Speedster22i Configuration User Guide

UG033 – March 7, 2016

Copyright Info

Table of Contents

Copyright Info **2**

Table of Contents **3**

Overview **4**

Power-Up and Configuration Sequence **5**

Device Power-Up 5
Provide SerDes Reference and Sbus Clocks 5
Read Non-Volatile Memories 6
Clear Configuration Memory 6
Bitstream Sync and Device ID 6
Load Configuration Bits 6
CRC Check 7
Startup Sequence 7
User Mode 7

Configuration Modes and Pins **8**

CPU 9

Serial x1 Flash 11

Flash Programming on the Achronix Development Board 12

Serial x4 Flash 15

JTAG 17

Configuration Pins and Clock Selection 18

Bitstream File Generation Through ACE **20**

Design Security **22**

Revision History **24**

Overview

The configuration architecture in Speedster22i HD devices is composed of a few key pieces:

1. Configuration pins enabling data transfer from an external interface to the FPGA
2. FPGA Configuration Unit (FCU) which is the IP block containing the modes, interfaces, state machines and other control logic to take data from the pins, perform the necessary FPGA mode transitions and assemble the incoming data stream into a form to be ultimately provided to the rest of the FPGA
3. Configuration registers in the IO ring and the configuration memory in the fabric core which are the recipients of the bitstream data coming from the FCU

Figure 1 below provides a block level diagram showing the pieces of the configuration architecture . Data from the configuration pins are brought into the FCU located in the IO ring. Depending on the configuration mode, this data passes through one of four interfaces and is then provided into the control logic and state machines in the FCU. At this point, the data bus is standardized to a common interface. This data is interpreted here and then fans out to the configuration registers in the IO ring and a bus to be parallelly loaded into column based configuration memory frames in the FPGA fabric core. Once all of the configuration bits have been successfully loaded, the FCU transitions the FPGA into user mode, providing the capability for the user to provide stimuli and enable operation.

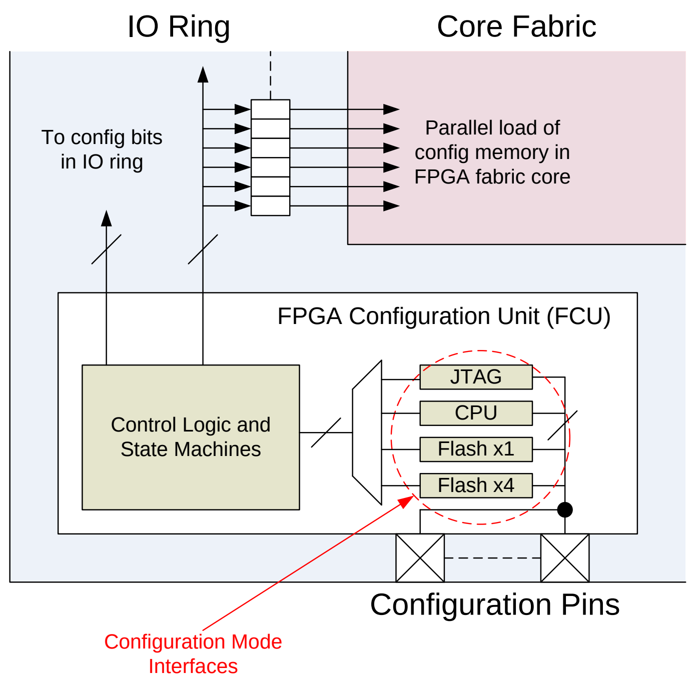

IO Ring
Core Fabric
To config bits in IO ring
Parallel load of config memory in FPGA fabric core
FPGA Configuration Unit (FCU)
Control Logic and State Machines
JTAG
CPU
Flash x1
Flash x4
Configuration Pins
Configuration Mode Interfaces

Figure 1: FPGA Configuration Blocks

Power-Up and Configuration Sequence

The requirements for the power-up and configuration sequencing for Speedster22i HD devices are illustrated in Table 1 and detailed below.

Table 1: Power-Up and Configuration Sequencing Steps

1	2	3	4	5	6	7	8	9
Device power-up	Provide SerDes reference and Sbus clocks	Read non-volatile memories	Clear configuration memory	Bitstream sync and device ID	Load configuration bits	CRC check	Startup sequence	User Mode

Device Power-Up: The first step in bringing up the Speedster22i HD FPGAs is to appropriately power it up. The Power Sequencing section of the Speedster22i Pin Connections and Power Supply Sequencing User Guide provides an illustration of how the power supplies and configuration related pins/signals need to be asserted to ensure a successful FPGA power-up. To summarize these requirements:

a. Power-up all power supplies except for PA_VDD1, PA_VDD2 and VDDL to full rail.

b. Power-up PA_VDD1 and PA_VDD2 after VCC reaches full rail.

c. CONFIG_RSTN should be low while the above rails are being powered-up. This will ensure that the FPGA powers-up in a reset state.

d. At this point, start providing FCU clocks on the appropriate clock input (based on the configuration mode), to ensure that the FCU state machine will be active and cycle through the appropriate states when CONFIG_RSTN is released. Note that the FCU_CLK is limited to 6MHz in all configuration modes.

e. After some time (~ms), release CONFIG_RSTN. Once the FPGA is out of reset, steps 2 and 3 in Table 1 above, the reading of the non-volatile memories and the clearing of the configuration memory, will be performed. After the configuration memory is cleared, the CONFIG_STATUS signal will be released.

f. Power-up VDDL and wait for it to reach full rail. At this point, the FPGA is ready to accept the bitstream.

Provide SerDes Reference and Sbus Clocks: Enable on-board circuitry and oscillators to provide free-running clocks to the SerDes reference and Sbus clock input pins. Before, during and after configuration, the clocks must not be stopped or varied in any way. The Sbus clock must be less than 100 MHz and must be less than the SerDes reference clock frequency.

Read Non-Volatile Memories: After coming out of reset, the FCU reads the non-volatile memory (fuse) contents and latches the data coming out. The fuses are factory set to zero and can be programmed. Manufacturing and ID related fuses are set during ATE testing. Fuses that pertain to design security are available for customers to program. Please refer to the section on Design Security for operation details.

Clear Configuration Memory: After the non-volatile memory is read, the FCU enters the state to clear configuration memory. Configuration memory cells are 6-T SRAM cells and are cleared one frame at a time by writing 0s into them. If this state is entered after a full FPGA power-up, it is imperative that all configuration memory be cleared prior to powering up VDDL. Otherwise, with SRAM cells powering up in unknown states, the presence of one-hot muxes in the routing interconnect will undoubtedly mean that there will be shorts leading to contention, and as a result unexpected behavior as far as current profiles and draws.

