up is optional unless required by an external load.

Figure 35: Parallel Daisy Chain Configuration (x4) Interface Example

See **Resistors in Configuration Circuitry** on page 46 for the resistor values.
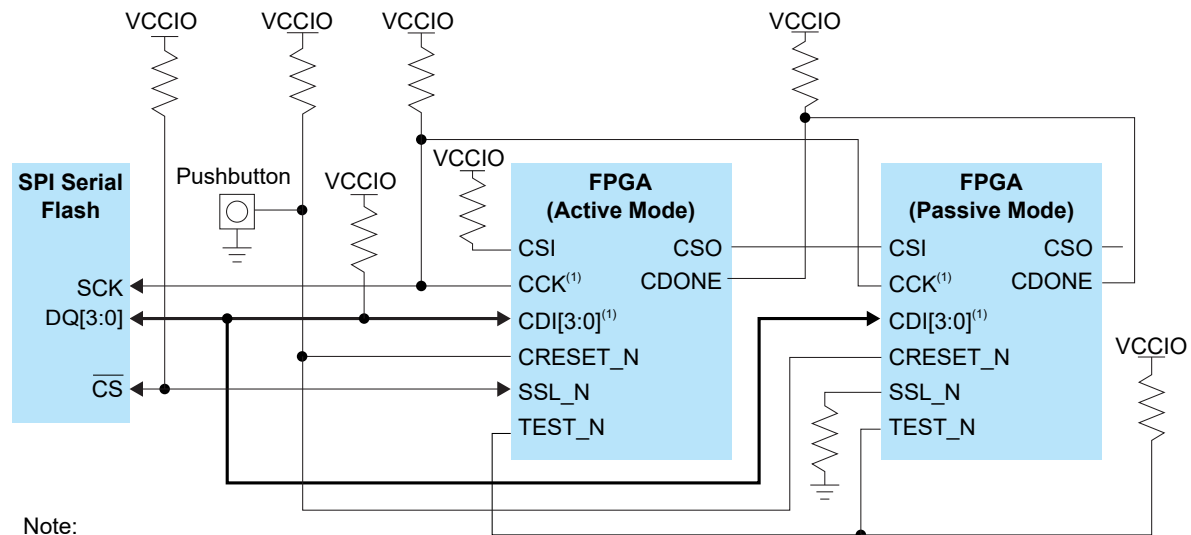
Note:
¹ The external pull-up is optional unless required by an external load.

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.

Figure 36: Serial Daisy Chain Configuration (x32 Passive Mode) Interface Example

See **Resistors in Configuration Circuitry** on page 46 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.

> **(!) 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 kΩ pull-up and pull-down resistors do not significantly affect either the single-ended or differential signals.

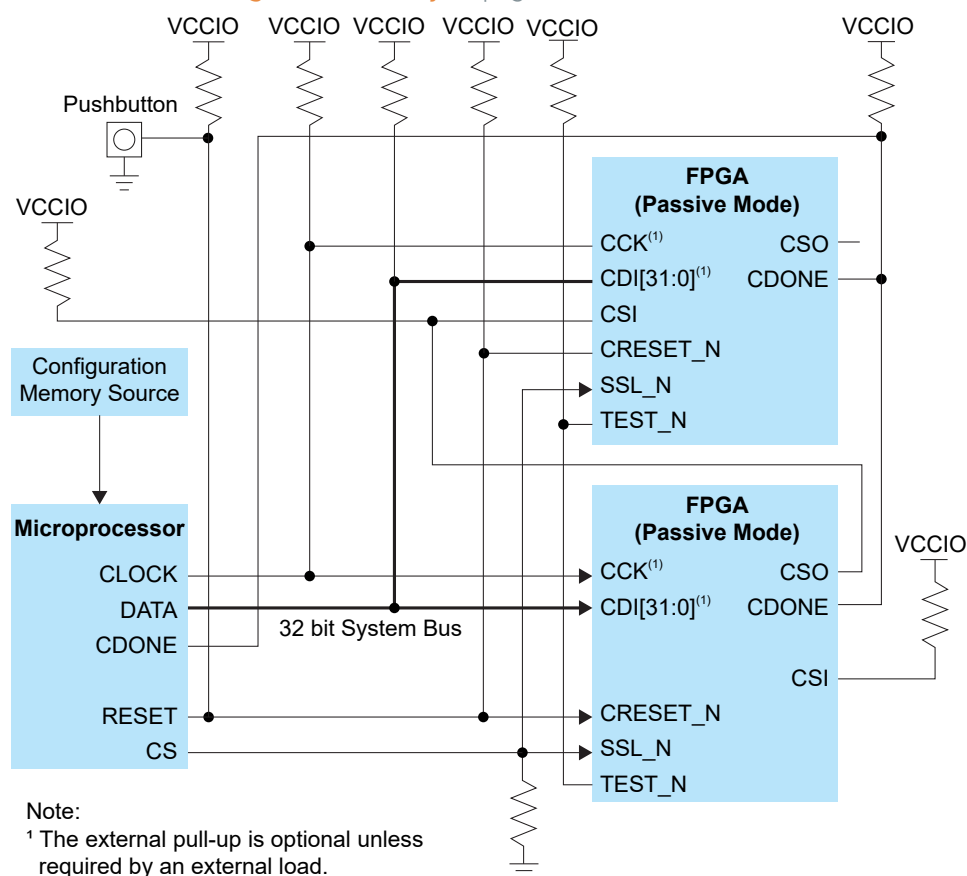Alternatively, you can calculate your own pull-up or pull-down resistance, R_{USER}, shown in the following sections.

> **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 Titanium Data Sheet in **Support Center**.

