Habib University
Digital System Design - CE325

Quiz (Designing an FPGA) - Report

Instructor: Dr. Arsalan Javed

Sadiqah Mushtaq - 07152

1 Architecture

My FPGA comprises of a total of 6 modules:

1. two_bit_full_adder
2. D_FlipFlop
3. Controller Module
4. CLB Module
5. Config_FPGA Module
6. FPGA_new

2 *FPGA Architecture*

2.1 *two_bit_full_adder* Module

- **Inputs**:
  - `A` and `B`: Two 2-bit input vectors representing the operands.
  - `Cin`: Carry input.
- **Outputs**:
  - `sum`: A 2-bit output vector representing the sum.
  - `Cout`: Carry output.
- **Functionality**:
  - Implements a two-bit full adder circuit using combinational logic.
  - Utilizes a 'case' to make a lookup tables.
  - Cin, A and B are concatenated and given as cases to the lookup table. Since that makes the input 5 bits, there are a total of $2^5 = 32$ entries in the lookup table.

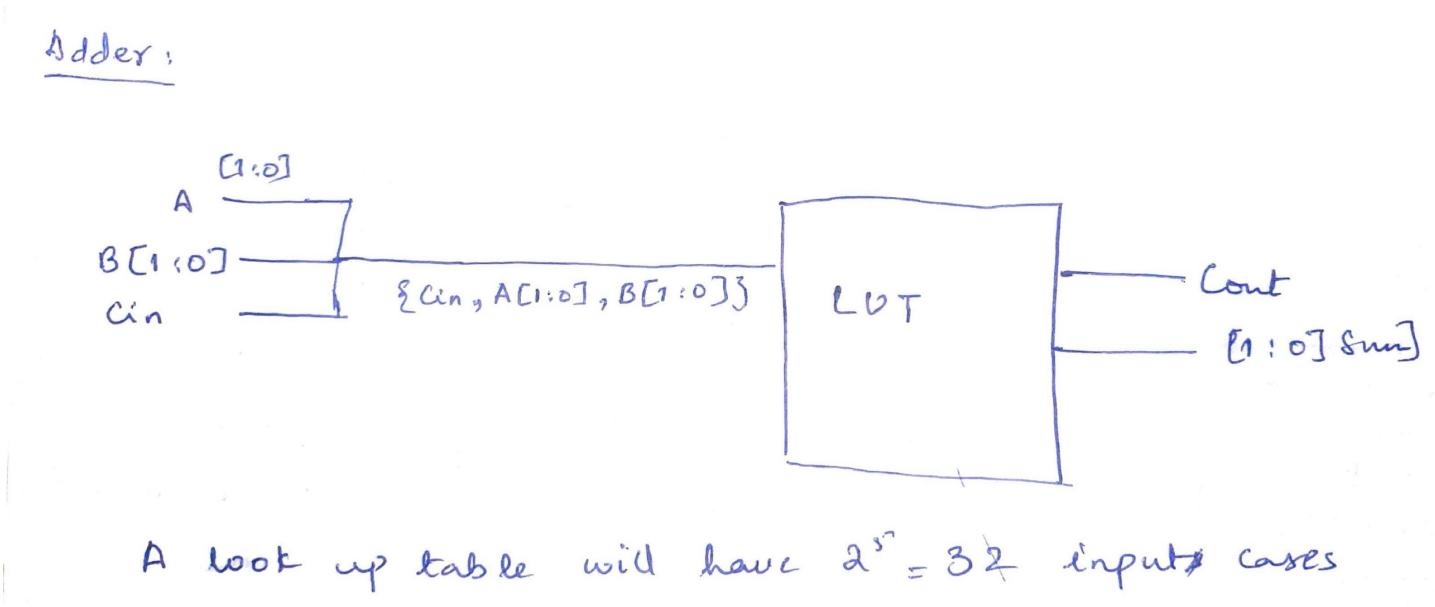

Figure 1: Adder Architecture drawn

2.2 *D_FlipFlop* Module

- **Inputs**:
  - `d`: Data input, a 2-bit vector.
  - `clk`: Clock input.
  - `reset`: Reset input.
- **Outputs**:
  - `q`: 2-bit output vector.
- **Functionality**:
  - Implements a D flip-flop circuit.
  - Updates the output `q` based on the input data `d` on each clock edge.
  - Resets the output `q` to a default value when the reset signal is asserted.
  - is used to delay the sum and Cout from the adder if the select signal of mux is asserted in the CLB.

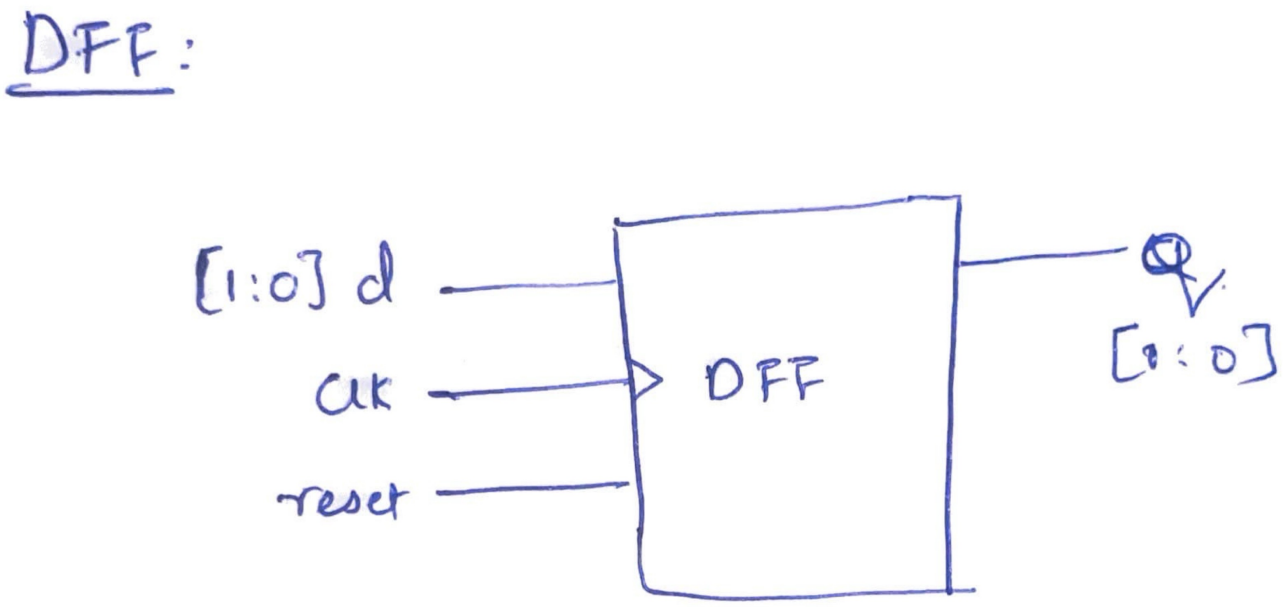

Figure 2: D Flip Flop Architecture drawn

2.3 *Controller Module*

- **Inputs**: `bitfile_input` (an 8-bit configuration data)
- **Outputs**: Four sets of 2-bit control signals (`Control_signal1` to `Control_signal4`), each controlling a set of muxes in the configuration block.
- **Functionality**: The Controller module takes the 8-bit configuration data and slices it into four sets of 2-bit control signals. These signals determine which configuration options are selected for each CLB and how they are connected. In the schematic, it simply realised as 5 buffers.

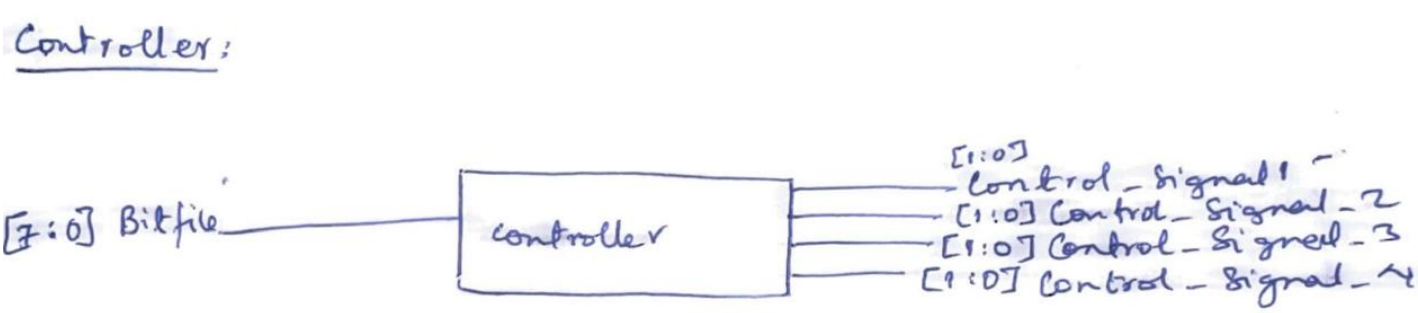

Figure 3: Controller Architecture drawn

2.4 *CLB Module*

- **Inputs**: `clk`, `reset`, two 2-bit inputs `In_1` and `In_2`, a carry input `Cin`, and a multiplexer select signal `mux_in`.
- **Outputs**: Two 2-bit outputs `out1_final` and `out2_final`.
- **Functionality**: The CLB module implements a configurable logic block. It consists of a two-bit full adder followed by two D flip-flops. The input data is processed through the adder, and the result is latched by the flip-flops. The `mux_in` signal selects whether to use the output directly or with a one-cycle delay from the flip flop.

