CPE 166 Advance Logic Design

Lab Section 2

Shammah Thao

11/13/20

Contents

Introduction ...................................................................................................................................3

Part 1: Hamming Code Generator…………………......................................................................3

Design Purpose ..............................................................................................................................3

Verilog Data: .................................................................................................................................3

Verilog Code..................................................................................................................................4

Results Discussion ........................................................................................................................8

Part 2: Pseudorandom Number Generator………….....................................................................8

Design Purpose .............................................................................................................................8

Verilog Data: ................................................................................................................................8

Verilog Code.................................................................................................................................9

Results Discussion .......................................................................................................................11

Part 3: Algorithmic State Machine (ASM) Charts…....................................................................11

Design Purpose ............................................................................................................................11

Verilog Data: ...............................................................................................................................11

Verilog Code................................................................................................................................13

Results Discussion ......................................................................................................................16

Part 4: Stopwatch………………………………….....................................................................17

Design Purpose ...........................................................................................................................17

Verilog Data: ..............................................................................................................................17

Verilog Code...............................................................................................................................18

Results Discussion .....................................................................................................................22

Conclusion................................................................................................................................. 23

**Introduction:**

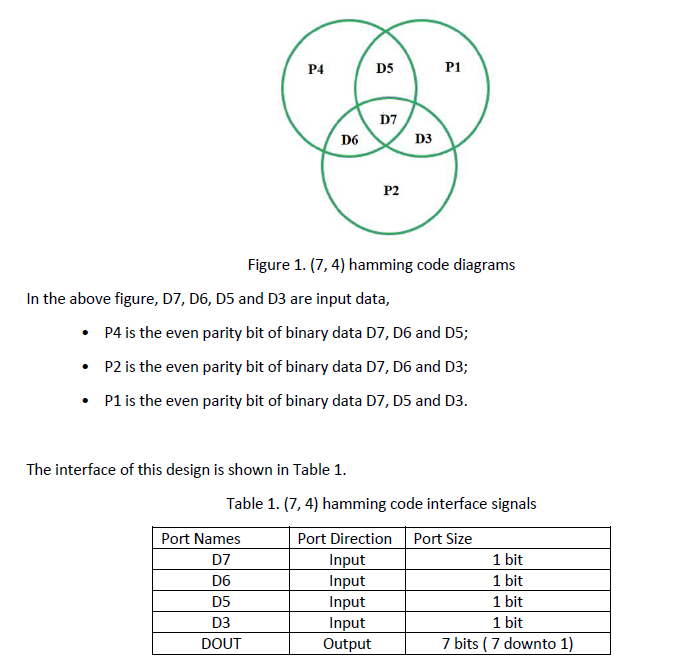
This lab was a 4-part lab that features coding in VHDL. VHDL, is another type of hardware description language originally founded in 1983, the hardware description language is used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general-purpose parallel programming language. Like Verilog, it accomplishes the same tasks needed to create logical circuits, with a slightly higher level of description than Verilog. VHDL that we did in this lab is going to be the foundation for other coding in VHDL.

**PART 1: (7, 4) Hamming Code Generator**

Design Purpose:

For this part of the lab, we made a Hamming Code generator. Hamming is a linear error-correcting code that encodes four bits of data into seven bits by adding three even parity bits. To begin we first used the given coding for the Parity function then expanded it to an Even Parity 3-bit input then are both combined into the hamming code.

Verilog Design:



Verilog Coding:

Step 1

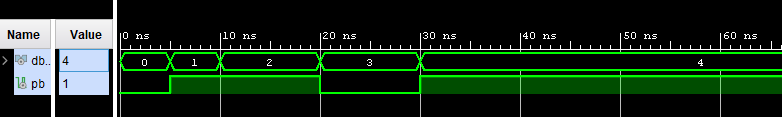
My\_Pack.vhdl

|  |
| --- |
| Source Code |
| 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; |