Once the memory clear is complete, the pin CONFIG_STATUS is released by the device, and the weak external pull-up will pull this signal high to indicate that the FCU is ready to read the bitstream.

It should be noted that only the configuration memory is cleared in this step. The embedded BRAM and LRAM memory cells are NOT cleared and should be assumed to power up to unknown states after configuration and in user mode. There is a separate option to preload the memory contents using an initialization file.

Another important point is that when the device powers up, the CONFIG_STATUS output pin may temporarily be in an unknown state. The CONFIG_STATUS signal should therefore not be monitored if a CONFIG_RSTN pulse has not been issued. After a pulse has been issued on CONFIG_RSTN, CONFIG_STATUS can be monitored. When CONFIG_STATUS goes high after that point, VDDL can be powered-up and the bitstream can be provided.

Bitstream Sync and Device ID: Speedster22i HD FPGA bitstreams always start with a sync code which is pre-programmed to 0xAA55AA55. The sync code is always written in the bitstream by the ACE software and is transparent to the user. Followed by sync, a Device specific ID Code is checked to avoid programming the device with bitstream meant for other devices. This is also a pre-programmed code provided by ACE.

Load Configuration Bits: The configuration bitstream is a series of data words which are ultimately made to internally form a bus and get shifted into a register chain before being parallelly loaded into configuration memory frames in the FPGA fabric. There are also command words which control whether the IO ring configuration registers or the core configuration memory gets loaded.

The configuration file size and the configuration time are directly proportional to the number of configuration memory frames that need to be programmed in the FPGA fabric. The configuration file size is also dependent on the programming mode used, but strictly as a raw hex file (see section on Bitstream File Generation Through ACE below) the bitstream size can vary from <1MB for very small designs to close to 100MB for the largest designs that fill up the entire FPGA and preload the BRAM memories.

CRC Check: At the end of the bitstream, a CRC check is performed on the bitstream to ensure that the data going into the configuration memory is error-free. This is disabled in ACE for ES devices, but will be done by default on all production devices.

Startup Sequence: After the configuration memory is programmed, the command sequence to enter the user mode can be issued by the bitstream. Entering and exiting user mode is controlled by a startup/shutdown state machine also implemented in the FCU. This operates independently of the configuration state machine and so the configuration state machine can process bitstream commands even after entering user mode.

The startup sequence consists of sequentially asserting a number of signals to ensure proper operation during user mode. These events are highlighted in Table 2 below.

Table 2: Startup Sequence Events

Stage	Event
1	Assert Global Clock Enable
2	Assert I/O Enable
3	Assert Global Reset Enable
4	Assert Global Core Enable
5	Assert Config Done
6	Assert User Mode Enable

The shutdown sequence is very similar in nature to the startup sequence and essentially entails deasserting these same signals in reverse order.

User Mode: Once the device enters user mode, the design has been fully programmed and the user can start sending and receiving data to/from the FPGA and performing intended operations.

If, at any point in time, a re-configuration is desired to load a different bitstream, this can be done using the following steps:

a. Bring CONFIG_RSTN low. This will reset the FCU.

b. Provide FCU clocks on the appropriate clock input as before.

c. Release CONFIG_RSTN. Once the FPGA is out of reset, the reading of the non-volatile memories and the clearing of the configuration memory, will be performed. The CONFIG_STATUS signal will then be released.

d. There is no need to power down and re-power up VDDL. VDDL can be kept at the nominal level during this entire process. The FPGA is ready to accept the bitstream.

Configuration Modes and Pins

Speedster22iHD devices have four configuration modes: CPU, Serial Flash x1, Serial Flash x4 and JTAG. The selection between the first 3 is done by tying CONFIG_MODESEL pins to the values shown in Table 3. The fourth configuration mode, which is JTAG, is independent of the mode pins and can be enabled by setting the appropriate bits in the User Data Register of the JTAG TAP Controller. Once JTAG mode is enabled, it overrides all other configuration modes until disabled.

Table 3: Configuration Modes and CONFIG_MODESEL Settings

Configuration Mode	CONFIG_MODESEL[2:0]
CPU	100
Serial x1 Flash	001
Serial x4 Flash	010
JTAG	Always active

Figure 2 below shows a simplified block diagram view of the different configuration interfaces connecting up to the Speedster22iHD FPGA.

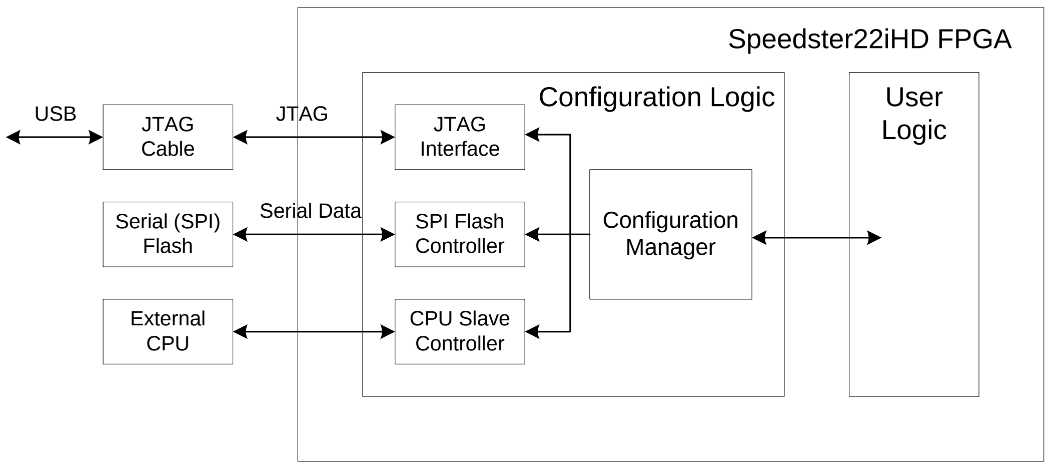


Figure 2: Configuration Interface Connections to the Speedster22iHD FPGA

CPU

In CPU mode, an external CPU acts as the master and controls the programming operations for the FPGA. CPU mode is an 8-bit parallel interface, clocked using CPU_CLK, with chip select support to indicate valid data. This is generally the fastest programming mode as it provides for the widest data width interface and a maximum supported clock rate of 6MHz. Figure 3 below provides a block diagram of how the external CPU would be hooked up to Speedster22iHD FPGA.

