Laboratory 03

CPE 166 Advanced Logic Design

Section 03

Vigomar Kim Algador

Fall 2022

TABLE OF CONTENT

Introduction — **3**

Part 1: (7,4) Hamming Code Generator — **4 - 7**

Design Purpose and Engineering Data 4

Step 1: Parity Function 5

Step 2: Even Parity Bit of 3-bit Input Data 5

Step 3: Final (7,4) Hamming Code Generator 6

Result Discussion 7

Part 2: 8 – Pseudorandom Number Generator — **8 – 9**

Design Purpose and Engineering Data 8

LFSR Design Review 8

LFSR Design 9

Result Discussion 9

Part 3: Algorithmic State Machine (ASM) Charts — **10 - 11**

Design Purpose and Engineering Data 10

ASM Design 10

Result Discussion 11

Part 4: Stopwatch Design — **12 - 17**

Design Purpose and Engineering Data 12

Step 1: clkdiv block 13

Step 2: fsm block 13

Step 3: watch block 15

Step 4: Final stopwatch block 16

Result Discussion 17

Conclusion — **18**

INTRODUCTION

This laboratory is an introduction on VHDL design using Vivado. This laboratory is divided into 4 parts. For the first part of this laboratory, I will be designing a (7,4) Hamming Code generator discussing error detection and parity in which by definition is a linear error-correcting code. For the second part of the laboratory, I will be designing a Pseudorandom Number Generator with the use of Linear Feedback Shift Register (LFSR). For the next part, I will be designing an Algorithmic State Machine (ASM) chart. I will be applying the knowledge learnt with the concept of finite state machine from the previous lecture and additional information such as state blocks, conditional blocks, and decision blocks. For the final part, I will be designing a stopwatch using a hierarchical approach design with the use of clock division, finite state machine, and the watch concept.

PART 1: (7,4) HAMMING CODE GENERATOR

Design Purpose and Engineering Data

This part of the laboratory introduced to student about the concept of error detection and parity. This topic, (7,4) Hamming Code, is a linear error-correcting code that encodes four bits of data into seven bits by adding three even parity bits. The key is to use of extra parity bits to allow the correction of a single bit error. Below is the Venn Diagram for the Hamming Code.

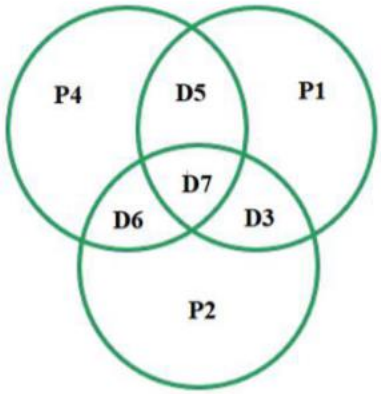

Figure 1. (7,4) Hamming Code Diagram

From the figure above, I identified the designated even parity bit (P1, P2, and P4) of binary data (D7, D6, D5, D3). To tell which are designated to is depending on the even parity such as P4 with the input data of D5, D7, and D6 that were enclosed in one circle. Understanding the concept led me to know the interface of the design which is shown below.

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)

Step 1: Parity Function

The first step is to know the concept of even parity bit that will be used to ensure the total number of bits (including parity bits) in the message is even for error detection. Using the provided package from the laboratory manual, I followed the code and add to a new file named MY_PACK.vhd.

```
1  -- Vigomar Kim Algador
2  -- CpE166 Section 03
3  -- Laboratory 03 Part 01
4  -- Parity Function
5  -- MY_PACK.vhd
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 package MY_PACK is
11     function PARITY (D : std_logic_vector) -- declaration of function
12     return std_logic;
13 end MY_PACK;
14
15 package body MY_PACK is
16     function PARITY (D : std_logic_vector) -- implementation of function inside the package body
17     return std_logic is
18     variable TMP : std_logic;
19     begin
20         TMP := D(0);
21         for J in 1 to D'high loop
22             TMP := TMP xor D(J);
23         end loop; -- works for any size of D
24         return TMP;
25     end PARITY; -- even parity
26 end MY_PACK;
```

Figure 2. MY_PACK VHDL Code

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

In the next step, I need to create a file named PAR.vhd using the source code provided in the laboratory manual. This part shows the directly call of the PARITY function to obtain the even parity bit value of the three-bit input data. After using the provided source code, I was able to write the testbench and able to generate the waveform result for it shown below.

