Speedster7t Configuration User Guide (UG094)

Speedster FPGAs

Achronix®
Data Acceleration

Copyrights, Trademarks and Disclaimers

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

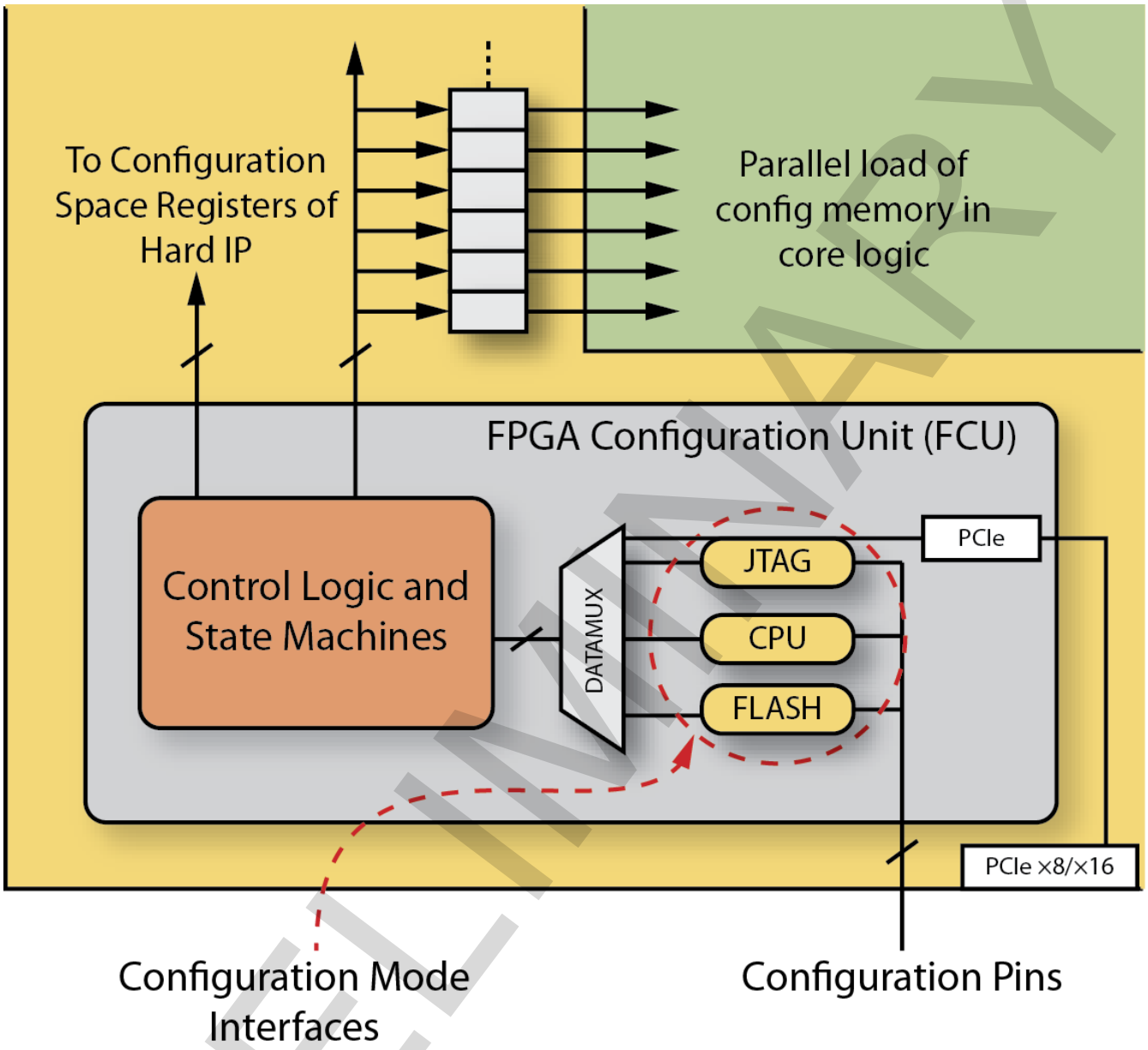
Table of Contents

Chapter - 1: Overview

At startup, Speedster7t FPGAs require configuration by the end user via a bitstream. This bitstream can be programmed through one of four available interfaces in the FPGA configuration unit (FCU).The FPGA configuration unit (FCU) refers to logic that controls the configuration process of the Speedster7t FPGA. It is responsible for receiving data on a variety of core interfaces (depending on a selected programming mode), decoding instructions, and sending configuration bit values to the appropriate destination (core configuration memory, the core's boundary ring configuration memory, FCU registers, etc.). The FCU is also responsible for any core-level housekeeping that happens on reset de-assertion (e.g., clearing of configuration memory) as well as controlling the startup and shutdown sequences that drive resets to the rest of the core as well as CRC checks, SEU mitigation and security.

Data from the configuration pins is brought into the FCU located in the core's boundary logic. Depending on the configuration mode, this data passes through one of four interfaces and is then provided to the control logic and state machines in the FCU. At this point, the data bus is standardized to a common interface (configuration mode independent). This data is processed and propagated to the configuration registers in the core's boundary ring, to the core's configuration memory, or to the hard IP blocks in the FPGA's I/O ring.

Once all of the configuration bits are successfully loaded, the FCU transitions the Speedster7t FPGA into user mode, enabling the user to provide stimuli and enable operation.



42074227-01.2019.12.18

Figure 1: ***Speedster7t Configuration Block***

Chapter - 2: Interface Performance

The table below lists the various configuration interfaces supported by the Speedster7t FPGA and their corresponding maximum operating frequency.

Table 1: ***Configuration Modes and Maximum Frequencies***

Configuration Mode	Maximum Frequency
JTAG	250 MHz
CPU	250 MHz
Serial flash	250 MHz

All of the programming modes and interfaces are capable of running up to 250 MHz at the configuration pins. The FCU and all associated circuitry is also capable of running up to 250 MHz. Since the internal data bus in the FCU is 128 bits wide, and in most configuration modes, the data pin count is less than 128, the incoming data stream goes through a gearbox to reduce the throughput. This configuration ensures that the internal programming circuitry runs at less than 250 MHz to process the incoming data stream. In the widest data mode (CPU ×128), the gearbox is bypassed and the entire configuration interface can run at the full 250 MHz bandwidth. Depending on the mode and configuration data width, the total bandwidth varies, and the programming time changes accordingly.

Chapter - 3: Configuration Modes for Speedster7t FPGAs

Speedster7t FPGAs support four configuration modes: Flash, JTAG, CPU and PCI Express. The selection between these modes is controlled by setting the FCU_CONFIG_MODESEL pins to the values shown in the table below. Both JTAG and PCIe modes are independent of the FCU_CONFIG_MODESEL pin setting and have to be enabled by sending FCU commands that set the appropriate bits in FCU register space. The JTAG mode can be enabled by writing to the user data register of the JTAG TAP controller and the PCI Express mode is enabled by writing to the PCIe mode enable register in the FCU address space. JTAG mode overrides all other configuration modes until disabled.

Table 2: *Pin Settings for Various Configuration Modes*

Configuration Mode	Data Width	FCU_CONFIG_MODESEL [3:0]	FCU_CONFIG_SYSCLK_BYPASS(3)	FCU_CONFIG_CLKSEL(3)
JTAG(1)	–	XXXX(2)	X	1
PCIe	–	XXXX	X	0
NoOp	–	0000	X	X
Flash single device (1D)	1 (SPI)	0001	0/1	0
	2 (Dual)	1000		
	4 (Quad)	1010		
	8 (Octa)	1100		
Flash four devices (4D)	1 (SPI)	0010		
	2 (Dual)	1001		
	4 (Quad)	1011		
	8 (Octa)	1101		
CPU	1	0011	1	0
	8	0100		
	16	0101		
	32	0110		
	128(4)	0111		

Configuration Mode	Data Width	FCU_CONFIG_MODESEL [3:0]	FCU_CONFIG_SYSCLK_BYPASS(3)	FCU_CONFIG_CLKSEL(3)

Table Notes

1. Always active. Enabled in the JTAG TAP controller.
2. If `FCU_CONFIG_MODELSEL[3:0]` pins are set such that flash or CPU configuration mode is selected, then the JTAG override should be issued after flash programming has completed or the CPU mode interface is inactive.
3. These straps select the configuration clock source.

FCU_CONFIG_SYSCLK_BYPASS	Clock Selected
0	On-chip oscillator clock
1	`FCU_CPU_CLK`

4. Speedster7t FPGAs have 32 dedicated data I/O pins for the CPU interface, which supports an up to ×32 interface. For ×128 mode, the upper 96 pins are shared with the DDR4 interface.

Configuration via CPU

In CPU configuration mode, an external CPU acts as the master controlling the programming operations to Speedster7t FPGA and offers a high-speed method for loading configuration data. Depending on the setting of FCU_CONFIG_MODESEL pins, the CPU mode can be either a 1-, 8-, 16-, 32-bit wide or 128-bit wide parallel interface, clocked using `FCU_CPU_CLK`, with chip select support to indicate valid data. This mode is the fastest programming mode as it provides the widest data interface and a maximum supported clock rate of 250 MHz.

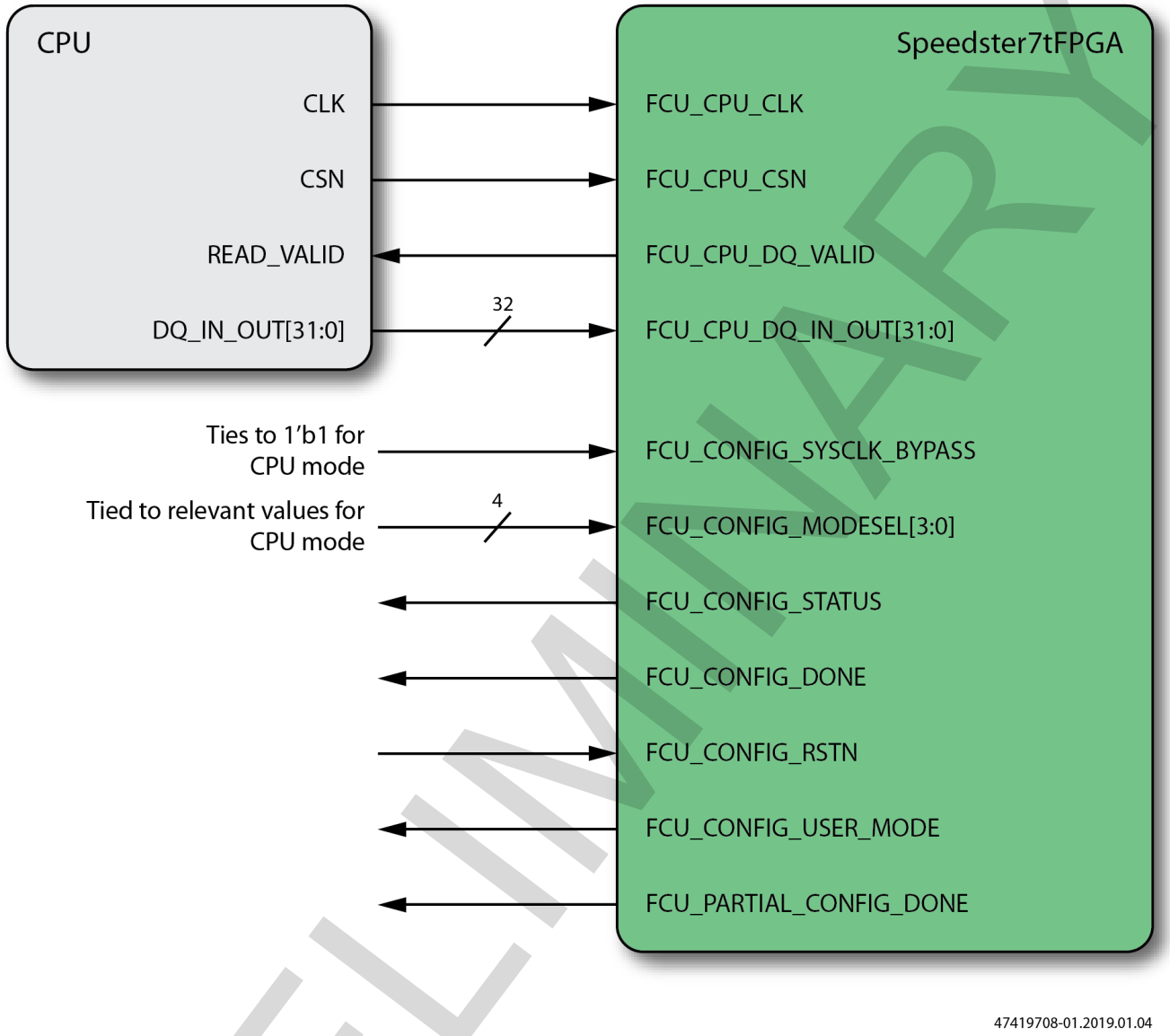


Figure 2: ***External CPU Connectivity to a Speedster7t FPGA***

> **Note**
>
> The CPU master needs only to connect to the first 1, 8, 16, 32 bits of `FCU_CPU_DQ_IN_OUT` depending on the CPU mode selected. All unused signals should be tied to ground.

As described in Configuration Sequence and Power-up (see page 38) section, the configuration mode-specific operations occur between the release of `FCU_CONFIG_STATUS` (indicating that the configuration memory has been cleared and that the Speedster7t FPGA is ready to accept bitstream data) and the assertion of `FCU_CONFIG_DONE` (stating completion of configuration). The example waveform below for CPU×8 mode illustrates the sequence of events, clocking and control signal states needed for successful configuration in CPU mode:

1. After FCU_CPU_RSTN is de-asserted, FCU_CPU_CLK must continue to cycle to ensure that the FPGA cycles through the FCU states and the configuration memory is cleared. At that point, FCU_CONFIG_STATUS is driven high.
2. After at least 5 clock cycles of FCU_CONFIG_STATUS being driven high, FCU_CPU_CSN must be pulled low to begin writing the bitstream data into the Speedster FPGA. When the last set of data is written into the Speedster7t FPGA, FCU_CPU_CSN is pulled back high.
3. Once FCU_CPU_CSN is pulled high, FCU_CPU_CLK needs to continue being clocked. Once the FCU cycles through all of the configuration states, FCU_CONFIG_DONE is driven high to indicate that the Speedster7t FPGA was successfully programmed.
4. As the FCU_CPU_CLK toggles, the FCU cycles through its states to move the Speedster7t FPGA from programming mode into user mode, taking the fabric out of reset and performing operations to enable user-mode functionality for all parts of the core. The FCU_CONFIG_USER_MODE signal is asserted to indicate when the Speedster7t FPGA has successfully transitioned into user mode.

