CPE166 Advanced Logic Design

Lab Number 3

Anthony Chavez

Table of Contents

Introduction

Lab 3 was a four-part lab that served as an introduction to circuit designing in VHDL. The four parts of the lab were a (7, 4) Hamming Code Generator, a Pseudorandom Number Generator, an Algorithmic State Machine (ASM) Charts, and a Stopwatch Design. All code and test bench simulations were written and tested in the Xilinx Vivado IDE.

Laboratory Report

Part 1 – (7, 4) Hamming Code Generator

Design Purpose:
The Hamming Code is used to correct a single bit error using extra parity bits. Hamming (7, 4) is a linear error-correcting code that encodes four bits of data into seven bits by adding three even parity bits. This project was used to become familiar with designing in VHDL.

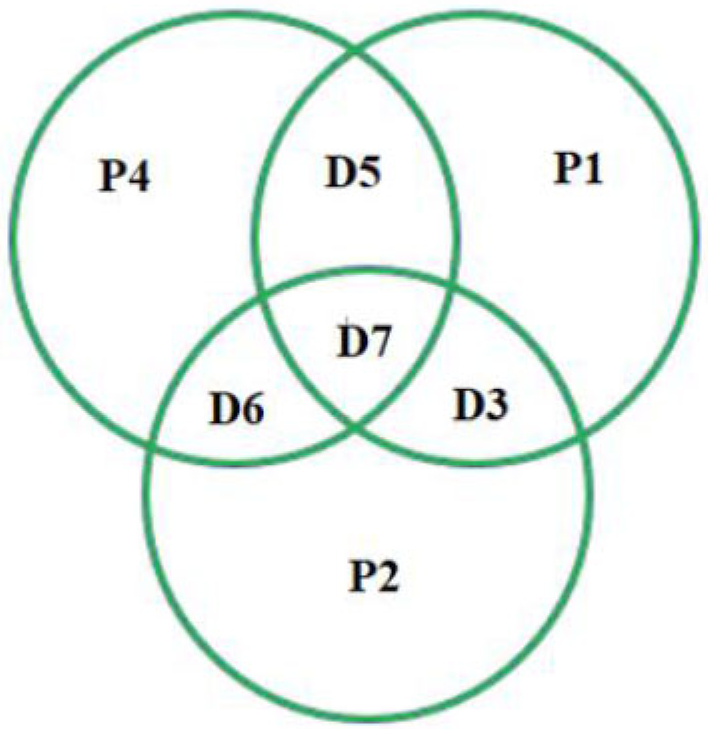


Figure 1. (7, 4) hamming code diagrams

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)

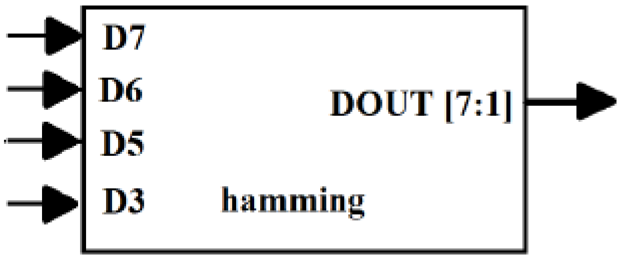


Figure 2. (7, 4) hamming code generator design block diagram

VHDL Design:

```
--------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/3/20
-- Module Name: MY_PACK - Behavioral
-- Description: Even Parity Package
-- Project Part Number: Lab 3, Part 1
--------------------------------------------------------------------------------


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;
```

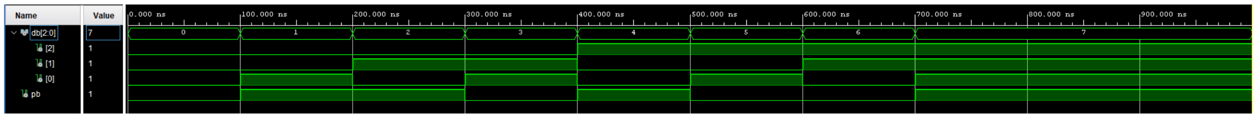
```
        begin
            TMP := D(0);
            for J in 1 to D'high loop
                TMP := TMP xor D(J);
            end loop; -- works for any size of D
        return TMP;
    end PARITY; -- even parity
end MY  PACK;
```

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/3/20
-- Module Name: PAR - Behavioral
-- Description: Even Parity Bit of 3-bit Input Data
-- Project Part Number: Lab 3, Part 1
----------------------------------------------------------------------------------



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;
```

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/5/20
-- Module Name: hamming - Behavioral
-- Description: (7,4) Hamming Code Generation Scheme
-- Project Part Number: Lab 3, Part 1
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity hamming is
    Port (D7, D6, D5, D3: in std_logic;
        DOUT: out std_logic_vector(6 downto 0));
end hamming;

architecture Behavioral of hamming is
    component PAR
        port(db : in std_logic_vector(2 downto 0);
             pb : out std_logic);
    end component;
begin
    DOUT(6)<=D7; DOUT(5)<=D6; DOUT(4)<=D5; DOUT(2)<=D3;
    PAR4: PAR port map(db(2)=>D7,db(1)=>D6,db(0)=>D5,pb=>DOUT(3));
    PAR2: PAR port map(db(2)=>D7,db(1)=>D6,db(0)=>D3,pb=>DOUT(1));
    PAR1: PAR port map(db(2)=>D7,db(1)=>D5,db(0)=>D3,pb=>DOUT(0));
end Behavioral;
```

VHDL Testbench Design and Simulation Waveforms:

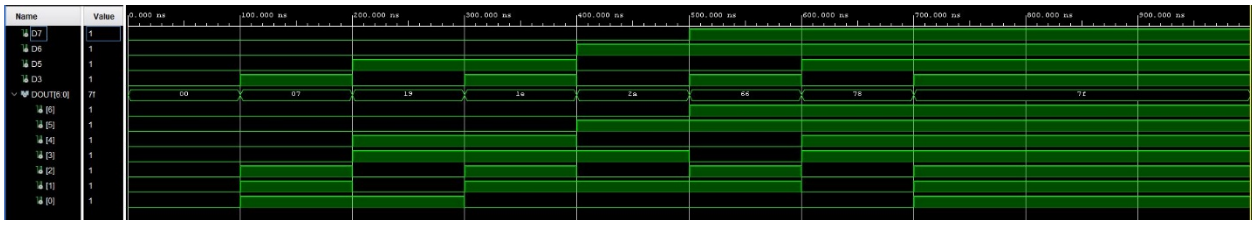
```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/3/20
-- Module Name: PAR_tb - Behavioral
-- Description: Testbench for PAR
-- Project Part Number: Lab 3, Part 1
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PAR_tb is
--  Port ( );
end PAR_tb;

architecture Behavioral of PAR_tb is
    component PAR
        port(db: in std_logic_vector(2 downto 0);
             pb: out std_logic);
    end component;
    signal db: std_logic_vector(2 downto 0);
    signal pb: std_logic;
    begin
        uut: PAR port map (db=>db, pb=>pb);
        process begin
            db <= "000";
            wait for 100 ns;
            db <= "001";
            wait for 100 ns;
            db <= "010";
            wait for 100 ns;
            db <= "011";
            wait for 100 ns;
            db <= "100";
            wait for 100 ns;
            db <= "101";
            wait for 100 ns;
            db <= "110";
            wait for 100 ns;
            db <= "111";
            wait for 100 ns;
            wait;
        end process;
end Behavioral;
```



This waveform was used to verify the generation of the even parity signal of the three-bit test data in VHDL. If the binary data bits (db) contains an odd number of 1's, then the parity bit (pb) will become logic high. If db contains an even number of 1's, then the parity bit will become logic low.

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/5/20
-- Module Name: hamming_tb - Behavioral
-- Description: (7,4) Hamming Code Generation Scheme testbench
-- Project Part Number: Lab 3, Part 1
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity hamming_tb is
--  Port ( );
end hamming_tb;

architecture Behavioral of hamming_tb is
```

```
    component hamming
        Port (D7, D6, D5, D3: in std_logic;
              DOUT: out std_logic_vector(6 downto 0));
    end component;
    signal D7, D6, D5, D3 : std_logic;
    signal DOUT : std_logic_vector(6 downto 0);
begin
    uut: hamming port map(D7=>D7,D6=>D6,D5=>D5,D3=>D3,DOUT=>DOUT);
    process begin
        D7<='0'; D6<='0'; D5<='0'; D3<='0';     -- 0000
        wait for 100 ns;
        D7<='0'; D6<='0'; D5<='0'; D3<='1';     -- 0001
        wait for 100 ns;
        D7<='0'; D6<='0'; D5<='1'; D3<='0';     -- 0010
        wait for 100 ns;
        D7<='0'; D6<='0'; D5<='1'; D3<='1';     -- 0011
        wait for 100 ns;
        D7<='0'; D6<='1'; D5<='0'; D3<='0';     -- 0100
        wait for 100 ns;
        D7<='1'; D6<='1'; D5<='0'; D3<='1';     -- 1101
        wait for 100 ns;
        D7<='1'; D6<='1'; D5<='1'; D3<='0';     -- 1110
        wait for 100 ns;
        D7<='1'; D6<='1'; D5<='1'; D3<='1';     -- 1111
        wait for 100 ns;
        wait;
    end process;

