CE-325: Digital System Design
Quiz 02 - FPGA

Huzaifah Tariq Ahmed - ha07151

6th May 2024

1 System Architecture

1.1 Configurable Logic Block (CLB)

CLB's role in the architecture is to receive two 2-bit inputs and a 1-bit carry in and compute addition of these, and provide 2-bit sum output and a 1-bit carry out. It has clock and reset signals also going into it. It needs clock as it does the addition on the positive edge of the clock. The addition happens in two stages, as it goes through two 1-bit adders. We then have two D-FFs one for storing the carry out and the other for storing the 2-bit sum output.

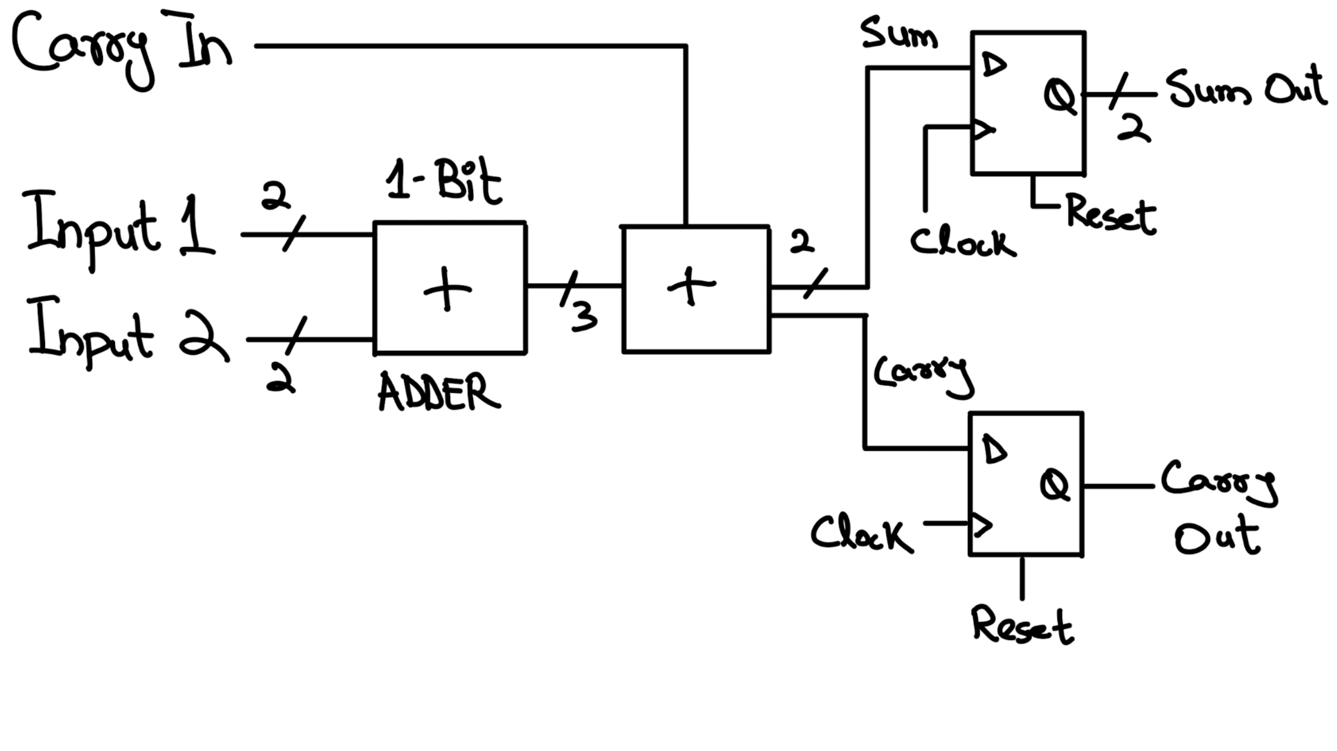

Figure 1: CLB Proposed Architecture

1.2 Routing Controller

Routing controller's role is to simply split the 8-bit input bit-file into four 2-bit numbers, generating four control signals which will control the routing for the four CLBs. We have one control signal for each CLB.

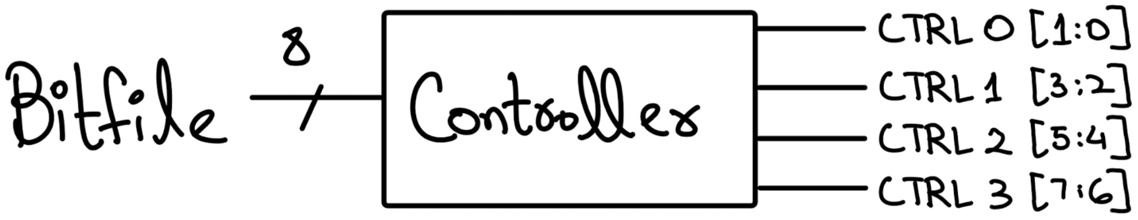

Figure 2: Controller Proposed Architecture

1.3 Input and Carry Routing

In our architecture the routing module's role is to do input routing, carry in routing and CLB sum routing. In the architecture below we can see the carry and input routing done inside the routing module. The diagram shows this routing for only one CLB, however this logic will be replicated for the remaining three CLBs as well. Here we can see that the routing block splits the 8 bit inputs into four 2-bit numbers and then provides them as input to the MUX in every CLB. Here in this MUX the right input for the CLB is decided based on the control signal generated by the controller, based on the bitfile configuration. This is done for both the inputs of a CLB. We also have a third MUX here for deciding the carry in signal for this CLB. This decision is made between the carry out values of the other three CLBs and an external carry in signal given at the start. Again the same control signal will be acting as the select for this MUX. The selected inputs and carry in then go into the CLB, which generates sum and carry outputs.