At any point during the configuration, if FCU_CPU_CSN is asserted low, then the bus FCU_CPU_DQ_IN_OUT should have valid data or NOPs, if FCU_CPU_CSN is high, the data on FCU_CPU_DQ_IN_OUT is ignored. Once the bitstream is programmed, FCU_CPU_CSN can be held low while sending NOPs to the Speedster7t FPGA. This action will not affect the assertion of FCU_CONFIG_DONE or FCU_CONFIG_USER_MODE signals.

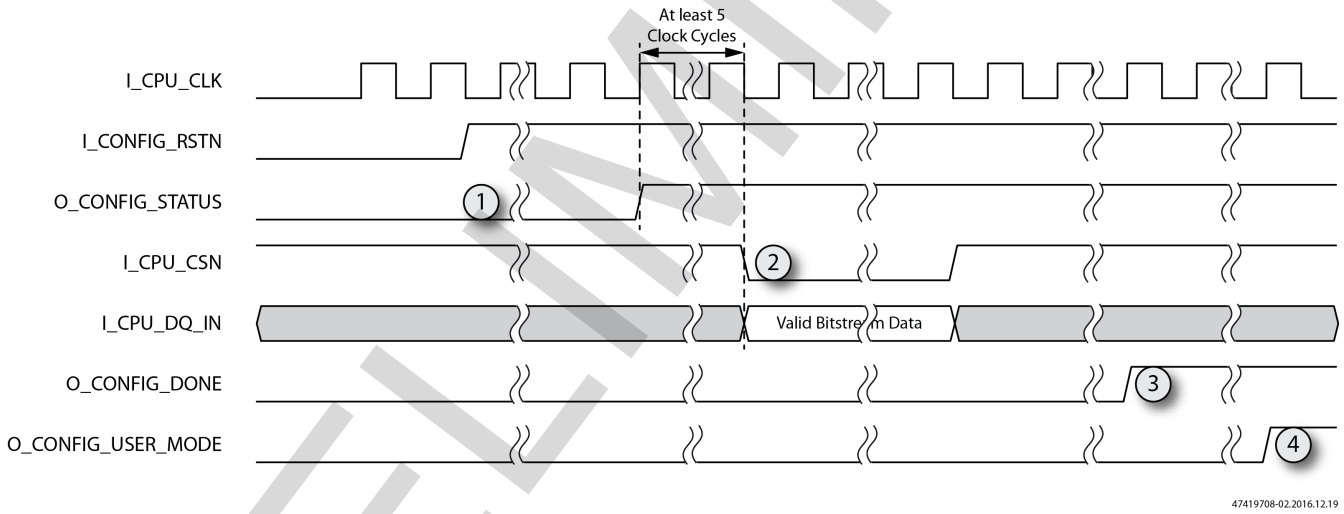


Figure 3: ***Clocking and Control Signals for Successful Configuration***

Programming Data Ordering

In Speedster7t FPGAs, the configuration memory data bus is 128 bits wide, but the command and FCU register buses are 32 bits wide. Data transmission occurs MSB to LSB at both the byte and 32-bit packet levels. Commands are executed 32 bits at a time, but the data register is 128 bits wide and requires that four sets of 32-bit packets be transmitted. At the 128-bit full payload level, the data transmission needs to occur in the following order: i3, i2, i1, i0, where i*x* is a 32-bit packet. The sequence of instructions is i0, i1, i2 and then i3.

This structure makes the bitstream programming implementation very uniform for CPU×1, CPU×8, CPU×16 and CPU×32 modes. The various potential data orders are illustrated in the example waveforms below, each showing the transmission of the same bitstream contents in the five different CPU widths.

Note

The figures in this section are to show methodologies and generalized scenarios. For detailed waveforms for specific commands, refer to the respective section in FCU Command List. Also, the JTAG ID values in the waveforms below are indicative and not specific to a device.

CPU×32

As shown in the waveform below, a command is issued on each clock cycle in CPUx32 mode:

- The first 128-bit payload shows that the order of loading is NOP, Instance ID, JTAG ID and then Sync, with each 32-bit packet transmitted MSB to LSB. However, as indicated above, the sequence in which these are processed by the FCU are Sync, JTAG ID, Instance ID and finally NOP.
- The second 128-bit payload operates the same way where the write command is transmitted first followed by three NOPs but the execution occurring in the reverse order with the write command being executed last. Also, when a write or read command is issued, it needs to be the last 32-bit FCU command in the 128-bit sequence. This requirement is because the FCU expects data input or provides data output immediately following the write and read operations respectively.
- Once the write command has been issued for a particular frame, subsequent clocks have CMEM frame data transmitted on every clock, again in 128-bit payload sets.

The signal FCU_CPU_CSN must be held low during the entire time when FCU commands are being issued for write operations. If FCU_CPU_CSN is asserted during the (128/CPU_data_width) continuous clock cycles of one request, that request is discarded. Once the FCU_CPU_CSN signal returns low, the next request is handled normally.

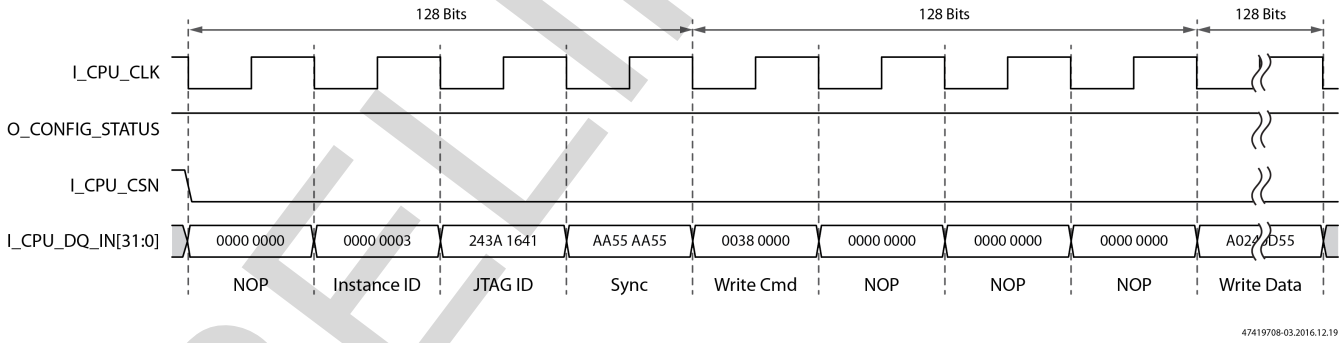


Figure 4: ***Bitstream Programming in CPU×32 Mode***

CPU×16

CPU×16 mode is very similar to CPU×32 mode. The only difference is that 16-bits of data are transmitted on each FCU clock cycle, i.e., each FCU command is transmitted over two FCU clock cycles, MSB to LSB (as shown in the waveform below).

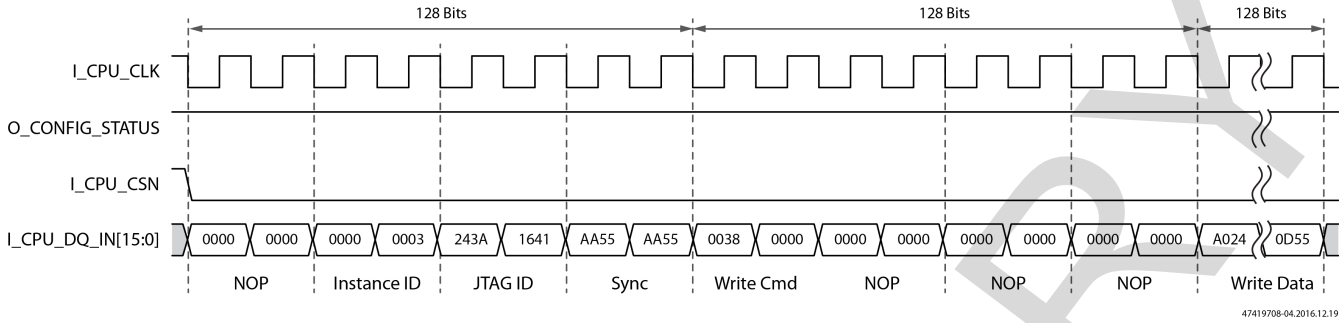


Figure 5: *Bitstream Programming in CPU×16 Mode*

CPU×8

CPU×8 mode follows along the lines of CPU×16 and CPU×32 modes, with each FCU command requiring four FCU clock cycles for transmission, MSB to LSB, as detailed in the waveform below.

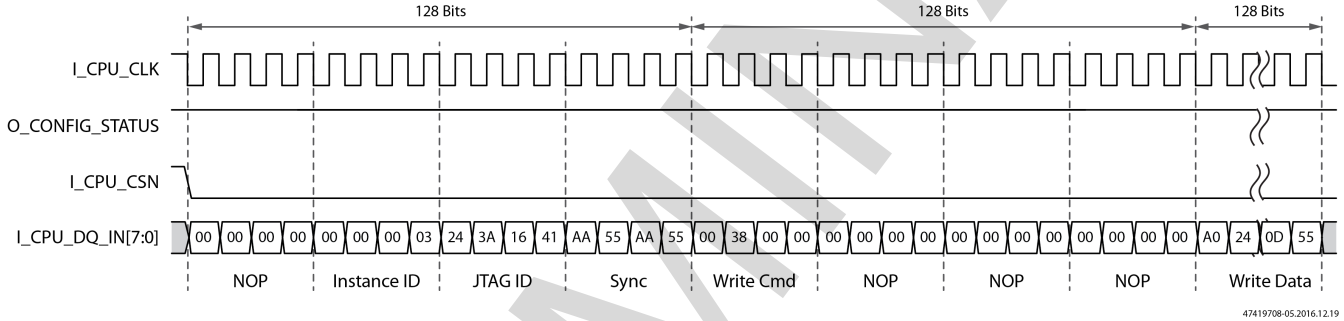


Figure 6: *Bitstream Programming in CPU×8 Mode*

CPU×1

In CPU×1 mode, a single bit of the FCU command (or write data) is transmitted on each FCU clock cycle, MSB to LSB, for a 32-bit packet, but in reverse order for the 128-bit payload as described in the other CPU width modes. The waveform below shows these details.

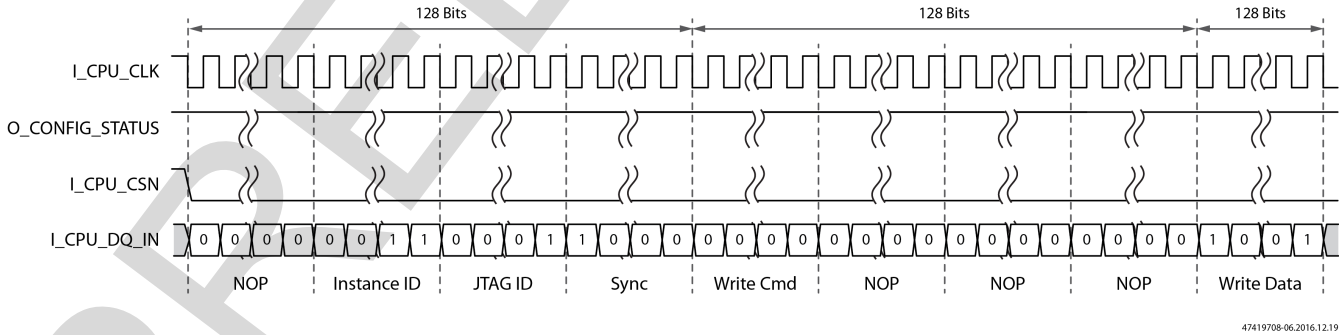


Figure 7: *Bitstream Programming in CPU×1 Mode*

> **Note**
>
> Contact Achronix Support for more details on the CPU x128 mode.

Data Ordering In the ACE Output File

The programming files generated by ACE lists the FCU commands/data in the exact same transmission order as shown in the waveforms above. The code snippets below highlight this ordering in the CPU×32 and CPU×128 modes.

CPU×32 ACE Programming File Snippet

```
NOP
Instance ID
JTAG ID
Sync
Write Cmd
NOP
NOP
NOP
Write Data
Write Data
....
```

CPU×128 ACE Programming File Snippet

```
{NOP, Instance ID, JTAG ID, Sync}
{NOP, NOP, NOP, NOP}
{NOP, NOP, NOP, NOP}
{NOP, NOP, NOP, NOP}
{Write Cmd, NOP, NOP, NOP}
{NOP, NOP, NOP, NOP}
{NOP, NOP, NOP, NOP}
{NOP, NOP, NOP, NOP}
{Write Data, Write Data, Write Data, Write Data}
....
```

Configuration via Flash Memories

Caution!

Speedster7t devices can interface to serial NOR flash devices only. Parallel NOR, NAND or other flash variants are *not* supported.

Flash programming mode allows flash memories to be used to configure Speedster7t devices. In this mode the FPGA is the master, and therefore, supplies the clock to the flash memory.

The clock supplied from the FPGA (on the FCU_FLASH_SCK pin) to the attached flash device(s) can be driven by the FCU_CPU_CLK or the on-chip oscillator clock depending on the configuration options selected as described in Configuration Modes for Speedster7t FPGAs (see page 9). The frequency of this clock can be selected from one of four variants of the clock sources arriving at the FCU: the original (divide-by-1), divide-by-2, divide-by-4 or divide-by-8. This selection is configured using the 'Serial Flash Clock Divider' drop-down menu in the 'Bitstream Generation Implementation Options' section of the ACE GUI. This setting ensures that only the flash state machine runs at the slower frequency. All other FCU and ACB circuitry will still operate at the original input clock frequency.

Notes

At power-on, the device defaults to divide-by-4 setting. The FCU then sets the appropriate configuration register to control the clock divider based on the user selection in ACE. The transition from a divide-by-4 clock to any other selected clock frequency is glitch-free.

Flash Device Configurations

Speedster7t FPGAs support two flash device configurations, single flash device (1D) and four flash devices (4D).

1D Configuration

The 1D programming configuration is composed of a Speedster7t FPGA acting as the master and communicating with a single flash device. The signal `o_flash_sck` is used for clocking, `o_flash_sdi` is the data output from the FPGA to communicate instructions to the flash device, and `i_flash_sdo[0]` is the single-bit FPGA input pin which receives the bitstream from the flash in x1 mode. The signal `o_flash_csn[0]` is pulled low as soon as communication between the FPGA and flash device begins, and stays low during the valid bitstream window.

The FPGA can communicate with the flash device in SPI, Dual, Quad or Octa modes in 1D configuration.

The figure below provides a block diagram of how a serial flash device can be connected to a Speedster7t FPGA and a SPI header for programming in ×1 mode.