end Behavioral;
```



This waveform was used to verify the (7,4) Hamming Code. The three parity bits are P4 (DOUT(3)), P2 (DOUT(1)), and P1 (DOUT(0)). The four input bits are D7 (DOUT(6)), D6 (DOUT(5)), D5 (DOUT(4)), and D3 (DOUT(2)). More specifically, the parity bits will become logic high or logic low if there is an odd number of 1's in there respective regions.

Result Discussion:
Based on the final testbench, I was able to obtain the correct output. One problem I came across when writing the VHDL code for this project was understanding how to call components from other VHDL files. I went to office hours and was able correct my confusion.

Part 2 – Pseudorandom Number Generator

Design Purpose:
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. We will use this circuit to design a 5-stage LFSR circuit in the Figure below. We will make sure this LFSR design uses a non-zero value as the initial seed value and the value of Q(4 downto 0) repeats every 31 clock cycles.

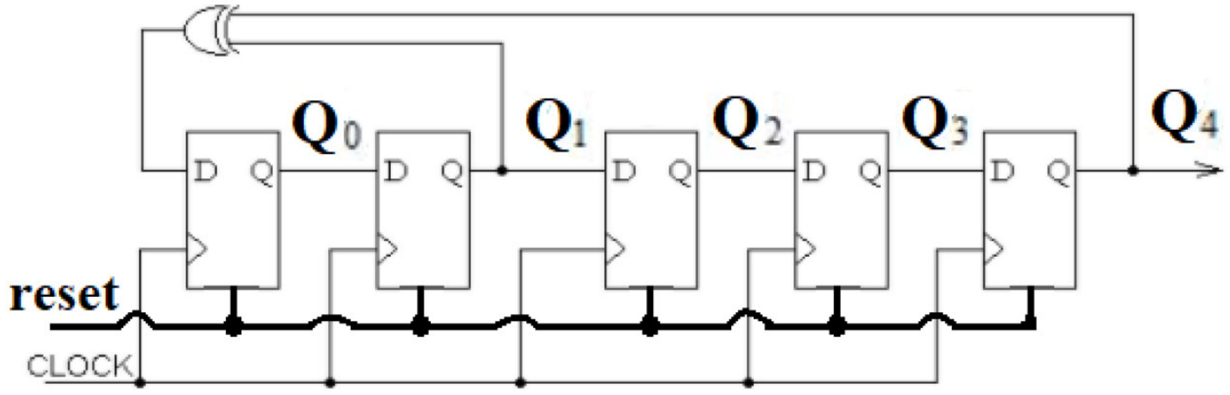


Figure 1. 5-stage LFSR diagram

VHDL Design:

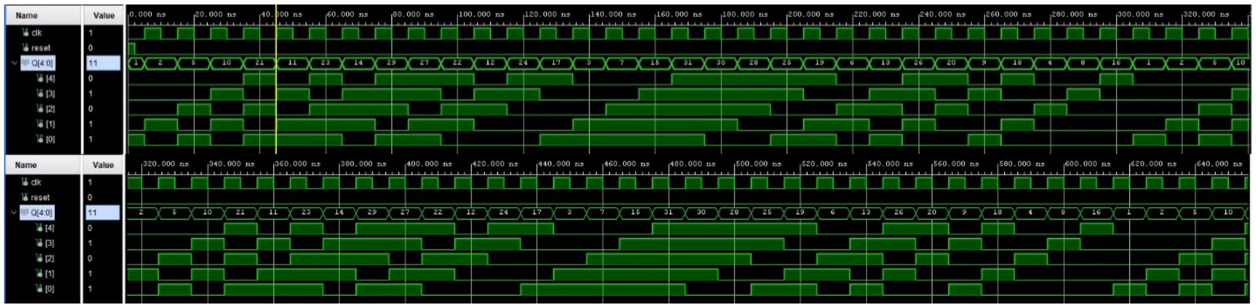
```
--------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/8/20
-- Module Name: pseudo_num_gen - Behavioral
-- Description: Pseudorandom Number Generator
-- Project Part Number: Lab 3, Part 2
--------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pseudo_num_gen is
   Port (reset, clk: in std_logic;
         Q: out std_logic_vector(4 downto 0));
end pseudo_num_gen;

architecture Behavioral of pseudo_num_gen is
   signal m: std_logic_vector(4 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(4 downto 1) <= m(3 downto 0);
                m(0) <= m(1) xor m(4);
        end if;
    end process;
    Q <= m;
end Behavioral;
```

VHDL Testbench Design and Simulation Waveforms:

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/8/20
-- Module Name: pseudo_num_gen_tb - Behavioral
-- Description: Pseudorandom Number Generator testbench
-- Project Part Number: Lab 3, Part 2
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pseudo_num_gen_tb is
end pseudo_num_gen_tb;

architecture Behavioral of pseudo_num_gen_tb is
    signal clk, reset: std_logic;
    signal Q: std_logic_vector (4 downto 0);
    component pseudo_num_gen
        Port ( reset, clk: in std_logic;
               Q: out std_logic_vector (4 downto 0) );
    end component;
    begin
        uut: pseudo_num_gen 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 Behavioral;
```



The first waveform shows the seed starting at 1 and the code begins to generate random numbers. However, after the 31st clock cycle, the sequence will repeat again. The second waveform picture shows a continuation of where the first picture ended.

Result Discussion:
Based on the testbench waveforms shown above, I was able to obtain the correct output. Pseudorandom Number Generator only changes the sequence if the seed is changed. If not, the same sequence will always be achieved.

Part 3 – Algorithmic State Machine (ASM) Charts

Design Purpose:
This project is to implement the following ASM charts and run simulations using VHDL. Any relevant material covered in class was used to understand the process of the ASM chart below and write the following program files.

