Summary Sheet:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (8 pts)</td>
<td></td>
</tr>
<tr>
<td>2 (22 pts)</td>
<td></td>
</tr>
<tr>
<td>3 (17 pts)</td>
<td></td>
</tr>
<tr>
<td>4 (22 pts)</td>
<td></td>
</tr>
<tr>
<td>5 (15 pts)</td>
<td></td>
</tr>
<tr>
<td>6. (16 pts)</td>
<td></td>
</tr>
</tbody>
</table>

Total

Re-Grade Information:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
1. Miscellaneous.
(a) VGA display calculation:

For Lab 5, assuming the board clock frequency is 100 MHz, what constant should be used for H_DISPLAY_END? For credit, please show work.
(For credit, show work here.)

36.83 (answer) (4 pts.)

$$H_{-}\text{DISPLAY\_END} = A - E = 37.77 - 0.94 \mu s$$
$$= 36.83 \mu s$$

$$\frac{36.83 \mu s}{100 \mu s} = 3683$$

(b) For credit, show work. For Lab 6, assume the latency for the multiplier components is 6 clock cycles, the adders 10 clock cycles, and the rest of the datapath (memory, shift register, etc.) 7 clock cycles. Also assume that there are 10,000,000 words in the input stream (a new input word is inputted every clock cycle). The board clock frequency is 25 MHz. (4 points)

How many nanosec before the first result is outputted?

$$\frac{1}{100 MHz} = 10 ns$$

$$7 + 6 + 10 + 10 = 33 \text{ clock cycles}$$

$$33 \times 40 ms = 1320 ms$$

How many clock cycles before the second result is outputted?

34 clock cycles

How many clock cycles before all the results are outputted?

33 + \(10,000,000 - 1\) clock cycles
2. **ASM and VHDL.**

(a) Shown in Figure 1 (next page) is the VHDL specification of an ASM controller. Analyze the VHDL code and complete the following timing diagram: Specify the values for state (0, 1, 2, or 3), and outputs P, Q, Y, and Z. Note that there are two timing diagrams, each is independent of the other. For the last one, the initial state is given (as state = 3).

Please show delays.
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY T2Prob1 IS
  PORT ( Clock, Resetn, X, R, S : IN STD_LOGIC;
         P, Q, Y, Z : OUT STD_LOGIC );
END T2Prob1;

ARCHITECTURE Behavior OF T2Prob1 IS
  SIGNAL state : STD_LOGIC_Vector (1 DOWNTO 0);
BEGIN
  PROCESS ( Resetn, Clock )
  IF Resetn = '0' THEN
    state <= "00" ;
  ELSIF (ClockEVENT AND Clock = '1') THEN
    CASE state IS
    WHEN "01" =>
      IF S = '0' THEN state <= "11" ;
      ELSE state <= "00" ;
      Q <= '1';
    END IF;
    WHEN "11" =>
      state <= "01" ;
    WHEN "00" =>
      IF X = '0' THEN state <= "00" ;
      ELSE state <= "11" ;
    END IF;
    WHEN OTHERS =>
      state <= "00" ;
    END CASE ;
  END IF;
END PROCESS;

P <= '1' WHEN state = "01" ELSE '0';

PROCESS (state, R, S)
BEGIN
  Y <= '0';
  Z <= '0';
  CASE state IS
  WHEN "01" =>
    IF S = '0' THEN Y <= '1';
    ELSE Z <= '1';
  END IF;
  WHEN "11" => IF R = '1'
    THEN Y <= '1';
  END IF;
  WHEN OTHERS =>
    END CASE;
  END PROCESS;
END Behavior;

Figure 1. To be used for problem 2

output of a flipflop

-- Note: Q is an output

-- P is an output

A * is an output of a flipflop; it is one clock cycle later.
3. Cyclone II Logic Element (LE)

PROCESS (IN1, IN2, IN3) BEGIN
CASE IN1 IS
WHEN '0' =>
   Z1 <= IN2 AND IN3;
WHEN OTHERS =>
   Z1 <= IN2 OR IN3;
END CASE;
IF (Clock'EVT AND Clock = '1') THEN
   IF IN4 = '1' THEN Z2 <= '0';
   ELSIF IN1 = '1' THEN Q <= IN3;
   END IF;
END IF;
END PROCESS;

Given the above PROCESS block, implement it in the above Cyclone II logic element (LE). Put your answers below. Each signal should be connected to 0, 1, X ("don't care"), NC (for not connected), or a signal name. If it is "don't care", you must put X (not 0 or 1).

(A) | (B) | (C) | (D) | (E) | (F) | (G) | (H) | (I) |
---|---|---|---|---|---|---|---|---|
   | IN4 | 0  |   |   | clock | IN1 | or IN4 |
---|---|---|---|---|---|---|---|---|
4. AltSynRam Problem

Complete the following timing diagram. Assume all flip-flops are initialized to '0'. Both RAM's has the same data (ramdat.mif).