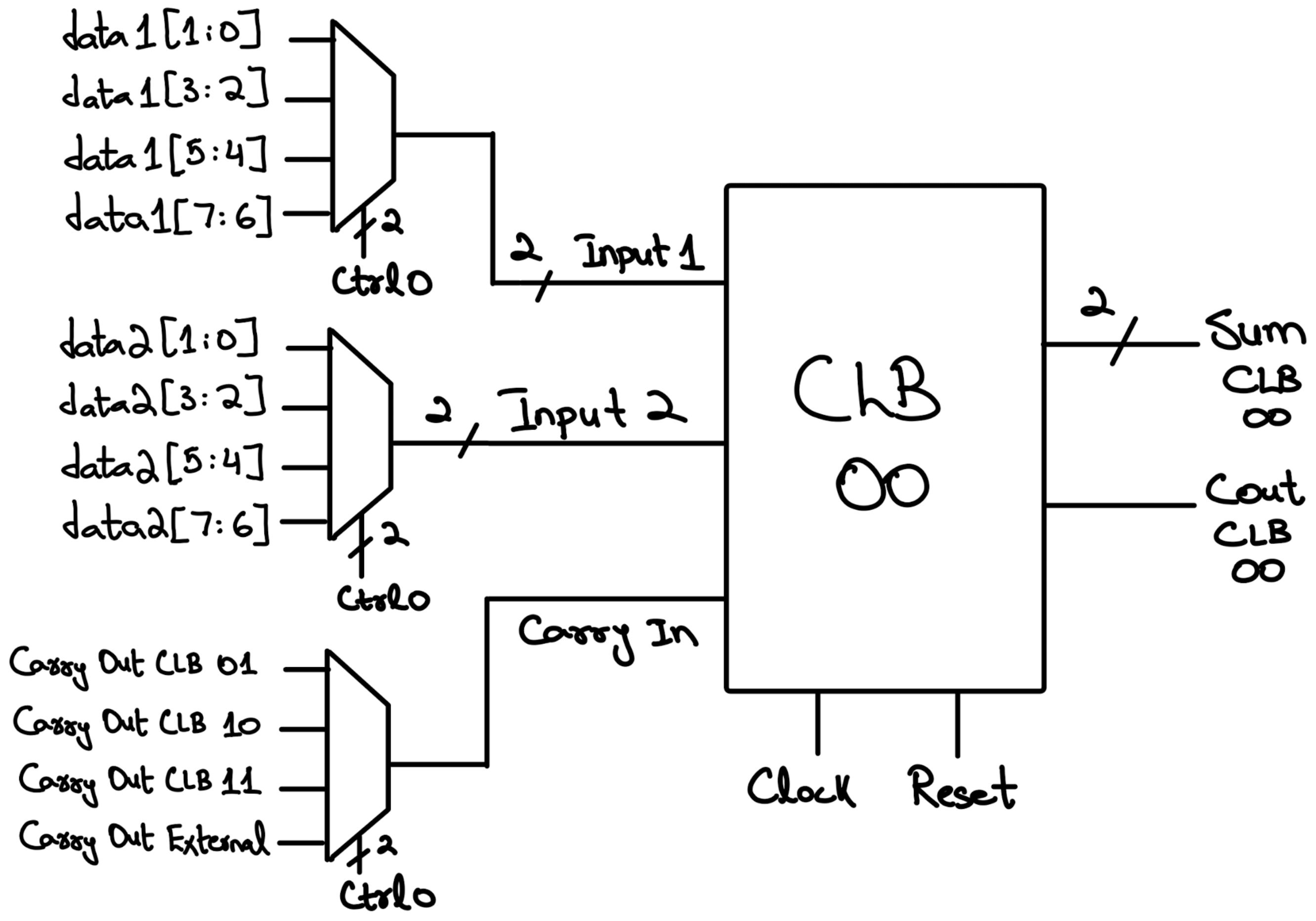

Figure 3: CLB Routing Proposed Architecture

1.4 CLB Sum Output Routing

As discussed above the routing block also routes the output sums of CLBs. It is basically a feed through port, but in my architecture I have given this configurable option to decide the output port through which this sum moves out of the routing block. Again it utilizes a MUX and the same control signal to decide this.

