Problem 1. Combinational Circuits (16 points)

(A) (4 points) Implement the function `lt2` that checks if `a` is less than `b` in Minispec. You may only use the bitwise logical operators: ~, &, |, ^. Assume that `a` and `b` are unsigned. When using negation (~), please include parentheses around the entire term being negated to clarify order of operation.

```plaintext
function Bit#(1) lt2 (Bit#(2) a, Bit#(2) b);
    return ________________________________________________________________;

endfunction
```
(B) (2 points) Fill out the truth table for the \( \text{lt2} \) function.

<table>
<thead>
<tr>
<th>( a[1] )</th>
<th>( a[0] )</th>
<th>( b[1] )</th>
<th>( b[0] )</th>
<th>( \text{lt2} )</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>

(C) (3 points) Manually synthesize the \( \text{lt2} \) function into a combinational circuit using only inverters, and 2-input AND, OR, and XOR gates. Make sure to clearly label all inputs and outputs. **For full credit, you should use at most 8 gates.**
(D) (3 points) Consider the recursive Minispec function \texttt{f_recursive}. Rewrite this function iteratively with a for loop. Your rewritten function should produce the same result but does not need to synthesize into the same circuit.

\begin{verbatim}
function Bit#(1) f_recursive#(Integer n) (Bit#(n) x);
    if (n == 1) return x;
    else begin
        Bit#(1) bottom = f_recursive#(n/2)(x[n/2-1:0]);
        Bit#(1) top = f_recursive#(n-n/2)(x[n-1:n/2]);
        return bottom ^ top;
    end
endfunction

function Bit#(1) f_iterative#(Integer n) (Bit#(n) x);

endfunction
\end{verbatim}
(E) (4 points) Consider the following Minispec function. Determine the result of running the function test(x) on each of the inputs specified below. Provide your result in binary.

```plaintext
function Bit#(4) test(Bit#(4) x) = case (x)
  2 : x | 1;
  3 : x + 3;
  5 : x << 2;
 10 : x & 6;
  default: x ^ (x + 1);
endcase;
```

Result of calling test(2) in binary (0b):____________________

Result of calling test(5) in binary (0b):____________________

Result of calling test(6) in binary (0b):____________________

Result of calling test(10) in binary (0b):____________________
Problem 2. Combinational and Sequential Logic Timing (19 points)

(A) (4 points) Given the timing parameters in the table below, what are the propagation and contamination delays of this circuit?

Gate | $t_{pd}$ (ns) | $t_{cd}$ (ns)
---|---|---
XOR2 | 2 | 1.5
INV1 | 1.5 | 1
AND2 | 3 | 2

Propagation delay (ns): _________

Contamination delay (ns): _________
(B) (4 points) Given the timing parameters in the table below, please indicate whether or not each constraint is satisfied by the circuit below and explain your answer. The registers are driven by a common clock with period $t_{clk} = 10\text{ns}$. Assume that $\text{IN}$ satisfies the hold time constraint.

<table>
<thead>
<tr>
<th>Component</th>
<th>$t_{pd}$ (ns)</th>
<th>$t_{cd}$ (ns)</th>
<th>$t_{SETUP}$ (ns)</th>
<th>$t_{HOLD}$ (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register (R1, R2, R3)</td>
<td>2</td>
<td>1</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>Combinational Logic (CL1, CL2)</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Setup time constraint (circle one): SATISFIED NOT SATISFIED

Explanation:

Hold time constraint (circle one): SATISFIED NOT SATISFIED

Explanation:
(C) (4 points) We would like for this circuit to be able to function with a clock period of 9ns. Using the replacement parts below, modify the circuit so that it can operate using a clock period of 9ns while satisfying the setup and hold time constraints. For full credit, minimize the number of replacements made. To help you out, the circuit and original component timings are replicated above.

<table>
<thead>
<tr>
<th>Component</th>
<th>( t_{pd} ) (ns)</th>
<th>( t_{cd} ) (ns)</th>
<th>( t_{SETUP} ) (ns)</th>
<th>( t_{HOLD} ) (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register - Original</td>
<td>2</td>
<td>1</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>Combinational Logic - Original</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>( t_{pd} ) (ns)</th>
<th>( t_{cd} ) (ns)</th>
<th>( t_{SETUP} ) (ns)</th>
<th>( t_{HOLD} ) (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register-New</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>CombinationalLogic-New</td>
<td>2</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

CL1: REPLACED (with CombinationalLogic-New) NOT REPLACED
CL2: REPLACED (with CombinationalLogic-New) NOT REPLACED
R1: REPLACED (with Register-New) NOT REPLACED
R2: REPLACED (with Register-New) NOT REPLACED
R3: REPLACED (with Register-New) NOT REPLACED
(D) (3 points) Demonstrate that your updated circuit satisfies both the setup and hold time constraints.