Step 2

PAR.vhdl

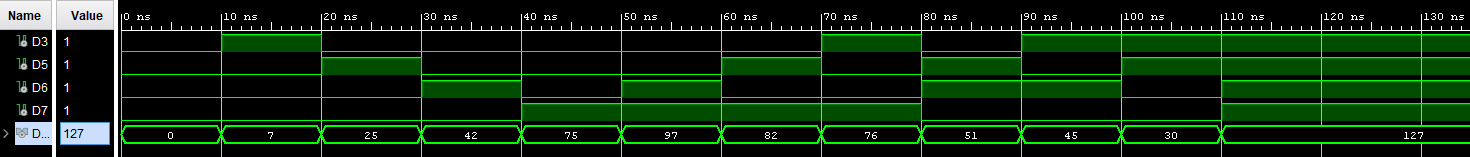
|  |  |
| --- | --- |
| Source Code | TestBench |
| library ieee;  use ieee.std\_logic\_1164.all;  use work.MY\_PACK.all;  entity PAR is  port( db: in std\_logic\_vector(2 downto 0);  pb: out std\_logic);  end PAR;  architecture ARCH of PAR is  begin  pb <= PARITY(db);  end ARCH; | library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  -- Uncomment the following library declaration if using  -- arithmetic functions with Signed or Unsigned values  --use IEEE.NUMERIC\_STD.ALL;  -- Uncomment the following library declaration if instantiating  -- any Xilinx leaf cells in this code.  --library UNISIM;  --use UNISIM.VComponents.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  dut : PAR  port map (db => db, pb =>pb);    db <= "000", "001" after 5 ns, "010" after 10 ns, "011" after 20 ns, "100" after 30 ns;    end Behavioral; |



Step 3

Hamming.vdhl

|  |  |
| --- | --- |
| Source Code | TestBench |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity hamming is  port (  D3 : in std\_logic;  D5 : in std\_logic;  D6 : in std\_logic;  D7 : in std\_logic;  DOUT : out std\_logic\_vector (7 downto 1)  );  end hamming;  architecture Behavior of hamming is  signal P1, P2, P4 : std\_logic;  begin  P1 <= D3 xor D5 xor D7;  P2 <= D3 xor D6 xor D7;  P4 <= D5 xor D6 xor D7;  DOUT <= D7 & D6 & D5 & P4 & D3 & P2 & P1;  end Behavior; | library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  -- Uncomment the following library declaration if using  -- arithmetic functions with Signed or Unsigned values  --use IEEE.NUMERIC\_STD.ALL;  -- Uncomment the following library declaration if instantiating  -- any Xilinx leaf cells in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity hamming\_tb is  -- Port ( );  end hamming\_tb;  architecture Behavioral of hamming\_tb is  component hamming  port (D3: in std\_logic;  D5: in std\_logic;  D6: in std\_logic;  D7: in std\_logic;  DOUT: out std\_logic\_vector);  end component;    signal D3: std\_logic;  signal D5 : std\_logic;  signal D6 : std\_logic;  signal D7 : std\_logic;  Signal DOUT : std\_logic\_vector (7 downto 1);    begin  dut: hamming port map(  D3 => D3,  D5 => D5,  D6 => D6,  D7 => D7,  DOUT => DOUT);  stimuli : process  begin  D3 <= '0';  D5 <= '0';  D6 <= '0';  D7 <= '0';  wait for 10ns;  D3 <= '1';  D5 <= '0';  D6 <= '0';  D7 <= '0';  wait for 10ns;  D3 <= '0';  D5 <= '1';  D6 <= '0';  D7 <= '0';  wait for 10ns;  D3 <= '0';  D5 <= '0';  D6 <= '1';  D7 <= '0';  wait for 10ns;  D3 <= '0';  D5 <= '0';  D6 <= '0';  D7 <= '1';  wait for 10ns;  D3 <= '0';  D5 <= '0';  D6 <= '1';  D7 <= '1';  wait for 10ns;  D3 <= '0';  D5 <= '1';  D6 <= '0';  D7 <= '1';  wait for 10ns;  D3 <= '1';  D5 <= '0';  D6 <= '0';  D7 <= '1';  wait for 10ns;  D3 <= '0';  D5 <= '1';  D6 <= '1';  D7 <= '0';  wait for 10ns;  D3 <= '1';  D5 <= '0';  D6 <= '1';  D7 <= '0';  wait for 10ns;  D3 <= '1';  D5 <= '1';  D6 <= '0';  D7 <= '0';  wait for 10ns;  D3 <= '1';  D5 <= '1';  D6 <= '1';  D7 <= '1';  wait for 10ns;  wait;  end process;    end Behavioral; |



Result Discussion:

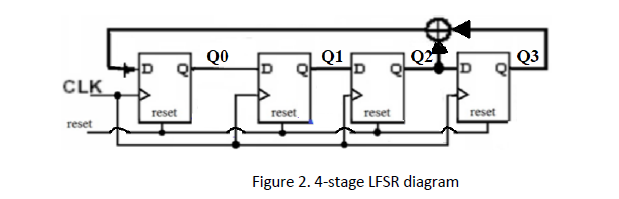
The code ran as expected. The only issues I had was at the beginning my testbench wasn’t meeting expectation, leading to some errors but after some correction I was about to fix it up for the demo. I also had an issue with my Par testbench which affect my final testbench since there was an error there, it wasn’t able to run the simulation, but I was able to catch on to that was able to correct it.

**Part 2: Pseudorandom Number Generator**

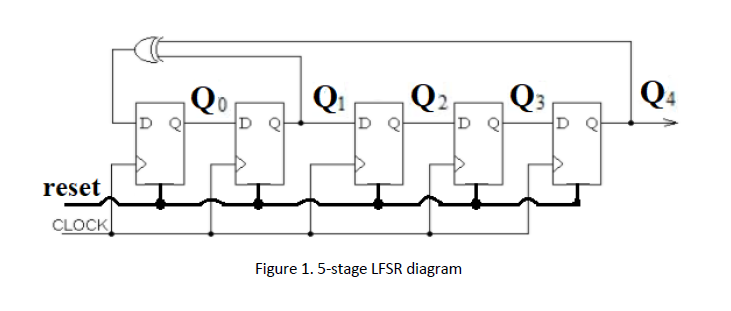
Design Purpose:

For this part of the lab, we made a Pseudorandom number generator using linear feedback shift register. The input bit is the output of a linear logic function of two or more of its previous states. The sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed in number generation and their reproducibility.

Verilog Design:



The figure above, features the linear feedback shift register used in the first part of the coding.

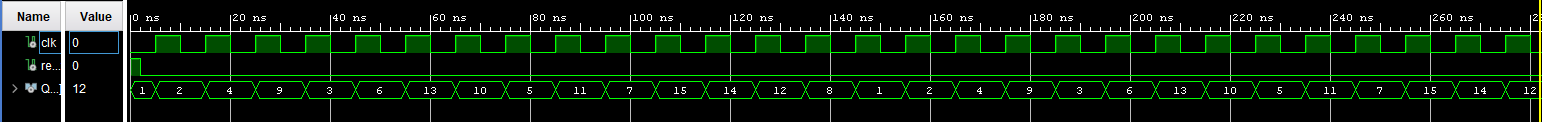


The figure above shows the final circuit for the Pseudorandom number generator.

Verilog:

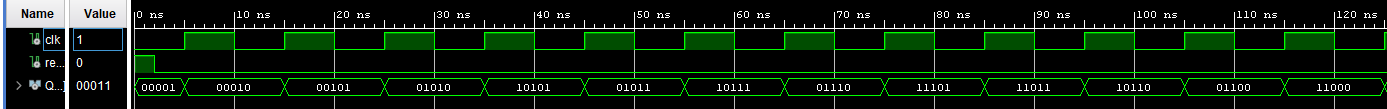
Step 1 and step 2

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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; | 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 2.

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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'); --- 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 beh; | 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; |



Result Discussion:

The coding for this part of the lab wasn’t as difficult as I thought it was going to be. I had to used VHDL examples from part one to be able to do this and the result came out okay. The coding was like the codes that was already given to us, and with some modification and adding some more codes in it was able to work

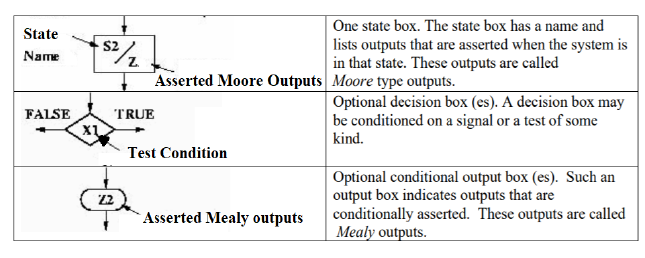
**Part 3: Algorithmic State machine (ASM) charts**

