Configuring Cyclone FPGAs

March 2003, ver. 1.1 Application Note 250

Introduction

You can configure Cyclone™ FPGAs using one of several configuration schemes, including the new active serial (AS) configuration scheme. This new scheme is used with the new, low cost serial configuration devices. Passive serial (PS) and Joint Test Action Group (JTAG)-based configuration schemes are also supported by Cyclone FPGAs. Additionally, Cyclone FPGAs can receive a compressed configuration bit stream and decompress this data in real-time, reducing storage requirements and configuration time.

This application note provides details on each of the three supported Cyclone configuration schemes.

Device Configuration Overview

Cyclone FPGAs use SRAM cells to store configuration data. Since SRAM memory is volatile, configuration data must be downloaded to Cyclone FPGAs each time the device powers up. You can download configuration data to Cyclone FPGAs using the AS, PS, or JTAG interfaces. See Table 1.

Table 1. Cyclone FPGA Configuration Schemes

Configuration Scheme	Description
Active serial (AS) configuration	Configuration using: ■ Serial configuration devices (EPCS1 or EPCS4)
Passive serial (PS) configuration	Configuration using: ■ Enhanced configuration devices (EPC4, EPC8, and EPC16) ■ EPC2, EPC1 configuration devices ■ Intelligent host (microprocessor) ■ Download cable
JTAG-based configuration	Configuration via JTAG pins using: ■ Download cable ■ Intelligent host (microprocessor) ■ Jam™ Standard Test and Programming Language (STAPL)

You can select a Cyclone FPGA configuration scheme by driving its MSEL1 and MSEL0 pins either high (1) or low (0), as shown in Table 2.

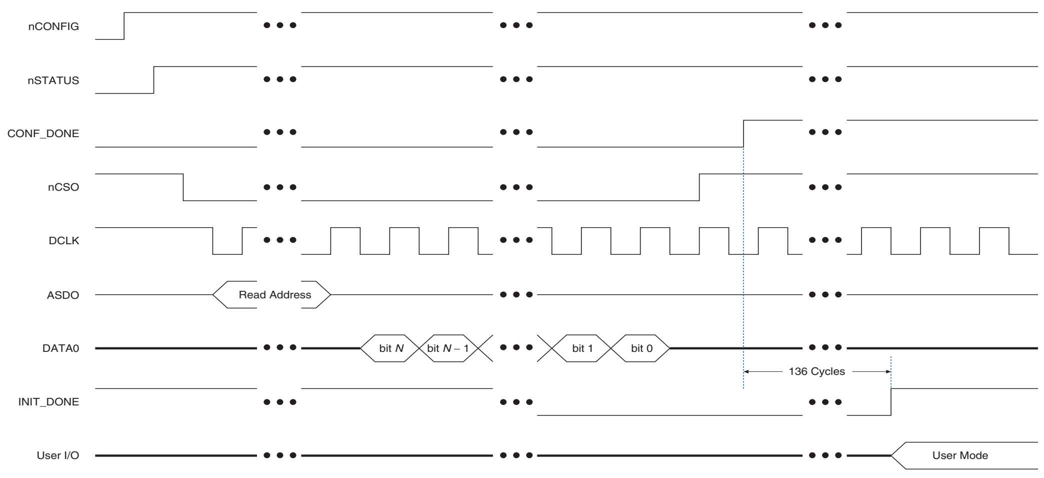
Table 2. Selecting Cyclone Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	AS
0	1	PS
0	*(1)*	JTAG-based *(2)*, *(3)*

Notes to Table 2:
(1) You can set MSEL0 either high or low.
(2) Do not leave MSEL pins floating. Connect them to a low- or high-logic level.
(3) JTAG-based configuration takes precedence over other schemes (i.e., ignores MSEL pin settings).

After configuration, Cyclone FPGAs will initialize registers and I/O pins, then enter user mode and function as per the user design. Figure 1 shows an AS configuration waveform.

Figure 1. AS Configuration Waveform



You can configure Cyclone FPGAs using the 3.3-, 2.5-, 1.8-, or 1.5-V LVTTL I/O standard on configuration and JTAG input pins. These devices do not feature a VCCSEL pin; therefore, you should connect the VCCIO pins of the I/O banks containing configuration or JTAG pins according to the I/O standard specifications.

Table 3 summarizes the approximate uncompressed configuration file size for each Cyclone FPGA. To calculate the amount of storage space required for multi-device configurations, add the file size of each device together.

Table 3. Cyclone Configuration File Sizes Note (1)

Device	SRAM Object File Size (Mbits)
EP1C3	0.628
EP1C4	0.925
EP1C6	1.167
EP1C12	2.324
EP1C20	3.559

Note to Table 3:
(1) These values are preliminary.

You should only use the numbers in Table 3 to estimate the SRAM Object File (**.sof**) size before design compilation. The exact file size can vary because different Altera® Quartus® II software versions can add a slightly different number of padding bits during programming. However, for any specific version of the Quartus II software, any design targeted for the same device has the same uncompressed configuration file size. If compression is used, the file size can vary after each compilation.

Data Compression

Cyclone FPGAs are the first FPGAs to support decompression of configuration data. This feature allows you to store compressed configuration data in configuration devices or other memory, and transmit this compressed bit stream to Cyclone FPGAs. During configuration, the Cyclone FPGA decompresses the bit stream in real time and programs its SRAM cells.

Cyclone FPGAs support compression in the AS and PS configuration schemes. Compression is not supported for JTAG-based configuration.

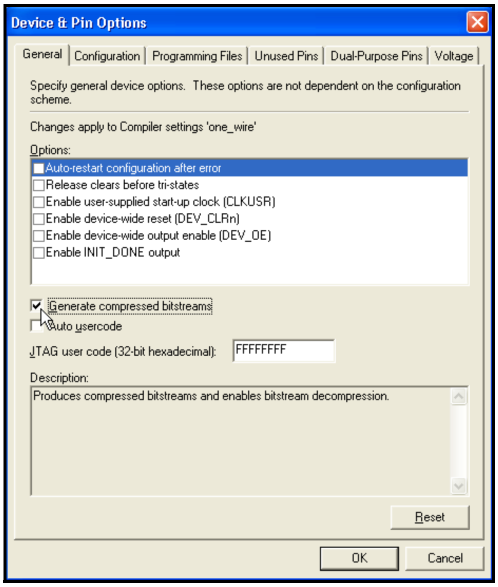
☞ Preliminary data indicates that compression reduces configuration bit stream size by 35 to 60%.

When you enable compression, the Quartus II software generates configuration files with compressed configuration data. This compression reduces the storage requirements in the configuration device or flash, and decreases the time needed to transmit the bit stream to the Cyclone FPGA.

There are two methods to enable compression for Cyclone bitstreams: before design compilation (in Compiler Settings menu) & after design compilation (in Convert Programming Files window).

To enable compression in the project's compiler settings, select "Device" under the "Assignments" menu to bring up the settings window. After selecting your Cyclone device open the "Device & Pin Options" window, and in the "General" settings tab enable the check box for "Generate compressed bitstreams" (as shown in Figure 2).

Figure 2. Enabling compression for Cyclone bitstreams in Compiler Settings


Device & Pin Options
General | Configuration | Programming Files | Unused Pins | Dual-Purpose Pins | Voltage
Reset
OK
Cancel

Specify general device options. These options are not dependent on the configuration scheme.

Changes apply to Compiler settings 'one_wire'

Options:

- ☐ Auto-restart configuration after error
- ☐ Release clears before tri-states
- ☐ Enable user-supplied start-up clock (CLKUSR)
- ☐ Enable device-wide reset (DEV_CLRn)
- ☐ Enable device-wide output enable (DEV_OE)
- ☐ Enable INIT_DONE output

☑ Generate compressed bitstreams

☐ Auto usercode

JTAG user code (32-bit hexadecimal): FFFFFFFF

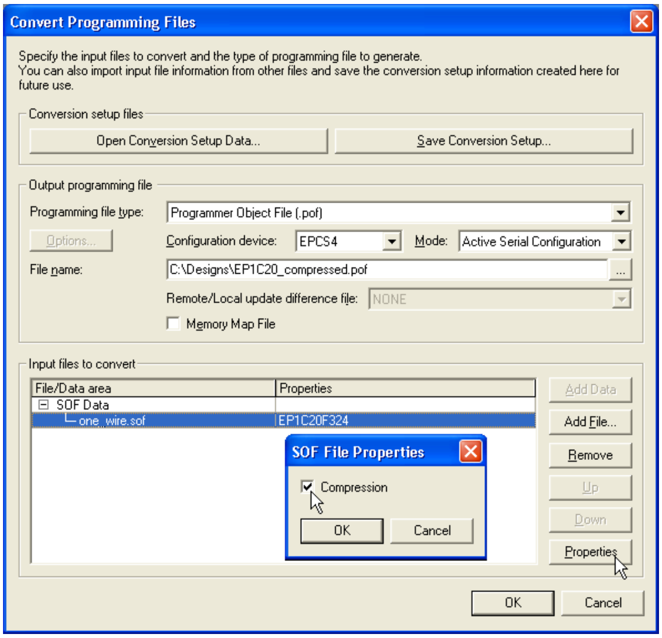
Description:

Produces compressed bitstreams and enables bitstream decompression.

Compression can also be enabled when creating programming files from the "Convert Programming Files". See Figure 3.

1. Open "File -> Convert Programming Files"
2. Select the Programming File type (POF, SRAM HEXOUT, RBF, or TTF)
3. For POF output files, select a configuration device
4. Select "Add File" and add Cyclone SOF file
5. Select the name of file you added to the "SOF Data" area and click on "Properties"
6. Enable the "Compression" checkbox

Figure 3. Enabling compression for Cyclone bitstreams in Convert Programming Files


Convert Programming Files

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: Programmer Object File (.pof)

Configuration device: EPCS4 Mode: Active Serial Configuration

File name: C:\Designs\EP1C20_compressed.pof

Remote/Local update difference file: NONE

☐ Memory Map File

Input files to convert

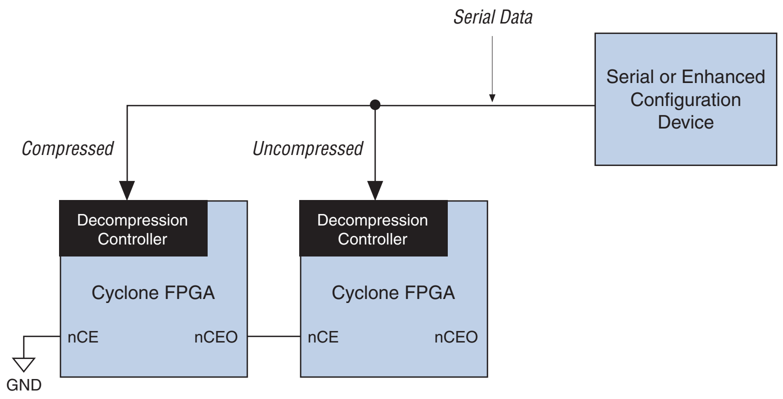
File/Data area	Properties
⊟ SOF Data	
└ one_wire.sof	EP1C20F324

SOF File Properties

☑ Compression

When multiple Cyclone devices are cascaded, the compression feature can be selectively enabled for each device in the chain. Figure 4 depicts a chain of two Cyclone FPGAs. The first Cyclone FPGA has compression enabled and therefore receives a compressed bit stream from the configuration device. The second Cyclone FPGA has the compression feature disabled and receives uncompressed data.

Figure 4. Compressed & Uncompressed Configuration Data in the Same Programming File *Note (1)*



Note to Figure 4:

(1) The first device in the chain should be set up in AS configuration mode (`MSEL[1..0]="00"`). The remaining devices in the chain must be set up in PS configuration mode (`MSEL[1..0]="01"`).

You can generate programming files for this setup from the Convert Programming Files window (File menu) in the Quartus II software.

The decompression feature supported by Cyclone FPGAs is separate from the decompression feature in enhanced configuration devices (EPC16, EPC8, and EPC4). The data compression feature in the enhanced configuration devices allows them to store compressed data and decompress the bit stream before transmitting to the target devices. When using Cyclone FPGAs with enhanced configuration devices, Altera recommends using compression on one of the devices, not both (preferably the Cyclone FPGA since transmitting compressed data reduces configuration time).

