Computer Aided Design
Assignment Nº3

Pooya Kabiri, Alireza Moradi, Ali M. Movahedian

Department of Computer Science

Iran University of Science and Technology

December 2020

0 Team

Alireza Moradi	96521479
Pooya Kabiri	96521434
AliMohammad Movahedian	96521497

1 Carry Look Ahead

1.1 Behavioral Implementation

```
module cla_behav(
        input [3:0] a,
        input [3:0] b,
        input cin,
        output reg [3:0] sum,
        output reg cout);

   wire [3:0] P;
   wire [3:0] G;
   reg [4:0] c;
   reg [4:0] temp;
   assign P[3:0] = a[3:0] ^ b[3:0];
   assign G[3:0] = a[3:0] & b[3:0];

   always @*
                begin
                   c[0] = cin;

                   c[1] = G[0] | (P[0] & c[0]);

```

```
                    c[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & c[0]);

                    c[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0])
                                                | (P[2] & P[1] & P[0] & cin);

                    c[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1])
                                               | (P[3] & P[2] & P[1] & G[0])
                                               | (P[3] & P[2] & P[1] & P[0] & cin);

                    temp[4:0] = {1'b0, P[3:0]} ^ c[4:0];
                    sum = temp[3:0];
                    cout = temp[4];
                end
endmodule
```

1.2 Gate Level Design

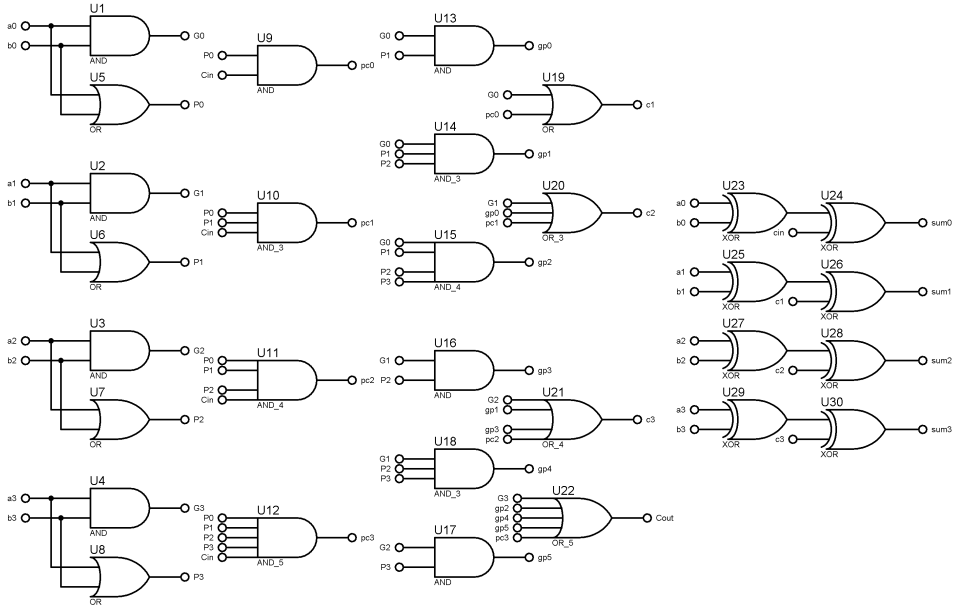

Figure 1: Proteus Implementation

1.3 Gate Level Implementation

```
module cla(
        input [3:0] a,
        input [3:0] b,
        input cin,
```

```
        output [3:0] sum,
        output cout);

        wire [3:1] c;
        wire [3:0] pc;
        wire [5:0] gp;
        wire [3:0] G,P;

        and #(10) (G[0], a[0], b[0]);
        and #(10) (G[1], a[1], b[1]);
        and #(10) (G[2], a[2], b[2]);
        and #(10) (G[3], a[3], b[3]);
        or #(10) (P[0], a[0], b[0]);
        or #(10) (P[1], a[1], b[1]);
        or #(10) (P[2], a[2], b[2]);
        or #(10) (P[3], a[3], b[3]);

        and #(10) (pc[0], P[0], cin);
        and #(10) (pc[1], P[1], P[0], cin);
        and #(10) (pc[2], P[2], P[1], P[0], cin);
        and #(10) (pc[3], P[3], P[2], P[1], P[0], cin);

        and #(10) (gp[0], G[0], P[1]);
        and #(10) (gp[1], G[0], P[1], P[2]);
        and #(10) (gp[2], G[0], P[1], P[2], P[3]);
        and #(10) (gp[3], G[1], P[2]);
        and #(10) (gp[4], G[1], P[2], P[3]);
        and #(10) (gp[5], G[2], P[3]);

        or #(10) (c[1], G[0], pc[0]);
        or #(10) (c[2], G[1], gp[0], pc[1]);
        or #(10) (c[3], G[2], gp[1], gp[3], pc[2]);
        or #(10) (cout, G[3], gp[2], gp[4], gp[5], pc[3]);

        xor #(30) (sum[0], a[0], b[0], cin);
        xor #(30) (sum[1], a[1], b[1], c[1]);
        xor #(30) (sum[2], a[2], b[2], c[2]);
        xor #(30) (sum[3], a[3], b[3], c[3]);
endmodule
```

