# Configuring PLDs with Flash Memory

## Introduction

As Altera® introduces higher-density programmable logic devices (PLDs) to the market, the size of the configuration bit streams also increases. By using a standard flash memory device, these high-density devices can be configured. A MAX® 3000 device can be used to stream the data from the flash memory into a large PLD. This configuration technique allows faster configuration times. Since a fixed-frequency oscillator (or any available clock on the system) is used to generate the clock for the configuration, the clock frequency can be as high as 57 MHz (the maximum for an APEX™ 20KE device).

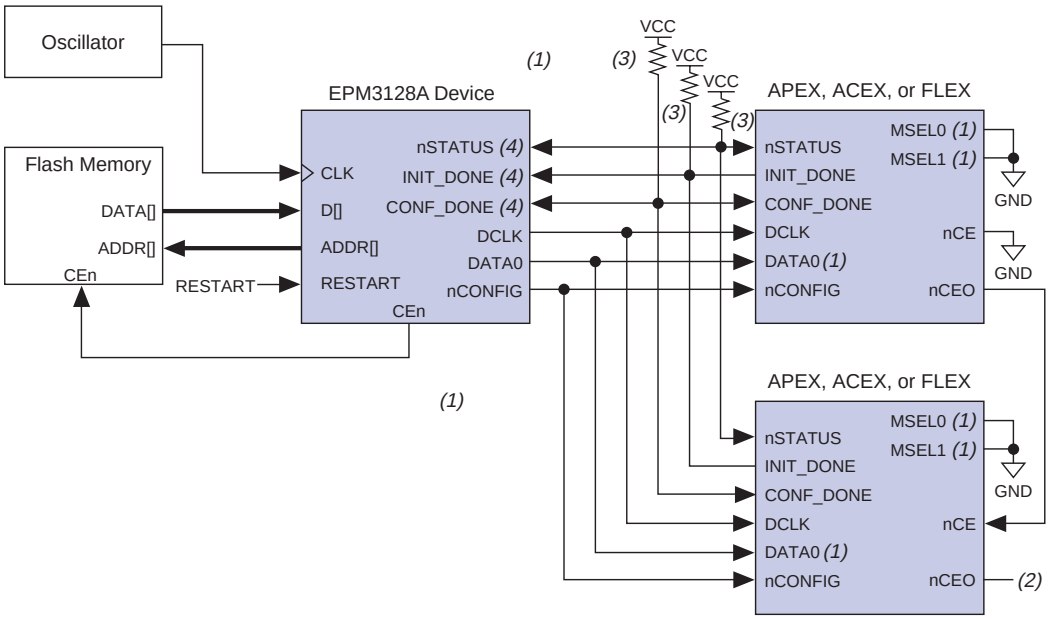
Flash memory is a type of nonvolatile memory that can be used as a data storage device. Flash memory can be erased and reprogrammed in units of memory called blocks.

This white paper describes how to configure a PLD with flash memory. By using a MAX 3000 device to configure higher density PLDs, the flash memory can store configuration data and the MAX 3000 device can serialize and transmit the data to the PLD. This configuration technique can be used with APEX, ACEX™, or FLEX® devices.

## Configuring PLDs

Figure 1 shows a device that uses an EPM3128A device and flash memory to configure the PLDs.

*Figure 1. Device Configuration Using Flash Memory & EPM3128A Device*




***Notes:***

(1) FLEX 6000 devices have a single MSEL pin, which is tied to ground. Additionally, its DATA0 pin is renamed DATA.
(2) The nCEO pin is left unconnected for the last device in the chain.
(3) Pull-up resistors are 1 kΩ except for APEX 20KE devices. For APEX 20KE devices, pull up resistors are 10 kΩ.
(4) The nSTATUS, CONF_DONE, and INIT_DONE pins are open-drain on the APEX, ACEX, and FLEX devices. The corresponding pins on the EPM3128A should also be open_drain.