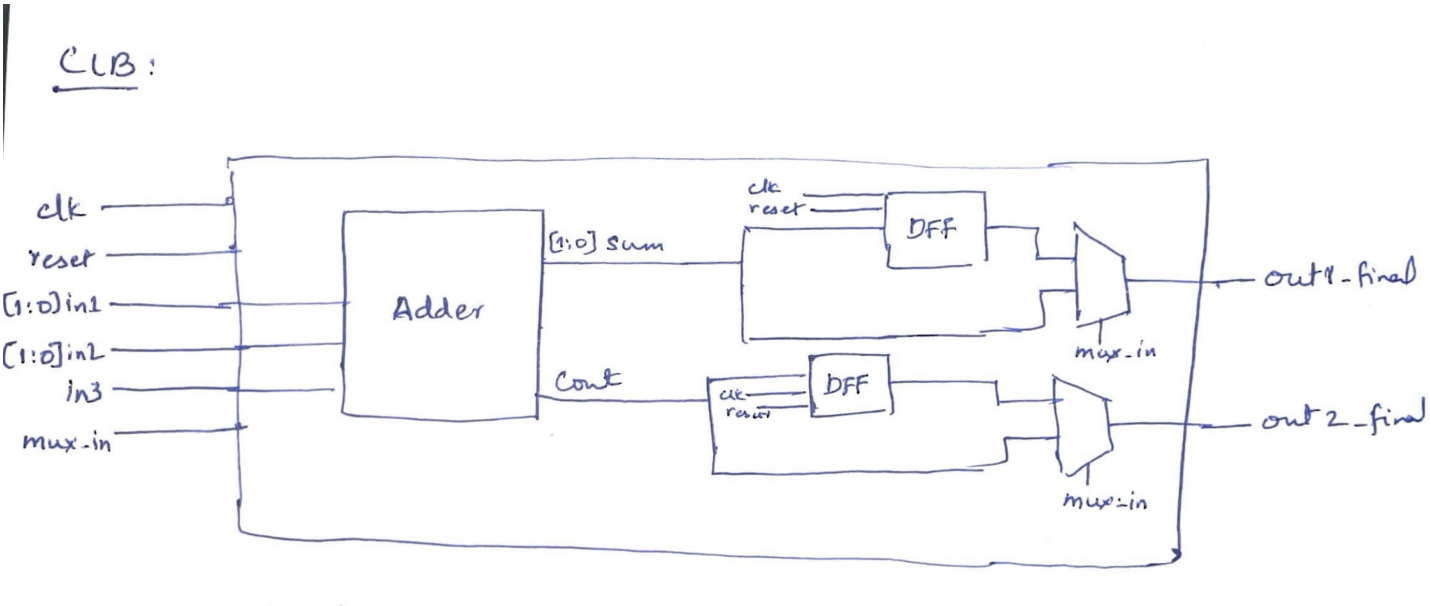

Figure 4: CLB Architecture drawn

2.5 *Config_FPGA Module*

- **Inputs**: Two sets of 8-bit inputs `In_1` and `In_2`, four sets of 2-bit sum outputs from CLBs (`Sum_CLB1` to `Sum_CLB4`), and four sets of carry outputs (`Cout_CLB1` to `Cout_CLB4`). Also, there are four control signals (`sel1` to `sel4`) to select the routing paths.
- **Outputs**: Concatenated 8-bit sum output `Sum`, and a carry output `Cout_final`. Also, it controls the carry inputs to each CLB (`Cin_CLB1` to `Cin_CLB4`) and the input slices for each CLB.
- **Functionality**: This module routes the output of each CLB to the input of subsequent CLBs based on the control signals. The 8-bit input data in the FPGA is routed through the routing lines in the configuration block. Each input is sliced into 2-bit segments, which are then selected by the muxes to be used as inputs for each CLB based on control signal(sliced bit bitfile) from the controllers. Additionally, the carry outputs from each CLB are routed back to the configuration block to be used as carry inputs for subsequent CLBs. The selection of all routing paths and input slices is determined by the control signals generated by the Controller module based on the bitfile. The following the highlevel architecture of the the block.

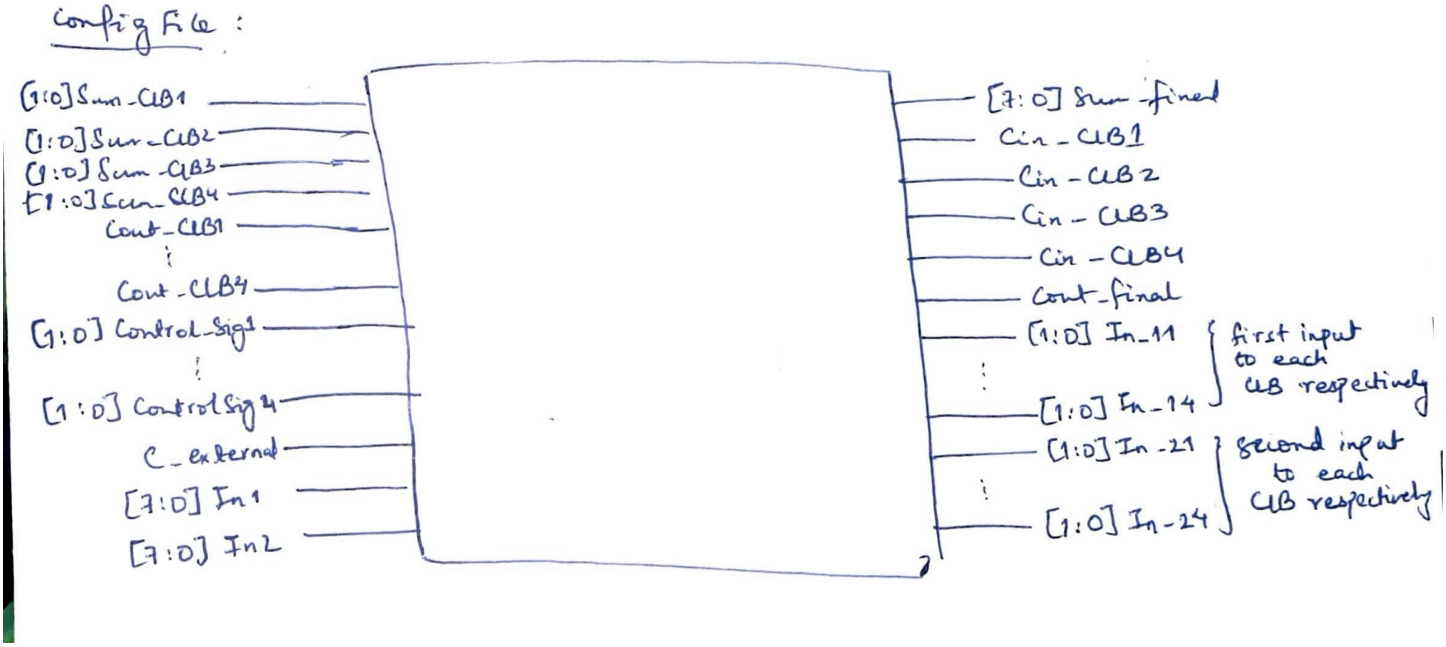

Figure 5: Config_FPGA Architecture drawn

2.5.1 Carry and Input Routing Architecture

In the proposed architecture, we illustrate how inputs are directed to the first CLB. This involves three muxes, each of which takes the same select signal as input. These muxes play a crucial role in determining which segment of input is added in that specific CLB. Additionally, they are responsible for selecting the carry-in value originating from a particular CLB.

