EEL 4712
Midterm 2 – Spring 2016
VERSION 1

Name: Solution

UFID:

Sign here to give permission for your test to be returned in class, where others might see your score:

IMPORTANT:
- Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
- **As always, the best answer gets the most points.**

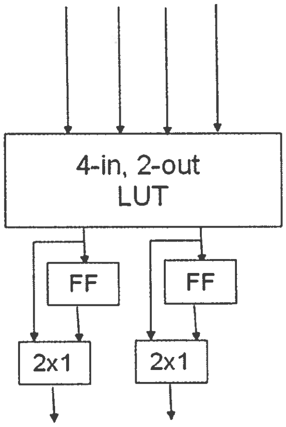
COVER SHEET:

Problem#:	Points
1 (15 points)	
2 (7 points)	
3 (7 points)	
4 (8 points)	
5 (20 points)	
6 (20 points)	
7 (20 points)	
Free	3

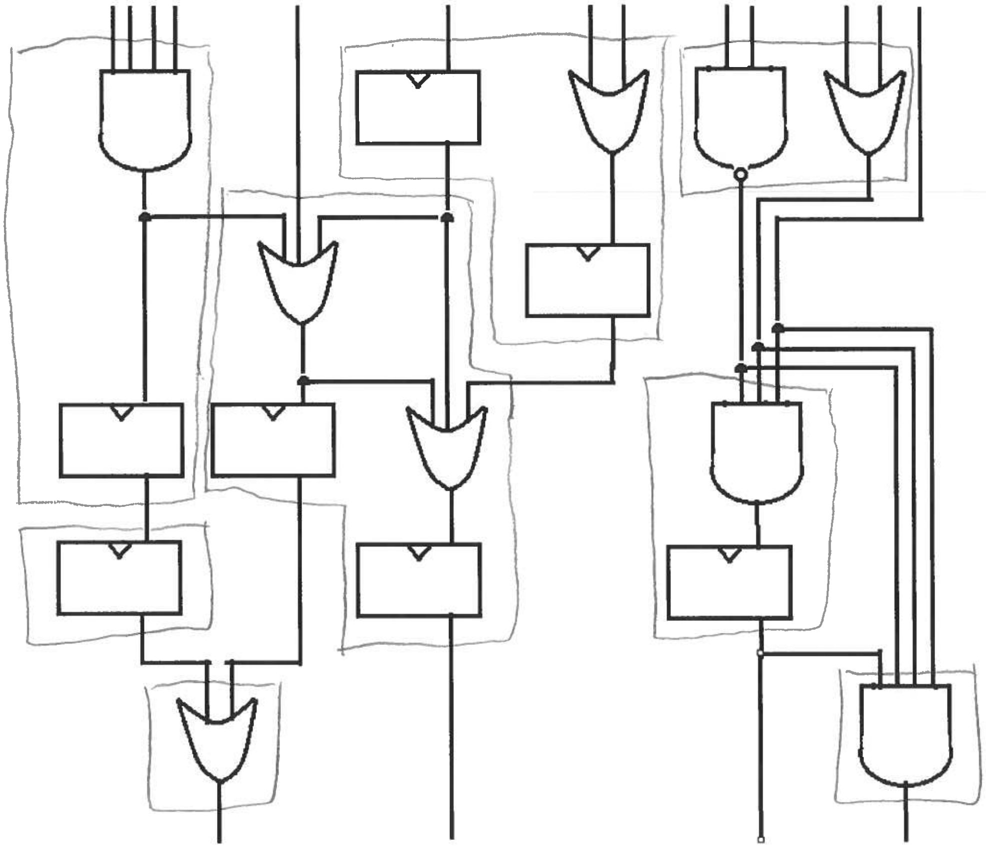
Total:

Regrade Info:

1. (15 points) Assume you are given an FPGA that consists of the following CLB structures with one 4-input, 2-output LUT and optional registers on each output.



Map the following circuit onto these CLBs by drawing boxes to represent CLBs. Assume that the rectangle components are flip flops and that everything else is combinational logic. Use the minimum number of CLBs for full credit.



2. (7 points) List the different interconnect resources that an FPGA uses to route between two components in an FPGA.

