An-Najah National University

Faculty of Engineering and IT

جامعة النجاح الوطنية

كلية الهندسة وتكنولوجيا المعلومات

Digital Circuit Design Lab2

Dr Ashraf Armoush

Wednesday 2:00 pm – 5:00 pm

First Semester

Experiment Information	
Experiment Name: Two Bit Adder	Experiment Number: #2
Performed: 7 Oct 2020	Submitted: 8 Oct 2020
Partner Students	
1- Taher Anaya	2- Hadi Jml

Introduction:

The counter is a digital device, and the output of the counter includes a predefined state based on the clock pulse applications. The output of the counter can be used to count the number of pulses. Generally, counters consist of a flip-flop arrangement, which can be a synchronous counter or asynchronous counter. In synchronous counter, only one clock i/p is given to all flip-flops, whereas in the asynchronous counter, the flip flop's o/p is the clock signal from the nearby one.

In this experiment, we will build different types of counters using VHDL, and we will be synthesizing each counter on the FPGA.

Objectives:

- Construct different types of counters.
- Build a clock divider.
- Build a push-button debouncer.
- Build a Time Mux.
- Build BCD to 7-Segment decoder.
- Use seven-segment displays with the FPGA.

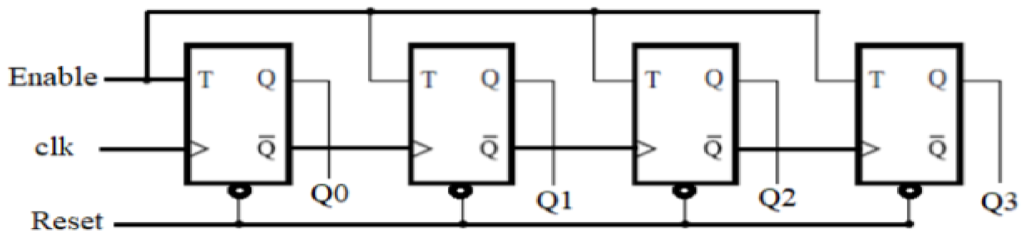
Procedure:

This experiment consists of 4 different parts:

1. Asynchronous Ripple Counter.
2. AutoUpDown Synchronous Counter.
3. Push Button UpDown Counter.
4. Two-Digit BCD Counter.

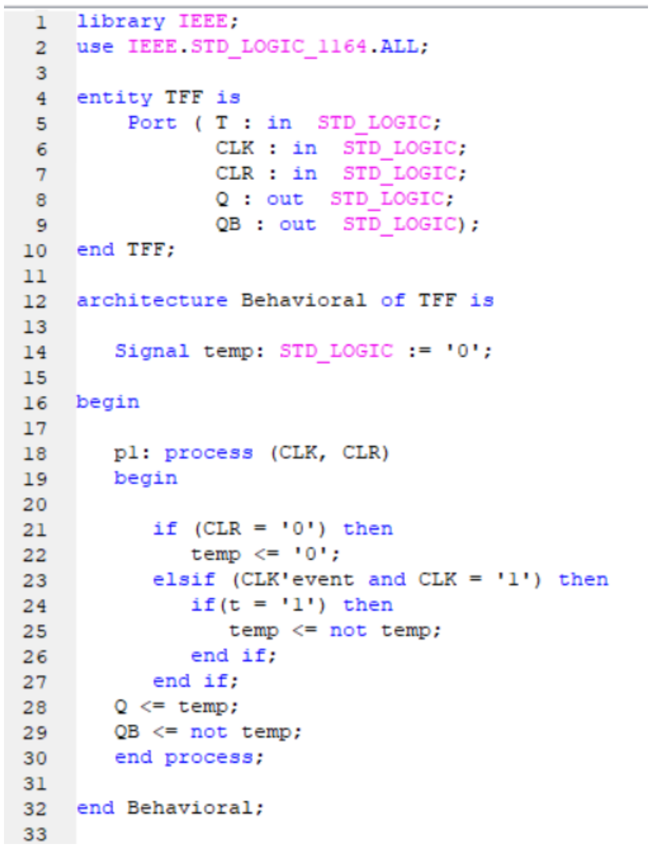
Asynchronous Ripple Counter

In this part, you will build a 4-bit Asynchronous ripple counter using T flip-flops.



First, we want to build a TFF to use for building the Ripple Counter.

And the VHDL code of a TFF is shown in the figure below.

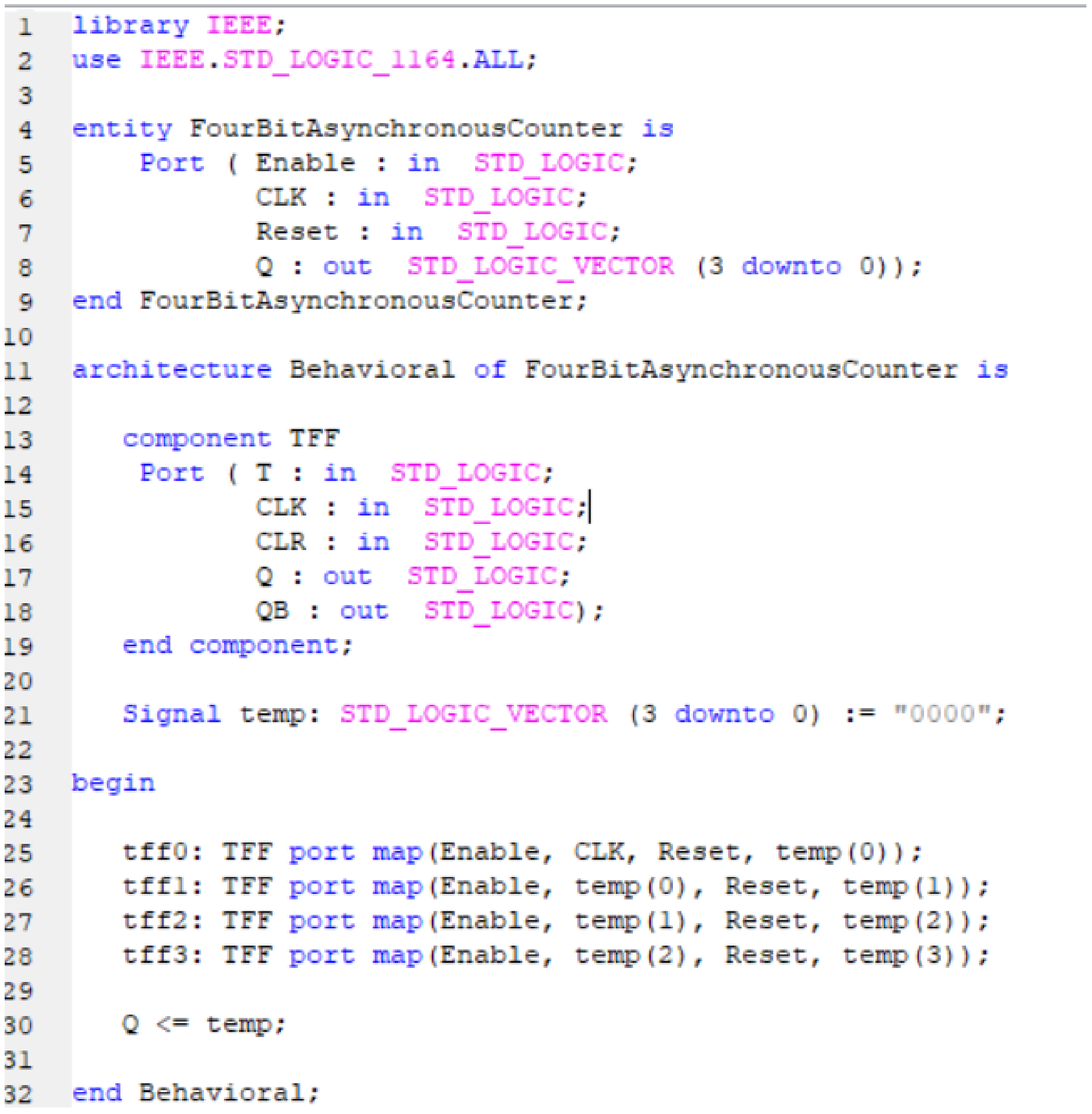


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TFF is
    Port ( T : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           CLR : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           QB : out  STD_LOGIC);
end TFF;