Figure 3. PAR VHDL (top left), PAR testbench (top right), and simulation (bottom)

Step 3: (7,4) Hamming Code

Finally, I will now write the hamming source code for this part of the laboratory connecting with the two other files MY_PACK and PAR in VHDL. Below is the VHDL code as well as the testbench and the final simulation result.

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 01
-- (7,4) Hamming Code
-- HAMMING.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.MY_PACK.all;

entity HAMMING is
port( D7, D6, D5, D3: in std_logic;
DOUT: out std_logic_vector (7 downto 1));
end HAMMING;

architecture Behavioral of HAMMING is
signal db1, db2, db4: std_logic_vector(2 downto 0);
signal pb1, pb2, pb4: std_logic;

begin
    db1 <= D7 & D5 & D3;
    db2 <= D7 & D6 & D3;
    db4 <= D5 & D6 & D7;

    pb1 <=  parity(db1);
    pb2 <= parity(db2);
    pb4 <= parity(db4);

    DOUT <= D7 & D6 & D5 & pb4 & D3 & pb2 & pb1;
end Behavioral;
```

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 01
-- (7,4) Hamming Code Testbench
-- HAMMING_tb.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture testbench of HAMMING_tb is
component HAMMING
Port (D7, D6, D5, D3: in std_logic;
      DOUT: out std_logic_vector(7 downto 1));
end component;

signal D7, D6, D5, D3 : std_logic;
signal DOUT : std_logic_vector(7 downto 1);

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 testbench;
```

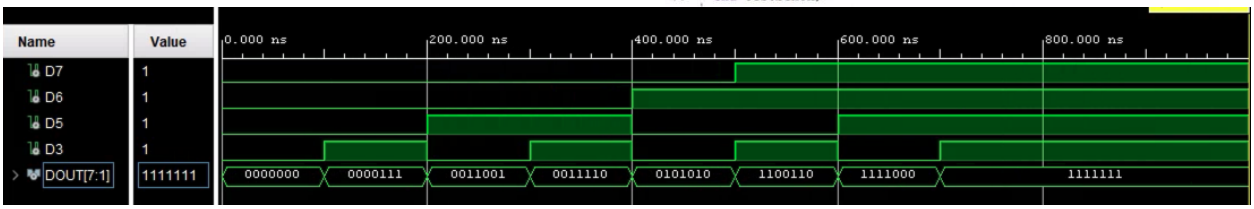

Figure 4. HAMMING VHDL (top left), testbench (top right), and simulation (bottom)

Result Discussion

For this part of this laboratory, I was able to design the (7,4) Hamming Code. From the figure 4 simulation, we can see the result of our Hamming Code VHDL with different inputs of D7, D6, D5, and D3. Analyzing the result simulation, we can check one of the inputs. When D5 and D3 is '1' which the input will be "0011", we can see that DOUT is "0011110" as P4 become '1' with the D5 only '1' while D7 and D6 are '0' to make it even and P2 become '1' with D3 only '1' while D7 and D6 are also '0'. P1 remains '0' as D5 and D3 are '1' which makes it even. In this case, we prove not just by conceptual calculation to the resulting simulation.

PART 2: PSEUDORANDOM NUMBER GENERATOR

Design Purpose and Engineering Data

For this part of the laboratory, I will be using the concept of linear feedback shift register (LFSR) for the pseudorandom number generator project. LFSR is a shift register whose input bit is the output of a linear logic function of two or more of its previous states. In this part, I will be designing 5-stage LFSR circuit which is shown below.

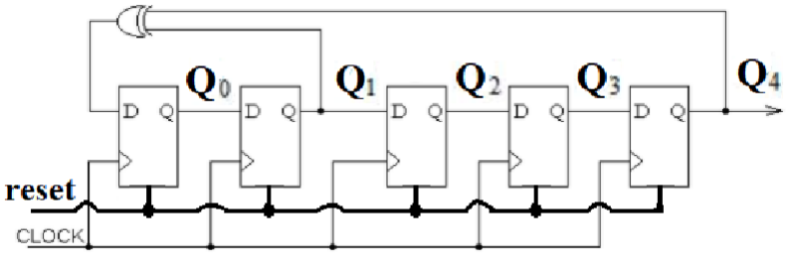

Figure 5. 5-stage LFSR diagram

LFSR Design Review

Before I started with the actual designing of 5-stage LFSR, I review the LFSR diagram that have learnt in the class with the 4-stage LFSR circuit which is shown below.