2 Protein Detection

2.1 BLAST

BLAST is a algorithm in fields of biotechnology used for comparing primary biological sequence information, such as the amino-acid sequences of proteins.

It can be referred as a query in which an input sequence, database to search and other optional parameters are passed into.

although a software algorithm by nature, BLAST can benefit from hardware accelerators e.g. on FPGAs.

2.2 Hardware Approach

Because of the sequential nature, BLAST can be modeled using a Finite-State-Machine.

There can be as many states as needed for deteecting a sequence of data (like the "open-window" example in previous homework).

FSMs can be implemented using a hardware logic (gates for example), so in this way a BLAST algorithm can be implemented by hardware in which the speed of processing probably increases significantly.

2.3 LDLD/MST Detector FSM

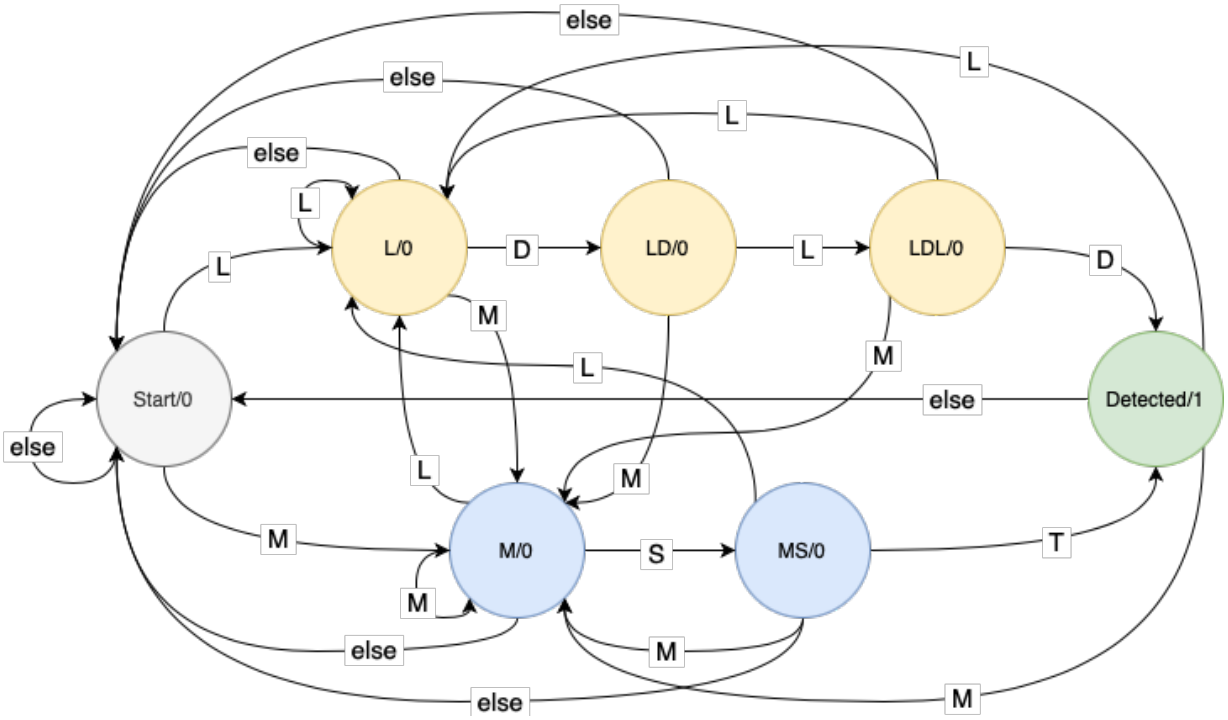

Figure 2: FSM

2.4 LDLD/MST Detector Verilog Implementation

```
module LDLD_MST_protein_det(
        input [7:0] x,
        input clk,
        input rst,
        output reg y);

        reg [2:0] curSt, nextSt;

