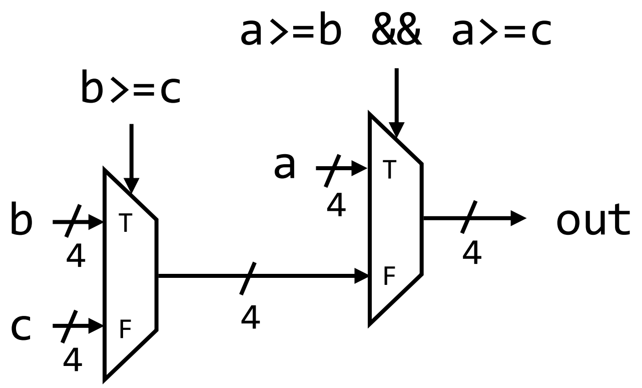
ECE 327/627
Digital Hardware Systems

Tutorial 1 Solutions

Q1

What hardware circuit is synthesized from this code?

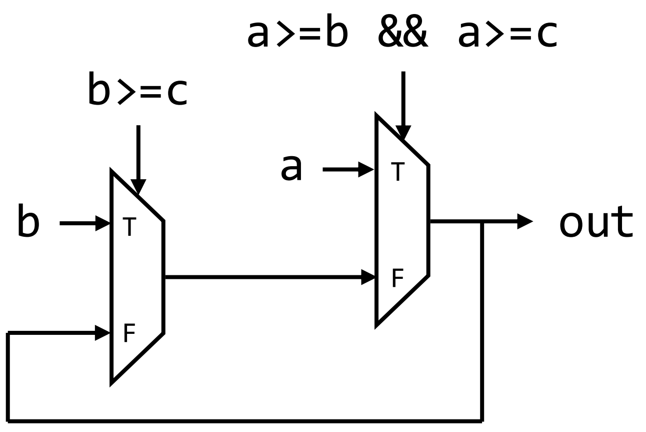


Step 1: *Determine input & output interfaces*
Step 2: *always_comb means it's a combinational circuit and output is not registered*
Step 3: *Nested if-else translates to a chain of multiplexers*
Step 4: *Priority of nested ifs determines the order of multiplexers. Highest priority (i.e., first if statement) is the multiplexer closest to the output*

```
module q1 (
	input [3:0] a,
	input [3:0] b,
	input [3:0] c,
	output logic [3:0] out
);

always_comb begin
	if (a>=b && a>=c) begin
		out = a;
	end else if (b>=c) begin
		out = b;
	end else begin
		out = c;
	end
end

endmodule
```

Q2

Write the SystemVerilog code that is synthesized into the following circuit.

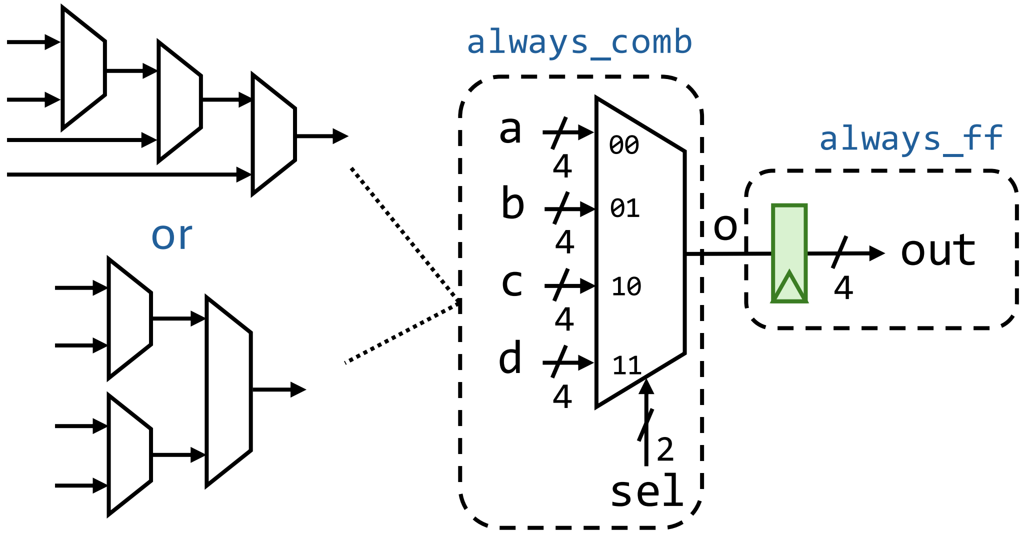


```
module q2 (
     input [3:0] a,
     input [3:0] b,
     input [3:0] c,
     output logic [3:0] out
);

always_comb begin
     if (a>=b && a>=c) begin
          out = a;
     end else if (b>=c) begin
          out = b;
     end
end

endmodule
```

Incomplete definitions of if-else statements in always blocks results in a combinational feedback from output to input. This is the logical representation of an inferred latch at the output. Almost always you do not mean to have latches in your design. Designing circuits with latches is tricky and needs careful consideration to avoid metastability.

