8. Configuring Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K Devices

CF51006-2.1

Introduction

Mercury™, APEX™ 20K (2.5 V), ACEX® 1K, and FLEX® 10K devices can be configured using one of four configuration schemes. All configuration schemes use either a microprocessor or configuration device.

This section covers how to configure Mercury, APEX 20K (2.5 V), ACEX 1K and FLEX 10K devices. APEX 20K (non-E and non-C) devices use a 2.5-V voltage supply for V_{CCINT}, while APEX 20KE and APEX 20KC devices use a 1.8-V voltage supply for V_{CCINT}. If your target FPGA is an APEX 20K device which uses a 1.8-V V_{CCINT}. For more information, refer to *Configuring APEX 20KE & APEX 20KC Devices* in the *Configuration Handbook*.

Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices can be configured using the passive serial (PS), passive parallel synchronous (PPS), passive parallel asynchronous (PPA), and Joint Test Action Group (JTAG) configuration schemes. The configuration scheme used is selected by driving the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device `MSEL1` and `MSEL0` pins either high or low as shown in Table 8–1. If your application only requires a single configuration mode, the `MSEL` pins can be connected to V_{CC} (V_{CCIO} of the I/O bank where the `MSEL` pin resides) or to ground. If your application requires more than one configuration mode, you can switch the `MSEL` pins after the FPGA is configured successfully. Toggling these pins during user-mode does not affect the device operation; however, the `MSEL` pins must be valid before a reconfiguration is initiated.

Table 8–1. Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	PS
1	0	PPS
1	1	PPA
(1)	*(1)*	JTAG Based *(2)*

Notes to Table 8–1:
(1) Do not leave the `MSEL` pins floating; connect them to a low- or high-logic level. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used, you should connect the `MSEL` pins to ground.
(2) JTAG-based configuration takes precedence over other configuration schemes, which means `MSEL` pin settings are ignored.

Tables 8–2 through 8–5 show the approximate configuration file sizes for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices.

Table 8–2. Mercury Raw Binary File (.rbf) Sizes

Device	Data Size (Bits)	Data Size (Bytes)
EP1M120	1,303,120	162,890
EP1M350	4,394,032	549,254

Table 8–3. APEX 20K (2.5 V) Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EP20K100	993,360	124,170
EP20K200	1,950,800	243,850
EP20K400	3,880,720	485,090

Table 8–4. ACEX 1K Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EP1K10	159,160	19,895
EP1K30	473,720	59,215
EP1K50	784,184	98,023
EP1K100	1,335,720	166,965

Table 8–5. FLEX 10K Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EPF10K30E	473,720	59,215
EPF10K50E	784,184	98,023
EPF10K50S	784,184	98,023
EPF10K100B	1,200,000	150,000
EPF10K100E	1,335,720	166,965
EPF10K130E	1,838,360	229,795
EPF10K200E	2,756,296	344,537
EPF10K200S	2,756,296	344,537
EPF10K10A	120,000	15,000
EPF10K30A	406,000	50,750
EPF10K50V	621,000	77,625
EPF10K100A	1,200,000	150,000
EPF10K130V	1,600,000	200,000
EPF10K250A	3,300,000	412,500
EPF10K10	118,000	14,750
EPF10K20	231,000	28,875
EPF10K30	376,000	47,000
EPF10K40	498,000	62,250
EPF10K50	621,000	77,625
EPF10K70	892,000	111,500
EPF10K100	1,200,000	150,000

Use the data in Tables 8–2 through 8–5 only to estimate the file size before design compilation. Different configuration file formats, such as a Hexidecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II or MAX+PLUS® II software, all designs targeted for the same device will have the same configuration file size.

The following chapter describes in detail how to configure Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using the supported configuration schemes. The last section describes the device configuration pins available. In this chapter, the generic term device(s) or FPGA(s) will include all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices.

For more information on setting device configuration options or creating configuration files, Refer to *Software Settings*, chapter 6 and 7 in volume 2 of the *Configuration Handbook.*

Passive Serial Configuration

You can perform Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K PS configuration using an Altera configuration device, an intelligent host (e.g., a microprocessor or Altera® MAX® device), or a download cable.

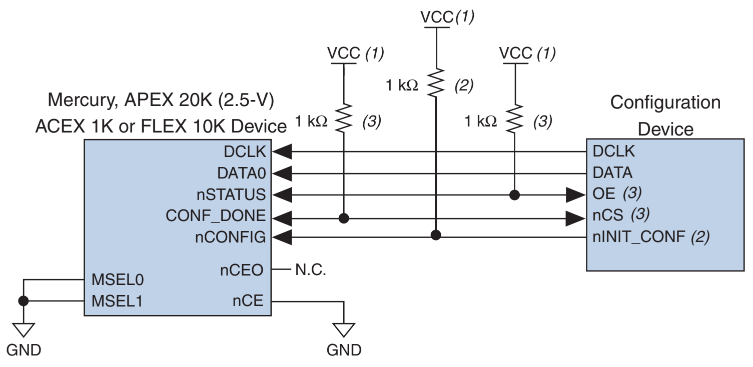
PS Configuration Using a Configuration Device

You can use an Altera configuration device, such as an enhanced configuration device, EPC2, or EPC1 device, to configure Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 8–1 shows the configuration interface connections between Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K devices and a configuration device.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on the configuration device and flash interface pins (e.g., `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`), refer to the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*, chapter 2 in volume 1 of the *Configuration Handbook*.

Figure 8–1. Single Device PS Configuration Using a Configuration Device



Notes to Figure 8–1:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the `nINIT_CONF`/`nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the Operating Conditions table of the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the *Configuration Handbook*.

When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the `INIT_CONF` JTAG instruction to initiate FPGA configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up on the `nINIT_CONF` pin is always active in enhanced configuration devices and EPC2 devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for EPC2, EPC1, and EPC1441 devices is 200 ms (maximum), and for enhanced configuration devices, the POR time can be set to either 100 ms or 2 ms, depending on its PORSEL pin setting. If the PORSEL pin is connected to GND, the POR delay is 100 ms. During this time, 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 both devices complete POR, they release their open-drain OE or nSTATUS pin, which is then pulled high by a pull-up resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. The beginning of configuration can be delayed by holding the nCONFIG or nSTATUS pin low.

VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the nSTATUS pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up on the OE pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 1-kΩ pull-up resistor on the OE/nSTATUS line is required. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins.

When nSTATUS is pulled high, OE of the configuration device also goes high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK.

After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up is not used, an external 1-kΩ pull-up resistor on the nCS/CONF_DONE line is required. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device will allow enough clock cycles for proper initialization.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, an external host must provide the initialization clock on the DCLK pin. Programming files generated by the Quartus II or MAX+PLUS II software already have these initialization clock cycles included in the file.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. You can turn on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 1-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. The enhanced configuration device and EPC2 device drive DCLK low and DATA high (EPC1 devices tri-state DATA) at the end of configuration.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the *Auto-Restart Configuration After Error* option available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box is turned on, the FPGA automatically initiates reconfiguration if an error occurs. The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

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 FPGA has not configured successfully. Enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit was sent for CONF_DONE to reach a high state. EPC1 and EPC2 devices wait for 16 DCLK cycles. In this case, the configuration device pulls its OE pin low, which in turn drives the target device's nSTATUS pin low. If the *Auto-Restart Configuration After Error* option is set in the software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns high, the configuration device tries to reconfigure the FPGA.

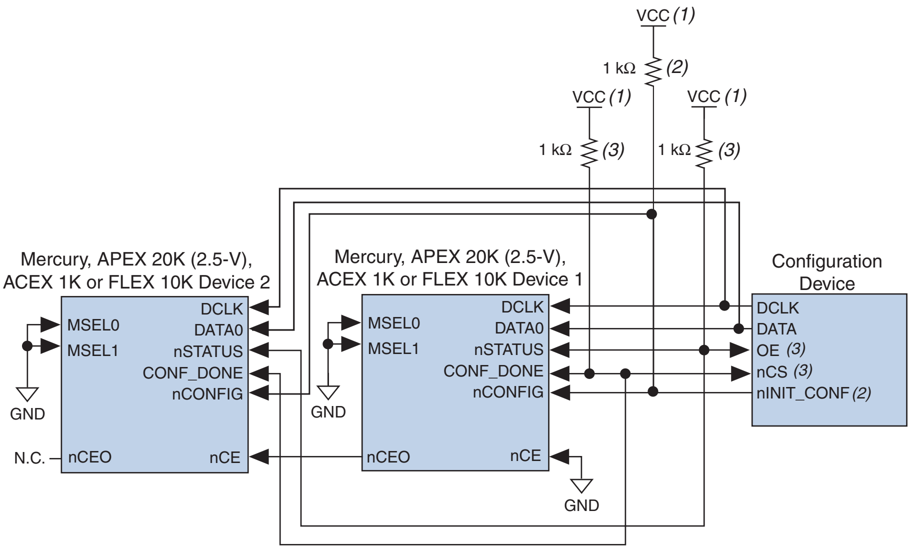
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully; therefore, your system should not pull CONF_DONE low to delay initialization. Instead, use the CLKUSR option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this activates the configuration device since it will see its nCS pin drive low. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 8–2 shows how to configure multiple devices with a configuration device. This circuit is similar to the configuration device circuit for a single device, except Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are cascaded for multi-device configuration.

Figure 8–2. Multi-Device PS Configuration Using a Configuration Device



Notes to Figure 8–2:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the `nINIT_CONF/nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

When performing multi-device configuration, you must generate the configuration device's Programmer Object File (**.pof**) from each project's SRAM Object File (**.sof**). You can combine multiple SOFs using the Quartus For more information on how to create configuration files for multi-device configuration chains, refer to *Software Settings*, chapter 6 and 7 in volume 2 of the *Configuration Handbook*.

In multi-device PS configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device.

