CPE166 Lab 3 Part 1
By: Prof. Pang

Lab 3

Part 1: (7, 4) Hamming Code Generator

The key of the Hamming Code is the use of extra parity bits to allow the correction of a single bit error. **Hamming** (**7**,**4**) is a linear error-correcting **code** that encodes four bits of data into seven bits by adding three even parity bits shown in figure 1. This project is to design (7, 4) Hamming Code generator in VHDL.

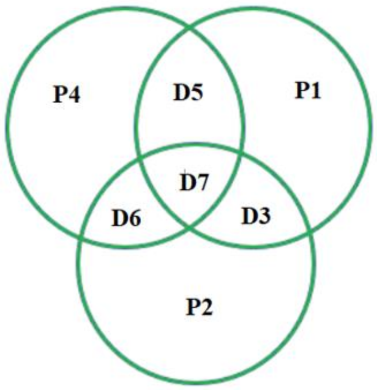


Figure 1. (7, 4) hamming code diagrams

In the above figure, D7, D6, D5 and D3 are input data,

- P4 is the even parity bit of binary data D7, D6 and D5;
- P2 is the even parity bit of binary data D7, D6 and D3;
- P1 is the even parity bit of binary data D7, D5 and D3.

The interface of this design is shown in Table 1.

Table 1. (7, 4) hamming code interface signals

Port Names	Port Direction	Port Size
D7	Input	1 bit
D6	Input	1 bit
D5	Input	1 bit
D3	Input	1 bit
DOUT	Output	7 bits (7 downto 1)

In the final design, D7, D6, D5, D3 are input data, DOUT is the final 7-bit hamming code.

Demo Requirement

You need to demonstrate the final simulation waveform of this design to your lab instructor.

CPE166 Lab 3 Part 1
By: Prof. Pang

Lab Procedure

Step 1. Parity Function

The even parity bit is the parity bit added to the data block. It is used to ensure that the total number of bits (including parity bits) in the message is even for error detection.

For this step, you will practice and review again after learning VHDL functions and program packages in class.

Simplified Sample Syntax

```
function function_name (parameter_list) return type is
        declarations
begin
        sequential statements
end function_name;
```

The following MY_PACK.vhd is used to create a package that can be used. Therefore, in your project, you need to add this source file first.

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;

package MY_PACK is

 function PARITY (D : std_logic_vector)       -- declaration of function

 return std_logic;

end MY_PACK;

package body MY_PACK is

function PARITY (D : std_logic_vector)       -- implementation of function inside the package body

 return std_logic is

 variable TMP : std_logic;

 begin

      TMP := D(0);

      for J in 1 to D'high loop

          TMP := TMP xor D(J);

       end loop;      -- works for any size of D

 return TMP;

 end PARITY;      -- even parity

end MY_PACK;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

Step 2. Even Parity Bit of 3-bit Input Data

In the following PAR.vhd design, the "use work.MY_PACK.all" statement includes the MY_PACK package. As a result, you can directly call the PARITY function in the PAR.vhd design to obtain the even parity bit value of the three-bit input data.

```
library ieee;

use ieee.std_logic_1164.all;

use work.MY_PACK.all;

entity PAR is

 port(  db: in std_logic_vector(2 downto 0);

           pb: out std_logic);

end PAR;

architecture ARCH of PAR is

begin

       pb <= PARITY(db);

 end ARCH;
```

After including the above two VHDL files into your project, you need to write a testbench file for PAR design. Then, you can run simulations to verify the generation of the even parity signal of the three-bit test data in VHDL.

Step 3. (7, 4) Hamming Code Generator

(7,4) hamming code encodes four bits of data into seven bits by adding three even parity bits. The contents below are for you to review what you have learned in class.

For the four binary data bits D7 D6 D5 D3, use three even parity bits P4 P2 P1 according to the following diagram.