Configuration Schemes

This section describes the various configuration schemes you can use to configure Cyclone FPGAs. Descriptions include an overview of the protocol, pin connections, and timing information. The schemes discussed are:

- AS configuration (serial configuration devices)
- PS configuration
- JTAG-based configuration

Active Serial Configuration (Serial Configuration Devices)

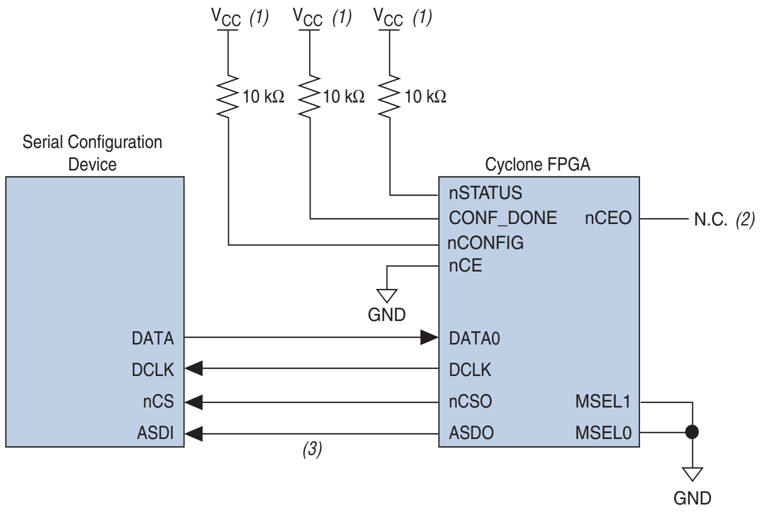
In the AS configuration scheme, Cyclone FPGAs are configured using the new serial configuration devices. These configuration devices are low cost devices with non-volatile memory that feature a simple four-pin interface and a small form factor. These features make serial configuration devices an ideal solution for configuring the low-cost Cyclone FPGAs.

For more information on serial configuration devices, see the *Serial Configuration Devices (EPCS1 & EPCS4) Data Sheet*.

Serial configuration devices provide a serial interface to access configuration data. During device configuration, Cyclone FPGAs read configuration data via the serial interface, decompress data if necessary, and program their SRAM cells. This scheme is referred to as an AS configuration scheme because the FPGA controls the configuration interface. This scheme is in contrast to the PS configuration scheme where the configuration device controls the interface.

Serial configuration devices have a four-pin interface: serial clock input (`DCLK`), serial data output (`DATA`), AS data input (`ASDI`), and an active-low chip select (`nCS`). This four-pin interface connects to Cyclone FPGA pins, as shown in Figure 5.

Figure 5. AS Configuration of a Single Cyclone FPGA



Notes to Figure 5:

(1) Power up the ByteBlaster II Vcc with a 3.3V supply.
(2) The nCEO pin is left unconnected.
(3) Cyclone FPGAs use the ASDO to ASDI path to control the configuration device.

Connecting the MSEL[1..0] pins to 00 selects the AS configuration scheme. The Cyclone chip enable signal, nCE, must also be connected to ground for successful configuration.

During system power up, both the Cyclone FPGA and serial configuration device enter a power-on reset (POR) period. As soon as the Cyclone FPGA enters POR, it drives nSTATUS low to indicate it is busy and drives CONF_DONE low to indicate that it has not been configured. After POR, which typically lasts 100 ms, the Cyclone FPGA releases nSTATUS and enters configuration mode when this signal is pulled high by the external 10-kΩ resistor.

The serial clock (DCLK) generated by the Cyclone FPGA controls the entire configuration cycle (see Figure 1 on page 2) and this clock line provides the timing for the serial interface. Cyclone FPGAs use an internal oscillator to generate DCLK. Typical DCLK frequency during AS configuration is 15 MHz.

The serial configuration device latches input/control signals on the rising edge of DCLK and drives out configuration data on the falling edge. Cyclone FPGAs drive out control signals on the falling edge of DCLK and latch configuration data on the rising edge of DCLK.

In configuration mode, the Cyclone FPGA enables the serial configuration device by driving the nCSO output pin low that is connected to the chip select (nCS) pin of the configuration device. The Cyclone FPGA's serial clock (DCLK) and serial data output (ASDO) pins are used to read configuration data. The configuration device provides data on its serial data output (DATA) pin that is connected to the DATA0 input on Cyclone FPGAs.

After all configuration bits are received by the Cyclone FPGA, it releases the open-drain CONF_DONE pin allowing the external 10-kΩ resistor to pull this signal to a high level. Initialization begins only after the CONF_DONE line reaches a high level. Initialization completes within 136 clock cycles and the device enters user mode.

You can select the clock used for initialization by using the **User Supplied Start-Up Clock** option in Quartus II software. The Quartus II software uses the 10-MHz internal oscillator by default to initialize the Cyclone FPGA. When you enable the **User Supplied Start-Up Clock** option, the software uses the CLKUSR pin as the initialization clock.

If an error occurs during configuration, the Cyclone FPGA asserts the nSTATUS signal low indicating a data frame error, and the CONF_DONE signal will stay low. With the **Auto-Restart Configuration on Frame Error** option enabled in the Quartus II software, the Cyclone FPGA resets the configuration device by pulsing nCSO, releases nSTATUS after a reset time-out period (about 30 micro-seconds), and retries configuration. After successful configuration, the CONF_DONE signal is tri-stated by the target device and then pulled high by the pull-up resistor.

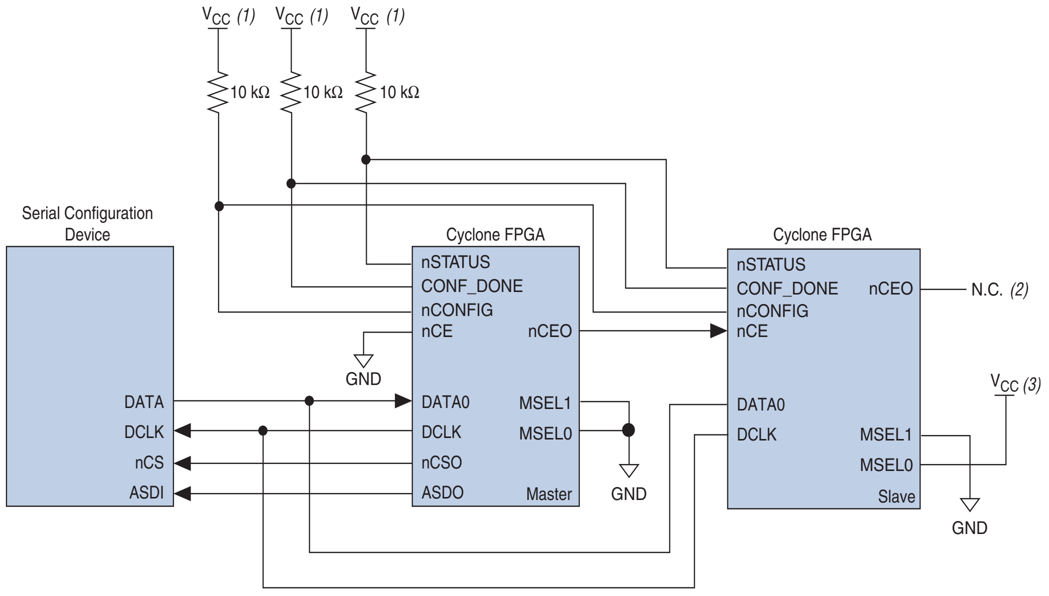
All AS configuration pins, DATA0, DCLK, nCSO, and ASDO, have weak internal pull-up resistors. These pull-up resistors are always active.

Configuring Multiple Devices (Cascading)

You can configure multiple Cyclone FPGAs using a single serial configuration device. You can cascade multiple Cyclone FPGAs using the chip-enable (nCE) and chip-enable-out (nCEO) pins. The first device in the chain must have its nCE pin connected to ground. You must connect its nCEO chip-enable-out pin to the chip-enable (nCE) pin of the next device in the chain. When the first device captures all of its configuration data from the bit stream, it drives the nCEO pin low enabling the next device in the chain. You must leave the nCEO pin of the last device unconnected.

This first Cyclone FPGA in the chain is the configuration master and controls configuration of the entire chain. You must connect its MSEL pins to select the AS configuration scheme. The remaining Cyclone FPGAs are configuration slaves and you must connect their MSEL pins to select the PS configuration scheme. Figure 6 shows the pin connections for this setup.

Figure 6. Configuring Multiple Devices Using a Serial Configuration Device (AS)



Notes to Figure 6:

(1) Power up the ByteBlaster II Vcc with a 3.3V supply.
(2) The nCEO pin is left unconnected.
(3) Connect MSEL0 to the Vcc supply voltage of the I/O Bank it resides in.

As shown in Figure 6, the nSTATUS and CONF_DONE pins on all target FPGAs are connected together with external pull-up resistors. These pins are open-drain bidirectional pins on the FPGAs. When the first device asserts nCEO (after receiving all of its configuration data), it releases its CONF_DONE pin. But the subsequent devices in the chain keep this shared CONF_DONE line low until they have received their configuration data. When all target FPGAs in the chain have received their configuration data and have released CONF_DONE, the pull-up resistor drives a high level on this line and all devices simultaneously enter initialization mode. If an error occurs at any point during configuration, the nSTATUS line is driven by the failing FPGA. If you enable the **Auto Restart Configuration on Frame Error** option, reconfiguration of the entire chain begins.

Furthermore, while you can cascade Cyclone FPGAs, serial configuration devices cannot be cascaded or chained together. If the configuration bit stream size exceeds the capacity of a serial configuration device, you must select a larger configuration device and / or enable the compression feature. While configuring multiple devices, the size of the bit stream is the sum of the individual devices' configuration bit streams.

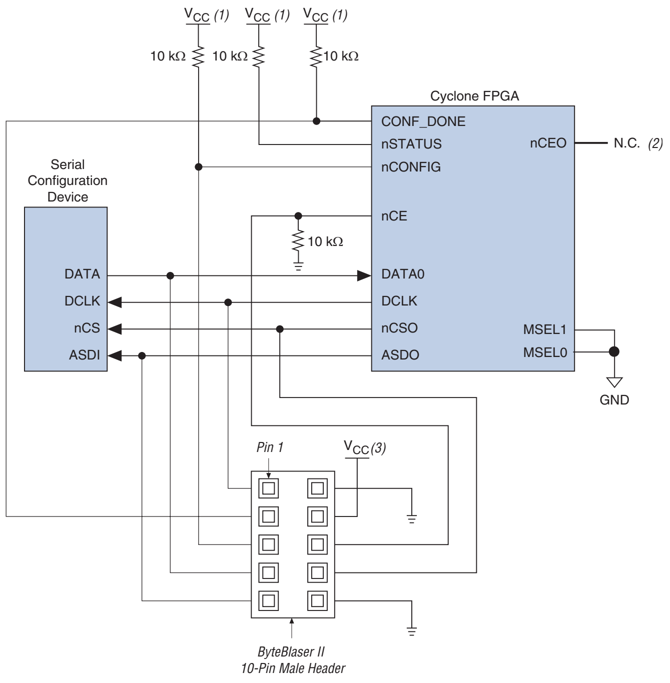
Programming Serial Configuration Devices

Serial configuration devices are non-volatile, flash-memory-based devices. You can program these devices in-system using the ByteBlaster™ II download cable. Alternatively, you can program them using the Altera Programming Unit (APU) or supported third-party programmers.

You can perform in-system programming of serial configuration devices via the AS programming interface. During in-system programming, the download cable disables FPGA access to the AS interface by driving the chip-enable `nCE` pin high. Cyclone FPGAs are also held in reset by a low level on `nCONFIG`. After programming is complete, the download cable releases `nCE` and `nCONFIG`, allowing the pull-down and pull-up resistor to drive `GND` and `VCC`, respectively. Figure 7 shows the download cable connections to the serial configuration device.

For more information on the ByteBlaster II cable, see the *ByteBlaster II Download Cable Data Sheet*.

Figure 7. In-System Programming of Serial Configuration Devices



Notes to Figure 7:

(1) Connect these pull-up resistors to 3.3V supply.
(2) The nCEO pin is left unconnected.
(3) Power up the ByteBlaster II Vcc with a 3.3V supply.

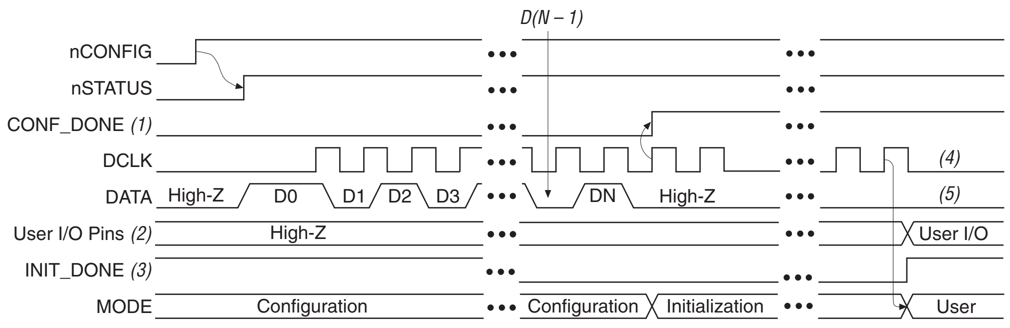
You can program serial configuration devices by using the Quartus II software with the APU and the appropriate configuration device programming adapter. All serial configuration devices are offered in an eight-pin small outline integrated circuit (SOIC) package and can be programmed using the PLMSEPC-8 adapter.

In production environments, Serial Configuration devices can be programmed using multiple methods. Altera programming hardware (APU) or other third-party programming hardware can be used to program blank Serial Configuration devices before they are mounted onto PCBs. Alternatively, you can use an on-board microprocessor to program the Serial Configuration device in-system using C-based software drivers provided by Altera. For more information on these and other new methods, please refer to the Cyclone Literature web page.