Connection box, tracks, switch boxes

3. (7 points) The Cyclone III EP3C16 has 56 M9k block RAMs, which provide 8192 bits of memory. Assuming an image consisting of 24-bit pixels, calculate the maximum ***square*** image size you could store in the FPGA.

Total bits = 56 * 8192 = 458752
Total pixels = 458752 / 24 = 19114

Maximum square image = floor(sqrt(19114)) = 138x138

4. a. (4 points) You are given a 75 MHz clock and must divide it to create a 23 kHz clock. How many 75 MHz cycles does the divider have to wait for a ***full*** period of the 23 kHz clock? i.e. The divider can produce a frequency that is slight slower than 23 kHz, but no faster.

   1/75M * x = 1/23k
   x = 75M/23k = 3260.8

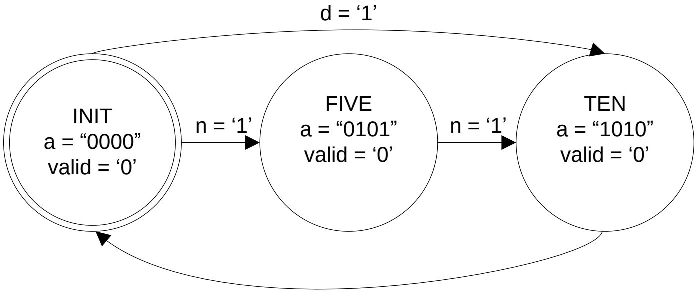
   # of cycles = ceil(3260.8) = 3261

   b. (4 points) Calculate the actual frequency of the divided clock. Show at least two fractional digits.

1/75M * 3261 = 1/f

F = 75M/3261 = 22999.08 Hz

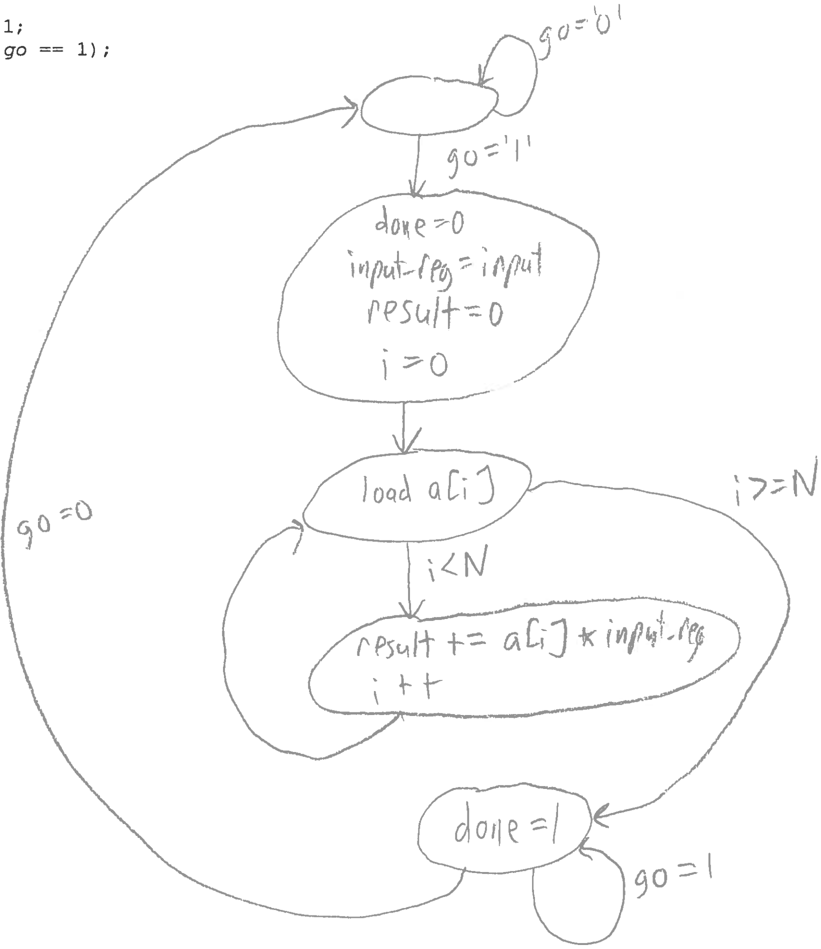
5. (18 points) Fill in the code to implement the following Moore finite state machine (FSM) *using the 2-process FSM model.* **Assume that INIT is the initial state. Assume that *d* takes priority over *n*. Assume there are implicit edges back to the current state for any condition not explicitly shown. Transitions without conditions are always taken.** Use the next page if extra room is needed.



