1. **T2ASM/VHDL.** Given below is a timing diagram (functional simulation) showing the desired timings among the states and signals of a controller.

<table>
<thead>
<tr>
<th>Name</th>
<th>100.0ns</th>
<th>200.0ns</th>
<th>300.0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resetn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>SA</td>
<td>SB</td>
<td>SA</td>
</tr>
<tr>
<td>InBit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BuffFull</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OutFlag</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Construct an ASM diagram that will produce the above behavior. (8 pts.)

(b) Complete the VHDL specification (on the next page) of your ASM diagram. Please don’t change the structure of the code. In other words, **you have to use the first CASE statement** to implement state transitions and **the second CASE statement** to implement the conditional and unconditional outputs. (10 pts.)

```vhdl
ENTITY Test2P1 IS
    PORT ( Clock, Resetn, Inbit : IN STD_LOGIC; -- Resetn is active low, asynchronous BuffFull, OutFlag : OUT STD_LOGIC );
END Test2P1;
```
1(b) ARCHITECTURE ASMArch OF Test2P1 IS
  TYPE ASMstateType IS (SA, SB, SC); -- User defined signal type
  SIGNAL state : ASMstateType;
BEGIN
  PROCESS (Resetn, clock) -- state transitions
  BEGIN
    IF Resetn = '0' THEN
      state <= SA;
    ELSIF (clock EVENT AND clock = '1') THEN
      CASE state IS
        WHEN SA =>
          IF InBit = '0' THEN state <= SC;
          ELSE state <= SB;
          END IF;
        WHEN SB =>
          state <= SA;
        WHEN SC =>
          IF InBit = '0' THEN state <= SC;
          ELSE state <= SA;
          END IF;
      END CASE;
    END IF;
  END PROCESS;
  PROCESS (state, InBit, BufFull, OutFlag) -- conditional and uncond. outputs
  BEGIN
    BufFull <= '0'; -- all outputs default to '0'
    OutFlag <= '0';
    CASE state IS -- You have to use this CASE statement for the outputs.
      WHEN SA =>
        OutFlag <= '1';
        IF InBit = '0' THEN BufFull <= '1';
        END IF;
      WHEN SB =>
        BufFull <= '1';
      WHEN SC =>
        BufFull <= '1';
    END CASE;
  END PROCESS;
END ASMArch;
2. Testbench for GCD Calculator Lab. Given below is the testbench for the GCD calculator.

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

entity gcd_tb is
end gcd_tb;

architecture TB of gcd_tb is

constant WIDTH : positive := 32;
constant TIMEOUT : time := 1 ns;
signal clkEn : std_logic := '1';
signal clk : std_logic := '0';
signal rst : std_logic := '1';
signal go : std_logic := '0';
signal done : std_logic;
signal x : std_logic_vector(WIDTH-1 downto 0);
signal y : std_logic_vector(WIDTH-1 downto 0);
signal output : std_logic_vector(WIDTH-1 downto 0);

begin

UUT : entity work.gcd(FSMD) <=
GENDER MAP (1
WIDTH => WIDTH
PORT MAP
  clock => clk,
  reset => rst,
  go => go,
  done => done,
  a => x,
  b => y,
  output => output);

clk <= not clk and clkEn after 20 ns;

process

  function GCD (x, y : integer)
  return std_logic_vector is
    variable tmpX, tmpY : integer;
  begin
    tmpX := x;
    tmpY := y;
    while (tmpX /= tmpY) loop
      if tmpX < tmpY then
        tmpY := tmpY-tmpX;
      else
        tmpX := tmpX-tmpY;
      end if;
    end loop;
    return std_logic_vector(to_unsigned(tmpX, WIDTH));
  end GCD;

(a) Explain:
What "work" indicates? gcd entity is in the same folder as this code.
What "FSMD" indicates?
gcd can have several architectures, the one being used here is named FSMD.

(b) Put the PORT MAP statement here to "connect" the entity gcd to the testbench. Note that some signal names in gcd (clock, reset, a, and b) have been changed.