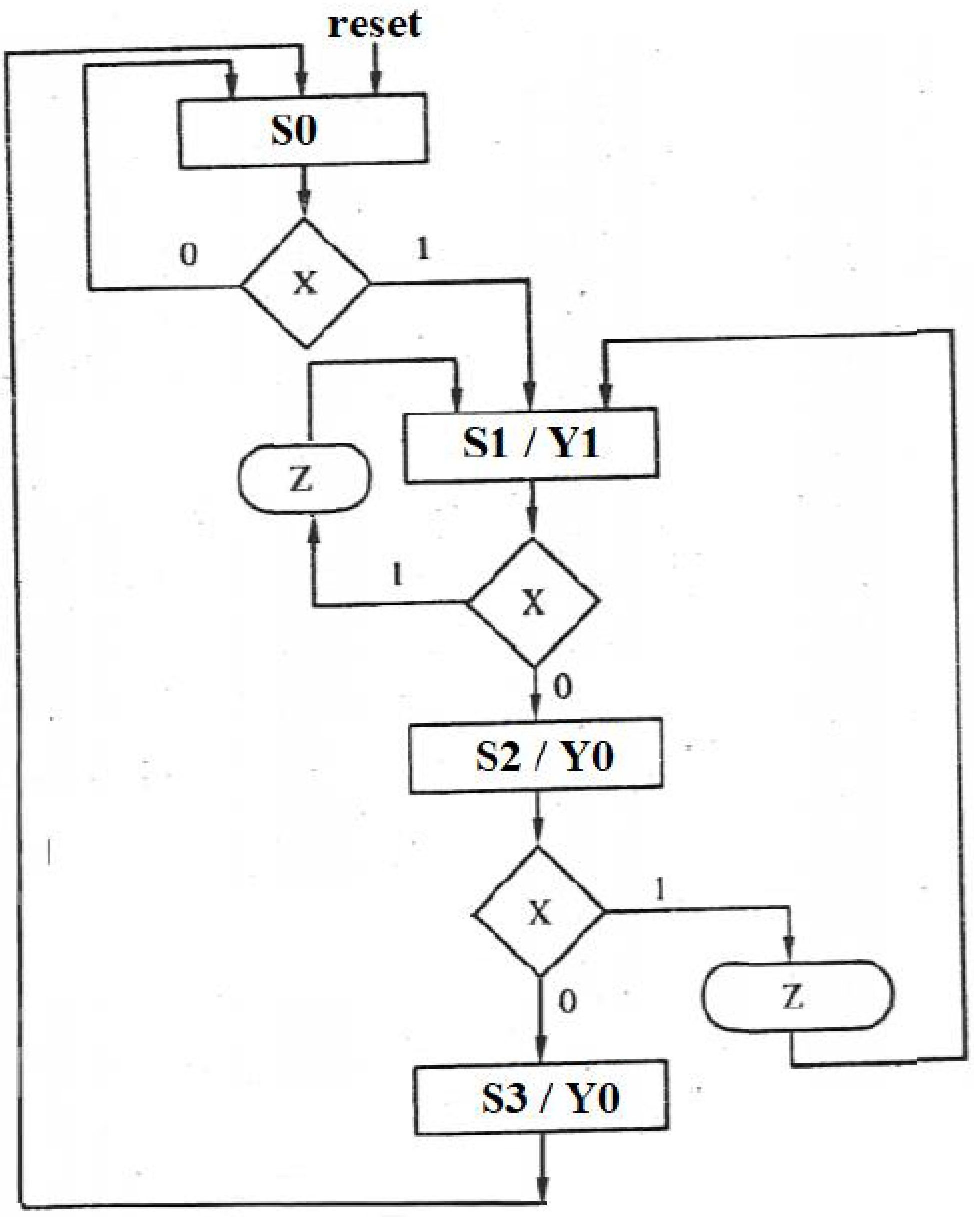


Figure 1. ASM diagram

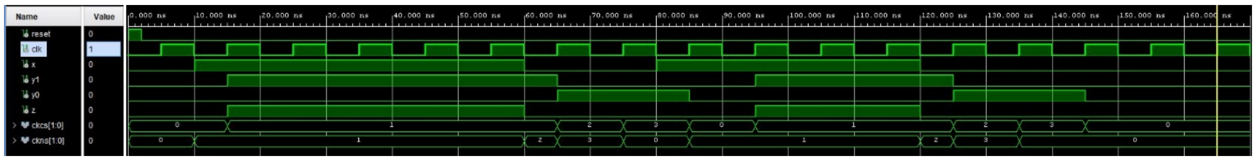
VHDL Design:

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/13/20
-- Module Name: chart - Behavioral
-- Description: ASM charts
-- Project Part Number: Lab 3, Part 3
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity chart is
    Port (reset, clk, x: in std_logic;
          y1, y0, z: out std_logic;
          ckcs, ckns: out std_logic_vector(1 downto 0));
end chart;

architecture Behavioral of chart is
    constant S0: std_logic_vector(1 downto 0) := "00";
    constant S1: std_logic_vector(1 downto 0) := "01";
    constant S2: std_logic_vector(1 downto 0) := "10";
    constant S3: 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 <= S0;
            elsif (rising_edge(clk)) then
                cs <= ns;
            end if;
        end process;
        process(cs, x)
        begin
            case (cs) is
                when S0 => if (x='1') then
                        ns <= S1;
                    else
                        ns <= S0;
                    end if;
                when S1 => if (x='0') then
                        ns <= S2;
                else
                   ns <= S1;
                end if;
             when S2 => if (x= '0') then
                   ns <= S3;
                else
                   ns <= S1;
                end if;
             when S3 => ns <= S0;
             when others=> ns <= S0;
             end case;
       end process;
       y1 <= '1' when (cs = S1) else '0';
       y0 <= '1' when ((cs = S2) or (cs = S3)) else '0';
       z <= '1' when (((cs = S1) and (x = '1')) or ((cs = S2) and (x = '1'))) else '0';
end Behavioral;
```

VHDL Testbench Design and Simulation Waveforms:

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/13/20
-- Module Name: testbench - Behavioral
-- Description: ASM charts testbench
-- Project Part Number: Lab 3, Part 3
------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity testbench is
end testbench;

architecture Behavioral of testbench is
   component chart
      port (reset, clk, x: in std_logic;
            y1, y0, z: out std_logic;
            ckcs, ckns: out std_logic_vector(1 downto 0));
   end component;
   signal reset, clk, x, y1, y0, z: std_logic;
   signal ckcs, ckns: std_logic_vector(1 downto 0);
   begin
      DUT: chart port map(reset, clk, x, y1, y0, z, ckcs, ckns);
      process
```

```
    begin
       clk <= '0';
       wait for 5 ns;
       clk <= '1';
       wait for 5 ns;
    end process;

    x <= '0',          -- S0
    '1' after 10 ns,   -- S1
    '1' after 40 ns,   -- S1
    '0' after 60 ns,   -- S2
    '1' after 80 ns,   -- S1
    '0' after 120 ns,  -- S2
    '0' after 160 ns;  -- S3 --> S0

    process
    begin
       reset <= '1';
       wait for 2 ns;
       reset <= '0';
       wait for 300 ns;
    wait;
    end process;
end Behavioral;
```



In the above waveform, all the possible outputs at each state were tested. At each positive rising edge of the clk, x will be checked for being asserted or unasserted and this will determine the output of the outputs Y0, Y1, and Z.