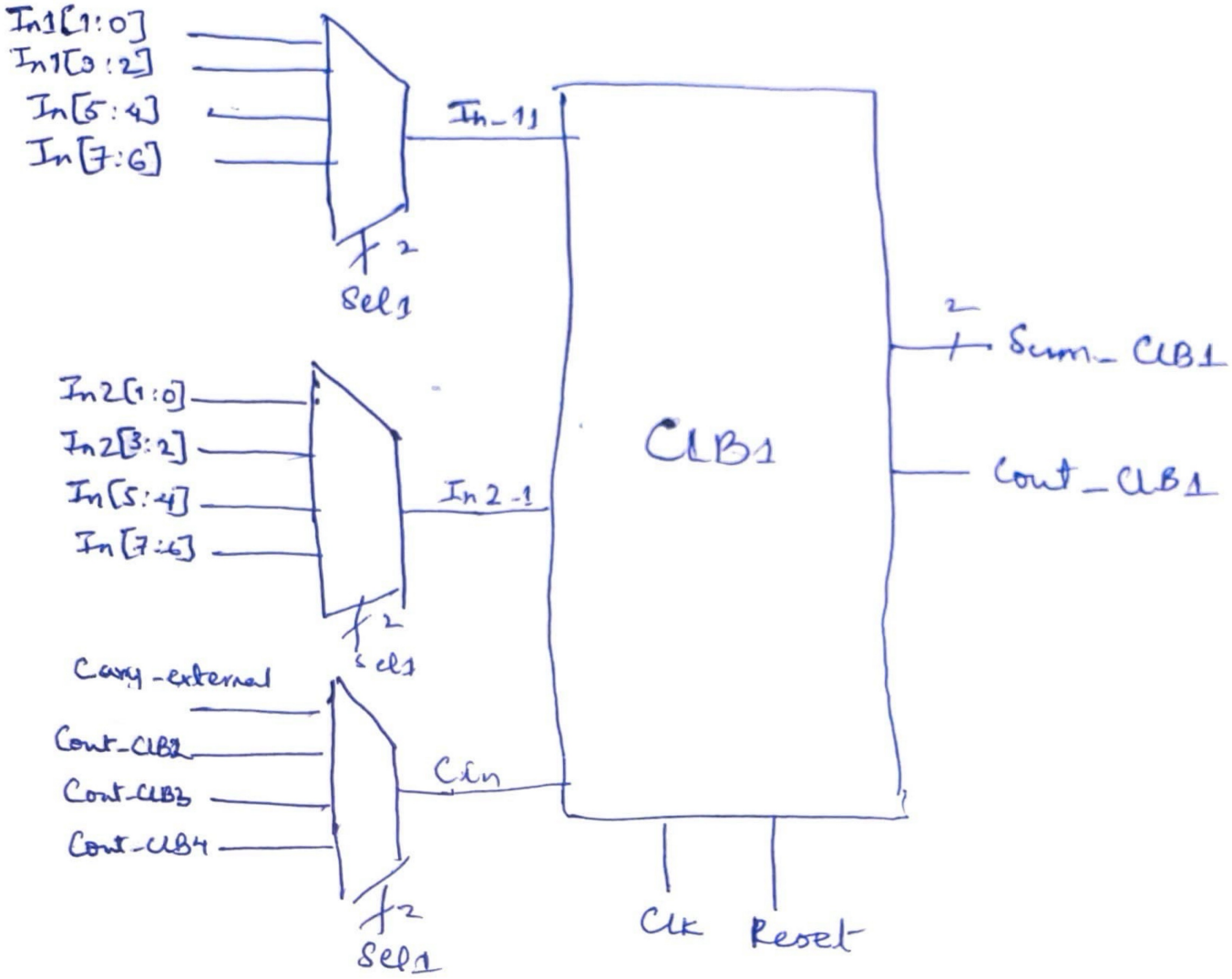

Figure 6: Carry and Input Routing Architecture

2.5.2 Sum Routing Architecture

In my proposed architecture, the routing block should also handles the routing of output sums from CLBs. I have provided a configurable option to determine which output port the sum exits through the routing block. This functionality is achieved using a mux, which utilizes the same control signal to make this determination. Ideally this should happen However, in my code I have not implemented Sum routing since it was utterly hard to cover all cases. Instead, I have concatenated all sums within the routing block assuming that the first sum is from CLB1.

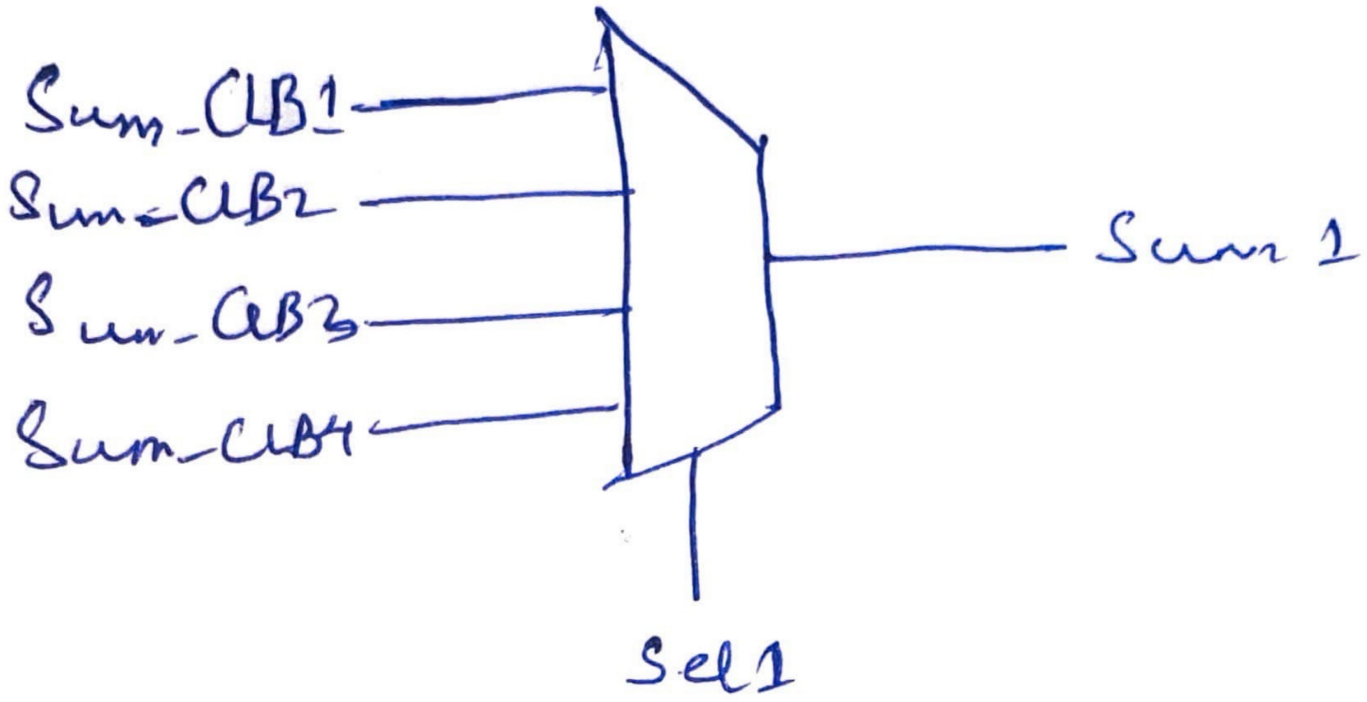

Figure 7: Sum Routing Architecture

2.6 *FPGA Module*

- **Inputs**: `clk`, `reset`, two sets of 8-bit inputs `In_1` and `In_2`, and an 8-bit configuration data `bitfile_input`.
- **Outputs**: An 8-bit sum output `Sum` and a carry output `Cout`.
- **Functionality**: The top-level module instantiates the Controller, four CLBs, and Config_FPGA modules. It controls the flow of data and configuration throughout the FPGA. The configuration data determines the interconnections between the CLBs, and the inputs are processed through the CLBs to produce the final output sum and carry.

3 Verilog Code

3.1 2 bit full AdderAdder

```
1  `timescale 1ns / 1ps
2
3  module two_bit_full_adder(
4      input [1:0] A, B,
5      input Cin,
6      output reg [1:0] sum,
7      output reg Cout
8  );
9
10 always @* begin
11     // LUT for addition
12     case({Cin, A, B})
13         5'b00000: begin
14             sum = 2'b00;
15             Cout = 1'b0;
16             end
17
```

```
18          5'b00001: begin
19              sum = 2'b01;
20              Cout = 1'b0;
21              end
22          5'b00010: begin
23              sum = 2'b10;
24              Cout = 1'b0;
25              end
26          5'b00011: begin
27              sum = 2'b11;
28              Cout = 1'b0;
29              end
30          5'b00100: begin
31              sum = 2'b01;
32              Cout = 1'b0;
33              end
34          5'b00101: begin
35              sum = 2'b10;
36              Cout = 1'b0;
37              end
38          5'b00110: begin
39              sum = 2'b11;
40              Cout = 1'b0;
41              end
42          5'b00111: begin
43              sum = 2'b00;
44              Cout = 1'b1;
45              end
46          5'b01000: begin
47              sum = 2'b10;
48              Cout = 1'b0;
49              end
50          5'b01001: begin
51              sum = 2'b11;
52              Cout = 1'b0;
53              end
54          5'b01010: begin
55              sum = 2'b00;
56              Cout = 1'b1;
57              end
58          5'b01011: begin
59              sum = 2'b01;
60              Cout = 1'b1;
61              end
62          5'b01100: begin
63              sum = 2'b11;
64              Cout = 1'b0;
65              end
66          5'b01101: begin
67              sum = 2'b00;
68              Cout = 1'b1;
69              end
70          5'b01110: begin
71              sum = 2'b01;
72              Cout = 1'b1;
73              end
74          5'b01111: begin
75              sum = 2'b10;
76              Cout = 1'b1;
77              end
78          5'b10000: begin
79              sum = 2'b01;
80              Cout = 1'b0;
```

```
81              end
82          5'b10001: begin
83              sum = 2'b10;
84              Cout = 1'b0;
85              end
86          5'b10010: begin
87              sum = 2'b11;
88              Cout = 1'b0;
89              end
90          5'b10011: begin
91              sum = 2'b00;
92              Cout = 1'b1;
93              end
94          5'b10100: begin
95              sum = 2'b10;
96              Cout = 1'b0;
97              end
98          5'b10101: begin
99              sum = 2'b11;
100             Cout = 1'b0;
101             end
102         5'b10110: begin
103             sum = 2'b00;
104             Cout = 1'b1;
105             end
106         5'b10111: begin
107             sum = 2'b01;
108             Cout = 1'b1;
109             end
110         5'b11000: begin
111             sum = 2'b11;
112             Cout = 1'b0;
113             end
114         5'b11001: begin
115             sum = 2'b00;
116             Cout = 1'b1;
117             end
118         5'b11010: begin
119             sum = 2'b01;
120             Cout = 1'b1;
121             end
122         5'b11011: begin
123             sum = 2'b10;
124             Cout = 1'b1;
125             end
126         5'b11100: begin
127             sum = 2'b00;
128             Cout = 1'b1;
129             end
130         5'b11101: begin
131             sum = 2'b01;
132             Cout = 1'b1;
133             end
134         5'b11110: begin
135             sum = 2'b10;
136             Cout = 1'b1;
137             end
138         5'b11111: begin
139             sum = 2'b11;
140             Cout = 1'b1;
141             end
142         default:  begin
143             sum = 2'b00;
```

```
144             Cout = 1'b0;
145             end
146     endcase
147 end
148
149 endmodule
```