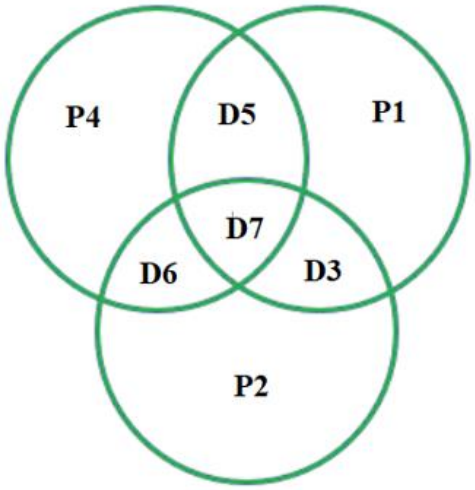


Figure 1. (7, 4) hamming code diagrams

In the above figure,

- P4 is the even parity bit of binary data D7, D6 and D5;
- P2 is the even parity bit of binary data D7, D6 and D3;
- P1 is the even parity bit of binary data D7, D5 and D3.

For the 4-bit input data D7 D6 D5 D3, the final (7, 4) hamming code will be

- D7, D6, D5, P4, D3, P2, P1

Now, it is time to design the hamming.vhd with the following interface signals according to the above (7, 4) Hamming code generation scheme.

Table 1. (7, 4) hamming code interface signals

Port Names	Port Direction	Port Size
D7	Input	1 bit
D6	Input	1 bit
D5	Input	1 bit
D3	Input	1 bit
DOUT	Output	7 bits (7 downto 1)

In the hamming.vhd design, D7, D6, D5, D3 are input data, DOUT is the final 7-bit hamming code.

DOUT is the concatenation of the following data bits:

- D7, D6, D5, P4, D3, P2, P1
  (bit7, bit6, bit5, bit 4, bit3, bit2, bit 1 of DOUT)

P4, P2, P1 are the internal parity signals used inside hmming.vhd.

In this design, you can use the PAR component in step 1, and create three PAR instances to generate P4, P2 and P1.

The block diagram of the hamming.vhd is shown below.

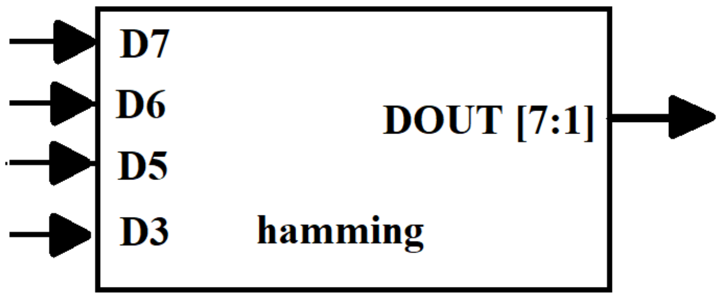


Figure 2. (7, 4) hamming code generator design block diagram

After completing hamming.vhd, you need to write testbench to run simulations.

CPE166 Lab 3 Part 1
By: Prof. Pang

Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample VHDL Codes:

(1). Concatenation Operator (&)

```
architecture arch of  one_design is

signal    a:   std_logic_vector ( 3 downto 0);

signal   b, c, d, e:  std_logic;

begin

       a <= b & c & d & e;  -- Concatenate four single bit signals to create one 4-bit signal

      …

end arch;
```

(2). cir.vhd   and cir_tb.vhd

```
library ieee;

use ieee.std_logic_1164.all;

entity cir is

port(     a: in std_logic;

             b: in std_logic;

            c: in std_logic;

             f: out std_logic

);

end cir;

architecture dataflow of cir is

begin

   f <= ( a and b ) or c;

end dataflow;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;


entity tb_cir is

end tb_cir;


architecture tb of tb_cir is


   component cir

      port (a : in std_logic;

            b : in std_logic;

            c : in std_logic;

            f : out std_logic);

   end component;


   signal a : std_logic;

   signal b : std_logic;

   signal c : std_logic;

   signal f : std_logic;

begin

   dut : cir  port map (

             a => a,

            b => b,

            c => c,

            f => f   );

   stimuli : process

   begin

      a <= '0';
```

```
        b <= '0';
        c <= '0';
        wait for 10 ns;
        a <= '0';
        b <= '1';
        c <= '0';
        wait for 10 ns;
        a <= '1';
        b <= '1';
        c <= '0';
        wait for 10 ns;
        a <= '0';
        b <= '1';
        c <= '1';
        wait for 10 ns;
        a <= '1';
        b <= '1';
        c <= '1';
        wait for 10 ns;
        a <= '0';
        b <= '0';
        c <= '0';
        wait for 10 ns;
        wait;
    end process;
end tb;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

(3). example.vhd and example_tb.vhd