architecture Behavioral of TFF is

   Signal temp: STD_LOGIC := '0';

begin

   p1: process (CLK, CLR)
   begin

      if (CLR = '0') then
         temp <= '0';
      elsif (CLK'event and CLK = '1') then
         if(t = '1') then
            temp <= not temp;
         end if;
      end if;
   Q <= temp;
   QB <= not temp;
   end process;

end Behavioral;

```

Procedure: *(cont.)*

Asynchronous Ripple Counter *(cont.)*

Now, we can use TFF to build a Ripple Counter.
And the VHDL code of a Ripple Counter is shown in the figure below.



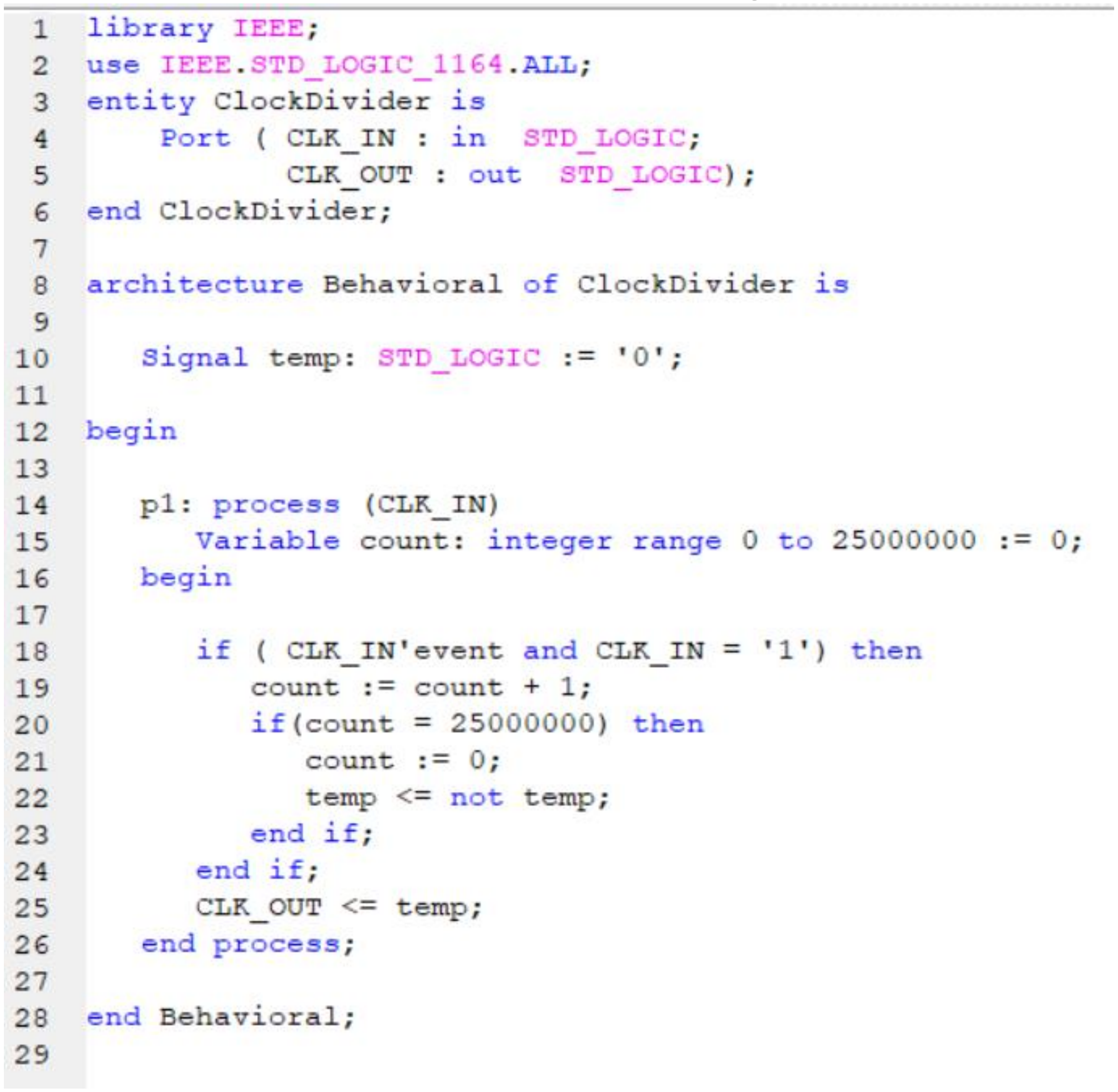
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FourBitAsynchronousCounter is
    Port ( Enable : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end FourBitAsynchronousCounter;

architecture Behavioral of FourBitAsynchronousCounter is

   component TFF
    Port ( T : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           CLR : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           QB : out  STD_LOGIC);
   end component;

   Signal temp: STD_LOGIC_VECTOR (3 downto 0)  := "0000";

begin

   tff0: TFF port map(Enable, CLK, Reset, temp(0));
   tff1: TFF port map(Enable, temp(0), Reset, temp(1));
   tff2: TFF port map(Enable, temp(1), Reset, temp(2));
   tff3: TFF port map(Enable, temp(2), Reset, temp(3));

   Q <= temp;

end Behavioral;
```

Procedure: *(cont.)*

Asynchronous Ripple Counter *(cont.)*

Because the system operates on 50MHz frequency, we should build an entity to divide the input clock (C9) to generate an output clock with 1Hz frequency.

And the VHDL code of a Clock Divider is shown in the figure below.

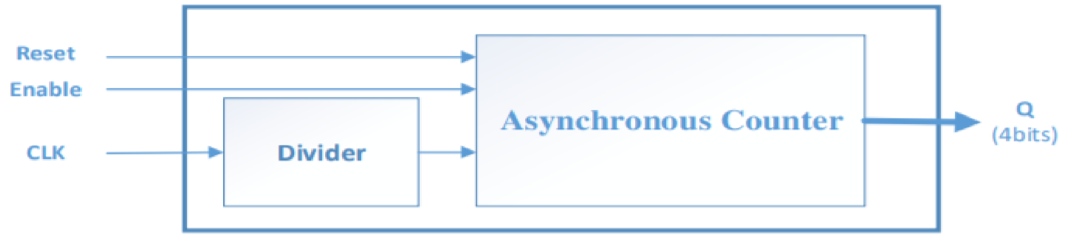


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ClockDivider is
    Port ( CLK_IN : in  STD_LOGIC;
           CLK_OUT : out  STD_LOGIC);
end ClockDivider;

architecture Behavioral of ClockDivider is

   Signal temp: STD_LOGIC := '0';

begin

   p1: process (CLK_IN)
      Variable count: integer range 0 to 25000000 := 0;
   begin

      if ( CLK_IN'event and CLK_IN = '1') then
         count := count + 1;
         if(count = 25000000) then
            count := 0;
            temp <= not temp;
         end if;
      end if;
      CLK_OUT <= temp;
   end process;