User-Defined Pull-Up Resistor Values

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

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:

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

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

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

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

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

> ⚠ **Important:** Refer to your Titanium FPGA data sheet for the timing specifications for these parameters.

Table 22: 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.[4][5] Test condition at 10 kΩ pull-up resistance and 10 pF output loading on CDONE pin.

Table 23: Active Mode

Symbol	Parameter
f_{MAX_M}	Active mode internal configuration clock frequency.
$f_{MAX\_M\_EXTCLK}$	Active mode external configuration clock frequency.
t_{SU}	Setup time. Test condition at 1.8 V I/O standard and 0 pF output loading.
t_H	Hold time. Test condition at 1.8 V I/O standard and 0 pF output loading.

Table 24: 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.

---

[4] The FPGA may go into user mode before t_{USER} has elapsed. However, Efinix recommends that you keep the system interface to the FPGA in reset until t_{USER} has elapsed.

[5] For JTAG programming, the min t_{USER} configuration time is required after CDONE goes high and the FPGA receives the ENTERUSER instruction from the JTAG host (TAP controller in UPDATE_IR state).

Table 25: 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

Titanium FPGAs support an SPI flash memory interface for active mode configuration. Use these guidelines to help choose the correct flash device for your Titanium 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: Titanium FPGA Bitstream Size** on page 4).
  - *Multiple images*—Find the configuration bits a single image uses (refer to **Table 2: Bitstream Size for Multiple Images** on page 5). 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 Ti60 FPGAs uses $(3 \times 13{,}686{,}496) + (2{,}048 \times (3 - 1)) = 41{,}063{,}584$ bits.

- **Configuration Bus Width**—Determine the supported configuration bus width for the SPI flash device in Supported Configuration Modes section of the **Titanium FPGA Data Sheets**.

- **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: Clock Setting Recommendations** on page 18.

- **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

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 52 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 37: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections

Refer to the voltage level translator data sheet for the capacitor values.
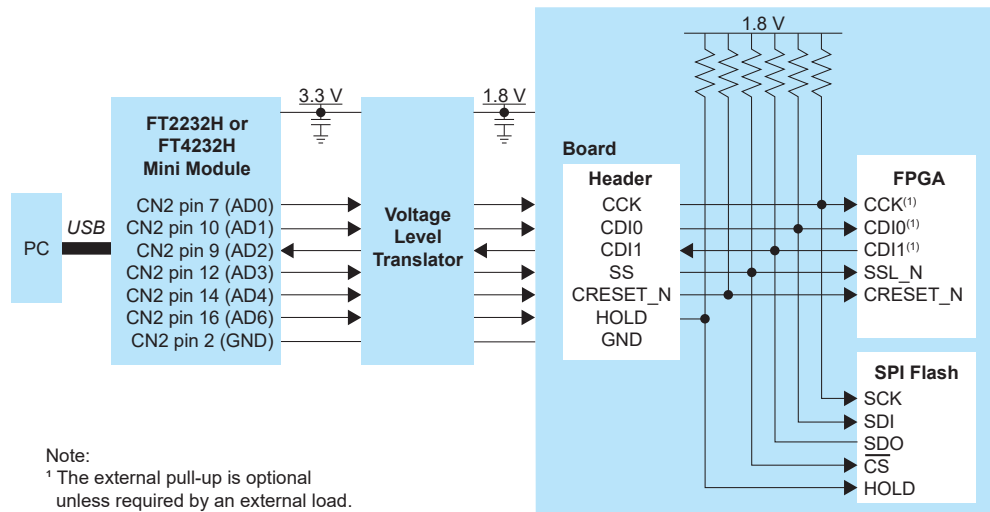
JTAG Programming Connections

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 Titanium FPGAs with a JTAG Mini Module, use this connection:

Figure 38: Connect FT2232 Mini-Module to JTAG Pins

Figure 39: Example of FT2232 Mini-Module Connected to JTAG Pins Using 1.8V

Figure 40: Connect FT4232 Mini-Module to JTAG Pins
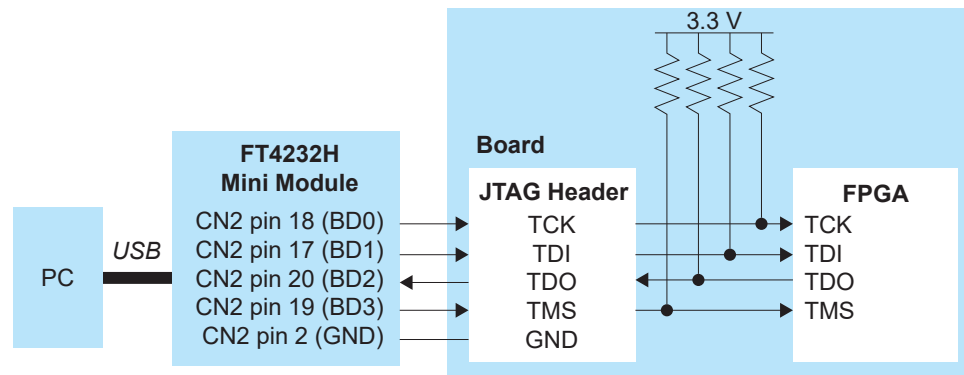
Figure 41: Example of FT4232 Mini-Module Connected to JTAG Pins Using 1.8V
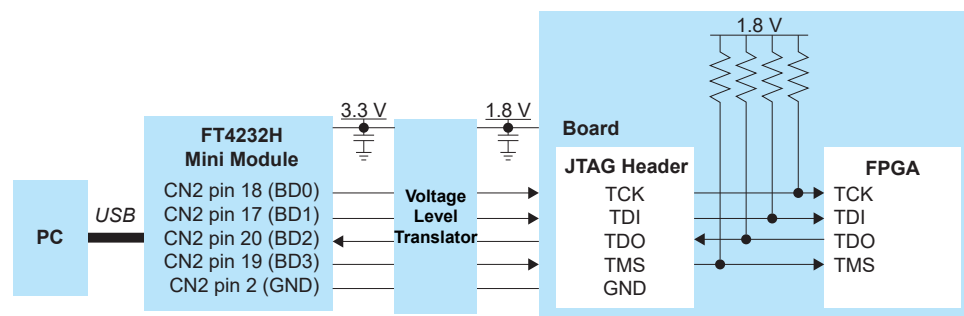
Using the Efinity Programmer

The Efinity® software has a Programmer you use to configure Titanium FPGAs. You can run the Programmer using the GUI or with the command line.