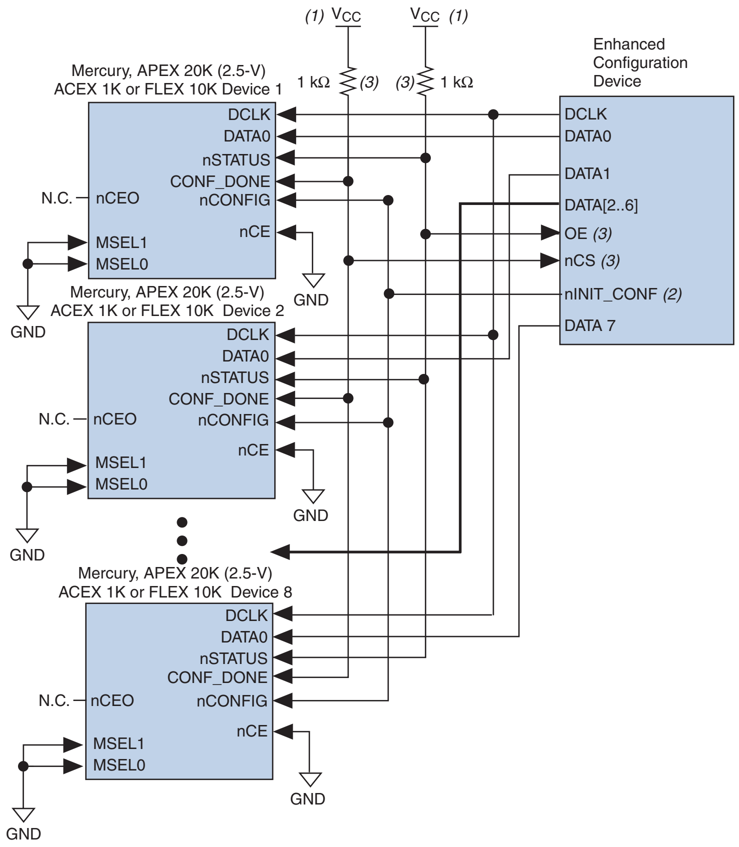
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the *Auto-Restart Configuration After Error* option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

Enhanced configuration devices also support parallel configuration of up to eight devices. The n-bit (n = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure FPGAs or a chain of FPGAs. In addition, these devices do not have to be the same device family or density; they can be any combination of Altera FPGAs. An individual enhanced configuration device DATA line is available for each targeted FPGA. Each DATA line can also feed a daisy chain of FPGAs. Figure 8–3 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 8–3. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device


(1) VCC
VCC (1)
1 kΩ (3)
(3) 1 kΩ
Enhanced Configuration Device
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 1
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 2
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 8
DCLK
DATA0
nSTATUS
CONF_DONE
nCONFIG
nCEO
nCE
MSEL1
MSEL0
N.C.
GND
DATA1
DATA[2..6]
OE (3)
nCS (3)
nINIT_CONF (2)
DATA 7

Notes to Figure 8–3:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the `nINIT_CONF/nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

The Quartus II software only allows the selection of n-bit PS configuration modes, where n must be 1, 2, 4, or 8. However, you can use these modes to configure any number of devices from 1 to 8. When configuring SRAM-based devices using n-bit PS modes, use Table 8–6 to select the appropriate configuration mode for the fastest configuration times.

Table 8–6. Recommended Configuration Using n-Bit PS Modes

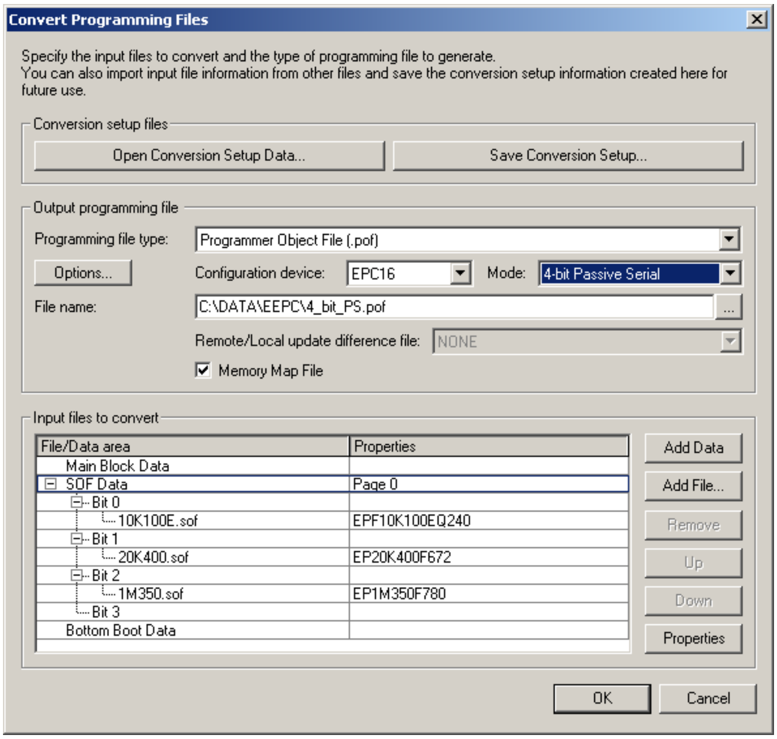
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 8–6:

(1) Assume that each DATA line is only configuring one device, not a daisy chain of devices.

For example, if you configure three FPGAs, you would use the 4-bit PS mode. For the DATA0, DATA1, and DATA2 lines, the corresponding SOF data is transmitted from the configuration device to the FPGA. For DATA3, you can leave the corresponding Bit3 line blank in the Quartus II software. On the printed circuit board (PCB), leave the DATA3 line from the enhanced configuration device unconnected. Figure 8–4 shows the Quartus II **Convert Programming Files** window (Tools menu) setup for this scheme.

Figure 8–4. Software Settings for Configuring Devices Using n-Bit PS Modes



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: EPC16 Mode: 4-bit Passive Serial

File name: C:\DATA\EEPC\4_bit_PS.pof

Remote/Local update difference file: NONE

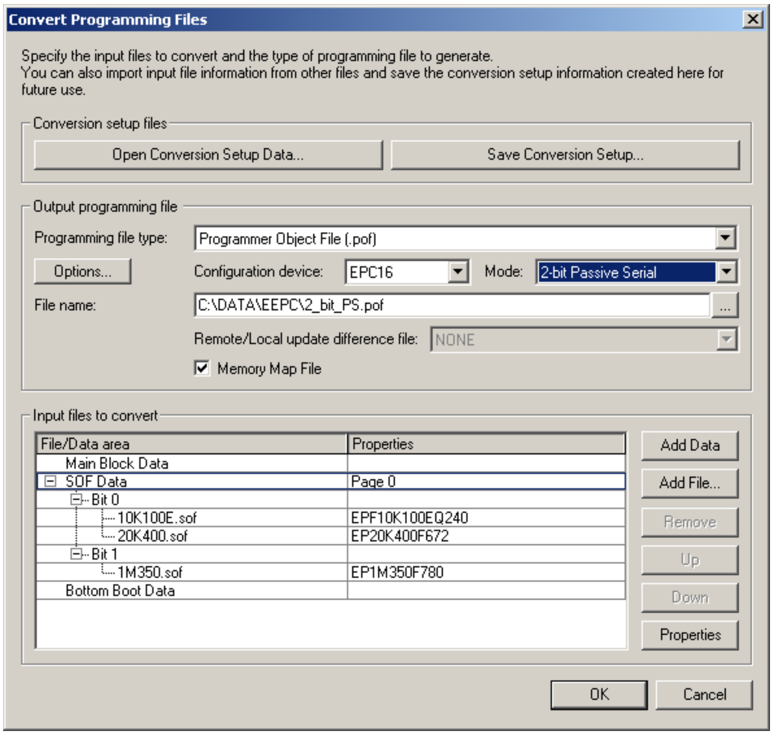
[✓] Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
10K100E.sof	EPF10K100EQ240
Bit 1	
20K400.sof	EP20K400F672
Bit 2	
1M350.sof	EP1M350F780
Bit 3	
Bottom Boot Data	

Alternatively, you can daisy chain two FPGAs to one `DATA` line while the other `DATA` lines drive one device each. For example, you could use the 2-bit PS mode to drive two FPGAs with `DATA Bit0` (EPF10K100E and EP20K400 devices) and the third device (the EP1M350 device) with `DATA Bit1`. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but can increase the total system configuration time (Figure 8–5).

Figure 8–5. Software Settings for Daisy Chaining Two FPGAs on One DATA Line


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

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: Programmer Object File (.pof)

Options... Configuration device: EPC16 Mode: 2-bit Passive Serial

File name: C:\DATA\EEPC\2_bit_PS.pof

Remote/Local update difference file: NONE

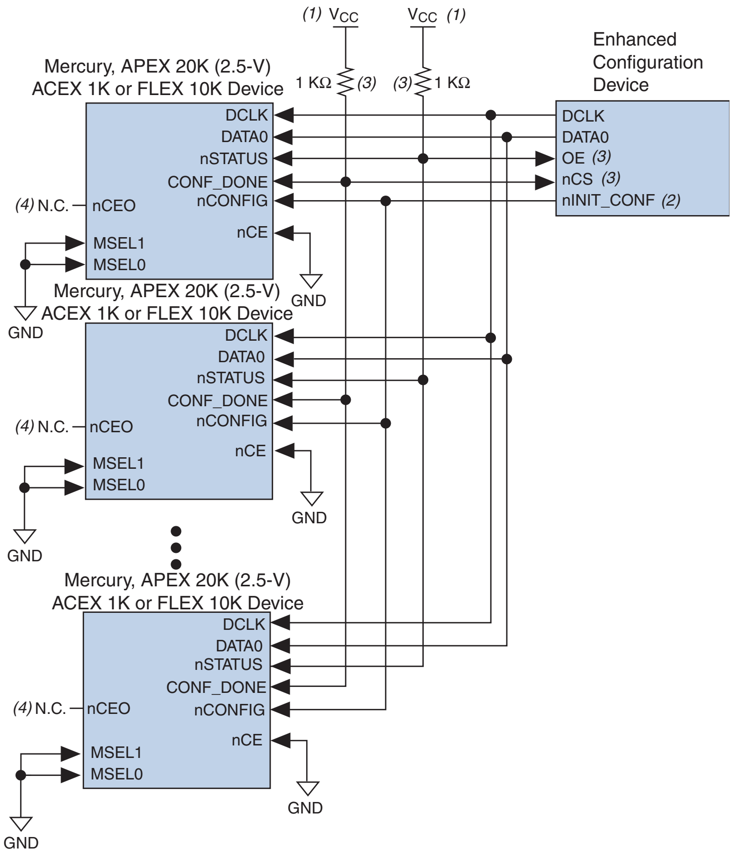
☑ Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
⊟ SOF Data	Page 0
⊟ Bit 0	
10K100E.sof	EPF10K100EQ240
20K400.sof	EP20K400F672
⊟ Bit 1	
1M350.sof	EP1M350F780
Bottom Boot Data	

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DCLK` and `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–6 shows multi-device PS configuration when the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are receiving the same configuration data.