end Behavioral;

```

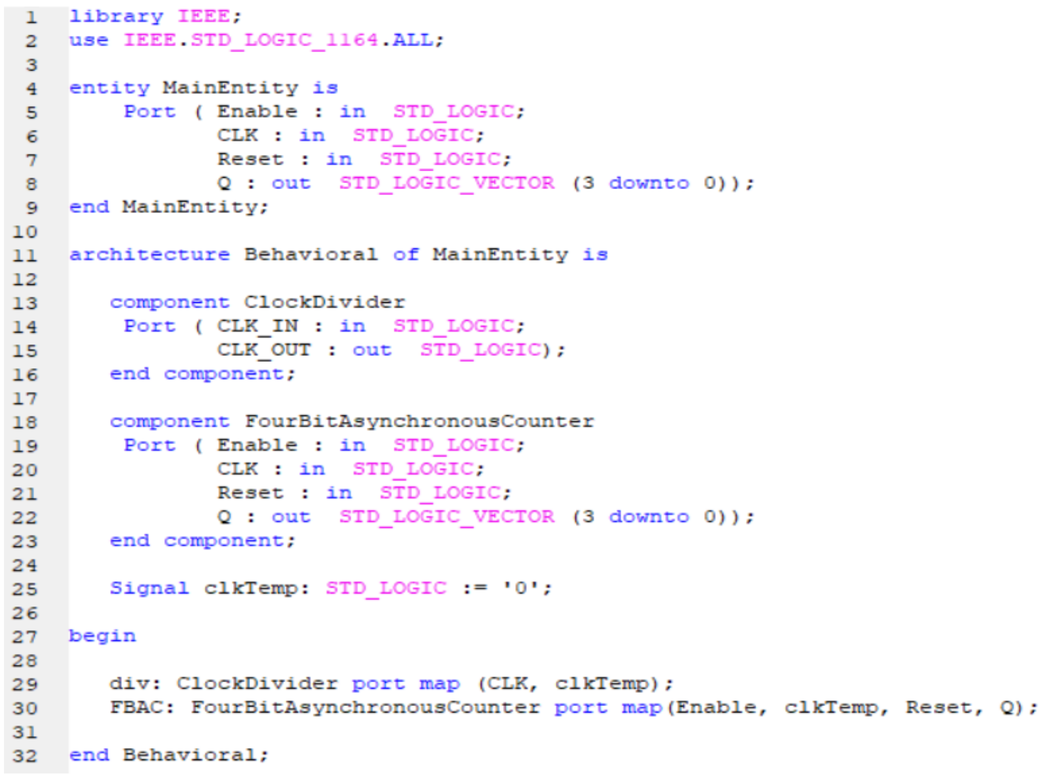
Procedure: (cont.)

Asynchronous Ripple Counter *(cont.)*

Now, we should build the main VHDL file to implement the first part. Our file should include two components from the clock divider and the asynchronous counter, as shown in the figure.



And the VHDL code of the main file is shown in the figure below.



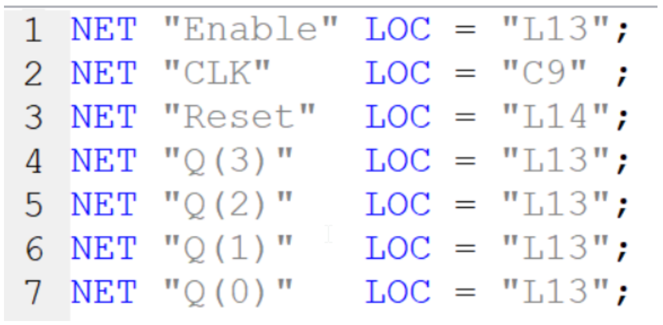
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MainEntity is
    Port ( Enable : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end MainEntity;

architecture Behavioral of MainEntity is

   component ClockDivider
    Port ( CLK_IN : in  STD_LOGIC;
           CLK_OUT : out  STD_LOGIC);
   end component;

   component FourBitAsynchronousCounter
    Port ( Enable : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
   end component;

   Signal clkTemp: STD_LOGIC := '0';

begin

   div: ClockDivider port map (CLK, clkTemp);
   FBAC: FourBitAsynchronousCounter port map(Enable, clkTemp, Reset, Q);

end Behavioral;
```

Procedure: *(cont.)*

Asynchronous Ripple Counter *(cont.)*

Note that the main file contains three inputs and four outputs, and this means we need three switches and 4 LEDs, so we must build a UCF file to match inputs with switches and outputs with LEDs.

And the Constraint File is shown in the figure below.

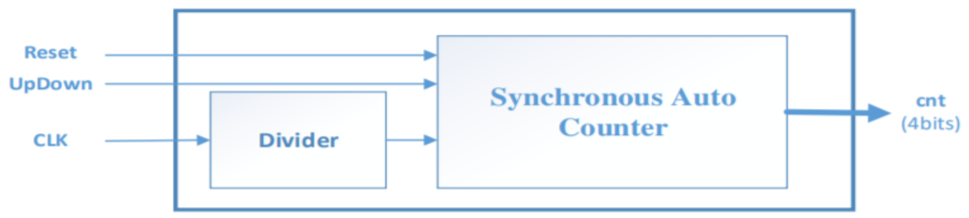


```
NET "Enable" LOC = "L13";
NET "CLK"    LOC = "C9"  ;
NET "Reset"  LOC = "L14";
NET "Q(3)"   LOC = "L13";
NET "Q(2)"   LOC = "L13";
NET "Q(1)"   LOC = "L13";
NET "Q(0)"   LOC = "L13";
```

That's all in the first part, now let's move on to the second part.

AutoUpDown Synchronous Counter

In this part, we should implement an automatic up-down counter. This is a 4-bit counter that increment/decrement automatically using the system clock. And because the system is operating on 50MHz frequency, we need to divide the clock to notice the change in LED's counting.



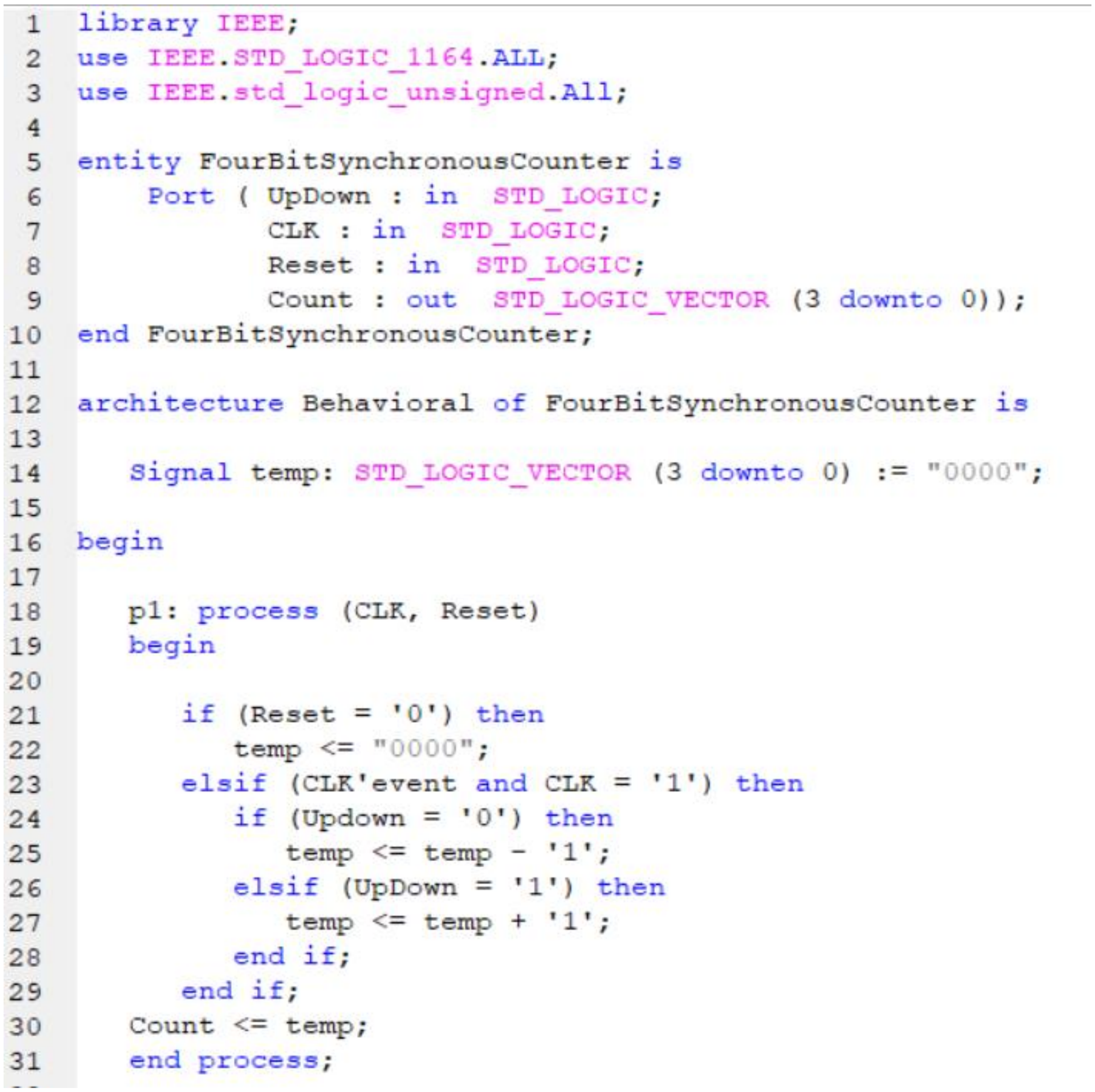
The first component we need in this part is Divider, and we implemented it in the previous part.