Learn more: Titanium FPGAs have built-in security features to help you protect your intellectual property and to prevent t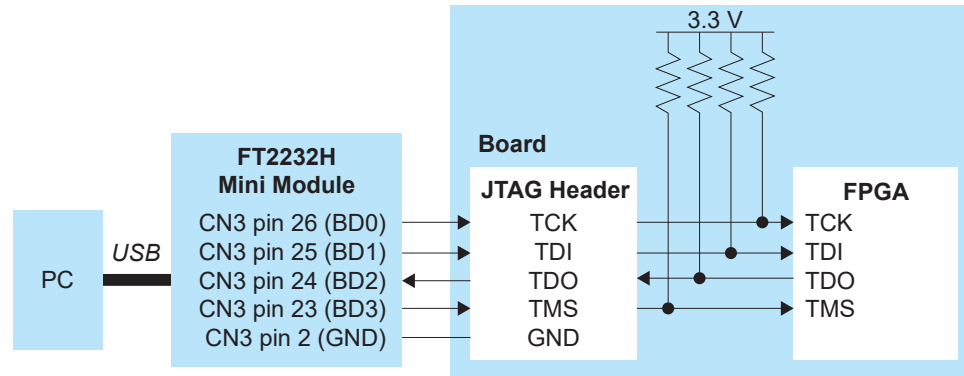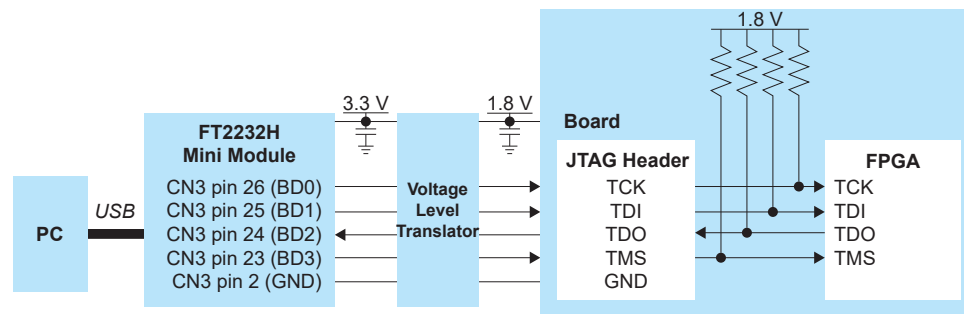ampering. Refer to the Securing Titanium Bitstreams section of the **Efinity Software User Guide** for more information about using the security features.

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 65.

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.

Bitstream Compression

When you generate a bitstream for Titanium FPGAs, the Efinity® software compresses the bitstream by default. This compression results in a bitstream size that is about half of the maximum size.

> ⚠ **Important:** If you are using the Titanium or security features (AES-256 encryption and/or asymmetric authentication), the software cannot compress the bitstream. Therefore, compression is disabled when you use these features.

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 54.

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

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 <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 27: 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 55
	Internal reconfiguration	Up to 4	Program Multiple Images (Internal Reconfiguration) on page 56
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors	Program a Daisy Chain on page 57
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	Program Multiple Images (Bitstream and Data) on page 57

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**.

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 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.

> (i) **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.

Titanium Family JTAG Device IDs

The following table lists the Titanium JTAG device IDs.

Table 28: Titanium JTAG Device IDs

FPGA	Package	JTAG Device ID
Ti35	All	0x10661A79
Ti60	All	0x10660A79
Ti85	All	0x006C2A79
Ti90	J361, J484, G400, G529	0x00691A79
	L484	0x00688A79
Ti120	J361, J484, G400, G529	0x00692A79
	L484	0x0068CA79
Ti135	All	0x006C0A79
Ti165	All	0x006A1A79
Ti180	M484	0x00680A79
	J361, N484, J484D1 G400, G529	0x00690A79
	L484	0x00684A79
Ti240	All	0x006A2A79
Ti375	All	0x006A0A79

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 Titanium 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="0x10661A79" ir_width="5" istr_code="11000" />
    <device chip_num="2" id_code="0x10661A79" ir_width="5" istr_code="11000" />
    <device chip_num="3" id_code="0x10661A79" ir_width="5" istr_code="11000" />
</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 Titanium FPGAs, use 11000 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 (New)

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 (New) mode, is an improved version of the legacy SPI Active using JTAG Bridge mode, and is available in the Efinity software v2023.2 and higher. This mode is substantially faster than the legacy mode and 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.

Notice: If you would like to incorporate the RTL files for the new and improved flash loader into your own design, the files are located in the *<Efinity directory>***/pgm/rtl/spiloaderv2** directory.

