Discussion 2

Rahul Kumar
About Me

- PhD student in EECS
- Advised by Bora
- Also did undergrad at Berkeley
- Research:
  - Automation/tooling for analog/mixed-signal design
  - SRAM22: open-source SRAM compiler for SKY130
  - You might get a chance to exercise SRAM22 in the ASIC labs!
  - Substrate: framework for writing generators (in Rust!)
  - Currently working on chiplet interconnect
Verilog

Content from Verilog Primer slides, previous offerings of EECS151
module full_adder (input x, input y, input cin, output s, output cout);
endmodule
module top_level (input switch0,  
    input switch1,  
    input switch2,  
    output LED0,  
    output LED1);  
  full_adder add (  
    .x(switch0),  
    .y(switch1),  
    .cin(switch2),  
    .s(LED0),  
    .cout(LED1)  
  );  
endmodule
module two_bit_adder (input [1:0] x, input [1:0] y, output [2:0] sum);
    wire [1:0] partial_sum;
    wire carry_out;
endmodule

wire [7:0] d;
wire [31:0] e;
wire [31:0] f;
assign f = {d, e[23:0]}; // Concatenation + Slicing
assign f = { 32{d[5]} }; // Replication + Indexing
Behavioral Operators

module full_adder (input x, input y, input cin, output s, output cout);
    assign s = x ^ y ^ cin;
    assign cout = (a && b) || (cin && (a ^ b));
endmodule

wire operand1  [31:0];
wire operand2  [31:0];
wire result    [32:0];
assign result  = operand1 + operand2;
2'd1 2-bit literal (decimal 1)
16'hAD14 16-bit literal (hexadecimal 0xAD14)
8'b01011010 8-bit literal (binary 0b01011010)
module equality_checker(input a [31:0], input b [31:0], output c [1:0]);
    assign c = a == b ? 2'd0 : (a < b ? 2'd1 : 2'd2);
endmodule
**wire vs. reg**

- **Wires:**
  - only support combinational logic
  - must be driven by an assign statement
  - cannot be driven in an always block
  - must be used to connect to the outputs of a child module
  - must be used for module input ports

- **Regs:**
  - can support either combinational or sequential logic
  - must be assigned in an always block
```verilog
input w;
input y;
input x;
reg [1:0] z;
always @(*) begin
    if(x) begin
        z[0] = w;
        z[1] = y;
    end
    else begin
        z[0] = y;
        z[1] = x;
    end
end
```
input [1:0] x;
reg [1:0] y;
always @(*) begin
    case(x)
        0: y = 2'd0;
        1: y = 2'd3;
        2: y = 2'd2;
        default: y = 2'd2;
    endcase
end
Latch synthesis

- Occurs when a reg isn’t always assigned a value
- Can be hard to debug
- Can result in mismatch between simulation and implementation (e.g. on an FPGA)
- Avoid by:
  - Assigning variables a default value
  - Using else/default clauses
  - Using continuous assignment
- Rule of thumb: if a variable is assigned in one path, it must be assigned in all paths.

```verilog
input [1:0] x;
reg [1:0] y;
always @(*) begin
  if(x == 2'b10) begin
    y = 2'd3;
  end else if(x == 2'b11) begin
    y = 2'd2;
  end
end
```
Blocking and non-blocking assignment

- **Sequential logic:**
  - `use always @(posedge clk)`
  - Use non-blocking assignments (`<=`)

- **Combinational logic**
  - `use always @(*)`
  - Use blocking assignments (`=`)

```verilog
class,input clk;
reg [1:0] x;
reg [1:0] next_x;

always @(*) begin
  next_x = x + 1;
end

always @(posedge clk) begin
  x <= next_x;
end
```
Practice

From EECS 151 FA21 Homework
Find the bug

```verilog
input [1:0] a;
input b, c;
reg x, y;
always @(a or b or c) begin
  case (a)
    2'b00 : x=b;
    2'b01 : x=c;
    2'b11 : y=b & c;
    2'b10 : y=b | c;
  endcase
end
```
module mux_4to1 (
    input a,b,c,d,
    input [1:0] sel,
    output out
);

always @(a or b or c or d) begin
    case (sel)
        2'b00 : out = a;
        2'b01 : out = b;
        2'b10 : out = c;
        2'b11 : out = d;
    endcase
end
endmodule
module edge_detect ( 
    input clk,
    input in,
    output reg edge_detect
);

    reg d;
    always@ (posedge clk) begin
        d <= in;
        edge_detect <= _______;
    end
endmodule
module jk_ff (  
  input j,  
  input k,  
  input clk,  
  output reg q  
);

always @ (posedge clk)  
  case (___1___)  
    2'b00 : q <= q;  
    2'b01 : q <= 0;  
    2'b10 : q <= ___2___;  
    2'b11 : q <= ___3___;  
  endcase  
endmodule

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q\text{new}</th>
<th>Q\text{old}</th>
<th>\sim Q_{old}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_{old}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>\sim Q_{old}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
module shift_reg (
    input clk,
    input resetn,
    input in,
    output reg out
 );

reg [2:0] mid;
always @(--1--)
    if (~resetn)
        {out, mid} <= __2__;
    else
        {out, mid} <= __3__;
endmodule