Q3

What hardware circuit is synthesized from this code?

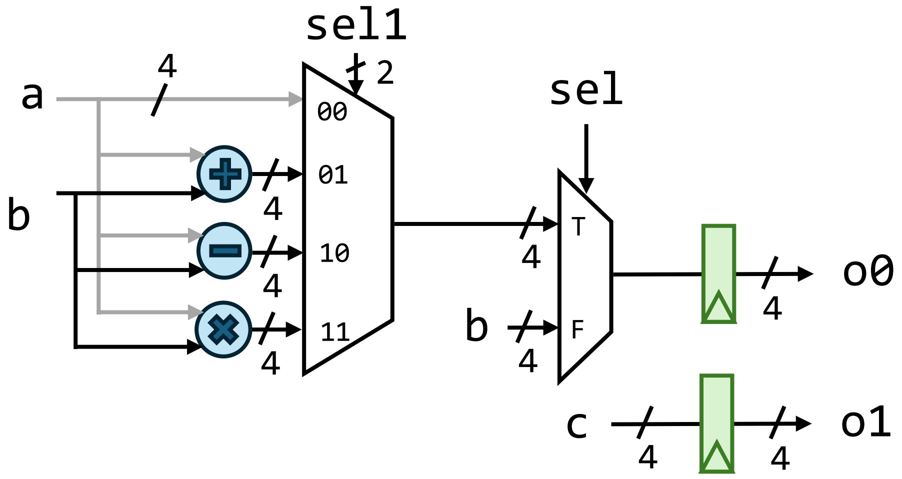


This circuit has an always_ff and an always_comb blocks. We can reason about each block separately and translate it to hardware, then stitch them together. Since the order of the if-else statement doesn't matter in this case (mutually exclusive – using == and not relational operators), you can think of it as a flat 4:1 multiplexer. This multiplexer can be implemented using three 2:1 multiplexers organized in a chain or a tree.

```
module q3 (
        input [3:0] a,
        input [3:0] b,
        input [3:0] c,
        input [3:0] d,
        input [1:0] sel,
        input clk,
        input rst,
        output logic [3:0] out
);

logic [3:0] o;

always_comb begin
        if (sel == 2’b00) o <= a;
        else if (sel == 2’b01) o <= b;
        else if (sel == 2’b10) o <= c;
        else o <= d;
end

always_ff @ (posedge clk) begin
        if (rst) out <= 4’d0;
        else out <= o;
end

endmodule
```

Q4

What hardware circuit is synthesized from this code?



Having an always_ff means that both outputs o0 and o1 are registered. This code can be refactored as:

```
o0 <= b;
if (sel) o0 <= // some value depending on case
if (!sel) o1 <= b;
o1 <= c;
```

An assignment before an if with a missing else represents the false (default) branch. An assignment after an if-else condition results in all previous assignments to be ignored. Finally, the case statement is translated to a flat 4:1 multiplexer. Remember, in SystemVerilog, last assignment wins!

```
module q4 (
    input [3:0] a,
    input [3:0] b,
    input [3:0] c,
    input sel,
    input [1:0] sel1,
    input clk,
    output logic [3:0] o0,
    output logic [3:0] o1
);

always_ff @ (posedge clk) begin
    o0 <= b;
    if (sel) begin
        case (sel1)
            2’b00: o0 <= a;
            2’b01: o0 <= a+b;
            2’b10: o0 <= a-b;
            2’b11: o0 <= a*b;
            default: o0 <= 0;
        endcase
    end else if (!sel) begin
        o1 <= b;
    end
    o1 <= c;
end

endmodule
```