Procedure: *(cont.)*

AutoUpDown Synchronous Counter *(cont.)*

Now we need to build the synchronous counter.

The VHDL code of the synchronous counter is shown in the figure below.

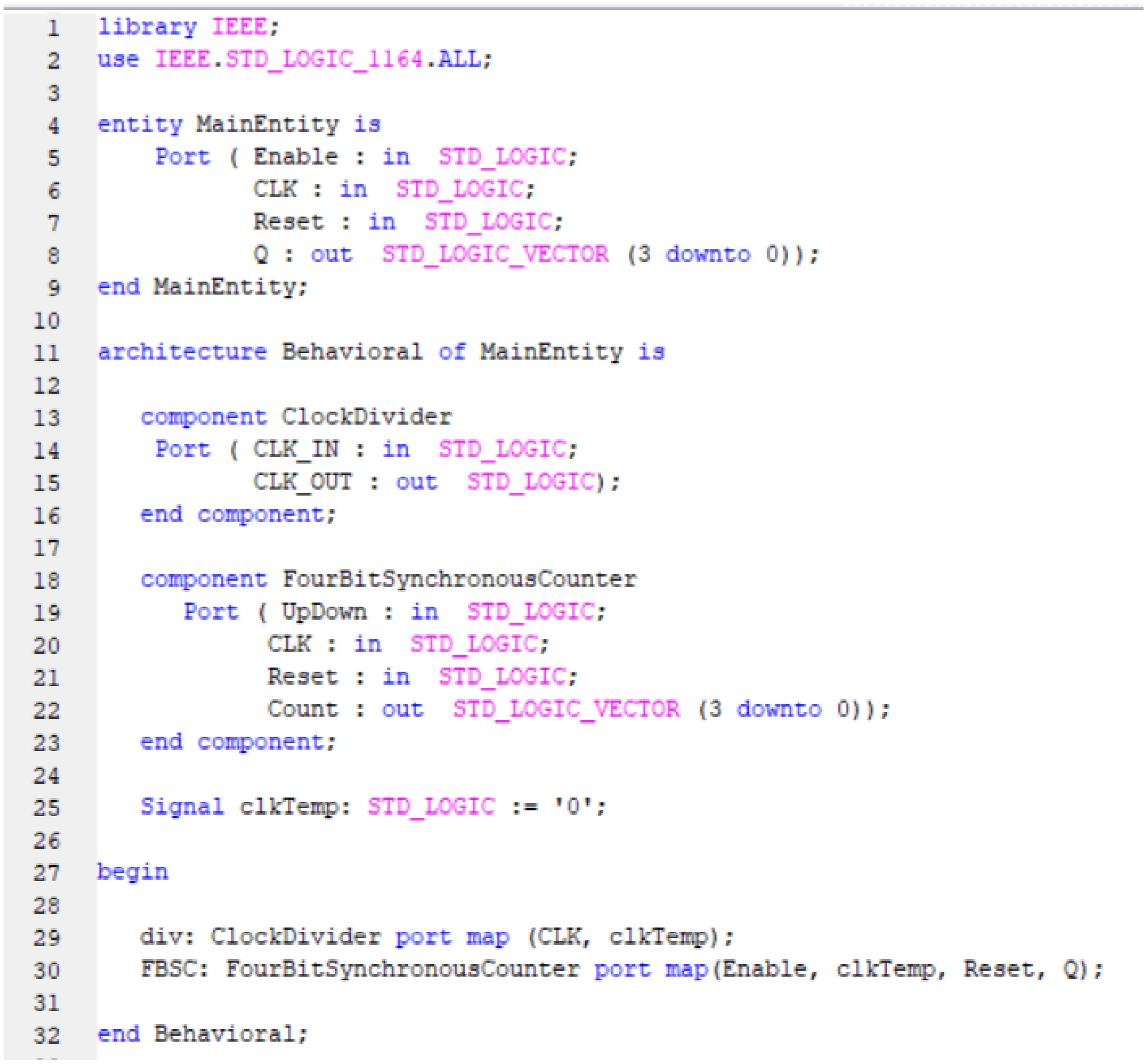


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.All;

entity FourBitSynchronousCounter is
    Port ( UpDown : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Count : out  STD_LOGIC_VECTOR (3 downto 0));
end FourBitSynchronousCounter;

architecture Behavioral of FourBitSynchronousCounter is

   Signal temp: STD_LOGIC_VECTOR (3 downto 0)  := "0000";

begin

   p1: process (CLK, Reset)
   begin

      if (Reset = '0') then
         temp <= "0000";
      elsif (CLK'event and CLK = '1') then
         if (Updown = '0') then
            temp <= temp - '1';
         elsif (UpDown = '1') then
            temp <= temp + '1';
         end if;
      end if;
   Count <= temp;
   end process;
```

Procedure: *(cont.)*

AutoUpDown Synchronous Counter *(cont.)*

Now, we should build the main VHDL file to implement the second part. Our file should include two components from the clock divider and the synchronous counter, as shown in the figure.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MainEntity is
    Port ( Enable : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end MainEntity;

architecture Behavioral of MainEntity is

   component ClockDivider
    Port ( CLK_IN : in  STD_LOGIC;
           CLK_OUT : out  STD_LOGIC);
   end component;

   component FourBitSynchronousCounter
      Port ( UpDown : in  STD_LOGIC;
             CLK : in  STD_LOGIC;
             Reset : in  STD_LOGIC;
             Count : out  STD_LOGIC_VECTOR (3 downto 0));
   end component;

   Signal clkTemp: STD_LOGIC := '0';

begin

   div: ClockDivider port map (CLK, clkTemp);
   FBSC: FourBitSynchronousCounter port map(Enable, clkTemp, Reset, Q);

end Behavioral;
```

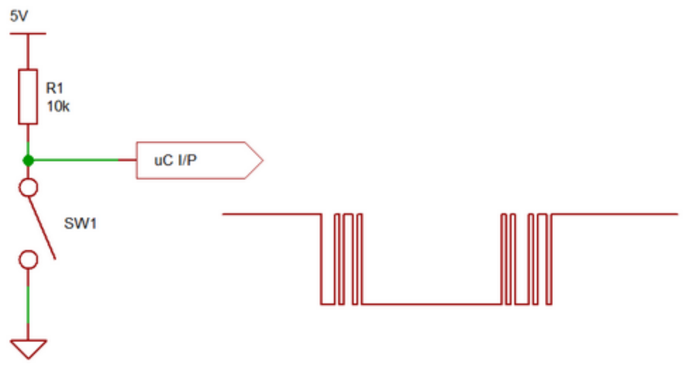
That's all in the second part, now let's move on to the third part.

Procedure: *(cont.)*

Push Button UpDown Counter

In this part, we want to build a synchronous counter like the previous part, but we'll using push-button to generate pulses instead of using the clock system, but there is a problem that we will face in this section, which is the **bounce**.

