11/14/20

CPE 166 Advanced Logic Design

LAB #3

XAVIER HOWELL

Table of contents

Introduction

In this lab we further implement the deign of VHDL. VHDL is a hardware programming language designed in the 80's. It is like Verilog but has a higher level of description. In this lab we use VHDL to program 4 main logic circuits. First is a Hamming code generator. Next is a pseudorandom number generator. Pseudorandom - "(of a number, a sequence of numbers, or any digital data) satisfying one or more statistical tests for randomness but produced by a definite mathematical procedure." Essentially is a random number generator created by a math equation. After that we created a algorithmic state machine. This is simply a state machine but created with a flowchart. The same way we would set up procedural code as a beginner with flowgorithm. The last part of this lab was to create a stopwatch. I was not able to complete this final step.

Report

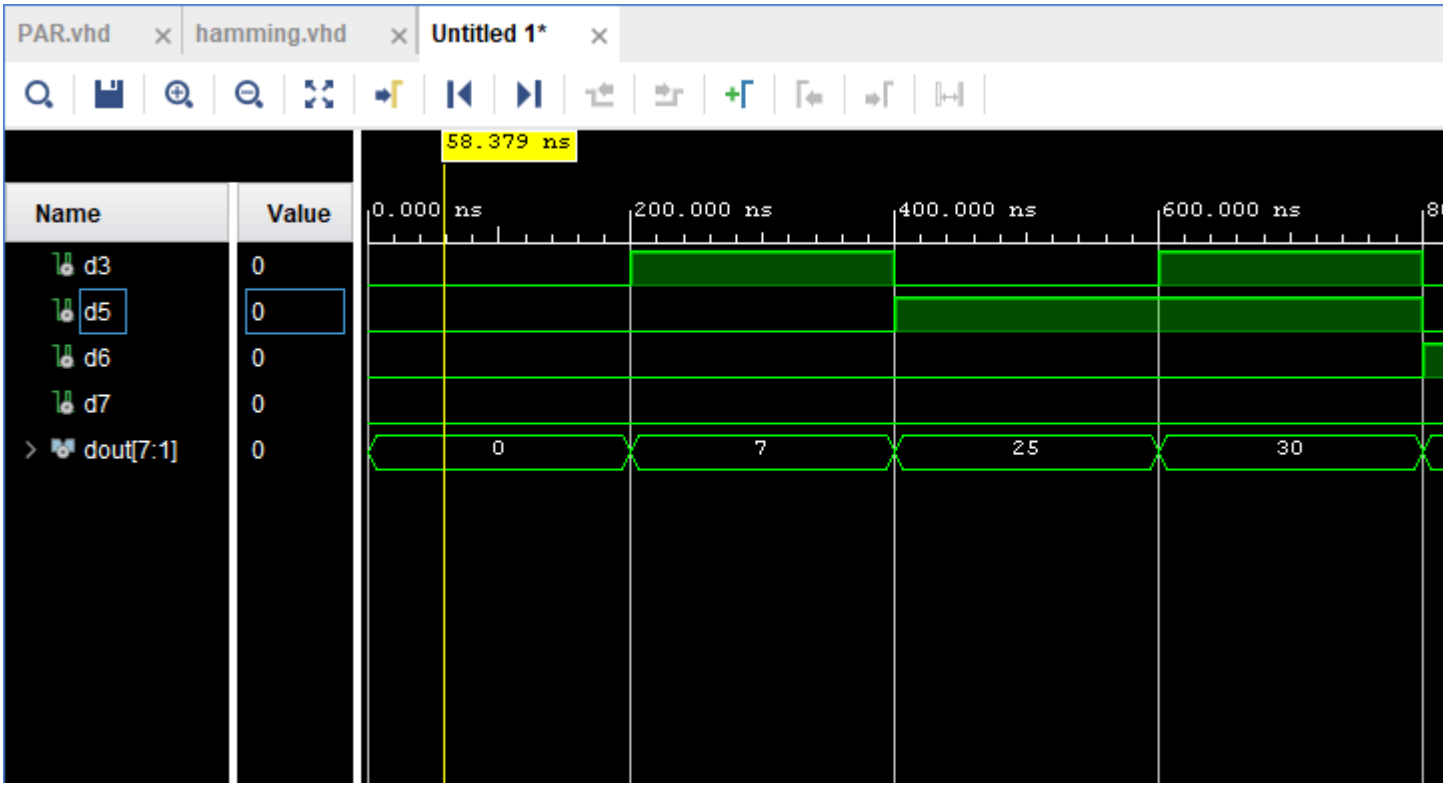
3.1 Design: Hamming Code is the use of extra parity bits to allow the correction of a single bit Error. Hamming can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. The purpose of part one is to do (7,4). That is a code generator that encodes four bits of data into seven.