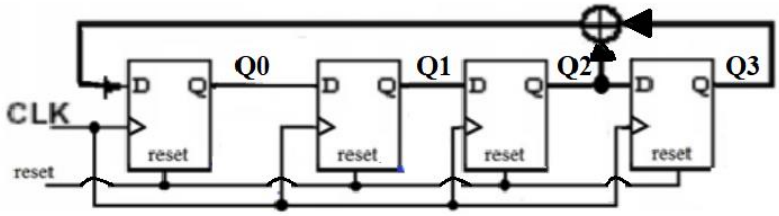

Figure 6. 4-stage LFSR diagram

Additionally, I was able to copy the given VHDL and testbench from the laboratory manual to analyze the simulation result shown below.

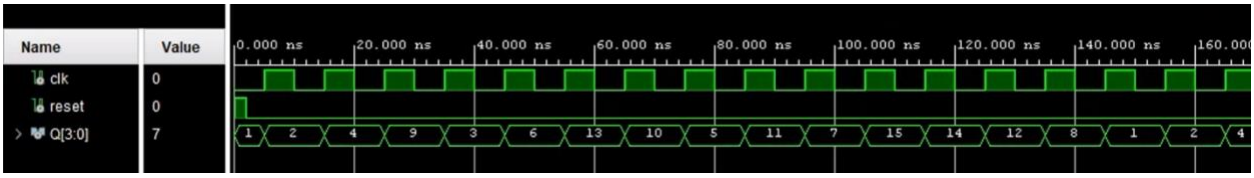

Figure 7. Simulation Result for a 4-stage LFSR circuit

LFSR Design

After reviewing the 4-stage LFSR diagram and VHDL, I was able to design similarly with 5-stage LFSR circuit. With some few changes, I was able to design and generate the 5-stage LFSR shown below.

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 02
-- lsfr.vhd
-- Psuedorandom Number Generator

Library ieee;
Use ieee.std_logic_1164.all;

Entity lfsr is
Port(reset, clk: in std_logic;
     Q: out std_logic_vector (4 downto 0));
End lfsr;

Architecture beh of lfsr is
Signal m: std_logic_vector (4 downto 0);
Begin
Process(reset, clk)
Begin
    if( reset = '1') then
        m <= ( 0=> '1', others =>'0');
    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 beh;
```

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 02
-- lsfr_tb.vhd
-- Psuedorandom Number Generator Testbench

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 (4 downto 0);
component lfsr
Port(reset, clk: in std_logic;
     Q: out std_logic_vector (4 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;
```

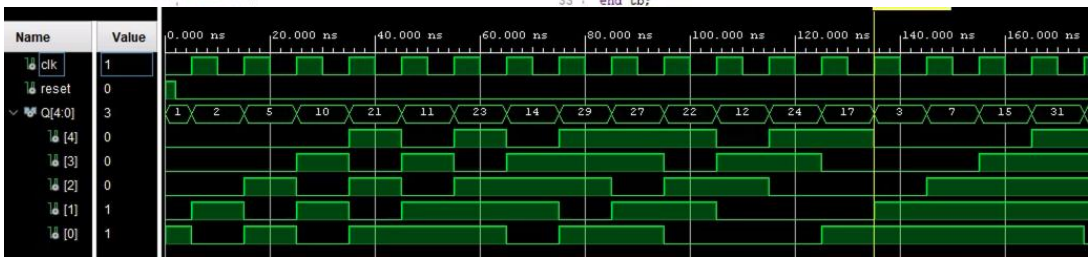

Figure 8. LFSR VHDL (top left), testbench (top right), and simulation (bottom)

Result Discussion

For this part of the laboratory, I was able to learn and analyze the concept of LFSR and the use of it for pseudorandom number generator. The 4-stage LFSR helps me to design the 5-stage LFSR and understand behind the concept. Additionally, the diagram is really helpful to where put the exclusive or. I was also able to verify the value of Q repeats every 31 clock cycles which is shown below.