Figure 8–6. Multiple-Device PS Configuration Using an Enhanced Configuration Device When FPGAs Receive the Same Data



Notes to Figure 8–6:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the `nINIT_CONF`/`nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

(4) The `nCEO` pins of all devices are left unconnected when configuring the same configuration data into multiple devices.

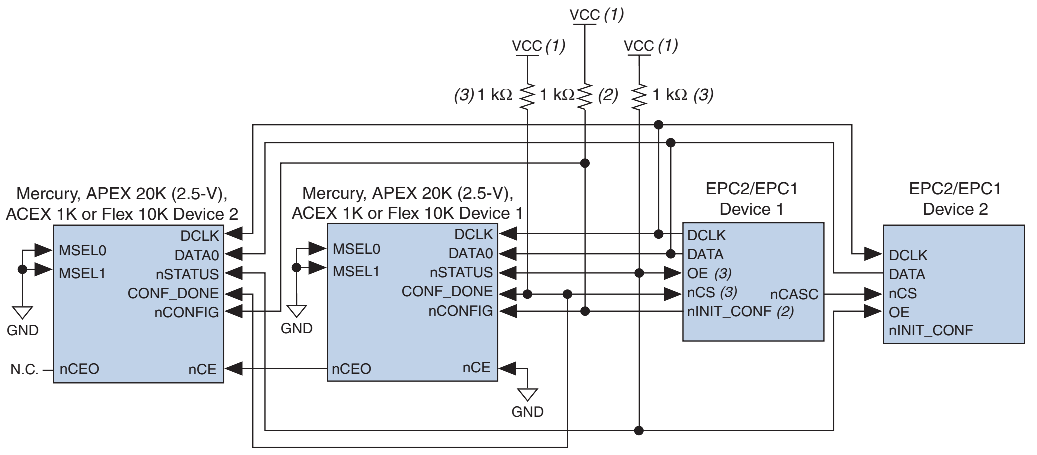
You can cascade several EPC2 or EPC1 devices to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. The first configuration device in the chain is the master configuration device, while the subsequent devices are the slave devices. The master configuration device sends `DCLK` to the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices and to the slave configuration devices. The first EPC device's `nCS` pin is connected to the `CONF_DONE` pins of the FPGAs, while its `nCASC` pin is connected to `nCS` of the next configuration device in the chain. The last device's `nCS` input comes from the previous device, while its `nCASC` pin is left floating. When all data from the first configuration device is sent, it drives `nCASC` low, which in turn drives `nCS` on the next configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

Enhanced configuration devices EPC4, EPC8, and EPC16 cannot be cascaded.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect `CONF_DONE` going high at the end of configuration, it resets the entire chain by pulling its `OE` pin low. This low signal drives the `OE` pin low on the slave configuration device(s) and drives `nSTATUS` low on all FPGAs, causing them to enter a reset state. This behavior is similar to the FPGA detecting an error in the configuration data.

Figure 8–7 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 8–7. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 8–7:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the `nINIT_CONF/nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

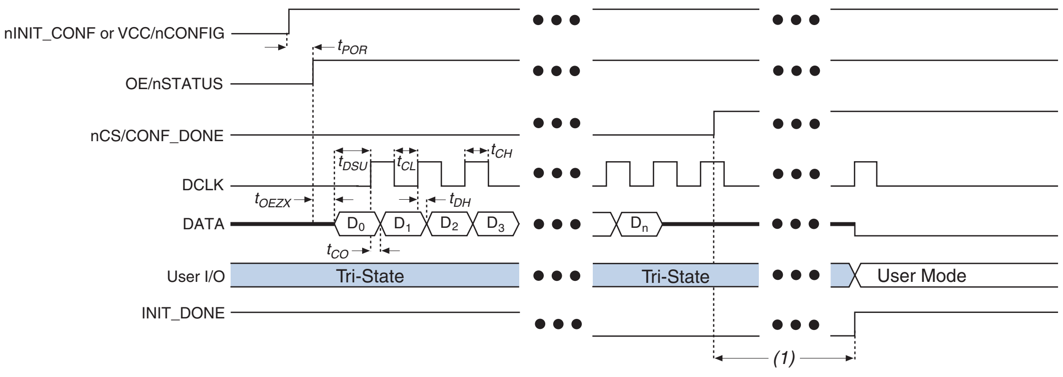
When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`. If multiple EPC2 devices are used to configure a Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K device(s), only the first EPC2 has its `nINIT_CONF` pin tied to the device's `nCONFIG` pin.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, refer to the *Configuring Mixed Altera FPGA Chains*, chapter 8 in volume 2 of the *Configuration Handbook*.

Figure 8–8 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 8–8. PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 8–8:

(1) Mercury devices enter user-mode 136 clock cycles after `CONF_DONE` goes high. APEX 20K devices enter user-mode 40 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the Mercury or APEX 20K internal oscillator or the `CLKUSR` pin. ACEX 1K and FLEX 10K devices enter user-mode 10 clock cycles after CONF_DONE goes high. The initialization clock can come from `DCLK` or `CLKUSR`.

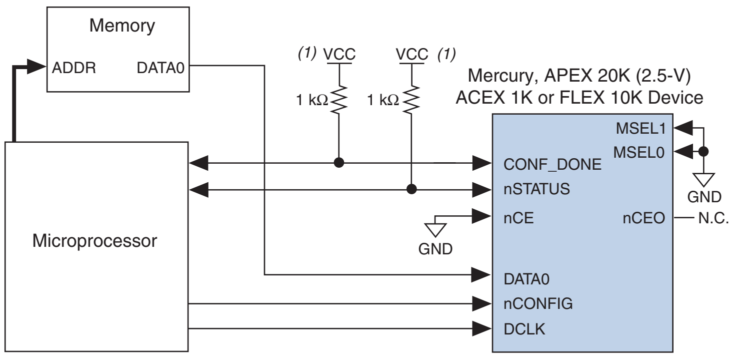
For timing information, refer to the *Altera Configuration Devices*, chapter 1 the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet*, chapter 2, or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* chapter 5 in volume 2 of the *Configuration Handbook*.

Device configuration options and how to create configuration files are discussed further in *Software Settings*, chapters 6 and 7 in volume 2 of the *Configuration Handbook*.

PS Configuration Using a Microprocessor

In the PS configuration scheme, an intelligent host (e.g., a microprocessor or CPLD) can transfer configuration data from a storage device (e.g., flash memory) to the target Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. Configuration data can be stored in RBF, HEX, or TTF format. Figure 8–9 shows the configuration interface connections between the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device and a microprocessor for single device configuration.

Figure 8–9. Single Device PS Configuration Using a Microprocessor



Note to Figure 8–9:

(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of three stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

☞ VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one bit at a time on the DATA0 pin. The least significant bit (LSB) of each data byte must be sent first.

The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK. Data is continuously clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device allows enough clock cycles for proper initialization.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, the clock on DCLK is the clock source for initialization. Programming files generated by the Quartus II or MAX+PLUS II software already have these initialization clock cycles included in the file. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR does not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin will go low. When initialization is complete, the INIT_DONE pin will be released and pulled high. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. To ensure DCLK and DATA are not left floating at the end of configuration, the microprocessor must drive them either high or low, whichever is convenient on your board.

Handshaking signals are not used in PS configuration mode. Therefore, the configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

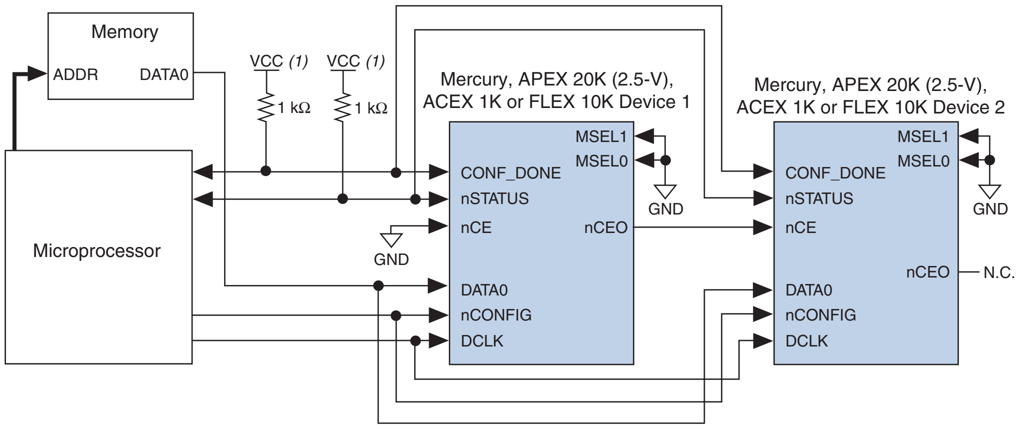
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, you can initiate a reconfiguration by transitioning the `nCONFIG` pin low-to-high. The `nCONFIG` pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices and 2 µs for ACEX 1K and Flex 10K devices. When `nCONFIG` is pulled low, the FPGA also pulls `nSTATUS` and `CONF_DONE` low and all I/O pins are tri-stated. Once `nCONFIG` returns to a logic high state and `nSTATUS` is released by the FPGA, reconfiguration begins.

Figure 8–10 shows how to configure multiple devices using a microprocessor. This circuit is similar to the PS configuration circuit for a single device, except Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are cascaded for multi-device configuration.

Figure 8–10. Multi-Device PS Configuration Using a Microprocessor



Note to Figure 8–10:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

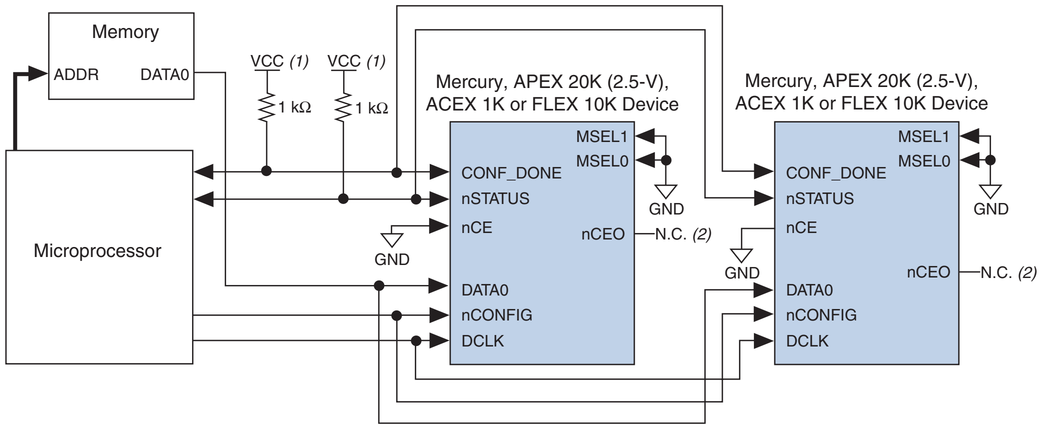
In multi-device PS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DCLK` and `DATA` lines are buffered for every fourth device. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on `nSTATUS`, it resets the chain by pulling its `nSTATUS` pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their `nSTATUS` pins after a reset time-out period (maximum of 40 µs). After all `nSTATUS` pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse `nCONFIG` low. If this option is turned off, the microprocessor must generate a low-to-high transition on `nCONFIG` to restart the configuration process.

