EEL 4712C - Digital Design: Lab Report 2

Cole Rottenberg
11062528

February 18th, 2024

Prelab Report

Prelab Questions

Demo:

1. Using the provided top_level.vhd, the 7-segment display entity you created in Lab 0, and the provided fsm.vhdp synthesize your circuit.
2. In Quartus, assign pins to the inputs (switches & buttons) and outputs (LEDs) such that the correct outputs are displayed on the 7-segment display.
3. Show your TA the RTL viewer showing the properly synthesized datapath.
4. Once the everything is working, show a TA the correct output for Fib(11) = 89.

Prelab Design and Implementation

The prelab for this lab required the implementation of a fibonacci calculator datapath. The datapath was to be implemented using a combinations of multiplexers, adders, and registers. Then, the design of the datapath was implemented in VHDL. The design was then tested by flashing the FPGA with the design and testing the output of the design. The following is the implementation of the datapath is in the Appendix under Listing 1.

Reflection

During the prelab, I learned the importantance of READING the entire lab manual before starting the prelab. I ended up implementing my own fibonacci logic in a single VHDL file. I wasted a good amount of time doing this, and I could have saved time by reading the lab manual. I was proud of my ability to implement the fibonacci logic in VHDL, but I was disappointed that I wasted time doing so.

Postlab Report

Problem Statement

The objective of this lab is to design a datapath capable of calculating the Nth Fibonacci number along with a testbench to verify its correctness. The algorithm for computing the Nth Fibonacci number is shown below; make sure you read and understand this code before continuing on to writing the datapath or testbench. The inputs to the datapath are a reset, a clock, and the input N, and a go signal. The output of the datapath is the Nth Fibonacci number. The function of the system is to calculate the Nth Fibonacci number.

Design

The design followed a given schematic that involved the use of multiplexers, adders, comparators, and registers. These components were then interconnected to form the fibonacci datapath. The signals that connected these units were primarily std_logic_vectors that were 24 bits wide. The algorithm used was based on starts at 2 and understanding that the first two numbers in the fibonacci sequence are 0 and 1. We also implemented essentially a for loop within VHDL

Implementation

The implementation of the design can be found in the Appendix under Listing 1. For individual components, the implementation was relatively straightforward. The mux can be seen in Listing 2, the adder in Listing 3, the register in Listing 4, and the comparator in Listing 5. The implementation of the datapath was more complex, but it was still relatively straightforward. The implementation of the datapath was completed in a few hours.

Testing

Testing the design was relatively straightforward. After working past syntax errors, the design was able to compile and after assigning the correct outputs to LEDs, the code worked immeadiately. The design was also tested in ModelSim with 11 as the input, and the output was 89, which is the correct output for the 11th number in the fibonacci sequence. In figure 1, the ModelSim simulation of the datapath can be seen. The simulation shows the correct output for the 11th number in the fibonacci sequence. The design was also tested on the FPGA board, and the output was 89, which is the correct output for the 11th number in the fibonacci sequence. The FPGA board with the working datapath can be seen in figure 2.

NOTE: Testing in ModelSim was done before the datapath redesign and was tested with my own fibonacci logic which is similar to the given logic. The given logic was tested on the FPGA board and worked as expected.

Conclusions

The work in this lab was incredibly foundational considering the practice of structural architecture and use of multiple components to create a single, complex system. The lab was successful in that the design was able to be implemented and tested on the FPGA board. The only problem encountered was the time wasted in the prelab. In the future, I will be sure to read the entire lab manual before starting the prelab.

