Problem 1. Binary Arithmetic (10 points)

(A) (4 points) What is \(\neg(0x5A) \oplus 0x3D\), where \(\neg\) is bitwise NOT and \(\oplus\) is bitwise XOR? Provide your result in both binary and hexadecimal.

\[
\text{Result in binary (0b): } \quad \_ \_ \_ \_ \_ \_ \_ \_ \_ \\
\text{Result in hexadecimal (0x): } \quad \_ \_ \_ \_ \_ \_ \_ \_ \_ \\
\]

(B) (4 points) What is 15 in 8-bit 2’s complement notation? What is \(-22\) in 8-bit 2’s complement notation? Show how to compute 15–22 using 2’s complement addition. What is the result in 8-bit 2’s complement notation?

\[
15\text{ in 8-bit 2’s complement notation (0b): } \_ \_ \_ \_ \_ \_ \_ \_ \_ \\
-22\text{ in 8-bit 2’s complement notation (0b): } \_ \_ \_ \_ \_ \_ \_ \_ \_ \\
15-22\text{ in 8-bit 2’s complement notation (show your work) (0b): } \_ \_ \_ \_ \_ \_ \_ \_ \_ \\
\]

(C) (2 points) What range of numbers encoded using two’s complement representation can be expressed using 5 bits? Provide your answer in decimal.

\[
\text{Smallest 5-bit two’s complement number (in decimal): } \_ \_ \_ \_ \_ \_ \\
\text{Largest 5-bit two’s complement number (in decimal): } \_ \_ \_ \_ \_ \_ \\
\]
Problem 2. Boolean Expressions (12 points)

(A) (6 points) Simplify the following Boolean expressions by finding a minimal sum-of-products expression for each one. (Note: These expressions can be reduced into a minimal SOP by repeatedly applying the Boolean algebra properties we saw in lecture.)

1. \((a + b \cdot \overline{c}) \cdot d + c\)

2. \(a \cdot (b + c)(c + a)\)

(B) (6 points) There are Boolean expressions for which no assignment of values to variables can produce True (e.g., \(a \cdot \overline{a}\)). These Boolean expressions are said to be non-satisfiable.

Are the following Boolean expressions satisfiable? If the expression is satisfiable, give an assignment to variables that makes the expression evaluate to True. If the expression is non-satisfiable, explain why.

1. \((\overline{x} + yz) \cdot (\overline{y}x + z) \cdot (\overline{z}y + x)\)

2. \((\overline{x} + yz) \cdot (\overline{y}x + z) \cdot (\overline{z}y + x) + (\overline{x} + yz) \cdot (\overline{y}x + z) \cdot (\overline{z}y + x)\)
Problem 3. Static Discipline (16 points)

The Q-module shown to the right has two inputs carrying voltages $V_A$ and $V_B$ and a single output carrying $V_C$. The output $V_C$ is 1 volt if the sum of the input voltages, $V_A + V_B$, has been more than 4 volts for at least 10 ns; it will be 5 volts if $V_A + V_B$, has been less than 3 volts for at least 10 ns; and otherwise is at some undetermined voltage between 0 and 5 volts.

To summarize: after inputs have been stable for 10 ns,

$$V_C = \begin{cases} 
1 \text{ volt} & \text{if } V_A + V_B > 4 \text{ volts} \\
5 \text{ volts} & \text{if } V_A + V_B < 3 \text{ volts} \\
0 \leq ??? \leq 5 \text{ volts} & \text{otherwise}
\end{cases}$$

You may assume that no negative voltages are used in the circuits of this problem.

In this problem, you will explore the possibility of using the Q-module as the basis for a new family of logic devices. To this end, we need a convention for representing logic values (0 and 1) as voltages. This convention should yield acceptably large noise margins. In particular, it should maximize noise immunity, defined as the width of the smaller of the two noise margins.

(A) (2 points) Suppose constant voltages are applied to the $V_A$ and $V_B$ input terminals of a Q-module, and an output voltage of 5 volts is measured at $V_C$ after 20 ns. Assuming the Q-module obeys the above specification, what can you conclude about the sum $V_A + V_B$ of the input voltages? Choose the best answer.

- C1: $V_A + V_B < 3$ volts
- C2: $V_A + V_B \leq 4$ volts
- C3: None of the above.

Best conclusion about $V_A + V_B$ (circle one): C1 ... C2 ... C3
You begin by exploring configurations of the Q-module that will perform as an inverter. You consider two different inverter proposals:

(B) (8 points) Select the proposal that gives the best noise immunity, and specify parameters for an appropriate logic mapping and the resulting noise immunity.