3.1.1 Schematic Explanation

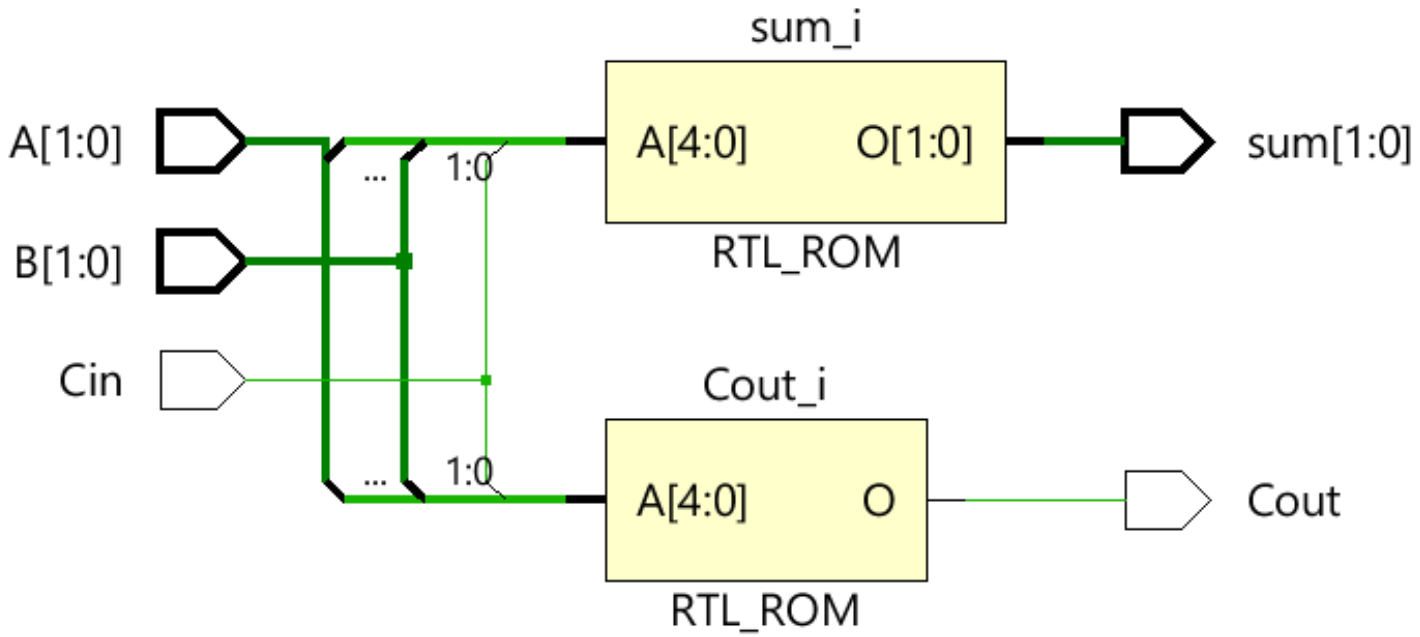

Figure 8: Adder Lookup Table Schematic

It makes sense that a Look up Table is realised as a ROM in hardware because both serve the purpose of storing precomputed values for quick retrieval based on inputs.

3.2 D Flip Flop

```
1  module D_FlipFlop(
2      input wire [1:0] d,        // Data input
3      input wire clk,      // Clock input
4      input wire reset,    // Reset input
5      output reg [1:0] q    // Output
6  );
7
8      // D flip-flop logic
9      always @(posedge clk or posedge reset) begin
10         if (reset) begin
11             q <= 2'b00;  // Reset value
12         end else begin
13             q <= d;      // Update q with input d on clock edge
14         end
15     end
16
17 endmodule
```

3.2.1 Schematic Explanation

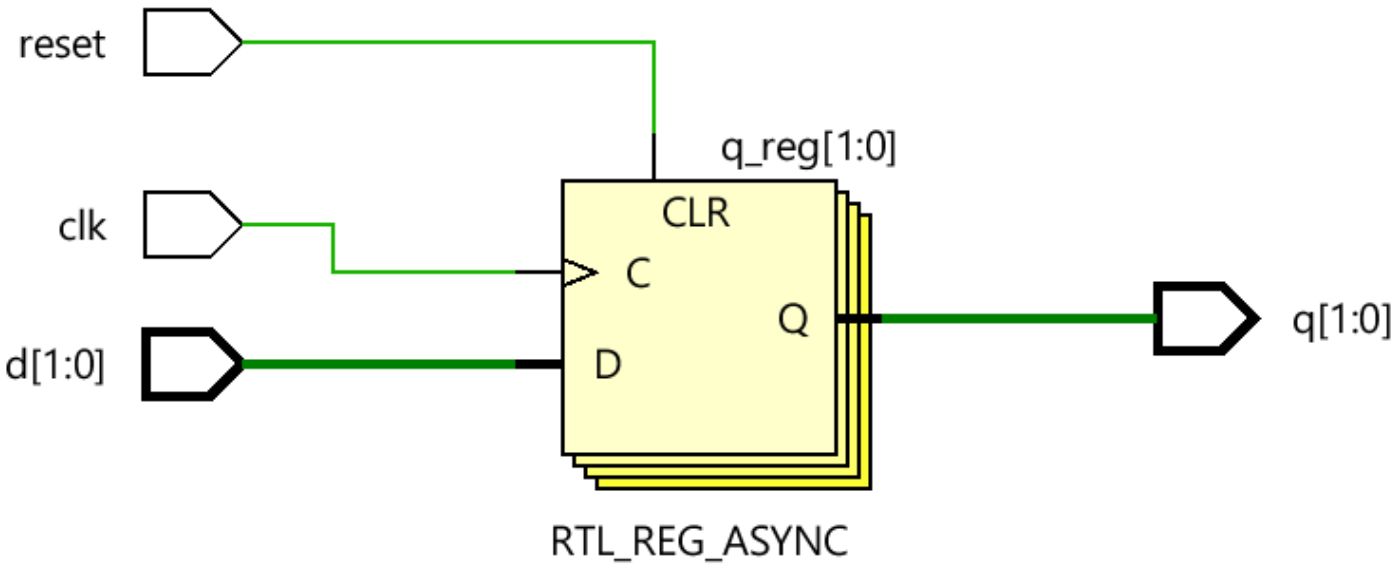

Figure 9: DFF Schematic

In the schematic DFF is realised as an RTL register which makes sense.

3.3 CLB

```
1  module CLB(
2      input wire clk,
3      input wire reset,
4      input wire [1:0] in1,
5      input wire [1:0] in2,
6      input wire in3,
7      input wire mux_in,
8      output reg [1:0] out1_final,
9      output reg out2_final
10 );
11
12 wire [1:0] out1, out1_delayed;
13 wire out2, out2_delayed;
14
15 // Instance of a two-bit full adder
16 two_bit_full_adder adder_inst1 (
17     .A(in1),         // Input 1
18     .B(in2),         // Input 2
19     .Cin(in3),         // Carry in
20     .sum(out1),         // Sum output
21     .Cout(out2)        // Carry output
22      );
23
24 D_FlipFlop dff_inst1(
25     .d(out1),
26     .clk(clk),
27     .reset(reset),
28     .q(out1_delayed)
29     );
30
31 D_FlipFlop dff_inst2(
```

```
32      .d(out2),
33      .clk(clk),
34      .reset(reset),
35      .q(out2_delayed)
36      );
37
38  always @* begin
39      case (mux_in)
40          1'b0: begin
41              out1_final = out1;
42              out2_final = out2;
43              end
44          1'b1: begin
45              out1_final = out1_delayed;
46              out2_final = out2_delayed;
47              end
48          default: begin
49              out1_final = out1;
50              out2_final = out2;
51              end
52          endcase
53
```

