Problem 1. Binary Arithmetic (10 points)

(A) (4 points) What is 0x79 << ~(0xFC) where ~ is bitwise NOT and << is bitwise LEFT SHIFT on 8-bit operands? Provide your result in both 8-bit binary and in hexadecimal.

\[
\text{Result in binary (0b)}: \quad 1100\ 1000 \\
\text{Result in hexadecimal (0x)}: \quad \text{C8}
\]

(B) (4 points) What is 0b1011 multiplied by 0b0101? You can treat both numbers as unsigned, and the result will be more than four bits. Show how you computed this with binary multiplication. What is this number in decimal?

\[
\text{0b1011 multiplied by 0b0101 (show your work) (0b)}: \quad 110111 \\
\text{in decimal}: \quad 55
\]

(C) (2 points) What is 0b10110101 in decimal assuming unsigned encoding (only positive integers)? What is it in decimal assuming two’s complement encoding?

\[
\text{0b10110101 assuming unsigned encoding (in decimal)}: \quad 181 \\
\text{0b10110101 assuming two’s complement encoding (in decimal)}: \quad -75
\]
Problem 2. Boolean Logic (12 points)

(A) (6 points) Simplify the following Boolean expressions by finding a minimal sum-of-products (minimal SOP) 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. \((r + a) \cdot y \cdot ((r + \bar{a}) \cdot \bar{y})\)
   \[= \bar{r} \cdot \bar{a} \cdot y \cdot ((r + \bar{a}) + y)\]
   \[= \bar{r} \cdot \bar{a} \cdot y\]

2. \((x + y) \cdot z + \bar{z} \cdot x + y \cdot (\bar{x} + z)\)
   \[= x \cdot z + y \cdot z + \bar{z} \cdot x + \bar{x} \cdot y\]
   \[= x + y \cdot z + \bar{x} \cdot y\]
   \[= x + y\]

(B) (6 points) Consider the truth table on the right, which defines function Y out of three input variables (A, B, and C).

1. Find a minimal sum-of-products expression for Y.

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

Minimal sum-of-products for \(Y(A,B,C) = \) \[a \cdot \bar{b} + \bar{b} \cdot c\] 

2. Using only combinational circuits built from gates specified by the truth table above, which of the following statements are true?

   (A) We can implement any Boolean function using only these gates.
   (B) We can’t implement every Boolean function using only these gates.
   (C) We can only implement functions with the same truth table as above.

Circle all true answers: B … C
Problem 3. The Digital Abstraction (14 points)

Rummaging through a box of old circuit components, you find a Q-device with the voltage transfer characteristic (VTC) shown to the right.

(A) (4 points) We want to use the Q-device as a binary digital inverter. Find a set of signaling thresholds ($V_{OL} < V_{IL} \leq V_{IH} < V_{OH}$) that maximizes noise immunity (the smallest of the noise margins).

\[
\begin{align*}
V_{OL} &= \underline{0} V \\
V_{IL} &= \underline{1} V \\
V_{IH} &= \underline{5} V \\
V_{OH} &= \underline{6} V
\end{align*}
\]

Noise Immunity = \underline{1} V

(B) (4 points) Find a different set of signaling thresholds from the one above that also maximizes noise immunity.

\[
\begin{align*}
V_{OL} &= \underline{2} V \\
V_{IL} &= \underline{3} V \\
V_{IH} &= \underline{3} V \\
V_{OH} &= \underline{4} V
\end{align*}
\]

Noise Immunity = \underline{1} V

Reading the Q-device’s datasheet, you discover that it was meant to be used as a quaternary digital device. In quaternary logic, each voltage represents one of four possible symbols, 0, 1, 2, or 3. (By contrast, in binary logic each voltage represents one of two possible symbols, 0 or 1.)

The Q-device uses the following mapping of quaternary digital symbols to output voltages:

<table>
<thead>
<tr>
<th>Quaternary Symbol</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{out}$</td>
<td>0 V</td>
<td>2 V</td>
<td>4 V</td>
<td>6 V</td>
</tr>
</tbody>
</table>

(C) (2 points) Fill in the Q-device’s truth table below.

<table>
<thead>
<tr>
<th>Input Symbol</th>
<th>Output Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
(D) (4 points) Specify the range of valid input voltages for each quaternary symbol that maximize noise immunity for the Q-device. What is its noise immunity?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Valid input voltages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$V_{in} \leq 1, V$</td>
</tr>
<tr>
<td>1</td>
<td>$1, V &lt; V_{in} \leq 3, V$</td>
</tr>
<tr>
<td>2</td>
<td>$3, V &lt; V_{in} \leq 5, V$</td>
</tr>
<tr>
<td>3</td>
<td>$V_{in} &gt; 5, V$</td>
</tr>
</tbody>
</table>

Noise Immunity = __1\, V__
Problem 4. Combinational Logic (28 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. Write a single-line expression that uses the ternary operator (? :).

\[
\text{function Bit#(n) } f(\text{Bit#(n)} a, \text{Bit#(1)} b); \\
\text{Bit#(n) } x = 0; \\
\text{for (Integer } i = 0 ; i < \text{valueOf}(n) ; i = i+1) \text{ begin} \\
\quad x[i] = a[i] ^ b; \\
\text{end} \\
\text{return } x; \\
\text{endfunction}
\]

\[
\text{function Bit#(n) } f_2(\text{Bit#(n)} a, \text{Bit#(1)} b); \\
\quad \text{return } (\quad b==1\quad) ? \quad \neg a \quad : \quad a \quad ; \\
\text{endfunction}
\]

(B) (5 points) Write the truth table for the combinational device described by the function below.

