

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

```
entity PromLoader_APEX 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;
    Delk       : out     std_logic;
    nConfig    : buffer  std_logic;
```

--To extend the size of the memory change the size of Std_logic_vetor for A output:

```
    ADDR      : out     std_logic_vector(15 downto 0);
    CEn       : out     std_logic;
end;
```

```
architecture rtl of PromLoader_APEX is
```

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

```
constant start      :std_logic_vector(2 downto 0) := "000";
constant wait_nCfg_2us :std_logic_vector(2 downto 0) := "100";
constant status     :std_logic_vector(2 downto 0) := "001";
constant wait_5us   :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(9 downto 0);
```

```
begin
```

--The following process is used to divide 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 CLOCK. The value compared to must be
--such that the condition is true at a maximum rate of 10MHz (tclk = 100ns min).

```

    if (div = 7) then
        case pp is
            when start =>
                count <= (others => '0');
                inc <= (others => '0');
                waitd <= (others => '0');
                pp <= wait_nCfg_2us;

```

--This state is used in order to verify the tcfg timing (nCONFIG low pulse width).
-- Tcfg = 8µs min => 80 clock cycles of a 10MHz clock (this clock is CLOCK divide
--by the divider -div-)

```

        when wait_nCfg_2us =>
            count <= (others => '0');
            inc <= (others => '0');
            waitd <= waitd + 1;
            if waitd = 80 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_5us;

```

--This state is used in order to verify the tcf2ck timing (nCONFIG high to --first rising edge on DCLK). Tcf2ck = 5µs min => 400 clock cycles of a --10MHz clock (this clock is CLOCK divide by the divider -div-)

```

when wait_5us =>
    count <= (others => '0');
    inc <= (others => '0');
    waitd <= waitd + 1;
    if waitd = 400 then
        pp <= config;
    end if;

```

--This state is used to increment the memory address. In the same state when --the Conf_Done is high some clock cycles are added to avoid issues with old --files (the 10 clocks cycles).

```

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 = 440 then -- Modification (add 40 clock

```

cycles for APEX)

```

        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 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;

```