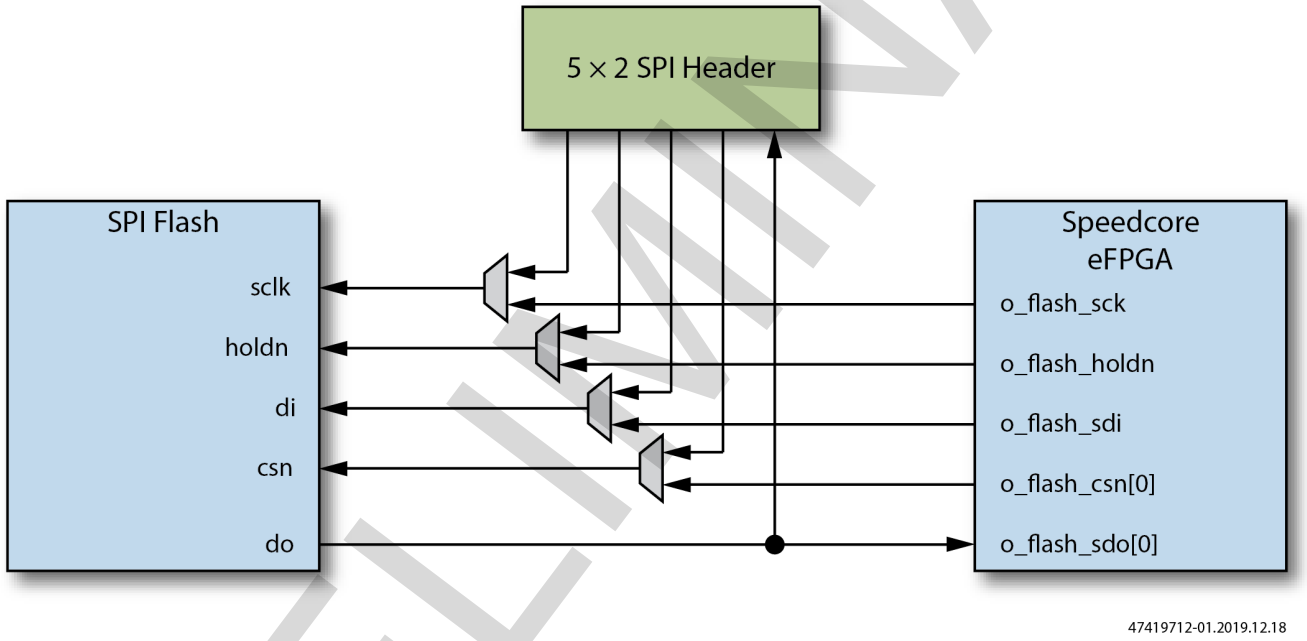


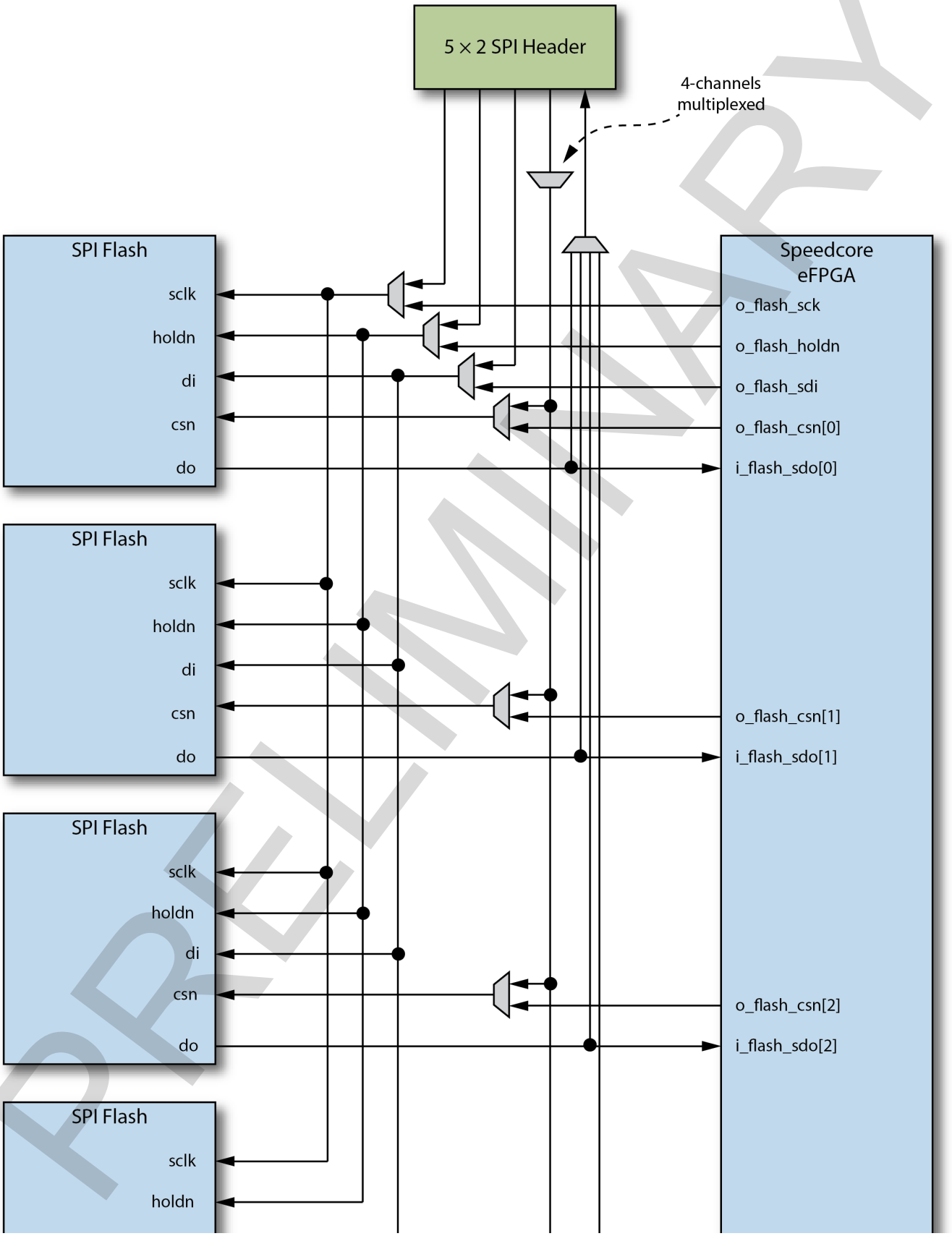
Figure 8: ***Speedster7t 1D Flash Programming Configuration***

4D Configuration

Serial 4D flash programming mode is essentially an enhanced and higher bandwidth implementation of the serial flash 1D configuration. The FPGA is again the master, and interfaces with not one but four flash memory devices to increase the data bandwidth four times.

When writing to the four flash memories, the four-channel multiplexer must ensure that `o_flash_csn[3:0]` is asserted for only a single flash memory at any given time. Through the SPI header, data is written to each flash device in sequence. When reading from the four flash memories, the FPGA pulls all of the `o_flash_csn[3:0]` signals low. Four-wide configuration data is read from the flash memories and transferred to the FPGA through the `i_flash_sdo` ports. Once bitstream operations are complete (flash memory contents are read), transitioning from the end of the bitstream to user mode is done the same way as in CPU and flash 1D modes.

Each flash device can operate in SPI, Dual, Quad or Octa modes. The figure below provides a block diagram of how four flash memories can be connected to a Speedster7t FPGA in a 4D configuration.



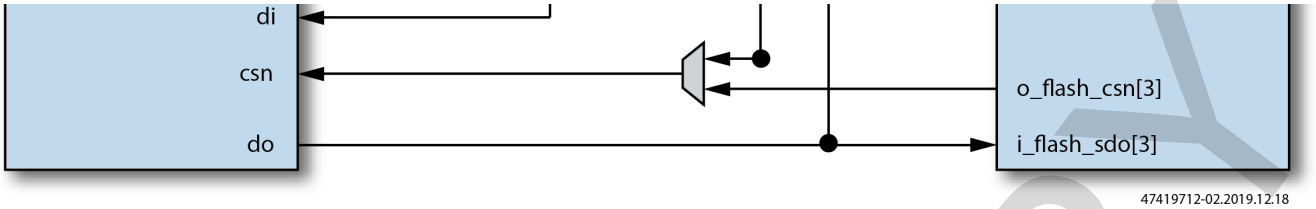


Figure 9: ***Speedster7t 4D Flash Programming Configuration***

Addressing Modes and Memory Organization

Addressing modes for the flash memory are based on the size of the device. A three-byte addressing mode is required for 128 Mb flash and smaller and a four-byte addressing mode is required to support memory sizes above 128 Mb. Writes to the flash memory are done as pages, with each page consisting of 256 bytes. The figure below shows the memory organization:

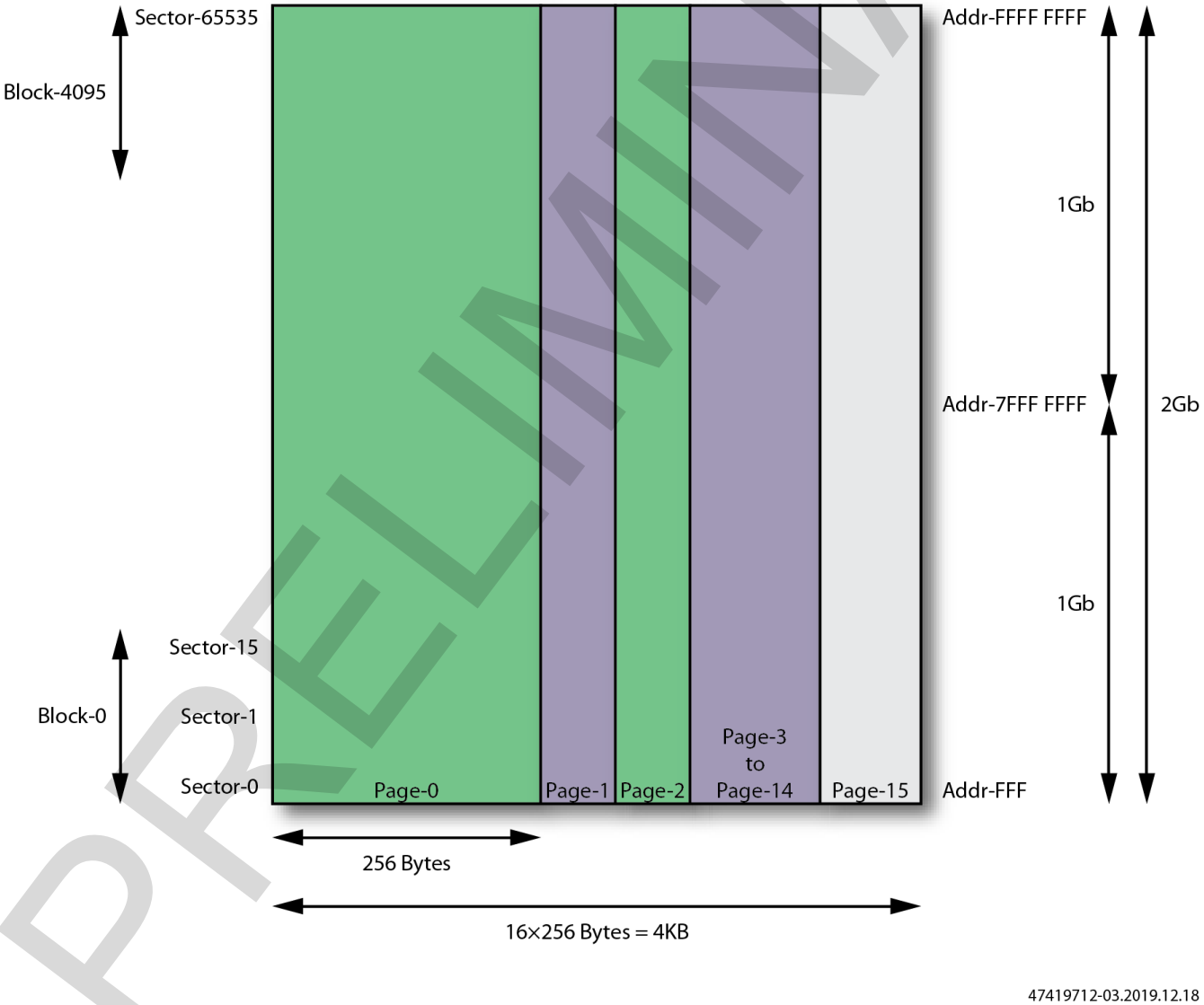


Figure 10: ***Speedster7t Flash Memory Organization***

Address Range

The below table shows the address ranges when two images are stored on a single flash devices, assuming that each image is 1Gb in size.

Table 3: ***Address Ranges for Two Bitstream Images on a Single Device***

Address Range (32 bits)	Description	Configuration Details
0x0000_0000 to 0x0000_00FF	Page-0 address space. This range contains header information described in the flash configuration header section. This address range cannot be used for storing actual bitstreams.	These addresses are not configurable by the user.
0x0000_0100 to 0x0800_00FF	FPGA image 1 address space.	The start address can be configured by the user via the current/fallback address in page-0 header. This example assumes the address starts at 0x0000_1000 for a 1 Gb bitstream.
0x0800_0100 to 0x1000_00FF	FPGA image 2 address space	The start address can be configured by the user via the current/fallback address in page-0 header. This example assumes the bitstream starts at address 0x0800_0100.

Flash Configuration Header (Page-0 Header)

The first 256 bytes in the flash memory (page 0) store control information that describe how the subsequent bitstream should be read from the flash device. This information can be written to the flash device in two ways:

- Via the JTAG interface along with the bitstream.
- Pre-programmed into the device by the manufacturer.

This space is not used for storing device bitstream.

Table 4: ***Page-0 Header Format***

Address	Bits	Value	Description
0x0 to 0x3	32	Read command	
0x4 to 0x7	32	Flash configuration header read count	
0x8 to 0xB	32	Bitstream read control	bit 0 – Flash read enable bit 1 – Flash fall back enable bit [7:2] – Retry count bit [21:8] – Timeout count bit 22 – Enable 4-byte addressing bit [27:23] – Dummy read cycles bit [31:28] – Flash SCK div count

Address	Bits	Value	Description
0xC to 0xF	32	Bitstream read address (new image)	
0x10 to 0x13	32	Bitstream fallback address (golden image)	
0x14 to 0x17	32	Fallback read command	
0x18 to 0x20	24	Reserved	

Flash Programming Protocol

With the `FCU_CONFIG_MODESEL[3:0]` , `FCU_CONFIG_CLKSEL` and `FCU_CONFIG_SYSCLK_BYPASS` straps set for serial flash programming, operations begin as soon as the FPGA is powered up and the FCU receives the clock input.

Immediately after reset is released, bitstream data is read out from flash device through the flash interface (at this time the default is SPI (×1) mode). The bitstream read is done as two stages as described below:

- Stage-1 – Flash configuration header read from flash device.
  - The FCU sends a default read command and address of 0x0000_0000 (32 bits) in SPI mode to the flash device and reads the flash configuration header.
  - Internal registers are then updated, including the start address for the bitstream and flash read command.
- Stage-2 – Bitstream read from flash device
  - Based on the read mode (×1/×2/×4/×8) obtained from the flash configuration header, the command and start address are sent to the flash device.
  - The FCU reads the first 512 bits of bitstream data from flash device and enters a wait state.
  - If encryption is not enabled, the FCU reads the complete bitstream and configures the FPGA. If encryption is enabled and the efuse key is ready, the FCU reads the header segment0 data and sends it to the secure boot core. The flash read state machine now waits for 2.6 ms after which the FCU reads the complete bitstream and configures the FPGA.