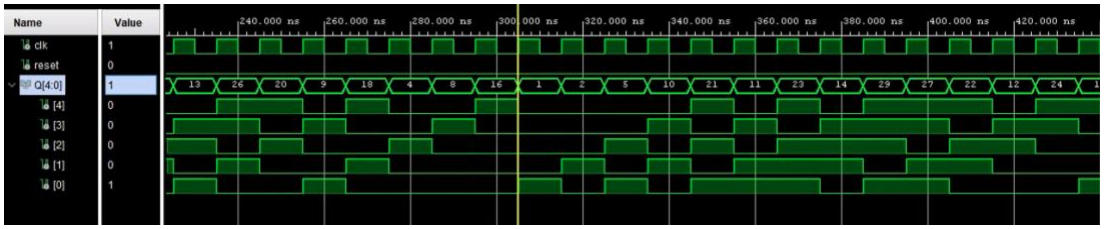

Figure 9. LFSR simulation at location where Q started over again

PART 3: ALGORITHMIC STATE MACHINE (ASM) CHARTS

Design Purpose and Engineering Data

For this part of the laboratory, I will be learning and implementing algorithmic state machine chart (ASM) with the given diagram below.

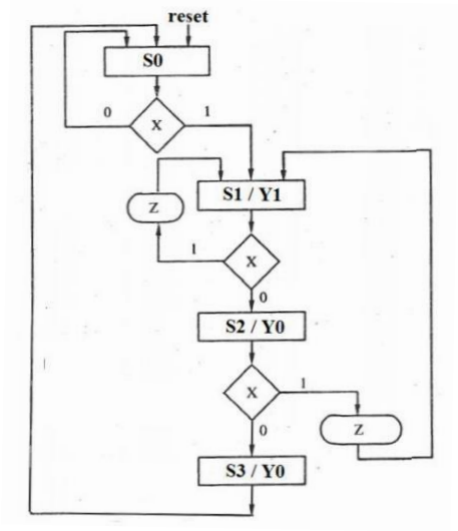

Figure 10. ASM Chart

Learning from the lecture class, I was able to review the definition in each box such as state, decision, and conditional.

ASM Design

After analyzing the chart, I was able to design the VHDL as well as testbench shown below.

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 03
-- ASM Chart
-- chart.vhd

Library ieee;
Use ieee.std_logic_1164.all;

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

architecture beh 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='1') then
        ns <= S1;
        else
        ns <= S2;
        end if;
    when S2 => if (x= '1') then
        ns <= S1;
        else
        ns <= S3;
        end if;
    when S3 => ns <= S0;
    when others=> ns <= S0;
end case;
end process;

y(0) <= '1' when ((cs=S2) or (cs=S3)) else '0';
y(1) <= '1' when (cs=S1) else '0';
z(0) <= '1' when ((cs=S1) and (x= '1')) else '0';
z(1) <= '1' when ((cs=S2) and (x='1')) else '0';
end beh;
```

Figure 11. ASM VHDL

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 03
-- ASM Chart testbench
-- testbech.vhd

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

signal reset, clk, x: std_logic;
signal y, z: std_logic_vector (1 downto 0);
signal ckcs, ckns: std_logic_vector (1 downto 0);

begin
DUT: chart port map(reset, clk, x, y(1 downto 0), z(1 downto 0), ckcs, ckns);

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 120 ns,
     '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;
```

Figure 12. ASM testbench

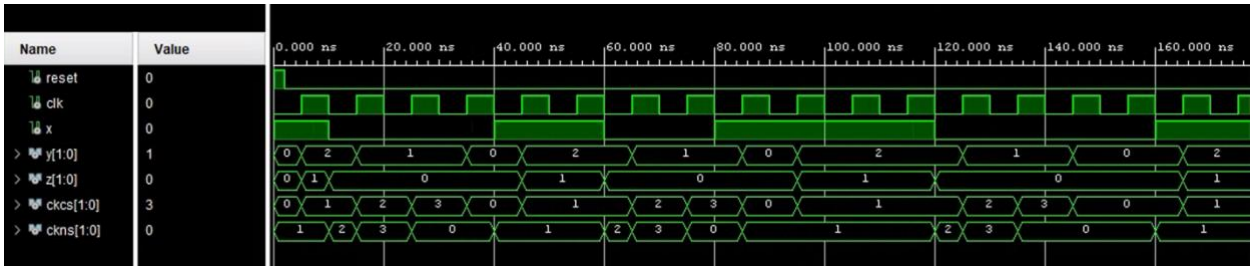

Figure 13. ASM simulation

Result Discussion

In this part of the laboratory, I was able to design the ASM chart. The overall concept wasn't difficult as it already discussed the finite state machine in Verilog which is similar to VHDL with some few modifications in the code. However, understanding the simulation is a little bit hard for me to analyze and keep track of.