(E) (4 points) After making the changes in part (C), what is the minimum clock period that will satisfy the setup and hold time constraints? What are the propagation and contamination delay of the updated circuit.

Minimum $t_{CLK}$ after replacements (ns): ________________

Propagation delay (ns): ________________

Contamination delay (ns): ________________
Problem 3. Finite State Machines (11 points)

Melon Usk bought his son YÆB-13 a brand new “Bop It”! The Bop It has five states, where each state corresponds to the Bop It asking the user to do some specific action: bop, pull, twist, flick, or spin. YÆB-13 thinks that the Bop It chooses its next state at random, but Melon actually hacked the toy so that he can choose the next state by inputting a zero or one into his Pear Watch.

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bop</td>
<td>0</td>
<td>Spin</td>
</tr>
<tr>
<td>Bop</td>
<td>1</td>
<td>Pull</td>
</tr>
<tr>
<td>Pull</td>
<td>0</td>
<td>Twist</td>
</tr>
<tr>
<td>Pull</td>
<td>1</td>
<td>Flick</td>
</tr>
<tr>
<td>Twist</td>
<td>0</td>
<td>Flick</td>
</tr>
<tr>
<td>Twist</td>
<td>1</td>
<td>Spin</td>
</tr>
<tr>
<td>Flick</td>
<td>0</td>
<td>Spin</td>
</tr>
<tr>
<td>Flick</td>
<td>1</td>
<td>Spin</td>
</tr>
<tr>
<td>Spin</td>
<td>0</td>
<td>Spin</td>
</tr>
<tr>
<td>Spin</td>
<td>1</td>
<td>Spin</td>
</tr>
</tbody>
</table>

(A) (4 points) Using the provided partial information, complete both the truth table and the state-transition diagram. In the diagram, fill in the missing state labels and label each transition with either 0 or 1 to indicate which input value causes the transition.

Add missing information to diagram and table above

(B) (2 points) Given the following inputs, determine what the next state will be (bop, pull, twist, flick, or spin) after the last input is processed. Assume the FSM starts at the Bop state for each series of inputs. Assume that an input of 100, for example, means the FSM got a 1 followed by two 0s.

(i) 0101

(ii) 111

(iii) 101

Final state: __________________

Final state: __________________

Final state: __________________
(C) (1 points) How many flip flops does the Bop It FSM require to encode all possible states?

Number of Flip Flops Required: __________________

(D) (2 points) Melon’s Pear Watch is glitching and he can now only input ones. After at least two more state transitions, what states of the Bop It will Melon no longer be able to access?

Which State(s) Become Inaccessible? ________________________________

(E) (2 points) Part E refers to a new FSM that is completely separate from the FSM in parts A-D. Below is a state-transition diagram for a state machine with possible inputs 0 and 1. Is the state-transition diagram valid? If yes, how many flip flops are needed to encode all possible states? If no, explain why not.

Is the state-transition diagram valid? (circle one):  VALID       INVALID

If VALID:

Number of Flip Flops Needed: _____________

If INVALID:

Explanation:
Problem 4. Sequential Circuits in Minispec (20 points)

Congratulations, you have been hired to program one of NASA’s 2022 ExoMars Mission rovers! The goal of your rover is to listen for user inputs and based on those inputs, move around a grid version of the surface of Mars and collect dirt samples (for science).

In particular, your rover will land on the grid at coordinate (0, 0) and may be told to move in some direction. Assume moving up increments \( y \), and moving right increments \( x \). You will have to be careful not to go out of bounds of the grid in order to avoid a perimeter made of space quicksand. Lucky for you, you have access to a function \( \text{isInBounds(Coordinate x, Coordinate y, UserInput direction)} \) which returns \( \text{True} \) if moving in the given direction from the given \((x, y)\) coordinate lands the rover in a coordinate which is in bounds. At any time, the rover may be told to excavate and collect dirt samples. It takes the rover 3 cycles to complete its dirt collection. The rover must not move while it is collecting dirt. Failure to give the rover enough time to collect its samples will result in failure of the mission.