Bitstream programming in all configuration modes is MSB to LSB. For transmitting a 32-bit FCU command, the ordering in the serial ×1 mode for 1D and 4D configuration is as follows:

- 1D flash configuration – The flash device transmits command bit 31 on the first clock and bits 30, 29, 28, etc. on subsequent clocks all the way down to bit 0 on the 32nd (last) clock.
- 4D flash configuration – The four flash devices transmit command bits [31:28] on the first clock, all the way down to bits [3:0] on the eighth (last) clock. The ordering within the 4-bit nibble corresponds to the flash device ordering. In other words, on the first clock, flash[3] transmits bit 31, flash[2] transmits bit 30, flash[1] transmits bit 29 and flash[0] transmits bit 28.

Error Fallback

The 'Error Fallback' feature helps in providing instructions to the FCU in the event of a failure during bitstream loading. Such a failure can happen due to the following scenarios:

- No IDCODE match after timeout expires
- CRC error after timeout expires

If any of these checks fail, loading is retried N times (as specified in the bitstream read control register in the flash configuration header). If failures persist and a fallback bitstream is enabled by the user, a fast read is issued to the fallback address but if there is no fallback bitstream enabled, the FCU abandons the bitstream loading operation.

Flash Modes

The following section describes the various modes supported for read and write operations to/from the flash device. Read operations from the flase device can be configured either as SPI, Quad or Octa modes for both 1D and 4D configurations while write operations to the flash device is always done in SPI mode.

> **Note**
>
> A flash write can be done by the user either the JTAG mode or PCIe mode. The PCIe or JTAG can access the data and command registers by indirect mode of addressing.

The following table describes the different combinations of the flash device configurations and modes supported in Speedster7t FPGA.

Flash programming mode /configuration	Flash interface width	No of flash devices	Write width SO[0] pin x No of Flash device	Read Width SO[n:0] x No of Flash device
SPI x1 (1D)	1	1	1	1
SPI x1 (4D)	1	4	4	4
Quad x4(1D)	4	1	1	4
Quad x4 (4D)	4	4	4	16
Octa x8 (1D)	8	1	1	8
Octa x8 (4D)	8	4	4	32

Read operation timing diagrams for each of the modes is described below:

SPI Mode (×1)

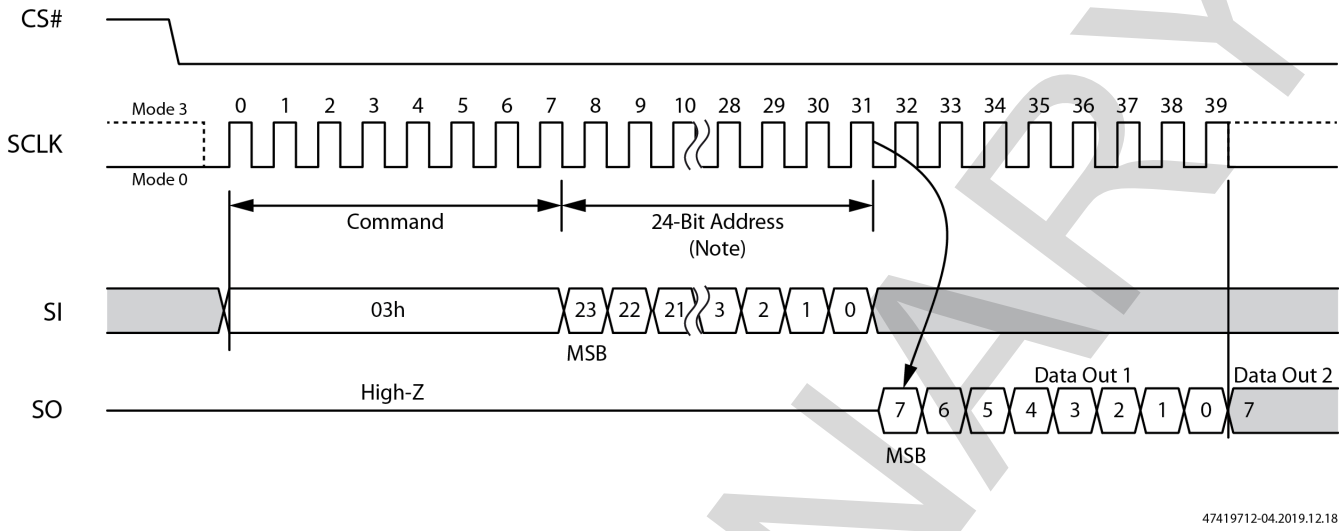


Figure 11: *SPI Mode (×1) Read*

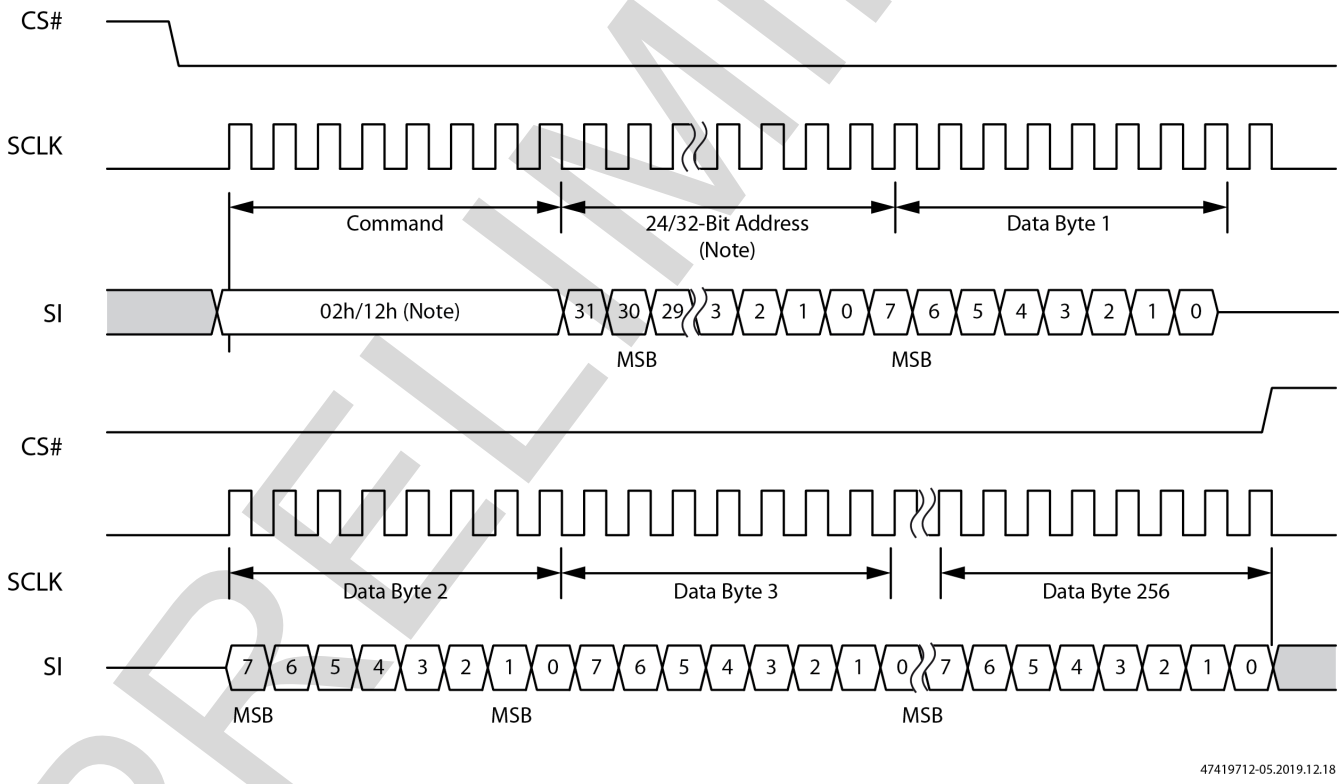


Figure 12: *SPI Mode (×1) Write*

Quad Mode (×4)

Reads

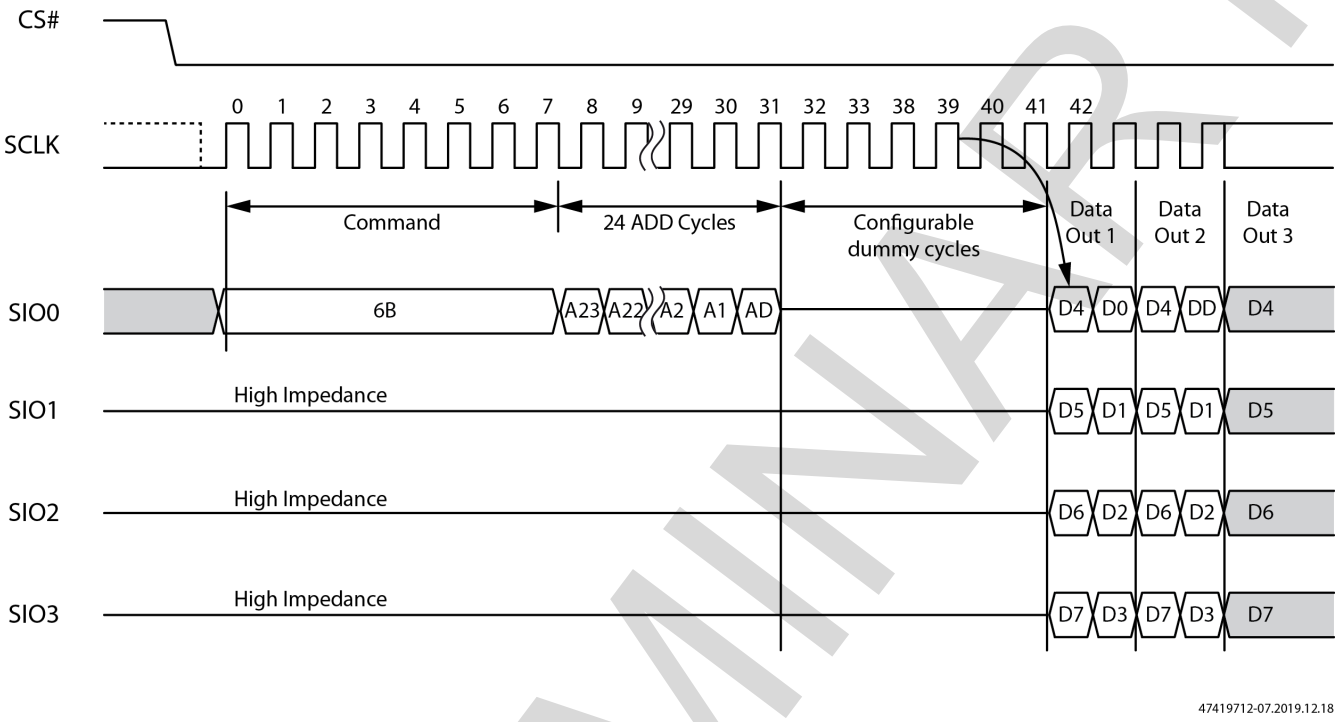


Figure 13: ***Quad Read Mode (QREAD)***

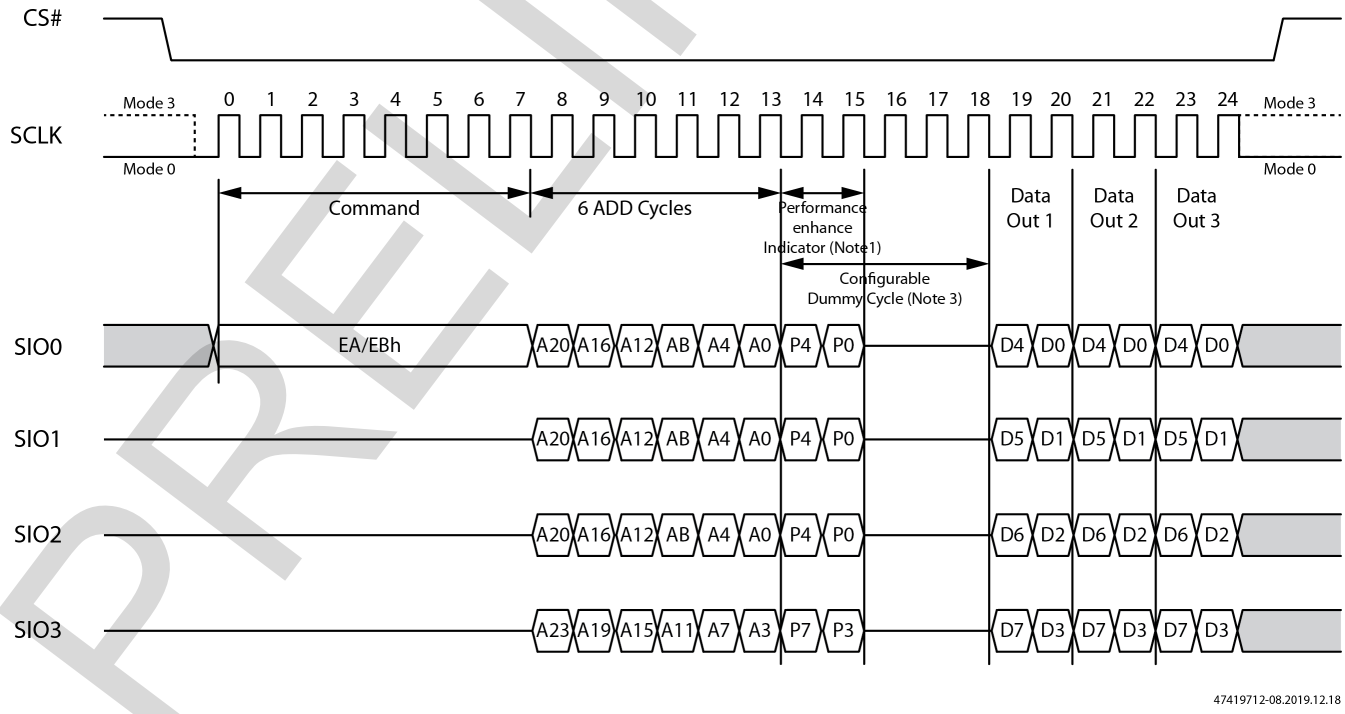


Figure 14: ***4× I/O Read (SPI Mode)***

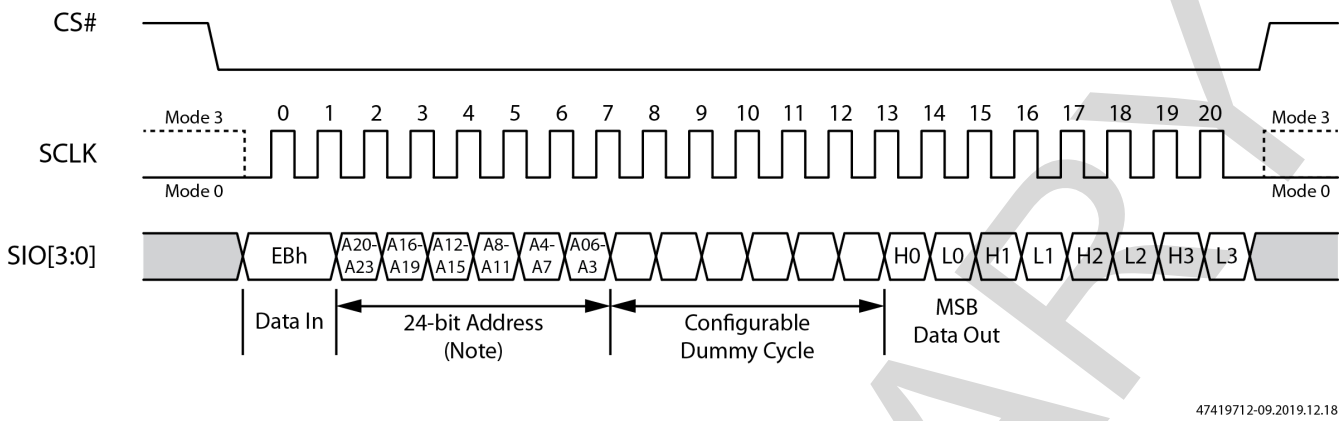


Figure 15: *4× I/O Read (QPI Mode)*

Octa Mode (×8)