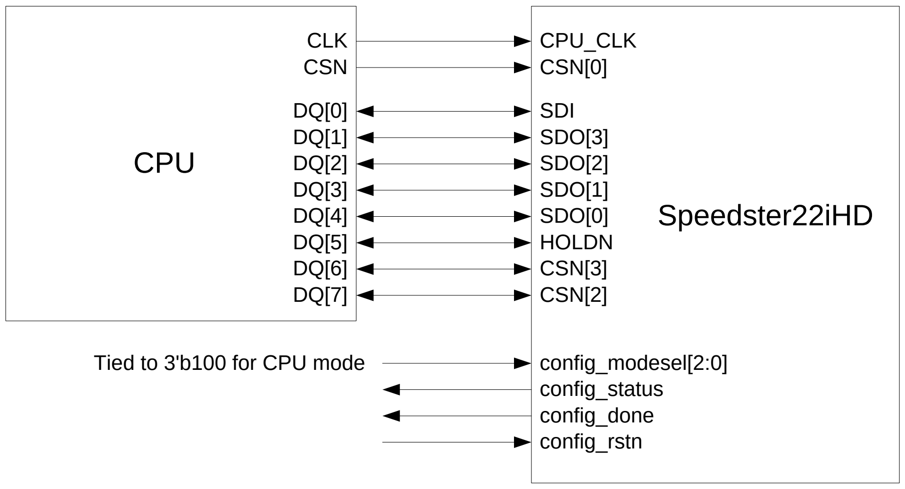


Figure 3: External CPU Connectivity to Speedster22iHD FPGA

As described in the Power-Up and Configuration Sequence section, the configuration mode specific operations occur between the release of CONFIG_STATUS (indicating that the configuration memory has been cleared and that the FPGA is ready to accept bitstream data) and the assertion of CONFIG_DONE (stating completion of configuration). The waveform in Figure 4 shows the sequence of events, clocking and control signal states needed for successful configuration in CPU mode.

Note that the data and control signals should be launched from the configuration controller on the negative edge of the CPU_CLK. The signals are latched at the FPGA on the rising edge of the CPU_CLK. With a configuration clock frequency of 6MHz (166ns), a half-clock period of 83ns would mean that no additional timing information should be necessary to ensure that the data and control signals reach the FPGA with sufficient timing margin.

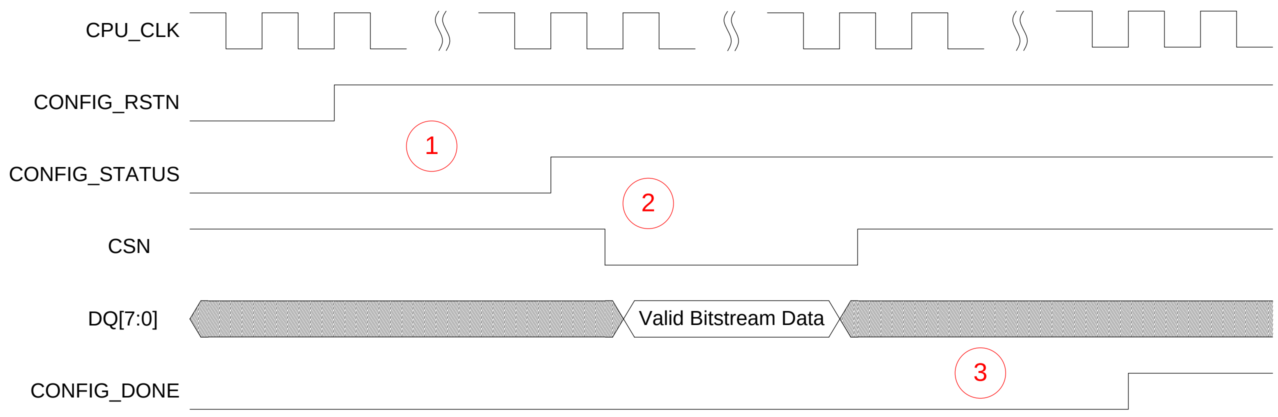


Figure 4: Clocking and Control Signals for Successful Configuration

In Figure 4 above:

1. After CONFIG_RSTN is deasserted, CPU_CLK needs to continue being clocked to ensure that the FPGA cycles through the FCU states and the configuration memory is cleared. At that point, CONFIG_STATUS is released and is pulled high.

2. Some time after CONFIG_STATUS is pulled high, CSN should be pulled low to begin writing the bitstream data into the FPGA. When the last set of data is written into the FPGA, CSN is pulled high.

3. Once CSN is pulled high, CPU_CLK needs to continue being clocked for a total of about 24,000 clock cycles. After 12,000 clock cycles, CONFIG_DONE should be asserted to indicate that configuration has completed, and the remaining 12,000 clock cycles are needed to ensure that the FCU can successfully transition into user mode.

The waveform in Figure 5 depicts the window of a sample valid bitstream programming section of the configuration.

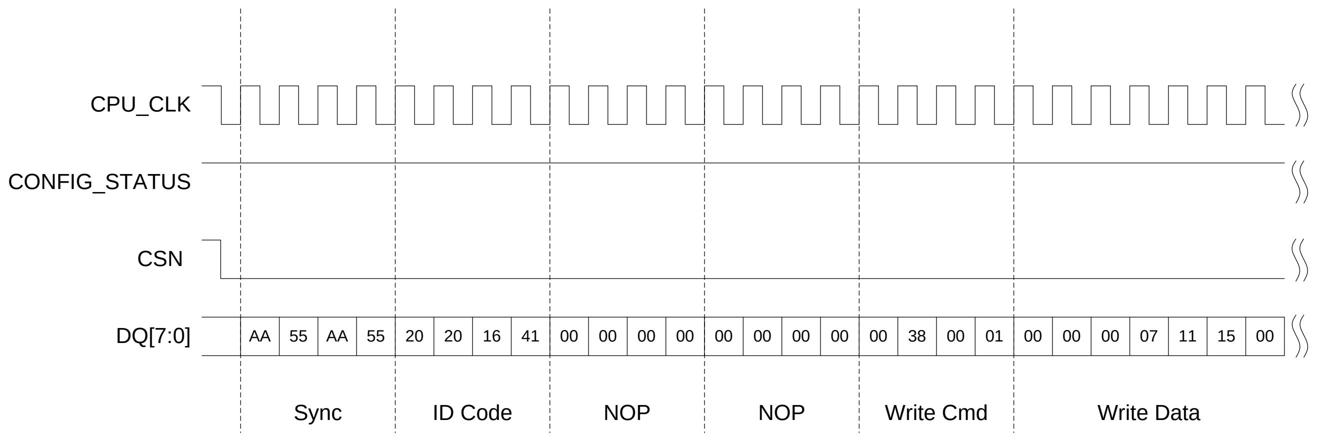

CPU_CLK
CONFIG_STATUS
CSN
DQ[7:0]
AA 55 AA 55 20 20 16 41 00 00 00 00 00 00 00 00 00 38 00 01 00 00 00 07 11 15 00
Sync
ID Code
NOP
NOP
Write Cmd
Write Data

Figure 5: Sample Valid Bitstream Programming

Serial x1 Flash

The Serial Flash programming mode allows flash memories to be used to configure the Speedster22iHD FPGA. In this mode the FPGA is the master, and therefore supplies the clock to the Flash memory.