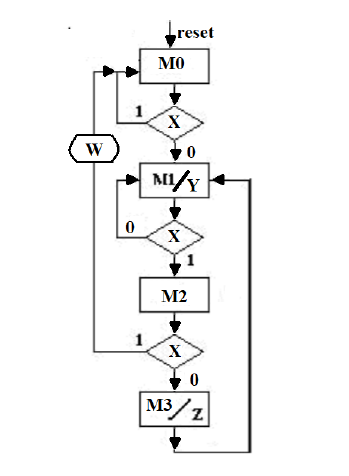
Design Purpose:

This part of the lab is learning how to code a Algorithmic State Machine (ASM) in VHDL. The algorithmic state machine is a method for design finite state machine. It is used to represent diagrams of digital integrated circuits. Its is like a state diagram but more structure and easier to understand. It is a method of describing the sequential operations of a digital system.

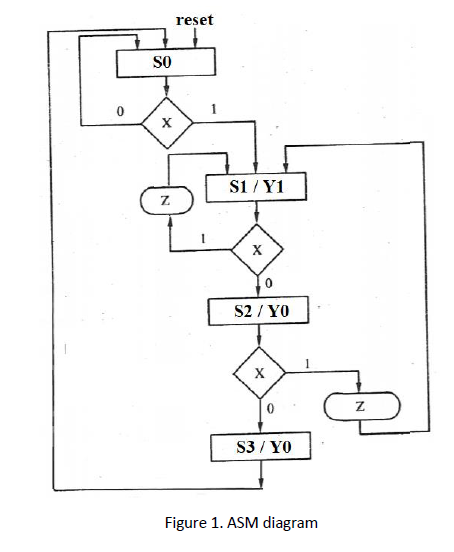
Verilog Design:

We had to first review that we knew about ASM from class then use its to understand the diagram that was given to us.





The figure above shows a ASM chart that has two moore output Y and Z. Y asserted in the M1 state and Z is asserted in the M3 state. It has a mealy output asserted when X is logic high in the M2 state.

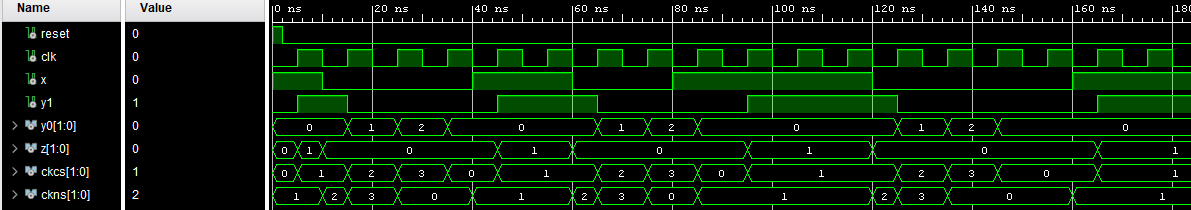


Verilog Chart:

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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; |  |

ASM

|  |  |
| --- | --- |
| Source Code | Testbench |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity asm is  port(  reset, clk, x: in std\_logic;  y1: out std\_logic;  y0, z: out std\_logic\_vector(1 downto 0);  ckcs, ckns: out std\_logic\_vector (1 downto 0)  );  end asm;  architecture Behavioral of asm 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  ckns <= ns;  ckcs <= cs;    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;  y1 <= '1' when (cs = s1) else '0';  z(0) <= '1' when (cs=s1) and (x ='1') else '0';  y0(0) <= '1' when (cs=s2) else '0';  z(1) <= '1' when (cs = s2) and (x='1') else '0';  y0(1) <= '1' when (cs = s3) else '0';    end Behavioral; | library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity asm\_tb is  end asm\_tb;  architecture Behavioral of asm\_tb is  component asm  port(reset, clk, x: in std\_logic;  y1: out std\_logic;  y0, z: out std\_logic\_vector(1 downto 0);  ckcs, ckns: out std\_logic\_vector (1 downto 0)  );  end component;  signal reset,clk,x,y1:std\_logic;  signal y0,z,ckcs,ckns:std\_logic\_vector(1 downto 0);  begin  DUT:asm port map(reset,clk,x,y1,y0,z,ckcs,ckns);  process  begin  clk <='0';  wait for 5ns;  clk <='1';  wait for 5ns;  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 Behavioral; |



Result Discussion:

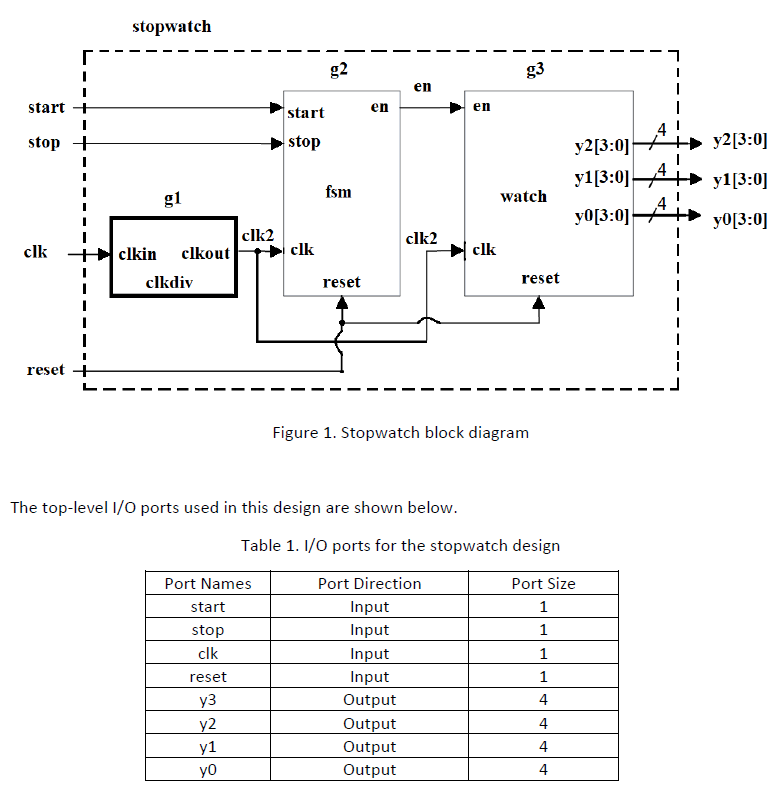
The coding for this one was not bad it was just some modification to the code that was already given to us and that was pretty much it. It was not that difficult overall, and the result came out okay.

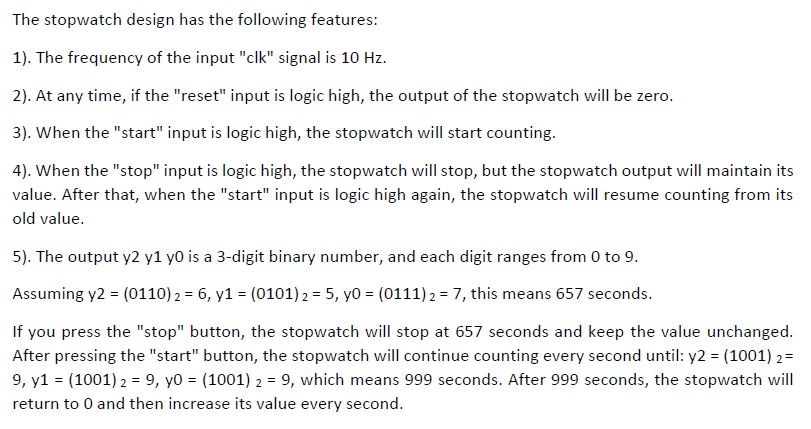
**Part 4: Stopwatch Design**

Design Purpose:

The project is to design a Stopwatch in vhdl by using hierarchical design approach. It goes from coding the clock function to the fsm then to the final watch code. The purpose would be to learn how to use hierarchical design in VHDL, which is different from Verilog coding.

Verilog Design:



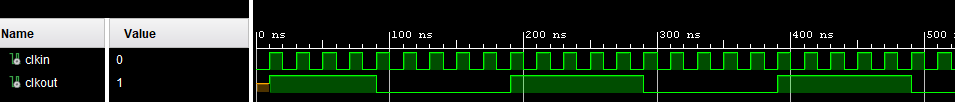


The diagram above was given to use to understand the hierarchical design of this coding project. It shows us what pot was need and which direction it was going. It also tells us how many of it are going to be used.