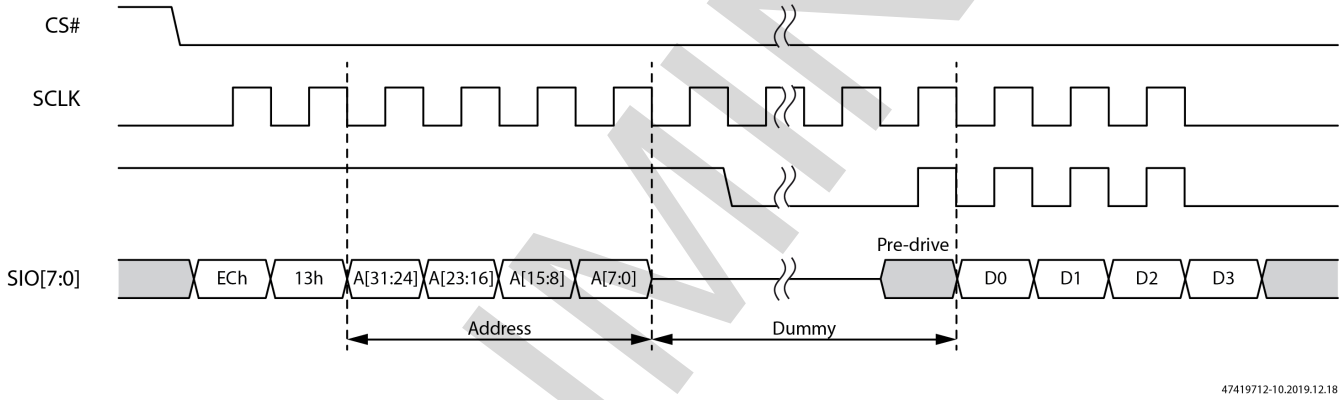


Figure 16: *Octa Mode (×8) Read*

Registers and Addressing

Table 5: *Flash Controller Register Map*

Register Name	Address	Description
Flash write control register	0x1038	Flash write control register
Flash write count	0x1048	Flash write count register
Flash write configuration register	0x1050	Flash configuration register
Flash write status	0x1060	Flash status register
Flash write data1	0x1040	Flash write data register
Flash write data2	0x11d4	Flash write data register

Register Name	Address	Description
Flash write data3	0x11d8	Flash write data register
Flash write data4	0x1044	Flash write data register
Flash current bitstream current address	0x12bc	Flash bitstream read current address
Flash fallback bitstream fallback address	0x12b8	Flash bitstream read fallback address
Flash write command 1	0x103c	Flash command register
Flash write command 2	0x104c	Flash command register
Flash write command 3	0x1054	Flash command register
Flash write command 4	0x1058	Flash command register

Table 6: ***Flash Write Control Register***

Register Field	Bit Position	Type	Reset value	Description
Flash write enable	0	RW	0x0	Initiate the flash write operation.
Flash write clock div count	4:1	RW	0x1	Clock divider. Set to 4'b0001 default, divide by 2 clock which is required for JTAG mode.
Flash write Stop	5	RW	0x0	Suspend the current operation.
Flash write wait	6	RW	0x0	Flash wait operation.
Flash write ×1 mode	7	RW	0x0	Flash write in SPI ×1 device mode.
Flash write ×4 mode	8	RW	0x0	Flash write in SPI ×4 device mode.
Reserved	31:9	RW	0x0	Reserved.

Table 7: ***Flash Write Configuration Register***

Register Field	Bit Position	Type	Reset Value	Description
Flash write data valid	0	RW [Write on clear]	0x0	Write data valid, Indicates to the flash interface when data is written to flash write register. Cleared when the flash interface reads the data and writes it into the internal registers.

Register Field	Bit Position	Type	Reset Value	Description
Flash write command valid	1	RW	0x0	Flash write command valid.
Flash write command count	8:2	RW	0x8	Write command count in number of bits.
Flash write data count	15:9	RW	0x127	Write data count in number of bits.
Flash write data request	16	R	0x1	Request write data, PCIe, should poll this bits, will be cleared once data is shifted to internal registers.

Table 8: ***Flash Write Status***

Register Field	Bit Pposition	Type	Reset Value	Description
Flash write error	0	RO	0x0	Flash write error, flags flash device status
Flash read error	1	RO	0x0	Flash read rrror, CRC error
Flash write done	2	RO	0x0	Flash write is complete
Flash read done	3	RO	0x0	Flash read is complete
Flash state machine status	[8:4]	RO	0x0	Write state machine status
Reserved	31:9	RO	0x0	Reserved

Configuration via JTAG

The Speedster7t JTAG TAP controller is compliant to IEEE Std 1149.1 and is used for programming the bitstream and debug via Snapshot in ACE. The `JTAG_TMS` and `JTAG_TCK` inputs determine whether an instruction register scan or data register scan is performed. `JTAG_TMS` and `JTAG_TDI` are sampled on the rising edge of `JTAG_TCK`, while `JTAG_TDO` changes on the falling edge. JTAG configuration and operation mode is independent of `FCU_CONFIG_MODESEL` settings.

JTAG implementation in Speedster7t FPGAs, which allows for bitstream programming as well as real-time in-system control and observation, is composed of the blocks shown in the figure below.

The external interface is a standard 5-pin JTAG interface, connected directly to the JTAG TAP controller. The TAP controller operates independently from the Speedster7t FPGA FCU. It is always active and uses JTAG_TCK for clocking. The TAP controller takes the data from the pins and converts it to DR instructions to communicate to the JTAG logic in the FCU. It also takes in data in the form of load/read requests, translating it to the appropriate signals to drive and expect on the JTAG pins.

The JTAG logic in the FCU uses these DR instructions and generates input data in the standard 128-bit Speedster7t FPGA frame size format, along with a data valid indicator, to be forwarded to the FCU data mux and ultimately FCU state machines for configuration memory loading. It also takes in 128-bit output data from the FCU, which also comes with a valid signal for debug and read-back operations. It provides an acknowledge signal to indicate to downstream circuitry that the data transfer was successful.

The FCU data mux simply selects between the configuration mode specific data buses coming in to the FCU. This logic is controlled by the static FCU_CONFIG_MODESEL straps and the JTAG override circuitry from the JTAG TAP controller.

Finally, the FCU state machine takes incoming data and uses it for loading the configuration memory. Conversely, it also provides output data from the configuration memory or Snapshot to be forwarded upstream.

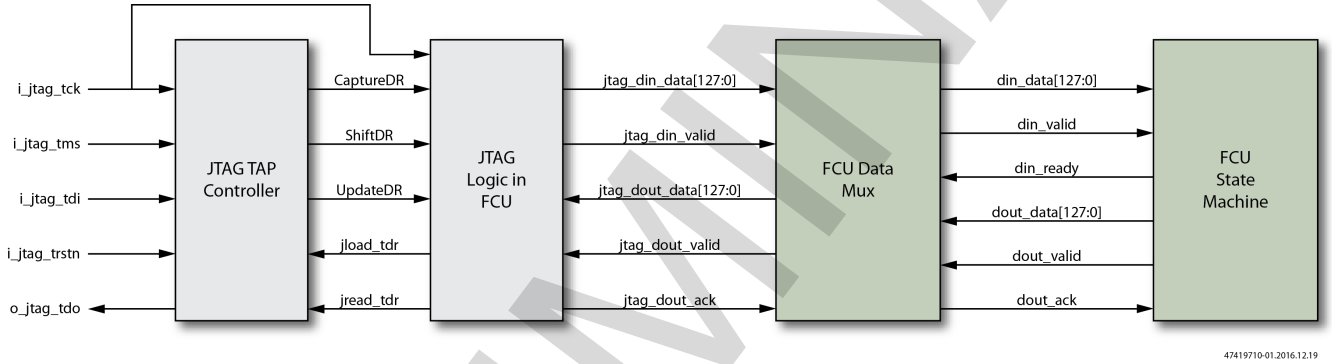


Figure 17: ***Block Diagram for JTAG Instruction Processing in FCU***

The JTAG programming sequence is highlighted in the waveform below to show the sequence of internal procedures that occur in the ACE generated jam file. An explanation of these steps is as follows:

1. DO_RESET_CHIP – An internal signal generates a pulse on the FCU reset circuitry to reset it, similar to providing a pulse on the FCU_CONFIG_RSTN input pin.
2. DO_ENTER_JTAG – A TAP command (override) is provided to place the Speedster7t FPGA FCU in JTAG mode. After this point, regardless of the FCU_CONFIG_MODESEL settings, the FCU configuration mode (and the data muxes) are set to listen to the JTAG inputs, and the FCU clock is sourced from JTAG_TCK.
3. DO_ERASE – This step cycles through the FCU states to ensure that the configuration memory is cleared. After this step, FCU_CONFIG_STATUS is asserted.
4. DO_PROGRAM – This is where the actual bitstream loading occurs. This operation consists of DRSCAN loops for every bit in the bitstream. Since the size of the bitstream is pre-determined, the loop count is set appropriately by ACE.
5. DO_ENTER_USER_MODE – IRSCAN and DRSCAN commands are provided to cycle through additional FCU states. Idle clocks are provided to ensure that the start-up state machine completes successfully, and in the process, o_config_done and FCU_CONFIG_USER_MODE is asserted. After this step, functions hosted within a Speedster7t FPGA are active.
6. DO_EXIT_JTAG – This is another TAP command performed in parallel once user-mode operations start to quickly provide additional instructions to remove the JTAG override on the FCU.

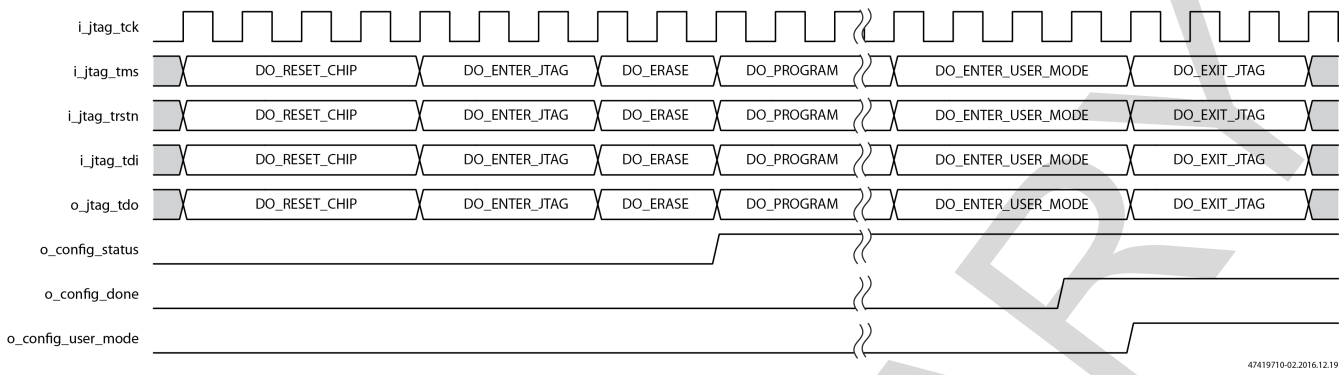


Figure 18: ***JTAG Bitstream Programming Sequence***

JTAG Instructions

Table below lists all JTAG instructions supported by Speedster7t FPGAs.

Table 9: ***JTAG Instructions***

Instruction	Opcode	DR Width	Function
BYPASS	23'b00000000000000000000000	1	The required BYPASS instruction allows a Speedster7t FPGA to remain in a functional mode and selects the bypass register to be connected between JTAG_TDI and JTAG_TDO. The BYPASS instruction allows serial data to be transferred through the FCU from JTAG_TDI to JTAG_TDO without affecting the operation of a Speedster7t FPGA.
EXTEST	23'b11111111111111111101000	–	The required EXTEST instruction places a Speedster7t FPGA into an external boundary-test mode and selects the boundary-scan register to be connected between JTAG_TDI and JTAG_TDO. Output pins operate in test mode, driven from the contents of the boundary-scan update latch. Input data captured in boundary-scan latches prior to shift operation. In other words, during this instruction, the boundary-scan register is accessed to drive test data outside of a Speedster7t FPGA via the boundary outputs and receive test data from outside of a Speedster7t FPGA via the boundary inputs.
EXTEST_PULSE	23'b11111111111111111101001	–	As the names suggest, EXTEST_PULSE generates a single pulse by entering and exiting the Run-Test/Idle state of the 1149.1 TAP controller.

Instruction	Opcode	DR Width	Function
EXTEST_TRAIN	23'b11111111111111111101010	–	EXTEST_TRAIN generates a stream of pulses while in the Run-Test/Idle state. A BSDL file for an 1149.6 device specifies the minimum number of pulses and the maximum time period allowed for pulse generation in the Run-Test/Idle state.
SAMPLE /PRELOAD	23'b11111111111111111111000	–	The required SAMPLE/PRELOAD instruction allows a Speedster7t FPGA to remain in its functional mode and selects the boundary-scan register to be connected between JTAG_TDI and JTAG_TDO. The output and input pins operate in normal mode. Input pin data and core logic output data captured in the boundary-scan latches. In other words, during this instruction, the boundary-scan register can be accessed via a data scan operation to take a sample of the functional data entering and leaving a Speedster7t FPGA. This instruction is also used to preload test data into the boundary-scan register before loading an EXTEST instruction.
IDCODE	23'b11111111111111111111110	32	The optional IDCODE instruction allows a Speedster7t FPGA to remain in its functional mode and selects the optional device identification register to be connected between JTAG_TDI and JTAG_TDO. The IDCODE register appears between JTAG_TDI and JTAG_TDO after power-up, after the TAP has been reset using the optional TRST pin, or by otherwise moving to the Test-Logic-Reset state.
HIGHZ	23'b11111111111111111001111	–	The optional HIGHZ instruction sets all outputs (including two-state as well as three-state types) to a disabled (high-impedance) state and selects the bypass register to be connected between JTAG_TDI and JTAG_TDO.
CLAMP	23'b11111111111111111101111	–	Provides for “guarding” chip outputs during in-circuit test or boundary-scan functional test. Output pins operate in test mode, driven from content of boundary-scan update latch. The one-bit bypass register is selected for shifting.
INTDR	23'b00000000000000000111101	97	Test data register is implemented internally to the TAP controller. This internal register is used for global configuration and monitoring of global status signals. These registers are associated with a specific user-defined instruction.