There are some important considerations when choosing and deploying flash solutions to program the HD1000. These are detailed below:

1. The HD1000 can interface with NOR Flash devices only. NAND flash or other flash variants will not work.

2. Only SPI (or extended SPI mode) is supported, where the data buses are single bit unidirectional lines. Dual and quad modes are NOT supported. Note that "quad" refers to the width of a data bus interfacing with a single flash device. This is different from the x4 flash mode configuration detailed below, which is a single bit SPI interface across 4 flash devices.

3. The largest HD1000 bitstreams can be in excess of 100MB (800Mbits). It is important that the flash solution chosen is able to accommodate the bitstream. This would require a single 1Gbit flash device in x1 mode, or 4 256Mbit flash devices in x4 programming mode.

4. The HD1000 flash interface relies on 3-byte addressing. There is no support for 4-byte addressing. This means that technically, only flash devices up to 128Mbit (16MB) can be supported. However, FPGA programming from flash relies on simply doing a "FAST_READ" operation with an initial address of all 0s. As a result, even if the address is 3-bytes, the entire FPGA can be programmed successfully from many single-die flash devices larger than 128Mbits (up to 1Gbit). Stacked die solution flash devices do not work in this way, and are not recommended. Please see Table 4 below for a list of supported/qualified flash vendors and device.

5. The FPGA should be configured to only read from the flash device. Flash device programming should be done externally using a SPI header and a 3rd party solution like the Totalphase Cheetah SPI Programmer. Clear instructions on how to use this interface for flash programming on the HD1000 Development board are provided below.

Figure 6 provides a block diagram of how a serial flash can be connected to the Speedster22iHD FPGA and a SPI header for programming in x1 mode.

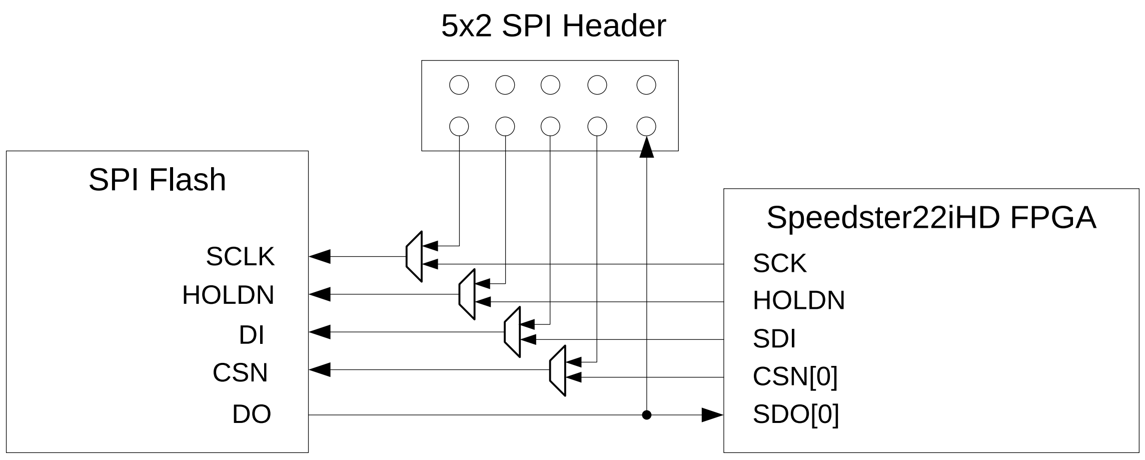


Figure 6: Flash Connectivity in x1 Mode

Table 4: Recommended Flash Vendors and Devices

Vendor	Device	Memory Size (Mbit)*	Package
Macronix	MX66L1G45GMI	1024	SOP/BGA
Macronix	MX66L51235F	512	SOP/BGA
Macronix	MX25L51245GMI	512	SOP/BGA
Spansion	S25FL512S	512	SOP/BGA
Spansion	S25FL256S	256	SOP/BGA
Micron	MT25QL01GBBA8E12	1024	BGA

* Memories with capacities below 1Gbit should only be used in x4 flash programming mode or if the bitstream is guaranteed to be small enough to be acommodated

Configuration operation in serial flash x1 mode is very similar to CPU mode. The only difference comes during the writing of the bitstream. SCK is used for clocking and the bitstream is a single bit interface provided through the SD[0] port. CSN[0] is pulled low during the valid bitstream window and is then pulled high once the last bit is clocked in. Transitioning from the end of the bitstream to user mode is done exactly as in CPU mode, with SCK providing the clock to the FPGA.

Flash Programming on the Achronix Development Board

The instructions below can be used to enable flash programming on the Achronix development board. The same methodology and steps can be used to validate flash programming on customer boards.

Writing to Flash:

1. Pre-requisities:
    a. Totalphase Cheetah Adapter.
    b. Split cable.
    c. Install USB driver for Cheetah adapter.

   d. Totalphase GUI.

2. Connect the Cheetah adapter to J36 using the split cable (See Figure 7 below):

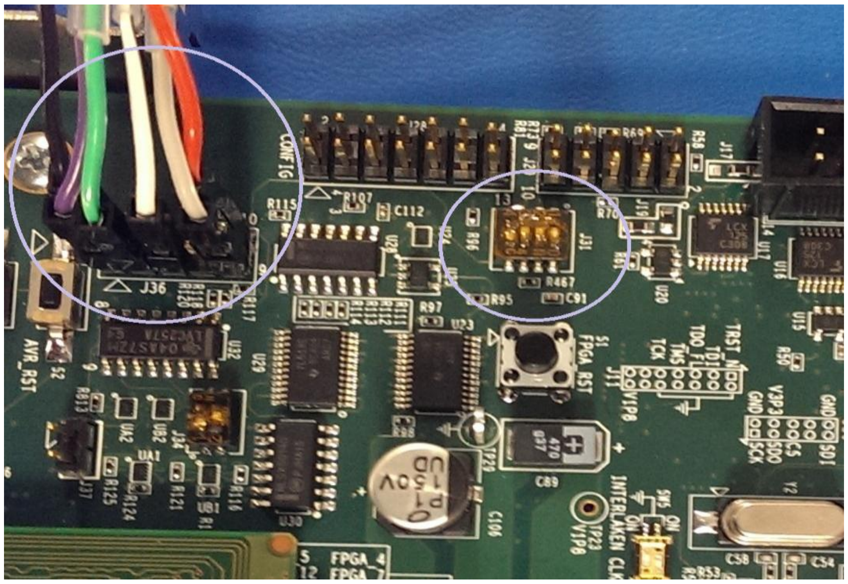


Figure 7: Achronix Development Board Configuration for Writing to Flash

The cheetah adapter connections to the SPI header are clarified below:

Black	NONE	NONE	NONE	Red
Purple	Green	White	NONE	Gray

Δ

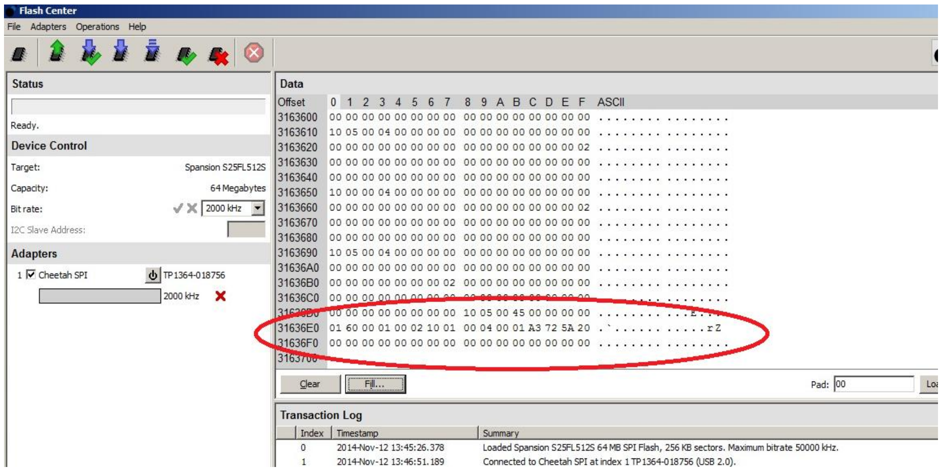
3. Set pin-4 of J31 to ON position. By default, all pins are OFF.

4. Launch Flash-Center-GUI.

5. The GUI folder contains a user-guide. Follow Section-2.4 and 2.6 to add Cheetah adapter and memory device.

6. Now, load an existing x1 Flash file using File->Load file. Refer to Appendix of this document to try an existing file.

7. You will see the contents written as shown in Figure 8 below.



Target: Spansion S25FL512S
Capacity: 64 Megabytes
Bit rate: 2000 kHz
I2C Slave Address:
1 ☑ Cheetah SPI TP1364-018756 2000 kHz

```
Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F  ASCII
3163600  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163610  10 05 00 04 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163620  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 02  ........ ........
3163630  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163640  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163650  10 00 00 04 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163660  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 02  ........ ........
3163670  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163680  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163690  10 05 00 04 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
31636A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
31636B0  00 00 00 00 00 00 00 02  00 00 00 00 00 00 00 00  ........ ........
31636C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
31636D0  00 00 00 00 00 00 00 00  10 05 00 45 00 00 00 00  ........ ...E....
31636E0  01 60 00 01 00 02 10 01  00 04 00 01 A3 72 5A 20  .`...... .....rZ
31636F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
3163700
```

Pad: 00

Index	Timestamp	Summary
0	2014-Nov-12 13:45:26.378	Loaded Spansion S25FL512S 64 MB SPI Flash, 256 KB sectors. Maximum bitrate 50000 kHz.
1	2014-Nov-12 13:46:51.189	Connected to Cheetah SPI at index 1 TP1364-018756 (USB 2.0).

Figure 8: Screenshot from Totalphase GUI when Writing to Flash

8. Now, choose the Program button to program this flash file into device. You may want to erase the entire device from the Operations menu, but that is not mandatory.

Programming FPGA from Flash:

1. The Totalphase Cheetah adapter is NOT required for reading from Flash.

2. Power off the Achronix development board.

3. Set the following two switches:
   a. **J31**: Flip #2 to ON position. Others will remain at OFF.

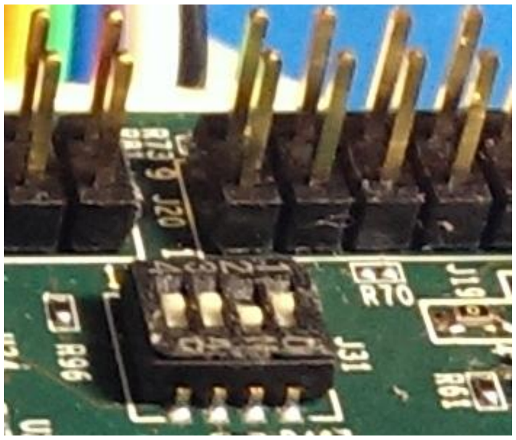


Figure 9: J31 Switch when Reading from Flash

   b. **Configuration Switch Box SW7**: #1 should be OFF and others should be ON. This will set CONFIG_MODESEL[2:0] to 001 which is what is required for x1 Flash programming.

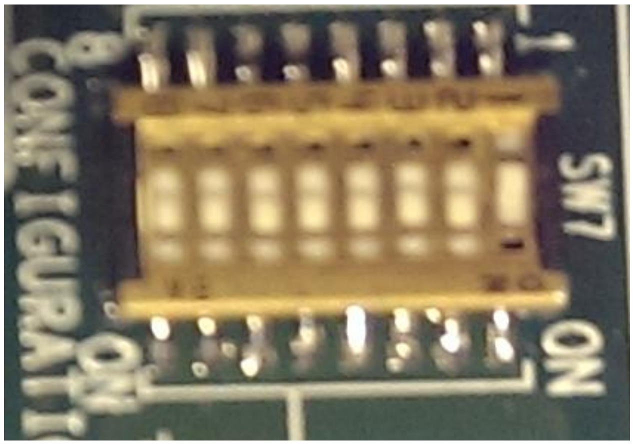


Figure 10: SW7 Configuration Switch Box for Reading from Flash in x1 Mode

4. Power the Achronix development board back on. The FPGA should be configured automatically from Flash.

Serial x4 Flash

Serial x4 Flash programming mode is essentially an enhanced and higher bandwidth implementation of the Serial x1 Flash mode. The FPGA is again the master, and interfaces with not 1 but 4 Flash memory modules to increase the data bandwidth from x1 to x4.