Verilog Coding:

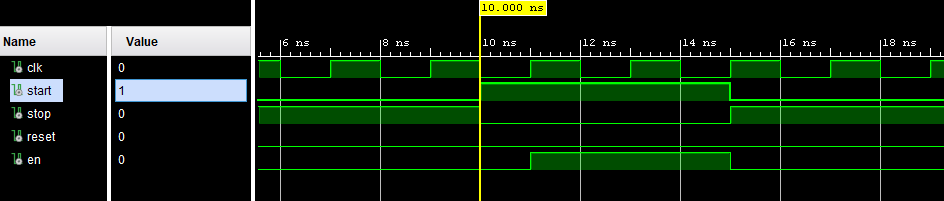
Clkdiv

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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; | 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 10 ns;  clkin <= '1';  wait for 10 ns;  end process;  end Behavioral; |



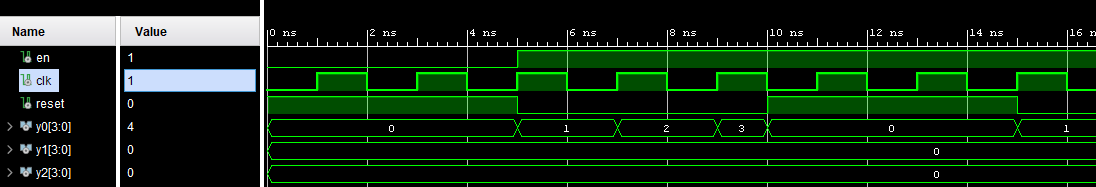
FSM

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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; | 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 5 ns;  reset <= '0';  stop <= '1'; -- S0  wait for 5 ns;  stop <= '0';  start <= '1'; -- S1  wait for 5 ns;  start <= '0';  stop <= '1'; -- S0  wait for 5 ns;  wait;  end process;  end Behavioral; |



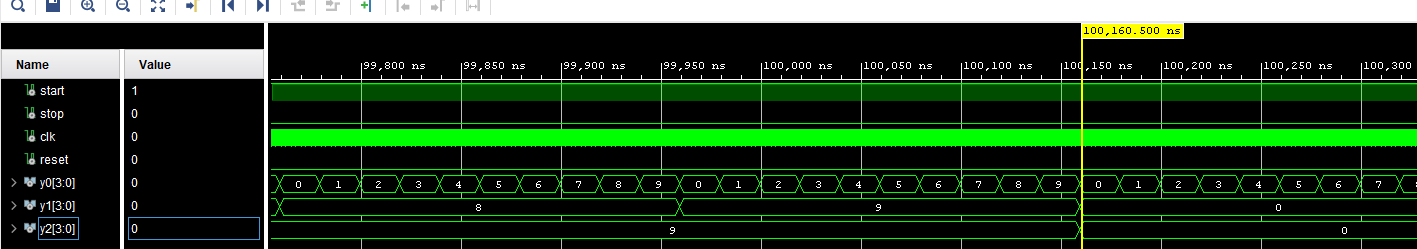
Watch

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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; | 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 5 ns;  reset <= '0';  en <= '1';  wait for 5 ns;  reset <= '1';  wait for 5 ns;  reset <= '0';  wait for 5 ns;  en <= '0';  wait for 5 ns;  en <= '1';  wait for 5 ns;  en <= '0';  wait;  end process;  end Behavioral; |



Stopwatch

|  |  |
| --- | --- |
| Source Code | Testbench |
| 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  h1: clkdiv port map(clkin => clk, clkout => clk2);  h2: fsm port map(clk => clk2, start => start, stop => stop, reset => reset, en => en);  h3: watch port map(en => en, clk => clk2, reset => reset, y0 => y0, y1 => y1, y2 => y2);  end Behavioral; | 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; |



Result Discussion:

For this part of the lab, it was a little confusing to do hierarchical design with the code since the structure of VHDL was different. I was kind of loss at where the code should be put at the beginning or the ending but was able to figure out with a little bit of help from other people. The testbench had to be modified to meet the criteria that the professor wanted but overall, it was okay.

**Conclusion:**

In conclusion, this was quite helpful toward learning how to code in VHDL. The coding at the beginning was quite confusing since VHDL seem quite different from Verilog coding. Even now, I am still not as confident coding in VHDL, but it was a great first-time learning process, as I was able to get some of the gist of it. Without the codes being given to us in the first few parts this lab would had been way more difficult then it already was. For me being unfamiliar with coding in VHDL and the structure of VHDL made it quite different for me to come to like coding in VHDL. Overall, this lab was helpful in learning VHDL and different ways of that VHDL was used.