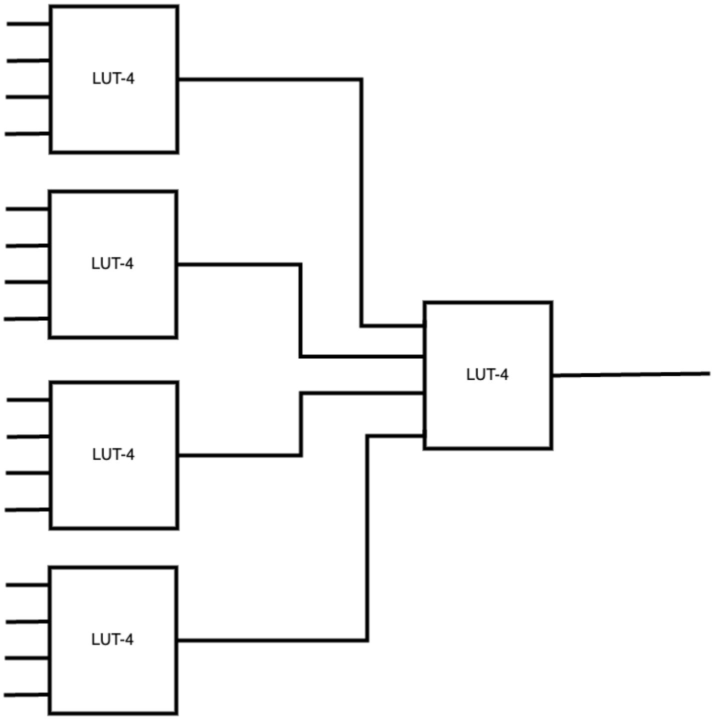
EECS 151/251A Discussion 3

Feburary 2, 2024

Problem 1: LUTs and Functions

Lookup Tables (LUTs) are the fundamental building blocks of FPGA architectures. A LUT-*N* can implement **any** *N* input logic function (ex. a LUT-4 can implement any logic function with four logical inputs). On an FPGA, LUTs can be connected together through special routing to implement functions with even more inputs. Consider the following arrangement of five LUT-4 blocks:



How many logic functions can this chain of LUT-4's implement?

Solution:

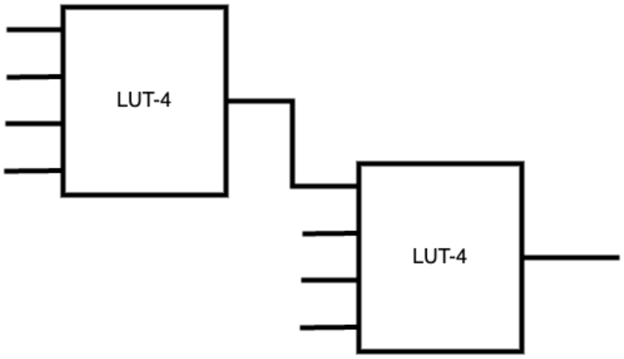
Let's break it down. **A function is a logical expression.** How many functions can single bit implement? 4. Bit A can be either 0 or 1, and can map to 0 or 1.

1. $A \rightarrow 0$
2. $A \rightarrow 1$
3. $\overline{A} \rightarrow 0$
4. $\overline{A} \rightarrow 1$

From this we can establish a general rule: 2^{2^n} for a LUT-N. Therefore, each LUT-4 can implement $2^{2^4} = 2^{16} = 65536$ functions. Since the inputs to the last LUT-4 is another LUT-4, each bit represents 2^{2^4} functions. Therefore, the total number of functions is $(2^{16})^5$.

Challenge yourself.

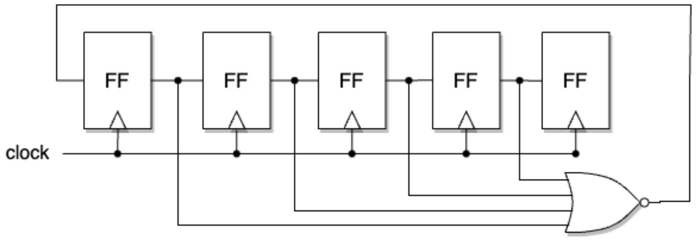
1. How many functions can be produced if the last LUT-4 is removed and the outputs are concatenate? $(2^{16})^4$
2. How many functions can be produced for the following circuit:



The answer is $\mathbf{2^{2^{32}}}$.

Problem 2: Self Starting Ring Counter

A ring counter is a special counter composed of flip-flops daisy-chained together to form a shift register and the output of the last flip-flop is connected to the output of the first. Below is a variant of self starting ring counter. (**Note:** the last flip-flops output is not the input to the first, but it's close enough to call it a ring counter :-)). It is self starting because there is no reset. The counter will reset itself! The counter is read out such that the first register is the LSb of the count value.



1. How does this self-initialize itself?
2. What type of counter is this?
3. Assume the register are initialized as 0, 1, 0, 1, 0. Create a table showing clock cycle, input to the chain, value of each register. Provide a waveform diagram for the first 10 cycles after initialization.
4. How does the circuit behave in steady state (steady state means after hundreds of cycle)?
5. This behavior can be create using a regular incrementing counter and a decoder. Write Verilog for this implementation.