\[
\text{function Bit#(2) } f(\text{Bit#(1)} a, \text{Bit#(1)} b, \text{Bit#(1)} c); \\
\text{Bit#(2) } ret = \text{zeroExtend}(a) + \text{signExtend}(b); \\
\text{case } \{a,b\} \\
\quad 0: ret = \{1, c\}; \\
\quad 2: ret = \{a ^ b, a & b\}; \\
\quad 3: ret = \neg \text{signExtend}(c); \\
\text{endcase} \\
\text{return } ret; \\
\text{endfunction}
\]

<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>1</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>1</td>
<td>1</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>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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 \( g2 \) to implement \( g \) in a single line of code. Fill in the blank with a single expression to make \( g2 \) equivalent to \( g \).

```haskell
function Bit#(1) g(Bit#(n) a, Bit#(n) b);
Bit#(2) ret = 'b10;
for (Integer i = valueOf(n)-1 ; i >= 0 ; i = i-1) begin
  if ({a[i], b[i]} == 'b01) ret = {0, ret[1] | ret[0]};
  else if ({a[i], b[i]} == 'b10) ret = {0, ret[0]};
end
return ret[1] | ret[0];
endfunction

function Bit#(1) g2(Bit#(n) a, Bit#(n) b);
return ______________;  // Fill in the blank
endfunction
```

(D) (5 points) Finish the following circuit diagram to implement function `computeB`, given below. You may only use 32-bit 2-to-1 multiplexers, constants (0, 1, 2, 3, ...) and logic gates (AND, NOT, OR, XOR). We have provided three 32-bit greater-than-or-equal (\( \geq \)) comparators for you.

```haskell
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
```
(E) (8 points) Show that one-bit comparators can be used to implement any combinational circuit by implementing an inverter, an AND gate, and an OR gate using only comparator gates. You may tie inputs to 1 or 0 if necessary, and may use one or multiple comparator gates. Clearly label all inputs and outputs.

\[
\text{EQ} = \overline{A \cdot B} \quad (A \text{ equals } B)
\]

\[
\text{LT} = \overline{A} \cdot B \quad (A \text{ less than } B)
\]

Logic diagram of inverter implementation using one-bit comparators:

Logic diagram of AND gate implementation using one-bit comparators:

Logic diagram of OR gate implementation using one-bit comparators:

(other answers are possible)
Problem 5. Sequential logic timing (12 points)

You are given the sequential circuit shown below together with the timing parameters for each of the circuit elements in the table next to it.

As is our convention for clocked devices, all timing specs for registers (\( t_{CD} \), \( t_{PD} \), \( t_{SETUP} \), and \( t_{HOLD} \)) are measured relative to the rising edge of the clock.

(A) (4 points) Using the timing specifications in the table above, determine the minimum clock period required for this sequential circuit to work properly.

\[
t_{CLK} \geq t_{PD,DREG} + t_{PD,INV} + t_{PD,XOR} + t_{SETUP,DREG}
\]

Minimum clock period for correct operation: \( 880 \) ps

(B) (3 points) What is the propagation delay \( t_{PD} \) for the output, OUT, of this sequential circuit relative to the rising edge of the clock?

\[
t_{PD} = t_{PD,DREG}
\]

\( t_{PD} \): \( 300 \) ps

(C) (5 points) What is the maximum value of \( t_{HOLD} \) for the D registers to ensure that this circuit will work correctly? *You may assume that \( in = 0 \) for this question.*

\[
t_{HOLD} \leq t_{CD,DREG} + t_{CD,AND}
\]

Maximum value for \( t_{HOLD} \) for correct operation: \( 130 \) ps
Problem 6. Sequential Logic in BSV (24 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.

```hs
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 <- mkReg(0);
    Reg#(Bit#(1)) validx <- mkReg(0);
    Reg#(Bit#(32)) x <- mkRegU();
    Reg#(Bit#(32)) i <- mkRegU();

    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
```

6.004 Fall 2018 - 9 of 11 - Quiz #1
(A) (8 points) The module using mkFoo is invoking all the methods of mkFoo at every clock cycle. Remember that an invoked method can only execute when it is ready. The register values at the beginning of t0 is given. Fill the register values at the beginning of the following four clock cycles in the table below.

<table>
<thead>
<tr>
<th>time value</th>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>17</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>validx</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>27</td>
<td>17</td>
<td>17</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>i</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

(B) (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

(C) (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 <= a - b;
  x <= a;
  validx <= 1;
  i <= i + 1;
endrule

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

Does this change the output sequence of getX() of the module?

(circle one) Yes ... No ... Can’t tell
(D) (2 points) Ignoring the changes in (C), suppose we modify the guard of `start` in the original code to \((a==0 \&\& validx == 0)\). Does this change the output sequence of `getX()`?

(circle one)   Yes  …  No  …  Can’t tell

(E) (8 points) A partial circuit diagram for the implementation of `mkFoo` is given below.

Give the Boolean expressions for each write enable signal of registers `a`, `validx`, `x`, and `i`. You may use only wire names from the picture (e.g., `t1`, `t2`, ..., `t9`, `start.en`, `getX.en`, `a`, `validx`, `x`, `i`) and at most a total of three AND(\(\cdot\)), OR(\(+\)), or NOT(\(\sim\)) gates across all expressions.

\[
\begin{align*}
    a.en & : \quad \text{start.en} + t5 \\
    validx.en & : \quad \text{getX.en} + t5 \\
    x.en & : \quad t5 \\
    i.en & : \quad \text{start.en} + t5
\end{align*}
\]

END OF QUIZ 1!