In your system, you can have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DCLK` and `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–11 shows multi-device PS configuration when Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are receiving the same configuration data.

Figure 8–11. Multiple-Device PS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–11:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

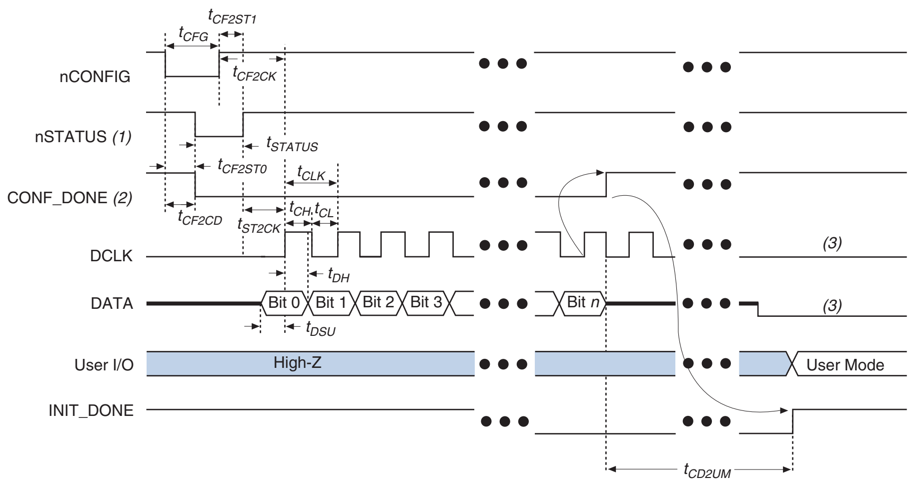
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, refer to *Configuring Mixed Altera FPGA Chains* chapter 8 in volume 2 of the *Configuration Handbook*.

Figure 8–12 shows the timing waveform for the PS configuration for Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices when using a microprocessor.

Figure 8–12. PS Configuration Using a Microprocessor Timing Waveform



Notes to Figure 8–12:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

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

(3) `DATA0` and `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Tables 8–7 through 8–10 defines the timing parameters for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices for PS configuration.

Table 8–7. PS Timing Parameters for Mercury Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	21		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(2)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(2)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	45		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns

Table 8–7. PS Timing Parameters for Mercury Devices (Part 2 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CH}	`DCLK` high time	10		ns
t_{CL}	`DCLK` low time	10		ns
t_{CLK}	`DCLK` period	20		ns
f_{MAX}	`DCLK` frequency		50	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	6	28	µs

Notes to Table 8–7:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–8. PS Timing Parameters for APEX 20K Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	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`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` maximum frequency		33.3	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 8–8:
(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 to obtain this value.

Table 8–9. PS Timing Parameters for ACEX & FLEX 10KE Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` maximum frequency		33.3	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	0.6	2	µs

Notes to Table 8–9:

(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.

(2) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Table 8–10. PS Timing Parameters for FLEX 10K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	30		ns
t_{CL}	`DCLK` low time	30		ns
t_{CLK}	`DCLK` period	60		ns

Table 8–10. PS Timing Parameters for FLEX 10K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
f_{MAX}	`DCLK` maximum frequency		16.7	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	0.6	2	µs

Notes to Table 8–10:

(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.

(2) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Software Settings,* chapters 6 and 7 in Volume 2 of the *Configuration Handbook*.

Configuring Using the MicroBlaster Driver

The MicroBlaster™ software driver allows you to configure Altera's FPGAs through the ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a RBF programming input file and is targeted for embedded passive serial 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, go to the Altera web site (**http://www.altera.com**).

PS Configuration Using a Download Cable

In this section, the generic term "download cable" includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable.

In PS configuration with a download cable, an intelligent host (e.g., a PC) transfers data from a storage device to the FPGA via the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

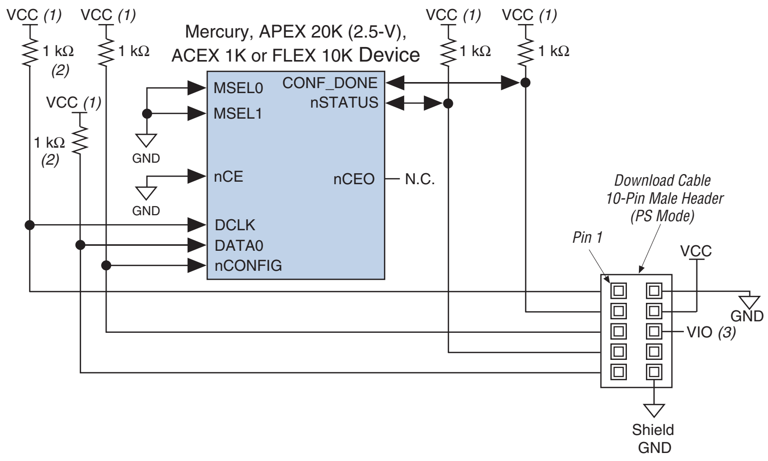
The configuration cycle consists of 3 stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. The programming hardware or download cable then places the configuration data one bit at a time on the device's `DATA0` pin. The configuration data is clocked into the target device until `CONF_DONE` goes high.

When using a download cable, setting the *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs. Additionally, the *Enable user-supplied start-up clock (CLKUSR)* option has no affect on the device initialization since this option is disabled in the SOF when programming the FPGA using the Quartus II programmer and download cable. Therefore, if you turn on the *CLKUSR* option, you do not need to provide a clock on `CLKUSR` when you are configuring the FPGA with the Quartus II programmer and a download cable. Figure 8–13 shows PS configuration for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cable.

Figure 8–13. PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 8–13:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) 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.

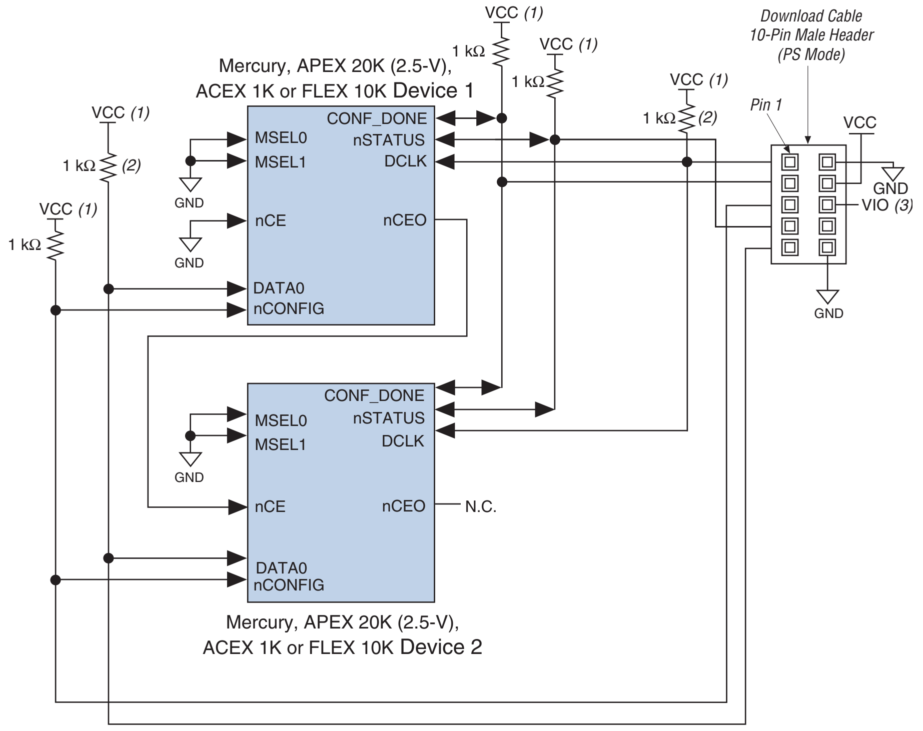
(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

You can use a download cable to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. The first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to the `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. All other configuration pins, `nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE` are connected to every 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. The *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs.

Figure 8–14 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices with a download cable.

Figure 8–14. Multi-Device PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 8–14:

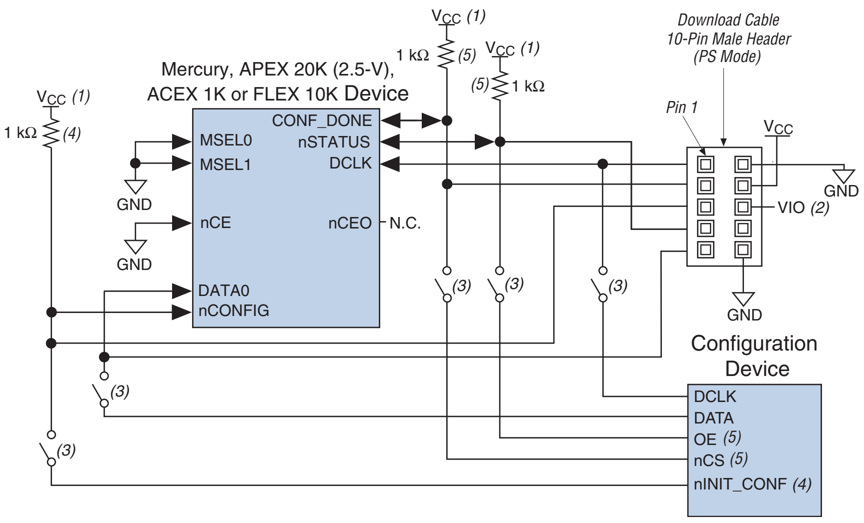
(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) 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.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

If you are using a download cable to configure device(s) on a board that also has configuration devices, you should electrically isolate the configuration device from the target device(s) and cable. One way to isolate the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer chip should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) between the cable and the configuration device. The last option is to remove the configuration device from the board when configuring the FPGA with the cable. Figure 8–15 shows a combination of a configuration device and a download cable to configure an FPGA.