Passive Serial Configuration

Cyclone FPGAs also feature the PS configuration scheme supported by all Altera FPGAs. In the PS scheme, an external host (configuration device, embedded processor, or host PC) controls configuration. Configuration data is clocked into the target Cyclone FPGAs via the `DATA0` pin at each rising edge of `DCLK`. The configuration waveforms for this scheme are shown in Figure 8.

Figure 8. PS Configuration Cycle Waveform



Notes to Figure 8:

(1) During initial power up and configuration, `CONF_DONE` is low. After configuration, `CONF_DONE` goes high to indicate successful configuration. If the device is reconfigured, `CONF_DONE` goes low after `nCONFIG` is driven low.

(2) User I/O pins are tri-stated during configuration. Cyclone FPGAs also have a weak pull-up resistor on I/O pins during configuration. After initialization, the user I/O pins perform the function assigned in the user's design.

(3) When used, the optional `INIT_DONE` signal is high when `nCONFIG` is low before configuration and during the first 136 clock cycles of configuration.

(4) In user mode, `DCLK` should be driven high or low when using the PS configuration scheme. When using the AS configuration scheme, `DCLK` is a Cyclone output pin and should not be driven externally.

(5) In user mode, `DATA0` should be driven high or low.

PS Configuration using Configuration Device

In the PS configuration device scheme, nCONFIG is usually tied to V_{CC} (when using EPC16, EPC8, EPC4, or EPC2 devices, you can connect nCONFIG to nINIT_CONF). Upon device power-up, the target Cyclone FPGA senses the low-to-high transition on nCONFIG and initiates configuration. The target device then drives the open-drain CONF_DONE pin low, which in-turn drives the configuration device's nCS pin low. When exiting POR, both the target and configuration device release the open-drain nSTATUS pin (typically Cyclone POR lasts 100 ms).

Before configuration begins, the configuration device goes through a POR delay of up to 100 ms (maximum) to allow the power supply to stabilize. You must power the Cyclone FPGA before or during the POR time of the enhanced configuration device. During POR, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When the target and configuration devices complete POR, they both release the nSTATUS to OE line, which is then pulled high by a pull-up resistor.

When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. When all devices are ready, the configuration device clocks out DATA and DCLK to the target devices using an internal oscillator.

After successful configuration, the Cyclone FPGA starts initialization using the 10-MHz internal oscillator as the reference clock. The CONF_DONE pin is released by the target device and then pulled high by a pull-up resistor. When initialization is complete, the target Cyclone FPGA enters user mode.

If an error occurs during configuration, the target device drives its nSTATUS pin low, resetting itself internally and resetting the configuration device. If you turn on the **Auto-Restart Configuration on Frame Error** option, the device reconfigures automatically if an error occurs. To set this option, select **Compiler Settings** (Processing menu), and click on the **Chips & Devices** tab. Select **Device & Pin Options**, and click on the **Configuration** tab.

If the **Auto-Restart Configuration on Frame Error** option is turned off, the external system (configuration device or microprocessor) must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}. When configuration is complete, the target device releases CONF_DONE, which disables the configuration device by driving nCS high. The configuration device drives DCLK low before and after configuration.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the target device has not configured successfully. (For CONF_DONE to reach a high state, enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit. EPC2 devices wait for 16 DCLK cycles.) In this case, the configuration device pulses its OE pin low for a few microseconds, driving the target device's nSTATUS pin low. If the **Auto-Restart Configuration on Frame Error** option is set in the Quartus II software, the target device resets and then releases its nSTATUS pin after a reset time-out period. When nSTATUS returns high, the configuration device reconfigures the target device.

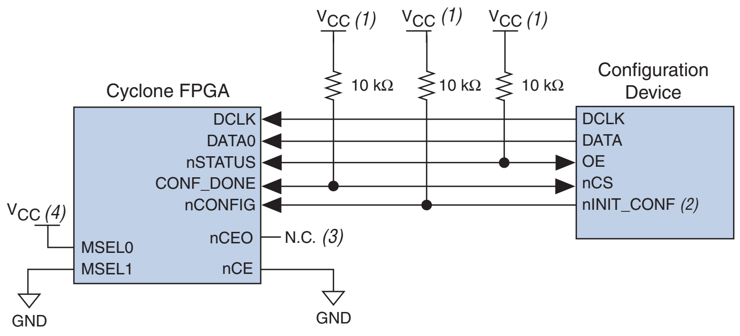
You should not pull CONF_DONE low to delay initialization. Instead, use the Quartus II software's **User-Supplied Start-Up Clock** option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain initialize together since their CONF_DONE pins are tied together. For more information on this option, see "Device Options" on page 43.

CONF_DONE goes high during the first few clock cycles of initialization. Hence when using the CLKUSR feature you would not see the CONF_DONE signal high until you start clocking CLKUSR. However, the device does retain configuration data and waits for these initialization clocks to release CONF_DONE and go into user mode.

☞ When using internal pull-up resistors on configuration devices, power the supply voltage on the Cyclone FPGA I/O pins (V_{CCIO}) to 3.3 V. EPC2, EPC4, EPC8, and EPC16 devices support 3.3-V operation but not 2.5-V operation. Therefore, you must set the V_{CCIO} voltages for the banks where programming pins reside to 3.3 V.

Figure 9 shows how to configure one Cyclone FPGA with one configuration device.

Figure 9. Single Device Configuration Circuit



Notes to Figure 9:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. This pull-up resistor is 10 kΩ. The EPC16, EPC8, EPC4, and EPC2 devices' OE and nCS pins have internal, user-configurable pull-up resistors. If you use internal pull-up resistors, do not use external pull-up resistors on these pins.

(2) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices and has an internal pull-up resistor that is always active. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor.

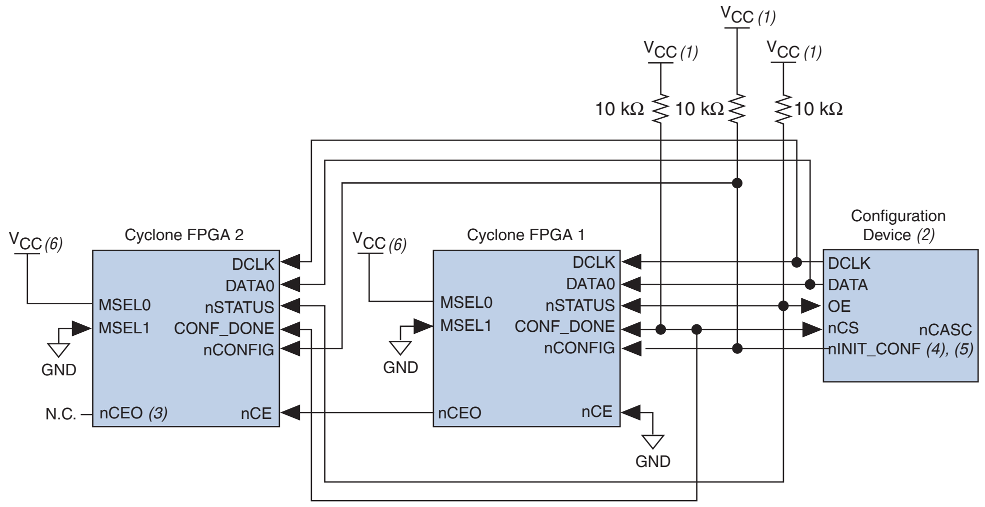
(3) The nCEO pin is left unconnected for the last device in the chain.

(4) Connect MSEL0 to the Vcc supply voltageof I/O Bank it resides in.

Configuring Multiple Cyclone FPGAs

You can use a single configuration device to configure multiple Cyclone FPGAs. In this setup, the nCEO pin of the first device is connected to the nCE pin of the second device in the chain. If there are additional devices, connect the nCE pin of the next device to the nCEO pin of the previous device. You should leave the nCEO pin on the last device in the chain unconnected. To configure properly, all of the target device CONF_DONE and nSTATUS pins must be tied together. Figure 10 shows an example of configuring multiple Cyclone FPGAs using a single configuration device.

Figure 10. Configuring Multiple Cyclone FPGAs with a Single Configuration Device



Notes to Figure 10:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. The EPC16, EPC8, EPC4, and EPC2 devices' OE and nCS pins have internal, user-configurable pull-up resistors. If you use internal pull-up resistors, do not use external pull-up resistors on these pins.
(2) EPC16, EPC8, and EPC4 configuration devices cannot be cascaded.
(3) The nCEO pin is left unconnected for the last device in the chain.
(4) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor.
(5) The nINIT_CONF pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the nINIT_CONF pin.
(6) Connect MSEL0 to the Vcc supply voltageof I/O Bank it resides in.

When performing multi-device PS configuration, you must generate the configuration device programming file (**.sof**) from each project. Then you must combine multiple **.sof** files using the Quartus II software through the **Convert Programming Files** dialog box.

For more information on how to create Programmer Object Files (**.pof**) for enhanced configuration devices, see *AN 218: Using Enhanced Configuration Devices*. For a description of the various configuration and programming files, see "Device Configuration Files" on page 49.

After the first Cyclone FPGA completes configuration during multi-device configuration, its nCEO pin activates the second device's nCE pin, prompting the second device to begin configuration. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

In addition, all nSTATUS pins are tied together; therefore, if any device (including the configuration device) detects an error, configuration stops for the entire chain. Also, if the configuration device does not detect CONF_DONE going high at the end of configuration, it resets the chain by pulsing its OE pin low for a few microseconds. For CONF_DONE to reach a high state, enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit. EPC2 devices wait for 16 DCLK cycles.

If the **Auto-Restart Configuration on Frame Error** option is turned on in the Quartus II software, the Cyclone FPGA releases its nSTATUS pins after a reset time-out period (about 30 micro-seconds). When the nSTATUS pins are released and pulled high, the configuration device reconfigures the chain. If the **Auto-Restart Configuration on Frame Error** option is not turned on, the devices drive nSTATUS low until they are reset with a low pulse on nCONFIG.

You can also cascade several EPC2 configuration devices to configure multiple Cyclone FPGAs. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the subsequent EPC2 device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted. You cannot cascade EPC16, EPC8, and EPC4 configuration devices.

Programming Configuration Devices

Enhanced configuration devices (EPC4, EPC8, and EPC16 devices) and EPC2 devices support in-system programming via JTAG. You can program these configuration devices using the Quartus II software and a download cable (e.g., ByteBlaster II, MasterBlaster™, or ByteBlasterMV™ cables).

You can also program configuration devices using the Quartus II software, the APU, and the appropriate configuration device programming adapter. Table 4 shows which programming adapter to use with each configuration device.

Table 4. Programming Adapters

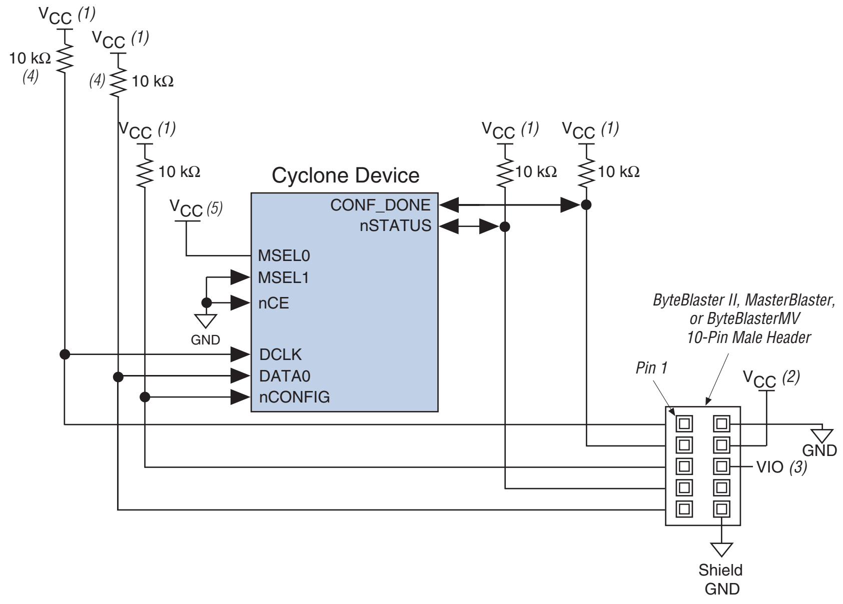
Device	Package	Adapter
EPC16	88-pin Ultra FineLine BGA® 100-pin PQFP	PLMUEPC-88 PLMQEPC-100
EPC8	100-pin PQFP	PLMQEPC-100
EPC4	100-pin PQFP	PLMQEPC-100
EPC2	20-pin J-Lead 32-pin TQFP	PLMJ1213 PLMT1213
EPC1	8-pin DIP 20-pin J-Lead	PLMJ1213 PLMJ1213

PS Configuration Using a Download Cable

Using a download cable in PS configuration, an intelligent host (e.g., your PC) transfers data from a storage device (e.g., your hard drive) to the Cyclone FPGA through a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin. The programming hardware then sends the configuration data one bit at a time on the device's `DATA0` pin. The data is clocked into the target device using `DCLK` until the `CONF_DONE` goes high.