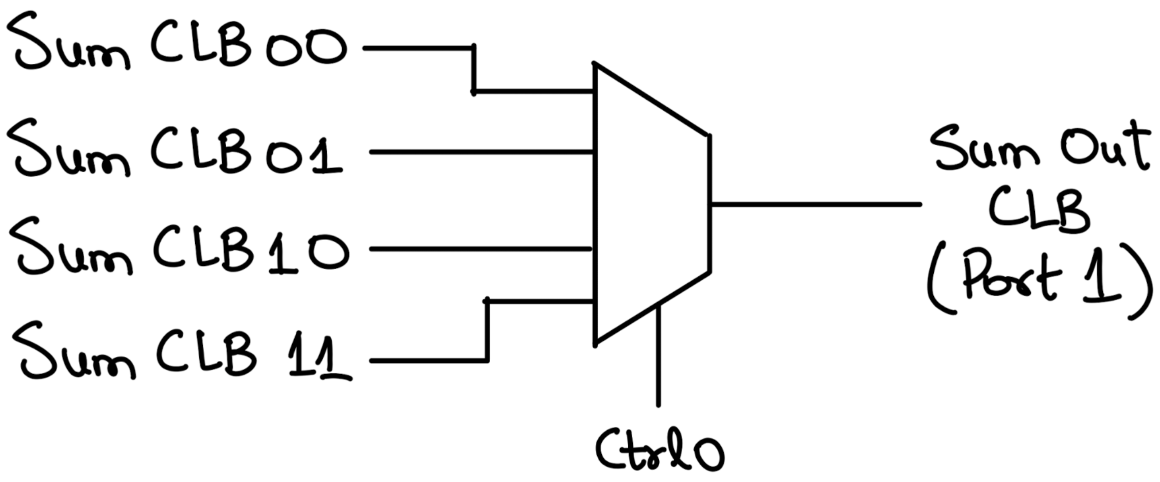

Figure 4: CLB Sum Routing Proposed Architecture

2 FPGA Design In Verilog

2.1 Configurable Logic Block (CLB)

2.1.1 Code

```verilog
`timescale 1ns/1ps

module CLB(num_1,
           num_2,
           carry_in,
           clk,
           reset,
           sum,
           carry_out);

  input [1:0] num_1, // Input for first 2-bit number
              num_2; // Input for second 2-bit number

  input carry_in, // Input carry bit
        clk,      // Clock input
        reset;    // Reset input

  output reg [1:0] sum; // Output for sum of numbers

  output reg carry_out; // Output for carry-out bit

  // Define the always block for sequential logic
  always @(posedge clk or posedge reset)
    if (reset)
```

```
        begin
          // Reset condition: set sum and carry-out to default values
          sum <= 2'b00000;    // Initialize sum to zero
          carry_out <= 1'b0;  // Initialize carry-out to zero
        end
    else
        // Adder logic: sum and carry-out updated based on inputs and
        //previous carry
        {carry_out, sum} <= num_1 + num_2 + carry_in; // Calculate sum and
        //carry-out

endmodule
```

2.1.2 Schematic

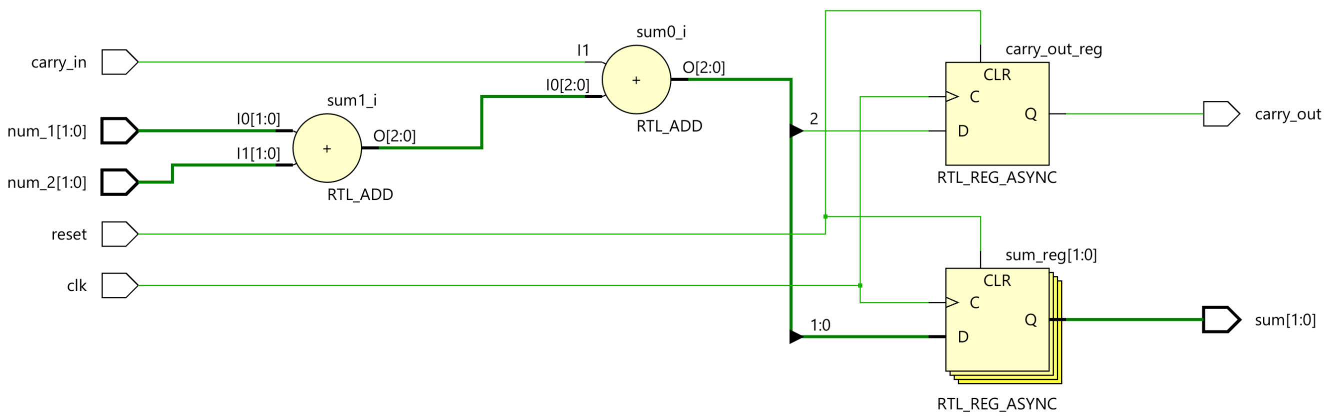

Figure 5: Configurable Logic Block (CLB) Schematic

We can see from this schematic that it basically realizes the proposed architecture of a CLB. It also has two 1-bit adders and two D-FFs for storing the sum and carry outputs. Like our proposed architecture this also gives the D-FFs clock and reset signals.