```

```
9         parameter Start = 0,
10        L = 1, LD = 2, LDL = 3,
11        M = 4, MS = 5,
12        Detected = 6;
13
14        always@(posedge clk) begin
15                if (rst) begin
16                        curSt = Start;
17                        y = 0;
18                end
19                else curSt = nextSt;
20        end
21
22        always@(curSt or x) begin
23                case(curSt)
24                        Start: begin
25                                y = 0;
26                                if (x == "L") nextSt = L;
27                                else if (x == "M") nextSt = M;
28                                else nextSt = Start;
29                        end
30
31                        L: begin
32                                y = 0;
33                                if (x == "D") nextSt = LD;
34                                else if (x == "L") nextSt = L;
35                                else if (x == "M") nextSt = M;
36                                else nextSt = Start;
37                        end
38
39                        LD: begin
40                                y = 0;
41                                if (x == "L") nextSt = LDL;
42                                else if (x == "D") nextSt = Start;
43                                else if (x == "M") nextSt = M;
44                                else nextSt = Start;
45                        end
46
47                        LDL: begin
48                                y = 0;
49                                if (x == "L") nextSt = L;
50                                else if (x == "D") nextSt = Detected;
51                                else if (x == "M") nextSt = M;
52                                else nextSt = Start;
53                        end
54
55                        Detected: begin
56                                y = 1;
57                                if (x == "L") nextSt = L;
```

```
                                else if (x == "M") nextSt = M;
                                else nextSt = Start;
                        end

                        M: begin
                                y = 0;
                                if (x == "L") nextSt = L;
                                else if (x == "S") nextSt = MS;
                                else if (x == "M") nextSt = M;
                                else nextSt = Start;
                        end

                        MS: begin
                                y = 0;
                                if (x == "L") nextSt = L;
                                else if (x == "T") nextSt = Detected;
                                else if (x == "M") nextSt = M;
                                else nextSt = Start;
                        end
                endcase
        end
endmodule
```

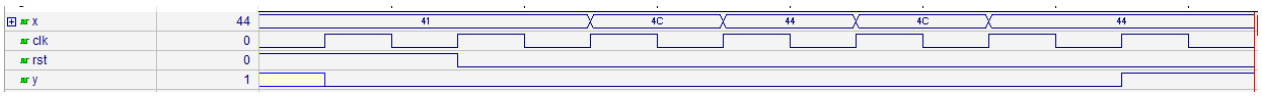

Figure 3: LDLD Simulation

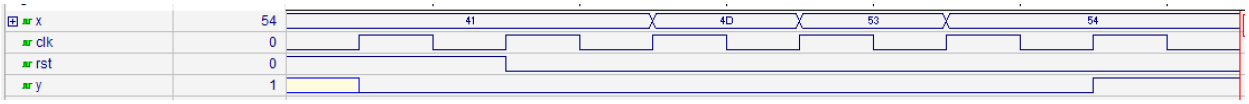

Figure 4: MST Simulation

3 Elevator

3.1 Turn on elevator

```
module start (input clk,
                input reset,
                input [4:0] in,         // indicates a new floor button that is pushed
                output reg out );         // 1 goes up, 0 goes down
          parameter off = 0,
                    on = 1;
        reg [1:0] cur_state, next_state;
        integer buttons [20:0];
```

```
9         integer dir, i, up_min_dist, down_min_dist = 0;
10        integer cur_floor = 1;
11
12        // zero out buttons array
13        initial
14      begin
15        for(i=0;i<=20;i=i+1)
16          buttons[i]=0;
17      end
18
19    always @ (posedge clk) begin
20      if (reset) begin
21                cur_state = off;
22                  out = 0;
23        end
24        else cur_state <= next_state;
25    end
26
27    always @ (cur_state or in) begin
28        if(in != 0) begin
29                // Iterate over pressed buttons
30                for(i=0; i<=20; i++)
31                        if(buttons[i] ==0) begin
32                                buttons[i] = in;
33                                break;
34                        end
35                        else begin
36                                if(buttons[i] < cur_floor) begin
37                                        dir--;
38                                        if(cur_floor - buttons[i] < down_min_dist) begin
39                                                down_min_dist = cur_floor - buttons[i];
40                                        end
41                                end
42                                else begin
43                                        dir++;
44                                        if(buttons[i] - cur_floor < up_min_dist) begin
45                                                up_min_dist = buttons[i] - cur_floor;
46                                        end
47                                end
48                        end
49
50                // Decide the directon
51                if(dir > 0) begin
52                        assign out = 1;
53                end
54                else if(dir < 0) begin
55                        assign out = 0;
56                end
57                else begin
```

```
                        if(up_min_dist < down_min_dist) begin
                                assign out = 1;
                        end
                        else begin
                                assign out = 0;
                        end
                end

                next_state = on;
        end
  end

endmodule
```