Figure 11 below provides a block diagram of how 4 Serial Flash memories can be connected to a Speedster22iHD FPGA in Flash x4 mode.

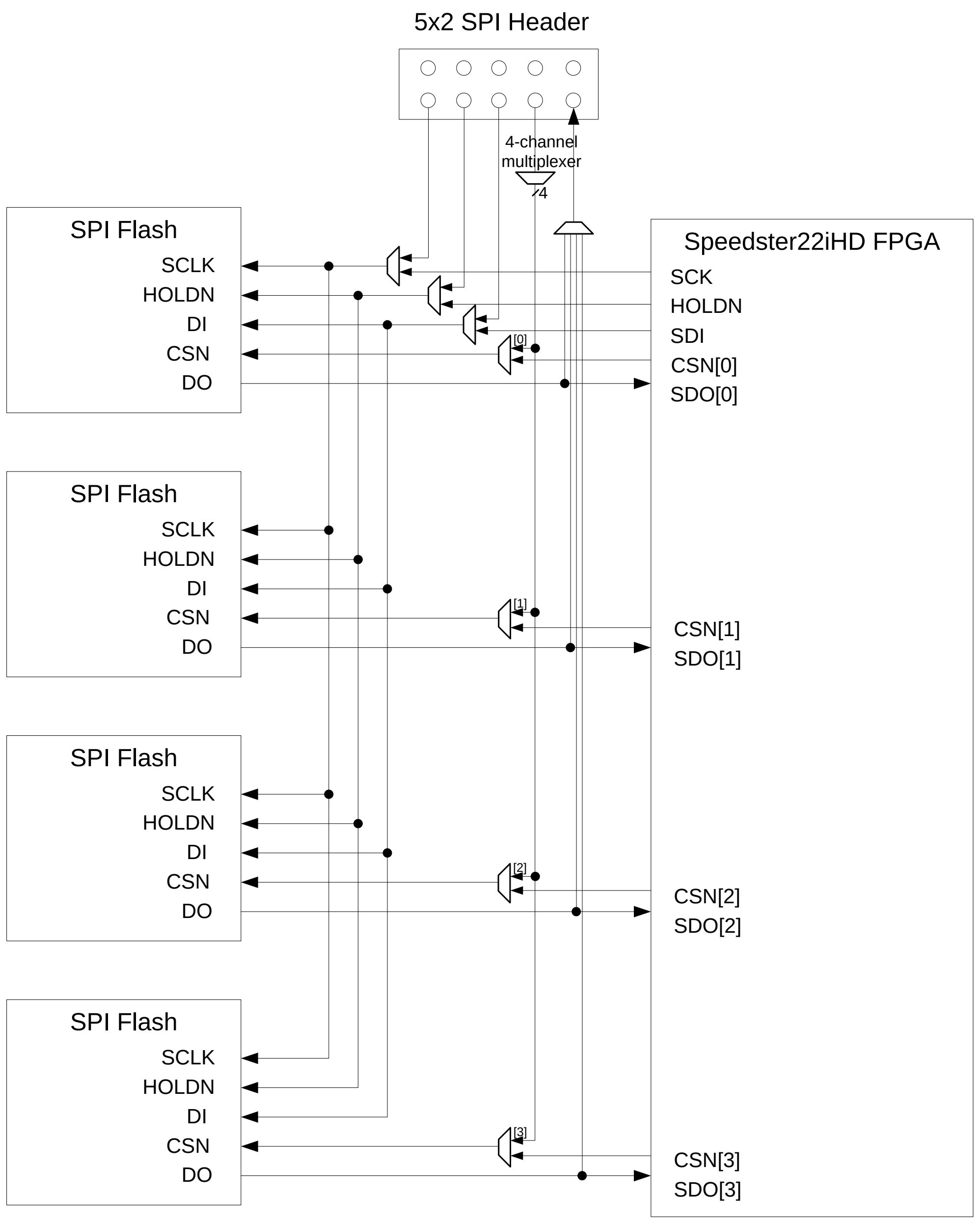


Figure 11: Flash Connectivity in x4 Mode

When writing to the 4 Flash memories, the 4-channel multiplexer would need to ensure that the CSN is asserted for a single Flash memory at any given time. Through the SPI header, data would be written to each flash device in sequence. When reading from the 4 Flash memories, the FPGA would pull all of the CSN signals low. 4 wide configuration data is read from the Flash memories and transferred to the FPGA through the SD ports. Once bitstream operations are completed (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 x1 modes.

JTAG

JTAG configuration and operation mode is independent of CONFIG_MODESEL settings, although the recommendation is to ensure that the CONFIG_MODESEL values are one of '100', '001', '010' or '000' to avoid unknown or illegal states.

The JTAG Tap controller design is compliant to the IEEE Std 1149.1. The TMS and TCK inputs determine whether an instruction register scan or data register scan is performed. TMS and TDI are sampled on the rising edge of TCK, while TDO changes on the falling edge.

Achronix recommends using an on-board JTAG header for Bitporter compatibility, direct programming through the STAPL jam file (see Bitstream File Generation Through ACE below) and the debug capability provided for Snapshot and SerDes debug in the PMA GUI.

JTAG configuration can be done for a Speedster22iHD device that in and of itself is the only device in the JTAG scan chain, or is part of a series of devices all connected up in the chain. Figure 12 below shows a block diagram of a single Speedster22iHD device in the JTAG scan chain. Figure 13 shows the case where multiple devices are connected in series.