Instruction	Opcode	DR Width	Function
JLOAD	23'b00000100000001100111010	128	Enables the scan in of the configuration bitstream into the configuration logic (in this mode, the SHIFT-DR state is used to scan in the bitstream). For the read-back, the data register is read back. All of these operations are done internally using a 128-bit parallel bus. Data is latched every 128 bits in the UPDATE-DR state.
JREAD	23'b00000100000001000111010	128	Enables the data register for read-back. When this instruction is decoded and CAPTURE-DR is executed, the data from the configuration logic is sampled as 32-bit data plus a valid bit. Multiple words of the configuration memory can be read back by cycling through the CAPTURE-DR/SHIFT-DR states. The 33-bit status register is selected between `JTAG_TDI` and `JTAG_TDO`.
JUSR1	23'b00000100000000100111010	User defined	This instruction enables the USER1 TDR.(†)
JUSR2	23'b00000100000000000111010	User defined	This instruction enables the USER2 TDR.(†)
JASYNCERR	23'b00000000000001110111010	–	This instruction enables the connection to the fabric error status scan register.

Table Note

† This TDR is implemented in the fabric and is used for supporting debug functionality in the fabric.

Chapter - 4: Configuration Pin Tables

Table 10: *Interface Pin Table*

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Usage</th></tr>
<tr><td colspan="3">Configuration Interface</td></tr>
<tr><td>FCU_CONFIG_MODESEL[3:0]</td><td>Input</td><td>Configuration mode selection inputs to define the FPGA configuration unit (FCU) mode of operation.
<table>
<tr><th>Configuration Mode</th><th>CFG_MODESEL[3:0]</th></tr>
<tr><td>Flash Serial 1x</td><td>0001</td></tr>
<tr><td>Flash Serial 4x</td><td>0010</td></tr>
<tr><td>CPU x1,x8,x16,x32,x128</td><td>0011 to 0111</td></tr>
<tr><td>Flash Dual x1</td><td>1000</td></tr>
<tr><td>Flash Dual x4</td><td>1001</td></tr>
<tr><td>Flash Quad x1</td><td>1010</td></tr>
<tr><td>Flash Quad x4</td><td>1011</td></tr>
<tr><td>Flash Octa x1</td><td>1100</td></tr>
<tr><td>Flash Octa x4</td><td>1101</td></tr>
<tr><td>JTAG</td><td>Always active mode</td></tr>
</table></td></tr>
<tr><td>FCU_CONFIG_STATUS(4)</td><td>Output</td><td>Active-high configuration status output signal indicating that the FCU has completed initial start-up and has cleared the CMEM and is awaiting FCU commands for bitstream programming. Once high, it stays asserted until the FCU is power cycled or reset for a re-initialization sequence or a CRC error is seen during bitstream load.</td></tr>
<tr><td>FCU_CONFIG_DONE(4)</td><td>Output</td><td>Active-high configuration done output signal indicating that bitstream loading completed successfully and that the device is ready to enter user mode. Once high, it stays asserted until the FCU is power cycled or reset for a re-initialization sequence.If a device configuration error occurs, the CONFIG_DONE output will remain low. Holding this pin low on the board can be used as a method to synchronize the start-up of multiple devices.</td></tr>
<tr><td>FCU_CONFIG_RSTN(1)</td><td>Input</td><td>Asynchronous active-low reset input clearing the configuration memory in the device and the logic in the FPGA configuration unit (FCU).</td></tr>
<tr><td>FCU_CONFIG_USER_MODE(4)</td><td>Output</td><td>Active-high output indicating that the device has transitioned into user mode. Once high, it stays asserted until the FCU is power cycled or reset for a re-initialization sequence.</td></tr>
</table>

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Usage</th></tr>
<tr><td>FCU_CONFIG_SYSCLK_BYPASS</td><td>Input</td><td>Active-high bypass configuration system clock setting. Along with CFG_CLKSEL, this setting allows for clock selection during programming.
<table>
<tr><th>SYSCLK_BYPASS</th><th>CFG_CLKSEL</th><th>CFG_MODESEL[3:0]</th><th>Configuration Clock</th></tr>
<tr><td>0</td><td>0</td><td>0000, 0001, 0010
1000 to 1101</td><td>On-chip Oscillator</td></tr>
<tr><td>1</td><td>0</td><td>0000, 0001, 0010,1000 to 1101</td><td>CPU clock</td></tr>
<tr><td>X</td><td>0</td><td>0011, 01XX</td><td>CPU clock</td></tr>
<tr><td>X</td><td>1</td><td>XXXX</td><td>JTAG TCK</td></tr>
</table></td></tr>
<tr><td>FCU_CONFIG_BYPASS_CLEAR</td><td>Input</td><td>Active-high input pin to bypass configuration memory clear during device initialization.</td></tr>
<tr><td>FCU_CONFIG_ERR_ENC[2:0]</td><td>Output</td><td>Active-high output pins highlighting the presence of a CRC, scrubbing or other errors in the bitstream during device programming. If asserted, it continues to stay high and users should expect that configuration loading never complete, and user mode is never entered.
<table>
<tr><th>FCU_CONFIG_ERR_ENC[2:0]</th><th>Status</th><th>Priority</th></tr>
<tr><td>001</td><td>CRC Error.</td><td>0 (Lowest)</td></tr>
<tr><td>010</td><td>Single-bit/multiple-bit scrubbing error</td><td>1</td></tr>
<tr><td>011</td><td>Secure Boot Failure OR Security error.</td><td>2</td></tr>
<tr><td>100</td><td>Efuse PUF enrollment error.</td><td>3</td></tr>
<tr><td>101</td><td>Asserted when AXI interface of IP configuration space register block does not receive a ready from the master</td><td>4 (Highest)</td></tr>
<tr><td>Other</td><td>Undefined</td><td></td></tr>
</table></td></tr>
<tr><td>FCU_LOCK</td><td>Output</td><td>Active-high status bit to indicate the FCU lock/unlock status</td></tr>
<tr><td>FCU_OSC_CLK</td><td>Output</td><td>This clock is internally generated from a ring oscillator. For debug purposes it can be bypassed and the external clock CPU_CLK can be used.</td></tr>
<tr><td>FCU_PARTIAL_CONFIG_DONE</td><td>Output</td><td>Active-high configuration done output signal indicating that bitstream loading completed successfully for partial reconfiguration of the device and that the device is ready to enter user mode.</td></tr>
<tr><td>FCU_STAP_SEL</td><td>Input</td><td>When asserted high, this signal enables the JTAG interface pins to be directly connected to the JTAG controller in the SerDes PMA blocks allowing SerDes configuration, debug and performance monitoring directly from the JTAG interface. For bitstream download and design debug using the JTAG interface, this pin must be held low. For SerDes PMA debug only mode, this pin must be held high.</td></tr>
</table>

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Usage</th></tr>
<tr><td>FCU_STATUS[1:0]</td><td>Output</td><td>FCU status bits showing the FCU state
<table>
<tr><th>FCU_STATUS</th><th>State</th></tr>
<tr><td>11</td><td>fcu_locked</td></tr>
<tr><td>10</td><td>sync_found</td></tr>
<tr><td>01</td><td>ID found</td></tr>
<tr><td>00</td><td>instance ID found / FCU unlocked</td></tr>
</table></td></tr>
<tr><td>FCU_STRAP[2:0]</td><td>Output</td><td>Unconnected spare outputs</td></tr>
<tr><td colspan="3">JTAG Interface</td></tr>
<tr><td>JTAG_TCK</td><td>Input</td><td>Clock input to the JTAG controller in the FCU.</td></tr>
<tr><td>JTAG_TRSTN</td><td>Input</td><td>Active-low reset input to the JTAG controller in the FCU.</td></tr>
<tr><td>JTAG_TDI</td><td>Input</td><td>Serial data input to the JTAG controller in the FCU. Synchronous to JTAG_TCK.</td></tr>
<tr><td>JTAG_TDO</td><td>Output</td><td>Serial data output from the JTAG controller in the FCU. Synchronous to JTAG_TCK.</td></tr>
<tr><td>JTAG_TMS</td><td>Input</td><td>Mode select input to the JTAG controller in the FCU. Synchronous to JTAG_TCK.</td></tr>
<tr><td colspan="3">Flash Memory Interface</td></tr>
<tr><td>FCU_FLASH_SCK</td><td>Output</td><td>Clock output from FCU to flash memory device(s).</td></tr>
<tr><td>FCU_FLASH_HOLDN</td><td>Output</td><td>Active-low hold output to flash memory device(s). This signal is used to pause serial communications between Speedster and the flash device without deselecting the device or stopping the serial clock. Synchronous to FLASH_SCK.</td></tr>
<tr><td>FCU_FLASH_CSN[3:0]</td><td>Output</td><td>Active-low chip select to enable/disable one or more of the attached flash memory devices.For x1 mode, only CSN[0] is used, for x4 mode connect each CSN[3:0] to a flash device</td></tr>
<tr><td colspan="3">CPU Interface</td></tr>
<tr><td>FCU_CPU_CLK</td><td>Input</td><td>Input clock from external CPU. The data/address bus is synchronous to this clock.</td></tr>
<tr><td>FCU_CPU_CSN</td><td>Input</td><td>Active-low CPU mode chip select.</td></tr>
<tr><td>FCU_CPU_DQ_IN_OUT[31:0]</td><td>Input/Output</td><td>Data Input/Output pins shared between the CPU and Flash interfaces. The CPU interface is inaccessible when the Flash mode is in use and vice-versa.</td></tr>
<tr><td>FCU_CPU_DQ_VALID</td><td>Output</td><td>Active-high control bit to indicate to the CPU the clock cycles when the CPU_DQ bus has valid read-back data. Synchronous to FCU_CPU_CLK.</td></tr>
</table>

Pin Name	Direction	Usage

Table Notes

1. FCU_CONFIG_RSTN needs to be held low, and cannot glitch during device power-up. All other input pins need only be stable when i_config_rstn is ready to be released after power-up.
2. Refer to the FCU_CPU_CSN Behavior and Implementation Details section of the user guide for details.
3. All configuration status related output signals are driven from registers. The reset value for these registers is '0', and the transition from '0' to '1' is glitch free after reset de-assertion and when reaching the appropriate FCU states.
4. Refer to the Power-Up and Configuration Sequence section of the user guide for details.
5. FCU_CPU_CLK can either start with a rising or a falling edge

Chapter - 5: FPGA Configuration Unit (FCU)

The FPGA configuration unit (FCU) refers to logic that controls the configuration process of the Speedster7t FPGA. It is responsible for receiving data on a variety of core interfaces (depending on a selected programming mode), decoding instructions, and sending configuration bit values to the appropriate destination (core configuration memory, the core's boundary ring configuration memory, FCU registers, etc.). The FCU is also responsible for any core-level housekeeping that happens on reset de-assertion (e.g., clearing of configuration memory) as well as controlling the startup and shutdown sequences that drive resets to the rest of the core as well as CRC checks, SEU mitigation and security.

Features

The following features are supported by the FCU:

- Multiple configuration modes (see Configuration Modes)
- Bitstream CRC
- AES encryption/decryption and bitstream security
- Configuration memory scrubbing and SEU mitigation (single-bit error correction, dual-bit error detection)
- Read-back

The FCU has two operating modes:

- **Power-on** – Triggered after the input `FCU_CONFIG_RSTN` is driven high. Once the FCU state machine starts, it progresses through a number of housekeeping activities, including the clearing of the configuration memory if needed. All of this processing happens without any additional inputs from the user; all instructions sent via one of the programming interfaces during this time are ignored. At the end of this mode the output pin `FCU_CONFIG_STATUS` is driven high (it was driven low earlier) and the FCU returns to the instruction processing mode.
- **Instruction processing** – The main mode of operation for the state machine. In this mode, the FCU functions as a simple CPU, processing incoming instructions and sending control signals downstream as directed. Instructions are received on 128-bit boundaries but processed 32 bits per clock cycle. The FCU can request data from the host or stall when it is processing the previous instruction. Depending on the programming interface being used, a set of output status signals generated by the FCU are used to determine how to proceed. Refer to Configuration Modes and FCU Command List for additional details.

FCU AXI Lite Master and Slave

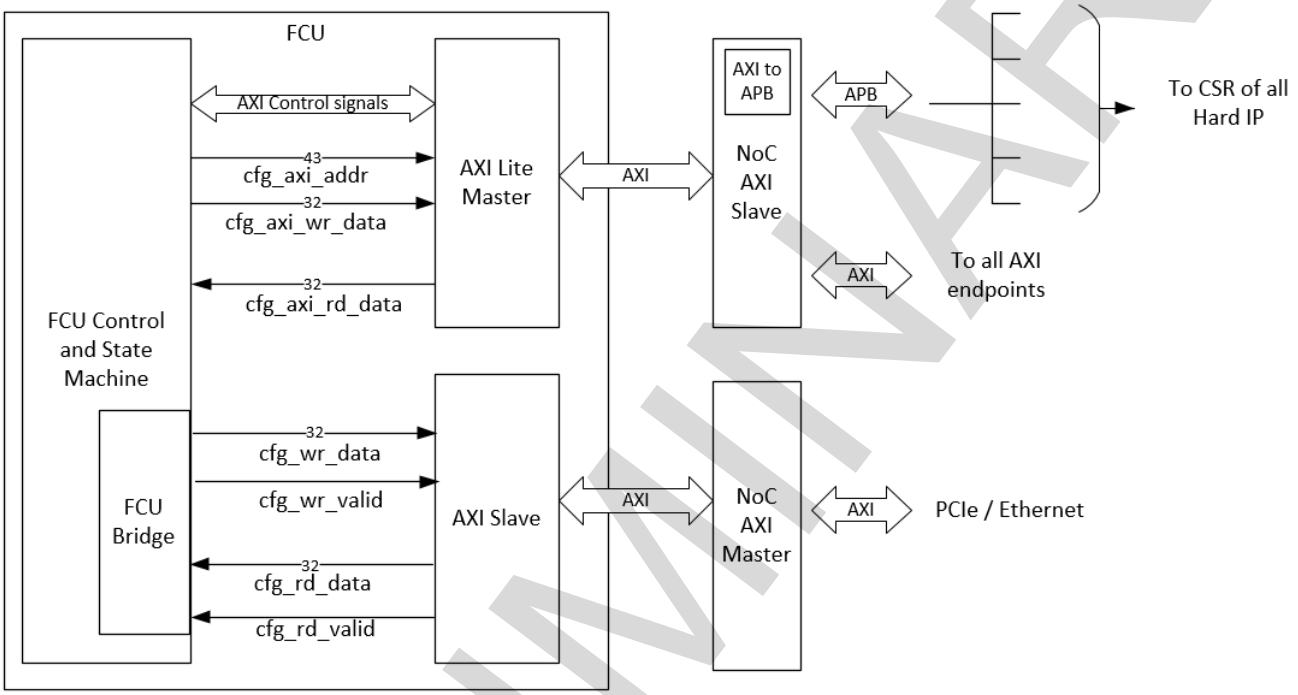
FCU configures all the hard IP Configuration Space Registers during bitstream programming. This includes, PLL and hard IPs like PCIe, DDR4, GDDR6 and Ethernet. FCU interfaces to an AXI Lite master which can be used to program the configuration registers of all hard IP and drive all AXI endpoints on the chip. It also interfaces to an AXI slave which interfaces to the NoC. A bitstream can be sent to the FCU via PCIe/Ethernet through the NoC interface.