Appendix

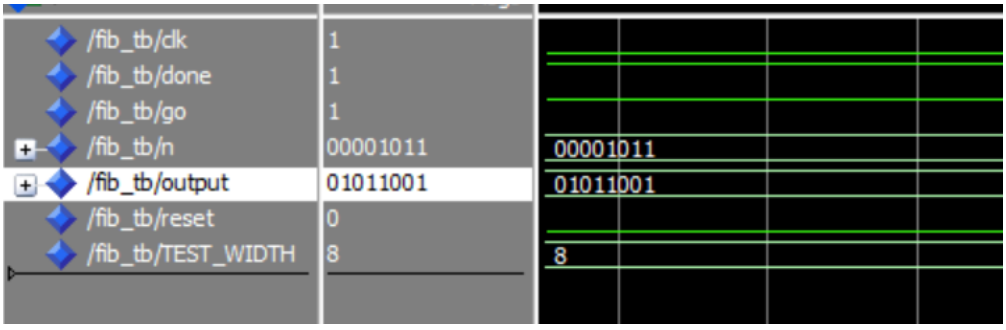

Figure 1: ModelSim Simulation of Datapath

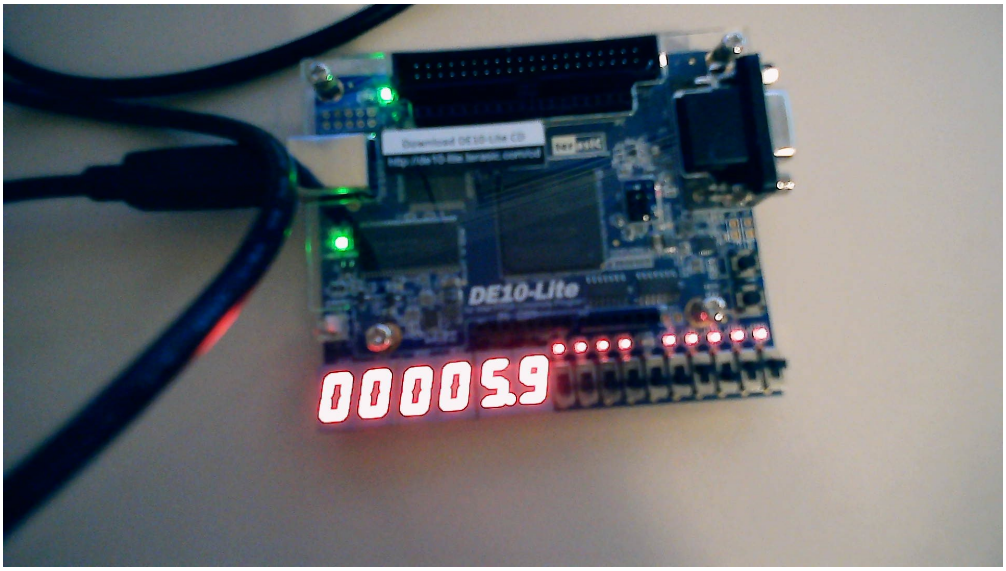

Figure 2: FPGA Board with Working Datapath

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  -- Datapath for fibonacci sequence
6
7  entity datapath is
8      generic(
9        width : integer := 24
10     );
11     port(
12       -- clk and rst
13       clk: in std_logic;
14       rst : in std_logic;
15
16       -- i/o
17       n : in std_logic_vector(5 downto 0);
18       result : out std_logic_vector(23 downto 0);
19       -- control inputs
20       x_sel : in std_logic;
21       y_sel : in std_logic;
22       i_sel : in std_logic;
23       result_sel : in std_logic;
24
25
26       n_en : in std_logic;
27       result_en : in std_logic;
28       x_en : in std_logic;
29       y_en : in std_logic;
30       i_en : in std_logic;
31       n_eq_0 : out std_logic;
32
33       -- status signals
```

```
34          i_le_n : out std_logic
35      );
36  end datapath;
37
38  -- The datapath is composed of registers, fibonacci logic and other components
39
40  -- The datapath starts with a register for the input n
41  -- if n_en is high, the input n is loaded into the register
42  -- afterwards, the register is connected to the fibonacci logic
43
44  architecture rtl of datapath is
45      signal i_reg_out, x_reg_out, y_reg_out, n_reg_out :
            std_logic_vector(width-1 downto 0);
46      signal i_mux_out, x_mux_out, y_mux_out, result_mux_out :
            std_logic_vector(width-1 downto 0);
47      signal add1_out, add2_out : std_logic_vector(width-1 downto 0);
48      signal n_scaled : std_logic_vector(width-1 downto 0);
49    -- Build input register and logic
50      begin
51
52    -- Connecting components to impl fibonacci logic using registers,mux,
          adders and comparators
53          I_MUX : entity work.mux
54              generic map(
55                  width => width
56              )
57              port map(
58                  input1 => add1_out,
59                  input2 => std_logic_vector(to_unsigned(2, width)),
60                  sel => i_sel,
61                  output => i_mux_out
62              );
63
64          X_MUX : entity work.mux
65              generic map(
66                  width => width
67              )
68              port map(
69                  input1 => y_reg_out,
70                  input2 => std_logic_vector(to_unsigned(0, width)),
71                  sel => x_sel,
72                  output => x_mux_out
73              );
74
75          Y_MUX : entity work.mux
76              generic map(
77                  width => width
78              )
79              port map(
80                  input1 => add2_out,
81                  input2 => std_logic_vector(to_unsigned(1, width)),
82                  sel => y_sel,
83                  output => y_mux_out
84              );
```