Figure 8–15. PS Configuration with a Download Cable & Configuration Device Circuit



Notes to Figure 8–15:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to a Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K device. 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.

(4) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active. This means an external pull-up resistor is not required on the nINIT_CONF/nCONFIG line. The nINIT_CONF pin does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(5) The enhanced configuration devices' and EPC2 devices' OE and nCS pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-up resistors on configuration device* option when generating programming files.

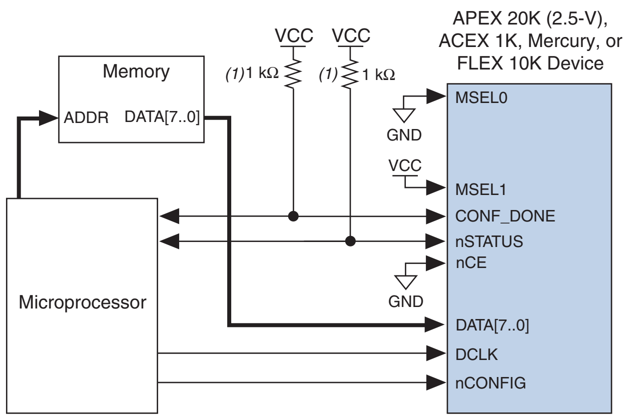
For more information on how to use the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cables, refer to the following data sheets:

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

Passive Parallel Synchronous Configuration

Passive parallel synchronous (PPS) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the serializing clock to the FPGA. The target device latches the byte-wide data on the `DATA[7..0]` pins on the rising edge of `DCLK` and then uses the next eight falling edges on `DCLK` to serialize the data internally. On the ninth rising `DCLK` edge, the next byte of configuration data is latched into the target device. Figure 8–16 shows the configuration interface connections between the FPGA and a microprocessor for single device configuration.

Figure 8–16. Single Device PPS Configuration Using a Microprocessor



Note to Figure 8–16:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 0K (2.5 V), ACEX 1K, and FLEX 10K devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of 3 stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the nCONFIG pin.

☞ VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one byte at a time on the DATA[7..0] pins. New configuration data should be sent to the FPGA every eight DCLK cycles.

The Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K device receives configuration data on its DATA[7..0] pins and the clock is received on the DCLK pin. On the first rising DCLK edge, a byte of configuration data is latched into the target device; the subsequent eight falling DCLK edges serialize the configuration data in the device. On the ninth rising clock edge, the next byte of configuration data is latched and serialized into the target device.

Data is clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K devices, the initialization process is synchronous and can be clocked by its internal oscillator (typically 10 MHz) or by the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

In ACEX 1K and FLEX 10K devices, initialization can be clocked by the clock input on the DCLK pin or by the optional CLKUSR pin. By default, the clock on DCLK is the clock source for initialization. The configuration files created by the Quartus II and the MAX+PLUS II software incorporate the extra bits for proper device initialization. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly. APEX 20K devices require 40 clock cycles, while ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin will go low. When initialization is complete, the INIT_DONE pin will be released and pulled high. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DCLK and DATA0 are not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. The DATA[7..1] pins are available as user I/O pins after configuration. When the PPS scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

The configuration clock (DCLK) speed must be below the specified frequency, as listed in Tables 8–12 through 8–14, to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time. An optional status pin (RDYnBSY) on the FPGA indicates when it is busy serializing configuration data and when it is ready to accept the next data byte. The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration on Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

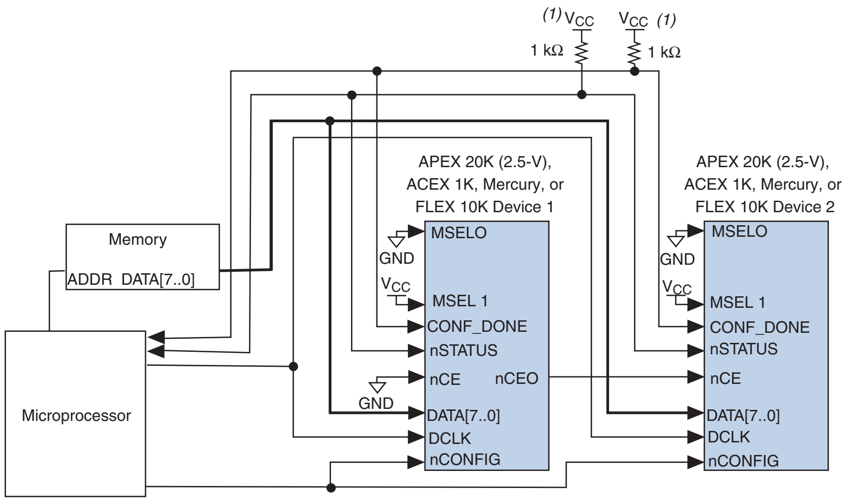
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices and 2 µs for ACEX 1K and FLEX 10K devices. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 8–17 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K devices using a microprocessor. This circuit is similar to the PPS configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 8–17. Multi-Device PPS Configuration Using a Microprocessor



Note to Figure 8–17:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Altera recommends keeping the configuration data valid on the `DATA[7..0]` bus for the 8 serializing clock cycles. The configuration data should be held valid on the `DATA` bus for the complete byte period because the `nCEO` of the first (and preceding) device can go low during the serializing `DCLK` cycles. Once the `nCEO` of the first (and preceding) device goes low, the second (and next) device becomes active and will begin trying to accept configuration data. If the configuration data is not valid on the first `DCLK` edge after `nCEO` goes low, then the second device will see incorrect configuration data and will never begin accepting configuration data. This situation will only arise if you are sharing the `DATA[7..0]` bus with other system data such that the configuration data is only valid for a portion of the byte period.

If your system requires to bus-share the DATA[7..0] line, you can work-around this by ensuring that the second (or next) device sees correct configuration data on the first rising edge of DCLK after the nCEO signal goes low. This can be achieved by delaying the nCEO signal by using external registers or by presenting the next byte of configuration data after the nCEO transition.

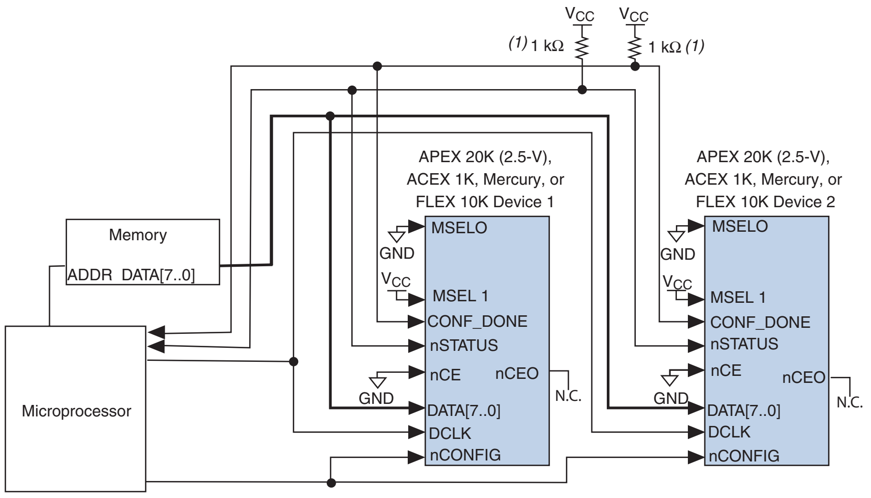
All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration on Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–18 shows multi-device PPS configuration when both devices are receiving the same configuration data.

Figure 8–18. Multiple-Device PPS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–18:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

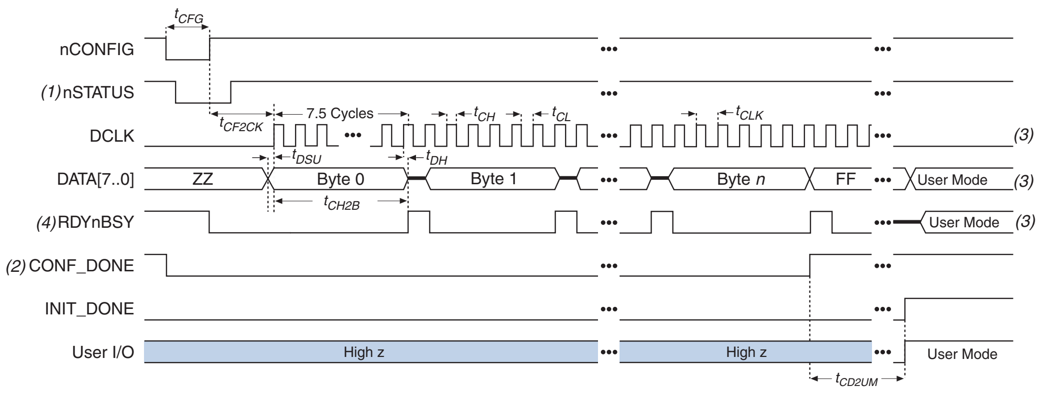
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10KE devices with other Altera devices that support PPS configuration, such as APEX 20KE devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, refer to *Configuring Mixed Altera FPGA Chains* chapter 8 in volume 2 of the *Configuration Handbook*.

Figure 8–19 shows the timing waveform for the PPS configuration scheme using a microprocessor.

Figure 8–19. Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10KE PPS Configuration Timing Waveform



Notes to Figure 8–19:

(1) Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE device holds `nSTATUS` low for approximately 5 µs after V_{CC} reaches its minimum requirement.

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

(3) `DATA0` and `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]` and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the design programmed into the device.

(4) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

Tables 8–12 through 8–14 define the timing parameters for Mercury, APEX 20K, ACEX 1K, and FLEX 10K devices for PPS configuration.

Table 8–11. PPS Timing Parameters for Mercury Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	21		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	45		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs

Table 8–11. PPS Timing Parameters for Mercury Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CH}	`DCLK` high time	10		ns
t_{CL}	`DCLK` low time	10		ns
t_{CLK}	`DCLK` period	20		ns
f_{MAX}	`DCLK` frequency		50	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	6	28	µs

Notes to Table 8–11:

(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.

(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.

