Problem 1: FPGAs

1. FPGA Logic Block

Consider an n-input LUT:

(a) How many unique logic functions can be implemented?
(b) As a function of n, how does the chip area consumed by the LUT scale with n?
(c) As a function of n, how does the delay through the LUT scale with n?

2. FPGA Mapping

In the logic circuit below, for each type of LUT, what is the minimum number of LUTs needed for implementation? Each implementation can use only one type of LUT.

3-LUT: 4-LUT:
Problem 2: Winnie the Pooh’s Emotional States

In this problem, you will write some SystemVerilog assertions to make sure the following Winnie the Pooh FSM is behaving correctly. If you need a refresher on SystemVerilog assertions, it may be helpful to refer to this guide.

Winnie the Pooh has three emotional states (in order of increasing happiness): SAD, NEUTRAL, HAPPY. He experiences at most one life_event per clock cycle: finding honey, losing a balloon, or it’s a rainy day. With an infinite number of balloons available, Winnie the Pooh can always lose a balloon without limitations if the event occurs.

When Winnie the Pooh finds honey, he will become one state happier, but if he is already in the HAPPY state, he will remain HAPPY. When he loses a balloon or experiences a rainy day, he will become one state less happy, but if he is already in the SAD state, he will remain SAD.

Because practicing gratitude is important, Winnie the Pooh wants to keep track of the number of happy events in his life. When he finds honey, happy_events increases by 1. Otherwise, happy_events remains constant.

In this question, please refer to the value of current_state as SAD, NEUTRAL, or HAPPY.
module pooh_fsm (
    input clk,
    input reset,
    input [2:0] life_event,
    output [1:0] pooh_state,
    output [8:0] happy_events
);

localparam SAD = 2'b00;
localparam NEUTRAL = 2'b01;
localparam HAPPY = 2'b10;

wire found_honey, lost_balloon, rainy_day;
assign found_honey = life_event[0];
assign lost_balloon = life_event[1];
assign rainy_day = life_event[2];

reg [1:0] current_state, next_state;
assign pooh_state = current_state;
reg [8:0] happy_events_reg;
assign happy_events = happy_events_reg;

always @(posedge clk) begin
    if (reset) begin
        current_state <= NEUTRAL;
        happy_events_reg <= 0;
    end else begin
        current_state <= next_state;
        if (found_honey) begin
            happy_events_reg <= happy_events_reg + 1;
        end
    end
end

always @(*) begin
    // next_state transition logic
    case (current_state)
        SAD: begin
            next_state = found_honey ? NEUTRAL : SAD;
        end
        NEUTRAL: begin
            next_state = found_honey ? HAPPY : (lost_balloon || rainy_day) ? SAD : NEUTRAL;
        end
        HAPPY: begin
            next_state = (lost_balloon || rainy_day) ? NEUTRAL : HAPPY;
        end
    endcase
end

    // SVA Assertions
endmodule
1. Complete the following assertion to check we have expected behavior when reset is high.
   Please refer to the value of `current_state` as SAD, NEUTRAL, or HAPPY.

   ```vhdl
   property correct_reset;
   @(posedge clk) ___(i)___ |-> #1 current_state == ___(ii)___ && happy_events_reg == 0;
   endproperty
   assert property (correct_reset);

   (i): 
   (ii): 
   ```

2. Complete the following assertion to make sure `happy_events_reg` increments correctly when
   Winnie the Pooh finds honey.

   ```vhdl
   property happy_events_increments;
   @(____(i)____) disable iff (reset)
   found_honey |-> #1 happy_events_reg == $past(____(ii)____) + 1;
   endproperty
   assert property (happy_events_increments);

   (i): 
   (ii): 
   ```

3. Complete the following assertion to make sure Winnie the Pooh reaches the SAD state.

   ```vhdl
   property sad_pooh;
   @(posedge clk) disable iff (reset)
   current_state == __(i)__ && (lost_balloon || rainy_day) |-> #1 current_state == __(ii)__;
   endproperty
   assert property (____(iii)____);

   (i): 
   (ii): 
   (iii): 
   ```
Problem 3: More K-maps!

1. Please translate the following 4-input Karnaugh-map into the most simplified expression.

\[
\begin{array}{cccc}
\text{A}\times\text{C} & 00 & 01 & 11 & 10 \\
00 & 0 & 0 & 1 & 1 \\
01 & 0 & 1 & 1 & 1 \\
11 & 0 & 1 & 1 & 1 \\
10 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Simplified Expression: 

2. Please translate the following 4-input Karnaugh-map into a simplified product-of-sums expression.

\[
\begin{array}{cccc}
\text{A}\times\text{B} & 00 & 01 & 11 & 10 \\
00 & 0 & 0 & 1 & 1 \\
01 & 1 & 1 & 0 & 1 \\
11 & 0 & 0 & 1 & 0 \\
10 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Product-of-Sums Expression: 

3. Suppose we want to design a 2-bit unsigned comparator circuit which computes if the 2-bit input, AB, is greater than or equal to the 2-bit input, CD (ie. the output is 1 if \( AB \geq CD \)). For clarity, A and C are the MSBs of their respective inputs. Write a simplified sum-of-products expression. The Karnaugh-map will not be graded, but is included for your convenience.

Sum-of-Products Expression: \[ \text{ } \]