When using programming hardware for the Cyclone FPGA, turning on the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle because the Quartus II software must restart configuration when an error occurs. Figure 11 shows the PS configuration setup for the Cyclone FPGA using a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable.

Figure 11. PS Configuration Circuit with ByteBlaster II, MasterBlaster, or ByteBlasterMV Cable



Notes to Figure 11:

(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (`VIO` pin) or ByteBlasterMV cable.

(2) Power supply voltage: V_{CC} = 3.3 V for the ByteBlaster II, MasterBlaster, and ByteBlasterMV cable.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. This pin is a no-connect pin for the ByteBlasterMV header.

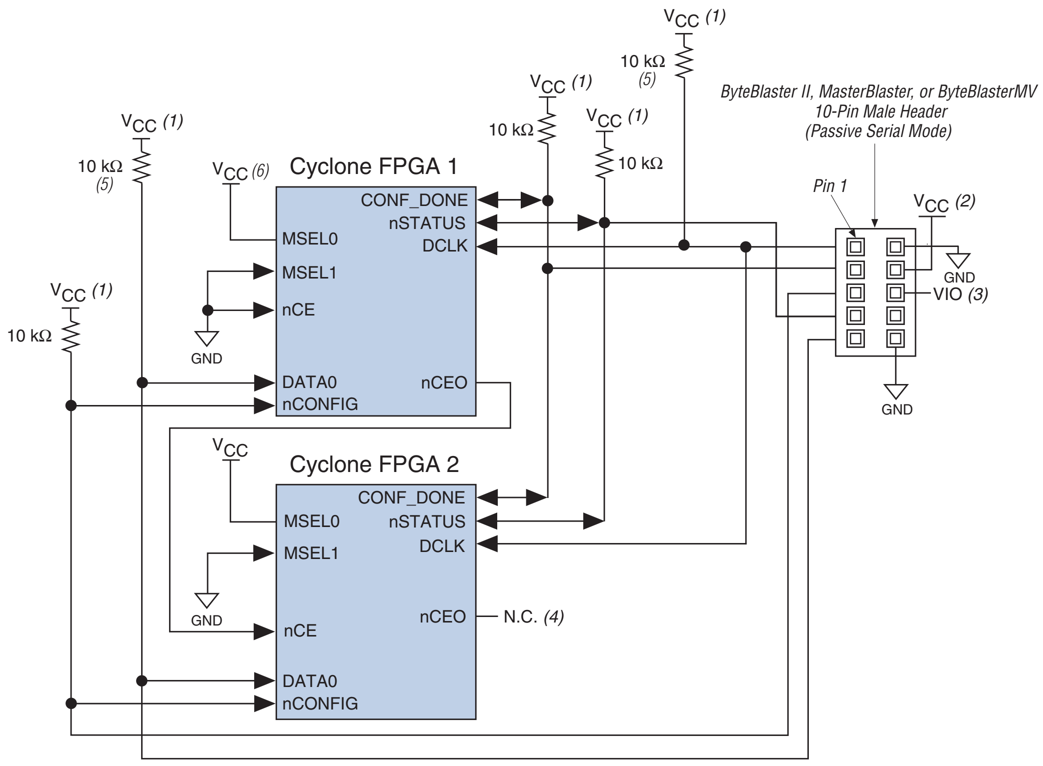
(4) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

(5) Connect `MSEL0` to the Vcc supply voltageof I/O Bank it resides in.

You can use the download cable to configure multiple Cyclone FPGAs by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. All other configuration pins are connected to each device in the chain.

Because all `CONF_DONE` pins are tied together, all devices in the chain initialize and enter user mode at the same time. In addition, because the `nSTATUS` pins are tied together, the entire chain halts configuration if any device detects an error. In this situation, the Quartus II software must restart configuration; the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle. Figure 12 shows how to configure multiple Cyclone FPGAs with a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable.

Figure 12. Multi-Device PS Configuration with a ByteBlaster II, MasterBlaster, or ByteBlasterMV Cable



Notes to Figure 12:

(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (VIO pin) or ByteBlasterMV cable.

(2) Power supply voltage: V_{CC} = 3.3 V for the ByteBlaster II, MasterBlaster, and ByteBlasterMV cable.

(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

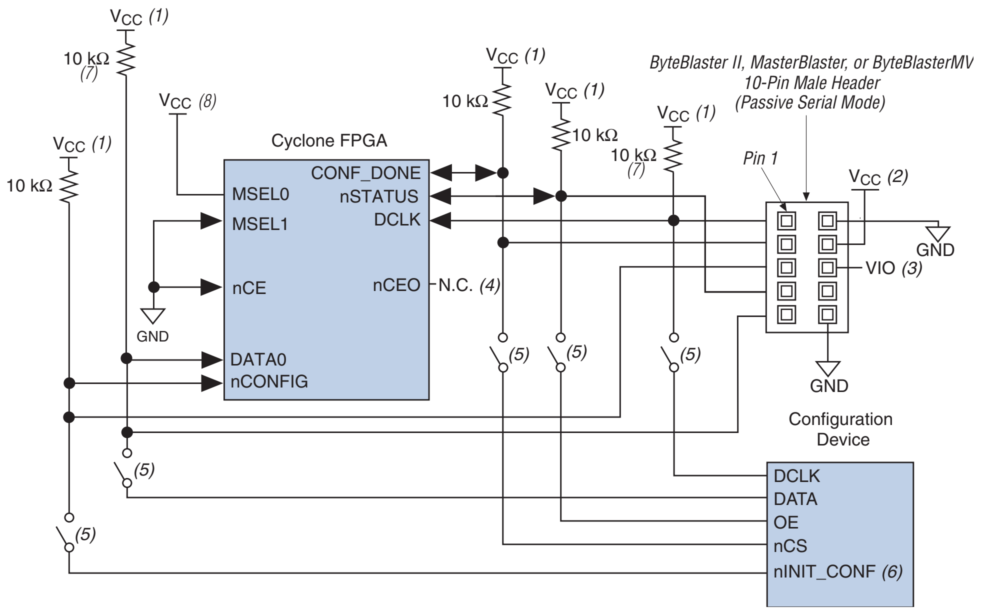
(4) The nCEO pin is left unconnected for the last device in the chain.

(5) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

(6) Connect MSEL0 to the Vcc supply voltageof I/O Bank it resides in.

If you are using a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable to configure device(s) on a board that also is populated with configuration devices, you should electrically isolate the configuration devices from the target device(s) and cable. One way to isolate the configuration devices is to add logic, such as a multiplexer, that can select between the configuration devices and the cable. The multiplexer allows bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`CONF_DONE`, `nSTATUS`, `DCLK`, `nCONFIG`, and `DATA0`) between the cable and the configuration devices. The last option is to remove the configuration devices from the board when configuring with the cable. Figure 13 shows a combination of a configuration device and a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable to configure a Cyclone FPGA.

Figure 13. Configuring with a Combined PS & Configuration Device Scheme



Notes to Figure 13:

(1) You should connect the pull-up resistor to the same supply voltage as the configuration device.
(2) Power supply voltage: V_{CC} = 3.3 V for the ByteBlaster II, MasterBlaster, and ByteBlasterMV cable.
(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the target device's V_{CCIO}. This is a no-connect pin for the ByteBlasterMV header.
(4) The `nCEO` pin is left unconnected.
(5) You should not attempt configuration with a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable while a configuration device is connected to a Cyclone FPGA. Instead, you should either remove the configuration device from its socket when using the download cable or place a switch on the five common signals between the download cable and the configuration device. Remove the ByteBlaster II, MasterBlaster, or ByteBlasterMV cable when configuring with a configuration device.
(6) If `nINIT_CONF` is not used, `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.
(7) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.
(8) Connect `MSEL0` to the Vcc supply voltageof I/O Bank it resides in.

For more information on how to use the ByteBlaster II, MasterBlaster, or ByteBlasterMV cables, see the following documents:

- *ByteBlaster II Parallel Port Download Cable Data Sheet*
- *MasterBlaster Serial/USB Communications Cable Data Sheet*
- *ByteBlasterMV Parallel Port Download Cable Data Sheet*

PS Configuration from a Microprocessor

In PS configuration with a microprocessor, a microprocessor transfers data from a storage device to the target Cyclone FPGA. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the nCONFIG pin and the target device must release nSTATUS. The microprocessor then places the configuration data one bit at a time on the DATA0 pin of the Cyclone FPGA. The least significant bit (LSB) of each data byte must be presented first. Data is clocked continuously into the target device using DCLK until the CONF_DONE signal goes high.

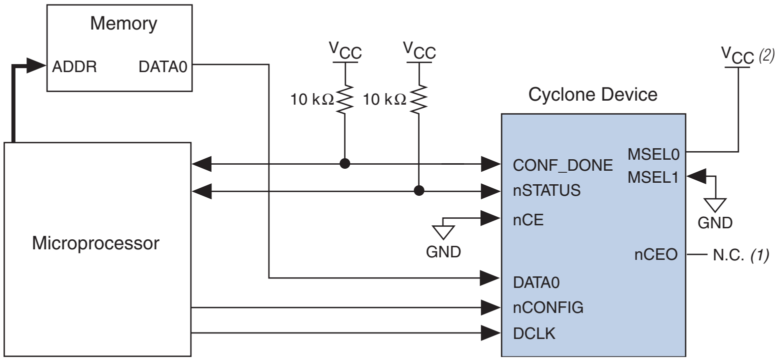
The Cyclone FPGA starts initialization using the internal oscillator after all configuration data is transferred. The device's CONF_DONE pin goes high to show successful configuration and the start of initialization. Driving DCLK to the device after configuration does not affect device operation.

Since the PS configuration scheme is a synchronous scheme, the configuration clock speed must be below the specified maximum frequency to ensure successful configuration. Maximum DCLK frequency supported by Cyclone FPGAs is 100 MHz (See Table 5 on page 27). No maximum DCLK period (i.e., minimum DCLK frequency) exists. You can pause configuration by halting DCLK for an indefinite amount of time.

If the target device detects an error during configuration, it drives its nSTATUS pin low to alert the microprocessor. The microprocessor can then pulse nCONFIG low to restart the configuration process. Alternatively, if the **Auto-Restart Configuration on Frame Error** option is turned on in the Quartus II software, the target device releases nSTATUS after a reset time-out period. After nSTATUS is released, the microprocessor can reconfigure the target device without needing to pulse nCONFIG low.

The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration and initialization. If the microprocessor sends all data but CONF_DONE and INIT_DONE has not gone high, it must reconfigure the target device. CONF_DONE should go high within 1 ms, and INIT_DONE should go high within 15 ms. Figure 14 shows the circuit for PS configuration with a microprocessor.

Figure 14. PS Configuration Circuit with a Microprocessor



Note to Figure 14:

(1) The `nCEO` pin is left unconnected.

(2) Connect `MSEL0` to the Vcc supply voltageof I/O Bank it resides in.

Configuring Cyclone FPGAs with the MicroBlaster Software

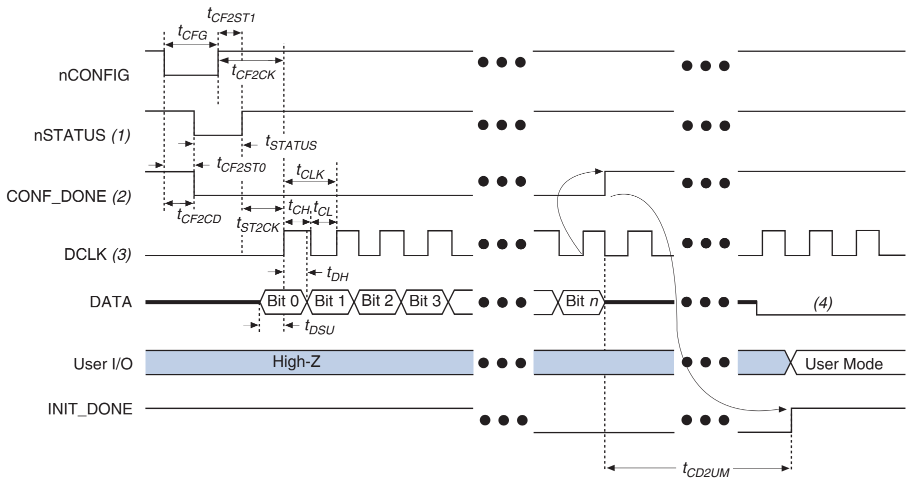
The MicroBlaster™ software driver allows you to configure Altera FPGAs, including Cyclone FPGAs, through the ByteBlaster II or ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a Raw Binary File (**.rbf**) programming input file and is targeted for embedded PS configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, see the *Configuring the MicroBlaster Passive Serial Software Driver White Paper* and source files on the Altera web site at **http://www.altera.com**.