Solution:

The NOR gate takes as input all *except* the last flip-flop in the shift register. Therefore, the NOR output is `1` only if all of the first four registers are zero. Since, the NOR output is the input of the shift register, wherever the previous condition occurs the shift register is initialize with a `1`. Note this occurs regardless of the value in the last flip-flop.

1. The input to the shift register is `1` when the first four flip-flops are zero.

2. A one-hot counter

3.

Cycle	NOR Output	Reg0	Reg1	Reg2	Reg3	Reg4
0	0	1	1	0	1	0
1	0	0	1	1	0	1
2	0	0	0	1	1	0
3	0	0	0	0	1	1
4	1	0	0	0	0	1
5	0	1	0	0	0	0
6	0	0	1	0	0	0
7	0	0	0	1	0	0
8	0	0	0	0	1	0
9	1	0	0	0	0	1
10	0	1	0	0	0	0

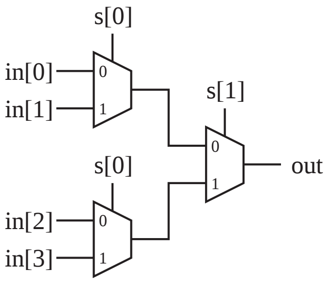
4. It counts in powers of two: 1, 2, 4, 8, 16, 1, 2, 4, ...

5.
```verilog
module ring_cnt(
  input clk,
  output reg [4:0] cnt);

  integer j;  // Used for bit reversal

  wire [4:0] reg_in;  // Input to register
  reg [5:0] reg_out;  // Output of register


  // Create shift register
  genvar i;
  generate
    for (i=0; i<5; i=i+1) begin
      REGISTER regX ( .clk(clk),
                      .d([reg_ini]),
                      .q(reg_out[i]));
    end
  endgenerate

  // Procedural Assignment
  always @(*) begin
    for(j=0;j<5;j=j+1) begin
      cnt[j] = reg_out[4-j];
    end
  end


  // Signal Assignment
  assign reg_in = {reg_out[3:0], ~|reg_out[3:0]};

endmodule
```

Problem 3: Decoder-Based Multiplexer

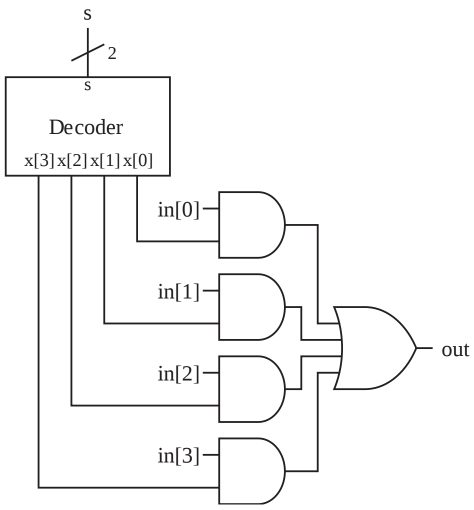
(a) Design a 4-to-1 multiplexer using one of the decoders you designed above. The select signals must be input to the decoder and must not be used anywhere else. Provide an exhaustive test.

(b) What could be a potential benefit of using this decoder-based multiplexer against the following design:



Solution:

(a) Diagram:



Design:

```
`include "decoder.v"

module multiplexer(
                        input  [1:0]  s,
                        input  [3:0]  in,
                        output        out
                        );

    wire [3:0]                   x;
    wire                         t0, t1, t2, t3;

    decoder1 dc1(.s(s), .x(x));

    and(t0, in[0], x[0]);
    and(t1, in[1], x[1]);
    and(t2, in[2], x[2]);
    and(t3, in[3], x[3]);
    or(out, t0, t1, t2, t3);

endmodule
```

Testbench:

```
module multiplexer_tb;
    reg [1:0] s;
    reg [3:0] in;
    reg       expected;
    wire      out;

    // loop variables
    integer   i, j;

    // instantiate duts
    multiplexer mux1(.s(s), .in(in), .out(out));

    // expected outputs
    always @(*) begin
        case (s)
          2'b00:   expected = in[0];
          2'b01:   expected = in[1];
          2'b10:   expected = in[2];
          2'b11:   expected = in[3];
          default: expected = 1'bx;
        endcase
    end

    // begin test
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars;
        for(j = 0; j < 16; j = j + 1) begin
            in = j;
            for(i = 0; i < 4; i = i + 1) begin
                s = i;
                #1;
                $display("s:␣%b,␣in:␣%b,␣out:␣%b,␣expected:␣%b",
                          s, in, out, expected);
                // Break early if failed
                if(out !== expected) begin
                    $display("FAILED,␣expected␣%b,␣got␣%b",
                              expected, out);
                    $finish();
                end
            end
        end
        $display("ALL␣TESTS␣PASSED!");
        $finish();
    end
```

endmodule

(b) Smaller delay (fewer logic levels) from data input to output, trading off delay from select signals to output.