3.2 Turn off elevator

```
module stop (input clk,
                  input reset,
                  input [1:0] in,        // 0 stop, 1 up, 2 down
                  output reg out );        // 1 stops, 0 moves
          parameter off = 0,
                    on = 1;
        reg [1:0] cur_state, next_state;

  always @ (posedge clk) begin
    if (reset) begin
              cur_state = on;
                out = 0;
        end
        else cur_state <= next_state;
  end

  always @ (cur_state or in) begin
        if(cur_state == on) begin
                if(in == 0) begin
                        next_state = off;
                        assign out = 1;
                end
        end
  end

endmodule
```

3.3 Manage air conditioner

```
module ac (input clk,
                  input reset,
                  input in,        // 0: 1degree colder, 1: 1 degree hotter
                  output reg out );        // 1: between 24,26
          parameter normal = 0,
                        cold = 1,
                        hot = 2;
        reg [1:0] cur_state, next_state;
        integer temp = 25;

  always @ (posedge clk) begin
    if (reset) begin
              cur_state = normal;
                out = 1;
        end
        else cur_state <= next_state;
  end

  always @ (cur_state or in) begin
        case(cur_state)
                normal: begin
                        if(temp > 26) begin
                                next_state = hot;
                                assign out = 0;
                        end
                        else if(temp < 24) begin
                                next_state = cold;
                                assign out = 0;
                        end
                end
                hot: begin
                        repeat(3) @(posedge        clk);
                        temp--;
                        if(temp <= 26) begin
                                next_state = normal;
                                assign out = 1;
                        end
                end
                cold: begin
                        repeat(3) @(posedge        clk);
                        temp++;
                        if(temp <= 24) begin
                                next_state = normal;
                                assign out = 1;
                        end
                end
```

```
        endcase

        if(in == 1) begin
                temp++;
        end
        else begin
                temp--;
        end
  end

endmodule
```

4 Timing

4.1 Critical Path

The critical path is shown using the blue line.

$$Critical\ Delay = t_{pd-NOT} + t_{pd-AND3} + t_{pd-OR4} = 30 + 50 + 60 = 140$$

4.2 Maximum Frequency

$$T_c > t_{pcq} + t_{pd} + t_{setup} + t_{skew} \Rightarrow T_c > 40 + 140 + 60 + 40$$

$$\Rightarrow T_c > 280\ ps \Rightarrow f_{max} = 3.571\ GHz$$

4.3 Timing Requirement

$$t_{cd} = t_{shortest-path} = t_{cd-AND3} + t_{cd-OR4} = 40 + 50 = 90$$

$$t_{ccq} + t_{cd} > t_{hold} + t_{skew} \Rightarrow 20 + 90 > 120 + 40? \Rightarrow Not\ Satisfied.$$

So we add tri-state buffers to multiple paths in order for timing requirement to be satisfied.

Tri-state buffers are added both in the longest and shortest path. The fixed circuit is shown in Figure 5.

