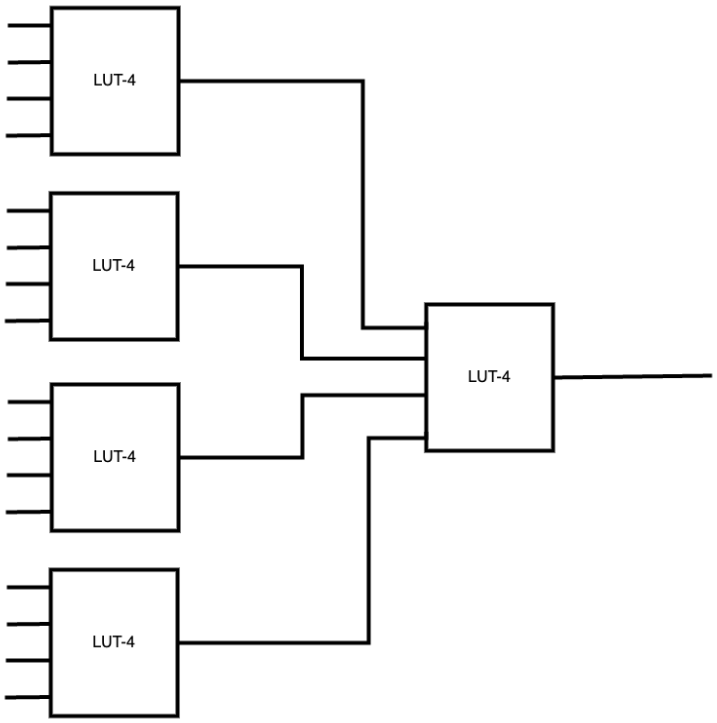
EECS 151/251A Discussion 2

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:

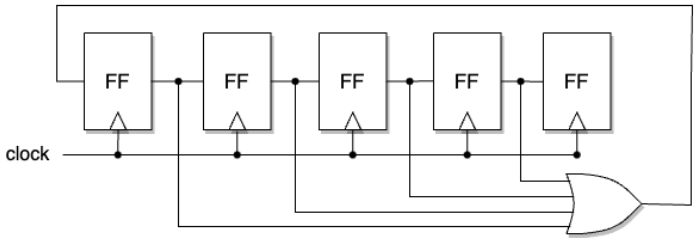
Let's break it down. **A function is a logical expression.** How many function can single bit implement? 2 (either 1 or 0). How many functions can two bits implement?

1. A <*func*> B
2. A <*func*> B
3. A <*func*> B
4. A <*func*> B

Four! Therefore, we are really asking the number of permutations. Each bit has 2 states, therefore a LUT-4 can implement $2 * 2 * 2 * 2 = 16$ functions. If each input of the last LUT-4 is a LUT-4 output, then the last LUT-4 can implement $16 * 16 * 16 * 16 = 2^{4^4} = 2^{16} = 65536$ functions

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.
```
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
```

Four! Therefore, we are really asking the number of permutations. Each bit has 2 states, therefore a LUT-4 can implement $2*2*2*2 = 16$ functions. If each input of the last LUT-4 is a LUT-4 output, then the last LUT-4 can implement $16*16*16*16 = 2^{4^4} = 2^{16} = 65536$ functions

Problem 3: Bit-Stream Reverse Engineering

Imagine you obtained a bit-stream from somewhere and want to figure out what it is.

(a) The bit-stream is `0x1777A5A5965A9696`. This bitstream is fed in from right to left (i.e. `6` is fed first and `1` is fed last).

(b) Every LUT in the FPGA has N inputs (the value of N is part of the mystery). The LUTs are numbered 0, 1, 2, ...

(c) Each LUT has an output labeled y_i and inputs labeled x_{i_j}, where i is the LUT number and j is the input number.

(d) For programming, the LUTs are connected in a shift register. The bit-stream will be shifted in from LUT0 and then pushed to the subsequent LUTs. (This implies that the left most `1` must be part of the function for LUT0.)

(e) The shift register is ordered in the ascending order of input for each LUT. That is, the first register stores the output for an input 00...0, the second register stores the output for an input 00...01 (only x_{i_0} is 1), and so on. (This implies that if $N = 2$, LUT0 programmed with `1` works as an AND gate.)

(f) One of the LUTs is programmed as a 2-input gate. Furthermore, it is the only LUT programmed as a 2-input gate.

Answer the following questions:

1. How many LUTs would the above bit stream program?
2. Write down the function of each LUT in Boolean expression (no need to simplify but notice there are some patterns that can be concisely expressed with XORs).

Solution:

1. Four 4-LUTs. Based on the condition that there is only one LUT implementing a 2-input gate, $N = 2$ is invalid because both LUT1 and LUT2 implement 2-input OR gates (`7`), and $N = 3$ is invalid because LUT1 implements a 2-input OR gate (`77`) while LUT2 implements a 2-input XNOR gate (`A5`). For $N = 5$ and $N = 6$ (6 is the maximum because of the length of bit-stream), there are no LUTs implementing 2-input gates. $N = 4$ is the only choice where only LUT1 implements a 2-input gate.

2.

$$
\begin{aligned}
y_0 &= (x_{0\_0} + x_{0\_1}) \cdot (x_{0\_2} + x_{0\_3}) + x_{0\_0} \cdot x_{0\_1} \\
y_1 &= \overline{x_{1\_0} \oplus x_{1\_2}} \\
y_2 &= \overline{x_{2\_0} \oplus x_{2\_2} \oplus (x_{2\_1} + x_{2\_3})} \\
y_3 &= \overline{x_{3\_0} \oplus x_{3\_1} \oplus x_{3\_2}}
\end{aligned}
$$