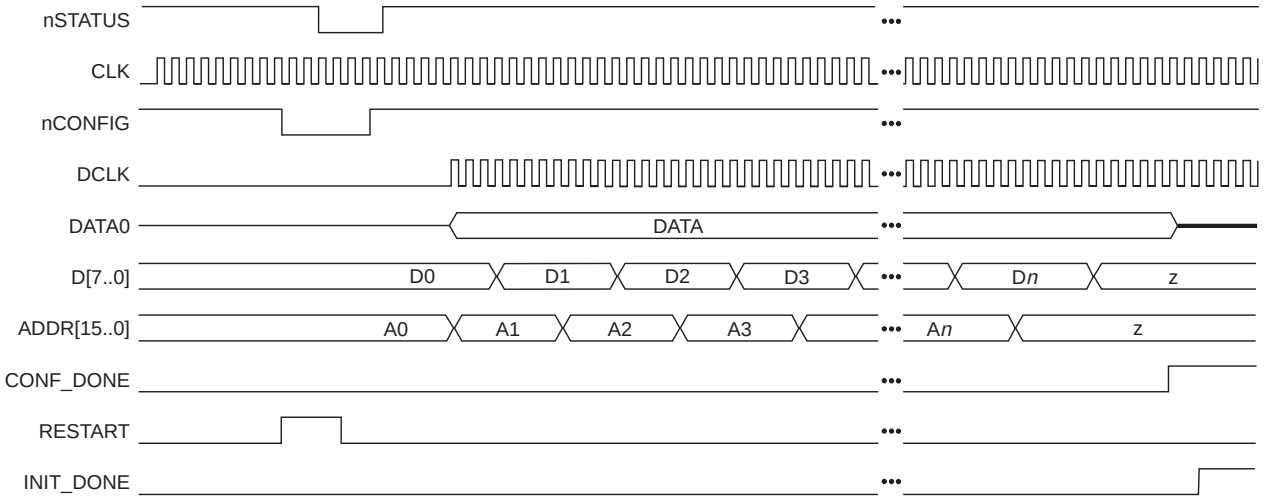
A VHDL design file called **MAXconfig**, shown in the "Configuration Design File" section, allows an EPM3128A device to control the configuration process. The **MAXconfig** design configures the PLD using the configuration data stored in the attached flash memory. The **MAXconfig** design contains a sequencer and an address generator, which drives the correct data to the PLD's programming pins. The MAXconfig design file is available on the Altera web site at http://www.altera.com/document/wp/maxconfig.txt.

When the **MAXconfig** design is reset, the **MAXconfig** design reads the data from the flash memory, one byte at a time. The **MAXconfig** design then serializes and sends the data to the APEX, ACEX, or FLEX device. The serialized data is sent to the PLD using the passive serial interface pins such as `DCLK`, `DATA`, `nSTATUS`, `INIT_DONE`, and `nCONFIG`. Since the passive serial mode is used, the flash pins are not directly connected to the APEX, ACEX, or FLEX device.

Flash memory can be programmed prior to being put onto a board with standard programming equipment or it can be programmed in-system by a processor or test equipment. Since different flash memories have different algorithms, consult the flash memory data sheet for programming information.

Figure 2 shows a configuration timing waveform of an EPM3128A device downloading data to an APEX, ACEX, or FLEX device.

*Figure 2. Configuration Timing Waveform*




## Configuration Design File

This section shows the **MAXconfig** design file that controls the configuration process on APEX, ACEX, or FLEX devices:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MAXconfig is
port
       (
       clock          : in          std_logic;
       init_done      : in          std_logic;
       nStatus        : in          std_logic;
       D              : in          std_logic_vector(7 downto 0);
       restart        : in          std_logic;
       Conf_Done      : in          std_logic;

       Data0          : out         std_logic;
       Dclk           : out         std_logic;
       nConfig        : buffer      std_logic;
```

```
--To increase the size of the memory, change the size of std_logic_vector for ADDR output and
--std_logic_vector signal inc:

       ADDR          : out         std_logic_vector(15 downto 0);
       CEn           : out         std_logic);
-- The polarity of the CEn signal is determined by the type of Flash device
end;

architecture rtl of MAXconfig is

--The following encoding is done in such way that the LSB represents the nConfig signal:

constant start                              :std_logic_vector(2 downto 0) := "000";
constant wait_nCfg_8us                      :std_logic_vector(2 downto 0) := "100";
constant status                             :std_logic_vector(2 downto 0) := "001";
constant wait_40us                          :std_logic_vector(2 downto 0) := "101";
constant config                             :std_logic_vector(2 downto 0) := "011";
constant init                               :std_logic_vector(2 downto 0) := "111";


signal pp                                   :std_logic_vector(2 downto 0);
signal count                                :std_logic_vector(2 downto 0);
signal data0_int, dclk_int                  :std_logic;
signal inc                                  :std_logic_vector(15 downto 0);
signal div                                  :std_logic_vector(2 downto 0);
signal waitd                                :std_logic_vector(11 downto 0);
--The width of signal ‘waitd’ is determined by the frequency. For 57 MHz (APEX 20KE devices),
--‘waitd’ is 12 bits. For 33 MHz (FLEX 10KE and ACEX devices) ‘waitd’ is 11 bits. To calculate
--the width of the ‘waitd’ signal fordifferent frequencies, calculate the following:
--(multiply tcf2ck * clock frequency)+ 40
--Then convert this value to binary to obtain the width.
--For example, for 33 MHz (FLEX 10KE & ACEX devices), converting 1360 ((40us * 33MHz)+40=1360)
--to binary code, the ‘waitd’ is an 11-bit signal. So signal ‘waitd’ will be:
--signal waitd                              :std_logic_vector(10 downto 0);

begin
--The following process is used to divide the CLOCK:
       PROCESS (clock,restart)
       begin
              if restart = '0' then
                     div <= (others => '0');
              else
                     IF (clock'EVENT AND clock = '1') THEN
                            div <= div + 1;
                     end if;
              end if;
       END PROCESS;