Result Discussion:
I was able to achieve the correct outputs for Y0, Y1, and Z when appropriate at each state in the ASM chart. This program was not too difficult since it is basically a finite state machine design. Thus, I used the vending machine program from Lab 2 Part 4 as a reference as well as utilizing the sample code provided in the lab instructions.

Part 4 – Stopwatch Design

Design Purpose:
This project is to design a stopwatch in VHDL by using the hierarchical design approach shown in the figure below. The top-level I/O ports used in this design are shown in the table below.

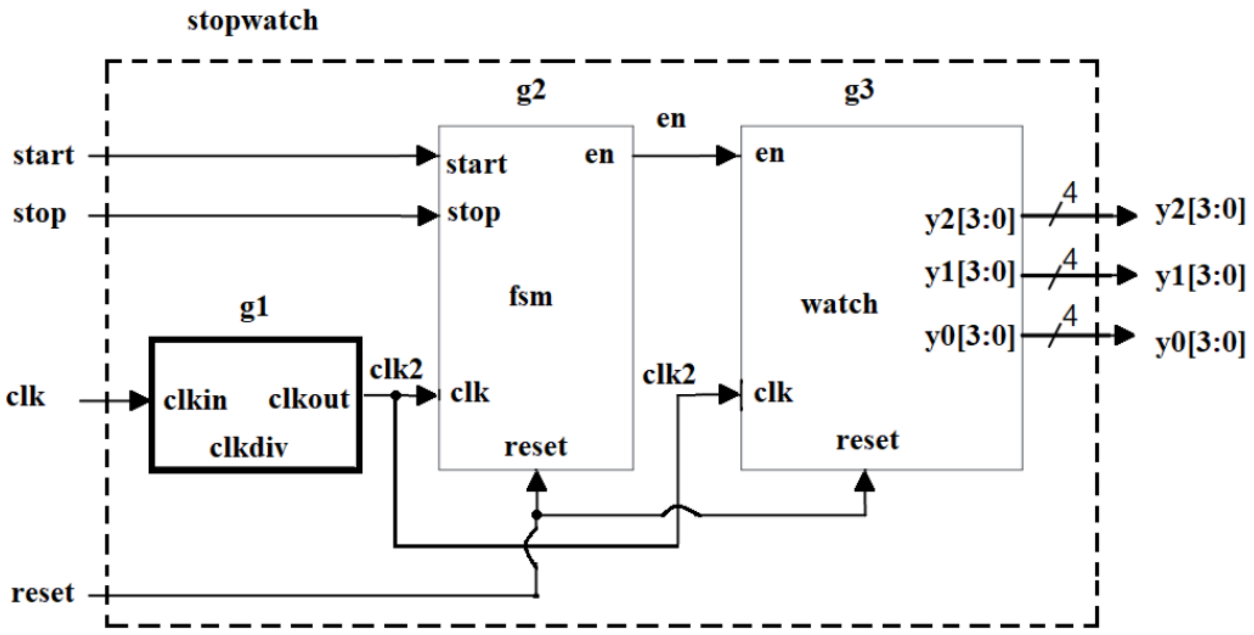


Figure 1. Stopwatch block diagram

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

The following features were implemented in the final stopwatch design:

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.

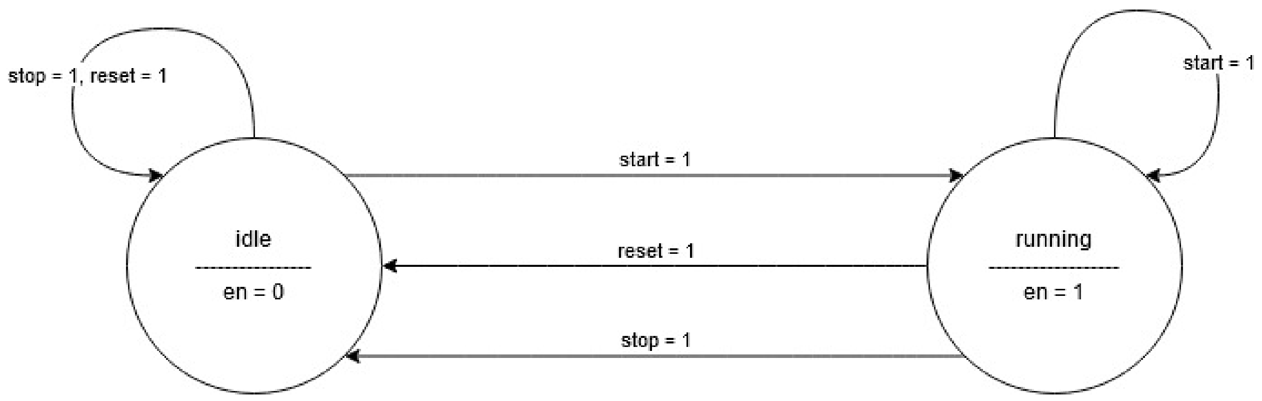
3). When the "start" input is logic high, the stopwatch will start counting.

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.