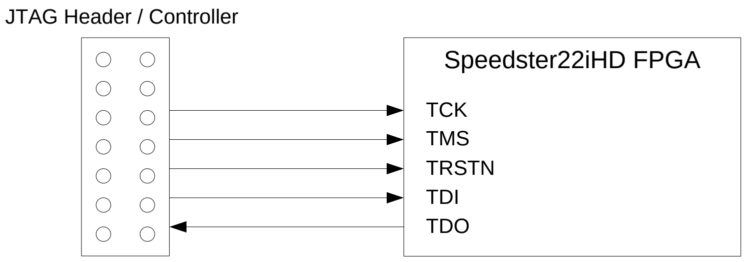


Figure 12: Single Speedster22iHD Device Connectivity to JTAG Header

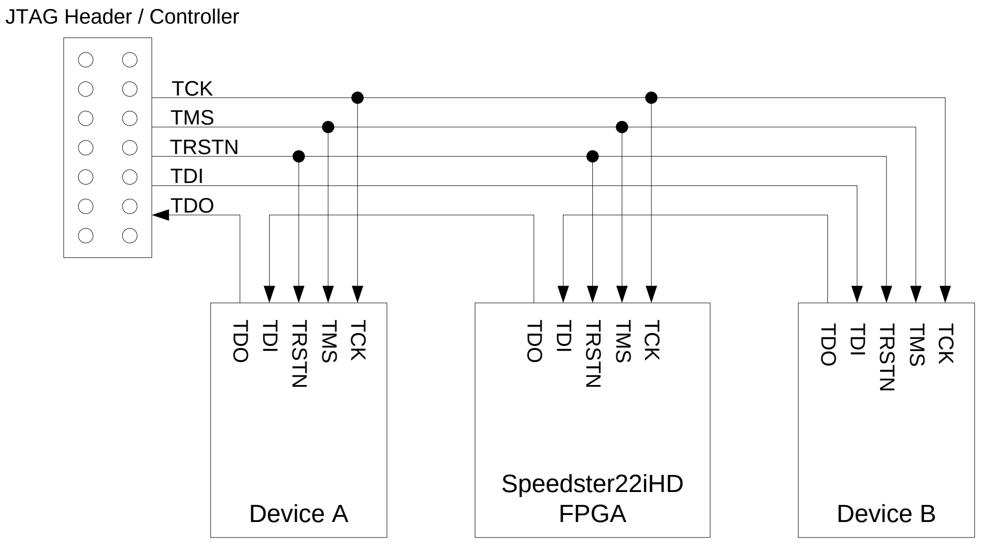


Figure 13: Multiple Device Connectivity to JTAG Header

Configuration Pins and Clock Selection

Table 5 below lists the names and functions of all of the configuration and JTAG pins used in the four different configuration modes.

Table 5: Configuration/JTAG Pins and Functions

External Pin Name	CPU	Serial Flash x1	Serial Flash x4	JTAG
SDI	DQ[0]	Serial data output to flash memory		-
SD[3]	DQ[1]		Input of configuration data from flash	-
SD[2]	DQ[2]		Input of configuration data from flash	-
SD[1]	DQ[3]		Input of configuration data from flash	-
SD[0]	DQ[4]	Input of configuration data from flash		-
SCK	-	Flash clock output		-
HOLDN	DQ[5]	Hold output to flash		-
CSN[3]	DQ[6]		Active-low chip select	-
CSN[2]	DQ[7]		Active-low chip select	-
CSN[1]	-		Active-low chip select	-
CSN[0]	Active-low chip select			-
CPU_CLK	CPU clock	-	-	-
CONFIG_RSTN	Active-low configuration reset			
CONFIG_DONE	Open-drain configuration done output			
CONFIG_STATUS	Open-drain SRAM initialization complete output			
CONFIG_MODESEL [2:0]	Config mode select. Set to '100'.	Config mode select. Set to '001'.	Config mode select. Set to '010'.	Config mode select. Not used in JTAG mode, but these pins should be set to '100', '001', '010' or '000'.
CONFIG_SYSCLK_ BYPASS	Bypass config system clock. Tie to '0' or '1'.	Bypass config system clock. Set to '0'.		Bypass config system clock. Tie to '0' or '1'.
CONFIG_CLKSEL	Selects configuration clock. Set to '0'.			Tie to '0' or '1'
TDI	-	-	-	Input of config data from JTAG controller
TDO	-	-	-	Serial data output to JTAG controller
TMS	-	-	-	Mode select from JTAG controller
TRSTN	-	-	-	Active-low reset from JTAG controller
TCK	-	-	-	Clock from JTAG controller

Table 6 highlights the different clock sources that can be selected in the various configuration modes, and Figure 14 illustrates the same FPGA configuration clock selection logic.

Table 6: Clock Sources for Configuration Modes and Settings

CONFIG_SYS_CLK _BYPASS	CONFIG_CLKSEL	CONFIG_MODESEL [2:0]	FCU CLK
0	0	001, 010	On-chip oscillator
1	0	001, 010	CPU_CLK
0/1	0	100	CPU_CLK
0	1	000, 001, 010, 100	TCK

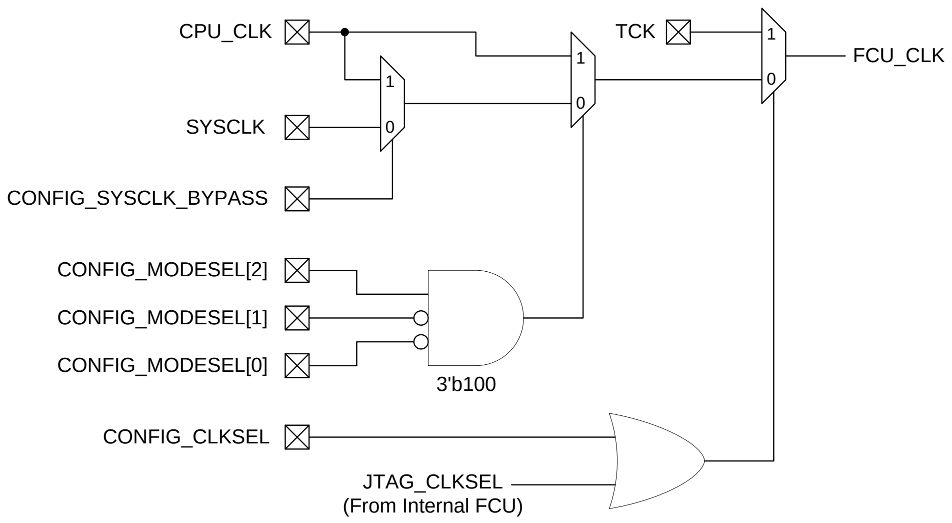


Figure 14: FPGA Configuration Clock Selection Logic

Note that if programming will be done exclusively using JTAG mode, it is important to understand how to control the CONFIG_MODESEL and clock selection pins. In order to clear the FPGA configuration memory after a power-on or a reset of the device, an active (non-JTAG) clock needs to be selected to cycle through the FCU states. For example, if CONFIG_MODESEL is set to '100' (thereby selecting CPU_CLK prior to the JTAG override), CPU_CLK needs to be toggled to ensure correct operation.