(A) (10 points) Fill in the code below to track your rover’s state as a sequential circuit in Minispec! Recall that a Minispec module’s rule cannot directly call its method(s).

typedef Bit#(8) Coordinate;

typedef enum {Up, Down, Left, Right, Excavate} UserInput;

module Rover;
Reg#(Coordinate) x(____);
Reg#(Coordinate) y(____);

// Number of cycles left in the excavation process
Reg#(Bit#(2)) excavateStepsLeft(0);

input Maybe#(UserInput) in default = Invalid;

method Coordinate getXCoord = __________;
method Coordinate getYCoord = __________;
method Bool isExcavating = ____________________________;

rule tick;
  
  UserInput userIn;
  
  if (____________________) && (____________________) begin
    userIn = __________________;
    if (userIn != __________________) begin
      if (____________________) begin
        // Tells rover to move on next cycle
        move(userIn);
        case (____________________)
          __________________;
          __________________;
          __________________;
          __________________;
          __________________;
        endcase
      end
    end else begin
      // Tells rover to start excavation on next cycle
      excavate();
      __________________;
    end
  end else if (____________________) begin
    excavateStepsLeft <= __________________;
  end
endrule
endmodule
(B) (10 points) Consider the following minispec module, GCD, and its testbench GCDTest. Fill in the table below indicating the values of x, y, and result during each cycle. For result, either write Inv for Invalid, or the valid value.

typedef struct {Bit#(16) a; Bit#(16) b;} GCDArgs;

module GCD;
    Reg#(Bit#(16)) x(1);
    Reg#(Bit#(16)) y(0);

    method Maybe#(Bit#(16)) result = (x == 0)? Valid(y) : Invalid;
    input Maybe#(GCDArgs) in default = Invalid;

    rule gcd;
        if (isValid(in)) begin
            let args = fromMaybe(?, in);
            x <= args.a;
            y <= args.b;
        end else if (x != 0) begin
            if (x >= y) begin
                x <= x - y;
            end else begin
                x <= y;
                y <= x;
            end
        end
    endrule
endmodule

module GCDTest;
    GCD gcd;
    Reg#(Bit#(5)) cycle(0);
    rule test;
        if (cycle == 1)
            gcd.in = Valid(GCDArgs { a: 3, b: 9 });
        if (cycle == 3)
            gcd.in = Valid(GCDArgs { a: 12, b: 8 });
        cycle <= cycle + 1;
    endrule
endmodule

<table>
<thead>
<tr>
<th>cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>result</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 5. Arithmetic Pipelines (18 points)

You are an intern at G Company, the proud maker of the G circuit, depicted on the right. Things have been going well, until one day you find that the throughput of the G circuit is too low. Your largest customer wants to integrate the G circuit into their new device, but they need a throughput at least double what you can currently provide. Since the G circuit is currently only combinational, you know exactly what to do: Pipeline it!

(A) (4 points) For your first iteration, you decide to start small with a two-stage pipeline. Using the diagram below, please show a two-stage pipeline with maximal throughput using a minimum number of registers. Make sure to note the location of your registers with dots. In addition, calculate the overall throughput and latency of your circuit.

Latency (ns): __________
Throughput (ns⁻¹): __________
(B) (5 points) After doing the first iteration, you are starting to get pretty comfortable with the G circuit and decide to add an additional stage. Using the diagram below, please show a three-stage pipeline with maximal throughput using a minimum number of registers. Make sure to note the location of your registers with dots. In addition, calculate the overall throughput and latency of your circuit.

Latency (ns): __________

Throughput (ns⁻¹): __________
(C) (6 points) Having done two pipelines, you feel ready for the final challenge: the maximal-throughput pipeline. Using the diagram below, please show a pipeline that maximizes the throughput using a minimum number of registers. Make sure to note the location of your registers with dots. Additionally, calculate the latency and throughput of your circuit. For full credit you pipelined circuit should use the smallest number of pipeline stages required to achieve maximum throughput.

Latency (ns): __________

Throughput (ns⁻¹): __________
(D) (3 points) Your pipelined G circuit is a smashing success! Now, you have customers who wants to integrate another pipelined circuit, Circuit H, shown on the right. However, this circuit has multiple different timing specifications! In the below table, you have timing specifications for two different implementations of H: Version A, and Version B. Using your maximal-throughput G circuit from part C, select the version of H that minimizes your latency of the combined G-H circuit. Additionally, calculate the latency and throughput of the combined G-H circuit.