Shown below is my finite state machine design:



VHDL Design:

```
--------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/18/20
-- Module Name: clkdiv - Behavioral
-- Description: Clock Division
-- Project Part Number: Lab 3, Part 4
--------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity clkdiv is
   Port (clkin: in std_logic;
         clkout: out std_logic);
end clkdiv;

architecture Behavioral of clkdiv is
   signal cnt: std_logic_vector(3 downto 0) := "0000";
   begin
      process(clkin)
      begin
         if(rising_edge(clkin)) then
            if(cnt = 9) then
               cnt <= (others=>'0');
               clkout <= '1';
            elsif(cnt < 4) then
               cnt <= cnt + 1;
               clkout <= '1';
            else
               cnt <= cnt + 1;
               clkout <= '0';
            end if;
         end if;
      end process;
end Behavioral;
```

```
--------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: fsm - Behavioral
-- Description: Finite State Machine Circuit of stopwatch
-- Project Part Number: Lab 3, Part 4
--------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm is
   Port (clk, start, stop, reset: in std_logic;
       en: out std_logic);
end fsm;

architecture Behavioral of fsm is
   constant S0: std_logic_vector(1 downto 0) := "00"; -- idle
   constant S1: std_logic_vector(1 downto 0) := "01"; -- running
   signal cs, ns: std_logic_vector(1 downto 0);
   begin
      process(reset, clk)
      begin
         if(reset = '1') then
            cs <= S0;
         elsif (rising_edge(clk)) then
            cs <= ns;
         end if;
      end process;
      process(cs, start, stop)
      begin
         case(cs) is
            when S0 => if (start='1') then
               ns <= S1;
               else
                  ns <= S0;
               end if;
            when S1 => if (stop='1') then
               ns <= S0;
               else
                  ns <= S1;
               end if;
            when others => ns <= S0;
         end case;
      end process;
      en <= '1' when (cs = S1 and start = '1') else '0';
end Behavioral;
```

```
--------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: watch - Behavioral
-- Description: watch logic of stopwatch
-- Project Part Number: Lab 3, Part 4
--------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity watch is
   Port (en, clk, reset: in std_logic;
         y0, y1, y2: out std_logic_vector(3 downto 0));
end watch;

architecture Behavioral of watch is
   signal y0_reg: std_logic_vector(3 downto 0);
   signal y1_reg: std_logic_vector(3 downto 0);
   signal y2_reg: std_logic_vector(3 downto 0);
   begin
      process(clk, reset)
      begin
         if(reset = '1') then
            y0_reg <= (others=>'0');
            y1_reg <= (others=>'0');
            y2_reg <= (others=>'0');
            elsif rising_edge(clk) then
               if(y0_reg = 9) then
                  y0_reg <= (others=>'0');
               elsif(en = '1') then
                  y0_reg <= y0_reg + 1;
            end if;
            if(y0_reg = 9) then
               if(y1_reg = 9) then
                  y1_reg <= (others=>'0');
               else
                  y1_reg <= y1_reg + 1;
               end if;
            end if;
            if(y0_reg = 9 and y1_reg = 9) then
               if(y2_reg = 9) then
                  y2_reg <= (others=>'0');
               else
                  y2_reg <= y2_reg + 1;
```

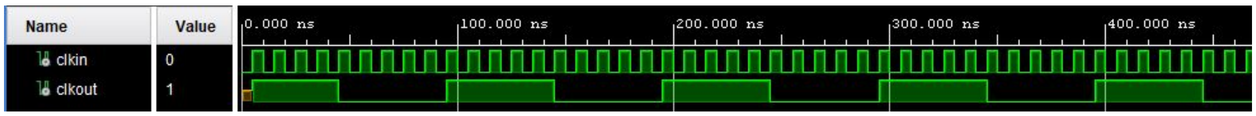
```
                end if;
            end if;
        end if;
    end process;
    y0 <= y0_reg when (en = '1') or (reset = '1');
    y1 <= y1_reg when (en = '1') or (reset = '1');
    y2 <= y2_reg when (en = '1') or (reset = '1');
end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: stopwatch - Behavioral
-- Description: stopwatch
-- Project Part Number: Lab 3, Part 4
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity stopwatch is
   Port (start, stop, clk, reset: in std_logic;
         y0, y1, y2: out std_logic_vector(3 downto 0));
end stopwatch;

architecture Behavioral of stopwatch is
   signal en, clk2: std_logic;
   component clkdiv
      port (clkin: in std_logic;
            clkout: out std_logic);
   end component;
   component fsm
      port (clk, start, stop, reset: in std_logic;
            en: out std_logic);
   end component;
   component watch
      port (en, clk, reset: in std_logic;
            y0, y1, y2: out std_logic_vector(3 downto 0));
   end component;
```

```
    begin
        g1: clkdiv port map(clkin=>clk, clkout=>clk2);
        g2: fsm port map(clk=>clk2, start=>start, stop=>stop, reset=>reset, en=>en);
        g3: watch port map(en=>en, clk=>clk2, reset=>reset, y0=>y0, y1=>y1, y2=>y2);
end Behavioral;
```

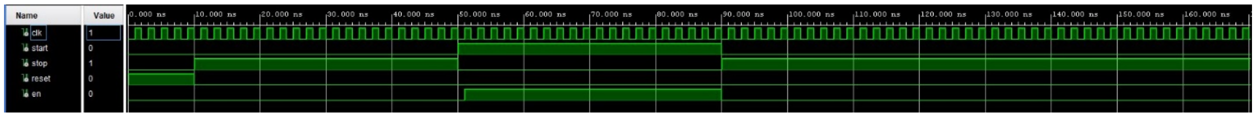
VHDL Testbench Design and Simulation Waveforms:

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/18/20
-- Module Name: clkdiv_tb - Behavioral
-- Description: Clock Division Testbench
-- Project Part Number: Lab 3, Part 4
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clkdiv_tb is
end clkdiv_tb;

architecture Behavioral of clkdiv_tb is
    component clkdiv
        port(clkin: in std_logic;
             clkout: out std_logic);
    end component;
    signal clkin, clkout: std_logic;
    begin
        DUT: clkdiv port map(clkin, clkout);
        clocking: process
        begin
            clkin <= '0';
            wait for 5 ns;
            clkin <= '1';
            wait for 5 ns;
        end process;
end Behavioral;
```



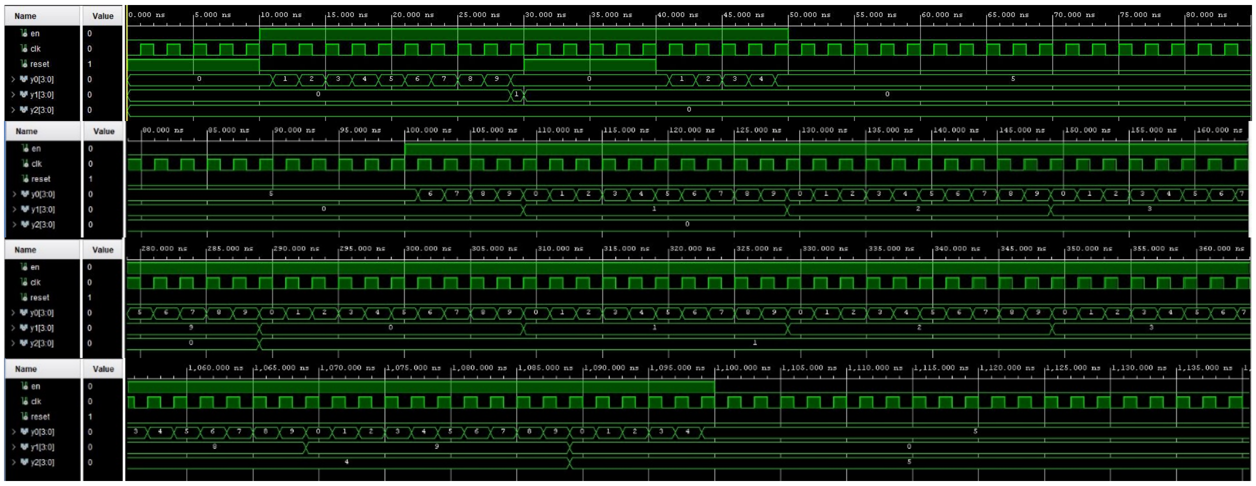
This testbench was used to test the clock division circuit of the stopwatch design. As can be seen in the waveform, the clkin is running at 1 ns logic high and becomes logic low for 1 ns. We want the clkout to be divided by 10 so the clk should be logic high for 5 ns and logic low for 5 ns which we can see in the above waveform under clkout.

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: fsm_tb - Behavioral
-- Description: Finite State Machine Circuit of stopwatch testbench
-- Project Part Number: Lab 3, Part 4
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm_tb is
end fsm_tb;

architecture Behavioral of fsm_tb is
    component fsm
        port (clk, start, stop, reset: in std_logic;
              en: out std_logic);
    end component;
    signal clk, start, stop, reset, en: std_logic;
    begin
        DUT: fsm port map(clk, start, stop, reset, en);
        process
        begin
            clk <= '0';
            wait for 1 ns;
            clk <= '1';
            wait for 1 ns;
        end process;

        process
        begin
            start <= '0'; -- initailize start
            stop <= '0';  -- initialize stop
            reset <= '1';
            wait for 10 ns;
            reset <= '0';
            stop <= '1';  -- S0
            wait for 40 ns;
            stop <= '0';
            start <= '1'; -- S1
            wait for 40 ns;
            start <= '0';
            stop <= '1';  -- S0
            wait for 300 ns;
            wait;
        end process;
end Behavioral;
```



This testbench was used to test the finite state machine component of the stopwatch design. When the reset or stop signal is logic high the enable signal (en) should be logic low. This means the stopwatch is in the idle state and no counting should occur. When the start signal becomes logic high, the enable signal should be logic high. This means the stopwatch is in the running state and counting will occur. Both these conditions are met in the above waveform.