3.1 VHDL

```
20
21
22  library ieee;
23
24  use ieee.std_logic_1164.all;
25
26  use work.MY_PACK.all;
27
28  entity PAR is
29
30  port( db: in std_logic_vector(2 downto 0);
31      pb: out std_logic);
32  end PAR;
33
34  architecture ARCH of PAR is
35
36  begin
37      pb <= PARITY(db);
38  end ARCH;
```

```
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.MY_PACK.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 is
    Port (D7,D6,D5,D3 : in std_logic;
          DOUT : out std_logic_vector (7 downto 1));
end hamming;

architecture Behavioral of hamming is
begin
    dout <= ((D7) & (D6) & (D5) & (PARITY(D7&D6&D5)) & (D3) & (PARITY(D7&D6&D3)) & (PARITY(D5&D7&D3)));


end Behavioral;
```

3.1 Testbench



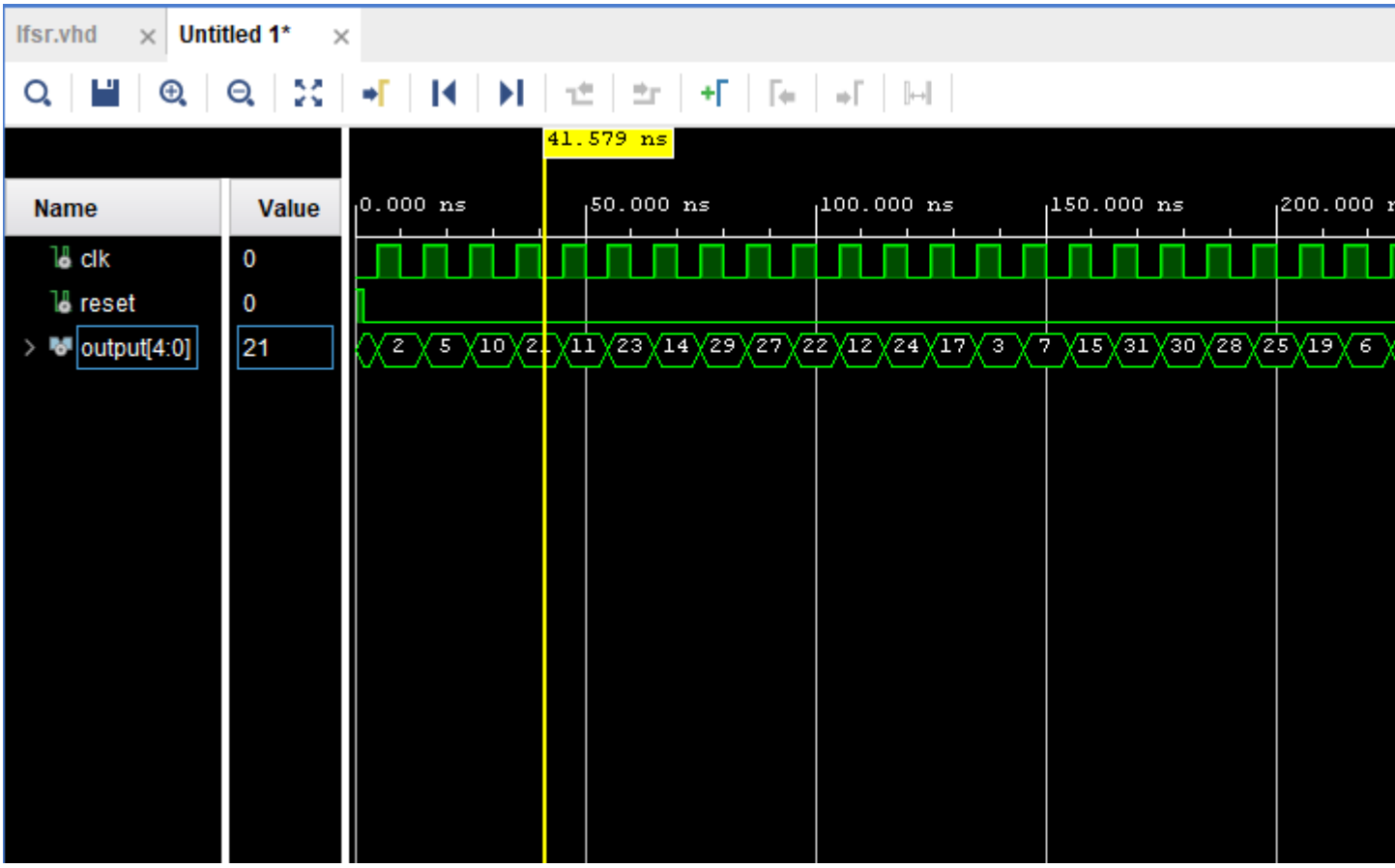
3.2 Design:

Part two is two build a linear feedback shift resister. A linear feedback shift register (also known as a 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 be using this to generate a random number generator.