The Titanium**.bit** files include a custom JTAG USERCODE in the bitstream:
- Single flash **.bit** files—0x96C09A03
- Dual flash **.bit** files—0xC07FCFE2

Note: For Titanium FPGAs, the Programmer automatically loads the **.bit** file based on the FPGA target. The Programmer has separate pre-built **.bit** files for the new JTAG bridge, as well as the legacy JTAG bridge. The files are not interchangeable; therefore, if you are choosing the **.bit** file yourself, make sure that you choose the correct one for the JTAG bridge mode you are using. Legacy ones are in the *<Efinity version>***/pgm/fli/titanium/legacy** directory.

For Trion FPGAs, you need to specify the pre-built file to use.

Note: Efinix strongly recommends that you use the default new mode for JTAG bridge programming. However, if you would like guidance on using the legacy JTAG bridge, please refer to the **Efinity Software User Guide**.

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 (New)** or **SPI Active x8 using JTAG Bridge (New)** programming mode.
4. Turn on the **Auto configure JTAG Bridge Image** option.

   For Titanium FPGAs, the Programmer automatically loads the **.bit** file. Skip step 5 if you want to use the pre-loaded **.bit** file.
5. (Optional) Specify the **.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.

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

> **Important:** If you are using the Titanium RSA bitstream authentication security feature, you need to use a signed **.bit** file. Copy the bundled **.bit** file from *<Efinity version>*/**pgm/fli/titanium** to another directory and sign it. Then point to the signed **.bit** file in the Programmer. You can also create your own **.bit** file if you prefer.
>
> Refer to Using the Efinity Bitstream Security Key Generator for information on signing existing **.bit** files.

Efinix strongly recommends you to disable JTAG if you are using the security features to achieve the highest security level. While disabled, you can still programming the flash with JTAG Bridge by connecting to a soft JTAG tap IP and four GPIOs.

JTAG Programming with FTDI Chip Hardware

These instructions describe how to program Titanium FPGAs using the FTDI Chip FT2232H and FT4232H Mini Modules. Efinix® has tested the hardware for use with Titanium 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_new`, or `jtag_bridge_x8_new`[6] (new mode, see Program using a JTAG Bridge (New) on page 60)
- `jtag_bridge`, or `jtag_bridge_x8`[7] (legacy mode, see Program using a JTAG Bridge (Legacy))

> **Note:** To use the JTAG bridge modes, you must have already configured the FPGA with the JTAG SPI flash loader.
>
> The Efinity software v2023.2 and higher includes pre-built flash loader **.bit** files in *<Efinity installation directory>*/**pgm/fli/**<family>.
>
> Refer to the JTAG SPI Flash Loader Core User Guide for information on using the legacy flash loader.

---

[6] The `jtag_bridge_x8_new` mode is only supported in some Titanium FPGAs. Refer to the data sheet for the modes your FPGA supports.
[7] The `jtag_bridge_x8` mode is only supported in some Titanium FPGA s. Refer to the data sheet for the modes your FPGA supports.

> ⚠ **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
4232h	Titanium Ti60 BGA225 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`.
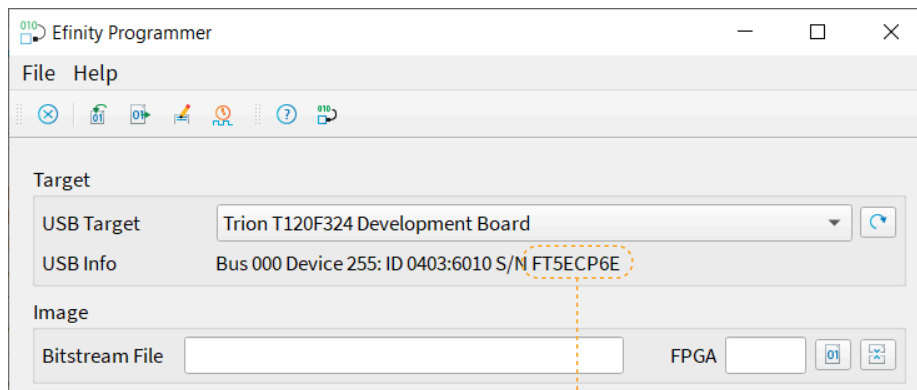
The string after S/N is
the FTDI serial number

<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: Titanium Ti60 F225 Development Board as the only board attached to your computer, use:

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

Example: Titanium Ti60 F225 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:4232h:FT5ECP6E/1
    --aurl ftdi://ftdi:4232h: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: Titanium 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: Titanium Ti60 F225 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:4232h:FT5ECP6E/1 --aurl ftdi://ftdi:4232h: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
  - `jtag_bridge_x8`—SPI active x8 using JTAG bridge mode (used with two flash devices)[8]

  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.