(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–12. PPS Timing Parameters for APEX 20K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	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`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` frequency		33.3	MHz

Table 8–12. PPS Timing Parameters for APEX 20K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CD2UM}	CONF_DONE high to user mode *(4)*	2	8	µs

Notes to Tables 8–12:

(1) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.
(2) The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.
(3) This parameter depends on the DCLK frequency. The RDYnBSY signal goes high 7.5 clock cycles after the rising edge of DCLK. This value was calculated with a DCLK frequency of 10 MHz.
(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 40 to obtain this value.

Table 8–13. PPS Timing Parameters for ACEX 1K & FLEX 10KE Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		4 *(1)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	5		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH2B}	First rising DCLK to first rising RDYnBSY *(2)*	0.75 *(3)*		µs
t_{CH}	DCLK high time	15		ns
t_{CL}	DCLK low time	15		ns
t_{CLK}	DCLK period	30		ns
f_{MAX}	DCLK frequency		33.3	MHz
t_{CD2UM}	CONF_DONE high to user mode *(4)*	0.6	2	µs

Notes to Table 8–13:

(1) This value is obtainable if users do not delay configuration by extending the nSTATUS low pulse width.
(2) The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.
(3) This parameter depends on the DCLK frequency. The RDYnBSY signal goes high 7.5 clock cycles after the rising edge of DCLK. This value was calculated with a DCLK frequency of 10 MHz.
(4) The minimum and maximum numbers apply only if DCLK is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 10 to obtain this value.

Table 8–14. PPS Timing Parameters for FLEX 10K Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs
t_{CH}	`DCLK` high time	30		ns
t_{CL}	`DCLK` low time	30		ns
t_{CLK}	`DCLK` period	60		ns
f_{MAX}	`DCLK` frequency		16.7	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	0.6	2	µs

Notes to Table 8–14:
(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.
(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.
(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.
(4) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

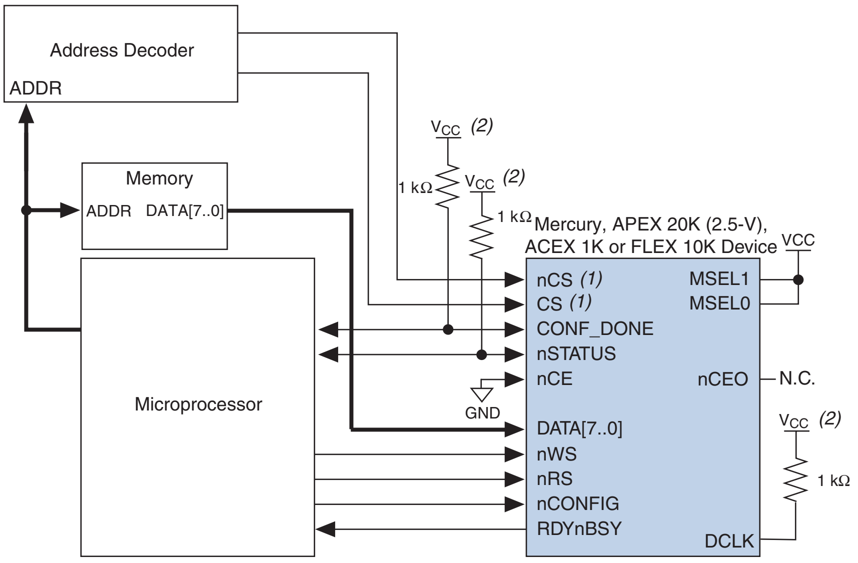
Device configuration options and how to create configuration files are discussed further in *Software Settings*, chapters 6 and 7 in volume 2 of the *Configuration Handbook*.

Passive Parallel Asynchronous Configuration

Passive Parallel Asynchronous (PPA) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the accompanying strobe signals to the FPGA. When using PPA, you should pull the `DCLK` pin high through a 1-kΩ pull-up resistor to prevent unused configuration input pins from floating.

Figure 8–20 shows the configuration interface connections between the FPGA and a microprocessor for single device PPA configuration. The microprocessor or an optional address decoder can control the device's chip select pins, nCS and CS. The address decoder allows the microprocessor to select the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device by accessing a particular address, which simplifies the configuration process. The nCS and CS pins must be held active during configuration and initialization.

Figure 8–20. Single Device PPA Configuration Using a Microprocessor *Note (1)*



Notes to Figure 8–20:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

During PPA configuration, it is only required to use either the nCS or CS pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, nCS can be tied to ground while CS is toggled to control configuration. The device's nCS or CS pins can be toggled during PPA configuration if the design meets the specifications set for t_{CSSU}, t_{WSP}, and t_{CSH} listed in Tables 8–15 through 8–17.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of 3 stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should then assert the target device's `nCS` pin low and/or `CS` pin high. Next, the microprocessor places an 8-bit configuration word (one byte) on the target device's `DATA[7..0]` pins and pulses the `nWS` pin low.

On the rising edge of `nWS`, the target device latches in a byte of configuration data and drives its `RDYnBSY` signal low, which indicates it is processing the byte of configuration data. The microprocessor can then perform other system functions while the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device is processing the byte of configuration data.

During the time `RDYnBSY` is low, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device internally processes the configuration data using its internal oscillator (typically 10 MHz). When the device is ready for the next byte of configuration data, it will drive `RDYnBSY` high. If the microprocessor senses a high signal when it polls `RDYnBSY`, the microprocessor sends the next byte of configuration data to the FPGA.

Alternatively, the `nRS` signal can be strobed low, causing the `RDYnBSY` signal to appear on `DATA7`. Because `RDYnBSY` does not need to be monitored, this pin doesn't need to be connected to the microprocessor. Data should not be driven onto the data bus while `nRS` is low because it will cause contention on the `DATA7` pin. If the `nRS` pin is not used to monitor configuration, it should be tied high.

To simplify configuration and save an I/O port, the microprocessor can wait for the total time of t_{BUSY}(max) + t_{RDY2WS} + t_{W2SB} before sending the next data byte. In this set-up, nRS should be tied high and RDYnBSY does not need to be connected to the microprocessor. The t_{BUSY}, t_{RDY2WS} and t_{W2SB} timing specifications are listed in Tables 8–15 through 8–17.

Next, the microprocessor checks nSTATUS and CONF_DONE. If nSTATUS is not low and CONF_DONE is not high, the microprocessor sends the next data byte. However, if nSTATUS is not low and all the configuration data has been received, the device is ready for initialization. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device allows enough clock cycles for proper initialization. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, in PPA, the device uses its internal oscillator (typically 10MHz) to clock the initialization cycle. The ACEX 1K or FLEX 10K device will take care to provide itself with enough clock cycles for proper initialization.

You also have the flexibility to synchronize initialization of multiple devices by using the *CLKUSR* option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin will go low. When initialization is complete, the INIT_DONE pin will be released and pulled high. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DATA0 is not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. After configuration, the nCS, CS, nRS, nWS, RDYnBSY, and DATA[7..1] pins can be used as user I/O pins. When the PPA scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

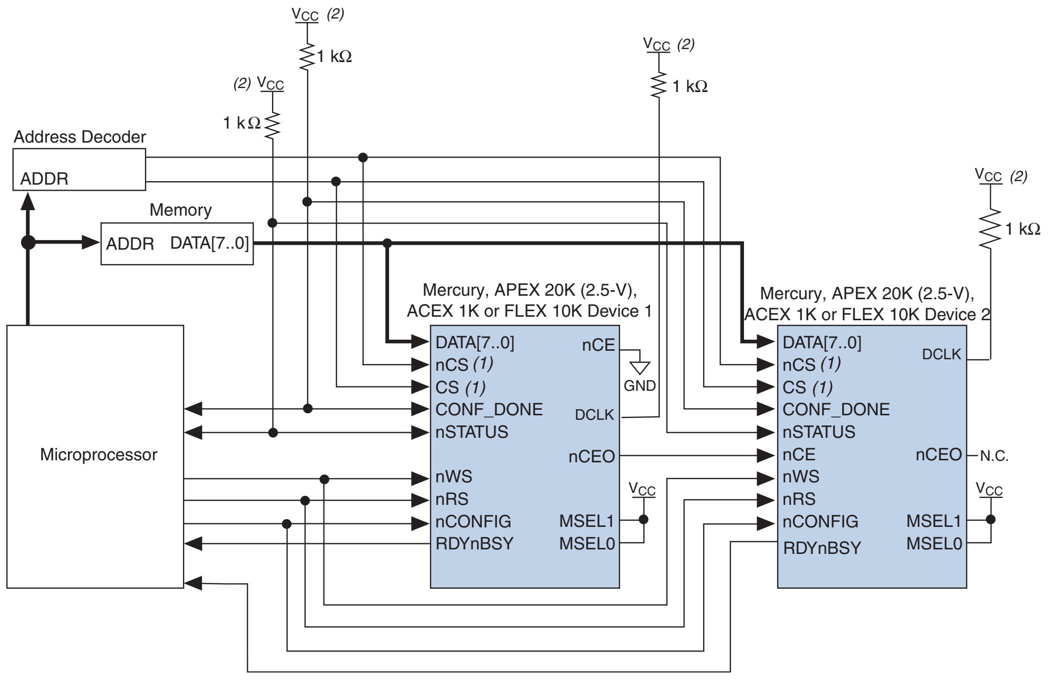
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE has not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices, and 2 µs for ACEX 1K and FLEX 10K devices. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 8–21 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a microprocessor. This circuit is similar to the PPA configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 8–21. Multi-Device PPA Configuration Using a Microprocessor



Notes to Figure 8–21:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPA configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Each device's `RDYnBSY` pin can have a separate input to the microprocessor. Alternatively, if the microprocessor is pin limited, all the `RDYnBSY` pins can feed an `AND` gate and the output of the `AND` gate can feed the microprocessor. For example, if you have 2 devices in a PPA configuration chain, the second device's `RDYnBSY` pin will be high during the time that the first device is being configured. When the first device has been successfully configured, it will driven `nCEO` low to activate the next device in the chain and drive its `RDYnBSY` pin high. Therefore, since `RDYnBSY` signal is driven high before configuration and after configuration before entering user-mode, the device being configured will govern the output of the `AND` gate.

The `nRS` signal can be used in multi-device PPA chain since the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device will tri-state its `DATA[7..0]` pins before configuration and after configuration before entering user-mode to avoid contention. Therefore, only the device that is currently being configured will respond to the `nRS` strobe by asserting `DATA7`.