2.2 Routing Block

2.2.1 Code

```
`timescale 1ns / 1ps

module routing(bitfile,
               cout_00,
               cout_01,
               cout_10,
                cout_11,
                cout_ext,
                data_1,
                data_2,
                sin_00,
                sin_01,
                sin_10,
                sin_11,
                cin_00,
                cin_01,
                cin_10,
                cin_11,
                num_1_00,
                num_2_00,
                num_1_01,
                num_2_01,
                num_1_10,
                num_2_10,
                num_1_11,
                num_2_11,
                sout_00,
                sout_01,
                sout_10,
                sout_11,
                final_cout);

    input [7:0] bitfile; //8 bit bitfile for generating control signals

    input cout_00,
          cout_01,
          cout_10,
          cout_11,
          cout_ext; //carry out from all clbs

    input [7:0] data_1,
                data_2; //initial data input

    input [1:0] sin_00,
                sin_01,
                sin_10,
                sin_11; // sum outputs of clbs

    output reg cin_00,
               cin_01,
               cin_10,
               cin_11; //carry in for each clb

    output reg [1:0] num_1_00,
                     num_2_00,
                     num_1_01,
                    num_2_01,
                    num_1_10,
                    num_2_10,
                    num_1_11,
                    num_2_11; //data inputs for all clbs

    output reg [1:0] sout_00,
                    sout_01,
                    sout_10,
                    sout_11; //sum of all clbs routed out

    output reg final_cout; //final carry out

    // Splitting the 10-bit input into five 2-bit control signals
    wire [1:0] ctrl_0 = bitfile[1:0];  //clb00
    wire [1:0] ctrl_1 = bitfile[3:2];  //clb01
    wire [1:0] ctrl_2 = bitfile[5:4];  //clb10
    wire [1:0] ctrl_3 = bitfile[7:6];  //clb11

    // Control logic for each CLB based on the bitfile sections
    always@(*)
    begin
      case(ctrl_0)
            2'b00 : begin
                    cin_00 <= cout_ext;
                    num_1_00 <= data_1[1:0];
                    num_2_00 <= data_2[1:0];
                    sout_00 <= sin_00;
                    end
            2'b01 : begin
                    cin_00 <= cout_01;
                    num_1_00 <= data_1[3:2];
                    num_2_00 <= data_2[3:2];
                    sout_00 <= sin_01;
                    end
            2'b10 : begin
                    cin_00 <= cout_10;
                    num_1_00 <= data_1[5:4];
                    num_2_00 <= data_2[5:4];
                    sout_00 <= sin_10;
                    end
            2'b11 : begin
                    cin_00 <= cout_11;
                    num_1_00 <= data_1[7:6];
                    num_2_00 <= data_2[7:6];
                    sout_00 <= sin_11;
                    final_cout <= cout_00;
                    end
        endcase

        case(ctrl_1)
            2'b00 : begin
                    cin_01 <= cout_00;
                    num_1_01 <= data_1[3:2];
                    num_2_01 <= data_2[3:2];
                    sout_01 <= sin_01;
                    end
            2'b01 : begin
                    cin_01 <= cout_ext;
                    num_1_01 <= data_1[1:0];
                    num_2_01 <= data_2[1:0];
                    sout_01 <= sin_00;
                    end
            2'b10 : begin
                    cin_01 <= cout_10;
                    num_1_01 <= data_1[5:4];
                    num_2_01 <= data_2[5:4];
                    sout_01 <= sin_10;
                    end
            2'b11 : begin
                    cin_01 <= cout_11;
                    num_1_01 <= data_1[7:6];
                    num_2_01 <= data_2[7:6];
                    sout_01 <= sin_11;
                    final_cout <= cout_01;
                    end
        endcase

        case(ctrl_2)
            2'b00 : begin
                    cin_10 <= cout_00;
                    num_1_10 <= data_1[3:2];
                    num_2_10 <= data_2[3:2];
                    sout_10 <= sin_01;
                    end
            2'b01 : begin
                    cin_10 <= cout_01;
                    num_1_10 <= data_1[5:4];
                    num_2_10 <= data_2[5:4];
                    sout_10 <= sin_10;
                    end
            2'b10 : begin
                    cin_10 <= cout_ext;
                    num_1_10 <= data_1[1:0];
                    num_2_10 <= data_2[1:0];
                    sout_10 <= sin_00;
                    end
            2'b11 : begin
                    cin_10 <= cout_11;
                    num_1_10 <= data_1[7:6];
                    num_2_10 <= data_2[7:6];
                    sout_10 <= sin_11;
                    final_cout <= cout_10;
                    end
        endcase

        case(ctrl_3)
            2'b00 : begin
                    cin_11 <= cout_00;
                    num_1_11 <= data_1[3:2];
                    num_2_11 <= data_2[3:2];
                    sout_11 <= sin_01;
                    end
            2'b01 : begin
                    cin_11 <= cout_01;
                    num_1_11 <= data_1[5:4];
                    num_2_11 <= data_2[5:4];
                    sout_11 <= sin_10;
                    end
            2'b10 : begin
                    cin_11 <= cout_10;
                    num_1_11 <= data_1[7:6];
                    num_2_11 <= data_2[7:6];
                    sout_11 <= sin_11;
                    final_cout <= cout_11;
                    end
            2'b11 : begin
                    cin_11 <= cout_ext;
                    num_1_11 <= data_1[1:0];
                    num_2_11 <= data_2[1:0];
                    sout_11 <= sin_00;
                    end
        endcase

    end

endmodule
```

2.2.2 Schematic

In the carry routing schematic below we can see that like our architecture this also utilizes MUX for selection of every carry in signal. Although not a separate module for a controller we can see our bitfile being split into 2-bit numbers which are going into all the MUX as control signals. Apart from the four MUX for deciding carry in for each clb, you can see seven additional MUX. They were not in the proposed architecture however I later added them in order for my FPGA to be able to select that out of the four CLBs, which should be selected to give their carry out signal as the over all final carry out. Before I was assuming to always take that carry out from CLB-11, however I think FPGA should be a configurable architecture with nothing assumed before hand.