```
library ieee ;

use ieee.std_logic_1164.all ;


entity example is

  port( a: in std_logic;

        b,c: in std_logic_vector (1 downto 0);

        f:  out std_logic_vector (1 downto 0)

  );

end example;


architecture gates of example is

signal  m, n:  std_logic_vector(1 downto 0);

begin

    m <=  (a & a ) and b;

   -- AND gate comment: m(0) <= a & b(0);

   -- AND gate comment: m(1) <= a & b(1);


    n <=  (a & a ) and c;

    f <=  m or n;

end gates;
```

CPE166 Lab 3 Part 1
By: Prof. Pang

```
library ieee;

use ieee.std_logic_1164.all;


entity tb_example is

end tb_example;


architecture tb of tb_example is

   component example

      port (a : in std_logic;

            b : in std_logic_vector (1 downto 0);

            c : in std_logic_vector (1 downto 0);

            f : out std_logic_vector (1 downto 0));

   end component;

   signal a : std_logic;

   signal b : std_logic_vector (1 downto 0);

   signal c : std_logic_vector (1 downto 0);

   signal f : std_logic_vector (1 downto 0);

begin

   dut : example

   port map (a => a,

            b => b,

            c => c,

            f => f);

   a <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;

   b <= "00", "01" after 5 ns, "10" after 10 ns, "11" after 20 ns, "10" after 30 ns;

   c <= "11", "10" after 10 ns, "01" after 30 ns, "11" after 40 ns;

end tb;
```

CPE166 Lab 3 Part 2
By: Prof. Pang

Lab 3

Part 2: Pseudorandom Number Generator

A linear feedback shift register (LFSR) is a shift register whose input bit is the output of a linear logic function of two or more of its previous states. The LFSR circuit can be used as pseudorandom number generator. This project is a design of a 5-stage LFSR circuit shown below.

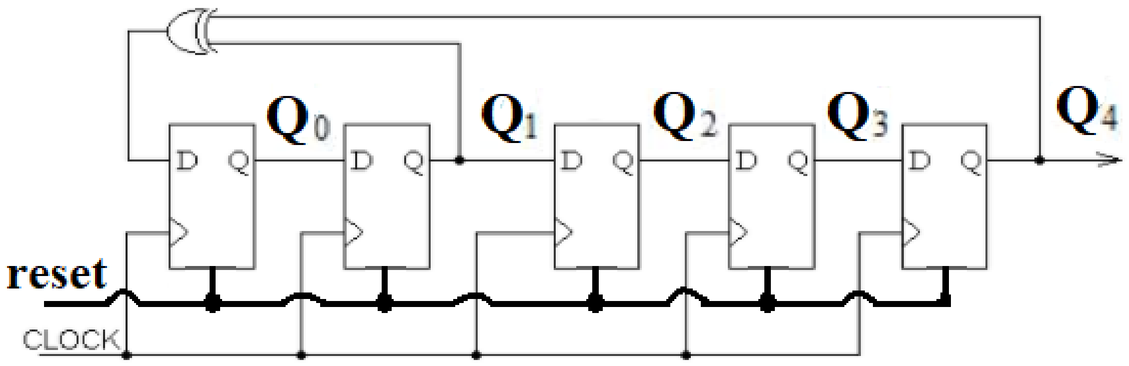


Figure 1. 5-stage LFSR diagram

Demo Requirement

Make sure that this LFSR design uses a non-zero value as the initial seed value. You need to demonstrate the final simulation waveform of this design to your lab instructor and show that the value of Q (4 drops to 0) repeats every 31 clock cycles.

CPE166 Lab 3 Part 2
By: Prof. Pang

Lab Procedure

Step 1. LFSR Design Review

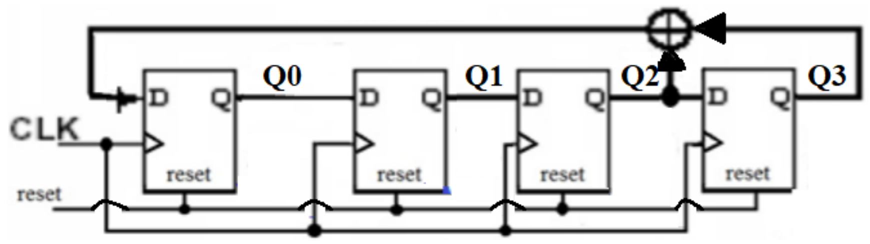


Figure 2. 4-stage LFSR diagram

Review the above circuit you have learnt in class with the VHDL design below.