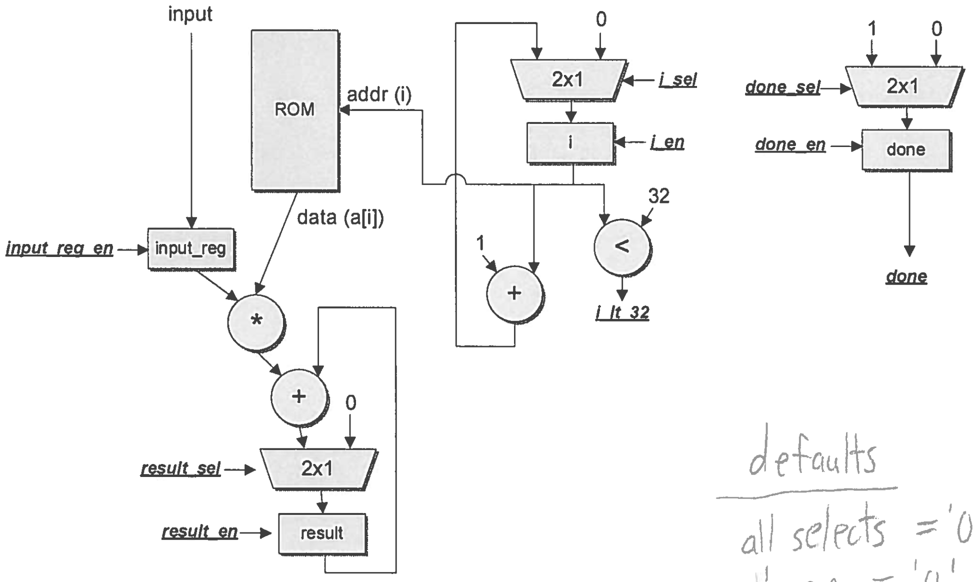
```
library ieee;
use ieee.std_logic_1164.all;

entity fsm is
    port (
        clk, rst : in  std_logic;
        n, d     : in  std_logic;
        a        : out std_logic_vector(3 downto 0);
        valid    : out std_logic
        );
end fsm;

architecture PROC2 of fsm is

    type STATE_TYPE is (S_INIT, S_FIVE, S_TEN);
    signal state, next_state : STATE_TYPE;

begin

    process(clk, rst)
    begin
        if (rst = '1') then
            state <= S_INIT;
        elsif (rising_edge(clk)) then
            state <= next_state;
        end if;
    end process;

    process(state, n, d)
    begin

        valid      <= '0';
        next_state <= state;

        case state is
            when S_INIT =>
                a <= "0000";

                if (d = '1') then
                    next_state <= S_TEN;
                elsif (n = '1') then
                    next_state <= S_FIVE;
                end if;

            when S_FIVE =>
                a <= "0101";
                if (n = '1') then
                    next_state <= S_TEN;
                end if;

            when S_TEN =>
                a     <= "1010";
                valid <= '1';

                next_state <= S_INIT;

        end case;
    end process;

end PROC2;
```

6. (20 points) Create an FSMD that implements the following pseudo-code. **Do not write VHDL and instead leave the FSMD in graphical form** (i.e., state machine with corresponding operations in each state). Make sure to specify all operations and state transitions. Note that *result*, *go*, *input,* and *done* are I/O. For the array a[i], assume that your circuit has a ROM that stores the entire array, and that all values are already stored in the ROM. Assume the ROM has a one-cycle read latency. Show the read operation in your FSMD as "load a[i]". Make sure to not use data from the ROM until one cycle after this load.

```
const int N = 32; // In VHDL: generic( N : positive )

Inputs: go (std_logic), input (std_logic_vector)
Outputs: result (std_logic_vector), done (std_logic)

int i, result, input_reg;
int a[N]; // Stored as a ROM in the circuit, assume already filled with appropriate data

// reset values for outputs
done = 0; result = 0;

while (1) {

        while (go == 0);
        done = 0;
        // Store input into register
        input_reg = input;

        result = 0;
        for (i=0; i < N; i++) {
                result = result + a[i] * input_reg;
        }

        done = 1;
        while (go == 1);
}
```