When you push a button, two metal parts come together. For the user, it might seem that the contact is made instantly. That is not quite correct. Inside the switch, there are moving parts. When you push the button, it initially makes contact with the other metal part, but just in a brief split of a microsecond. Then it makes contact a little longer, and then again, a little longer. In the end, the switch is fully closed. The button is bouncing between in-contact and not in-contact. "When the switch is closed, the two contacts separate and reconnect, typically 10 to 100 times throughout about 1ms."

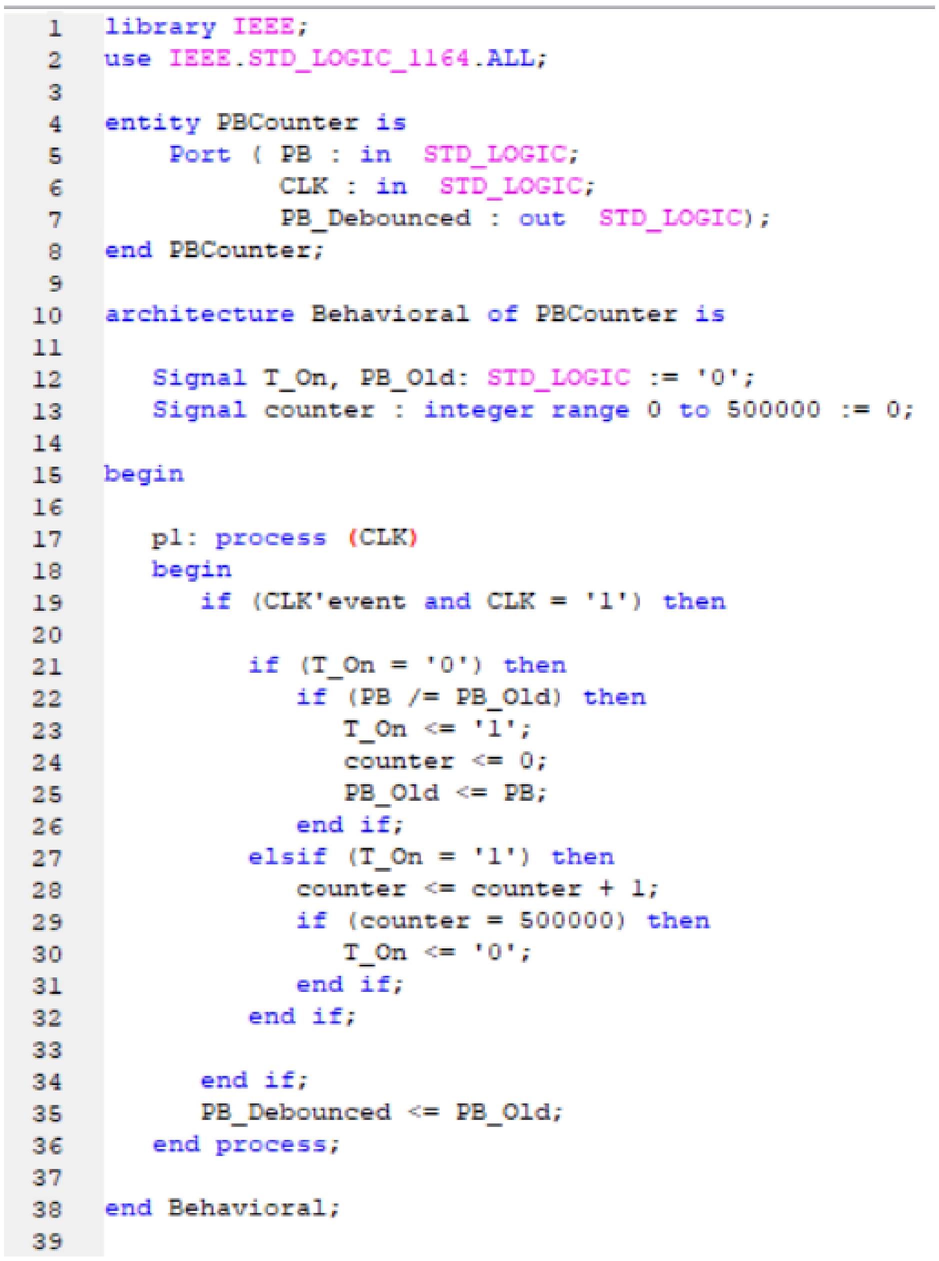


Now, we should implement a module to debounce the push button. The debouncing module is just a delay that ensures the correct state of the button.

<u>Procedure:</u> *(cont.)*

<u>AutoUpDown Synchronous Counter (cont.)</u>

The VHDL code of the debouncer is shown in the figure below.



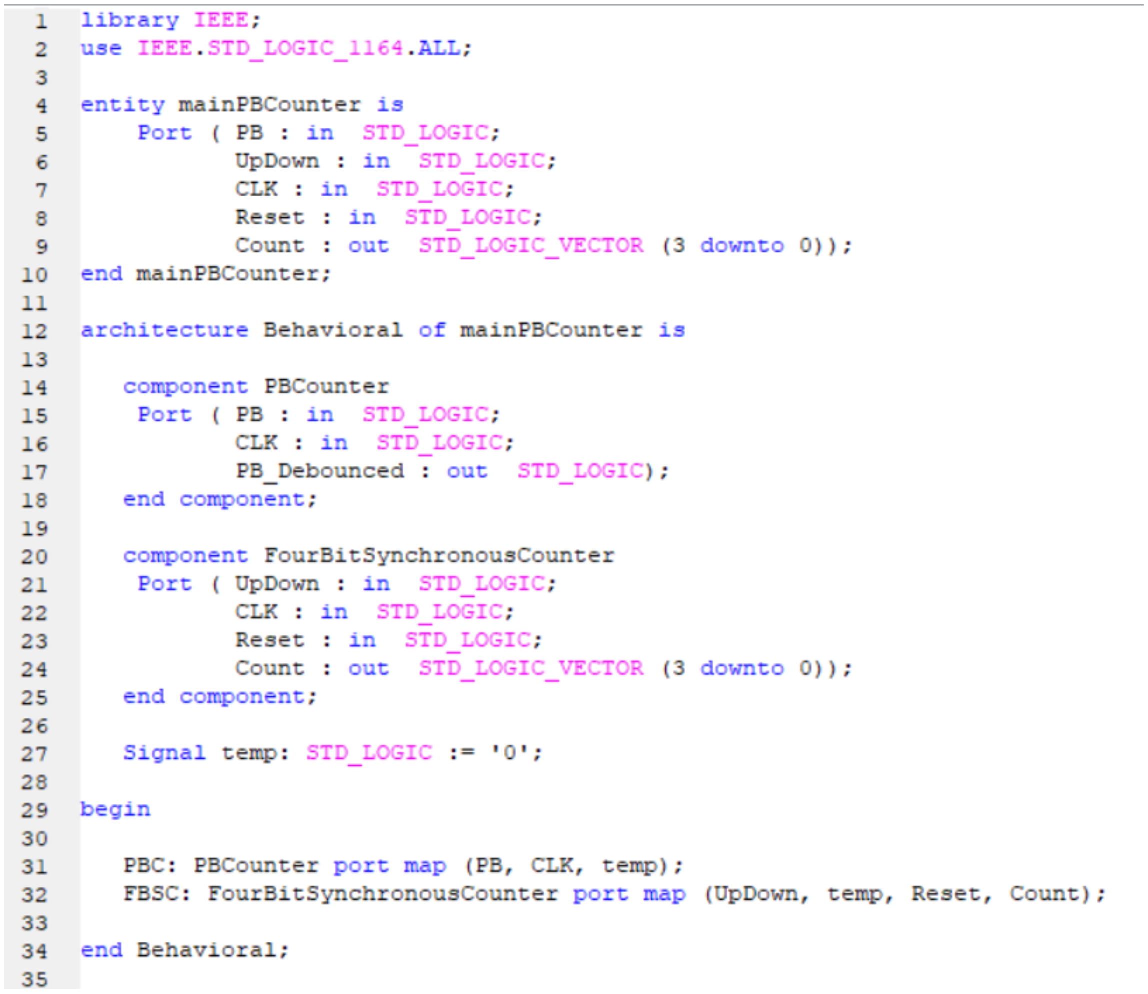
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PBCounter is
    Port ( PB : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           PB_Debounced : out  STD_LOGIC);
end PBCounter;