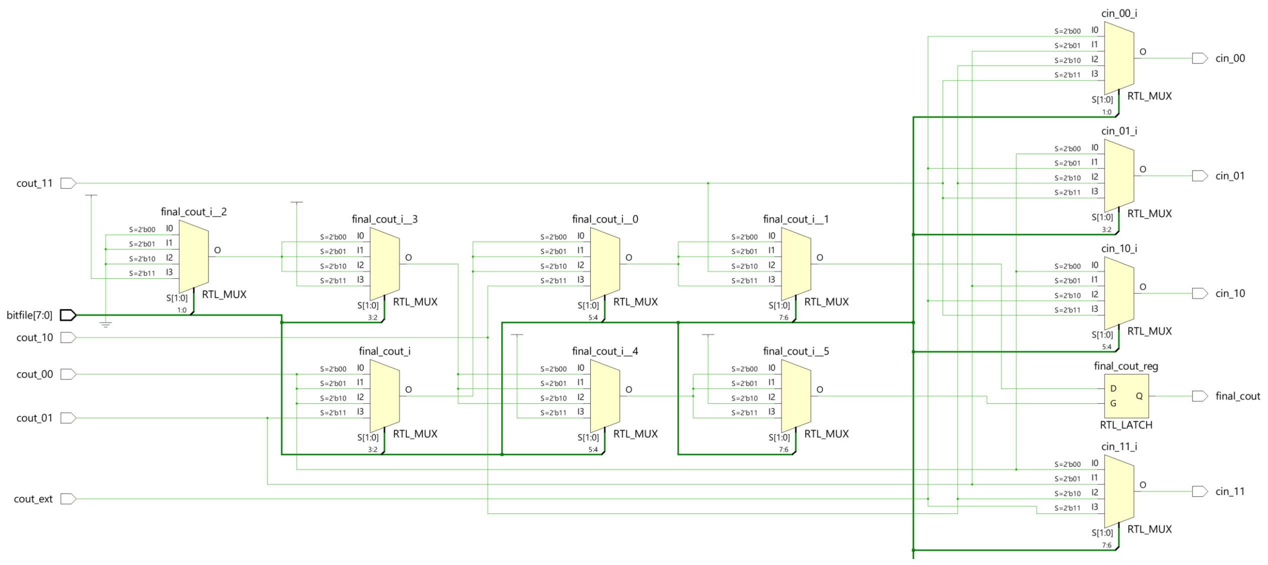

Figure 6: Carry Routing Block Schematic

Looking at the Input Routing Block we can again see the same MUX based architecture proposed earlier. Here again we have 2 MUX per CLB for selection of both inputs of the CLB based on the control signal. All the outputs of these MUXes go into the CLBS as input.