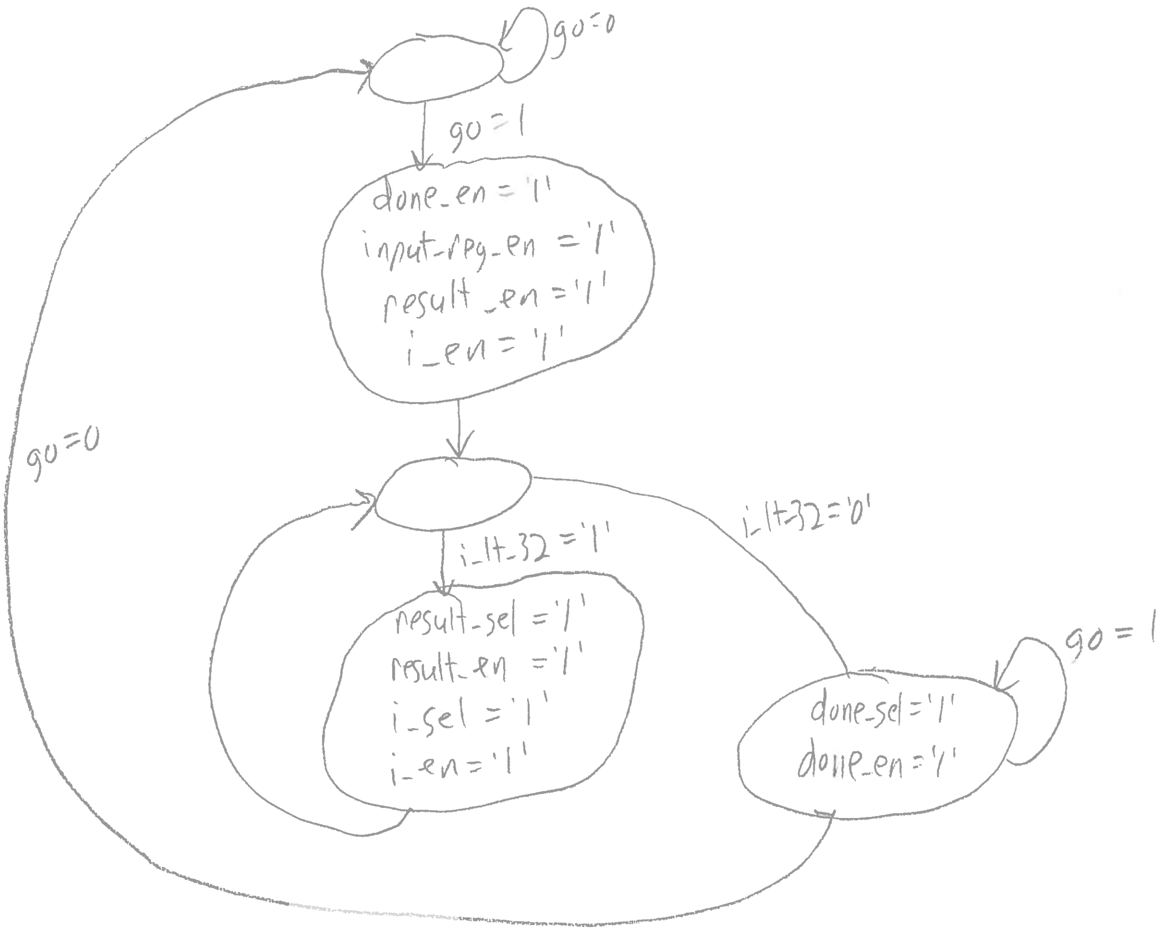
7. (20 points) Draw an FSM capable of controlling the illustrated datapath to perform the pseudo-code in question 6, by assigning or reading from the underlined control signals. Assume that *go* is an input to the controller and that left mux inputs have a select value of 1. Also assume that ROM contents store the a[] array and that these values have already been stored. Note that this datapath assumes that *N=32*. **Do not write any VHDL code, just show the FSM and control signals. Be sure to mention default signal values to save space.**