PART 4: Stopwatch Design

Design Purpose and Engineering Data

For this part of the laboratory, I will be designing a stopwatch in VHDL by using the hierarchical design approach shown below.

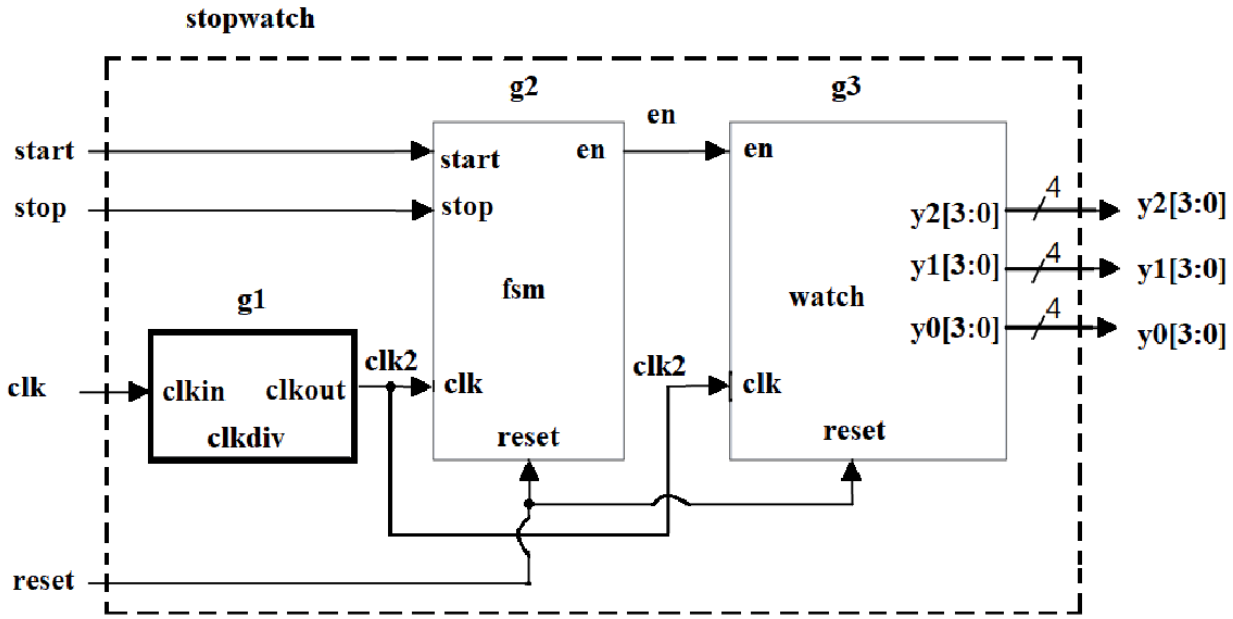

Figure 14. Stopwatch Block Diagram

From the diagram above, I was able to check the I/O ports used in this design which the inputs are start, stop, clk, and reset with the port size of 1 bit and the outputs are y2, y1, and y0 with 4-bit port size. Before starting this project, there are few features designing the stopwatch that I need to consider.

1. The frequency of the input “clk” signal is 4 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 output will main the value. After that, when the “start” is logic high again, the stopwatch will resume counting.
5. The output y2 y1 y0 is a 3-digit binary number, and each digit ranges from 0 to 9. This means that after 999 seconds, the stopwatch will return to 0 and then increase its value every second.

Step 1: clkdiv block

First, I was able to design the clkdiv in which the input clkin signal is 4 Hz and generates the 1 Hz output clkout. Below is the VHDL as well as testbench and simulation for clock division.

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- clock division
--clkdiv.vhd

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 arch1 of clkdiv is
signal  cnt:  std_logic_vector (3 downto 0) := "0000";
begin
process(clkin)
begin
if (rising_edge(clkin)) then
    if ( cnt = 3 )  then
        cnt <= (others=>'0');
        clkout <= '1';
    elsif (cnt < 1 ) then
        cnt <= cnt + 1;
        clkout <= '1';
    else
        cnt <= cnt + 1;
        clkout <= '0';
    end if;
end if;
end process;
end arch1;
```

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- clock division testbench
--clkdiv_tb.vhd

library IEEE;
use IEEE.std_logic_1164.all;
entity clkdiv_tb is
end clkdiv_tb;

architecture tb 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 tb;
```