```
Library ieee;

Use ieee.std_logic_1164.all;

Entity lfsr is

        Port (  reset, clk:   in std_logic;

                   Q:  out std_logic_vector (3 downto 0)  );

End lfsr;

Architecture beh of lfsr is

Signal   m:  std_logic_vector (3 downto 0);

Begin

               Process(reset, clk)

               Begin

                    If ( reset = '1')  then

                           m <=  ( 0=> '1', others =>'0');                 --- value of “0001”

                    elsif (rising_edge(clk) ) then

                           m(3 downto 1) <= m(2 downto 0);

                            m(0) <= m(2) xor m(3);

                    end if;

              end process;

             Q <= m;
```

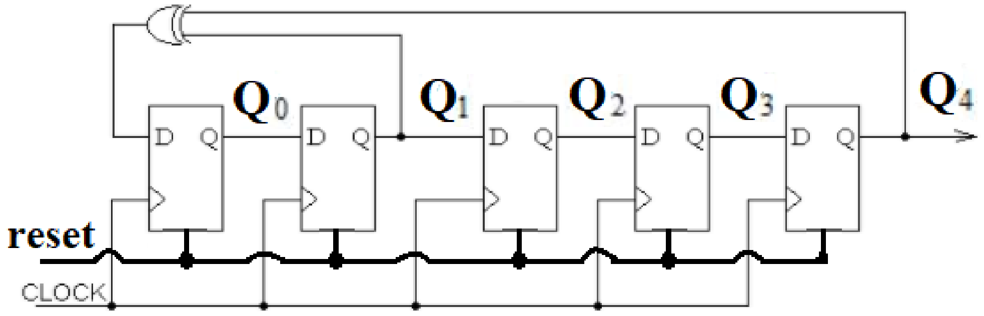
```
end beh;
```

Step 2. LFSR Testbench Design Review for Fig. 2 Circuit

```
Library ieee;

Use ieee.std_logic_1164.all;

Entity lfsr_tb is

End lfsr_tb;

Architecture tb of lfsr_tb is

signal clk, reset: std_logic;

signal Q:   std_logic_vector (3 downto 0);

component lfsr

    Port (  reset, clk:   in std_logic;

            Q:  out std_logic_vector (3 downto 0)  );

End component;

Begin

    uut:  lfsr port map(reset, clk, Q);

    Process

    Begin

       Clk <= '0';

       Wait for 5 ns;

      Clk <= '1';

      Wait for 5 ns;

    End process;


   Process

   Begin

     Reset <= '1';

     Wait for 2 ns;
```

```
          Reset <= '0';

         Wait for 200 ns;

         Wait;

       End process;