Passive Serial Timing

For successful configuration using the PS scheme, several timing parameters such as setup, hold, and maximum clock frequency must be satisfied. The enhanced configuration and EPC2 devices are designed to meet these interface timing specifications. If you use a microprocessor or another intelligent host to control the PS interface, ensure that you meet these timing requirements.

Figure 15 shows the PS timing waveform for Cyclone FPGAs.

Figure 15. PS Timing Waveform for Cyclone FPGAs



Notes to Figure 15:

(1) Upon power-up, the Cyclone FPGA holds nSTATUS low for about 100 ms.

(2) Upon power-up and before configuration, CONF_DONE is low.

(3) In user mode, DCLK should be driven high or low when using the PS configuration scheme. When using the AS configuration scheme, DCLK is a Cyclone output pin and should not be driven externally.

(4) DATA should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Table 5 contains the PS timing information for Cyclone FPGAs.

Table 5. PS Timing Parameters for Cyclone Devices *Note (1)*

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		800	ns
t_{CF2ST0}	`nCONFIG` low to nSTATUS low		800	ns
t_{CF2ST1}	`nCONFIG` high to nSTATUS high		40 *(4)*	µs
t_{CFG}	`nCONFIG` low pulse width *(2)*	40		µs
t_{STATUS}	nSTATUS low pulse width	10	40 *(4)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	nSTATUS high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	7		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	4		ns
t_{CL}	`DCLK` low time	4		ns
t_{CLK}	`DCLK` period	10		ns
f_{MAX}	`DCLK` maximum frequency		100	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	6	20	µs

Notes to Table 5:
(1) This information is preliminary.
(2) This value applies only if the internal oscillator is selected as the clock source for device initialization. If the clock source is CLKUSR, multiply the clock period by 270 to obtain this value. CLKUSR must be running during this period to reset the device.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for device initialization. If the clock source is CLKUSR, multiply the clock period by 140 to obtain this value.
(4) You can obtain this value if you do not delay configuration by extending the nSTATUS low-pulse width.

JTAG-Based Configuration

JTAG has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on printed circuit boards (PCBs) with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. You can also use the JTAG circuitry to shift configuration data into Cyclone FPGAs. The Quartus II software automatically generates **.sof** files that can be used for JTAG configuration.

For more information on JTAG boundary-scan testing, see *AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

A device operating in JTAG mode uses four required pins, TDI, TDO, TMS, and TCK. Cyclone FPGAs do not support the optional TRST pin. The three JTAG input pins, TCK, TDI, and TMS, have weak internal pull-up resistors. All user I/O pins are tri-stated during JTAG configuration.

Cyclone is designed such that JTAG instructions have precedence over any device operating modes. So JTAG configuration can take place without waiting for other configuration to complete (e.g., configuration with serial or enhanced configuration devices). If you attempt JTAG configuration in Cyclone FPGAs during non-JTAG configuration, non-JTAG configuration will be terminated and JTAG configuration will be initiated. Table 6 shows each JTAG pin's function.

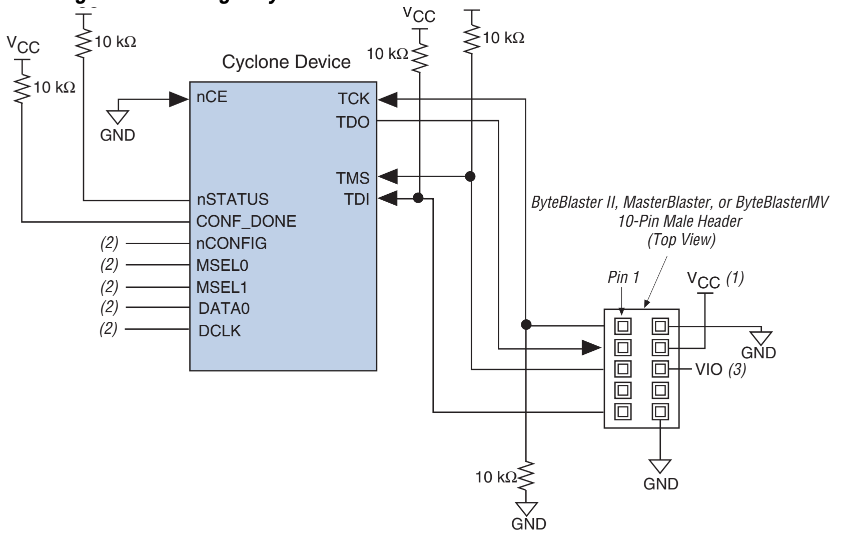
Table 6. JTAG Pin Descriptions

Pin	Description	Function
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the Test Access Port (TAP) controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge.

JTAG Configuration Using a Download Cable

During JTAG configuration, data is downloaded to the device on the board through a ByteBlaster II, ByteBlasterMV, or MasterBlaster download cable. Configuring devices through a cable is similar to programming devices in-system. See Figure 16 for pin connection information.

Figure 16. JTAG Configuration of Single Cyclone FPGA



Notes to Figure 16:

(1) You should connect the pull-up resistor to the same supply voltage as the download cable.

(2) You should connect the nCONFIG, MSEL0, and MSEL1 pins to support a non-JTAG configuration scheme. If you only use JTAG configuration, connect nCONFIG to V_{CC}, and MSEL0 and MSEL1 to ground. Pull DATA0 and DCLK to high or low.

(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

To configure a single device in a JTAG chain, the programming software places all other devices in bypass mode. In bypass mode, devices pass programming data from the TDI pin to the TDO pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the TDO pin one clock cycle later.

Cyclone FPGAs have dedicated JTAG pins. Not only can you perform JTAG testing on Cyclone FPGAs before and after, but also during configuration. While other device families do not support JTAG testing during configuration, Cyclone FPGAs support the BYPASS, IDCODE, and SAMPLE instructions during configuration without interrupting configuration. All other JTAG instructions may only be issued by first interrupting configuration and reprogramming I/O pins using the CONFIG_IO instruction.

The CONFIG_IO instruction allows I/O buffers to be configured via the JTAG port, and when issued, interrupts configuration. This instruction allows you to perform board-level testing prior to configuring the Cyclone FPGA or waiting for a configuration device to complete configuration. Once configuration has been interrupted and JTAG testing is complete, the part must be reconfigured via JTAG (PULSE_CONFIG instruction) or by pulsing nCONFIG low.

The chip-wide reset and output enable pins on Cyclone FPGAs do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of Cyclone FPGAs, you should consider the regular configuration pins. Table 7 shows how you should connect these pins during JTAG configuration.

Table 7. JTAG Termination of Unused Pins

Signal	Description
nCE	Drive all Cyclone devices in the chain low by connecting nCE to ground, pulling it down via a resistor, or driving it low by some control circuitry.
nSTATUS	Pulled to V_{CC} through a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, pull up each nSTATUS pin to V_{CC} individually. *(1)*
CONF_DONE	Pulled to V_{CC} through a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, pull up each CONF_DONE pin to V_{CC} individually. *(1)*
nCONFIG	Driven high by connecting to V_{CC}, pulling up through a resistor, or driving it high by some control circuitry.
MSEL0, MSEL1	Do not leave these pins floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie these pins to ground.
DCLK	Do not leave these pins floating. Drive low or high, whichever is more convenient.
DATA0	Do not leave these pins floating. Drive low or high, whichever is more convenient.

Note to Table 7:

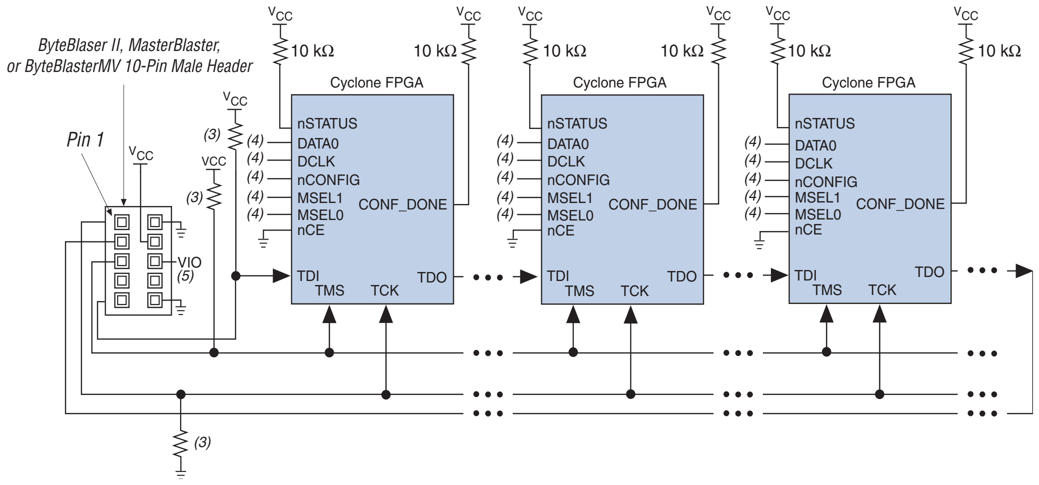
(1) nSTATUS going low in the middle of JTAG configuration indicates that an error has occurred; CONF_DONE going high at the end of JTAG configuration indicates successful configuration.

JTAG Configuration of Multiple Devices

When programming a JTAG device chain, one JTAG-compatible header, such as the ByteBlaster II header, is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capacity of the download cable. However, when four or more devices are connected in a JTAG chain, Altera recommends buffering the TCK, TDI, and TMS pins with an on-board buffer.

JTAG-chain device configuration is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 17 shows multi-device JTAG configuration.

Figure 17. Multi-Device JTAG Configuration



Notes to Figure 17:

(1) Cyclone, APEX™ II, APEX 20K, Mercury™, ACEX® 1K, and FLEX® 10K devices can be placed within the same JTAG chain for device programming and configuration.

(2) For more information on all configuration pins connected in this mode, refer to Table 6 on page 28.

(3) These pull-up / pull-down resistors are 10 kΩ.

(4) Connect the `nCONFIG`, `MSEL0`, and `MSEL1` pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground. Pull `DATA0` and `DCLK` to either high or low.

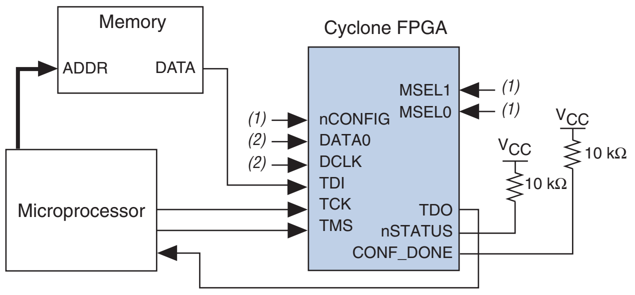
(5) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

The Quartus II software verifies successful JTAG configuration upon completion. The software checks the state of `CONF_DONE` through the JTAG port. If `CONF_DONE` is not high, the Quartus II software indicates that configuration has failed. If `CONF_DONE` is high, the software indicates that configuration was successful.

If V_{CCIO} is tied to 3.3 V, both the I / O pins and the JTAG `TDO` port drive at 3.3-V levels.

Figure 18 shows the JTAG configuration of a Cyclone FPGA with a microprocessor.

Figure 18. JTAG Configuration of Cyclone FPGAs with a Microprocessor



Notes to Figure 18:

(1) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

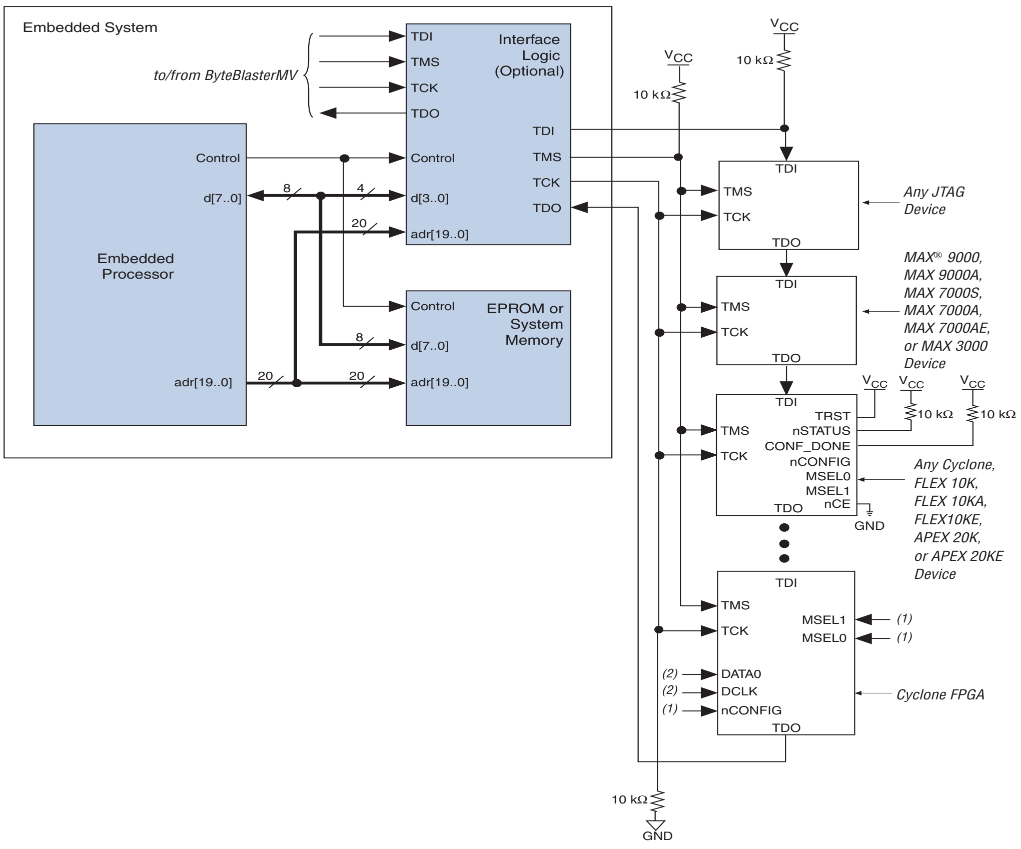
(2) Pull `DATA0` and `DCLK` to either high or low.