```
85
86          RESULT_MUX : entity work.mux
87              generic map(
88                  width => width
89              )
90              port map(
91                  input1 => y_reg_out,
92                  input2 => std_logic_vector(to_unsigned(0, width)),
93                  sel => result_sel,
94                  output => result
95              );
96
97          -- REGISTERS
98
99          I_REG : entity work.reg
100             generic map(
101                 width => width
102             )
103             port map(
104                 clk => clk,
105                 reset => rst,
106                 d => i_mux_out,
107                 q => i_reg_out,
108                 en => i_en
109             );
110
111         X_REG : entity work.reg
112             generic map(
113                 width => width
114             )
115             port map(
116                 clk => clk,
117                 reset => rst,
118                 d => x_mux_out,
119                 q => x_reg_out,
120                 en => x_en
121             );
122
123         Y_REG : entity work.reg
124             generic map(
125                 width => width
126             )
127             port map(
128                 clk => clk,
129                 reset => rst,
130                 d => y_mux_out,
131                 q => y_reg_out,
132                 en => y_en
133             );
134
135         N_REG : entity work.reg
136             generic map(
137                 width => width
138             )
```

```
139             port map(
140                 clk => clk,
141                 reset => rst,
142                 d => (width-1-6 downto 0 => '0') & n,
143                 q => n_reg_out,
144                 en => n_en
145             );
146 
147         -- ADDERS
148 
149         ADD1 : entity work.add
150             generic map(
151                 width => width
152             )
153             port map(
154                 in1 => std_logic_vector(to_unsigned(1, width)),
155                 in2 => i_reg_out,
156                 output => add1_out
157             );
158 
159         ADD2 : entity work.add
160             generic map(
161                 width => width
162             )
163             port map(
164                 in1 => x_reg_out,
165                 in2 => y_reg_out,
166                 output => add2_out
167             );
168 
169         -- COMPARATOR
170         N_COMP : entity work.comparator
171             generic map(
172                 width => width
173             )
174             port map(
175                 a => n_reg_out,
176                 b => std_logic_vector(to_unsigned(0, width)),
177                 eq => n_eq_0
178             );
179 
180         LE_COMP : entity work.comparator
181             generic map(
182                 width => width
183             )
184             port map(
185                 a => i_reg_out,
186                 b => n_reg_out,
187                 le => i_le_n
188             );
189 end rtl;
```

Listing 1: Datapath VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity mux is
    generic(
        width : integer := 6
    );
    port(
        input1 : in std_logic_vector(width-1 downto 0);
        input2 : in std_logic_vector(width-1 downto 0);
        sel : in std_logic;
        output : out std_logic_vector(width-1 downto 0)
    );

end entity mux;

architecture rtl of mux is
begin
    process(input1, input2, sel)
    begin
        if sel = '0' then
            output <= input1;
        else
            output <= input2;
        end if;
    end process;
end architecture rtl;
```

Listing 2: Mux VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-- Adder entity
entity add is
    generic(width: integer := 6);
    port(
        in1 : in std_logic_vector(width-1 downto 0);
        in2 : in std_logic_vector(width-1 downto 0);
        output : out std_logic_vector(width-1 downto 0)
    );
end add;

-- Adder architecture
architecture arch of add is
begin
    output <= std_logic_vector(unsigned(in1) + unsigned(in2));
end arch;
```

Listing 3: Add VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
```

```
3  use ieee.numeric_std.all;
4  entity reg is
5      generic(
6  __ width : integer := 6
7  __ );
8      port(
9          clk: in std_logic;
10         reset: in std_logic;
11         d: in std_logic_vector(width-1 downto 0);
12         q: out std_logic_vector(width-1 downto 0);
13         en : in std_logic
14     );
15 end entity;
16
17 architecture rtl of reg is
18 begin
19     process(clk, reset)
20     begin
21         if reset = '1' then
22             q <= (others => '0');
23         elsif rising_edge(clk) then
24             if en = '1' then
25                 q <= d;
26             end if;
27         end if;
28     end process;
29 end rtl;
```

Listing 4: Reg VHDL Code

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  -- Comparator entity
6  entity comparator is
7      generic(width : integer := 6);
8      port (
9          a : in std_logic_vector(width-1 downto 0);
10         b : in std_logic_vector(width-1 downto 0);
11         le : out std_logic;
12         eq : out std_logic
13     );
14 end comparator;
15
16 -- Comparator architecture
17
18 architecture behavioral of comparator is
19     begin
20         process(a,b)
21         begin
22         -- Less than or equal to
23         if (unsigned(a) <= unsigned(b)) then
24             le <= '1';
25         else
```

```
26              le <= '0';
27          end if;
28          -- Equal to
29          if (unsigned(a) = unsigned(b)) then
30              eq <= '1';
31          else
32              eq <= '0';
33          end if;
34          end process;
35  end behavioral;
```

Listing 5: Comparator VHDL Code