End tb;
```

Step 3. LFSR Design and Testbnech Simulation of Fig. 1 Circuit



Steps 1 and 2 are for optional exercise purposes.

Step 3 is a required task for this laboratory experiment.

You need to write a VHDL design and testbench for this 5-stage LFSR, and run simulations to verify that the value of Q (4 downto 0) repeats every 31 clock cycles.

CPE166 Lab 3 Part 3
By: Prof. Pang

Lab 3

Part 3: Algorithmic State Machine (ASM) Charts

This project is to implement the following ASM charts and run simulations using VHDL.

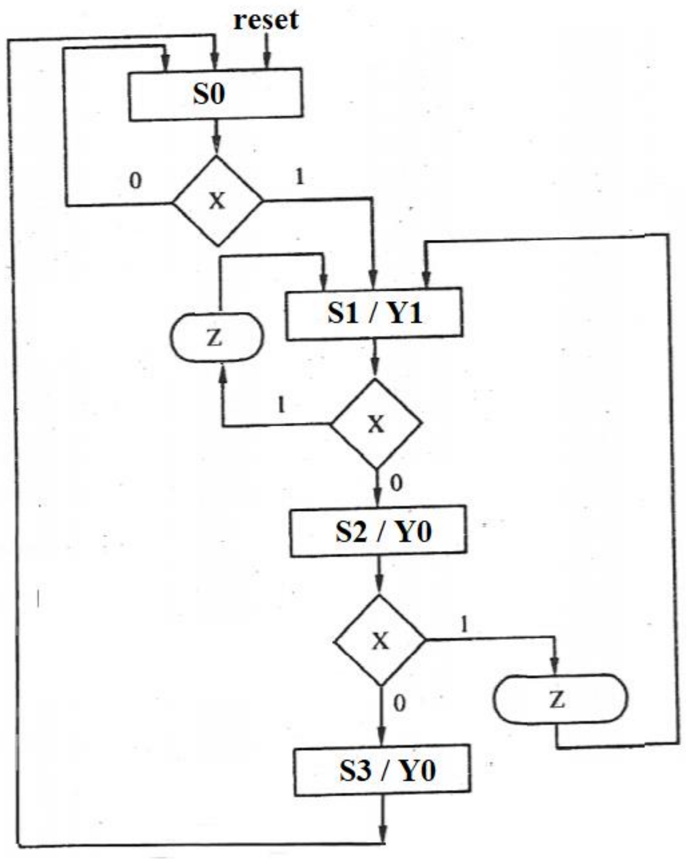


Figure 1. ASM diagram

Demo Requirement

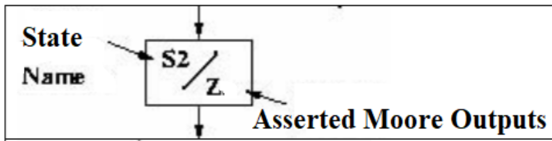
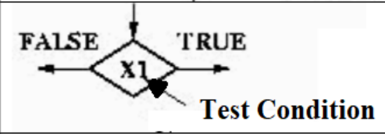
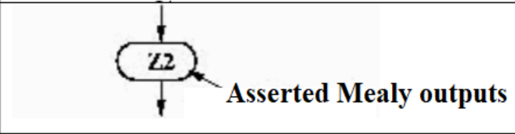
You need to demonstrate the final simulation waveform of this design to your lab instructor.

CPE166 Lab 3 Part 3
By: Prof. Pang

Lab Procedure

Step 1. ASM Review

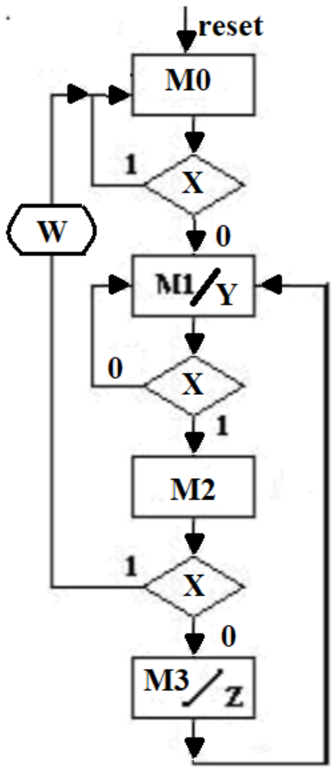
First, you need to review the definition of ASM charts you have learnt in class.

	One state box. The state box has a name and lists outputs that are asserted when the system is in that state. These outputs are called *Moore* type outputs.
	Optional decision box (es). A decision box may be conditioned on a signal or a test of some kind.
	Optional conditional output box (es). Such an output box indicates outputs that are conditionally asserted. These outputs are called *Mealy* outputs.

CPE166 Lab 3 Part 3
By: Prof. Pang

Step 2. VHDL design and testbench review

The ASM chart below has two moore outputs Y and Z. Y is asserted in the M1 state and Z is asserted in the M3 state. It has a mealy output, which is asserted when X is logic high in the M2 state.