Connecting the JTAG Chain to the Embedded Processor

There are two ways to connect the JTAG chain to the embedded processor. The most straightforward method is to connect the embedded processor directly to the JTAG chain. In this method, four of the processor pins are dedicated to the JTAG interface, saving board space but reducing the number of available embedded processor pins.

Figure 19 illustrates the second method, which is to connect the JTAG chain to an existing bus through an interface programmable logic device (PLD). In this method, the JTAG chain becomes an address on the existing bus. The processor then reads from or writes to the address representing the JTAG chain.

Figure 19. Embedded System Block Diagram



Notes to Figure 19:

(1) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

(2) Pull `DATA0` and `DCLK` to either high or low.

Configuring Cyclone FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including Cyclone FPGAs, through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in **.rbf** format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on other platforms. For more information on the JRunner software driver, see JRunner Software Driver: An Embedded Solution to the JTAG Configuration and the source files on the Altera web site.

Jam STAPL

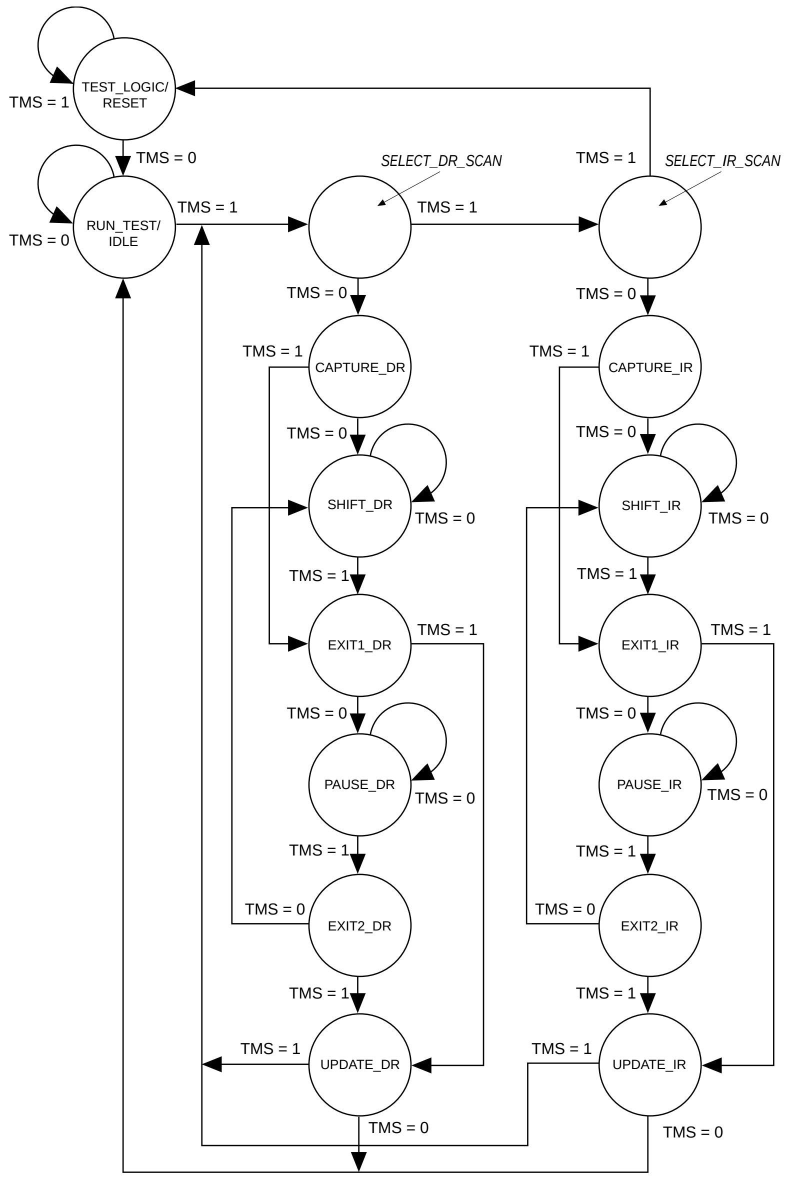
Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

☞ Both JTAG connection methods should include space for the MasterBlaster or ByteBlasterMV header connection. The header is useful during prototyping because it allows you to verify or modify the Cyclone FPGA's contents. During production, you can remove the header to save cost.

Program Flow

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine. The TAP controller is a 16-state, state machine that is clocked on the rising edge of `TCK`, and uses the `TMS` pin to control JTAG operation in a device. Figure 20 shows the flow of an IEEE Std. 1149.1 TAP controller state machine.

Figure 20. JTAG TAP Controller State Machine



While the Jam Player provides a driver that manipulates the TAP controller, the Jam Byte-Code File (**.jbc**) provides the high-level intelligence needed to program a given device. All Jam instructions that send JTAG data to the device involve moving the TAP controller through either the data register leg or the instruction register leg of the state machine. For example, loading a JTAG instruction involves moving the TAP controller to the `SHIFT_IR` state and shifting the instruction into the instruction register through the `TDI` pin. Next, the TAP controller is moved to the `RUN_TEST/IDLE` state where a delay is implemented to allow the instruction time to be latched. This process is identical for data register scans, except that the data register leg of the state machine is traversed.

The high-level Jam instructions are the `DRSCAN` instruction for scanning the JTAG data register, the `IRSCAN` instruction for scanning the instruction register, and the `WAIT` command that causes the state machine to sit idle for a specified period of time. Each leg of the TAP controller is scanned repeatedly, according to instructions in the **.jbc** file, until all of the target devices are programmed.

Figure 21 illustrates the functional behavior of the Jam Player when it parses the **.jbc** file. When the Jam Player encounters a `DRSCAN`, `IRSCAN`, or `WAIT` instruction, it generates the proper data on `TCK`, `TMS`, and `TDI` to complete the instruction. The flow diagram shows branches for the `DRSCAN`, `IRSCAN`, and `WAIT` instructions. Although the Jam Player supports other instructions, they are omitted from the flow diagram for simplicity.

Figure 21. Jam Player Flow Diagram (Part 1 of 2)

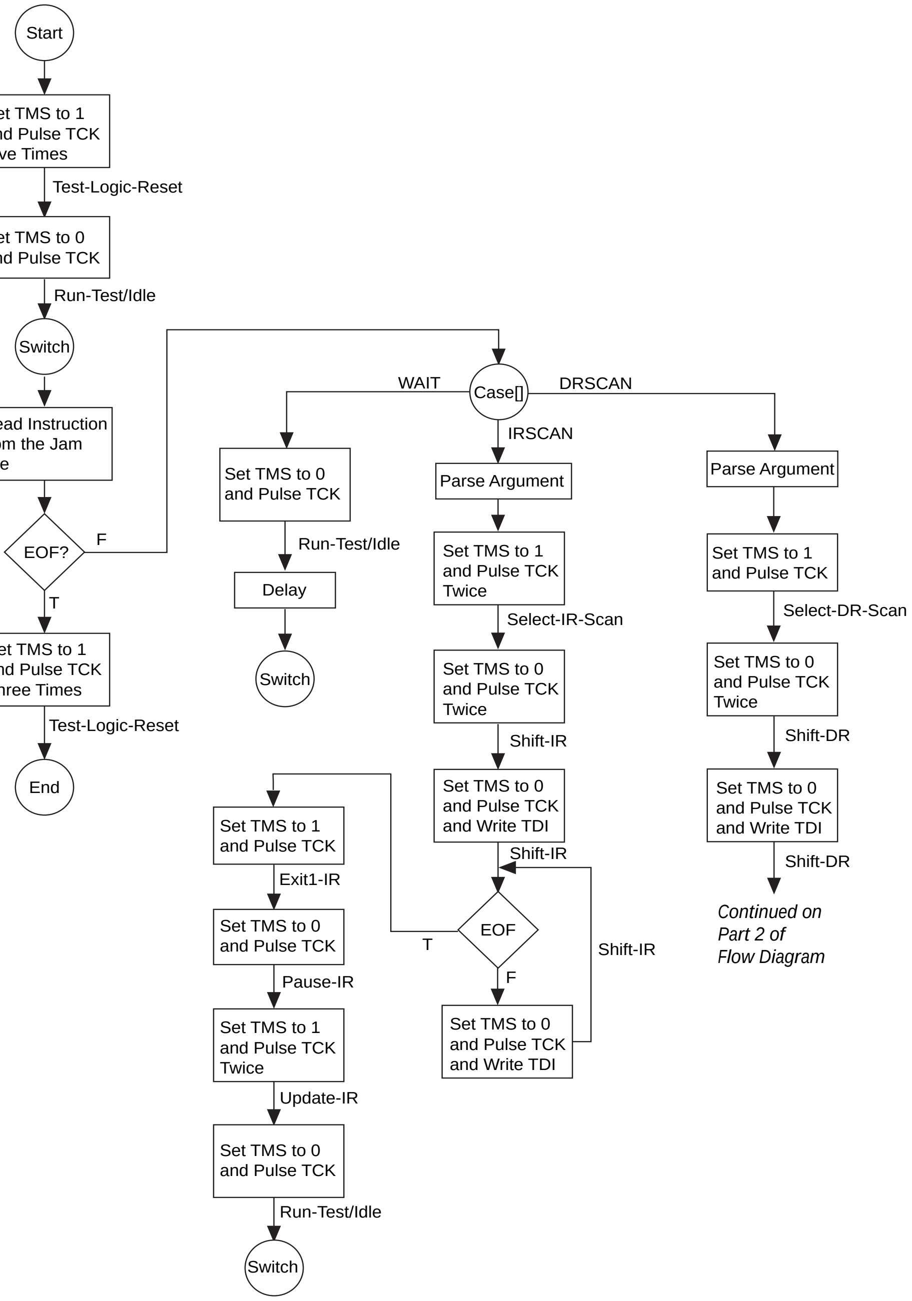
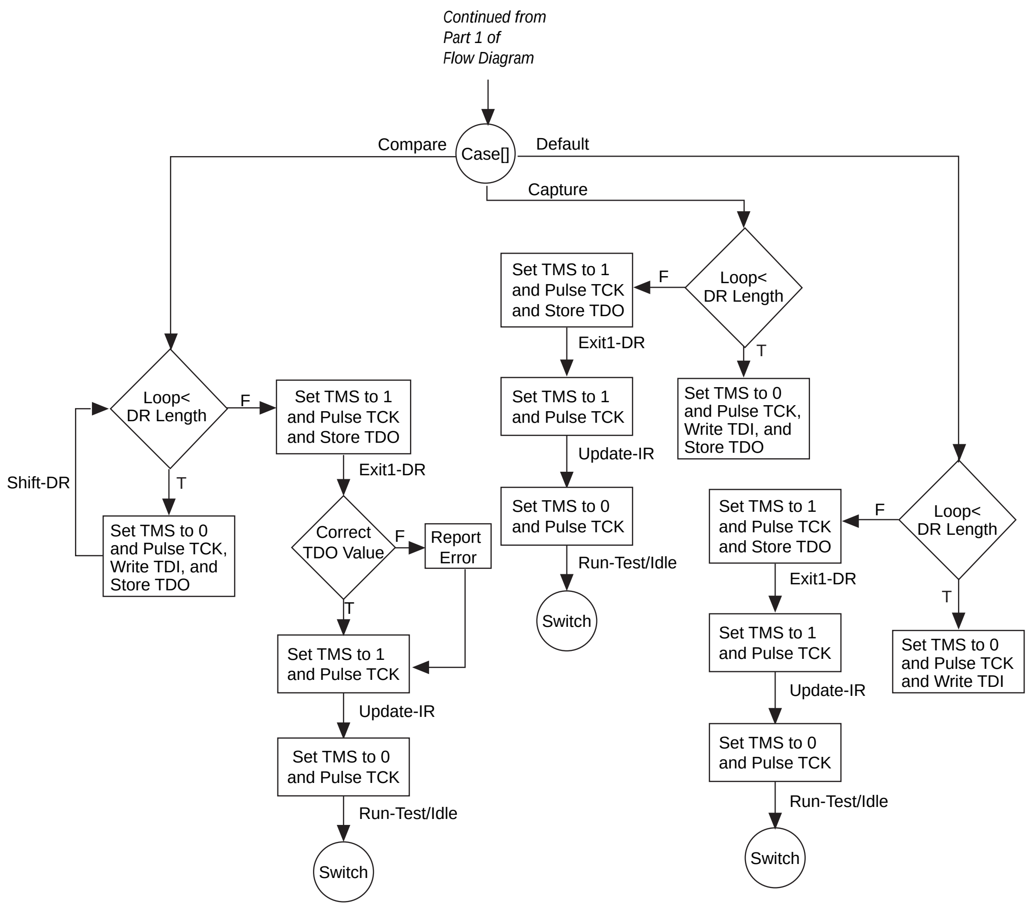