Best proposal (1 or 2): _____

Values for $V_{OL}$: _____; $V_{IL}$: _____; $V_{IH}$: _____; $V_{OH}$: _____

Noise immunity: _____

Next, you consider logic mappings that allow using a single Q-module directly as a 2-input combinational device, as depicted to the right. Your goal is to find a set of logic mapping parameters for which the 2-input circuit at the right computes a useful logic function, and does so with acceptable noise margins.

(C) (3 points) Consider a logic convention for which a Q-module can serve as a logic device computing an interesting (non-constant) function of its inputs and that maximizes the noise immunity. Specify the resulting noise immunity.

Noise immunity: _____ volts

(D) (3 points) Identify the function computed by the single Q-module given the above convention, by specifying a Boolean expression for C in terms of inputs A and B.

Boolean expression for computed function: ________________________________
Problem 4. Combinational Logic (30 points)

(A) (5 points) The following BSV function $f$ performs a basic operation using $a$ and $b$. We want $f_2$ to implement the same function as $f$. Fill in the blank in $f_2$ to make the two functions equivalent. Use a single expression. Assume $n$ is a power of 2.

```
// TLog#(n) is the log base 2 of n
function Bit#(1) f(Bit#(n) a, Bit#(TLog#(n)) b);
    Bit#(n) x = a;
    for (Integer i = 0 ; i < valueOf(TLog#(n)) ; i = i+1) begin
        // (2**i) is 2 to the ith power
        x = (b[i] == 1) ? x >> (2**i) : x;
    end
    return x[0];
endfunction

function Bit#(1) f2(Bit#(n) a, Bit#(TLog#(n)) b);
    return ____________________;
endfunction
```
(B) (6 points) Write the truth table for the combinational device described by the function below.

```
function Bit#(2) f(Bit#(1) a, Bit#(1) b, Bit#(1) c);
    Bit#(8) x = 8'hB7; // hex value 0xB7
    Bit#(2) ret = 1;
    if (c == 1) begin
        x = x + 1;
    end
    case ({a, b})
        1: ret = x[1:0];
        2: ret = x[3:2];
        3: ret = x[7:6] ^ 2'b11;
    endcase
    return ret;
endfunction
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>ret[1]</th>
<th>ret[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(C) (5 points) The following BSV function \( g \) performs a specific arithmetic operation on \( n \)-bit operands \( a \) and \( b \). We want the function \( g_2 \) to implement \( g \) in a single line of code. Fill in the blank with an expression of the form “\( a \) operation \( b \)” to make \( g_2 \) equivalent to \( g \).

```bsv
function Bit#(n) g(Bit#(n) a, Bit#(n) b);
    Bit#(TAdd#(n,1)) c = 1; // note initial value
    Bit#(n) ret = 0;
    for (Integer i = 0 ; i < valueOf(n) ; i = i+1) begin
        ret[i] = a[i] ^ ~b[i] ^ c[i];
        c[i+1] = (a[i] & ~b[i]) | (a[i] & c[i]) | (~b[i] & c[i]);
    end
    return ret;
endfunction