architecture Behavioral of PBCounter is

   Signal T_On, PB_Old: STD_LOGIC := '0';
   Signal counter : integer range 0 to 500000 := 0;

begin

   p1: process (CLK)
   begin
      if (CLK'event and CLK = '1') then

         if (T_On = '0') then
            if (PB /= PB_Old) then
               T_On <= '1';
               counter <= 0;
               PB_Old <= PB;
            end if;
         elsif (T_On = '1') then
            counter <= counter + 1;
            if (counter = 500000) then
               T_On <= '0';
            end if;
         end if;

      end if;
      PB_Debounced <= PB_Old;
   end process;

end Behavioral;

```

<u>Procedure:</u> *(cont.)*

<u>AutoUpDown Synchronous Counter (cont.)</u>

The second component we need it in this part is a synchronous counter, and we implemented it in the previous part.

Now, we should build the main VHDL file to implement the third part. Our file should include two components from the debouncer and the synchronous counter, as shown in the figure.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mainPBCounter is
    Port ( PB : in  STD_LOGIC;
           UpDown : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Count : out  STD_LOGIC_VECTOR (3 downto 0));
end mainPBCounter;

architecture Behavioral of mainPBCounter is

   component PBCounter
    Port ( PB : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           PB_Debounced : out  STD_LOGIC);
   end component;

   component FourBitSynchronousCounter
    Port ( UpDown : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Count : out  STD_LOGIC_VECTOR (3 downto 0));
   end component;

   Signal temp: STD_LOGIC := '0';

begin

   PBC: PBCounter port map (PB, CLK, temp);
   FBSC: FourBitSynchronousCounter port map (UpDown, temp, Reset, Count);

end Behavioral;

```

That's all in the third part, now let's move on to the last part.

Procedure: *(cont.)*

Two-Digit BCD Counter

In this part, we will build a two-digit BCD counter that counts from 00 to 99. The result is to be displayed on a two-seven-segment display module connected to the kit. The circuit diagram for this module is shown in the figure.

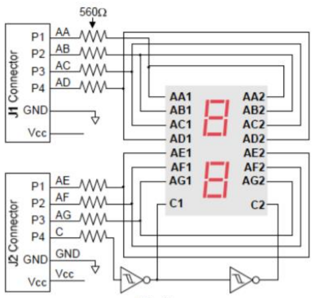
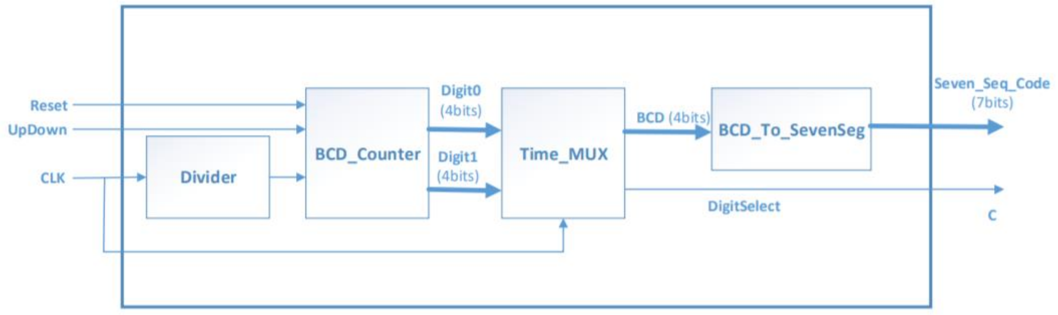


Fig1.

The seven segments in this module are common cathode. This means to light a LEDs segment should be pulled high.
Our design should include four components, as shown in the figure.

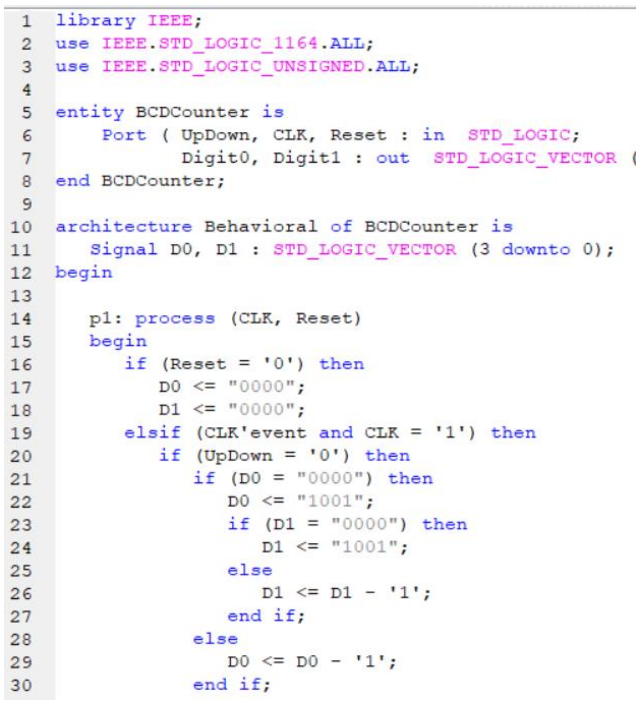


The first component we need in this part is Divider, and we implemented it in the previous parts. The second component we need is a two-digit (two-decade) BCD counter counts in decimal from 00 decimal to 99 decimals.

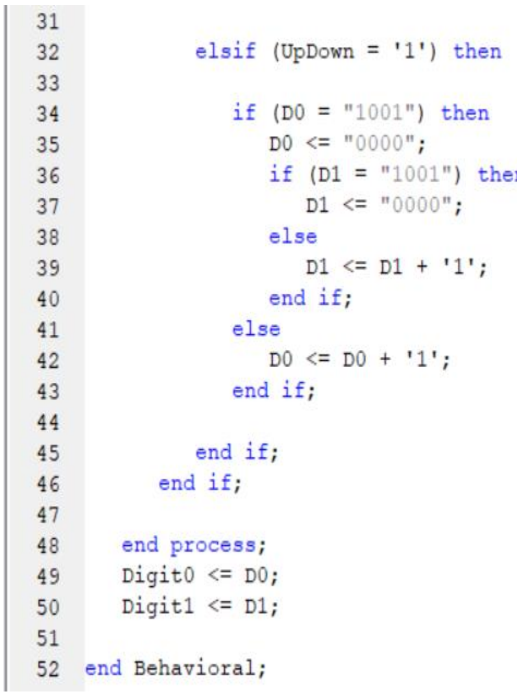
Procedure: *(cont.)*

Two-Digit BCD Counter (cont.)

