# Introduction to Hardware Design
# Basic Digital Logic: Solutions

Profs. Sundeep Rangan, Siddharth Garg

1. *Sequential updates*: Consider the following SystemVerilog code:

```
always_ff @(posedge clk) begin
    x <= x + v;
    if (x > 30) begin
        v <= -10;
    end else if (x < 0) begin
        v <= 10;
    end
end
```

   Starting from $(x, v) = (15, 10)$, what are the values of $(x, v)$ for the next 5 clock cycles?

   **Solution:**

   Fill solution here.

2. *Linear system*: Consider the following SystemVerilog code:

```
always_ff @(posedge clk) begin
    x1 <= ((5*x1 + 3*x2)>>3);
    x2 <= ((4*x1 - 4*x2)>>3);
end
```

   where `>>3` denotes a right shift by 3 bits (division by 8), and all values are 32-bit signed integers.

   (a) Suppose that $(x_1, x_2) = (8, 0)$, what is the value of $(x_1, x_2)$ in the next clock cycle? Remember that assignments (`<=`) are non-blocking, so they update in parallel.

   (b) Write a python function that simulates this linear system with initial conditions `x1_0` and `x2_0` for `nit` clock cycles and outputs the values in a array `X` of shape `(nit+1,2)`. The first row of `X` should be the initial conditions.

   **Solution:**

   Fill solution here.

3. *ReLU function*: We wish to implement the function:

$$y = ax^2 + \max\{bx, 0\} + c,$$

   for an input $x$ and constants $a$, $b$, and $c$.

   Write the SystemVerilog code to implement this function over two clock cycles. Specifically, the input $x$ should be registered in the first clock cycle, and the output $y$ should be produced in the second clock cycle. Make sure that no clock cycle requires two or more multiplications that cannot be parallelized.

**Solution:**

Fill solution here.

4. *ReLU+quadratic function*: Consider the following SystemVerilog code:

```
always_comb begin
    act_in = w1*xreg+b1;
    if (act_in > 0) begin
        a = act_in;
    end else begin
        a = 0;
    end
    xsq = ((xreg*xreg)>>2);
    y = xsq + w2*a + b2;
end
always_ff @(posedge clk) begin
    xreg <= x;
end
```

where `>>2` denotes a right shift by 2 bits (division by 4). So, the code registers the input `x` into `xreg` on each clock cycle, and produces the output `y` in a single clock cycle based on the registered value. Assume that $w_1$, $b_1$, $w_2$, and $b_2$ are constants. Rewrite the code to operate over two clock cycles. Specifically, register the input on $x$ on the first clock cycle, and compute the output $y$ on the two clock cycles later. This requires introducing additional registers to store intermediate values.

**Solution:**

Fill solution here.

5. *Bouncing ball*: We simulate a ball moving in one-dimensional space between two walls at positions 0 and 100. The ball has a position $x$ and a velocity $v$. At each time step, the ball first moves according to its velocity:

$$x \leftarrow x + v.$$

If this motion causes the ball to go past a wall, the ball "bounces" and reverses direction. The bounce should behave the same way a real ball would: the ball cannot pass through the wall, and the rebound distance should be consistent with how far it would have travelled past the wall.

For example:

- If $(x, v) = (40, 10)$, then the ball moves to 50, which is inside the interval, so the next state is $(50, 10)$.
- If $(x, v) = (96, 10)$, then the ball would move to 106, which is past the right wall at 100. After bouncing, the ball ends up at position 94 with velocity $-10$.
- If $(x, v) = (3, -10)$, then the ball would move to $-7$, which is past the left wall at 0. After bouncing, the ball ends up at position 7 with velocity 10.

Write the SystemVerilog code for the updates for $x$ and $v$. You do not need to include the module declaration, just the `always_ff` and `always_comb` blocks.

**Solution:**

Fill solution here.

6. *Exponent* Suppose we wish to implement the function

$$y = x^i,$$

with an integer exponent $i \in \{0, 1, 2, 3\}$. The input $x$ and output $y$ are signed short integers – do not worry about overflow. Write a SystemVerilog module that computes $y$. The output should always be 2 cycles after the input, even if $i = 0, 1$ or 2. Use only one multiplication in each clock cycle.

Hint: You will need to use delay lines to store the input $x$ and exponent $i$. This problem is a bit hard since it uses pipelining. We will discuss pipelining in more detail later units.

**Solution:**

Fill solution here.

7. *Testbench code* A System Verilog Testbench instantiates a module to compute some function $y = f(x)$:

```
// Testbench signals
logic signed [31:0] x;
logic signed [31:0] y;
logic clk;

// Instantiate the device under test (dut)
function dut (
    .clk(clk),
    .x(x),
    .y(y)
);

// Test vectors
logic signed [31:0] xtest [0:3] = '{10, 20, -5, 7};
```

Write an `initial` block that:

- On cycle 0, sets `rst=1`, and then de-asserts it on the next cycle.
- Starting on cycle 4, it sets `x=xtest[0]`, prints the output `y` from the DUT on cycle 8.
- On cycle 9, it sets sets `x=xtest[1]`, prints the output `y` from the DUT on cycle 12.
- Continue as above for the remaining test inputs.

To display a number you can use the syntax:

```
$display("x = %0d", x)
```

**Solution:**

Fill solution here.