Start
Set TMS to 1 and Pulse TCK Five Times
Test-Logic-Reset
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Read Instruction from the Jam File
EOF?
F
T
Set TMS to 1 and Pulse TCK Three Times
Test-Logic-Reset
End
WAIT
Case[]
DRSCAN
IRSCAN
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Delay
Switch
Parse Argument
Set TMS to 1 and Pulse TCK Twice
Select-IR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-IR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-IR
EOF
T
F
Shift-IR
Set TMS to 0 and Pulse TCK and Write TDI
Set TMS to 1 and Pulse TCK
Exit1-IR
Set TMS to 0 and Pulse TCK
Pause-IR
Set TMS to 1 and Pulse TCK Twice
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Parse Argument
Set TMS to 1 and Pulse TCK
Select-DR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-DR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-DR
Continued on Part 2 of Flow Diagram

Figure 21. Jam Player Flow Diagram (Part 2 of 2)


Continued from Part 1 of Flow Diagram
Case[]
Compare
Default
Capture
Loop< DR Length
F
T
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK, Write TDI, and Store TDO
Shift-DR
Loop< DR Length
F
T
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Correct TDO Value
F
Report Error
T
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK, Write TDI, and Store TDO
Loop< DR Length
F
T
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK and Write TDI

Execution of a Jam program starts at the beginning of the program. The program flow is controlled using `GOTO`, `CALL/RETURN`, and `FOR/NEXT` structures. The `GOTO` and `CALL` statements refer to labels that are symbolic names for program statements located elsewhere in the Jam program. The language itself enforces almost no constraints on the organizational structure or control flow of a program.

☞ The Jam language does not support linking multiple Jam programs together or including the contents of another file into a Jam program.

Jam Instructions

Each Jam statement begins with one of the instruction names listed in Table 8. The instruction names, including the names of the optional instructions, are reserved keywords that you cannot use as variable or label identifiers in a Jam program.

Table 8. Instruction Names

BOOLEAN	INTEGER	PREIR
CALL	IRSCAN	PRINT
CRC	IRSTOP	PUSH
DRSCAN	LET	RETURN
DRSTOP	NEXT	STATE
EXIT	NOTE	WAIT
EXPORT	POP	VECTOR *(1)*
FOR	POSTDR	VMAP *(1)*
GOTO	POSTIR	-
IF	PREDR	-

Note to Table 8:

(1) This instruction name is an optional language extension.

Table 9 shows the state names that are reserved keywords in the Jam language. These keywords correspond to the state names specified in the IEEE Std. 1149.1 JTAG specification.

Table 9. Reserved Keywords

IEEE Std. 1149.1 JTAG State Names	Jam Reserved State Names
`Test-Logic-Reset`	`RESET`
`Run-Test-Idle`	`IDLE`
`Select-DR-Scan`	`DRSELECT`
`Capture-DR`	`DRCAPTURE`
`Shift-DR`	`DRSHIFT`
`Exit1-DR`	`DREXIT1`
`Pause-DR`	`DRPAUSE`
`Exit2-DR`	`DREXIT2`
`Update-DR`	`DRUPDATE`
`Select-IR-Scan`	`IRSELECT`
`Capture-IR`	`IRCAPTURE`
`Shift-IR`	`IRSHIFT`
`Exit1-IR`	`IREXIT1`
`Pause-IR`	`IRPAUSE`
`Exit2-IR`	`IREXIT2`
`Update-IR`	`IRUPDATE`

Example Jam File that Reads the IDCODE

Figure 22 illustrates the flexibility and utility of the Jam STAPL. The example reads the IDCODE out of a single device in a JTAG chain.

The array variable, `I_IDCODE`, is initialized with the IDCODE instruction bits ordered the LSB first (on the left) to most significant bit (MSB) (on the right). This order is important because the array field in the IRSCAN instruction is always interpreted and sent, MSB to LSB.

Figure 22. Example Jam File Reading IDCODE

```
BOOLEAN read_data[32];
BOOLEAN I_IDCODE[10] = BIN 1001101000; ‘assumed
BOOLEAN ONES_DATA[32] = HEX FFFFFFFF;
INTEGER i;
‘Set up stop state for IRSCAN
IRSTOP IRPAUSE;
‘Initialize device
STATE RESET;
IRSCAN 10, I_IDCODE[0..9]; ‘LOAD IDCODE INSTRUCTION
STATE IDLE;
WAIT 5 USEC, 3 CYCLES;
DRSCAN 32, ONES_DATA[0..31], CAPTURE read_data[0..31];
‘CAPTURE IDCODE
PRINT “IDCODE:”;
FOR i=0 to 31;
PRINT read_data[i];
NEXT i;
EXIT 0;
```

Combining Configuration Schemes

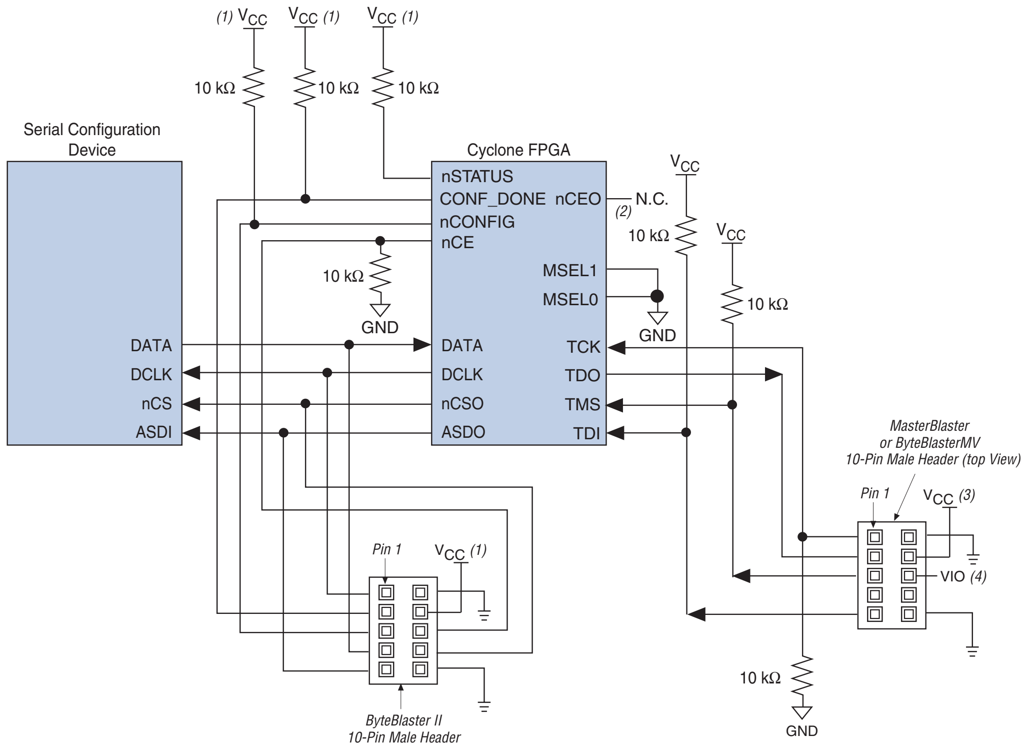
This section shows you how to configure Cyclone FPGAs using multiple configuration schemes on the same board.

Active Serial & JTAG

You can combine the AS configuration scheme with JTAG-based configuration. Set the `MSEL[1..0]` pins to `00` in this setup, as shown in Figure 23. This setup uses two 10-pin download cable headers on the board. The first header programs the serial configuration device in-system via the AS programming interface, and the second header configures the Cyclone FPGA directly via the JTAG interface.

If you try configuring the device using both schemes simultaneously, JTAG configuration takes precedence and AS configuration will be terminated.

Figure 23. Combining AS & JTAG Configuration



Notes to Figure 23:

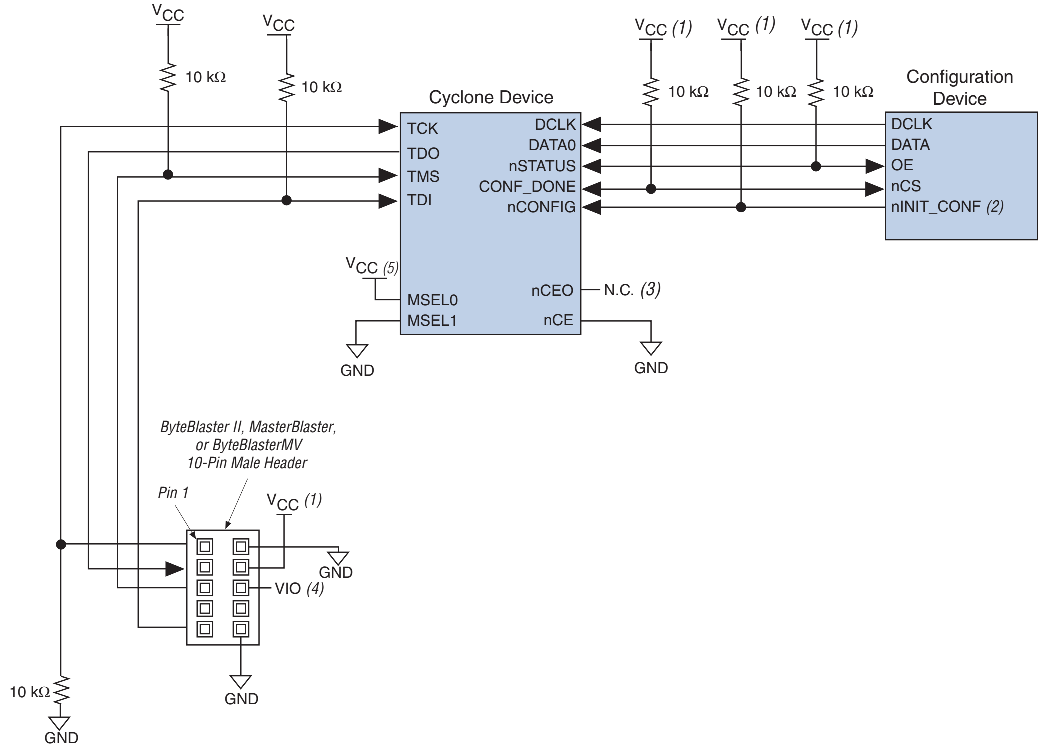
(1) Connect these pull-up resistors to 3.3 V.
(2) The `nCEO` pin is left unconnected.
(3) You should connect the pull-up resistor to the same supply voltage as the download cable.
(4) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the MasterBlaster Serial / USB Communications Cable Data Sheet for this value.

Passive Serial & JTAG

The PS- and JTAG-based configuration are also supported on the same board. Set the `MSEL[1..0]` pins to 01 in this setup. Figure 24 shows the pin connections required for configuring Cyclone FPGAs using PS and JTAG interfaces on the same board. The JTAG chain only connects to the Cyclone FPGA in Figure 24, but could also connect to the configuration device for in-system programming of that device.

If you try configuring the device using both schemes simultaneously, JTAG configuration takes precedence and PS configuration will be terminated.

Figure 24. Combining PS & JTAG Configuration



Notes to Figure 24:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. The EPC16, EPC8, EPC4, and EPC2 devices' `OE` and `nCS` pins have internal, user-configurable pull-up resistors. If you use internal pull-up resistors, do not use external pull-up resistors on these pins.

(2) The `nINIT_CONF` pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If `nINIT_CONF` is not used, `nCONFIG` must be pulled to V_{CC} through a resistor.

(3) The `nCEO` pin is left unconnected for the last device in the chain.