<table>
<thead>
<tr>
<th>Name</th>
<th>0ps</th>
<th>50.0ns</th>
<th>100.0ns</th>
<th>150.0ns</th>
<th>200.0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aDATAIn</td>
<td>51</td>
<td>51</td>
<td>52</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>aWRaddr</td>
<td>02</td>
<td>01</td>
<td>03</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>aWRen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bSimpleRDaddr</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
</tr>
<tr>
<td>bDATAIn</td>
<td>50</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bWRaddr</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
</tr>
<tr>
<td>bWRen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bSimpleDualZ</td>
<td>80</td>
<td>80</td>
<td>81</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>aTrueDualZ</td>
<td>82</td>
<td>82</td>
<td>51</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>bTrueDualZ</td>
<td>80</td>
<td>80</td>
<td>81</td>
<td>81</td>
<td>50</td>
</tr>
</tbody>
</table>

ramdat.mif

Depth = 256;
Width = 8;
Address_radix = hex;
Data_radix = hex;
Content
Begin
00 : 80; 07 : 87;
01 : 81; 08 : 88;
02 : 82; 09 : 89;
03 : 83; 0A : 8A;
04 : 84; 0B : 8B;
05 : 85; 0C : 8C;
06 : 86; etc.

writer conflict either OK
5. Using altsyncram to implement a FIFO (First-in-first-out) component

**FIFO (block diagram)**

- `data[31..0]`: 32-bit data
- `q[31..0]`: 32-bit output
- `wrreq`: Write request
- `rdreq`: Read request
- `clock`: Clock input
- `sclr`: Synchronous clear

32 bit X 256 words

**sclr**: Synchronous clear to "empty" the FIFO

**rdreq**:
- 0: The output `q[31..0]` will hold the last value outputted from the FIFO.
- 1: The next value in the FIFO will be outputted from the FIFO `q[31..0]` at the next active clock transition.

**wrreq**:
- 0: The output `q[31..0]` will hold the last value outputted from the FIFO.
- 1: `data[31:0]` will be written in next location in the FIFO at the next active clock transition. (Output `q[31:0]` will hold last value outputted.)

**empty**:
- 0: The FIFO is not empty (Some inputted values have not been outputted).
- 1: The FIFO is empty. If a "rdreq" is asserted, then "junk" data will be outputted.

(a) Give me the VHDL statement(s) to produce the "empty" output. (3 pts.)

```
IF wraddress = rdaddress
THEN empty <= '1';
ELSE empty <= '0';
END IF;
```

(b) Complete the design of the FIFO by making the required connections below (For clarity, use labels when appropriate). Add any logic if necessary. (12 pts.)

**FIFO (design)**
6. FIR filter Datapath component, using GENERATE statement

Shown on the next page is the example code that we discussed in class for a 4-stop FIR filter.

(a) Give me the code required to implement the required 4-bit shift registers to produce reg(4 DOWNTO 1). (6 pts.)

- Restriction: you have to GENERATE and PORT MAP statements for this part.
- Assume that you have the following component:

  ```
  COMPONENT dff IS
    PORT ( clock : IN STD_LOGIC;
           d : IN STD_LOGIC;
           q : OUT STD_LOGIC );
  END COMPONENT;
  ```

- Important note: reg(0) is the input to the reg(1) flip-flops which contain the most recent data - reg(0) should be connected to the datapath input signal named "inData".

(Put your answer here, including any new TYPE or SIGNAL definitions)

```
reg(1) reg(2) reg(3) reg(4)
```

```
reg(0) <= inData;
```

(b) Give me the code required to implement all the adders of the FIR filter datapath component, producing the datapath output signal named "out". (8 pts.)

- Restriction: When possible, you have to GENERATE and PORT MAP statements.

(Put your answer here, including any new TYPE or SIGNAL definitions)

```
add2: FOR i IN 1 TO 2 GENERATE
  adds: adder PORT MAP (clock => clock,
                        dataa => mout (2 * i - 1),
                        datab => mout (2 * i),
                        result => adderOUT (i));
END GENERATE;
```

```
bottomadd: adder PORT MAP (clock => clock,
                          dataa => adderOUT (1);
                          datab => adderOUT (2);
                          result => dpathOUT);
```
(c) Given the following decimal number -10.6875 (which is -1010.1011 in binary), convert it to the IEEE 754 floating-point format: (2 pts.)

- Bit 31: sign bit
- Bits 30-23: exponent (with a bias of 127 = 111 1111 in binary)
- Bits 22-0: mantissa

\[ -1 \times 2^3 \]

\[ 10101011 \]

\[ +11 \]

\[ 10000000 \]

-- snippet of code to demonstrate Multi-dimensional arrays and GENERATE statement

ARCHITECTURE struct OF datapath IS

-- Definition of other components

COMPONENT multiplier IS
  PORT ( clock : IN STD_LOGIC;
          dataa : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
          datab : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
          result: OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END COMPONENT;

COMPONENT adder IS
  PORT ( clock : IN STD_LOGIC ;
          dataa : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
          datab : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
          result: OUT STD_LOGIC_VECTOR (31 DOWNTO 0) );
END COMPONENT;

SUBTYPE signalVectors IS STD_LOGIC_VECTOR(31 DOWNTO 0);
TYPE array4OfSignals IS ARRAY(4 DOWNTO 1) OF signalVectors;
TYPE array5OfSignals IS ARRAY(4 DOWNTO 0) OF signalVectors;

SIGNAL coeff: array4OfSignals;
SIGNAL reg: array5OfSignals; -- reg(4 DOWNTO 1) are outputs of the 4 registers
-- reg(0) is the input to the flip-flops with the most recent data
SIGNAL mout: array4OfSignals;

BEGIN
  -- Assume that coeff(4 DOWNTO 1) have been assigned here for your use.
  -- shift register code

  mults: FOR i IN 1 to 4 GENERATE
    multArray : multiplier PORT MAP (clock=>clk, dataa=>coeff(i),
                                   datab=>reg(i), result=>mout(i));
  END GENERATE mults;

  -- code for adders

END struct;