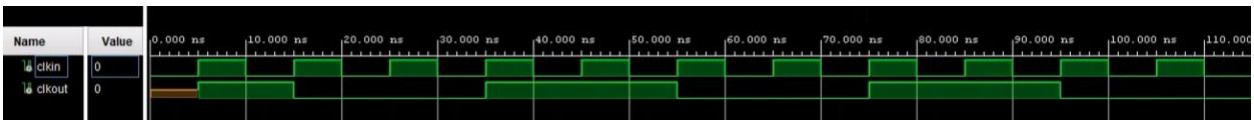

Figure 15. clkdiv VHDL (top left), testbench (top right), and simulation (below)

Step 2: fsm block

For this block, I need to understand idle state shown below.

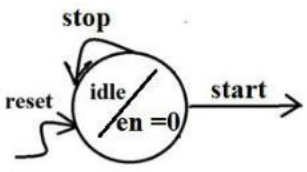

Figure 16. IDLE state of the fsm block

Setting the reset signal high will enter the first idle state which means the current watch is not counting and displays 0, which we called it “en”. If the “stop” button is pressed, the next state will still be idle state and if the “start” button is pressed, the finite state machine needs to enter another state. Below is the full VHDL, testbench, and simulation.

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- Finite State Machine
-- fsm.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm is
Port (reset, start, stop, clk: in std_logic;
      en: out std_logic;
      ckcs, ckns: out std_logic_vector (1 downto 0));
end fsm;

architecture Behavioral of fsm is
constant S0: std_logic_vector(1 downto 0) := "00";
constant S1: std_logic_vector(1 downto 0) := "01";
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, start, stop)
begin
    case (cs) is
        when S0 => if (start = '1') then ns <= S1;
                   elsif(stop = '1') then 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) else '0';
end Behavioral;
```

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- Finite State Machine
-- fsm.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm_tb is
end fsm_tb;

architecture testbench of fsm_tb is
component fsm
port(reset, start, stop, clk: in std_logic;
     en: out std_logic;
     ckcs, ckns: out std_logic_vector(1 downto 0));
end component;

signal reset, clk, start, stop, en: std_logic;
signal ckcs, ckns: std_logic_vector(1 downto 0);
begin
DUT: fsm port map(reset, start, stop, clk, en, ckcs, ckns);
process
begin
    clk <= '0'; wait for 5 ns;
    clk <= '1'; wait for 5 ns;
end process;

start <= '0', '1' after 50 ns, '0' after 100 ns, '1' after 160 ns;
stop <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 120 ns,
        '0' after 150 ns;

process
begin
    reset <= '1'; wait for 2 ns;
    reset <= '0'; wait for 400 ns;
    wait;
end process;
end testbench;
```

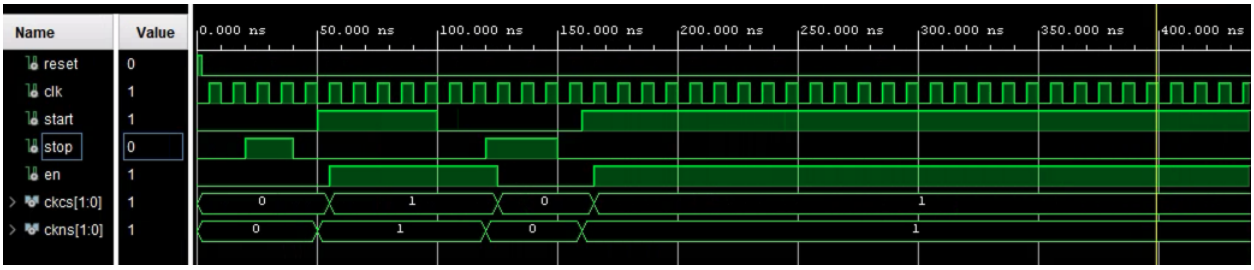

Figure 17. fsm VHDL (top left), testbench (top right), and simulation (below)

Step 3: watch block

This block will be like the main timer which we have three inputs: reset, en, and 1Hz clk; and three 4-bit outputs: y2, y1, and y0. Important features for this is that when “reset” is logic high, the watch will output 0 and only when “en” is logic high will increase its value every 1 second. Each of the output signals ranges from “0000” to “1001” meaning the timer goes from 0 to 9. Below is the full VHDL, testbench, and simulation.