3.3.1 Schematic Explanation

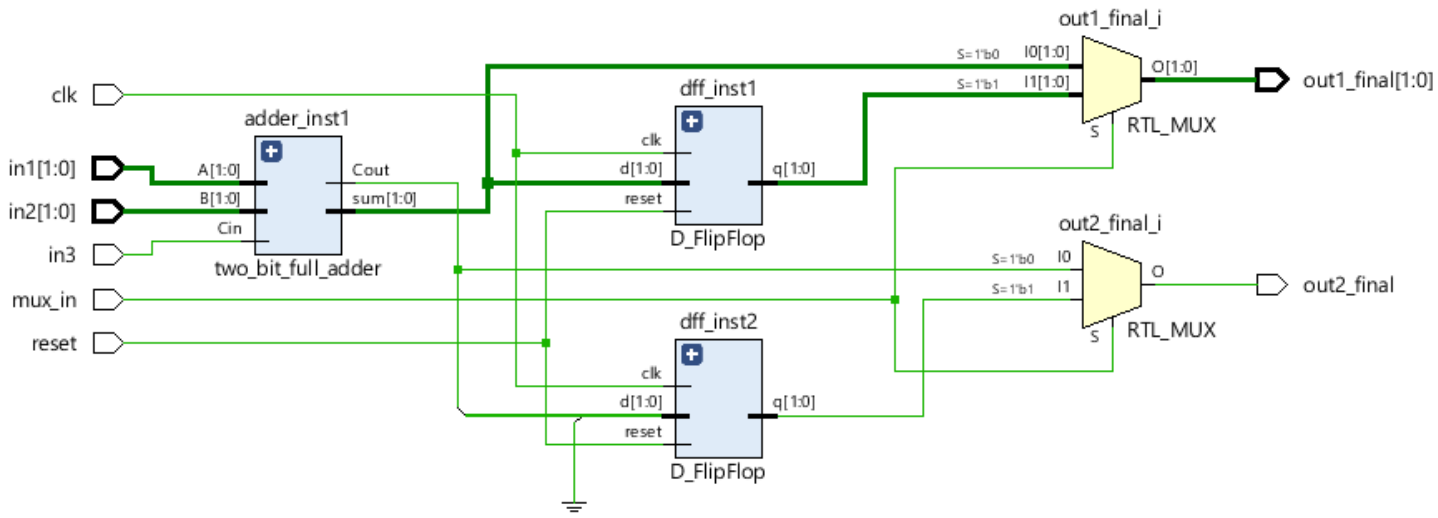

Figure 10: CLB Schematic

This is realised as anticipated in the architecture, contianing the module for Adder, D flip flop and mux to decide between delayed or non-delayed output.

3.4 Controller

```
1  module Controller(
2      input wire [7:0] bitfile,
3      output reg [1:0] Control_signal1,
4      output reg [1:0] Control_signal2,
5      output reg [1:0] Control_signal3,
6      output reg [1:0] Control_signal4
7      );
```

```
8
9       always @* begin
10          // Slice the bitfile into 2-bit segments for each control signal
11          Control_signal1 = bitfile[1:0];
12          Control_signal2 = bitfile[3:2];
13          Control_signal3 = bitfile[5:4];
14          Control_signal4 = bitfile[7:6];
15      end
16
17  endmodule
```

3.4.1 Schematic Explanation

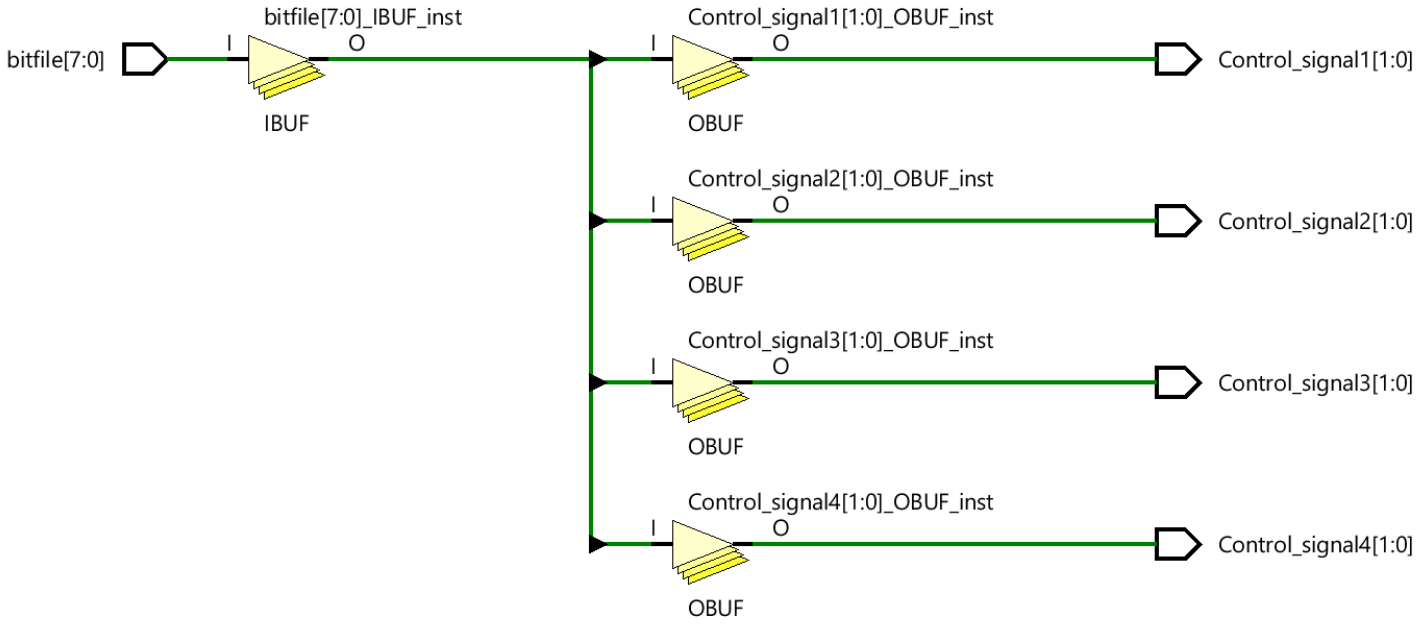

Figure 11: Input 1 Routing Schematic

It can be observed that the job of the controller is simply to divide the bitfile into equal segments of 2 bits each. It makes sense that this piece of code is realised as buffers.

3.5 Config_FPGA

In the following Verilog code, While I have routed the carry and the inputs, I have not implemented Sum Routing. Instead I have assumed for now that Sum from CLB1 is the first Sum. This is because I realised that within 8 bits of bitfile even if I had implemented it, it would have been hardcoding. Discussed it with my peers in class and even those who had implemented Sum routing were essentially hardcoding the sequence. I have, however, added an architecture to this report on how sum routing could have worked.