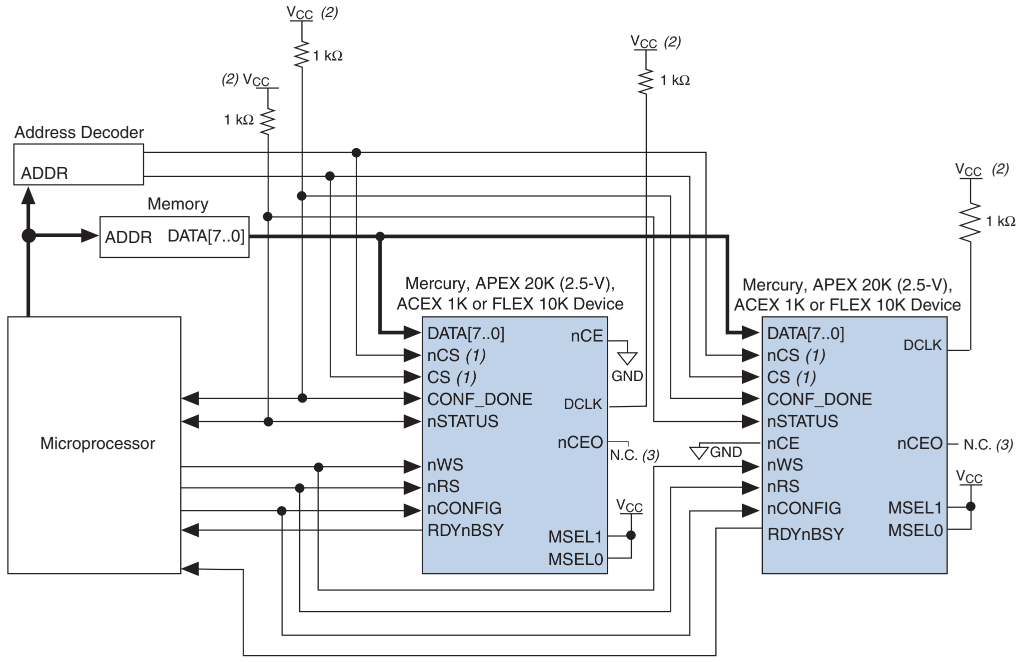
All other configuration pins (`nCONFIG`, `nSTATUS`, `DATA[7..0]`, `nCS`, `CS`, `nWS`, `nRS` and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DATA` lines are buffered for every fourth device. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on `nSTATUS`, it resets the chain by pulling its `nSTATUS` pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their `nSTATUS` pins after a reset time-out period (maximum of 40 µs). After all `nSTATUS` pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse `nCONFIG` low. If this option is turned off, the microprocessor must generate a low-to-high transition on `nCONFIG` to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DATA[7..1]`, `nCS`, `CS`, `nWS`, `nRS` and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–22 shows multi-device PPA configuration when both devices are receiving the same configuration data.

Figure 8–22. Multiple-Device PPA Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–22:

(1) If not used, the `CS` pin can be connected to V_{CC} directly. If not used, the `nCS` pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

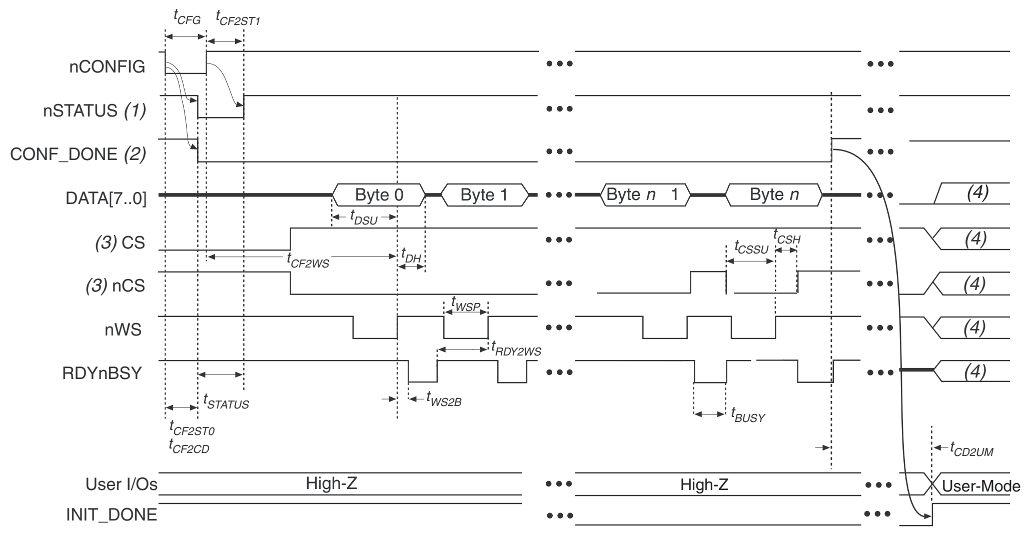
(3) The `nCEO` pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX, and FLEX 10KE devices with other Altera devices that support PPA configuration, such as Stratix® or APEX II devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, refer to *Configuring Mixed Altera FPGA Chains* chapter 8 of volume 2 in the *Configuration Handbook*.

Figure 8–23 shows the timing waveform for the PPA configuration scheme using a microprocessor.

Figure 8–23. PPA Configuration Timing Waveform



Notes to Figure 8–23:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

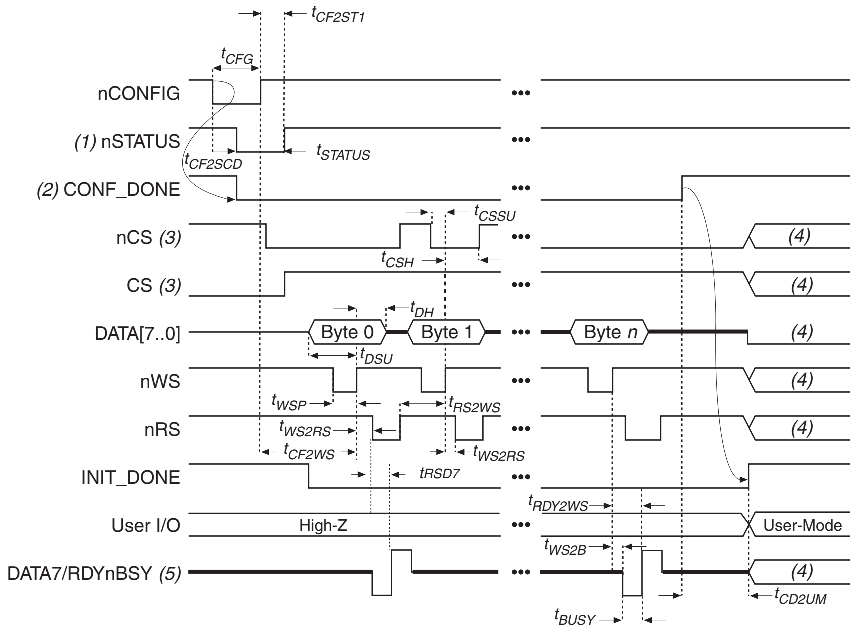
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS` and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

Figure 8–24 shows the timing waveform for the PPA configuration scheme when using a strobed `nRS` and `nWS` signal.

Figure 8–24. PPA Configuration Timing Waveform Using nRS & nWS



Notes to Figure 8–24:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

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

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS`, and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

(5) `DATA7` is a bidirectional pin. It is an input for configuration data input, but it is an output to show the status of `RDYnBSY`.

Tables 8–15 through 8–17 define the timing parameters for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices for PPA configuration.

Table 8–15. PPA Timing Parameters for Mercury Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	21		µs

Table 8–15. PPA Timing Parameters for Mercury Devices (Part 2 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(2)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(2)*	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.4	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	6	28	µs

Notes to Table 8–15:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–16. PPA Timing Parameters for APEX 20K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns

Table 8–16. PPA Timing Parameters for APEX 20K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.1	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 8–16:
(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 to obtain this value.

Table 8–17. PPA Timing Parameters for ACEX 1K & FLEX 10K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	20		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	10 *(2)* 15 *(3)*		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	5		µs
t_{DSU}	Data setup time before rising edge on `nWS`	20		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.4	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns

Table 8–17. PPA Timing Parameters for ACEX 1K & FLEX 10K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	0.6	2	µs

Notes to Table 8–17:
(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) This parameter value applies to EPF10K10, EPF10K20, EPF10K40, EPF10K50, all FLEX 10KA, and FLEX 10KE devices.
(3) This parameter value applies to only EPF10K70 and EPF10K100 devices.
(4) If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Software Settings*, chapter 6 and 7 in volume 2 of the *Configuration Handbook*.

JTAG Configuration

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on 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. The JTAG circuitry can also be used to shift configuration data into the device.

For more information on JTAG boundary-scan testing, refer to *Application Note 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`, and one optional pin, `TRST`. All user I/O pins are tri-stated during JTAG configuration. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are designed such that JTAG instructions have precedence over any device configuration modes. This means that JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K FPGAs during PS configuration, PS configuration will be terminated and JTAG configuration will begin.

Table 8–18 explains each JTAG pin's function.

Table 8–18. 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`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
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. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the 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`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
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. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST *(1)*	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

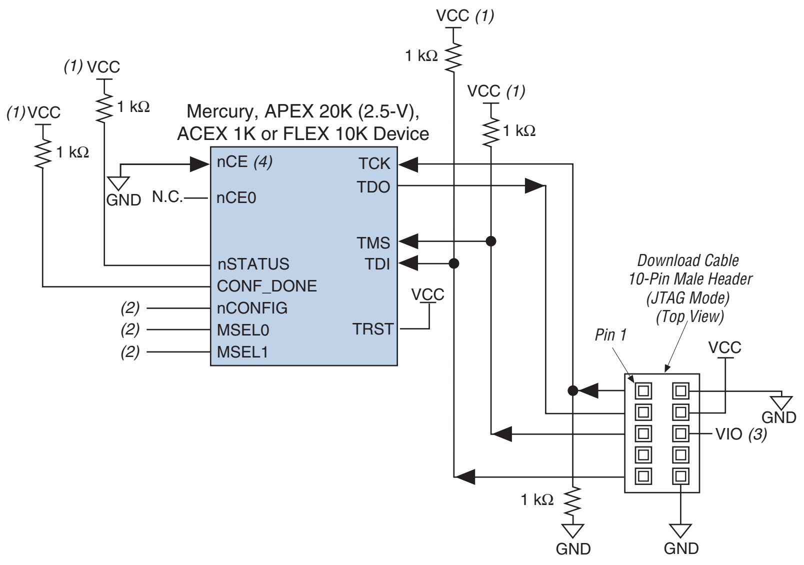
Note to Table 8–18:

(1) FLEX 10K devices in 144-pin thin quad flat pack (TQFP) packages do not have a `TRST` pin. Therefore, the `TRST` pin can be ignored when using these devices.