```
Library ieee;

Use ieee.std_logic_1164.all;

entity chart is

        port (  reset, clk, x:   in std_logic;

                  y, z, w:  out std_logic ;

             ckcs, ckns: out std_logic_vector (1 downto 0));

end chart;

architecture beh of chart is

constant M0: std_logic_vector(1 downto 0) := "00";

constant M1: std_logic_vector(1 downto 0) := "01";
```

```
        constant M2: std_logic_vector(1 downto 0) := "10";

        constant M3: std_logic_vector(1 downto 0) := "11";

        signal   cs, ns:  std_logic_vector (1 downto 0);

        begin

               ckcs <= cs;

               ckns <= ns;

                process(reset, clk)

                begin

                  If ( reset = '1')  then

                     cs <= M0;

                  elsif (rising_edge(clk) ) then

                     cs <= ns;

                  end if;

               end process;

              process(cs, x)

              begin

                case (cs) is

                  when M0 => if (x='1') then

                                   ns <= M0;

                             else

                                   ns <= M1;

                             end if;

                  when M1 => if (x='1') then

                                   ns <= M2;

                             else

                                   ns <= M1;

                             end if;
```

```
                        when M2 => if (x= '0') then
                                         ns <= M0;
                                    else
                                         ns <= M3;
                                    end if;
                        when M3 => ns <= M1;
                        when others=> ns <= M0;
                       end case;
                  end process;
                  y <= '1' when ( cs = M1 ) else '0';
                  z <= '1' when ( cs = M3 ) else '0';
                  w <= '1' when ( ( cs = M2 ) and ( x=  '1' ) ) else '0';
         end beh;


         library IEEE;
         use IEEE.std_logic_1164.all;


         entity testbench is
         end testbench;


         architecture tb of testbench is
         component chart
               port (  reset, clk, x:   in std_logic;
                      y, z, w:  out std_logic;
             ckcs, ckns: out  std_logic_vector (1 downto 0));
         end component;
         signal reset, clk, x, y, z, w: std_logic;
         signal   cs, ns:  std_logic_vector (1 downto 0);
```

```
begin

  DUT: chart port map(reset, clk, x, y, z, w, cs, ns);

 process

  begin

     clk <= '0';

    wait for 5 ns;

    clk <= '1';

    wait for 5 ns;

 end process;


    x <= '1', '0' after 10 ns, '1' after 40 ns, '0' after 60 ns, '1' after 80ns, '0' after 120ns,  '1' after
160 ns, '0' after 200 ns, '1' after 300 ns, '0' after 350 ns;


     Process

     Begin

         Reset <= '1';

         Wait for 2 ns;

          Reset <= '0';

         Wait for 400 ns;

              Wait;

     End process;