---

[8] Used with two flash devices. Only supported in some Titanium 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.
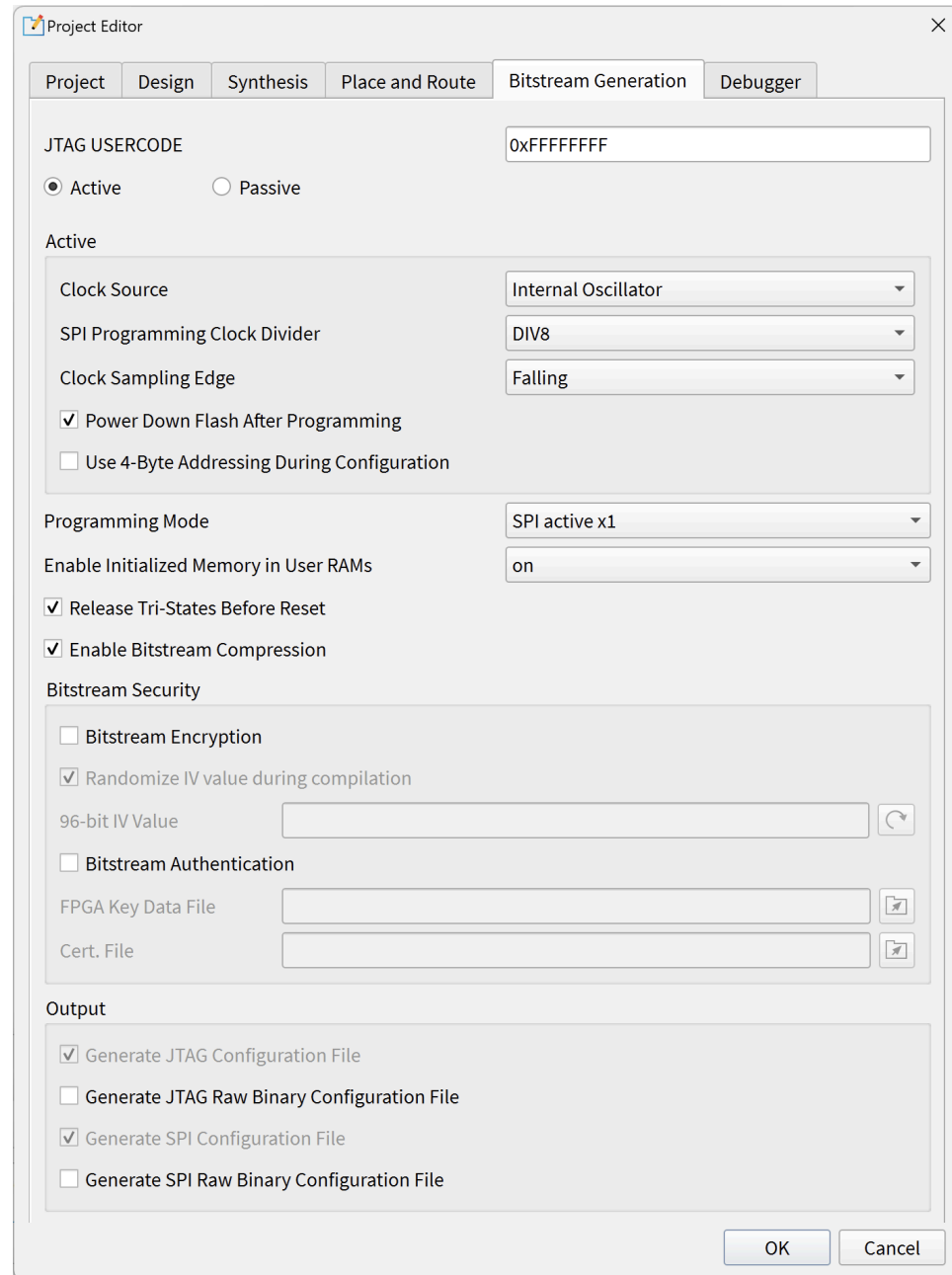
Table 29: Project-Specific Programming Options

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.
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**.
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	(Titanium only). When you turn this option on, the control block issues 4-byte addresses when it configures the FPGA. This option is not supported for Ti35 or Ti60 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. **on**: The bitstream has initialized memory. **off**: The bitstream does not have initialized memory. **smart**: For the Titanium family, this option gives a slightly smaller bitstream. 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
Enable Bitstream Compression	When turned on (default), the software compresses the bitstream. If you choose **Bitstream Encryption** or **Bitstream Authentication**, this option is turned off and disabled because you cannot compress a bitstream and use the security features simultaneously.
Bitstream Encryption	On: The software generates an encrypted bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box. Off: (default) The software generates a plaintext bitstream.
Randomize IV value during compilation	This option is used with bitstream encryption. The encryption/decryption uses a 96-bit initial vector (IV). If you want the software to generate a random IV for every compilation, leave this option turned on. If you want to specify an IV, turn this option off and specify the value in the **96-bit IV Value** box. On (default): Let the software generate the IV value. (The bitstream will be different every time you compile, even if nothing has changed in your design.) Off: The software does not generate the IV value; the user will specify it in the **96-bit IV Value** box. (If nothing has changed in your design, when you recompile, the bitstream remains the same)
96-bit IV Value	Click the refresh button next to this box to generate a random IV value. You can also enter a value you generate with another program.
Bitstream Authentication	On: The software generates a signed bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box and the RSA private key (**.pem**) file in the **RSA Private Key** box. Off: (default) The software generates an unsigned bitstream.
FPGA Key Data File	Specify the location and name of the **.bin** file you generated with the Efinity Bitstream Security Key Generator.
RSA Private Key	Specify the location and name of the RSA private key file (**.pem**).
Generate JTAG configuration file	On (always): Generate a **.bit** file for JTAG configuration.
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 (always): Generate a **.hex** file for SPI programming.
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 42: Setting Programming Options

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

Configuration Status Register

Titanium FPGAs have a configuration status register. You can use the Efinity Programmer to monitor the values in this register to help debug confuguration issues. View the register values in the **Advanced Device Configuration Status** dialog box, which you open by clicking the button of the same name.