function Bit#(n) g2(Bit#(n) a, Bit#(n) b);
    return __________;
endfunction
```

(D) (6 points) For \( n = 2 \), manually compile the BSV function \( g \) above into a combinational circuit. Please draw its logic diagram. You can use inverters, AND, OR, XOR, NAND, and NOR gates, as well as multiplexers. Please label the inputs and outputs bit by bit (\( a[0], a[1], b[0], b[1], ret[0], \) and \( ret[1] \)). You do not need to optimize or simplify the circuit.
(E) (8 points) Show that 1-bit 2-to-1 muxes can be used to implement any combinational circuit by implementing an inverter, an AND gate, and an OR gate using only 1-bit 2-to-1 muxes. You may tie inputs to 1 or 0 if necessary, and may use one or multiple muxes. Clearly label all inputs and outputs.

\[ Z = A \cdot S + B \cdot \overline{S} \]

Logic diagram of inverter implementation using 2-input mux:

Logic diagram of AND gate implementation using 2-input mux:

Logic diagram of OR gate implementation using 2-input mux:
Problem 5. Sequential Logic Timing (12 points)

(A) (4 points) Find the propagation delay ($t_{SD}$) and contamination delay ($t_{CD}$) of the circuit shown below, using the $t_{CD}$ and $t_{PD}$ information for the gate components shown in the table below.

<table>
<thead>
<tr>
<th>Gate</th>
<th>$t_{CD}$</th>
<th>$t_{PD}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>0.1ns</td>
<td>1.0ns</td>
</tr>
<tr>
<td>NAND2</td>
<td>0.2ns</td>
<td>1.5ns</td>
</tr>
<tr>
<td>NAND3</td>
<td>0.3ns</td>
<td>1.8ns</td>
</tr>
<tr>
<td>XOR2</td>
<td>0.6ns</td>
<td>2.5ns</td>
</tr>
</tbody>
</table>

$t_{CD} =$ ___________ns

$t_{PD} =$ ___________ns

(B) (8 points) The circuit above implements a full subtractor (FS), which, similar to a full adder, computes $X - Y - \text{Bin}$ in two’s complement. You combine two FS circuits as shown below to implement a two-bit self-decrementing counter.

The table below shows the timing specifications for the D registers and FS circuit (not the same implementation as above!). Find the maximum value for the D register’s hold time, $t_{HOLD}$, and the minimum clock cycle period, $t_{CLK}$, for which the circuit operates correctly.

<table>
<thead>
<tr>
<th>D reg</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{CD}$</td>
<td>0.2ns 0.1ns</td>
</tr>
<tr>
<td>$t_{PD}$</td>
<td>0.3ns 4.1ns</td>
</tr>
<tr>
<td>$t_{SETUP}$</td>
<td>1.2ns 1.2ns</td>
</tr>
<tr>
<td>$t_{HOLD}$</td>
<td>???</td>
</tr>
</tbody>
</table>

maximum value for $t_{HOLD}$: ________ns

minimum value for $t_{CLK}$: ________ns
Problem 6. Sequential Logic in BSV (20 points)

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.

```verilog
interface Seq;
    method Action start(Bit#(32) a);
    method Bit#(32) getI;
    method Bit#(32) getY;
    method Action clear;
endinterface

module mkSeq(Seq);
    Reg#(Bit#(32)) i <= mkReg(0);
    Reg#(Bit#(32)) y <= mkReg(0);
    Reg#(Bit#(32)) x <= mkReg(0);

    rule computeStep if (x > 1);
        i <= i+1;
        if (x > y) y <= x;

        if (x[0] == 1) x <= x + x + x + 1;
        else x <= x >> 1;
    endrule

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

    method Bit#(32) getI if (x == 1);
        return i;
    endmethod

    method Bit#(32) getY if (x == 1);
        return y;
    endmethod

    method Action clear if (x == 1);
        x <= 0;
    endmethod
endmodule
```

(A) (3 points) If at the beginning of `computeStep` the values in the registers are $i = 17$, $y = 25$, and $x = 11$, what are the values after rule `computeStep` runs once?

1. $y =$
2. $x =$
(B) (5 points) If `start` is called with value `a = 10`, what will the output of `getI()` and `getY()` be when it is ready?

1. Return value of `getI()` =

2. Return value of `getY()` =

(C) (5 points) We have verified that for all possible 32-bit values of `a` except 0, when the module is started with `start(a)`, the `computeStep` rule eventually writes 1 to `x` and results become ready.

Assume that `a` is a nonzero value. Let’s denote the sequence of `k` values that `x` goes through as `[x_0, x_1, ..., x_{k-1}]`. Note that `x_0 = a` (the initial value written by `start`), and `x_{k-1} = 1` (the final value written by `computeStep` after some number of cycles).

Write expressions valid for all `a > 0` for the return values of `getI()` and `getY()` as a function of `k` and `[x_0, x_1, ..., x_{k-1}]`.

1. Expression for `getI()` =

2. Expression for `getY()` =

(D) (3 points) Suppose we modify the `computeStep` rule as shown below. Does this change the functionality of the module?

<table>
<thead>
<tr>
<th>Original code</th>
<th>Modified code</th>
</tr>
</thead>
</table>
| `rule` computeStep if (x > 1);
  i <= i+1;
  if (x > y) y <= x;
  if (x[0]==1) x <= x + x + x + 1;
  else x <= x >> 1;
endrule                                             | `rule` computeStep if (x > 1);
  i <= i+1;
  if (x[0]==1) x <= x + x + x + 1;
  else x <= x >> 1;
endrule                                             |

Does this change the functionality of the module? (circle one) Yes … No … Can’t tell
(E) (4 points) Draw the wires for the sequential-machine implementation of mkSeq. The Seq interface is shown again below for your convenience. Include and name all input and output wires: arguments, results, enable signals, and ready signals.

```markdown
interface Seq;
  method Action start(Bit#(32) a);
  method Bit#(32) getI;
  method Bit#(32) getY;
  method Action clear;
endinterface
```

END OF PRACTICE QUIZ 1!