<table>
<thead>
<tr>
<th></th>
<th>Version A</th>
<th>Version B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Period</td>
<td>7ns</td>
<td>8ns</td>
</tr>
<tr>
<td>Stages</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Version (circle one):  Version A  Version B

Overall latency (ns): _________

Overall throughput (ns⁻¹): _________
Problem 6. RISC-V Processor (16 points)

Tomasulo is writing a RISC-V assembly program that is moving a lot of memory from one location to another. As a result, his code is full of snippets like:

\[
\begin{align*}
&\text{lw} \ x3, \ 0(x1) \\
&\text{sw} \ x3, \ 0(x2)
\end{align*}
\]

Instead of wasting a temporary register and an additional instruction, Tomasulo decides to extend the RISC-V ISA with a new instruction to execute in a single cycle:

\[
lwsw \ rs1, \ rs2
\]

The new instruction \textit{lwsw} computes the source and destination memory address using the contents of registers \textit{rs1} and \textit{rs2}, respectively, and copies the value at the source address directly to the destination address:

\[
\text{mem}[\text{reg}[\text{rs2}]] \leftarrow \text{mem}[\text{reg}[\text{rs1}]]
\]
(A) (3 points) Tomasulo hopes to be able to simply add some new signals as inputs to the selection muxes in the existing RISC-V processor diagram. Can this be done with the existing processor components as described in lecture and shown on the previous page? If not, describe all the changes necessary to support this instruction.

Possible? YES NO

If not, explain all required changes to support this instruction:

Tomasulo’s friend from Intel warns that supporting this feature will become a challenge later as the processor becomes more complex and advises him to abandon this idea. Tomasulo trusts this friend and decides to take their advice.

Tomasulo’s friend felt bad for crushing his dreams of a lws w instruction and decides to help optimize Quicksort from Lab 2. Tomasulo’s friend notices the following snippet of assembly is executed many times:

```
slli x2, x1, 2
add x2, x3, x2
lw x4, 0(x2)
```

In order to increase the efficiency of his code, Tomasulo’s friend suggests combining these three instructions into a single instruction that can be executed in one cycle. This is the new instruction Tomasulo comes up with to add to the RISC-V ISA:

```
lwae rd, (rs1, rs2)
```

The new lwae (load word array element) instruction loads a word from memory using the contents of register rs1 as a base of an array of 32-bit values and the contents of register rs2 as an index into that array.

```
reg[rd] <= mem[reg[rs1] + (reg[rs2] << 2)]
```
The encoding of the new `lwa`e instruction described above is as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>rs2</td>
<td>rs1</td>
<td>011</td>
<td>rd</td>
<td>000011</td>
</tr>
</tbody>
</table>

Note that `lwa`e does not have a value for immediate, because it will only shift the index by 2.

(B) (1 point) What is the 32-bit binary encoding of the following instruction? Provide your answer in hexadecimal notation.

`lwa`e x3, (x1, x2)

**Encoding in hexadecimal (0x):** ________________

In the diagram below, Tomasulo added an additional, dedicated `<<2` module which takes the second register value and shifts it to the left by two. Note that the output of this new `<<2` module is labeled `r2s2l2Res`. 
(C) (6 points) For each of the following signals, does the mux being controlled by that signal need an extra input to accommodate the new instruction? If so, indicate the **name of the signal that needs to be added as an input to the mux**. If not, indicate which existing value of the **mux control signal** is required to make the instruction work properly.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Needs new input?</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WERFSEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDSEL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New input/Existing control signal: 

(D) (6 points) Additionally, decide for each of the following control signals what their values should be when executing the `lwa` instruction. If the value of the signal doesn’t matter, then put N/A. The possible values for each signal are provided below.

**AluFunc:** Add, Sub, And, Or, Xor, Slt, Sltu, Sll, Srl, Sra

**BrFunc:** Eq, Neq, Lt, Ltu, Ge, Geu

**MWR:** Read, Write

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AluFunc</td>
<td>______________</td>
</tr>
<tr>
<td>BrFunc</td>
<td>______________</td>
</tr>
<tr>
<td>MWR</td>
<td>______________</td>
</tr>
</tbody>
</table>

END OF QUIZ 2!