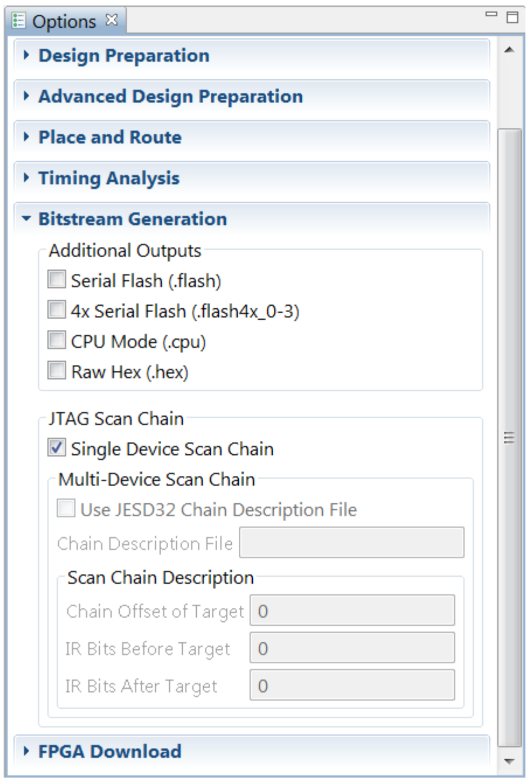
Please ensure that the clock source for the FPGA FCU_CLK does not change during configuration unless it is done in a glitchless manner. It is also not advised to toggle the FCU_CLK frequency during configuration.

The on-chip internal oscillator on the HD1000 is a crystal the provides approximately a 10MHz output clock. However, this clock frequency may vary significantly over process and temperature, so it is advised not to try to calculate exact configuration times in this mode of operation. Note that the Serial Flash Clock Divider setting needs to be set to 2 or 4 to ensure that the maximum 6MHz configuration clock frequency requirement is not violated when the internal oscillator is used for programming.

Bitstream File Generation Through ACE

ACE has a straightforward interface to generate the bitstream files required to implement all of the supported configuration modes. The bitstream files will get generated in the 'FPGA Programming – Generate Bitstream' step of the compilation flow.

The STAPL jam file needed for JTAG mode configuration will by default, always be generated. The 'Bitstream Generation' section of the Project Options menu, shown in Figure 15 below, provides users with a menu selection to generate bitstream files for the other configuration modes as well.


Options

- Design Preparation
- Advanced Design Preparation
- Place and Route
- Timing Analysis
- Bitstream Generation

Additional Outputs
- [ ] Serial Flash (.flash)
- [ ] 4x Serial Flash (.flash4x_0-3)
- [ ] CPU Mode (.cpu)
- [ ] Raw Hex (.hex)

JTAG Scan Chain
- [✓] Single Device Scan Chain

Multi-Device Scan Chain
- [ ] Use JESD32 Chain Description File

Chain Description File

Scan Chain Description

Chain Offset of Target	0
IR Bits Before Target	0
IR Bits After Target	0

- FPGA Download

Figure 15: ACE Screen Capture of Bitstream Generation Options

These bitstream file types are described in a little more detail below:

1. JTAG: STAPL .jam file for JTAG mode programming. There are options for generating a single and a multi-device scan chain configuration file in which the scan chain details need to be specified.

2. Serial Flash: A single serial flash (.flash) binary file. This is literally a full binary file of the bitstream data that could be burned into a single flash memory. There are NO newline characters in the file. It is completely binary.

3. 4x Flash: A 4x Flash (.flash4x_0-3) binary file supporting configuration from 4 flash memory devices. This the same full flash memory binary as above, but split into 4 files intended for a x4 flash memory configuration. Each binary file would be programmed and read from a single flash device in SPI mode. There are NO newline characters in the files and is again completely binary.

4. CPU Mode: File formatted for CPU mode programming (.cpu). This contains the entire bitstream organized as 9 bits per line. The MSB is the read/write bit (always 1 for write), and the other 8 bits are 8 bits of bitstream data per line.

5. In addition a raw hex (.hex) file generation option is provided. This contains bitstream data in hexadecimal format with 32-bits of data per line (and no read/write bit like the .cpu file).

Design Security

Speedster22iHD devices provide design security features using a 256-bit Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining (CBC) mode. The FPGA contains a non-volatile memory (known as a high-security or HS eFuse) for the storage of the required AES key.

Design security on Speedster22iHD devices is provided by putting the device in secure mode. This puts the following two mechanisms into effect:

- FPGA configuration bitstream encryption: the FPGA only accepts encrypted bitstreams. During configuration the FCU decrypts the encrypted bitstream using a decryption key based off of the same encryption key.
- Readback disable: Configuration bitstream readback is disabled, meaning that the design information cannot be read out and copied. HS eFuse readback capability is also blocked.

Enabling design security features requires two functions:

a. Generation of encrypted bitstreams after enabling AES encryption and specifying the encryption key that will be programmed into the FPGA in ACE. This is simply done in the ACE Bitstream Options GUI interface by checking the appropriate box and typing in the actual key to be written.

b. One-time blowing of HS eFuses to program in the key needed for AES.

The blowing of eFuses has to be very carefully integrated into the design security implementation process, since it is irreversible. Recovery from unintentionally blown fuses is not feasible, and should be diligently validated for correct operation before enabling it in a production flow. Also please note that as specified in the Pin Connections and Power Supply Sequencing User Guide, one of the fuse power rails, VCCFHV_EFUSE[3:1] needs to be powered by its own separate regulator to ensure that this rail can be increased to the voltage level needed for fuse blowing without affecting the rest of the FPGA operation. Therefore, the FPGA board and setup needs to provide for this ability.

The fuse blowing process consists of 3 phases:

1. Run phase 1 programming steps to cycle through the FCU states, write required values to the eFuse registers and bring the device to a state where eFuses are ready to be blown
2. Raise VCCFHV_EFUSE[3:1] to 2.2V and VCCRAM_EFUSE[3:1]/VDDA_NOM_E/W to 1.1V. Run phase 2 steps needed to blow the eFuses.

3. Lower VCCFHV_EFUSE[3:1], VCCRAM_EFUSE[3:1] and VDDA_NOM_E/W all back down to 1.0V. Run phase 3 steps to validate the eFuse blowing process and return the FCU back to a state to resume programming operations.

Once the eFuses are blown, the Speedster22iHD FPGA will be ready to accept encrypted bitstreams as part of regular programming operation.

Revision History

The following table shows the revision history for this document.

Date	Version	Revisions
12/18/2013	1.0	Initial Achronix release.
07/13/2015	1.1	Updated config clock frequency and details. Provided more information on flash mode support.
02/21/2016	1.2	Clarified limit of 6MHz on configuration clock. Provided details on re-configuration. Added split cable requirement for flash.
03/07/2016	1.3	Put in SerDes ref clock requirement. Updated BYPASS_CLR_MEM and timing instructions.