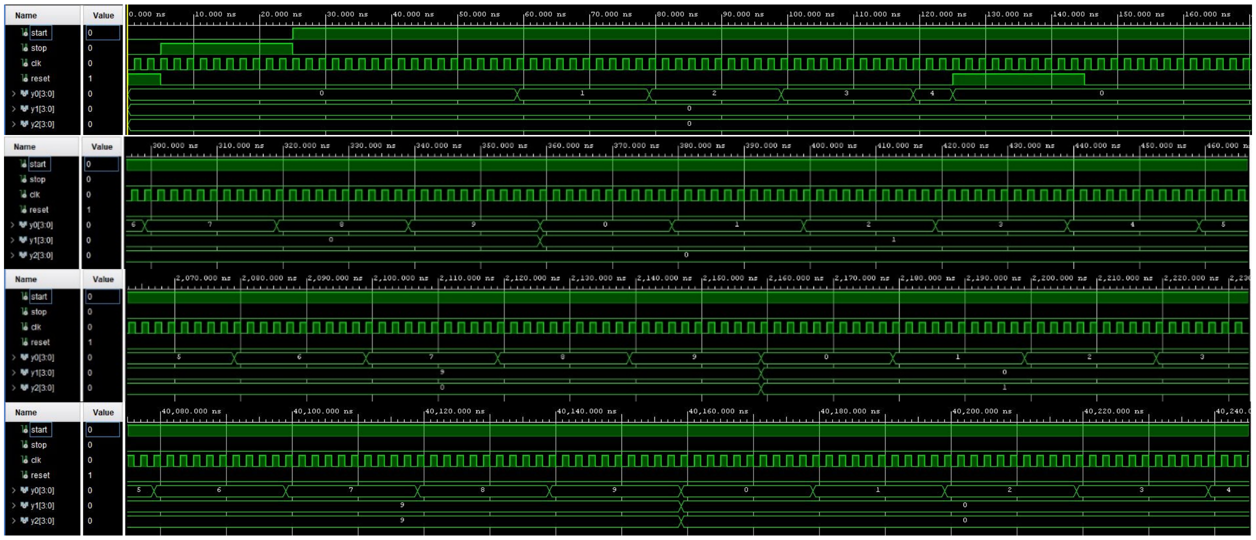
```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: watch_tb - Behavioral
-- Description: watch circuit testbench
-- Project Part Number: Lab 3, Part 4
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity watch_tb is
end watch_tb;

architecture Behavioral of watch_tb is
    component watch
        port (en, clk, reset: in std_logic;
              y0, y1, y2: out std_logic_vector(3 downto 0));
    end component;
    signal en, clk, reset: std_logic;
    signal y0, y1, y2: std_logic_vector(3 downto 0);
    begin
        DUT: watch port map(en, clk, reset, y0, y1, y2);
        process
        begin
            clk <= '0';
            wait for 1 ns;
            clk <= '1';
            wait for 1 ns;
        end process;

        process
        begin
            en <= '0';  -- start watch in idle state
            reset <= '1';  -- initialize reset
            wait for 10 ns;
            reset <= '0';
            en <= '1';
            wait for 20 ns;
            reset <= '1';
            wait for 10 ns;
            reset <= '0';
            wait for 10 ns;
            en <= '0';
            wait for 50 ns;
            en <= '1';
            wait for 1000 ns;
            en <= '0';
            wait;
        end process;
 end Behavioral;
```



This testbench was used to test the watch component of the stopwatch design. The component is responsible for the counting mechanism. In the first waveform, we see the watch component properly starts counting when the en signal is logic high and when a reset signal is detected, the outputs are all zeroed out. When the en signal is logic low, the counting stops until the en signal is logic high again. Notice how the counting output doesn't change between 50 ns to 100 ns.

When the y0 output reaches 9, the y1 output is incremented by one. Same thing when y1 output reaches 9, the y2 output is incremented by one. However, when the outputs y0, y1, and y2 are all 9, the outputs are reset to zero and the counting starts over again.

```
----------------------------------------------------------------------------------
-- Author: Anthony Chavez
-- Date Last Revised: 10/19/20
-- Module Name: stopwatch_tb - Behavioral
-- Description: stopwatch testbench
-- Project Part Number: Lab 3, Part 4
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity stopwatch_tb is
end stopwatch_tb;

architecture Behavioral of stopwatch_tb is
    component stopwatch
        port (start, stop, clk, reset: in std_logic;
              y0, y1, y2: out std_logic_vector(3 downto 0));
    end component;
    signal start, stop, clk, reset: std_logic;
    signal y0, y1, y2: std_logic_vector(3 downto 0);
    begin
        DUT: stopwatch port map(start, stop, clk, reset, y0, y1, y2);
        process
        begin
            clk <= '0';
            wait for 1 ns;
            clk <= '1';
            wait for 1 ns;
        end process;

        process
        begin
            start <= '0'; -- initailize start
            stop <= '0';  -- initialize stop
            reset <= '1'; -- initialize outputs
            wait for 5 ns;
            reset <= '0';
            stop <= '1';
            wait for 20 ns;
            stop <= '0';
            start <= '1';
            wait for 100 ns;
            reset <= '1';
            wait for 20 ns;
            reset <= '0';
            --wait for 800 ns;
            wait;
        end process;
 end Behavioral;
```



This testbench checks the final stopwatch design. When the reset signal is logic high, the outputs y0, y1, and y2 are zeroed out. When the stop signal is logic high all counting stops until the start signal is logic high and the stop signal is logic low. When the output y0 reaches 9, y1 will be incremented by 1 and y0 will reset to 0. When the output y1 reaches 0, y2 will be incremented by 1 and y1 will reset to 0. When all the outputs y0, y1, and y2 are 9, the outputs are reset to zero and the counting starts over again.

Result Discussion:
Based on the final waveform of the stopwatch design, I was able to verify that I achieved the correct output. All the supported features listed in the design purpose section were met as shown in the waveforms. One problem I came across when designing this project was not being able to identify a timing issue in my finite state machine component. I asked questions during lab time and was able to learn how to add new signal to my waveform. This allowed me to see when certain signals were triggering certain outputs to occur.

Conclusion

Overall, I think the four parts of the lab helped me understand how to write VHDL programs. The hamming code generator was fairly easy to understand which allowed me to become familiar with writing VHDL programs. I learned how to properly define the entity portion of the program and eventually figured out how to set up the architecture portion of the program. Some differences in the syntax was difficult to get used to since switching from Verilog. For example I had to get used to using "<=" when assigning values to signals and writing a "=" instead of a "==" when writing conditional statements. The stopwatch project helped bring together all the fundamentals and I was able to achieve the desired results.