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- watch
-- watch.vhd

Library ieee;
Use  ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity watch is
port(reset, en, clk: in std_logic;
     Y0, Y1, Y2: out std_logic_vector(3 downto 0));
end watch;

architecture arch2 of watch is
signal  cnt0:  std_logic_vector (3 downto 0) := "0000";
signal  cnt1:  std_logic_vector (3 downto 0) := "0000";
signal  cnt2:  std_logic_vector (3 downto 0) := "0000";
begin

process(clk)
begin
    if (reset = '1') then
        cnt0 <= "0000";
        cnt1 <= "0000";
        cnt2 <= "0000";
    end if;
    if (rising_edge(clk) and en='1') then
        if ( cnt0 = "1001")  then
            cnt1 <= cnt1 + 1;
            cnt0 <= "0000";
            if (cnt1 = "1001") then
                cnt2 <= cnt2 + 1;
                cnt1 <= "0000";
                if (cnt2 = "1001" and cnt1 = "1001" and cnt0 = "1001") then
                    cnt0 <= "0000";
                    cnt1 <= "0000";
                    cnt2 <= "0000";
                end if;
            end if;
        else
            cnt0 <= cnt0 + 1;
        end if;
end if;
end process;

y0 <= cnt0;
y1 <= cnt1;
y2 <= cnt2;
end arch2;
```

```vhdl
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- watch testbench
-- watch_tb.vhd

library IEEE;
use IEEE.std_logic_1164.all;
entity watch_tb is
end watch_tb;

architecture tb of watch_tb is
component watch
port(reset, en, clk: in std_logic;
     Y0, Y1, Y2: out std_logic_vector(3 downto 0));
end component;

signal reset, en, clk: std_logic;
signal Y0, Y1, Y2: std_logic_vector(3 downto 0);

begin
DUT: watch port map(reset, en, clk, Y0, Y1, Y2);
process
begin
clk <= '1'; wait for 5ns;
clk <= '0'; wait for 5ns;
end process;

process
begin
    reset <= '1'; wait for 50 ns;
    en <= '1'; reset <= '0'; wait for 150 ns;
    en <= '0'; reset <= '0'; wait for 50 ns;
    en <= '1'; wait for 1000 ns;
end process;
end tb;
```

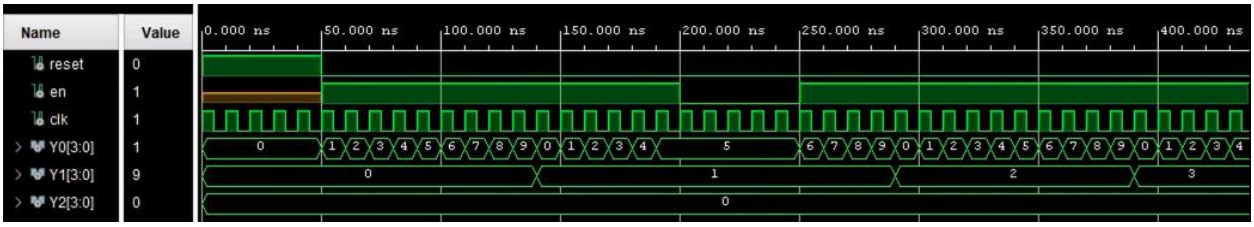

Figure 18. watch VHDL (top left), testbench (top right), and simulation (below)

Step 4: Final stopwatch block

For the final step, I will need to combine all the blocks designed earlier from steps 1 to 3. The full VHDL, testbench, and simulation shown below.

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- Stopwatch
-- stopwatch.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.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 clk2: std_logic;
signal enable: std_logic;

component clkdiv
port(clkin: in std_logic;
     clkout: out std_logic);
end component;

component fsm
port(reset, start, stop, clk: in std_logic;
     en: out std_logic);
end component;

component watch
port(reset, en, clk: in std_logic;
     Y0, Y1, Y2: out std_logic_vector(3 downto 0));
end component;

begin
    g1: clkdiv port map (clk, clk2);
    g2: fsm port map (reset, start, stop, clk, enable);
    g3: watch port map (reset, enable, clk, Y0, Y1, Y2);
end Behavioral;
```

```
-- Vigomar Kim Algador
-- CpE166 Section 03
-- Laboratory 03 Part 04
-- Stopwatch testbench
-- stopwatch_tb.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity stopwatch_tb is
end stopwatch_tb;

architecture testbench 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 <= '1'; wait for 5ns;
clk <= '0'; wait for 5ns;
end process;

process
begin
    reset <= '1'; wait for 10 ns;
    start <= '1'; stop <= '0'; reset <= '0';
    wait for 500 ns;
    start <= '0'; stop <= '1'; reset <= '0';
    wait for 100 ns;
    start <= '1'; stop <= '0'; reset <= '0';
    wait for 100000 ns;
end process;
end testbench;
```