The VHDL code of the BCD-counter is shown in the figures below.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BCDCounter is
    Port ( UpDown, CLK, Reset : in  STD_LOGIC;
           Digit0, Digit1 : out  STD_LOGIC_VECTOR (
end BCDCounter;

architecture Behavioral of BCDCounter is
   Signal D0, D1 : STD_LOGIC_VECTOR (3 downto 0);
begin

   p1: process (CLK, Reset)
   begin
      if (Reset = '0') then
         D0 <= "0000";
         D1 <= "0000";
      elsif (CLK'event and CLK = '1') then
         if (UpDown = '0') then
            if (D0 = "0000") then
               D0 <= "1001";
               if (D1 = "0000") then
                  D1 <= "1001";
               else
                  D1 <= D1 - '1';
               end if;
            else
               D0 <= D0 - '1';
            end if;
```



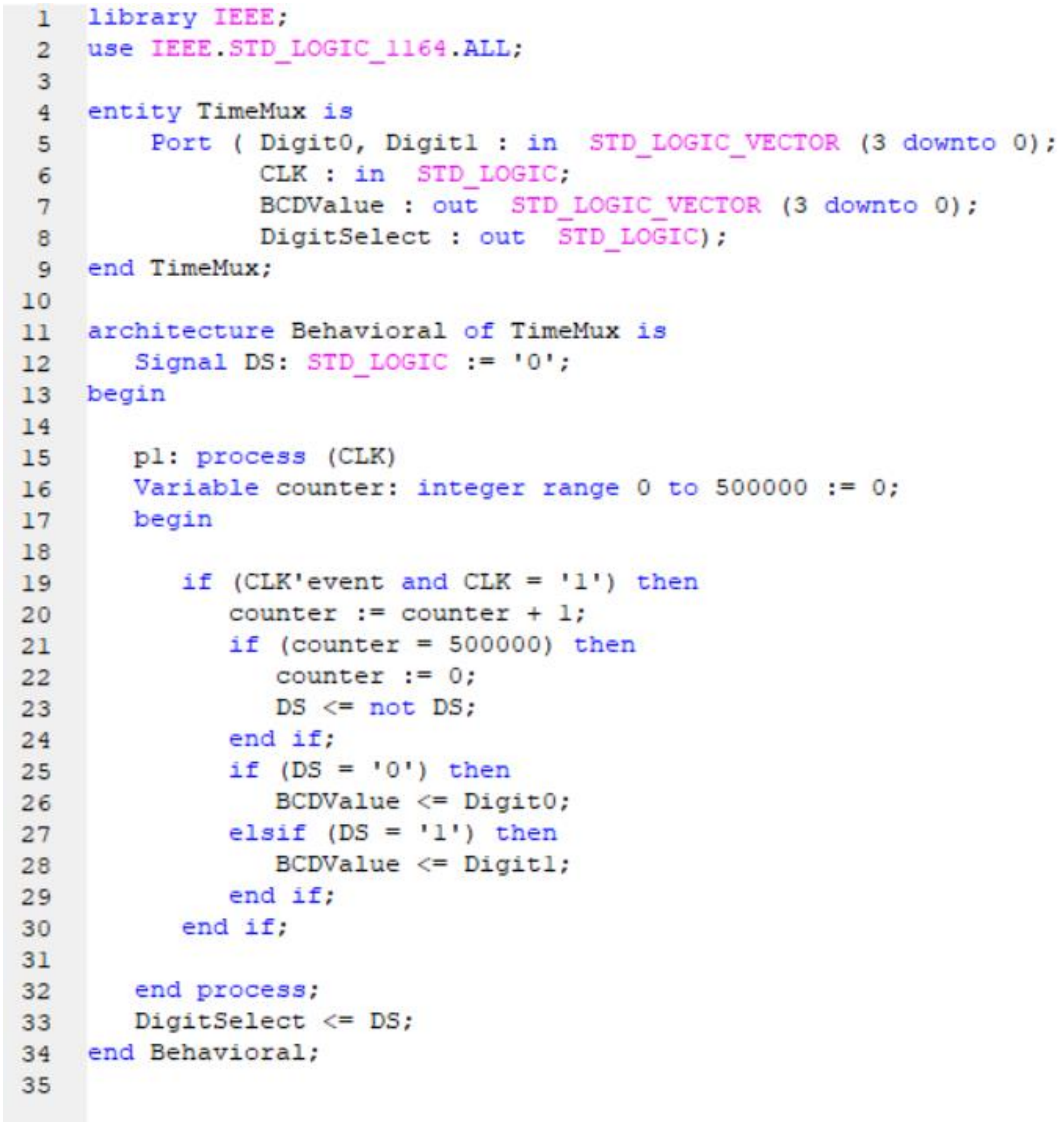
```vhdl

         elsif (UpDown = '1') then

            if (D0 = "1001") then
               D0 <= "0000";
               if (D1 = "1001") then
                  D1 <= "0000";
               else
                  D1 <= D1 + '1';
               end if;
            else
               D0 <= D0 + '1';
            end if;

         end if;
      end if;

   end process;
   Digit0 <= D0;
   Digit1 <= D1;

end Behavioral;
```

But we cannot simultaneously display a digit on both 7-segment LED displays. Instead, repeatedly and continuously displaying a digit on each display faster than the human eye can respond, both displays will appear to be illuminated at the same time. In this part, you will implement a multiplexer to select between the two digits (Tens and Ones) of the BCD_Counter. Each 7-segment LED display should be illuminated for 10 ms. The output DigSelect signal is used to indicate the current active digit (0 and 1 for digit0 and digit1 respectively), so we need to build the third component in this part, which is Time-Mux.

Procedure: *(cont.)*

Two-Digit BCD Counter (cont.)

The VHDL code of the Time-Mux is shown in the figure below.



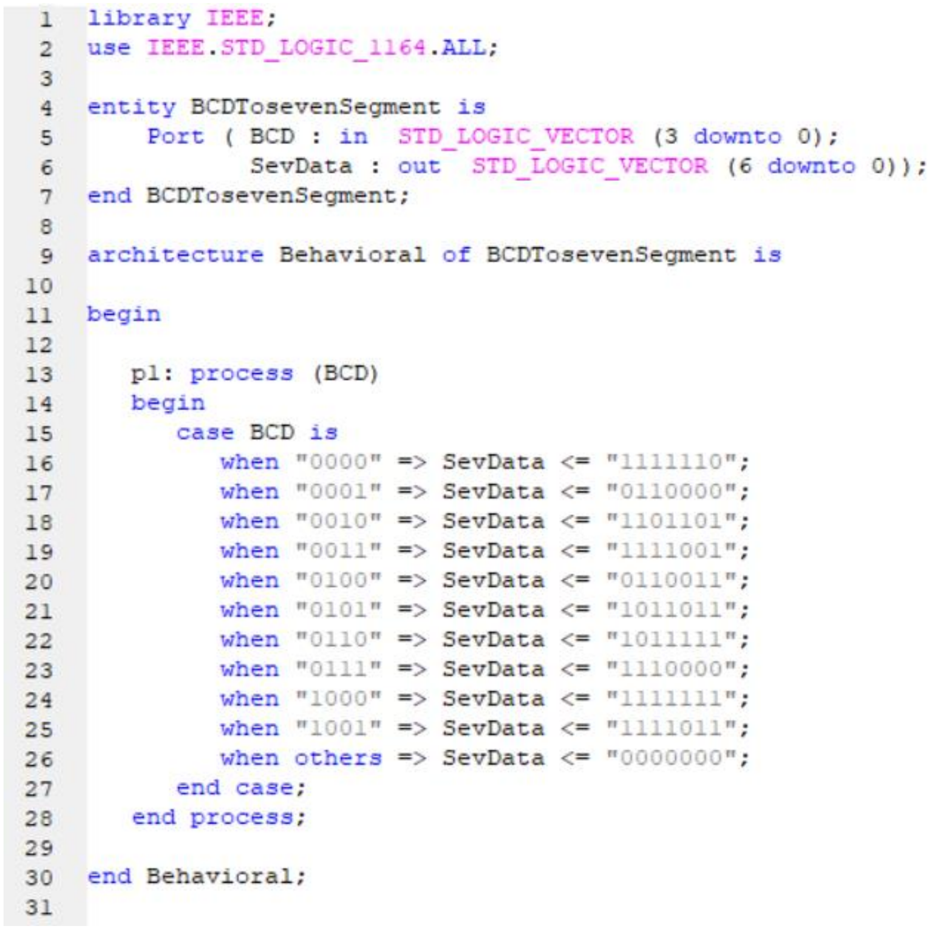
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TimeMux is
    Port ( Digit0, Digit1 : in  STD_LOGIC_VECTOR (3 downto 0);
           CLK : in  STD_LOGIC;
           BCDValue : out  STD_LOGIC_VECTOR (3 downto 0);
           DigitSelect : out  STD_LOGIC);
end TimeMux;

architecture Behavioral of TimeMux is
   Signal DS: STD_LOGIC := '0';
begin

   p1: process (CLK)
   Variable counter: integer range 0 to 500000 := 0;
   begin

      if (CLK'event and CLK = '1') then
         counter := counter + 1;
         if (counter = 500000) then
            counter := 0;
            DS <= not DS;
         end if;
         if (DS = '0') then
            BCDValue <= Digit0;
         elsif (DS = '1') then
            BCDValue <= Digit1;
         end if;
      end if;

   end process;
   DigitSelect <= DS;
end Behavioral;

```

Procedure: *(cont.)*

Two-Digit BCD Counter (cont.)