Cout_final is the Carry from the last CLB. In some cases in the case statements, it is not getting updated. In such a case statement I have latched onto the previous value of Cout. Naturally, that is getting realised as a Latch in hardware as well. Well, I am unsure if with regards to hardware a latch can exist in the routing channel.

```
1  `timescale 1ns / 1ps
2
3  module config_fpga(
4      input wire [7:0] In_1,
5      input wire [7:0] In_2,
6      input wire [1:0] Sum_CLB1,
7      input wire [1:0] Sum_CLB2,
```

```
8       input wire [1:0] Sum_CLB3,
9       input wire [1:0] Sum_CLB4,
10      input wire Cout_CLB1,        // carry coming from CLB1
11      input wire Cout_CLB2,        // carry coming from CLB2
12      input wire Cout_CLB3,        // carry coming from CLB3
13      input wire Cout_CLB4,        // carry coming from CLB4
14      input wire C_external,       // carry that is not coming from any CLB
15      input wire [1:0] sel1,       // control signal from mux of CLB1
16      input [1:0] sel2,       // control signal from mux of CLB2
17      input [1:0] sel3,       // control signal from mux of CLB3
18      input [1:0] sel4,       // control signal from mux of CLB4
19      output [7:0] Sum_final,        // concatenated sum
20      output reg Cin_CLB1,        // Cin to CLB1
21      output reg Cin_CLB2,        // Cin to CLB2
22      output reg Cin_CLB3,        // Cin to CLB3
23      output reg Cin_CLB4,        // Cin to CLB4
24      output reg Cout_final,
25      output reg [1:0] In_11,  // first input to CLB1
26      output reg [1:0] In_12,  // first input to CLB2
27      output reg [1:0] In_13,  // first input to CLB3
28      output reg [1:0] In_14,  // first input to CLB4
29      output reg [1:0] In_21,  // second input to CLB1
30      output reg [1:0] In_22,  // second input to CLB2
31      output reg [1:0] In_23,  // second input to CLB3
32      output reg [1:0] In_24   // second input to CLB4
33      );
34
35      wire [1:0] num1_1, num1_2, num1_3, num1_4, num2_1, num2_2, num2_3, num2_4;
36          // Input 1
37          assign num1_1 = In_1[1:0];   // first slice
38          assign num1_2 = In_1[3:2];   // second slice
39          assign num1_3 = In_1[5:4];   // third slice
40          assign num1_4 = In_1[7:6];   // fourth slice
41          // Input 2
42          assign num2_1 = In_2[1:0];   // first slice
43          assign num2_2 = In_2[3:2];   // second slice
44          assign num2_3 = In_2[5:4];   // third slice
45          assign num2_4 = In_2[7:6];   // fourth slice
46
47  //    case
48  always @ * begin
49      // first CLB
50      case (sel1)
51          // This should make three muxes for each of the inputs to CLB
52          2'b00 : begin
53                  Cin_CLB1 <= C_external;
54                  In_11 <= num1_1; In_21 <= num2_1;
55                  Cout_final <= Cout_final;
56                  end  // This is the first CLB(A)
57
58          2'b01 : begin
59                  Cin_CLB1 <= Cout_CLB2;
60                  In_11 <= num1_2;
61                  In_21 <= num2_2;
62                  Cout_final <= Cout_final; // CLB2 --> CLB1
63                  end
64          2'b10 : begin
65                  Cin_CLB1 <= Cout_CLB3;
66                  In_11 <= num1_3;
67                  In_21 <= num2_3;
68                  Cout_final <= Cout_final;  // x --> CLB3 --> CLB1
69                  end
70          2'b11 : begin
```

```
71                Cin_CLB1 <= Cout_CLB4;
72                In_11 <= num1_4;
73                In_21 <= num2_4;
74                Cout_final <= Cout_CLB1;  // x --> x --> CLB4 --> CLB1
75                end
76            default: begin Cin_CLB1 <= Cin_CLB1; In_11 <= In_11; In_21 <= In_22; Cout_final <=
      Cout_final; end
77        endcase
78
79        // second CLB
80        case (sel2)
81            2'b00 : begin
82                    Cin_CLB2 <= C_external;
83                    In_12 <= num1_1;
84                    In_22 <= num2_1;
85                    Cout_final <= Cout_final; // This is the first CLB(A)
86                    end
87            2'b01 : begin
88                    Cin_CLB2 <= Cout_CLB1;
89                    In_12 <= num1_2;
90                    In_22 <= num2_2;
91                    Cout_final <= Cout_final;  // CLB1 --> CLB2
92                    end
93            2'b10 : begin
94                    Cin_CLB2 <= Cout_CLB3;
95                    In_12 <= num1_3;
96                    In_22 <= num2_3;
97                    Cout_final <= Cout_final;  // x --> CLB3 --> CLB2
98                    end
99            2'b11 : begin
100                   Cin_CLB2 <= Cout_CLB4;
101                   In_12 <= num1_4;
102                   In_22 <= num2_4;
103                   Cout_final <= Cout_CLB2;  // x --> x --> CLB4 --> CLB2
104                   end
105           default: begin Cin_CLB2 <= Cin_CLB2;In_12 <= In_12; In_22 <= In_22; Cout_final <=
      Cout_final; end
106       endcase
107
108       //
109       case (sel3)
110           2'b00 : begin
111                   Cin_CLB3 <= C_external;
112                   In_13 <= num1_1;
113                   In_23 <= num2_1;
114                   Cout_final <= Cout_final; // This is the first CLB(A)
115                   end
116           2'b01 : begin
117                   Cin_CLB3 <= Cout_CLB1;
118                   In_13 <= num1_2;
119                   In_23 <= num2_2;
120                   Cout_final <= Cout_final; // CLB1 --> CLB3
121                   end
122           2'b10 : begin
123                   Cin_CLB3 <= Cout_CLB2;
124                   In_13 <= num1_3;
125                   In_23 <= num2_3;
126                   Cout_final <= Cout_final; // x --> CLB2 --> CLB3
127                   end
128           2'b11 : begin
129                   Cin_CLB3 <= Cout_CLB4;
130                   In_13 <= num1_4;
131                   In_23 <= num2_4;
```

```
132                 Cout_final <= Cout_CLB3;  // x --> x --> CLB4 --> CLB3
133                 end
134         default: begin Cin_CLB3 <= Cin_CLB3; In_13 <= In_13; In_23 <= In_23; Cout_final <=
    Cout_final; end
135     endcase
136
137     case (sel4)
138         2'b00 : begin
139                 Cin_CLB4 <= C_external;
140                 In_14 <= num1_1;
141                 In_24 <= num2_1;
142                 Cout_final <= Cout_final; // This is the first CLB(A)
143                 end
144         2'b01 : begin
145                 Cin_CLB4 <= Cout_CLB1;
146                 In_14 <= num1_2;
147                 In_24 <= num2_2;
148                 Cout_final <= Cout_final; // CLB1 --> CLB4
149                 end
150         2'b10 : begin
151                 Cin_CLB4 <= Cout_CLB2;
152                 In_14 <= num1_3;
153                 In_24 <= num2_3;
154                 Cout_final <= Cout_final;  // x --> CLB2 --> CLB4
155                 end
156         2'b11 : begin
157                 Cin_CLB4 <= Cout_CLB3;
158                 In_14 <= num1_4;
159                 In_24 <= num2_4;
160                 Cout_final <= Cout_CLB4;  // x --> x --> CLB4 --> CLB4
161                 end
162         default: begin Cin_CLB4 <= Cin_CLB4; In_14 <= In_14; In_24 <= In_24; Cout_final <=
    Cout_final; end
163     endcase
164 end
165
166 assign Sum_final = {Sum_CLB4, Sum_CLB3, Sum_CLB2, Sum_CLB1};
167 endmodule
```

3.5.1 Schematic Explanation

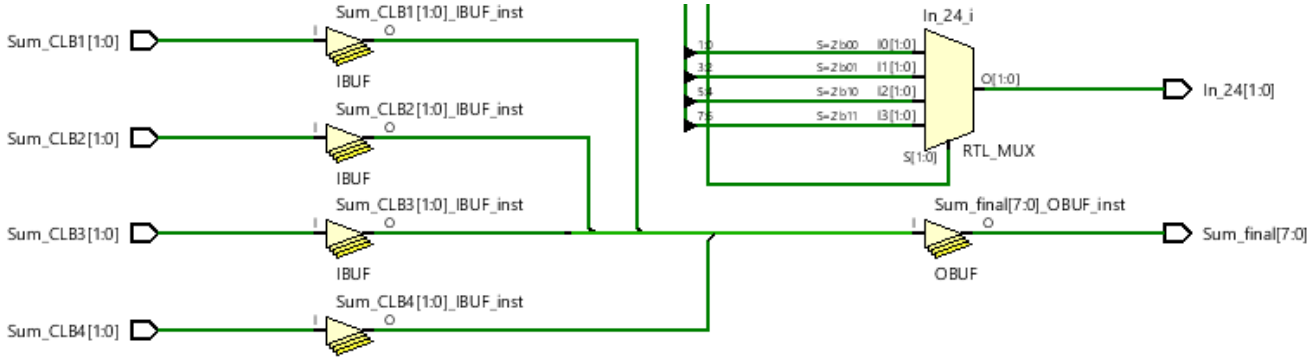

Figure 12: Concatenation of Sum within Routing Channel Schematic

This figure clearly shows that the Sum form each of the CLBs is getting concatenated to give the final value of sum. As mentioned earlier, I have not implemented Sum Routing. Instead I have assumed for now that Sum from CLB1 is the first Sum. This is because I realised that within 8 bits of bitfile even if I had implemented it, it would have been hardcoding even then.

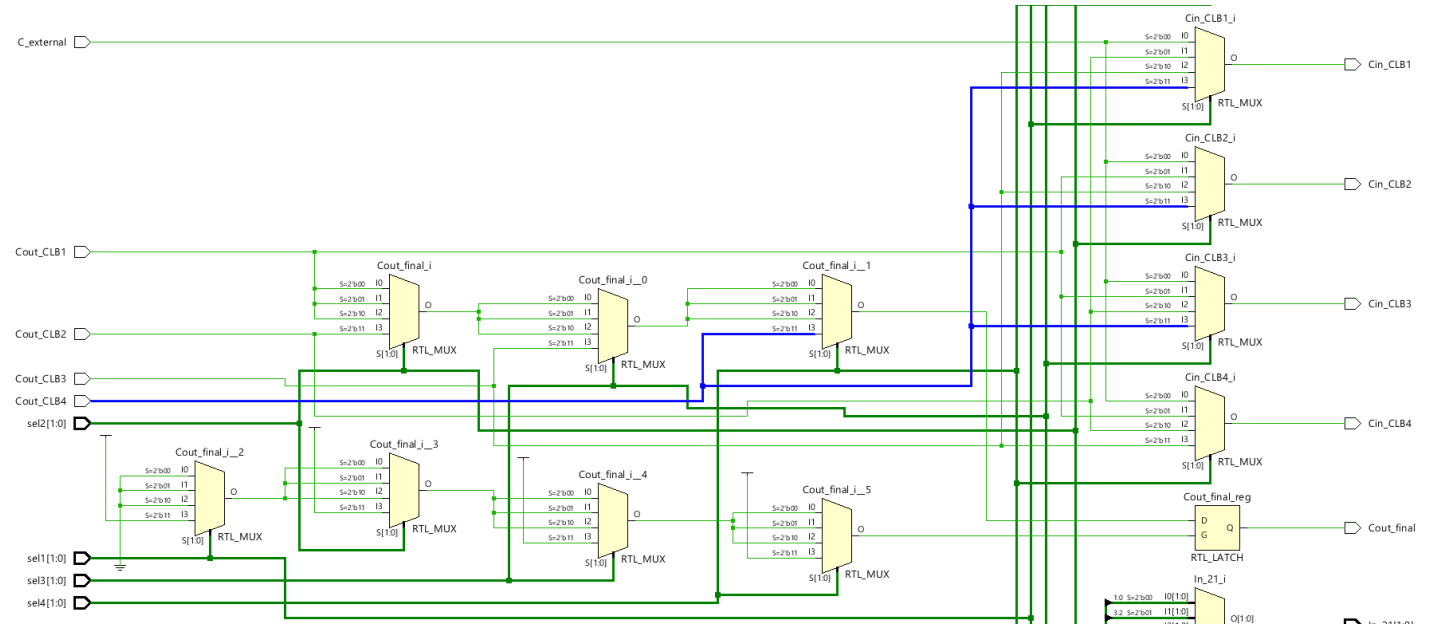

Figure 13: Carry Routing

It can be seen in the diagram that a latch is being realised for the final value of Carry out and I have explained why earlier. Apart from that, for each select line(control signal) it can be decided what what path is followed by the value of Cin. Each select line to the mux corresponds to a control for a particular CLB: Sel1 for CLB1, Sel2 for CLB2 and so on. A 00 value of sel means that Addition starts from this particular CLB and it should receive the external C i.e. initial Carry which is always 0. Similarly, the values of sel decide the route. This is done in order to make the FPGA configurable using bitfile.