(c) How is the GCD function used in this testbench?
for each x and y value, this function calculates the correct output so it can be used in an assert statement to verify the correctness of the hardware.
```
begin
  clkEn <= '1';
  rst <= '1';
  go <= '0';
  x <= std_logic_vector(to_unsigned(0, WIDTH));
  y <= std_logic_vector(to_unsigned(0, WIDTH));
  wait for 200 ns;
  rst <= '0';
  for i in 0 to 5 loop
    wait until clk'event and clk = '1';
    end loop; -- i
  for i in 1 to 2**WIDTH-1 loop
    x <= std_logic_vector(to_unsigned(i, WIDTH));
    for j in 1 to 2**WIDTH-1 loop
      go <= '1';
      y <= std_logic_vector(to_unsigned(j, WIDTH));
      wait until done = '1' for TIMEOUT;
      assert(done = '1') report "Done never asserted." severity warning;
      go <= '0';
      wait until clk'event and clk = '1';
      end loop;
    end loop;
    clkEn <= '0';
    report "DONE!!!!!!" severity note;
    wait;
  end process;
end TB;

(d) Explain the function of loop "i" and this statement.
This loop loops through all possible x values \(2^{\text{WIDTH}} - 1\).
For each i, i is casted and assigned to x.

(e) Explain the function of loop "j" and this statement.
For each x value, this loops through all y values \(2^{\text{WIDTH}} - 1\).
For each j, j is casted and assigned to y.

(f) Put back an assert statement here to verify that the input values produce a correct output.

\[ \text{ASSERT} \ (\text{output} = \gcd (i, j)) \text{ report "wrong" severity warning} \]

(g) How many bits are the input/output signals? 32
Complete the following timing diagram. (Please put values in hex.)

- Assume all flip-flops are initialized to '0'.
- Both RAM's has the same data (ramdat.mif).
- When there is a read/write conflict on the same address, read old value.

**Table:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>90</td>
</tr>
<tr>
<td>01</td>
<td>91</td>
</tr>
<tr>
<td>02</td>
<td>92</td>
</tr>
<tr>
<td>03</td>
<td>93</td>
</tr>
<tr>
<td>04</td>
<td>94</td>
</tr>
<tr>
<td>05</td>
<td>95</td>
</tr>
<tr>
<td>06</td>
<td>96</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Legend:**
- **CLOCK**
- **aDATAin**
- **aAddr**
- **aWREN**
- **bDATAin**
- **bAddr**
- **bWREN**
- **SinglePortZ**
- **aTrueDualZ**
- **bTrueDualZ**
4 (a). Complete the following VHDL specification for it. (6 pts)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Test2LEprob IS
  PORT ( IN1, IN2, CLEAR, LD, CLOCK: IN STD_LOGIC; -- synchronous
          Z1, Z2: OUT STD_LOGIC ); -- CLEAR, LD
END Test2LEprob;

ARCHITECTURE LEArch OF Test2LEprob IS
SIGNAL temp22: STD_LOGIC;
BEGIN
  PROCESS (CLOCK)
  BEGIN -- for maximum credit, all your code should be inside this PROCESS statement.
    IF (CLOCK 'EVENT AND CLOCK = '1') THEN
      IF CLEAR = '1' THEN
        Z2 <= '0';
      ELSIF LD = '1' THEN
        temp22 <= (NOT IN1 AND IN2) OR temp22;
      END IF;
      END IF;
      Z2 <= temp22; -- output of flip-flop
      Z1 <= (NOT IN1 AND IN2) OR temp22; -- no flip-flop
    END PROCESS;
  END PROCESS;
END LEArch;
4(b). You are to "program" the logic element (LE) of an Altera Cyclone III device to implement the circuit from the previous page. In other words, for each "location" (A through K), specify what should be connected to it. It can be 0, 1, X ("don't care"), or a signal name (like CLEAR). If it is "don't care", you must put X (not 0 or 1). Also specify the contents of the LUT (look up table). (14 pts)

Figure 2-3. LE in Normal Mode

<table>
<thead>
<tr>
<th>IN1</th>
<th>IN2</th>
<th>LD</th>
<th>CLEAR</th>
<th>CLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Put your answers here. 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) **either**
(B) CLEAR
(C) 10
(D) 1
(E) CLOCK
(F) LD+CLEAR
(G) 1
(H) 1
(I) 0
(J) X(not used)
(K) X(not used)

Contents of LUT:

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
5. FIR filter ASM problem

You are to design the controller (i.e., ASM chart on the next page) to fill the Input RAM for a Lab 6 FIR filter (from Location 0 to Location 256). Note that the Input RAM plays the same role as the Input ROM in Lab 6.