Q5

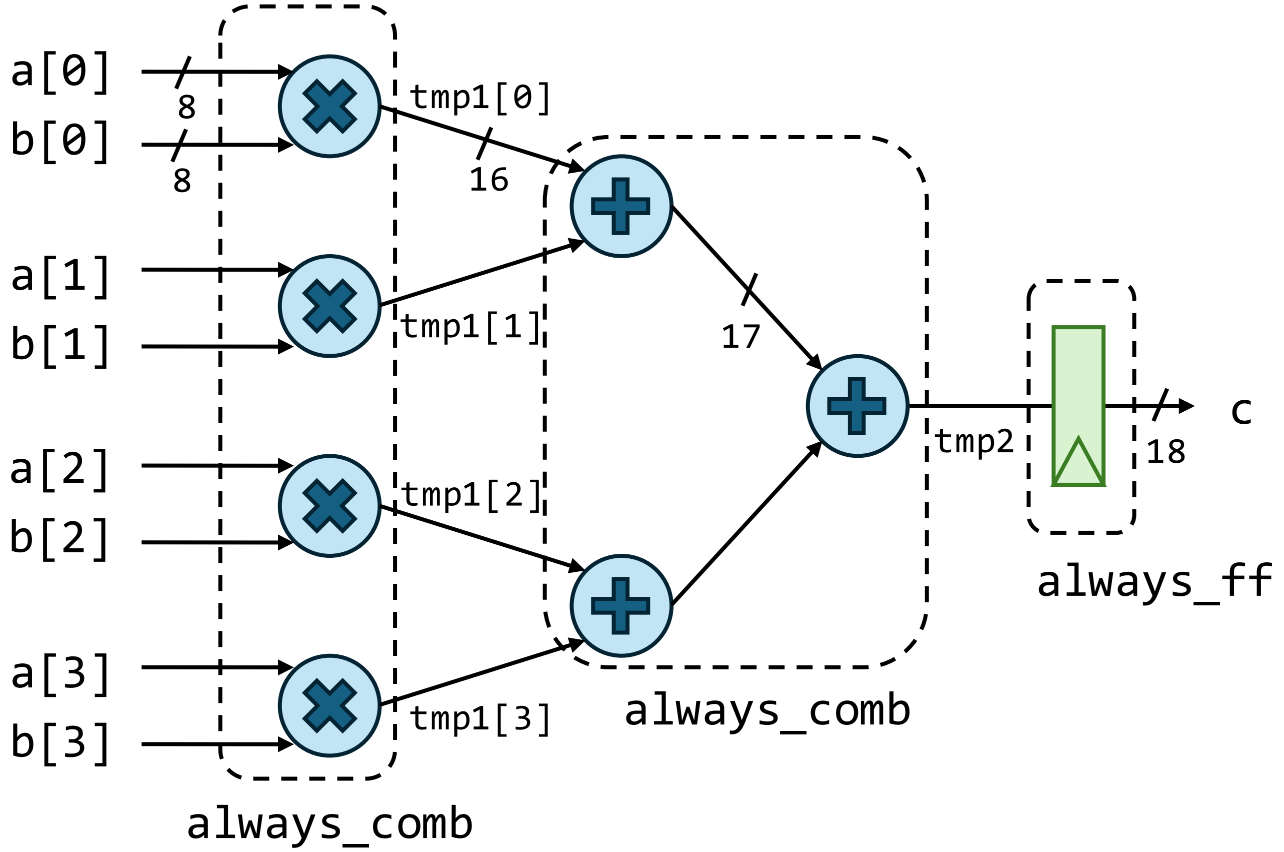
Draw the diagram of the circuit generated when elaborating this SystemVerilog code. What should be the values of M & L for this circuit to be functionally correct?

```
module q5 # (
     parameter K = 4,
     parameter N = 8,
     parameter M = …,
     parameter L = …
)(
     input clk,
     input rst,
     input [N-1:0] a [0:K-1],
     input [N-1:0] b [0:K-1],
     output logic [M-1:0] c
);

logic [L-1:0] tmp1 [0:K-1];
logic [M-1:0] tmp2;

integer i, j;
```

```
always_comb begin
     for (i=0; i<K; i=i+1) begin
          tmp1[i] = a[i] * b[i];
     end
end

always_comb begin
     tmp2 = 0;
     for (j=0; j<K; j=j+1) begin
          tmp2 = tmp2 + tmp1[j];
     end
end

always_ff @ (posedge clk) begin
     if (rst) c <= ’d0;
     else c <= tmp2;
end

endmodule
```

Q5

Draw the diagram of the circuit generated when elaborating this SystemVerilog code. What should be the values of M & L for this circuit to be functionally correct?


a[0]
b[0]
8
8
a[1]
b[1]
a[2]
b[2]
a[3]
b[3]
tmp1[0]
16
tmp1[1]
tmp1[2]
tmp1[3]
17
tmp2
18
c
always_comb
always_comb
always_ff