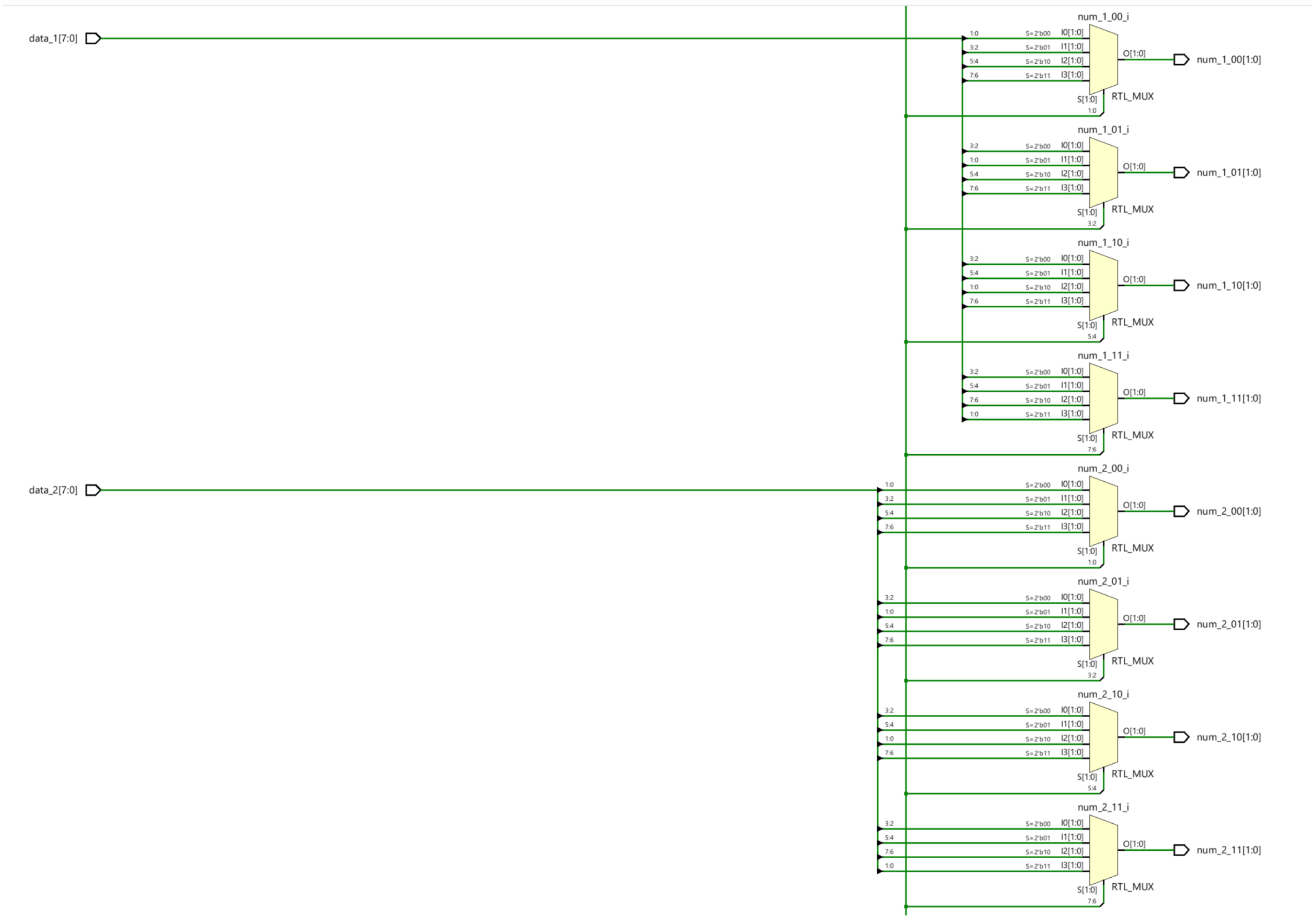

Figure 7: Input Routing Block Schematic

Looking at the Sum routing block we can see that again four MUX are utilized to select which out of the sums of four CLBs should go through a particular port. Again using the same control signal.

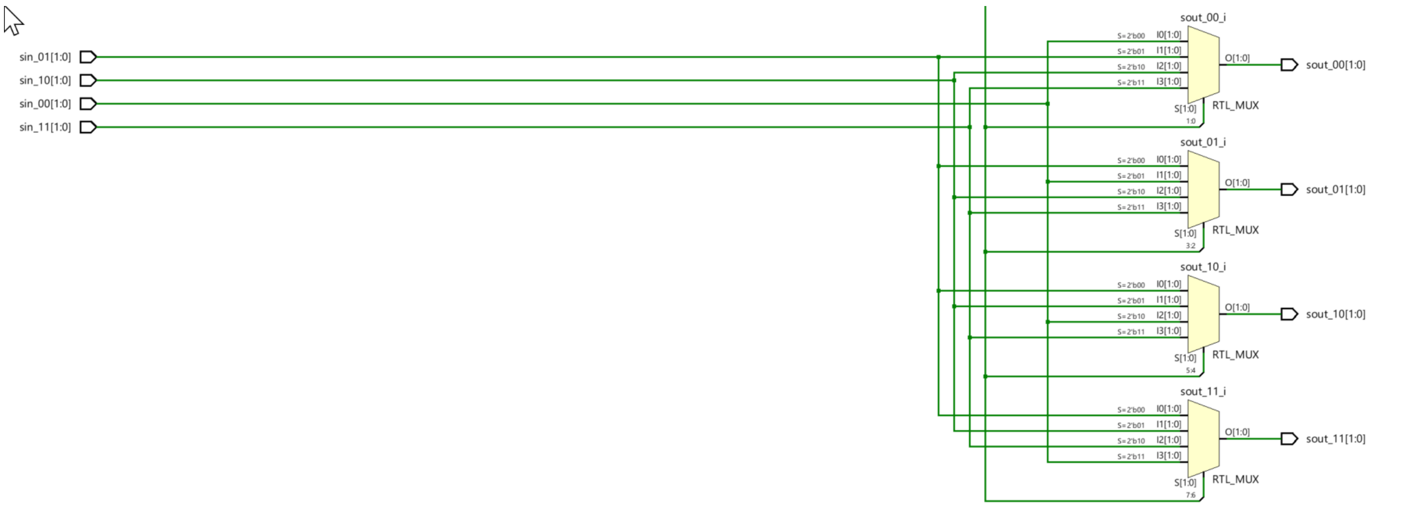

Figure 8: Sum Routing Block Schematic

One thing to note here is that I have configured this in a way that a single control signal control the inputs, carry in as well as output sum routing for that CLB. This is done so that we could make our FPGA configurable. However it was not possible to cater all combinations of sequences of CLBs during addition. Like by default the sequence of addition is from CLB 00 to 01 to 10 and finally 11. We can alter this sequence using the control sequence, however not all sequences are possible using the case logic used in the routing module for this.