- The controller will wait until it receives a RDY = ‘1’ from the Lab 6 FIR filter, then it will proceed to fill the Input RAM.
  - It will check to see if the FIFO is empty
    - If empty = ‘1’, it will wait.
    - If empty = ‘0’,
      - It will request a 32-bit value from the FIFO (set rdreq = ‘1’). The next value in the FIFO will be outputted from the FIFO $q[31..0]$ at the next active clock transition.
      - This value should be written into the next location in the Input RAM.
  - This will be done until the Input RAM is full (256 locations), each time making sure the FIFO is not empty.
- When finished, it will signal the Lab 6 FIR filter by setting Start = ‘1’ and return to the Wait state.

Note:

Function of 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.
- empty:
  - 0: The FIFO is not empty (there are some values in it).
  - 1: The FIFO is empty.
5. continued
(a) Complete the block diagram of the Controller, specifying all the Controller inputs and Controller outputs. (3 pts.)

(b) Put the ASM chart for the Controller here. (15 pts.)
For Lab 4, assuming the board clock frequency is 100 MHz and the constant to be used for HSYNC_END is 2988, what is the time in uS (microseconds) for E?

For credit, show work here. (5 pts.)

\[ H_{SYNC\_END} = 2988 \]

\[ T = \frac{100 \times 10^6}{2988} = 33.5 \text{ uS} \]

\[ H = 2988 \times 10 \text{ uS} = 29.88 \text{ uS} \]

\[ E = H - D - B = 29.88 - 25.17 - 3.77 = 0.94 \text{ uS} \]

(b) In Brandon Pollack’s lecture on the basics of single-cycle MIPS computer architecture, one way to implement the next address logic is shown in the figure below, which is fast but expensive in terms of extra hardware. What was the simple optimization given by Brandon which will reduce the hardware? Please give a one-sentence answer. (3 pts.)

Instead of using two 32-bit adders, use only one and use the carry in to increment the pc.

Next Address Logic: Expensive and Fast Solution

* Using a 32-bit PC:
  * Sequential operation: PC<31:2> = PC<31:2> + 1
  * Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt(Imm16)
  * In either case: Instruction Memory Address = PC<31:2> concat "00"
ENTITY __entity_name IS
  PORT(__input_name, __input_name
       __input_vector_name
       __bidir_name, __bidir_name
       __output_name, __output_name
       : IN STD_LOGIC;
       : IN STD_LOGIC_VECTOR(_high downto _low);
       : INOUT STD_LOGIC;
       : OUT STD_LOGIC);
END __entity_name;

ARCHITECTURE a OF __entity_name IS
  SIGNAL _signal_name : STD_LOGIC;
  SIGNAL _signal_name : STD_LOGIC;
BEGIN
  -- Process Statement
  -- Concurrent Signal Assignment
  -- Conditional Signal Assignment
  -- Selected Signal Assignment
  -- Component Instantiation Statement
END a;

SIGNAL __signal_name : __type_name;

__instance_name: __component_name GENERIC MAP (__component_par => __connect_par)
  PORT MAP (__component_port => __connect_port,
            __component_port => __connect_port);

WITH __expression SELECT
  _signal <= __expression WHEN __constant_value,
  _expression WHEN __constant_value,
  __expression WHEN __constant_value,
  _expression WHEN __constant_value;

  _signal <= __expression WHEN __boolean_expression ELSE
  __expression WHEN __boolean_expression ELSE
  __expression;

IF __expression THEN
  __statement;
  __statement;
ELSIF __expression THEN
  __statement;
  __statement;
ELSE
  __statement;
  __statement;
END IF;

WAIT UNTIL __expression;

CASE __expression IS
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN OTHERS =>
    __statement;
    __statement;
END CASE;
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:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (18 pts)</td>
<td></td>
</tr>
<tr>
<td>2 (16 pts)</td>
<td></td>
</tr>
<tr>
<td>3 (20 pts)</td>
<td></td>
</tr>
<tr>
<td>4 (20 pts)</td>
<td></td>
</tr>
<tr>
<td>5 (18 pts)</td>
<td></td>
</tr>
<tr>
<td>6. (8 pts)</td>
<td></td>
</tr>
</tbody>
</table>

Total [ ]

Re-Grade Information:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________