       PROCESS (clock,restart)
       begin
              if restart = '0' then
                     pp<=start;
                     count <= (others => '0');
                     inc   <= (others => '0');
                     waitd <= (others => '0');
              else
              if clock'event and clock='1' then

--The following test is used to divide the CLOCK. The value compared to must be such that the
--condition is true at a maximum rate of 57 MHz (tclk = 17.5 ns min) for APEX 20KE devices
--and at a maximum rate of 33 MHz (tclk=30ns min) for FLEX 10KE or ACEX devices.

                     if (div = 7) then
                            case pp is
                            when start =>
                                     count <= (others => '0');
                                     inc   <= (others => '0');
                                     waitd <= (others => '0');
```

```
                                        pp <= wait_nCfg_8us;
--This state is used in order to verify the tcfg timing (nCONFIG low pulse width).
--Tcfg = 8µs => min= 456 clock cycle of a 57 MHz clock (APEX 20KE devices). For different
--clocks, multiply 8µs to clock frequency. For example, for 33MHz (FLEX 10KE or ACEX devices)
this --value is 8*33=264. This clock is CLOCK divided by the divider -div-.
                                when wait_nCfg_8us =>
                                        count <= (others => '0');
                                         inc   <= (others => '0');
                                         waitd <= waitd + 1;
                                         if waitd = 456 then
--For 33 MHz FLEX 10KE or ACEX devices this line is: if waitd = 264 then

                                             pp <= status;
                                        end if;

--This state is used to have nCONFIG high.
                                when status =>
                                       count <= (others => '0');
                                       inc   <= (others => '0');
                                       waitd <= (others => '0');
                                       pp <= wait_40us;

--This state is used to generate the tcf2ck timing (nCONFIG high to first rising edge on DCLK).
--Tcf2ck = 40µs min => 2280 clock cycles of a 57MHz (APEX 20KE) clock. This clock is CLOCK
--divide by the divider -div-
--Tcf2ck = 40µs min => 1320 clock cycles of a 33MHz (FLEX 10KE/ACEX) clock. This clock is CLOCK
--divided by the divider -div-)
--For any other clock frequency, multiply tcf2ck * clock frequency.

                                when wait_40us =>
                                           count <= (others => '0');
                                           inc   <= (others => '0');
                                           waitd <= waitd + 1;
                                          if waitd = 2280 then
--For 33 MHz (FLEX 10KE or ACEX devices), this line is:    if waitd = 1320 then

                                                  pp <= config;
                                       end if;

--This state is used to increment the memory address. In the same state when
--the Conf_Done is high clock cycles are added in order to have the initialization completed.

                                when config =>
                                       count <= count + 1;
                                       if Conf_Done='1' then
                                              waitd <= waitd + 1;
                                       end if;
                                       if count=7 then
                                              inc <= inc + 1;
                                       end if;
                                       if waitd = 2320 then
--Modification: Add 40 clock cycles. For APEX 20KE devices, it is 2280+40=2320
--For FLEX 10KE and ACEX devices, it is 1320+40=1360. This line becomes:    if waitd= 1360 then

                                       pp<= init;
                                       end if;

                                when init =>
                                       count <= (others => '0');
                                       inc   <= (others => '0');
                                       waitd <= (others => '0');
                                       if nStatus = '0' then
                                              pp <= start;
                                       else
                                              pp <= init;
                                       end if;

                                when others =>
                                       pp <= start;
```

```
                        end case;
                    else
                        pp <= pp;
                        inc <= inc;
                        count <= count;
                    end if;
            end if;
        end if;
        end PROCESS;

        dclk_int <= div(2) when pp=config else '0';

--The following process is used to serialize the data byte :
        PROCESS (count,D,pp)
        begin
            if pp=config then
                        case count is
                        when "000" => data0_int <= D(0);
                        when "001" => data0_int <= D(1);
                        when "010" => data0_int <= D(2);
                        when "011" => data0_int <= D(3);
                        when "100" => data0_int <= D(4);
                        when "101" => data0_int <= D(5);
                        when "110" => data0_int <= D(6);
                        when "111" => data0_int <= D(7);
                        when others => null;
                        end case;
            else
                        data0_int <= '0';
            end if;
        end PROCESS;

        nConfig <= pp(0);
        CEn <= not nconfig;
        Dclk  <= '0' when pp(1)='0' else dclk_int;
        Data0 <= '0' when pp(1)='0' else data0_int;
        ADDR <= inc;
        end;
```

## Conclusion

Altera provides high-density PLDs that require larger configuration files. By using a flash memory device and an EPM3128A device in a design, a PLD can be quickly configured.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com