Table 30: Configuration Status Register

Name	Description
IN_USER	0: The FPGA is not in user mode.
	1: The FPGA is in user mode. IN_USER waits for all internal resets and tri-states to be released before it goes high.
	ⓘ **Note:** This bit is not supported in Ti60ES FPGAs.
CDONE	Configuration done, has the same value as the CDONE output pin.
	0: The FPGA is not configured.
	1: Configuation is complete.
NSTATUS	Configuration status, has the same value as the active-low NSTATUS output pin if the NSTATUS pin is not driven by user when the FPGA is in user mode.
	0: Indicates that the FPGA received a bitstream that was targeted for a different configuration mode or width, or a CRC error is detected during configuration. NSTATUS can also go low if there is a mismatch between the bitstream and the FPGA encryption/authentication keys.
	1: During configuration, indicates that the FPGA is in configuration mode.
CRC32_ERROR_CORE	0: No CRC errors were detected in the core configuration bits.
	1: One or more CRC errors were detected in the core configuration bits.
RMUPD_ERROR	0: No errors occurred during remote update.
	1: An error occurred during remote update configuration. Has the same value as the remote update error status signal sent to the core fabric.
CONFIG_END	0: Configuration is not complete.
	1: Configuration completed (whether successful or not).
SYNC_PAT_FOUND	0: Indicates that the FPGA is not receiving the expected synchronization pattern at start of the bitstream. Check for board or power issues.
	1: Indicates that the FPGA detected a synchronization pattern at start of the bitstream., and the clock and data connections to the FPGA are acceptable. Any configuration problems are likely digital or logical in nature. After successful configuration the status will return to 0.
SEU_ERROR	0: No SEU detection errors were found.
	1: An SEU detection error was found when reading back the SEU CRAM. Has the same value as the SEU detection error status signal to the core fabric.
CRC32_ERROR_PERIPH	0: No CRC errors were detected in the interface configuration bits.
	1: One or more CRC errors were detected in the interface configuration bits.

Name	Description
AES256_PASS	For an encrypted bitstream: 0: Decryption failed. The encryption keys used in to program the fuses may not match the ones used to encrypt the bitstream 1: The encrypted bitstream was decrypted successfully. If the bitstream is not encrypted, this register is always a 1. ⓘ **Note:** This bit is not supported in Ti60ES FPGAs.
RSA_PASS	When using RSA authentication: 0: The signature check failed. The RSA keys used to program the fuses may not match the ones used to sign the bitstream in the Efinity project. 1: The bitstream signature was verified successfully If RSA authentication is not used, this register is always a 1. ⓘ **Note:** This bit is not supported in Ti60ES FPGAs.
AES_ACTIVE	After the FPGA is configured, you can check this status bit for encryption: 0: AES is disabled in the current design. 1: AES is enabled in the current design.
RSA_ACTIVE	After the FPGA is configured, you can check this status bit for authentication: 0: RSA is disabled in the current device. 1: RSA is enabled in the current device.
USERCODE	Displays the 32-bit hex JTAG USERCODE.

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.

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 31: Monitoring with the Efinity Programmer

CDONE	NSTATUS	Programmer Message	Description
0	0	Failure to configure was detected	Configuration failed. This may be caused by: - The configured bitstream is for a different configuration mode or width, - CRC error is detected during configuration, - There is a mismatch between the bitstream and the FPGA encryption/authentication key
0	1	Programming ...	The FPGA is in configuration mode.
1	0	–	The FPGA is not functioning as expected. Bitstream transmission is completed but failed may be due to the following: - Wrong device ID - Remote update failure
1	1	Device is in user mode!	The FPGA is functioning correctly according to the user design.

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:

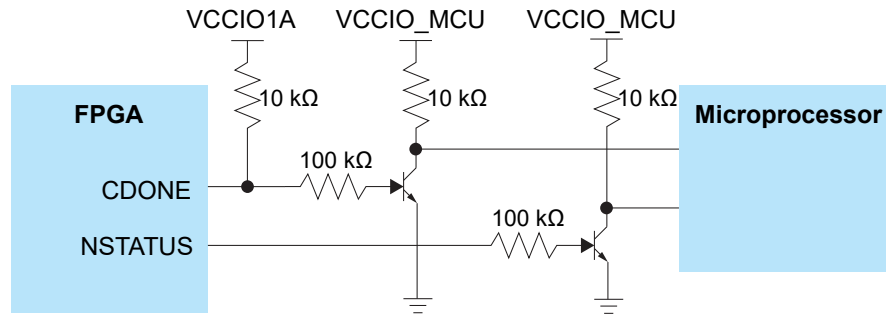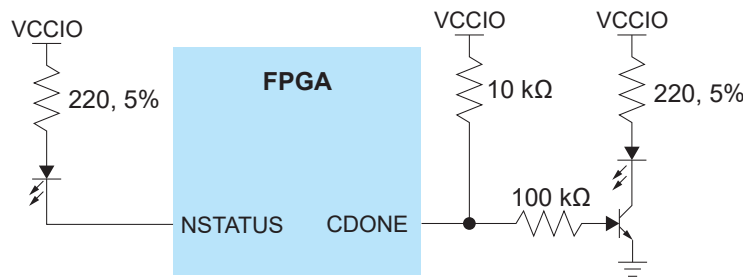*Figure 43: Connect CDONE and NSTATUS to a Microcontroller*

Figure 44: 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 32: Monitoring with a Microcontroller or LEDs

CRESET_N	CDONE	NSTATUS	Description
1	0	0	Configuration failed. This may be caused by: • The configured bitstream is for a different configuration mode or width • CRC error is detected during configuration
1	0	1	The FPGA is in configuration mode.
1	1	0	The FPGA is not functioning as expected. Bitstream transmission is completed but failed may be due to the following: • Wrong device ID • Remote update failure
1	1	1	The FPGA is in transition from configuration mode to user mode

Installing USB Drivers

To program Titanium 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.

For your own development board, Efinix suggests using the FTDI Chip FT2232H or FT4232H Mini Modules for JTAG programming Titanium 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 33: 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
> 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**. (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 Titanium 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 Titanium 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** or **SPI Active x8 using JTAG Bridge** mode to program the user image into the flash device. The Programmer sends the command through the Titanium 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 Titanium FPGA's `CRESET_N` signal to trigger reconfiguration using the new flash image.