2.3 Top Module

2.3.1 Code

```
// Set the time scale for simulation
`timescale 1ns/1ps

// Define the FPGA module with inputs and outputs
module fpga(data_1,
            data_2,
            bitfile,
            carry_in,
            clk,
            reset,
            sum,
            carry_out
           );

    input [7:0] data_1, //8 bit number for addition
                data_2; //8 bit number for addition

    input [7:0] bitfile; //8 bit bitfile for generating control signals in routing

    input carry_in, //initial external carry_in
          clk,      //clock signal
          reset;    //reset signal

    output [7:0] sum;  //final 8 bit sum output
    output carry_out;  //final 1 bit carry out

    // Declare internal wires for carry and sum signals
    wire cout_00,
         cout_01,
         cout_10,
         cout_11,
         cout_ext; //carry out wires for all clbs

    wire cin_00,
         cin_01,
         cin_10,
         cin_11;  //carry in wires for all clbs

```

```verilog
    wire [1:0] sum_clb_00,
               sum_clb_01,
               sum_clb_10,
               sum_clb_11; //sum output wires for each clb

    wire [1:0] sum_00,
               sum_01,
               sum_10,
               sum_11; //sum output wires of each clb routed
               //through routing module

    wire final_cout; //final carry out wire

    wire [7:0] data_1,
               data_2; //initial data input wires

    wire [1:0] num_1_00,
               num_2_00,
               num_1_01,
               num_2_01,
               num_1_10,
               num_2_10,
               num_1_11,
               num_2_11; //clb input wires

    // Instantiate the routing module
    routing routing_inst(
        .bitfile(bitfile),
        .cout_00(cout_00),
        .cout_01(cout_01),
        .cout_10(cout_10),
        .cout_11(cout_11),
        .cout_ext(carry_in),
        .data_1(data_1),
        .data_2(data_2),
        .sin_00(sum_clb_00),
        .sin_01(sum_clb_01),
        .sin_10(sum_clb_10),
        .sin_11(sum_clb_11),
        .cin_00(cin_00),
        .cin_01(cin_01),
        .cin_10(cin_10),
        .cin_11(cin_11),
        .num_1_00(num_1_00),
        .num_2_00(num_2_00),
        .num_1_01(num_1_01),
        .num_2_01(num_2_01),
        .num_1_10(num_1_10),
        .num_2_10(num_2_10),
        .num_1_11(num_1_11),
        .num_2_11(num_2_11),
        .sout_00(sum_00),
        .sout_01(sum_01),
        .sout_10(sum_10),
        .sout_11(sum_11),
        .final_cout(final_cout)
    );

    // Instantiate four CLB modules
    CLB clb_inst_00(
        .num_1(num_1_00),
        .num_2(num_2_00),
        .carry_in(cin_00),
        .clk(clk),
        .reset(reset),
        .sum(sum_clb_00),
        .carry_out(cout_00)
    );

    CLB clb_inst_01(
        .num_1(num_1_01),
        .num_2(num_2_01),
        .carry_in(cin_01),
        .clk(clk),
        .reset(reset),
        .sum(sum_clb_01),
        .carry_out(cout_01)
    );

    CLB clb_inst_10(
        .num_1(num_1_10),
        .num_2(num_2_10),
        .carry_in(cin_10),
        .clk(clk),
        .reset(reset),
        .sum(sum_clb_10),
        .carry_out(cout_10)
    );

    CLB clb_inst_11(
        .num_1(num_1_11),
        .num_2(num_2_11),
        .carry_in(cin_11),
        .clk(clk),
        .reset(reset),
        .sum(sum_clb_11),
        .carry_out(cout_11)
    );

    // Concatenate the sum outputs for final sum
    assign sum = {sum_11 , sum_10 , sum_01 , sum_00};

    // Assign the final carry-out
    assign carry_out = final_cout;

endmodule
```