☞ If V_{CCIO} of the bank where the JTAG pins reside, are tied to 3.3-V, both the I/O pins and JTAG `TDO` port will drive at 3.3-V levels.

During JTAG configuration, data can be downloaded to the device on the PCB through the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV header. Configuring devices through a cable is similar to programming devices in-system, except the `TRST` pin should be connected to V_{CC}. This ensures that the TAP controller is not reset. Figure 8–25. shows JTAG configuration of a single Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device.

Figure 8–25. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 8–25:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

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.

Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices have dedicated JTAG pins that always function as JTAG pins. JTAG testing can be performed on Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices both before and after configuration, but not during configuration. The chip-wide reset (`DEV_CLRn`) and chip-wide output enable (`DEV_OE`) pins on Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices 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 Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices, the dedicated configuration pins should be considered. Table 8–19 shows how these pins should be connected during JTAG configuration.

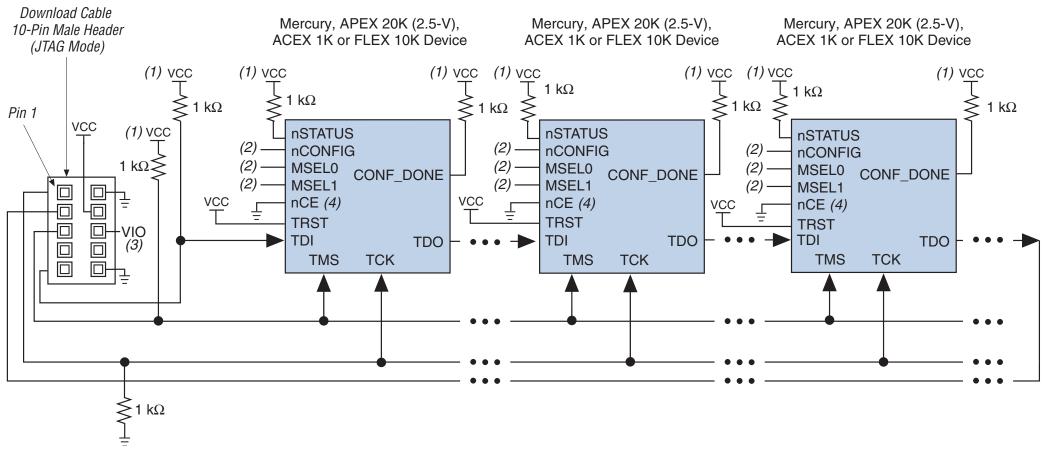
Table 8–19. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices in the chain, `nCE` should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, PPS or PPA configuration chains, the `nCE` pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices in the chain, `nCEO` can be left floating or connected to the `nCE` of the next device. See `nCE` description above.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie both pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling high via a resistor, or driven by some control circuitry.
nSTATUS	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `nSTATUS` pin should be pulled up to V_{CC} individually. `nSTATUS` pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `CONF_DONE` pin should be pulled up to V_{CC} individually. `CONF_DONE` going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.
DATA0	Should not be left floating. Drive low or high, whichever is more convenient on your board.

When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. 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 programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 8–26 shows multi-device JTAG configuration.

Figure 8–26. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 8–26:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to GND or driven low during JTAG configuration. In multi-device PS, PPS, and PPA configuration chains, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device will drive `nCE` of the next device low when it has successfully been JTAG configured.

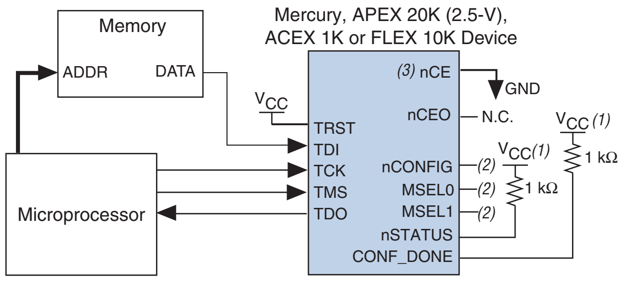
Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

For more information about configuring multiple Altera devices in the same configuration chain, refer to *Configuring Mixed Altera FPGA Chains* chapter 8 in volume 2 of the *Configuration Handbook*.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, 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. When Quartus II generates a JAM file for a multi-device chain, it contains instructions so that all the devices in the chain will be initialized at the same time.

Figure 8–27 shows JTAG configuration of a Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K FPGA with a microprocessor.

Figure 8–27. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 8–27:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

(2) 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.

(3) nCE must be connected to GND or driven low for successful JTAG configuration.

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.

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, refer to *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the jam player, visit the Altera web site at:

www.altera.com/support/software/download/programming/jam/jam-index.jsp

Configuring Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K 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, refer to the *JRunner Software Driver: An Embedded Solution to the JTAG Configuration White Paper* and the source files.

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device. Table 8–20 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 8–20. Dedicated Configuration Pins (Part 1 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`MSEL0` `MSEL1`	N/A	All	Input	Two-bit configuration input that sets the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device configuration scheme. See Table 8–6 for the appropriate connections. These pins must remain at a valid state during power-up, before `nCONFIG` is pulled low to initiate a reconfiguration and during configuration.
`VCCSEL`	N/A	All	Input	This pin is only available in Mercury devices. Dedicated input that ensures the configuration related I/O banks have powered up to the appropriate 1.8-V or 2.5-V/3.3-V voltage levels before starting configuration. A logic high (1.5 V, 1.8 V, 2.5 V, 3.3 V) means 2.5 V/3.3 V, and a logic low means 1.8 V.
`nCONFIG`	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode will cause the FPGA to lose its configuration data, enter a reset state, tri-state all I/O pins, and returning this pin to a logic high level will initiate a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, `nCONFIG` can be tied directly to V_{CC} or to the configuration device's `nINIT_CONF` pin.

Table 8–20. Dedicated Configuration Pins (Part 2 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nSTATUS`	N/A	All	Bidirectional open-drain	The FPGA 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.) Status output. If an error occurs during configuration, `nSTATUS` is pulled low by the target device. Status input. 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. If a configuration device is used, driving `nSTATUS` low will cause the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on `nSTATUS` in user-mode, the FPGA will not reconfigure. To initiate a reconfiguration, `nCONFIG` must be pulled low. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.
`CONF_DONE`	N/A	All	Bidirectional open-drain	Status output. The target FPGA drives the `CONF_DONE` pin low before and during configuration. Once all configuration data is received without error and the initialization 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. Driving `CONF_DONE` low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.
`nCE`	N/A	All	Input	Active-low chip enable. The `nCE` pin activates the device with a low signal to allow configuration. The `nCE` pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, `nCE` of the first device is tied low while its `nCEO` pin is connected to `nCE` of the next device in the chain. The `nCE` pin must also be held low for successful JTAG programming of the FPGA.

Table 8–20. Dedicated Configuration Pins (Part 3 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nCEO`	N/A	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's `nCE` pin. The `nCEO` of the last device in the chain is left floating.
`DCLK`	N/A	Synchronous configuration schemes (PS and PPS)	Input	Clock input used to clock data from an external source into the target device. Data is latched into the FPGA on the rising edge of `DCLK`. In PPA mode, `DCLK` should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, `DCLK` will be driven low after configuration is done. In schemes that use a control host, `DCLK` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
`DATA0`	N/A	All	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the `DATA0` pin. After configuration, EPC1 and EPC1441 devices tri-state this pin, while EPC2 devices drive this pin high. In schemes that use a control host, `DATA0` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
`DATA[7..1]`	I/O	Parallel configuration schemes (PPS and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on `DATA[7..0]`. In serial configuration schemes, they function as user I/O pins during configuration, which means they are tri-stated. After PPA or PPS configuration, `DATA[7..1]` are available as a user I/O pins and the state of these pin depends on the Dual-Purpose Pin settings.
`DATA7`	I/O	PPA	Bidirectional	In the PPA configuration scheme, the `DATA7` pin presents the `RDYnBSY` signal after the `nRS` signal has been strobed low. In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `DATA7` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.

Table 8–20. Dedicated Configuration Pins (Part 4 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nWS`	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the `DATA[7..0]` pins. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nWS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
`nRS`	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the `RDYnBSY` signal on the `DATA7` pin. If the `nRS` pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nRS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
`RDYnBSY`	I/O	PPS and PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPS and PPA configuration schemes, this pin will drive out high after power-up, before configuration and after configuration before entering user-mode. In non-PPS and non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPS and PPA configuration, `RDYnBSY` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
`nCS/CS`	I/O	PPA	Input	Chip-select inputs. A low on `nCS` and a high on `CS` select the target device for configuration. The `nCS` and `CS` pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the `nCS` or `CS` pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, `nCS` can be tied to ground while `CS` is toggled to control configuration. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nCS` and `CS` are available as a user I/O pins and the state of these pins depends on the Dual-Purpose Pin settings.

Table 8–21 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration they function as user I/O pins, which means they are tri-stated with weak pull-up resistors.

Table 8–21. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
`CLKUSR`	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software
`INIT_DONE`	N/A if option is on. I/O if option is off.	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 1-kΩ pull-up. Once the option bit to enable `INIT_DONE` is programmed into the device (during the first frame of configuration data), the `INIT_DONE` pin will go low. When initialization is complete, the `INIT_DONE` pin will be released and pulled high and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the *Enable INIT_DONE output* option in the Quartus II software.
`DEV_OE`	N/A if option is on. I/O if option is off.	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 pin is enabled by turning on the *Enable device-wide output enable (DEV_OE)* option in the Quartus II software.
`DEV_CLRn`	N/A if option is on. I/O if option is off.	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 pin is enabled by turning on the *Enable device-wide reset (DEV_CLRn)* option in the Quartus II software.

JTAG pins must be kept stable before and during configuration. JTAG pin stability prevents accidental loading of JTAG instructions. Table 8–22 describes the dedicated JTAG pins.

Table 8–22. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
TDI	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	N/A	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. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	N/A	Input	Input pin that provides the control signal to determine the transitions of the 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. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The TRST pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.