For the SPI Active x8 using JTAG Bridge mode, choose **TRUE** for the **Enable dual flash programming** option.

Figure 45: SPI Flash Programming Flow
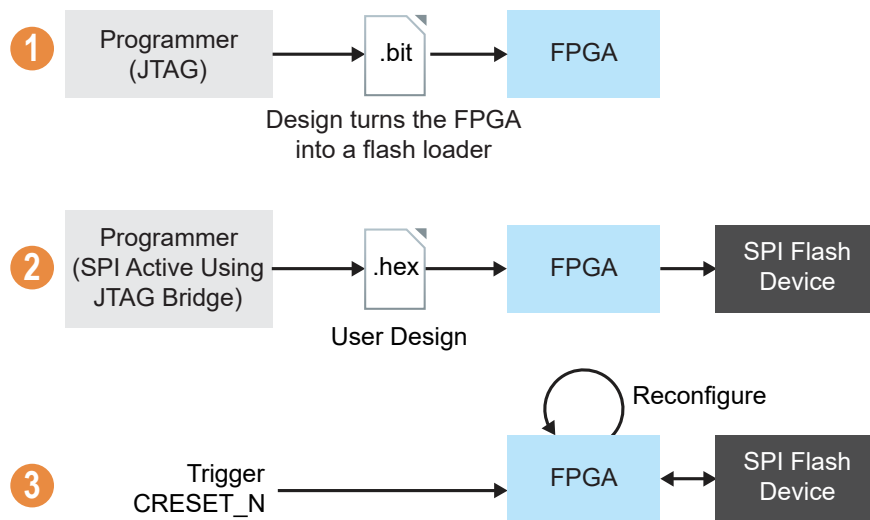
Learn more: For more information on using the JTAG SPI Flash Loader and the **SPI Active using JTAG Bridge** or **SPI Active x8 using JTAG Bridge** programming mode, refer to the JTAG SPI Flash Loader Core User Guide.

Revision History

Table 34: Revision History

Date	Version	Description
March 2025	2.9	Added Figure 6: SPI Active (x1) Configuration on page 15
		Added Figure 7: SPI Active (x1) Configuration (Detailed View) on page 15
		Added Figure 15: SPI Passive (x1, Mode 3) Configuration on page 24
		Added Figure 16: SPI Passive (x1, Mode 3) Configuration (Detailed View) on page 24
		Added Figure 19: JTAG Programming Waveform on page 28
		Added Figure 20: JTAG Programming Waveform (Detailed View) on page 28
		Correct typo in Figure 32: Configuration Flow Diagram on page 40.
		Added note to Program Using a JTAG Chain on page 59. (DOC-2310)
		Updated Table 28: Titanium JTAG Device IDs on page 58 for J484D1.
January 2025	2.8	Fixed typo in title of referenced document. (DOC-2302)
		Added Table 21: Status Register Protect Bits on page 36. (DOC-2254)

Date	Version	Description
December 2024	2.7	Updated Table 31: Monitoring with the Efinity Programmer on page 70. (DOC-2256)
		Updated entry for `SYNC_PAT_FOUND` in Table 30: Configuration Status Register on page 68. (DOC-2181)
		Added SPI Active Mode for SIP Packages on page 16 (DOC-2177)
		Added SPI Passive Mode for SIP Packages (DOC-2177)
		Updated Figure 21: 1.8 V JTAG Connection on page 29. (DOC-2171)
		Updated JTAG_VCCIO_SEL description in Table 3: Dedicated Configuration Pins on page 7. (DOC-2171)
		Updated Figure 43: Connect CDONE and NSTATUS to a Microcontroller on page 71 and Figure 44: Connect CDONE and NSTATUS to LEDs on page 71. (DOC-2165)
		Updated Figure 28: Flash Programming Board Setup on page 35. (DOC-1408)
		Changed column name from Pins to Configuration Functions in Table 4: Dual-Purpose Configuration Pins on page 8. (DOC-2038)
		Added note after Table 4: Dual-Purpose Configuration Pins on page 8 directing the reader to the pinout file. (DOC-2038)
		Updated Table 1: Titanium FPGA Bitstream Size on page 4 for Ti85, Ti135, Ti165, Ti240, and Ti375. (DOC-1809 and DOC-2035)
		Updated JTAG device IDs for Ti375, Ti135, and Ti85: (DOC-1851)
		Updated Table 12: Bitstream Bits in Series on page 21 in SPI Passive Mode to clarify Byte Order and Bit Order. (DOC-1520)
		Updated various tables in SPI Passive Mode on page 20 to clarify Bit Cycles and Byte Cycles. (DOC-1520)
		Link added to Ti60 Data Sheet for details on t$_{DMIN}$. (DOC-1525)
		Updated Table 2: Bitstream Size for Multiple Images on page 5 for Ti165, Ti240, and Ti375. (DOC-1809)
		Note added in relation to Table 10: Clock Setting Recommendations on page 18. (DOC-1865)
		Updated Figure 2: Active (x1) on page 12. (DOC-1865)
		Updated Figure 37: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections on page 50. (DOC-1910)
		Updated Figure 38: Connect FT2232 Mini-Module to JTAG Pins on page 51. (DOC-1910)
		Added Figure 39: Example of FT2232 Mini-Module Connected to JTAG Pins Using 1.8V on page 51. (DOC-1910)
		Updated Figure 40: Connect FT4232 Mini-Module to JTAG Pins on page 51. (DOC-1910)
		Added Figure 41: Example of FT4232 Mini-Module Connected to JTAG Pins Using 1.8V on page 51. (DOC-1910)
		Added Figure 9: Configuration with Internal Flash on page 16
		Updated Figure 10: Cascading an External SPI Flash for User Data Storage on page 17
		Updated Table 11: Supported SPI Flash Address Width for Configuration on page 19
		Updated Figure 37: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections on page 50
		Updated Figure 28: Flash Programming Board Setup on page 35
March 2024	2.6	Removed JTAG IDs: (DOC-1760)
		• Ti90/Ti120: M361, M484, and F529
		• Ti180: M361 and F529