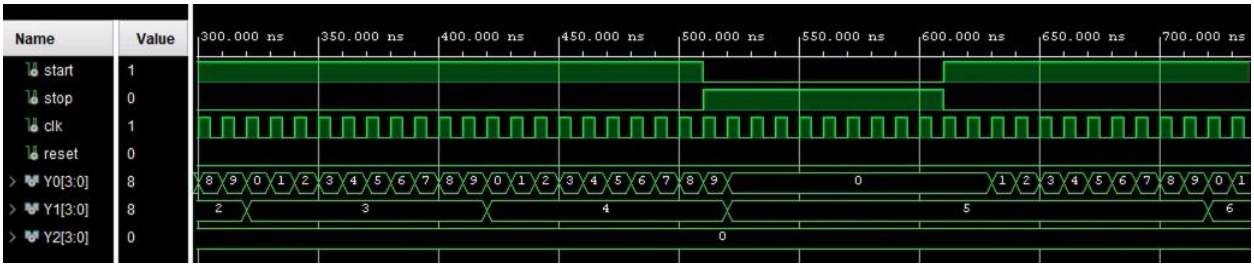

Figure 19. stopwatch VHDL (top left), testbench (top right), and simulation (below)

From the simulation above, we can see that the stop is logic high which the output still the same with y1 still 5. On the other hand, I was able to verify that the timer will starts again at 0 when the output reaches 999 seconds which is shown below.

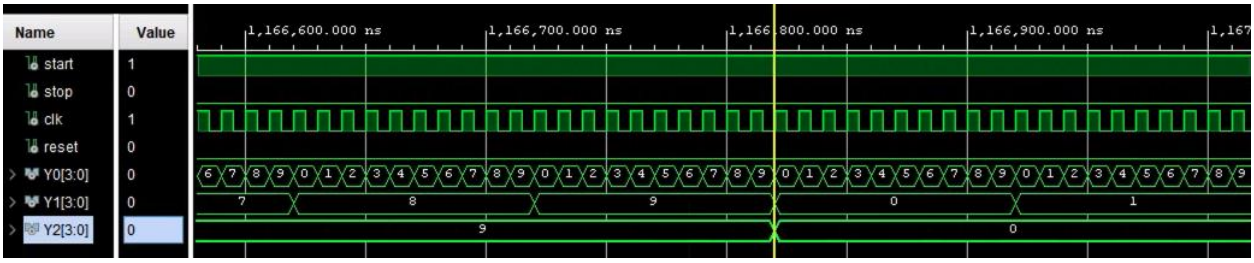

Figure 20. stopwatch when y0, y1, and y2 reaches 9.

Result Discussion

In this part of the laboratory, I was able to successfully design the stopwatch. Understanding the concept is fine but doing it from scratch is difficult. I have encountered many issues in this part especially the steps 3 and 4. The parts that were hard are the features needed like when the stop is pressed, the timer needs to stop and remains the same and then pressing start should continue. Another one is where the timer should start again at 0 when the output reaches 999 secs.

CONCLUSION

The whole laboratory discusses the basic concept of VHDL design and testbench, and different topics discussed in the lecture. From the previous laboratory, we use Verilog which is similar and few modifications for designing VHDL in different topics. This laboratory helps me to understand more about (7,4) Hamming Code generator, LFSR, ASM, and stopwatch. For the (7,4) Hamming Code Generator, I was able to understand the concept by hand but using the functions for MY_PACK and PARITY seems a little bit confusing for me instead of just designing one VHDL file with the use of XOR for each input. For the second part, I was able to design a Pseudorandom Number Generator and understand the concept behind it. I was also able to verify that it repeats every 31 clock cycles. The third part of this laboratory is easy for me as I already learned the concept of FSM and apply to Algorithmic State Machine with new information from the lecture class. For the final part, I was able to successfully design a stopwatch using clock division, FSM, and watch concept. I think this is the most difficult part as I encountered a lot of problems and demonstrate few times to check if the output is correct. Overall, I was able to learn and understand the concept of each topic and the use of VHDL.