AXI Lite Master

During bitstream programming, the FCU receives 128 bit segment of the bitstream every four clocks. The segment comprises of a 4-bit AXI write command, 43-bit address, 32-bits data payload and padding. Writes to the configuration space registers of the hard IP is done by the FCU AXI master which forms AXI write transactions by sending the address, write data and command to the NoC AXI slave.

The NoC AXI slave converts these AXI transactions to APB transactions and configures the hard IP configuration space registers. AXI writes are non-blocking i.e. the FCU does not accept any back-pressure from the NoC AXI Slave and response is not checked. The NoC AXI Slave is responsible for maintaining throughput requirements.

During reads, FCU receives AXI Read command and address. FCU forms AXI read transactions by sending address and command to the NoC AXI slave. FCU waits for the responses from the NoC AXI slave, latches the data and then accepts new commands.



AXI Slave

FCU AXI bridge communicates with PCIe or user logic via NoC AXI master interface. The NoC AXI master sends a transaction to the FCU according to the AXI slave address map. The FCU AXI slave receives the data from NoC AXI master and converts it to FCU specific packet format. The FCU bridge handles data transfer between AXI slave and FCU.

PCIe mode is enabled by writing the PCIe mode-enable register in the FCU address space [OFFSET ADDRESS=0x0]. PCIe sends the FCU address and writes the PCIe mode-enable register and the same register is read back. Once acknowledged, the bitstream can be sent from the PCIe interface via the FCU AXI Slave.

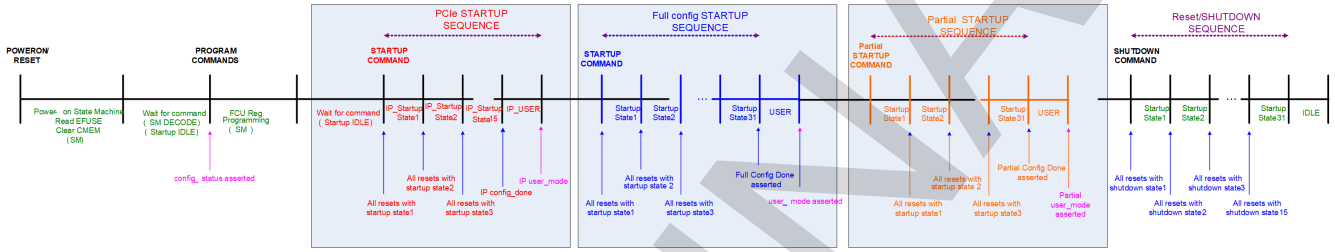
PCIe mode enable bit overrides all other external settings on the `FCU_CONFIG_MODESEL[3:0]` pins. The first two CMEM reads are dummy and the FCU AXI slave responds to the master with zero data.

CRC

If CRC is enabled completely, an accumulative CRC is computed for each 128-bit data packet that comes through the datamux. The final CRC must match a hard-coded value in order to allow a startup or shutdown sequence to begin. The CRC register is set to 32'hFFFFFFFF on reset and whenever the CRC register is written to. The current CRC computation can be read back at any time through an FCU register. CRC can also be completely bypassed.

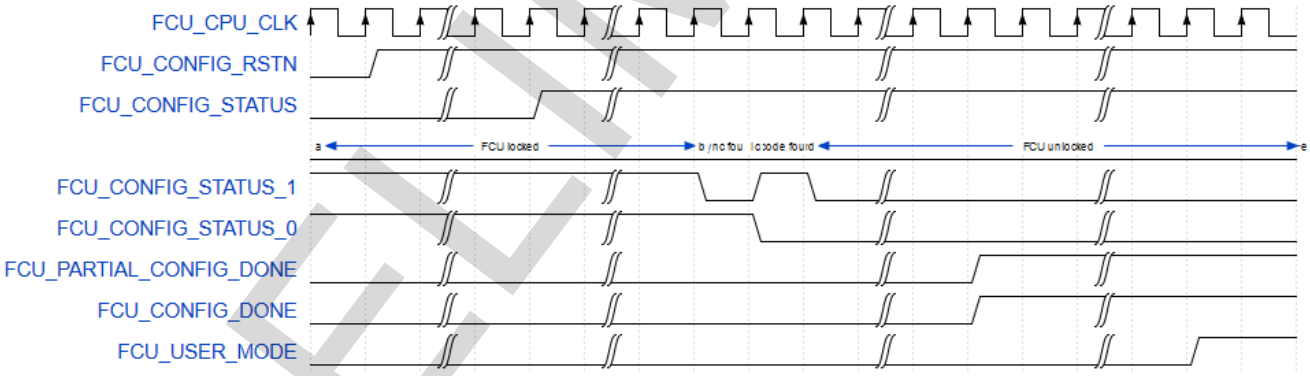
Chapter - 6: Configuration Sequence and Power-Up

The FCU has a startup sequencing block responsible for the initial power-up sequence out of reset. During power-on and bitstream programming, the startup state machine remains in its default IDLE state. After programming is finished and the chip is ready to be put into user mode, the state machine progresses through a number of startup states, de-asserting resets to the rest of the chip in a certain sequence. The final state of the startup process is user mode where it remains until it receives a request to initiate the shutdown process. The shutdown process is much like the startup process, but done in reverse (asserting resets along the way) and ending in the IDLE state.



The FCU startup sequencing block has three stages, the first two to support two-stage programming of the fabric and the third for partial reconfiguration.

After receiving a trigger, the state machines progresses through 32 start-up (or shut-down) states. There is an option of having each state wait for one or more PLLs to lock before continuing to the next state. The final startup state waits for assertion of `FCU_CONFIG_DONE` signal before asserting `FCU_USER_MODE`.



The FCU startup state machine generates 32 resets where 16 resets are connected to fabric and the other 16 resets are connected to the hard IPs. The fabric resets are staggered to avoid inrush currents.

Chapter - 7: Partial Reconfiguration

Partial reconfiguration enables the user to reprogram a part of the fabric with a smaller bitstream. Each region that can be reconfigured independently is called a fabric cluster or just cluster. The Speedster7t FPGA has 80 clusters which can be reconfigured independently. Partial reconfiguration can only be initiated after the device has entered user-mode.

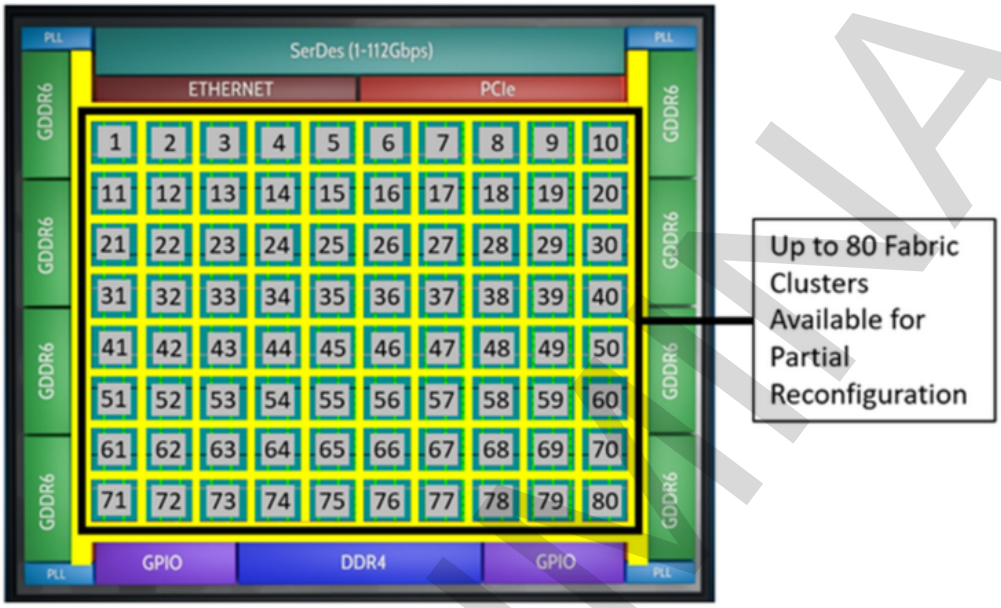


Figure - Partial reconfiguration available on any of the 80 fabric clusters connected to user instantiated NAPs

There are many advantages of partial-reconfiguration:

- Enable dynamic functionality for certain blocks in the design
- Smaller FPGA logic, functionality can be programmed on the FPGA when needed
- Faster programming times

Design considerations

Partial-reconfiguration introduces additional complexity in the design. Defining correct functional hierarchy is very important for designs that use partially reconfigurable modules. It is important to ensure that there would be no functional issues when the target module is being partially reconfigured and no outputs driven from that module are being actively used during partial-bitstream programming since the rest of the FPGA fabric is alive and performing regular tasks during partial reconfiguration.

Timing paths into and out of the module may change after partial reconfiguration and it is important to ensure that there are no timing violations after partial-reconfiguration for a design that met timing earlier. A good practice is to use the most challenging module for initial timing closure and ideally all inputs and outputs are registered.

Also, port definitions for the the new module and the module being swapped out must be the same. Reset scheme for the target module should be correctly defined and understood.

It is also important to define the correct placement constraints so that the target module is completely contained within the cluster marked for partial-reconfiguration and the resources for the module do not exceed the available resources for a cluster and optimizations across the cluster are disabled.

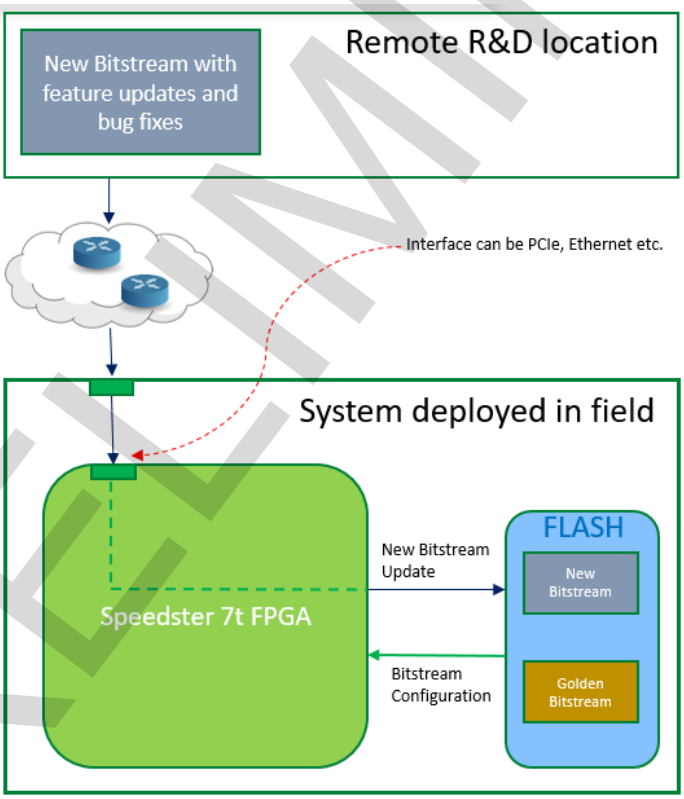
Chapter - 8: Remote Update

Introduction

Remote Update feature in Speedster7t FPGAs implements device reconfiguration using dedicated remote system upgrade circuitry in the FCU. The ability to upgrade an image remotely on an FPGA deployed in the field helps the user deliver feature enhancements and bug fixes without recalling the product, reduces time-to-market and extends product life.

The Remote Update logic within the FCU commands the configuration module to start a reconfiguration cycle. Error detection is enabled during and after the configuration process. If any errors are detected, the logic facilitates system recovery by reverting back to a safe, default factory configuration image and then provides error status information.

Implementation



1. The flash device holds two bitstreams:

- Known good working image, "Golden bitstream".
- New bitstream with enhanced features and/or bug fixes, "New Bitstream".

2. Initially the device boots from the golden bitstream and enters user mode.

3. System software initializes the "Current Bitstream Address" register, which is the start address of the New bitstream programmed by the user.

4. System software initializes the "Golden Bitstream Address" register, which is the start address of the Golden bitstream programmed by the user.

5. Based on configuration modes, software writes the command and start address into the flash configuration header

6. System initiates a reset, the FPGA re-configures from the Current Bitstream Address and reads the first 512 bits of bit stream data from flash device and enters in to wait state.

7. If encryption is not enabled, read the complete bit stream and re-configure the FPGA.

8. If encryption is enabled and efuse key is ready:

- Read the header segment0 data and send to secure boot core.
- Flash read state machine enters in to wait state of 2.6 ms.
- Flash interface read the complete bitstream and configure the FPGA.

Fallback on Error

After bitstream load, failure can happen in two scenarios:

- No IDCODE match after timeout expires
- CRC error after timeout expires

If any of these checks fail, retry N times, the number of retries is described in the flash configuration header.

If the failures persist and the system is unable to boot from the New Bitstream and fallback is in the fpga configuration header, then issue fast read to header fallback address.

The user should then update the New Bitstream or point the default boot address to the Golden Bitstream.

Chapter - 9: Design Security for Speedster 7t FPGA

Achronix recognizes the importance of protecting the sensitive IP a customer downloads onto their FPGA. To provide a high level of protection, Speedster7t FPGAs have a number of features to support bitstream encryption as well as authentication. These features ensure that no one can access the design configuration on the FPGA and also ensures that the design is the intended design. Speedster7t FPGAs provide this high level of security through the following features:

- Support for RSA authenticated and AES-GCM encrypted bitstream
- Dynamic power analysis (DPA) protection to prevent side-channel attacks
- Physically unclonable function (PUF) for tamper-proof protection
- Securely stores both public and encrypted private keys

With this security solution deployed, a customer's design is secure. Even with possession of the device, no one can extract the underlying design, the design cannot be reverse engineered, nor can the design be altered in any way.