Date	Version	Description
February 2024	2.5	Updated figures Flash Programming Board Setup and SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections. (DOC-1256)
		Updated on table Dedicated Configuration Pins and Dual-Purpose Configuration Pins. (DOC-1490)
		Added note in SPI Passive Mode section. Added statement and figure SPI Clocking and Phase Modes Diagram, and table SPI Interface Clocking and Sampling in About SPI Clocking and Sampling section. (DOC-1525)
		Added new sub-topic JTAG Operation. (DOC-1589)
		Removed link and corrected statement in Configuration Bus Width of Selecting the Right SPI Flash Device section. (DOC-1670)
		Updated notes in figure SPI Passive Mode (x1) Timing Sequence. (DOC-1690)
		Updated the waveform for figures SPI Active Mode (x1) Timing Sequence, SPI Passive Mode (x1, Mode 3) Timing Sequence, and JTAG Programming Waveform.
		Updated bitstram size in table Titanium FPGA Bitstream Size. Moved Bitstream Size for Multiple Images topic to Bitstream Size section.
		Updated table SPI Hardware Settings.
November 2023	2.4	Added reference note to power-up sequence in data sheet. (DOC-1351)
		Updated pull-up and pull-down resistors in Resistors in Configuration Circuitry. (DOC-1489)
		Updated SPI Flash Programming with FTDI Mini Module Connections. (DOC-1497)
		Updated initial CCK waveform of figure SPI Passive Mode (x1) Timing Sequence.
		Added reference note to power-down sequence and power supply current transient in data sheet.
		Added JTAG device IDs for the Ti135, Ti200, and Ti375.
August 2023	2.3	Added G400 package JTAG Device ID. (DOC-1385)
May 2023	2.2	Added IS25LP128 to list of supported flash devices. (DOC-1247)
February 2023	2.1	Added more description about valid and invalid image. (DOC-1118)
		Corrected user-defined pull-up/pull-down resistor formula. (DOC-1136)
		Updated VQPS power-down sequence note.
		Added connection requirement when unused for VQPS.
December 2022	2.0	Corrected SPI Clock Polarity and Phase Mode table. (DOC-946)
		Added note about not recommending user to pause FPGA configuration. (DOC-944)
		Added VQPS in power up sequence requirement. (DOC-951)
		Updated configuration pins external weak pull-up requirements. (DOC-1035)
		Added description about CRESET_N needs to be deasserted before JTAG configuration begins. (DOC-1069)
		Updated the Verifying Configuration topic. (DOC-1024)
		Added JTAG device IDCODE for J361, J484, G529 packages. (DOC-1084)
September 2022	1.9	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)
		Updated SPI Flash Programming with FTDI mini module diagram by adding voltage level translator. (DOC-844)
		Updated supported flash devices. (DOC-896)
		Updated Project-Based Programming Options topic for new options.

Date	Version	Description
July 2022	1.8	Added topic on SPI clocking and sampling. (DOC-625) Added maximum number of configuration bits for the Ti90, Ti120, and Ti180 FPGAs. Added topic on the flash size needed for multiple images. Corrected SSL_N connection to be bidirectional in active SPI mode connection diagrams. Updated configuration flow diagram. Updated DDR power supplies to match pinout and Efinity software. (DOC-795) Added JTAG device ID for Ti90, Ti120, and Ti180. Corrected configuration time estimation formula. Added 4-byte addressing mode (24 bit address) support for Ti90, Ti120, and Ti180.
April 2022	1.7	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. Updated Power Supply Current Transient and power sequence diagram for Packages without MIPI D-PHY Block. (DOC-761) Re-organized topics about working with bitstreams.
March 2022	1.6	Moved FTDI hardware connection diagrams into Programming Hardware Connections topic. Added topic about external pull-up resistors. (WEB-39) Updated connection diagrams with required pull-up resistors. (DOC-734) Updated power up sequence stating that all supplies must be powered up within 10 ms. (DOC-631)
January 2022	1.5	Improved connection diagrams to show pull-ups to point upwards. (DOC-612) Updated JTAG mode connection diagram. (DOC-546) 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.
December 2021	1.4	With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG. (DOC-638) Corrected the JTAG Device IDs.
November 2021	1.3	Added Configuration Status Register, Bitstream Compression and Exporting to .svf Format topics. Added support for FTDI FT4232H Mini Module. (DOC-597) Added Bitstream Bytes Packed into Parallel Bus for x16, x8, x4, x2, and x1. (DOC-626) Updated power up sequence and added power supply current transient specs. (DOC-643)

Date	Version	Description
November 2021	1.2	Added topic about verifying configuration. (DOC-508, DOC-486)
		Updated the Project-Based Programming Options topic. (DOC-523, DOC-550)
		Added XT25F family to list of supported flash devices. (DOC-529)
		Added connection requirements for unused resources.
		Added note about Titanium Ti35 and Ti60 only supports SPI flash with 3-byte addressing mode for configuration. (DOC-558)
		Added SPI active and passive mode without CSI topics. (DOC-563)
		Updated JTAG mode connection diagram. (DOC-546)
		Added Macronix MX75L and MX75U to supported flash devices. (DOC-573)
		Corrected clock divider settings for SPI active mode. (DOC-595)
August 2021	1.1	Updated power up sequence.
		Added note that FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495)
August 2021	1.0	Initial release.