**FA19, MT1 (Logic)**

1) *It's all logical...* (12 points, 15 minutes)

You need to design a circuit for the following truth table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Y = A'B' + B'C'D' + ABCD'**

b) Draw a Karnaugh map for this expression.

![Karnaugh Map](image)

1) *It's all logical...* (continued)

- **b)** Draw a Karnaugh map for this expression.
- **c)** Use the Karnaugh map from part b) to simplify the circuit and write the simplified sum-of-product representation.

**Y = A'B' + B'C'D' + ABCD'**

a) Write the sum of products expression for output Y directly from this truth table.

**Y = A'B'C'D' + A'B'C'D + A'B'CD' + A'B'C D + AB'C'D' + ABCD'**
Follow-up Question:
What kind of machine is this? Mealy

Follow-up Question:
How many registers do we need to maintain state? 4
FA19, MT1 (FSM)

2) State Machines (continued)
   b) If the state is represented by a four-bit register $S[3:0]$ and $S4 = 4'b0100$ and $S9 = 4'b1001$, complete the following diagram:

Mealy Machine: Out depends on state + inputs.

Clarification:
Only draw the logic relating to out, you may ignore CL0-3.
Question 1
Same circuit. Signed operands do not affect the circuit synthesized for addition.

Question 2
No. Without indicating that a is signed, >>> will not be inferred to an arithmetic shift.
Question 3
Leaves the assignment up to the synthesis tool, which depending on settings, could set X to 1 or 0 (or perhaps unconnected).

Question 4
Consider 2'b11. Left sets to 4, second sets to 3.

Assume that a case item with an “x” is matched when the case expression is 0 or 1. E.g. 3'b11x matches 3'b110 and 3'b111.
b) (2 pts) What circuit does this Verilog synthesize to?

```verilog
wire [1:0] a;
wire clk;
reg [1:0] b, c, d;
always @(posedge clk) begin
    b <= a;
    c = b + 2'b1;
    d <= c + 2'b2;
end
```

**First 1.** Note that `c` is available for assignment to `d` immediately, but it will evaluate to a register due to its location with the `always@(posedge clk)`.
3) **Datapathology** (24 points, 25 minutes)

The datapath below implements the RV32I instruction set.

---

**a)** In the RISC-V datapath above, mark what is used by a `jal` instruction.

`jal` (Jump and link): $R[rd] = pc+4; pc = pc + \{imm,1b\}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>offset[20:1]</td>
<td>dest</td>
<td>JAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Select one per row**

- **PCSel Mux:**
  - PC + 4 branch
  - Imm branch
  - (don’t care)

- **ASel Mux:**
  - pc branch
  - Reg[r1] branch
  - (don’t care)

- **BSel Mux:**
  - imm branch
  - Reg[r2] branch
  - (don’t care)

- **WBSel Mux:**
  - pc + 4 branch
  - alu branch
  - mem branch
  - (don’t care)

---

**Select all that apply**

- **Datapath units:**
  - Branch Comp
  - Imm. Gen
  - Load Extend

- **RegFile:**
  - Read Reg[r1]
  - Read Reg[r2]
  - Write Reg[rd]
3) Datapathology (24 points, 25 minutes)
The datapath below implements the RV32I instruction set.

b) In the RISC-V datapath above, mark what is used by a `auipc` instruction.

```
auipc (add upper immediate to pc): R[rd] = pc + {imm,12b'0}
```

Select one per row

<table>
<thead>
<tr>
<th></th>
<th>PCSel Mux</th>
<th>ASel Mux</th>
<th>BSel Mux</th>
<th>WB Sel Mux</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“pc + 4” branch</td>
<td>“pc” branch</td>
<td>“imm” branch</td>
<td>“pc + 4” branch</td>
<td>(don’t care)</td>
</tr>
<tr>
<td></td>
<td>“alu” branch</td>
<td>Reg[rs1] branch</td>
<td>Reg[rs2] branch</td>
<td>“alu” branch</td>
<td>(don’t care)</td>
</tr>
<tr>
<td></td>
<td>(don’t care)</td>
<td>(don’t care)</td>
<td>(don’t care)</td>
<td>(don’t care)</td>
<td></td>
</tr>
</tbody>
</table>

Select all that apply

<table>
<thead>
<tr>
<th></th>
<th>Datapath units:</th>
<th>RegFile:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Branch Comp</td>
<td>Read Reg[rs1]</td>
</tr>
<tr>
<td></td>
<td>Imm. Gen</td>
<td>Read Reg[rs2]</td>
</tr>
</tbody>
</table>
Problem 2: VERILOG LOGIC [15 points, 15 minutes]

Consider the following Verilog module:

```verilog
module my_module(
    input clk, load,
    input [2:0] in,
    output reg [2:0] out
);
always @ (posedge clk) begin
  if (load) out <= in;
  else begin
    out[0] <= out[0];
    if (out[1]) out[1] <= ~out[1];
    if (out[2]) out[2] <= ~out[2];
  end
endmodule
```

(a) Draw the circuit diagram for this design. You may use the module inputs (e.g. in[0]), constants, muxes, inverters, and/or 2-input logic gates.

(b) Say we load 3'b011 using our load and in input signals. We then deassert load. What is the value of out for the first 5 cycles?

<table>
<thead>
<tr>
<th>Cycle</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>000</td>
</tr>
</tbody>
</table>
Now, consider this similar module:

```verilog
module my_module;
  input clk, load,
  input [2:0] in,
  output reg [2:0] out
);
  always @(posedge clk) begin
    if (load) out = in;
    else begin
      out[0] = ~out[0];
      if (out[0])
        out[1] = ~out[1];
      if (out[0] & out[1])
        out[2] = ~out[2];
    end
  end
endmodule
```

(c) Say we load $\text{3F'011}$ again using our $\text{load}$ and $\text{in}$ input signals. What is the value of $\text{out}$ for the first 6 cycles?

<table>
<thead>
<tr>
<th>Cycle</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td>000</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>110</td>
</tr>
</tbody>
</table>