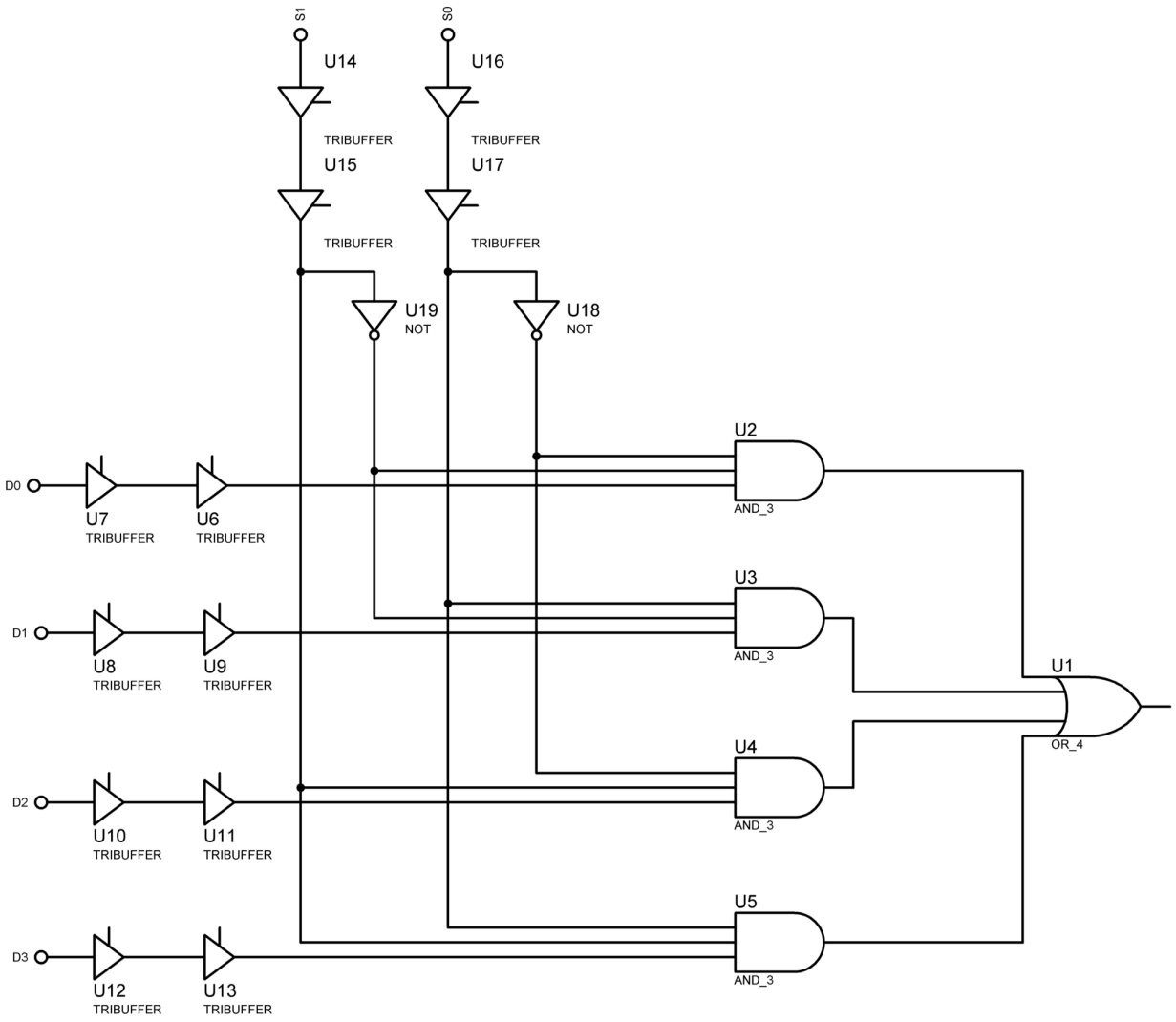

Figure 5: Fixed Circuit

Now,we calculate the new Contamination and Propagation Delays and check the timing requirements of the new circuit.

The orange lines indicates the shortest path and the blue line indicates the critical (longest) path.