3.2 VHDL

```
19  ----------------------------------------------------------------------------------
20
21
22  library IEEE;
23  use IEEE.std_logic_1164.ALL;
24
25  entity lfsr IS
26      port (reset, clk: in std_logic;
27            output: out std_logic_vector (4 downto 0));
28  end lfsr;
29
30  architecture beh of lfsr is
31  signal m : std_logic_vector (4 downto 0);
32  begin
33
34  process (reset,clk)
35      begin
36          if(reset = '1') then
37              m <= (0 => '1', others => '0');
38          elsif(rising_edge(clk)) then
39              m(4 downto 1) <= m(3 downto 0);
40              m(0) <= m(1) xor m(4);
41          end if;
42       end process;
43    output<=m;
44    end beh;
45
```

3.2 Testbench



3.3 Design:

Part three was a simple implementation of a ASM chart. These charts are a simpler way to look at and design circuits. The follow picture is of the chart we used. After design the parts and running the testbench we can trace back through the chart and make sure it flows logically.

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

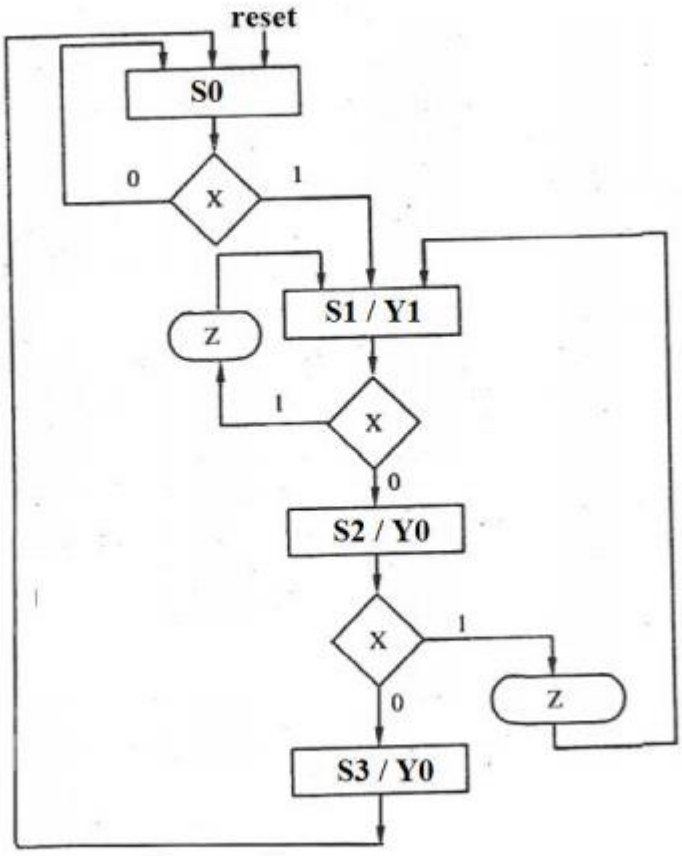


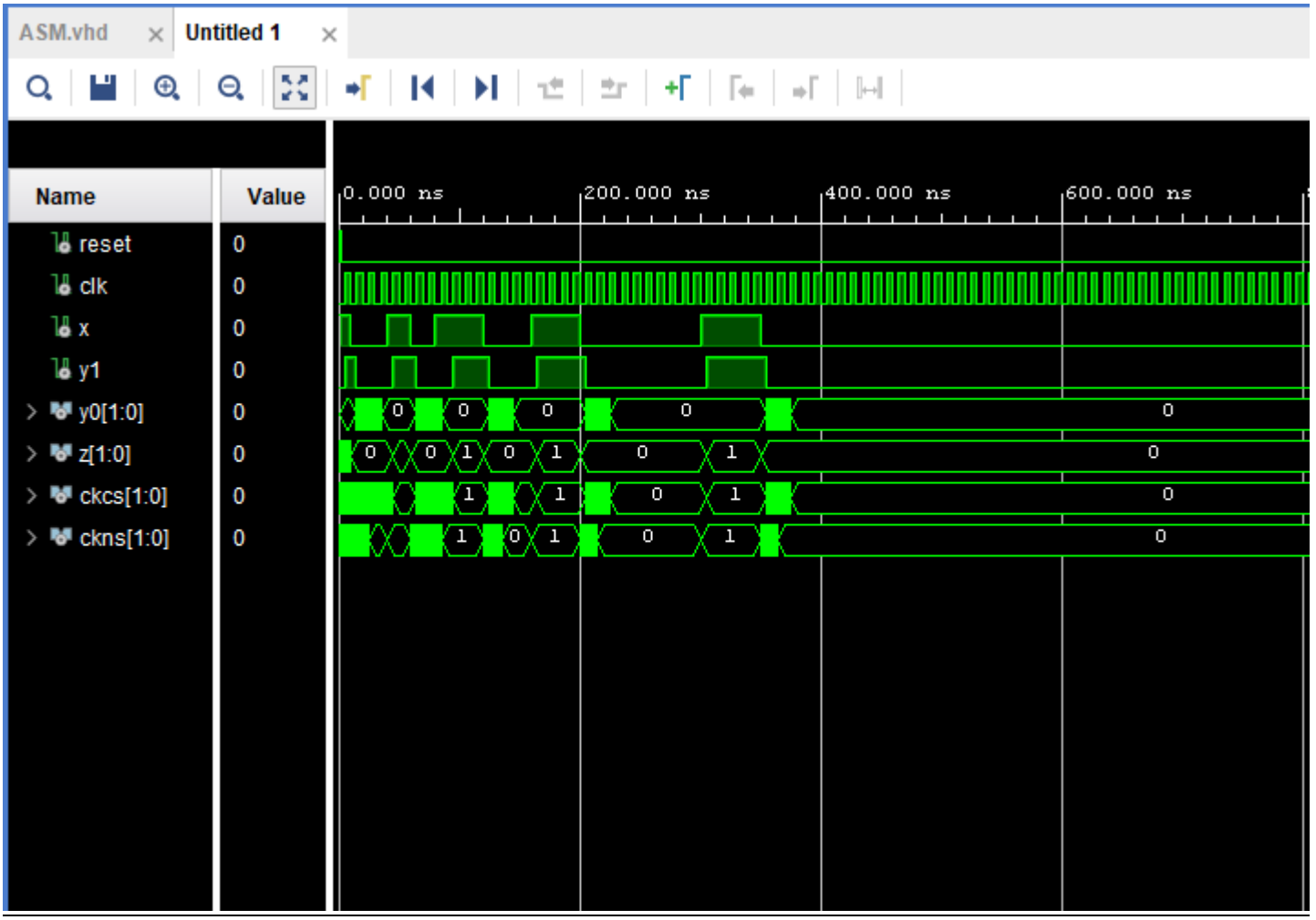
Figure 1. ASM diagram

3.3 VHDL

```
20
21
22  library ieee;
23  use ieee.std_logic_1164.all;
24
25  entity ASM is
26  port(reset, clk, x:   in std_logic;
27      y1: out std_logic;
28      y0, z: out std_logic_vector (1 downto 0);
29      ckcs, ckns: out std_logic_vector (1 downto 0));
30  end ASM;
31
32  architecture behavioral of ASM is
33
34  constant S0: std_logic_vector(1 downto 0) := "00";
