Before looking at the problems, let’s look at some important vocabulary for this section.

**Combinational circuit:** a circuit in which we combine different gates in the circuit, such as encoders, decoders, multiplexers, and demultiplexers. Combinational circuits can have n number of inputs and m number of outputs, and have no cycles (feedback) or state elements.

**Circuits with feedback:** circuits with cycles that can hold state. An example of such is a D Latch circuit.

**D Latch:** A D latch circuit’s output depends on a clock. If the clock is high, the input passes to output. If the clock is low, the latch holds its output. For the D latch, the latch is asynchronous and the outputs can change as soon as the inputs do.

**D Flip Flop:** a circuit with two stable states which can store one bit of state information. The output changes state by signals applied to one or more control inputs. A flip-flop is edge-triggered and only changes state when a control signal goes from high to low or low to high.
**Finite State Machine:** a computation model that can be used to simulate sequential logical and model problems in many fields such as math and artificial intelligence. An example of a finite state machine is shown below:

![Finite State Machine Diagram]

**Problem 1**

Write the truth tables for both a D latch and a D flip-flop. (Note: Q* is the next state of Q)

<table>
<thead>
<tr>
<th>D latch Truth Table</th>
<th>D flip-flop Truth Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td><strong>D</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

**Problem 2**

The following code implements a simple sequential circuit as a module that computes a function over a series of steps. Read the code and answer the questions about it below.

```plaintext
interface Foo;
    method Action start(Bit#(32) aIn);
    method ActionValue#(Bit#(32)) getX();
    method Bit#(32) getI();
endinterface

module mkFoo(Foo);
    Reg#(Bit#(32)) a <- mkReg0;
    Reg#(Bit#(1)) validx <- mkReg0;
    Reg#(Bit#(32)) x <- mkRegU();
    Reg#(Bit#(32)) i <- mkRegU;
endmodule
```
function Bit#(32) computeB(Bit#(32) in);
    Bit#(32) out = 0;
    if ( in >= 1 ) out = 1;
    if ( in >= 5 ) out = 5;
    if ( in >= 10 ) out = 10;
    return out;
endfunction

rule doComputeStep if (a > 0 && validx == 0);
    let b = computeB(a);
    a <= a - b;
    x <= a;
    validx <= 1;
    i <= i + 1;
endrule

method Action start(Bit#(32) aIn) if (a==0);
    a <= aIn;
    i <= 0;
endmethod

method ActionValue#(Bit#(32)) getX() if (validx == 1);
    validx <= 0;
    return x;
endmethod

method Bit#(32) getI() if (a==0);
    return i;
endmethod
endmodule

(A) (4 points) The module using mkFoo is invoking the `getX` method of mkFoo at every clock cycle. Remember that an invoked method can only execute when it is ready. If the `start` method is called the first time with `aIn = 28`, what will the output sequence from `getX()`? What is the output of `getI()` after the `start` method is called?

1. Return value sequence of `getX()`: 28, 18, 8, 3, 2, 1
2. Return value of `getI()`: 6

(B) (2 points) Suppose we get rid of register x and modify the rule `doComputeStep` and method `getX` as follows.

```
rule doComputeStep if (a > 0 && validx == 0);
    let b = computeB(a);
```

\[
a \leq a - b;
\]
\[
x \leftarrow a;
\]
\[
validx \leq 1;
\]
\[
i \leq i + 1;
\]
endrule

\[\text{method ActionValue}\#\text{Bit}\#(32) \text{getX()} \text{ if (validx} = 1);\]
\[
validx \leq 0;
\]
\[
\text{return } x;\]
\[
\text{return } a;\]
endmethod

Does this change the output sequence of \text{getX()}\ of the module?

(C) (2 points) Ignoring the changes in (C), suppose we modify the guard of \text{start} in the original code to \((a==0 \&\& validx == 0)\). Does this change the output sequence of \text{getX()}?  

\[
(\text{circle one}) \quad \text{Yes} \quad \ldots \quad \text{No} \quad \ldots \quad \text{Can’t tell}
\]

Problem 3.

Consider a "divisible-by-3" FSM that accepts a binary number entered one bit at a time, most significant bit first. The FSM has a one-bit output that indicates if the number entered so far is divisible by 3.

If the value of the number entered so far is N, then after the digit b is entered, the value of the new number \(N'\) is \(2N + b\). This leads to the following transition diagram where the states are labeled with the value of \(N \mod 3\).

\[
\begin{array}{ccc}
\text{Initial state} & \text{b=0} & \text{b=1} \\
0 & 1 & 2 \\
\text{b=0} & 1 & 0 \\
\text{b=1} & 0 & 1 \\
\end{array}
\]

(A) Construct a truth table for the FSM logic. Inputs include the state bits and the next bit of the number; outputs include the next state bits and the output.

\[
\begin{array}{ccc|ccc}
S1^t & S0^t & b & S1^{t+1} & S0^{t+1} & \text{output} \\
\hline
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]
0 1 0 | 1 0 0
0 1 1 | 0 0 0
1 0 0 | 0 1 0
1 0 1 | 1 0 0

(B) Based on the truth table, implement the FSM using D flip-flops.

\[
\begin{align*}
S0^{t+1} &= \sim S1' \sim S0' \ b + S1' \sim S0' \ b \\
S1^{t+1} &= \sim S1' \ S0' \sim b + S1' \sim S0' \ b \\
\text{Output} &= \sim S1' \sim S0'
\end{align*}
\]
Problem 4.

In this problem, we construct a sequential circuit to compute the $N^{th}$ Fibonacci number denoted by $F_N$. The following recurrence relation defines the Fibonacci sequence.

$$F_0 = 0, F_1 = 1, F_N = F_{N-1} + F_{N-2} \quad \forall \ N \geq 2$$

The circuit is similar to the GCD circuit discussed in the lecture. There are two registers $x$ and $y$ that store the Fibonacci values for two consecutive integers. In addition, a counter register $i$ is initialized to $N-1$ and decremented each cycle. The computation stops when register $i$ goes down to 0 and the result ($F_N$) is available in register $x$.

(A) What are the initial values for registers $x$ and $y$?

The initial values are $y = 0$, $x = 1$ respectively.

(B) Derive the next state computation equations for the three registers.

$$i^{t+1} = i^t - 1$$
$$y^{t+1} = x^t$$
$$x^{t+1} = x^t + y^t$$

(C) Derive the logic for the enable signal that determines when the registers are updated using the next state logic. Note that all three registers are controlled by a single enable signal.

This ensures that computation stops when counter $i$ becomes 0.

(D) Implement the sequential circuit using the next state and enable logic derived above.