Bitstream Authentication

Authentication of a bitstream ensures that the design on the device is the intended design. Achronix provides a two-step authentication process that first authenticates an encrypted bitstream before decrypting it, and then performs authentication a second time on the decrypted bitstream before configuring the device. First, a bitstream is encrypted using AES-GCM, which provides authenticated encryption. Next, the user provides an asymmetric private key to sign the encrypted bitstream using RSA-2048. Then, when an encrypted and signed bitstream is loaded into the FPGA, the device uses the public key stored in an electronic fuse (eFuse) on the device to authenticate the bitstream using the public key. Once authenticated, the bitstream decryption is enabled, and the bitstream is authenticated a second time while decrypting with AES-GCM. After the second authentication, the bitstream is used to configure the FPGA.

Bitstream Encryption

Bitstreams consist of sensitive intellectual property of the designer. Achronix provides tools to generate bitstreams that are encrypted and signed using very strong encryption with hardware designed to be resilient to side-channel attacks, such as dynamic power analysis (DPA). Additionally, the key derivation function (KDF) inside the secure boot portion of the FPGA, along with the physically unclonable function (PUF) ensure protection of the secret keys to decode and authenticate the bitstreams. Together these systems provide a solution that is safe from attacks such that even with possession of the device, an adversary cannot extract the underlying design, cannot change the system to perform another task other than the intended task, and cannot reverse engineer the core intellectual property.

The figure below shows an overview of the security system and how everything works together to protect the bitstream. The portions shown in yellow represent the blocks used for encryption/decryption, the blocks in blue are for authentication, and the green portions are areas handling authenticated and encrypted bitstreams.

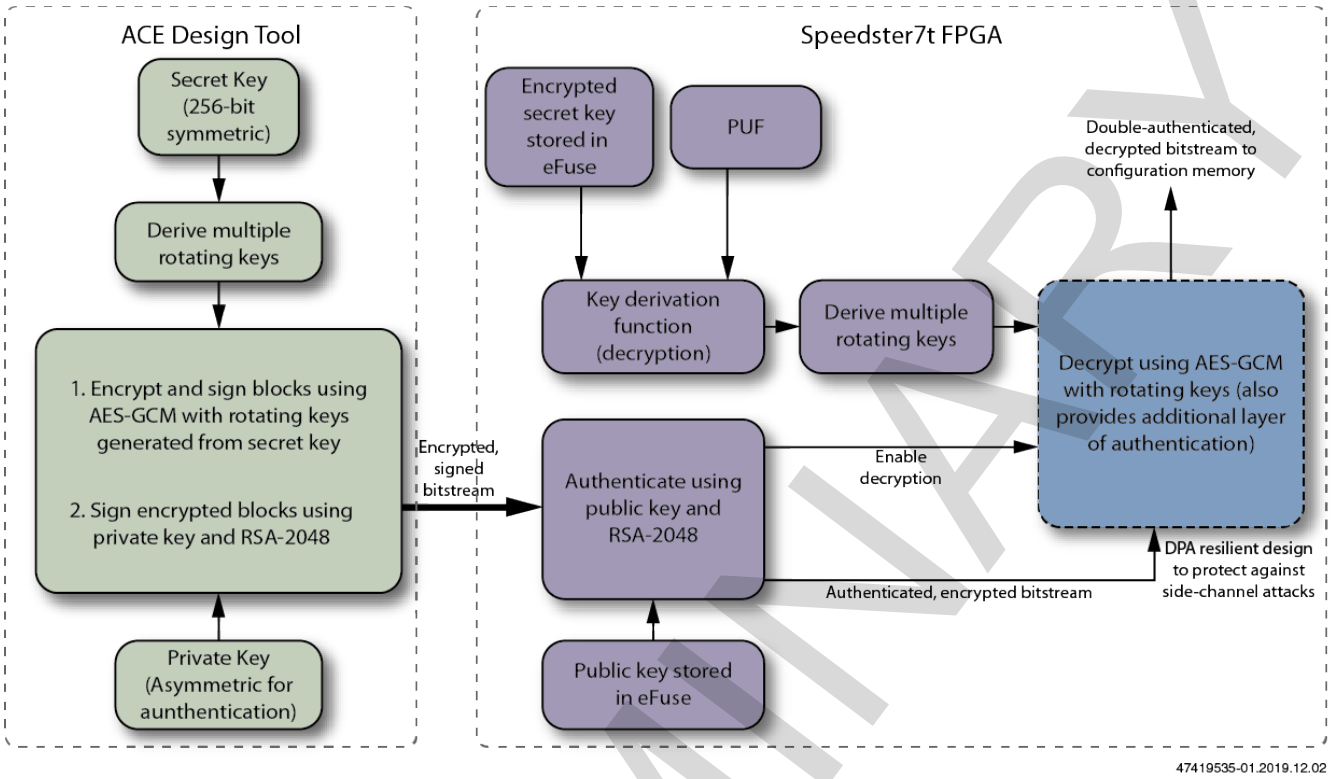


Figure 19: ***Bitstream Encryption/Authentication Block Diagram***

Generating Encrypted Bitstreams

To generate an encrypted bitstream, the user provides a 256-bit secret key to ACE. In order to provide better protection against side-channel attacks, ACE does not simply use this secret key to encrypt the entire bitstream. Instead, the secret key is used as an initial key. ACE then generates new derived keys based on the initial secret key to encrypt smaller segments of the bitstream, each with a different derived key and new nonce. Here the nonce, also known as an initialization vector (IV), is a random number only used once per segment, such that the same pattern is not generated while replaying or encrypting the same bitstream. Bitstream encryption is performed using the highly secure 256-bit AES-GCM encryption standard. Galois/counter mode (GCM) is an advanced form of symmetric-key block encryption which enhances 256-bit advanced encryption standard (AES) by using a nonce (one-time use random value) and a counter mode so that each segment of data is uniquely encrypted. ACE also uses a Galois message authentication code (GMAC) to simultaneously sign and authenticate the data, including the unencrypted preamble section of the bitstream to guarantee the bitstream has not been altered. To further protect the bitstream, ACE also signs each segment of the encrypted bitstream using RSA-2048. See the section on Bitstream Authentication (see page 43) above for more details on the RSA-2048 authentication.

Hardware Security

There are several security features available in the hardware to support decryption of encrypted bitstreams, safe storage of secret keys, and strict rule enforcement such that the device will be locked if security rules are violated. The main features for decryption and safe storage of keys use the physically unclonable function (PUF) which provides a unique secret value per individual chip, and the key derivation function (KDF) which uses the PUF as the key to encrypt/decrypt the real secret keys from the encrypted keys that are stored in an electronic fuse (eFuse).

Physically Unclonable Function

The PUF generates a unique secret identifier for each individual chip. It is created from random physical variations that occur during the semiconductor manufacturing process, such that the same circuit on a device creates completely different and unique values on each chip, even chips on the same wafer. The value of the PUF is random per individual chip, but remains constant over the lifetime of that chip. The PUF value is not known to Achronix or the manufacturer, and the value cannot be observed without destroying or altering the value of the PUF. This PUF value can be used to encrypt the user's secret key and store an encrypted version of the secret key in an eFuse. Then when an encrypted bitstream is loaded into the FPGA, the PUF value is used to temporarily decrypt the stored encrypted secret key. This secret key is then used to generate the multiple rotating keys to decrypt the bitstream blocks that configure the FPGA.

Key Derivation Function

The KDF uses 256-bit AES encryption in conjunction with the PUF to create an encrypted version of the user's secret key that can be stored in an eFuse. While it is theoretically possible to observe the contents of the eFuse if an adversary is in possession of the device and has access to advanced reverse engineering equipment, the stored key is an encrypted version of the secret key that uses the PUF value as the master key for encryption. Again, the PUF value cannot be known and is unique to each individual device, thus making the stored key safe. Additionally, when the KDF needs to decrypt an encrypted bitstream, it loads the encrypted key from the eFuse along with the PUF value and temporarily decrypts the secret key. The secret key is then used as the initial key for the module that generates the multiple derived keys for AES-GCM decryption of the bitstream before loading it to configuration memory in the FPGA.

The two figures below showing how the PUF and KDF are used to generate a secure encrypted key to store in an eFuse, as well as how they are used to recreate the secret key to decrypt the bitstream.

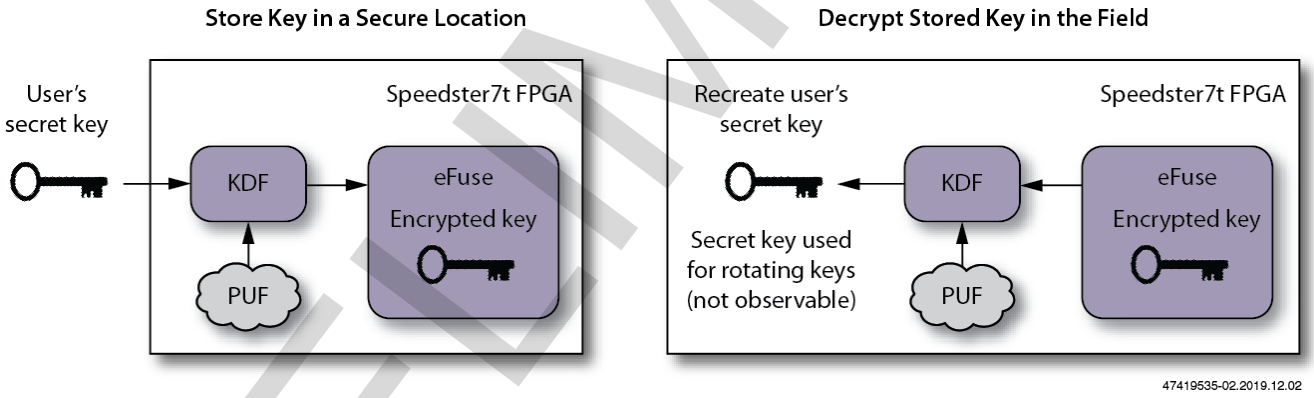


Figure 20: ***Safe Secret Key Storage***

Rules for Encryption

When using encrypted bitstreams, the FPGA device enforces a set of rules. If the security rules are violated, the FPGA locks up and cannot be used in any way without powering down the device. First, there is an ordering rule to how bitstreams can be loaded. There are three phases for bitstreams for Speedster7t devices, and they must follow these ordering rules.

1. Zero, one, or multiple pre-configuration bitstreams.
2. One, and only one, full configuration bitstream.
3. Zero, one, or multiple partial reconfiguration bitstreams.

Additionally, there are rules about which keys can be used for the encryption. The eFuses can store up to four secret keys — bitstreams can be encrypted using up to four different initial keys. The following rules must be followed to prevent locking the device.

1. If the `encrypted_bitstreams_only` eFuse bit has been set for the FPGA, the device will only accept encrypted bitstreams.
2. If any pre-configuration bitstream is encrypted, all pre-configuration bitstreams must be encrypted using the same key.
3. If either the pre-configuration bitstream or the full bitstream are encrypted, they both must be encrypted and both must use the same key.
4. Any partial reconfiguration bitstreams may use a different key if and only if the previous bitstream sets the `same_key` bit to 0 in the preamble, and the partial reconfiguration bitstream also sets that same bit to 0 in its preamble.

> **Note**
>
> It is acceptable to load an unencrypted bitstream after a previous encrypted bitstream. It is *not* acceptable to load an encrypted bitstream after a previous unencrypted bitstream.

Security Fuses

There are several eFuses that are related to the security features in Speedster7t devices. Some of these are set during manufacturing and cannot be changed by the customer, and others are available for customer use.

Fuses Set at Manufacturing

There are two fuses that can be set at manufacturing time to limit the features of the FPGA. The part number of the device indicates if these limitations exist in a part.

- **Bitstream decrypt disable** – If set, the FPGA cannot accept encrypted bitstreams.
- **DPA disable for bitstream decrypt** – If set, the FPGA still supports encrypted bitstreams, but there is limited hardware protection for differential power analysis (DPA) side-channel attacks that can potentially expose secret keys.

Fuses Set By Customer

There are several eFuses that can be set by the customer if using encrypted bitstreams:

- **Bitstream authentication key** – This fuse contains a 768-bit hash of the public key used for first-level authentication of encrypted bitstreams. This fuse is not readable.
- **Bitstream decryption key** – These fuses contain the four 256-bit secret keys that can be used for decryption and authentication of encrypted bitstreams. This fuse can contain the actual secret keys or the encrypted version of the secret keys (using PUF and KDF). This fuse is not readable.
- **Bitstream user register** – This fuse contains the 32-bit value set by the user to identify the key version used. The secret key itself cannot be read back, but the user register value can be read. The user keeps a mapping of key versions to keys.
- **Bitstream user lock** – This one-bit fuse, if set, disables further updates to authentication key, decryption key, and user register.
- **Encrypted bitstreams only** – This one-bit fuse, if set, forces the FPGA to only accept encrypted bitstreams that use one of the keys in the fuses.

Default Keys

Achronix provides a default public key for authentication and a default secret key for encryption/decryption of the bitstream. These keys are available to use for testing, so that a user has confidence the security system works. The default keys should not be used to protect sensitive designs — they are only made available for testing purposes. Additionally, once a user sets the eFuse to only accept encrypted bitstreams, the FPGA no longer accepts the default keys.

Loading Encrypted Bitstreams

Loading an encrypted bitstream is similar to loading an unencrypted bitstream. However, the most important difference is that once the unencrypted 512-bit preamble of the bitstream is loaded, the FPGA disables all read-out of any data, thus securing the device containing a user's sensitive IP, protecting it from being known, reverse engineered, or altered in any way. Below are the steps for loading encrypted bitstreams:

1. When the hardware detects an encrypted bitstream is being loaded, all readout and debug features are disabled by the hardware, disabling the ability for a user to read any internal state related to the FPGA fabric or the FCU.
2. Security rules for loading encrypted bitstreams are checked. If the rule checker fails, the FPGA enters a locked state and can only be re-enabled by a power cycle.

If the user's system uses a board management controller to load in the bitstreams, there are additional requirements the user needs to be aware of:

1. After the 512-bit preamble of the bitstream, the board management controller must pause and wait for some number of FCU clock cycles before sending the next portion of the encrypted bitstream.
2. After the first 12,688 bytes of the encrypted bitstream the board management controller must pause and wait at least 520,000 FCU clocks, or about 2 ms (assuming a 32-bit data path and 250 MHz FCU clock).
3. For encrypted bitstreams, a board management controller is limited to sending 32-bits per FCU clock. For unencrypted bitstreams, it can send data at a rate up to 128-bits per FCU clock.

Note

When using encrypted bitstreams, it is *not* possible to use any debug features of the FPGA. Debug features are *only* available when using unencrypted bitstreams.

Revision History

Version	Date	Description
0.1	12 Feb 2020	Initial draft release.