35  constant S1: std_logic_vector(1 downto 0) := "01";
36  constant S2: std_logic_vector(1 downto 0) := "10";
37  constant S3: std_logic_vector(1 downto 0) := "11";
38  signal cs, ns: std_logic_vector(1 downto 0);
39
40  begin
41      ckns <= ns;
42      ckcs <= cs;
43  process(reset, clk) begin
44  if(reset = '1') then
45  cs <= S0;
46  elsif(rising_edge(clk)) then
47  cs <= ns;
48  end if;
49
50  end process;
51
```

```
51
52  process(cs, x)
53  begin
54      case(cs) is
55      when S0 =>
56      if(x = '1') then
57      ns <= S1;
58      else
59      ns <= S0;
60      end if;
61
62      when S1 => if
63      (x ='1') then
64      ns <= S1;
65      else
66      ns <= S2;
67      end if;
68
69      when S2 =>
70      if(x ='1') then
71      ns <= S1;
72      else
73      ns <= S3;
74      end if;
75
76      when S3 =>
77      ns <= S0;
78      when others =>
79      ns <= S0;
80      end case;
81
82  end process;
83
84      y1 <= '1' when (cs = S1) else '0';
85      y0(0) <= '1' when (cs = S2) else '0';
86      y0(1) <= '1' when (cs = S3) else '0';
87      z(0) <= '1' when (cs = S1) and (x = '1') else '0';
88      z(1) <= '1' when (cs = S2) and (x = '1') else '0';
89
90  end Behavioral;
91
```

3.3 Testbench



Conclusion

In conclusion this lab exposed me to VHDL design. I may still prefer Verilog for its simpler/lower level design. The biggest take away for me was the introduction to ASM charts. Having a visualization for your development helps better understand and trace through testbenches to know your programing is running correctly.