End tb;
```

<u>Step 3. Final Project Work</u>

You need to write a VHDL design and testbench for the ASM charts shown in figure 1, and run simulations. Demonstrate your simulation waveform to your lab instructor.

CPE166 Lab 3 Part 4
By: Prof. Pang

Lab 4

Part 4: Stopwatch Design

This project is to design a stopwatch in VHDL by using the hierarchical design approach shown in the figure below.

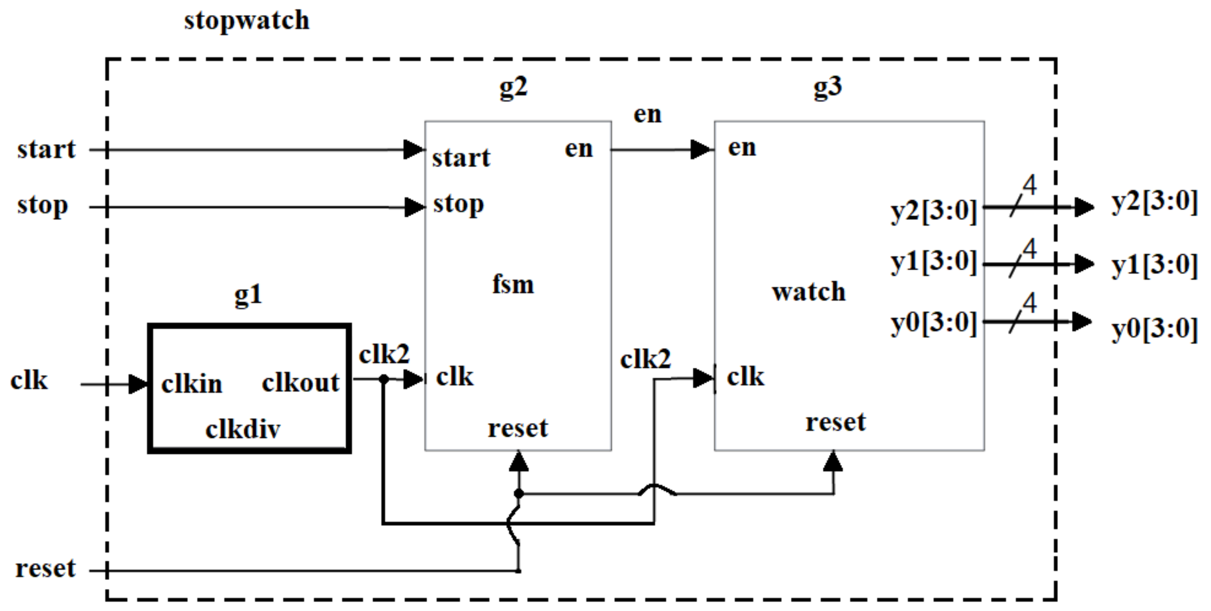


Figure 1. Stopwatch block diagram

The top-level I/O ports used in this design are shown below.

Table 1. I/O ports for the stopwatch design

Port Names	Port Direction	Port Size
start	Input	1
stop	Input	1
clk	Input	1
reset	Input	1
y3	Output	4
y2	Output	4
y1	Output	4
y0	Output	4

CPE166 Lab 3 Part 4
By: Prof. Pang

The stopwatch design has the following features:

1). The frequency of the input "clk" signal is 10 Hz.

2). At any time, if the "reset" input is logic high, the output of the stopwatch will be zero.

3). When the "start" input is logic high, the stopwatch will start counting.

4). When the "stop" input is logic high, the stopwatch will stop, but the stopwatch output will maintain its value. After that, when the "start" input is logic high again, the stopwatch will resume counting from its old value.

5). The output y2 y1 y0 is a 3-digit binary number, and each digit ranges from 0 to 9.

Assuming y2 = $(0110)_2$ = 6, y1 = $(0101)_2$ = 5, y0 = $(0111)_2$ = 7, this means 657 seconds.

If you press the "stop" button, the stopwatch will stop at 657 seconds and keep the value unchanged. After pressing the "start" button, the stopwatch will continue counting every second until: y2 = $(1001)_2$ = 9, y1 = $(1001)_2$ = 9, y0 = $(1001)_2$ = 9, which means 999 seconds. After 999 seconds, the stopwatch will return to 0 and then increase its value every second.

Demo Requirement

You need to demonstrate the final simulation waveform of the stopwatch.vhd design to your lab instructor.

CPE166 Lab 3 Part 4
By: Prof. Pang

Lab Procedure

Step 1. clkdiv Block

The clkdiv block gets the 10 Hz input clkin signal and generates the 1 Hz output clkout signal.

Design clkdiv.vhd and write a testbench to check your simulation results. The simulation waveform only needs to confirm that the output frequency is 10 times smaller than the input frequency.

Step 2. fsm Block

When the reset signal is logic high, the state machine enters the first idle state. Idle state means that the current watch is not counting, and the displayed output value is 0. Output "en" as 0 to disable counting. This can be represented in the state diagram below.

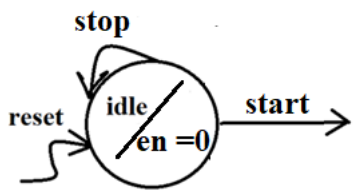


Figure 2. IDLE state of the fsm block

Figure 2 shows the state diagram of the idle state. In the idle state, the watch enable signal "en" is logic 0. if the "stop" button is pressed, the next state will still be the idle state. If the "start" button is pressed, the finite state machine needs to enter another state.

You can add more states to this design. In each state, you need to determine which state will become the next state based on the input signal values of "start", and "stop". In addition, you need to determine the output value of "en" based on whether you enable the watch to count or not.

To complete this step, you need to write fsm.vhd and testbench for simulation.

Step 3. watch Block

The watch block has three inputs: reset, en and 1 Hz clk. It has three 4-bit outputs: y2, y1, and y0.

1). When "reset" is logic high, the watch will output 0.

2). Only when "en" is logic high, the watch will increase its value every 1 second.

3). Each of the y2, y1 and y0 signals ranges from $(0000)_2$ to $(1001)_2$.

For example, if y2= $(0001)_2$ y1= $(1001)_2$ and y0= $(1001)_2$ at the current time, it means 199 seconds. After another 1 second, the watch needs to output y2= $(0010)_2$ y1= $(0000)_2$ and y0= $(0000)_2$ which means 200 seconds.

To complete this step, you need to write watch.vhd and testbench for simulation.

Step 3. Final stopwatch Block

The final stopwatch design is shown in Figure 1, with four 1-bit inputs, including reset, clk, start and stop, and three 4-bit outputs, including y2, y1, and y0.

In the architecture part of the design, two internal signals must be declared, including "en" and "clk2", and three components must be declared, including clkdiv, fsm and watch.

In addition, the main design part requires a clkdiv instance, an fsm instance and a watch instance. The two internal signals "en" and "clk2" are used for wiring of different modules.

Complete the final design and write testbench for simulations.

CPE166 Lab 3 Part 4
By: Prof. Pang

Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample VHDL Codes:

```
-- clk division circuit:  divide by 8

Library ieee;

Use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;

entity example1 is

       port (  clkin:   in std_logic;

               clkout:  out std_logic

         );

end example1;

architecture arch1 of example1 is

signal  cnt:  std_logic_vector (3 downto 0) := "0000";

begin

  process( clkin)

  begin

    if (rising_edge(clkin)) then

      if ( cnt = 7 )  then

        cnt <= (others=>'0');

        clkout <= '1';

      elsif (cnt < 3 ) then

        cnt <= cnt + 1;

        clkout <= '1';

      else

        cnt <= cnt + 1;

        clkout <= '0';
        end if;
      end if;
    end process;
end arch1;


library IEEE;
use IEEE.std_logic_1164.all;
entity testbench is
end testbench;
architecture tb of testbench is
component example1
         port (  clkin:   in std_logic;
                 clkout:  out std_logic
            );
end component;
signal clkin, clkout: std_logic;
begin
   DUT: example1 port map(clkin, clkout);
   clocking: process
   begin
        clkin <= '0';
        wait for 5 ns;
        clkin <= '1';
        wait for 5 ns;
     end process;


     End tb;
```

Sample VHDL Codes:

```
--  General counter: counting from 0 to 18 and then repeat.

Library ieee;

Use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;

entity example2 is

        port (  clk:   in std_logic;

               dout:  out std_logic_vector(4 downto 0)

           );

end example2;

architecture arch2 of example2 is

signal  cnt:  std_logic_vector (4 downto 0) := "00000";

begin

   process( clk)

   begin

     if (rising_edge(clk)) then

       if ( cnt = 18 )  then

          cnt <= (others=>'0');

       else

          cnt <= cnt + 1;

         end if;

       end if;

     end process;


    dout <= cnt;

end arch2;
```

CPE166 Lab 3 Part 4
By: Prof. Pang

```
library IEEE;

use IEEE.std_logic_1164.all;

entity testbench is

end testbench;

architecture tb of testbench is

component example2

        port (  clk:   in std_logic;

              dout:  out std_logic_vector(4 downto 0)

           );

end component;

signal clk:  std_logic;

signal dout: std_logic_vector(4 downto 0);

begin

   DUT: example2 port map(clk, dout);

   clocking: process

   begin

     clk <= '0';

      wait for 5 ns;

      clk <= '1';

      wait for 5 ns;

  end process;


end tb;
```