defaults

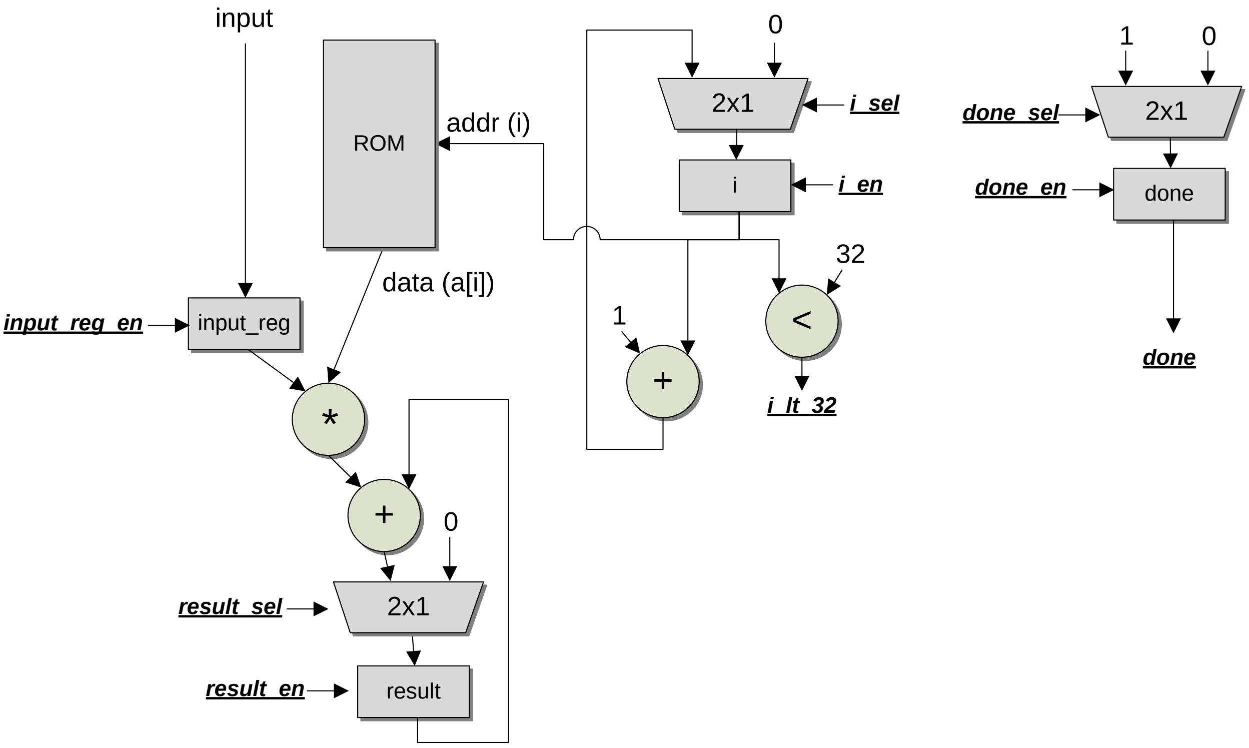
all selects = '0'

all en = '0'



Problem 6 Reference

```
const int N = 32; // In VHDL: generic( N : positive )

Inputs: go (std_logic), input (std_logic_vector)
Outputs: result (std_logic_vector), done (std_logic)

int i, result, input_reg;
int a[N]; // Stored as a ROM in the circuit, assume already filled with appropriate data

// reset values for outputs
done = 0; result = 0;

while (1) {

        while (go == 0);
        done = 0;
        // Store input into register
        input_reg = input;

        result = 0;
        for (i=0; i < N; i++) {
                result = result + a[i] * input_reg;
        }

        done = 1;
        while (go == 1);
}
```



input
ROM
addr (i)
data (a[i])
input_reg_en
input_reg
*
+
0
result_sel
2x1
result_en
result
0
2x1
i_sel
i
i_en
32
1
+
<
i_lt_32
1
0
done_sel
2x1
done_en
done
done