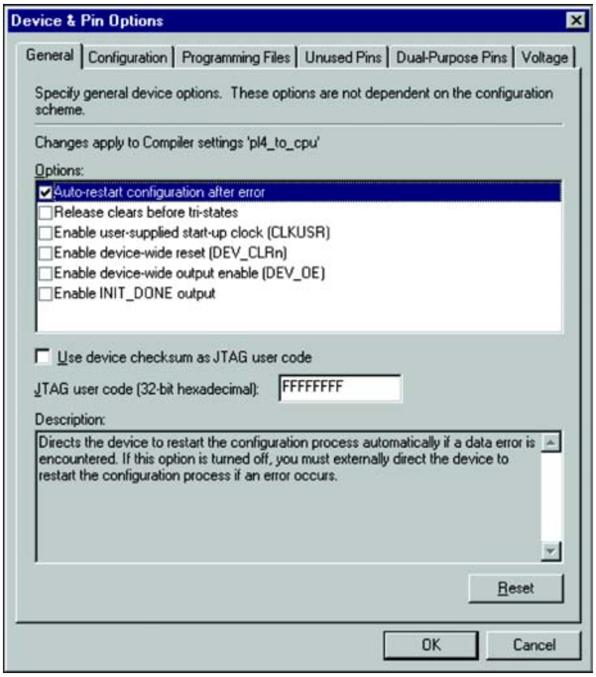
(4) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

(5) Connect `MSEL0` to the Vcc supply voltage of the I/O Bank it resides in.

Device Options

You can set Cyclone FPGA options in Altera's Quartus II development software using the **Device & Pin Options** dialog box. Select **Compiler Settings** (Processing menu), then click on the **Chips & Devices** tab. Figure 25 shows the **Device & Pin Options** dialog box.

Figure 25. Configuration Options Dialog Box



Specify general device options. These options are not dependent on the configuration scheme.

Changes apply to Compiler settings 'pl4_to_cpu'

Options:

- [✓] Auto-restart configuration after error
- [ ] Release clears before tri-states
- [ ] Enable user-supplied start-up clock (CLKUSR)
- [ ] Enable device-wide reset (DEV_CLRn)
- [ ] Enable device-wide output enable (DEV_OE)
- [ ] Enable INIT_DONE output

[ ] Use device checksum as JTAG user code

JTAG user code (32-bit hexadecimal): FFFFFFFF

Description:

Directs the device to restart the configuration process automatically if a data error is encountered. If this option is turned off, you must externally direct the device to restart the configuration process if an error occurs.

Table 10 summarizes each of these options.

Table 10. Cyclone Configuration Option Bits (Part 1 of 2)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Auto-restart configuration on frame error	If a data error occurs during configuration, you can choose how to restart configuration.	The configuration process stops until you direct the device to restart configuration. The nSTATUS pin is driven low when an error occurs. When nCONFIG is pulled low and then high, the device begins to reconfigure.	The configuration process restarts automatically. The nSTATUS pin drives low and releases. The nSTATUS pin is then pulled to V_{CC} by the pull-up resistor, indicating that configuration can restart. In the configuration device scheme, if the target device's nSTATUS pin is tied to the configuration device's OE pin, the nSTATUS reset pulse resets the configuration device automatically. The configuration device then releases its OE pin (which is pulled high) and reconfiguration begins. If an error occurs during passive configuration, the device can be reconfigured without the system having to pulse nCONFIG. After nSTATUS goes high, reconfiguration can begin.
Release clears before tri-states	During configuration, the device I/O pins are tri-stated. During initialization, you choose the order for releasing the tri-states and clearing the registers.	The device releases the tri-states on its I/O pins before releasing the clear signal on its registers.	The device releases the clear signals on its registers before releasing the tri-states. You can use this option to allow the design to operate before it drives out, so all outputs do not start up low.
Enable chip-wide reset	Enables a single pin to reset all device registers.	Chip-wide reset is not enabled. The DEV_CLRn pin is available as a user I/O pin.	Chip-wide reset is enabled for all registers in the device. All registers are cleared when the DEV_CLRn pin is driven low.

Table 10. Cyclone Configuration Option Bits (Part 2 of 2)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Enable chip-wide output enable	Enables a single pin to control all device tri-states.	Chip-wide output enable is not enabled. The `DEV_OE` pin is available as a user I/O pin.	Chip-wide output enable is enabled for all device tri-states. After configuration, all user I/O pins are tri-stated when `DEV_OE` is low.
Enable `INIT_DONE` output	Enables a pin to drive out a signal when the initialization process is complete and the device has entered user mode.	The `INIT_DONE` signal is not available. The `INIT_DONE` pin is available as a user I/O pin.	The `INIT_DONE` signal is available on the open-drain `INIT_DONE` pin. This pin drives low during configuration. After initialization, it is released and pulled high externally. The `INIT_DONE` pin must be connected to a 10-kΩ pull-up resistor. If the `INIT_DONE` output is used, the `INIT_DONE` pin cannot be used as a user I/O pin.
Data Compression	Enables Cyclone FPGAs to receive compressed configuration bit stream in Active and PS configuration schemes.	The Quartus II software generates uncompressed programming files and Cyclone FPGAs do not decompress data.	Quartus II software generates compressed programming files and Cyclone FPGAs decompress the bit stream during configuration.

Device Configuration Pins

Table 11 summarizes the Cyclone FPGA configuration pins.

Table 11. Pin Functions (Part 1 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`MSEL1` `MSEL0`	–	All	Input	Two-bit configuration input. Sets the Cyclone device configuration scheme. After configuration, the Cyclone FPGA is not affected by logic levels on this pin.
`nSTATUS`	–	All	Bidirectional open-drain	The device drives `nSTATUS` low immediately after power-up and releases it within 5 µs. (When using a configuration device, the configuration device holds `nSTATUS` low for up to 200 ms.) The `nSTATUS` pin must be pulled up to V_{CC} with a 10-kΩ resistor. If an error occurs during configuration, `nSTATUS` is pulled low by the target device. If an external source drives the `nSTATUS` pin low during configuration or initialization, the target device enters an error state. Driving `nSTATUS` low after configuration and initialization does not affect the configured device. However, if a configuration device is used, driving `nSTATUS` low will cause that device to attempt to configure the Cyclone FPGA.
`nCONFIG`	–	All	Input	Configuration control input. A low transition resets the target device; a low-to-high transition begins configuration. All I/O pins go tri-state when setting `nCONFIG` low.

Table 11. Pin Functions (Part 2 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
CONF_DONE	–	All	Bidirectional open-drain	Status output. The target device drives the CONF_DONE pin low before and during configuration. Once all configuration data is received without error and the initialization clock cycle starts, the target device releases CONF_DONE. Status input. After all data is received and CONF_DONE goes high, the target device initializes and enters user mode. The CONF_DONE pin must be pulled to V_{CC} with a 10-kΩ resistor. An external source can drive this pin low to delay the initialization process, except when configuring with a configuration device. Driving CONF_DONE low after configuration and initialization does not affect the configured device.
DCLK	–	PS AS	Input (PS) Output (AS)	In PS configuration, the clock input clocks data from an external source into the target device. In AS configuration, DCLK is an output from the Cyclone FPGA that provides timing for the configuration interface. After configuration, the logic levels on this pin do not affect the Cyclone FPGA.
ASDO	I/O	AS	Output	Control signal from the Cyclone FPGA to the serial configuration device in AS mode used to read out configuration data.
nCSO	I/O	AS	Output	Control signal from the Cyclone FPGA to the serial configuration device in AS mode that enables the configuration device.
nCE	–	All	Input	Active-low chip enables. The nCE pin activates the device with a low signal to allow configuration and should be tied low for single device configuration. The nCE pin must be held low during configuration, initialization, and user mode.
nCEO	I/O	Multi-device	Output	Output that drives low when device configuration is complete. During multi-device configuration, this pin feeds a subsequent device's nCE pin.
CLKUSR	I/O	All	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices.

Table 11. Pin Functions (Part 3 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`DATA0`	–	Configuration device PS PPA FPP	Input	Data input. In serial configuration mode, bit-wide configuration data is presented to the target device on the `DATA0` pin.
`INIT_DONE`	I/O	All	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. The `INIT_DONE` pin must be pulled to V_{CC} with a 10-k ohm resistor. The `INIT_DONE` pin drives low during configuration. Before and after configuration, the `INIT_DONE` pin is released and is pulled to V_{CC} by an external pull-up resistor. Because `INIT_DONE` is tri-stated before configuration, it is pulled high by the external pull-up resistor. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This option is set in the Quartus II software.
`DEV_OE`	I/O	All	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This option is set in the Quartus II software.
`DEV_CLRn`	I/O	All	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This option is set in the Quartus II software.
`TDI`	JTAG pins	All	Input	JTAG pins. JTAG pins must be kept stable before and during configuration. JTAG pin stability prevents accidental loading of JTAG instructions.
`TDO`			Output	
`TMS`			Input	
`TCK`			Input	

Device Configuration Files

The Quartus II software can create one or more configuration and programming files to support the configuration schemes discussed in this application note. This section describes these files.

SRAM Object File (.sof)

You should use an **.sof** during PS and JTAG configuration when the data is downloaded directly from the ByteBlaster II, MasterBlaster, or ByteBlasterMV download cables. For Cyclone FPGAs, the Quartus II Compiler's Assembler module automatically creates the **.sof** file for each device in your design. The Quartus II software controls the configuration sequence and automatically inserts the appropriate headers into the configuration data stream. All other configuration files are created from the **.sof**.

Programmer Object File (.pof)

A **.pof** is used by the Altera programming hardware to program a configuration device, including serial configuration devices and enhanced configuration devices. A **.pof** is automatically generated when a Cyclone project is compiled for the configuration device selected in the **Configuration** dialog box.

Raw Binary File (.rbf)

The **.rbf** is a binary file (e.g., one byte of **.rbf** data is eight configured bits 10000101 (85 Hex)) containing the configuration data. Store data so that the LSB of each data byte is loaded first. A mass storage device can store the converted image. The microprocessor can then read data from the binary file and load it into device. You can also use the microprocessor to perform real-time conversion during configuration. In the PS configuration scheme, the data is shifted in serially, LSB first.

Hexadecimal (Intel-Format) File (.hex)

A **.hex** file is an ASCII file in the Intel hexidecimal format. Third-party programmers use this file to program Altera's serial configuration devices. Microprocessors can also use the **.hex** file to store and transmit configuration data using the PS configuration scheme.

Tabular Text File (.ttf)

The **.ttf** file is a tabular ASCII file that provides a comma-separated version of the configuration data for the bit-wide PS configuration scheme. In some applications, the storage device containing the configuration data is neither dedicated to nor connected directly to the target device. For example, a configuration device can also contain executable code for a system (e.g., BIOS routines) and other data. The **.ttf** allows you to include the configuration data as part of the microprocessor's source code using the include or source commands. The microprocessor can access this data from a configuration device or mass-storage device and load it into the target device. A **.ttf** can be imported into nearly any assembly language or high-level language compiler.

Jam File (.jam)

A **.jam** file is an ASCII text file in the Jam device programming language that stores device programming information. These files are used to program, verify, and blank-check one or more devices in the Quartus II Programmer or in an embedded processor-type environment.

Jam Byte-Code File (.jbc)

A **.jbc** file is a binary version of a Jam file in a byte-code representation. The **.jbc** file stores device programming information used to program, verify, and blank-check one or more devices.

Configuration Reliability

The Cyclone architecture is designed to minimize the effects of power supply and data noise in a system, and to ensure that the configuration data is not corrupted during configuration or normal user-mode operation. A number of circuit design features ensure the highest possible level of reliability from this SRAM technology.

Cyclic redundancy code (CRC) circuitry validates each data frame (i.e., sequence of data bits) as it is loaded into the target device. If the CRC generated by the device does not match the data stored in the data stream, the configuration process is halted, and the nSTATUS pin is pulled and held low to indicate an error condition. CRC circuitry ensures that noisy systems will not cause errors that yield an incorrect or incomplete configuration.

The Cyclone FPGA architecture also provides a very high level of reliability in low-voltage brown-out conditions. Cyclone FPGA SRAM blocks require a certain V_{CC} level to maintain accurate data. This voltage threshold is significantly lower than the voltage required to activate the device's POR circuitry. Therefore, the target device stops operating if the V_{CC} starts to fail, and indicates an operation error by pulling and holding the nSTATUS pin low. You must then reconfigure the device before it can resume operation as a logic device. In active configuration schemes in which nCONFIG is tied to V_{CC}, reconfiguration begins as soon as V_{CC} returns to an acceptable level. The low pulse on nSTATUS resets the configuration device by driving OE low. In passive configuration schemes, the host system starts the reconfiguration process.

These device features ensure that Cyclone FPGAs have the highest possible reliability in a wide variety of environments, and provide the same high level of system reliability that exists in other Altera PLDs.

Board Layout Tips

Even though the DCLK signal (used in PS and AS configuration schemes) is fairly low-frequency, it drives edge-triggered pins on the Cyclone FPGA. Therefore, any overshoot, undershoot, ringing, or other noise can affect configuration. When designing the board, lay out the DCLK trace using the same techniques as laying out a clock line, including appropriate buffering. If more than five devices are used, Altera recommends using buffers to split the fan-out on the DCLK signal.

Revision History

The information contained in AN 250: Configuring Cyclone FPGAs , version 1.1, supersedes information published in previous versions. The following changes were made in AN 250: Configuring Cyclone FPGAs , version 1.1:

- Updates to existing figures and notes.
- Added two new screen captures, Figure 2 and Figure 3.
- Added information for EP1C4 and removed EPC1441.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com

Copyright © 2003 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

nsai
I.S. EN ISO 9001

Printed on Recycled Paper.