2.3.2 Schematic

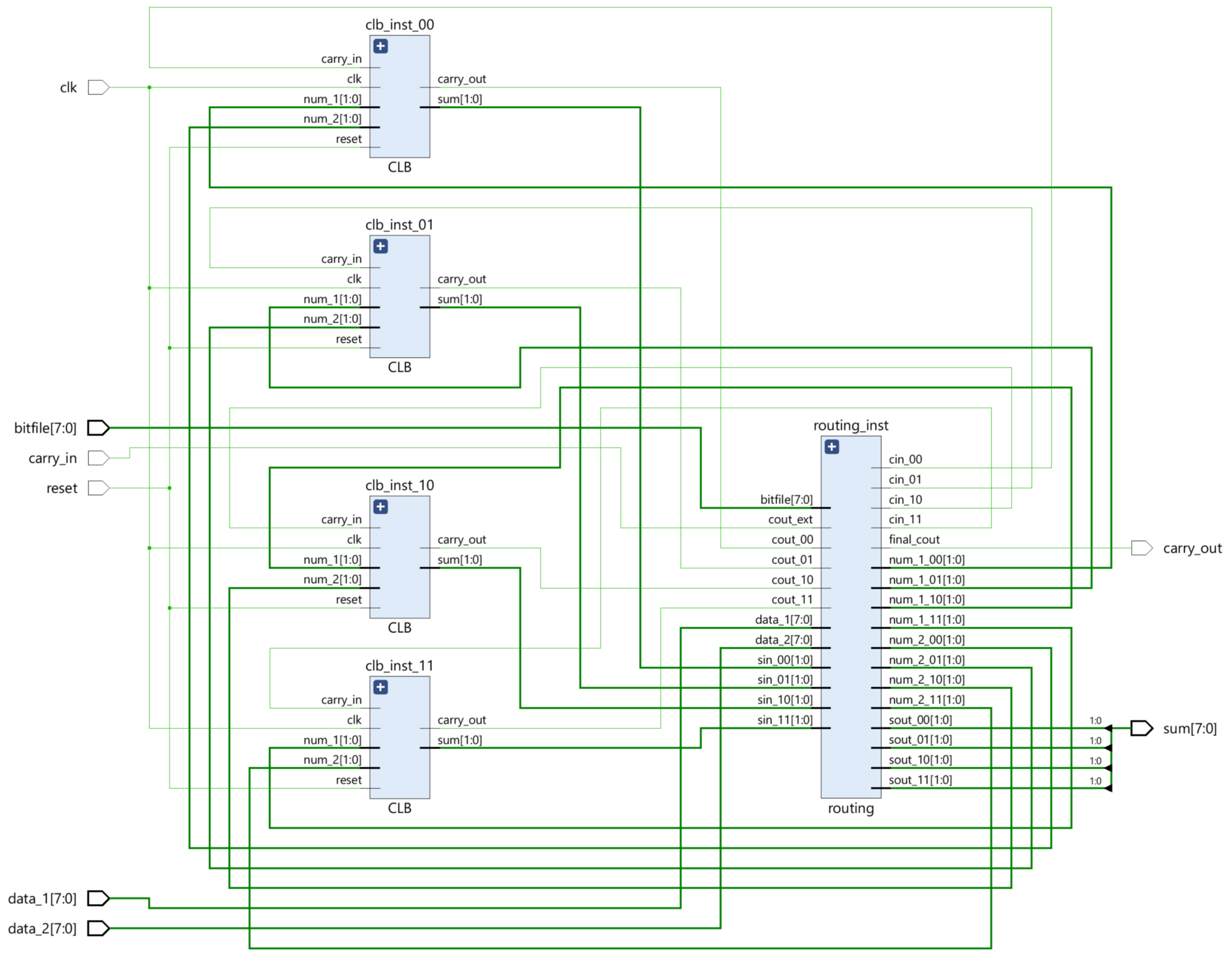

Figure 9: FPGA Top Module Schematic

In this top module schematic we can see the whole architecture being realized. We have four CLBs with routes coming and going into the routing module. All inputs and outputs are also routed through the routing module.

2.4 Test Bench

2.4.1 Code

```
`timescale 1ns/1ps

`include "fpga.v"
`include "clb.v"
`include "routing.v"

module testbench;

  // Inputs
  reg [7:0] data_1, data_2;
  reg [9:0] bitfile;
  reg carry_in, clk, reset;

  // Outputs
  wire [7:0] sum;
  wire carry_out;

  // Instantiate the FPGA module
  fpga dut(
    .data_1(data_1),
    .data_2(data_2),
    .bitfile(bitfile),
    .carry_in(carry_in),
    .clk(clk),
    .reset(reset),
    .sum(sum),
    .carry_out(carry_out)
  );

  // Clock generation
  always #5 clk = ~clk;

  // Initial reset
  initial begin
    $dumpfile("dump.vcd");
        $dumpvars;
    clk = 0;
    reset = 1;
    #10 reset = 0;  // Reset after 10 time units
    // Test scenario 1
    data_1 = 8'b11101010;  // Example data values
    data_2 = 8'b11010101;
    bitfile = 10'b1110010000;
    carry_in = 1'b0;
    #50;  // Wait for 100 time units
    // Test scenario 2
    data_1 = 8'b01101011;  // Example data values
    data_2 = 8'b01010101;
    bitfile = 10'b1110010000;
    carry_in = 1'b0;
    #50
    // Test scenario 3
    data_1 = 8'b10110110;  // Example data values
    data_2 = 8'b01101001;
    bitfile = 10'b1110010000;
    carry_in = 1'b0;
    #50
    $stop;  // Stop simulation
  end

  // Monitor for displaying outputs
  initial
    begin
    $monitor("Time=%0t,␣Sum=%b,␣Carry␣Out=%b", $time, sum, carry_out);
        end

endmodule
```

2.5 Simulation Results

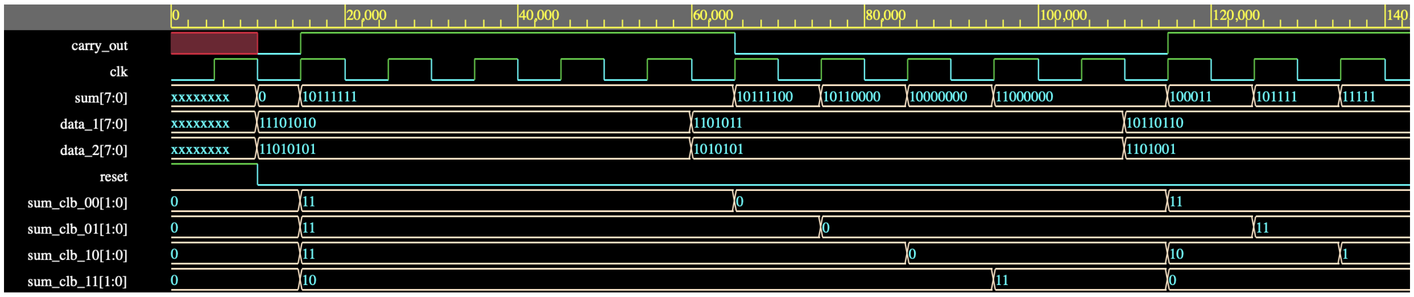

Figure 10: Simulation Result

In simulation I tested it for three different addition operations, and got back the correct results for each. However, it took four clock cycles for computing some combinations while for some it took a single clock cycle.