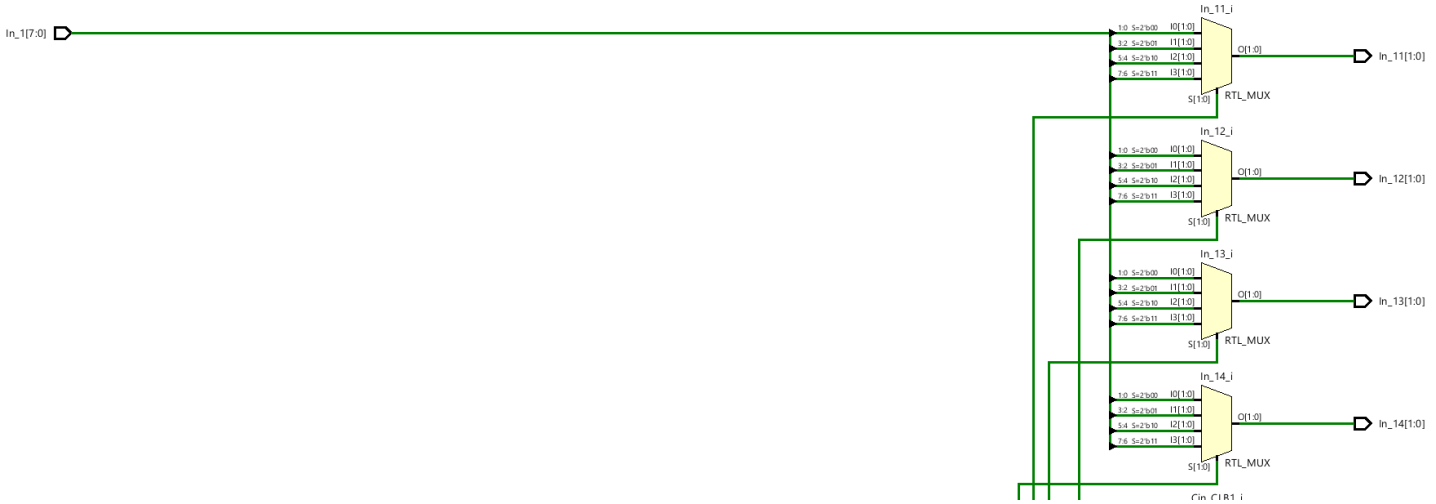

Figure 14: Input 1 Routing Schematic

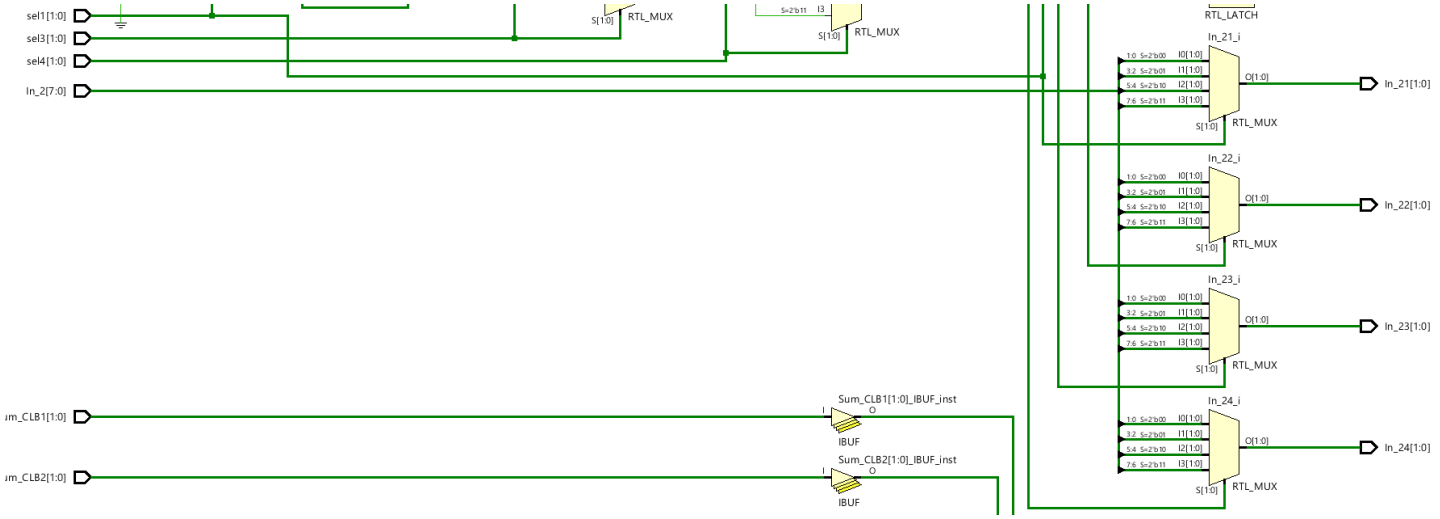

Figure 15: Enter Caption

The input is split into four equal segments, allocating two bits for each CLB. Muxes are employed to choose the input for each CLB based on the bitfile. Although it may not be explicitly depicted in the schematic, the select lines used for carry are also used in the muxes to designate the input.

To wrap up, it is important to emphasize that this routing scheme is still preliminary and can certainly benefit from refinement.

3.6 FPGA

```
1  `timescale 1ns / 1ps
2
3
4  module FPGA_new(
5      input wire clk,
6      input wire reset,
7      input wire [7:0] In_1,
8      input wire [7:0] In_2,
9      input wire [7:0] bitfile_input,
10     output wire [7:0] Sum,
11     output wire Cout
12     );
13
14     wire [1:0] Control_signal1, Control_signal2, Control_signal3, Control_signal4;
15     wire [1:0] In_11, In_12, In_13, In_14, In_21, In_22, In_23, In_24;
16     wire [1:0] sel1, sel2, sel3, sel4;
17     wire [1:0] out11_final, out12_final, out13_final, out14_final;
18     wire out21_final, out22_final, out23_final, out24_final;
19     wire Cin_CLB1, Cin_CLB2, Cin_CLB3, Cin_CLB4;
20
21
22     Controller controller_inst1(
23         .bitfile(bitfile_input),
24         .Control_signal1(sel1),
25         .Control_signal2(sel2),
26         .Control_signal3(sel3),
27         .Control_signal4(sel4)
28         );
29
30     CLB clb_inst1(
31         .clk(clk),
32         .reset(reset),
33         .in1(In_11),
```

```
34          .in2(In_21),
35          .in3(Cin_CLB1),
36          .mux_in(1'b0),                      // Not delayed
37          .out1_final(out11_final),
38          .out2_final(out21_final)
39      );
40
41      CLB clb_inst2(
42          .clk(clk),
43          .reset(reset),
44          .in1(In_12),
45          .in2(In_22),
46          .in3(Cin_CLB2),
47          .mux_in(1'b0),                       // Not delayed
48          .out1_final(out12_final),
49          .out2_final(out22_final)
50      );
51
52      CLB clb_inst3(
53          .clk(clk),
54          .reset(reset),
55          .in1(In_13),
56          .in2(In_23),
57          .in3(Cin_CLB3),
58          .mux_in(1'b0),                          // Not delayed
59          .out1_final(out13_final),
60          .out2_final(out23_final)
61      );
62
63      CLB clb_inst4(
64          .clk(clk),
65          .reset(reset),
66          .in1(In_14),
67          .in2(In_24),
68          .in3(Cin_CLB4),
69          .mux_in(1'b0),                  // Not delayed
70          .out1_final(out14_final),
71          .out2_final(out24_final)
72      );
73
74      // Instantiate the module
75      config_fpga config_inst1(
76          .In_1(In_1),
77          .In_2(In_2),
78          .Sum_CLB1(out11_final),
79          .Sum_CLB2(out12_final),
80          .Sum_CLB3(out13_final),
81          .Sum_CLB4(out14_final),
82          .Cout_CLB1(out21_final),
83          .Cout_CLB2(out22_final),
84          .Cout_CLB3(out23_final),
85          .Cout_CLB4(out24_final),
86          .C_external(1'b0),
87          .sel1(sel1),
88          .sel2(sel2),
89          .sel3(sel3),
90          .sel4(sel4),
91          .Sum_final(Sum),
92          .Cin_CLB1(Cin_CLB1),
93          .Cin_CLB2(Cin_CLB2),
94          .Cin_CLB3(Cin_CLB3),
95          .Cin_CLB4(Cin_CLB4),
96          .Cout_final(Cout),
```

```
97          .In_11(In_11),
98          .In_12(In_12),
99          .In_13(In_13),
100         .In_14(In_14),
101         .In_21(In_21),
102         .In_22(In_22),
103         .In_23(In_23),
104         .In_24(In_24)
105     );
106
107
108 endmodule
```