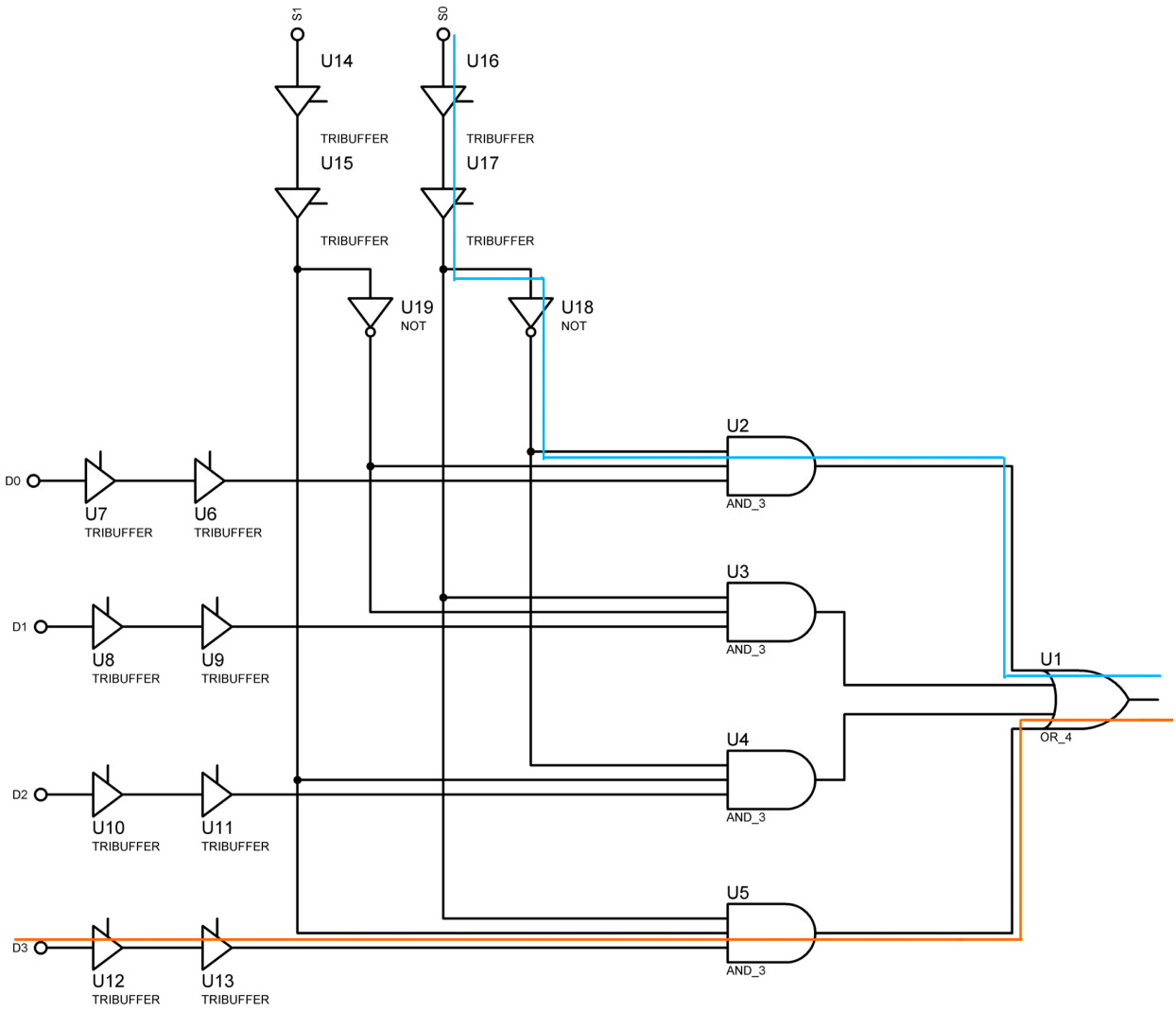

Figure 6: Critical and Shortest Paths

First, we calculate t_{cd} and t_{pd}:

$$t_{cd} = t_{shortest-path} = 2 * t_{cd-buffer-in2out} + t_{cd-AND3} + t_{cd-OR4} = 2 * 25 + 40 + 50 = 140$$

$$t_{pd} = t_{critical-path} = 2*t_{pd-buffer-in2out}+t_{pd-NOT}+t_{pd-AND3}+t_{pd-OR4} = 2*35+30+50+60 = 210$$

Then, we check hold time constraint:

$$t_{ccq} + t_{cd} > t_{hold} + t_{skew} \Rightarrow 20 + 140 > 120 + 40? \Rightarrow Satisfied.$$

Then, we check setup time constraint:

$$T_c > t_{pcq} + t_{pd} + t_{setup} + t_{skew} \Rightarrow T_c > 40 + 210 + 60 + 40$$

$$\Rightarrow T_c > 350\ ps \Rightarrow f_{max} = 2.857\ GHz$$

4.4 Limited Clock Period

In this case the circuit should satisfy the $T_c < 300\ ps$ constraint additional to Hold and Setup time constraints.

$$t_{pcq} + t_{pd} + t_{setup} + t_{skew} < T_c \leq 300$$
$$\Rightarrow 40 + t_{pd} + 60 + 40 < T_c \leq 300 \Rightarrow t_{pd} \leq 160\ ps$$

$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$
$$\Rightarrow 20 + t_{cd} > 120 + 40 \Rightarrow t_{cd} > 140$$

With given gate Contamination and Propagation delays, there is not a implementation that can satisfy both the needed t_{cd} and t_{pd}.

The circuit that satisfies the Hold and Setup time constraints with minimum clock period is the one plotted in section 4.3 (Figure 5).

Referring to the stated circuit, the minimum clock period is 350 ps so the maximum clock frequency is 2.857 GHz. It is important to note that the there is no other implementation which can satisfy timing constraints with a clock period smaller than 350 ps.