Now, we want to convert a 4-bit BCD number to a 7-bit control signal, which can be displayed on a 7-segment display. The seven segments in this module are common cathode. Therefore, our output signals should be active high.

The VHDL code of the BCD to 7-Segment Decoder is shown in the figure below.

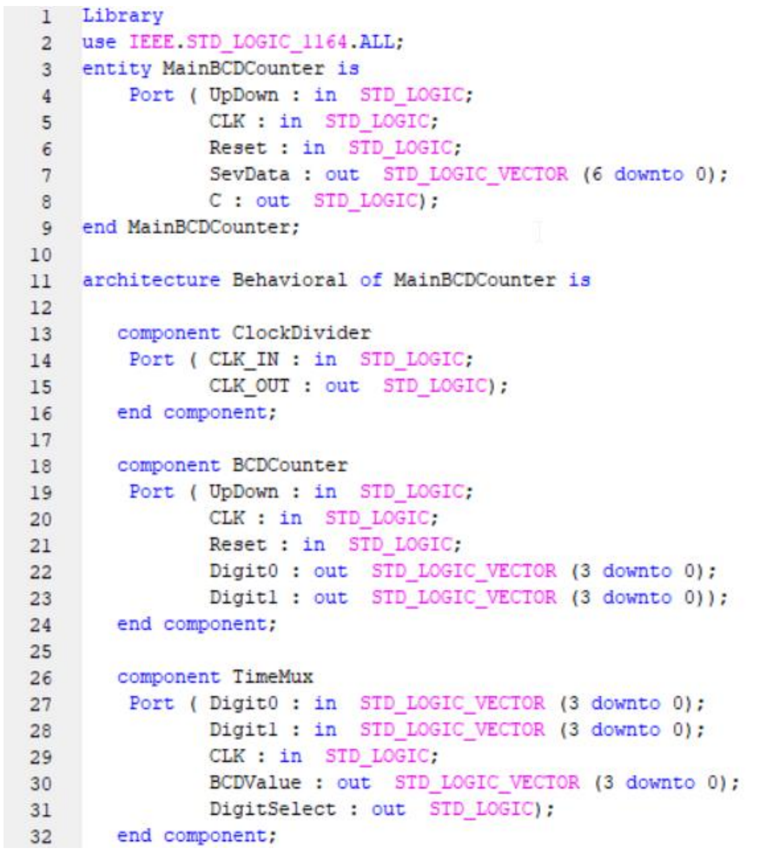


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BCDTosevenSegment is
    Port ( BCD : in  STD_LOGIC_VECTOR (3 downto 0);
           SevData : out  STD_LOGIC_VECTOR (6 downto 0));
end BCDTosevenSegment;

architecture Behavioral of BCDTosevenSegment is

begin

   p1: process (BCD)
   begin
      case BCD is
         when "0000" => SevData <= "1111110";
         when "0001" => SevData <= "0110000";
         when "0010" => SevData <= "1101101";
         when "0011" => SevData <= "1111001";
         when "0100" => SevData <= "0110011";
         when "0101" => SevData <= "1011011";
         when "0110" => SevData <= "1011111";
         when "0111" => SevData <= "1110000";
         when "1000" => SevData <= "1111111";
         when "1001" => SevData <= "1111011";
         when others => SevData <= "0000000";
      end case;
   end process;

end Behavioral;

```

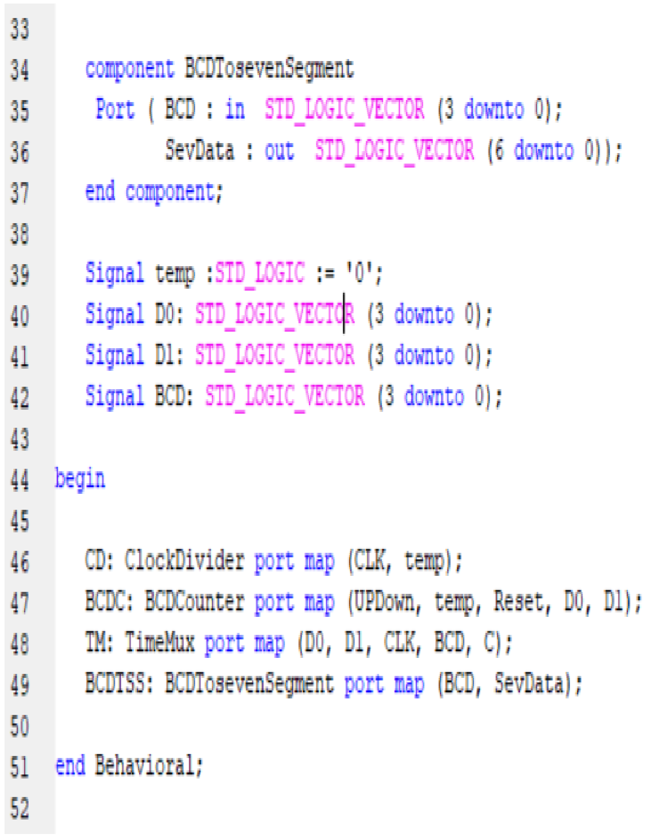
Procedure: *(cont.)*

Two-Digit BCD Counter (cont.)

Now, we should build the main VHDL file to implement the last part. Our file should include four components from the clock divider and BCD-Counter and Time-Mux and BCDToSevenSegment, as shown in the figure.



```
Library
use IEEE.STD_LOGIC_1164.ALL;
entity MainBCDCounter is
    Port ( UpDown : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           SevData : out  STD_LOGIC_VECTOR (6 downto 0);
           C : out  STD_LOGIC);
end MainBCDCounter;

architecture Behavioral of MainBCDCounter is

   component ClockDivider
    Port ( CLK_IN : in  STD_LOGIC;
           CLK_OUT : out  STD_LOGIC);
   end component;

   component BCDCounter
    Port ( UpDown : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Digit0 : out  STD_LOGIC_VECTOR (3 downto 0);
           Digit1 : out  STD_LOGIC_VECTOR (3 downto 0));
   end component;

   component TimeMux
    Port ( Digit0 : in  STD_LOGIC_VECTOR (3 downto 0);
           Digit1 : in  STD_LOGIC_VECTOR (3 downto 0);
           CLK : in  STD_LOGIC;
           BCDValue : out  STD_LOGIC_VECTOR (3 downto 0);
           DigitSelect : out  STD_LOGIC);
   end component;
```



```

   component BCDTosevenSegment
    Port ( BCD : in  STD_LOGIC_VECTOR (3 downto 0);
           SevData : out  STD_LOGIC_VECTOR (6 downto 0));
   end component;

   Signal temp :STD_LOGIC := '0';
   Signal D0: STD_LOGIC_VECTOR (3 downto 0);
   Signal D1: STD_LOGIC_VECTOR (3 downto 0);
   Signal BCD: STD_LOGIC_VECTOR (3 downto 0);

begin

   CD: ClockDivider port map (CLK, temp);
   BCDC: BCDCounter port map (UPDown, temp, Reset, D0, D1);
   TM: TimeMux port map (D0, D1, CLK, BCD, C);
   BCDTSS: BCDTosevenSegment port map (BCD, SevData);

end Behavioral;

```

That's all in this experiment.