

Exercise 1 – Parameterizing the Guessing Game

- a) Parameterize the comparator module created in Section 2 for bit-width N instead of just 3 bits.

```
// Compares A to B, inputs of width N
// and outputs lets us know if it's greater than (is_gt),
// less than (is_lt), or equal to (is_eq)
module comparator
    (input  logic [2:0] A, B,
     output logic is_lt, is_gt, is_eq);
    // subtraction result (intermediate)
    logic [2:0] sub;
    assign sub = A - B;

    assign is_eq = ~(sub[0] | sub[1] | sub[2]);
    assign is_lt = sub[2];
    assign is_gt = ~is_eq & ~is_lt;
endmodule // comparator
```

- b) Parameterize the Guessing Game module for bit-width N and secret number S :

```
// Game to check user's N-bit input guess against a hard-coded
// secret #, S
// - SW[2:0] is the guess, KEY[0] allows input to be checked
// - LEDR[0] is <, LEDR[1] is ==, LEDR[2] is >
module guessing_game
    (output logic [9:0] LEDR,
     input  logic [3:0] KEY, input  logic [9:0] SW);

    logic is_lt, is_eq, is_gt;

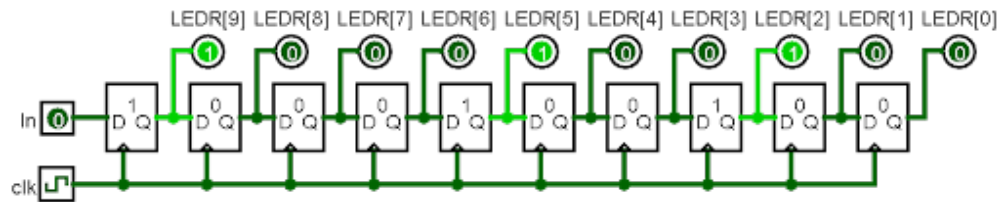
    comparator    number_comparator(
        .A(SW[2:0]),
        .B(3'b001),
        .is_lt, .is_eq, .is_gt
    );

    assign LEDR[0] = is_lt & ~KEY[0];
    assign LEDR[1] = is_eq & ~KEY[0];
    assign LEDR[2] = is_gt & ~KEY[0];

endmodule // guessing_game
```

Exercise 2 – Creating a String of Lights

Finish the module called `string_lights` that implements the system shown below (a string of 10 flip-flops/1-bit registers tied to the LEDRs) for the DE1-SoC.



- Use `SW[9]` as the reset, `SW[0]` as `In`, and `KEY[0]` as `clk`.
 - Since we are using a `KEY` for the clock, no need for `clock_divider`.
- Hint: flip-flops can be module instances or inferred from an `always_ff` block.

```
// flip-flop that samples d on the rising edge of the clk
// and transfers it to q
module D_FF1 (output logic q,
  input logic d, reset, clk);
  always_ff @(posedge clk)
    if (reset)
      q <= 0;
    else
      q <= d;
endmodule // D_FF1

module string_lights (output logic [9:0] LEDR,
  input logic [3:0] KEY,
  input logic [9:0] SW);
  logic clk, reset, in;
  // TODO: Finish creating the string of lights with or without the D_FF1 module!
```

```
endmodule // string_lights
```

Exercise 3 – Sequential Logic Test Bench

Finish the test bench for string_lights.

```
module string_lights_tb();
    logic [9:0] LEDR;
    logic [3:0] KEY;
    logic [9:0] SW;

    string_lights dut (.*);

    // TODO: Finish the test bench!
    // Remember that KEY[0] is acting as a clock for this system.

endmodule // string_lights_tb
```