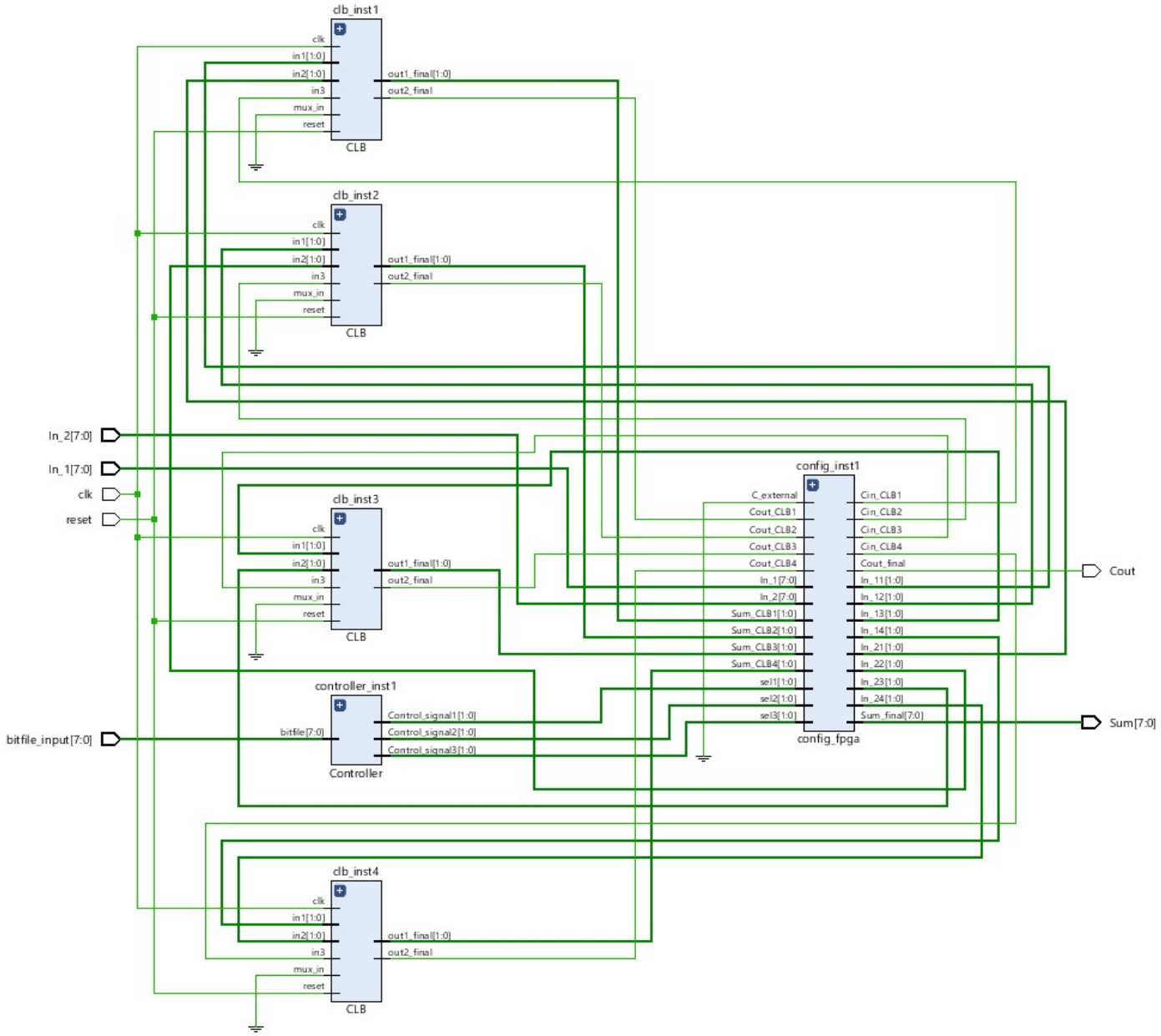

Figure 16: FPGA overall schematic

4 Analysis

Simulation 1: Non-Delayed outputs

In this test case, I have given the sel input to muxes in CLBs as $1'b0$. Therefore, the output is obtained in the same clock cycle without any delay. For sum, I have taken the following inputs and the table below shows there expected output. The simulation outputs below are also same as that of the expected outputs.

Test Case	`In_1` (binary)	`In_2` (binary)	`Sum` (binary)	`COut` (binary)
1	00001111	01010101	01100100	0
2	00101111	11011101	00001100	1
3	10101010	01010101	11111111	0
4	11110000	00001111	11111111	0

Table 1: Binary Sum and Carry Out for Each Test Case

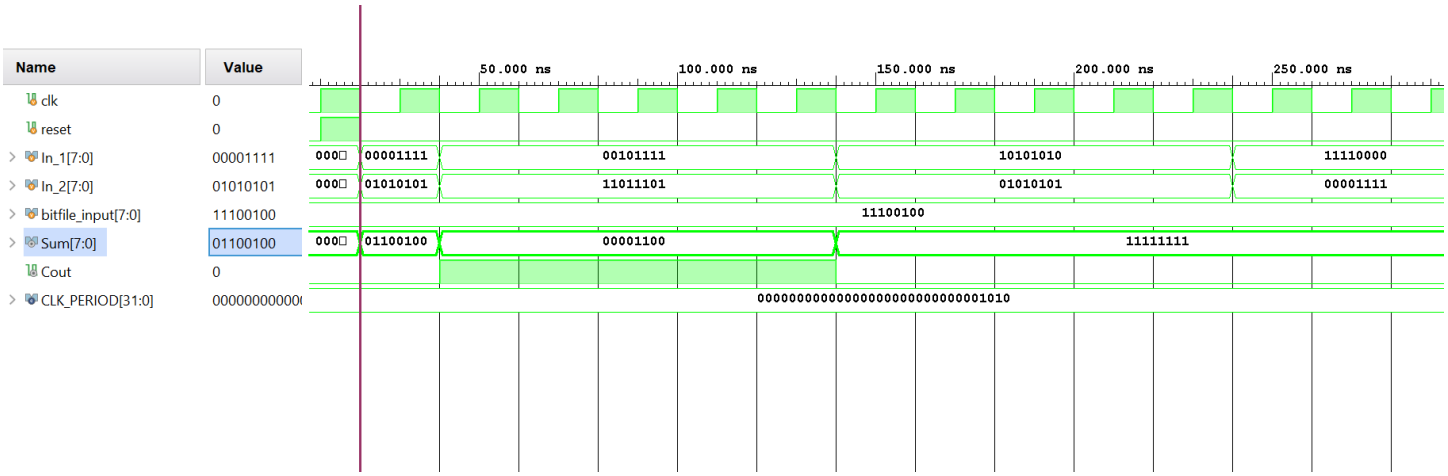

Figure 17: Simulation 1

Simulation 2: Delayed outputs

Now output from each CLB is delayed as it passes through a DFF. As can be observed, it takes multiple clock cycles for computing some combinations while for some it takes a single clock cycle. I have used the following Testcases.

Test Case	`In_1` (binary)	`In_2` (binary)	`Sum` (binary)	`COut` (binary)
1	10101010	01010101	11111111	0
2	11111111	00000001	00000000	1

Table 2: Binary Sum and Carry Out for Each Test Case

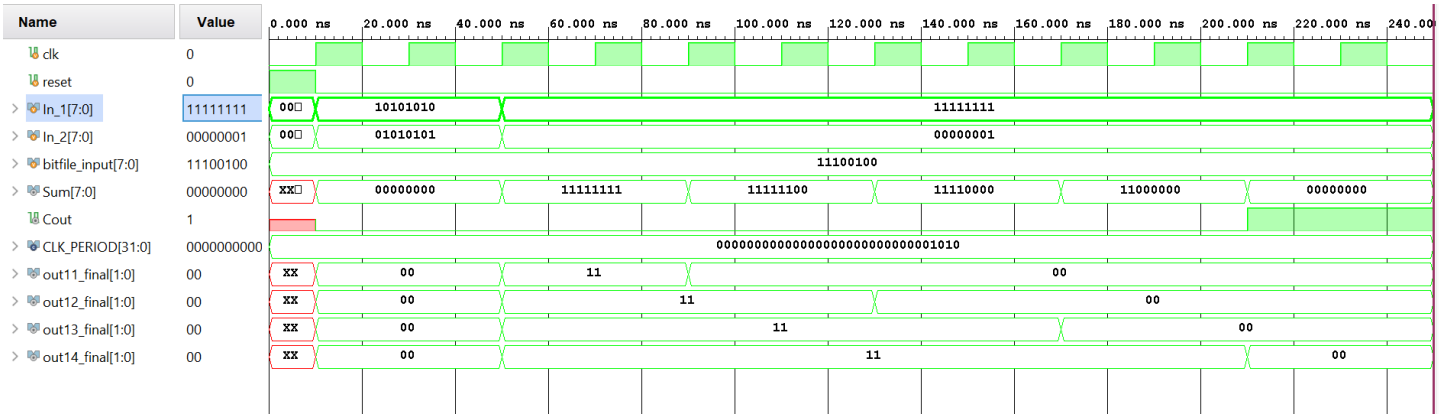

Figure 18: Simulation 2