4. Question 4

4.1. Design code

```
module fifo_memory #(
    /*----------- Parameters -----------*/
    parameter FIFO_WIDTH = 16,              /* Data in/out and memory word width */
    parameter FIFO_DEPTH = 512,             /* Memory depth */
    parameter ADDR_SIZE = $clog2(FIFO_DEPTH) /* Address size based on the memory depth */
)(
    /*--------------Inputs--------------*/
    input [FIFO_WIDTH-1:0] din_a,        /* Write Data: The input data bus used when writing the FIFO */
    input wen_a,                         /* Write Enable. If the FIFO is not full, make it high to write into FIFO */
    input ren_b,                         /* Read Enable. If the FIFO is not empty, make it high to read from FIFO */
    input clk_a,                         /* Clock for port a, used in the writing operation */
    input clk_b,                         /* Clock for port b, used in the reading operation */
    input rst,                           /* Active high synchronous reset. It resets dout_b and internal counter */
    /*--------------outputs-------------*/
    output reg [FIFO_WIDTH-1:0] dout_b,  /* Read Data: The output data bus used when reading from the FIFO */
    output reg full,                     /* Full Flag: Indicates FIFO is full */
    output reg empty                     /* Empty Flag: Indicates FIFO is empty */
);
    /*-----------FIFO MEMORY-----------*/
    reg [FIFO_WIDTH-1:0] fifo_mem [FIFO_DEPTH-1:0];
    /*------Write and Read Pointer-----*/
    reg [ADDR_SIZE-1:0] wr_ptr, rd_ptr;
    /*----Count of elements in FIFO----*/
    reg [ADDR_SIZE:0] fifo_count;

    /*-------Write into the FIFO-------*/
    always @(posedge clk_a ) begin
        if (rst) begin
            wr_ptr <= 0;
            fifo_count <= 0;
            full <= 0;
        end else if (wen_a && !full) begin
            fifo_mem[wr_ptr] <= din_a;
            wr_ptr <= wr_ptr + 1;
            fifo_count <= fifo_count + 1;
            empty <= 0;
            if (fifo_count + 1 == FIFO_DEPTH)
                full <= 1;
        end
    end

    /*-------Read from the FIFO-------*/
    always @(posedge clk_b) begin
        if (rst) begin
            rd_ptr <= 0;
            dout_b <= 0;
            empty  <= 1;
        end else if (ren_b && !empty) begin
            dout_b <= fifo_mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            fifo_count <= fifo_count - 1;
            full <= 0;
            if (fifo_count - 1 == 0)
                empty <= 1;
        end
    end

endmodule
```

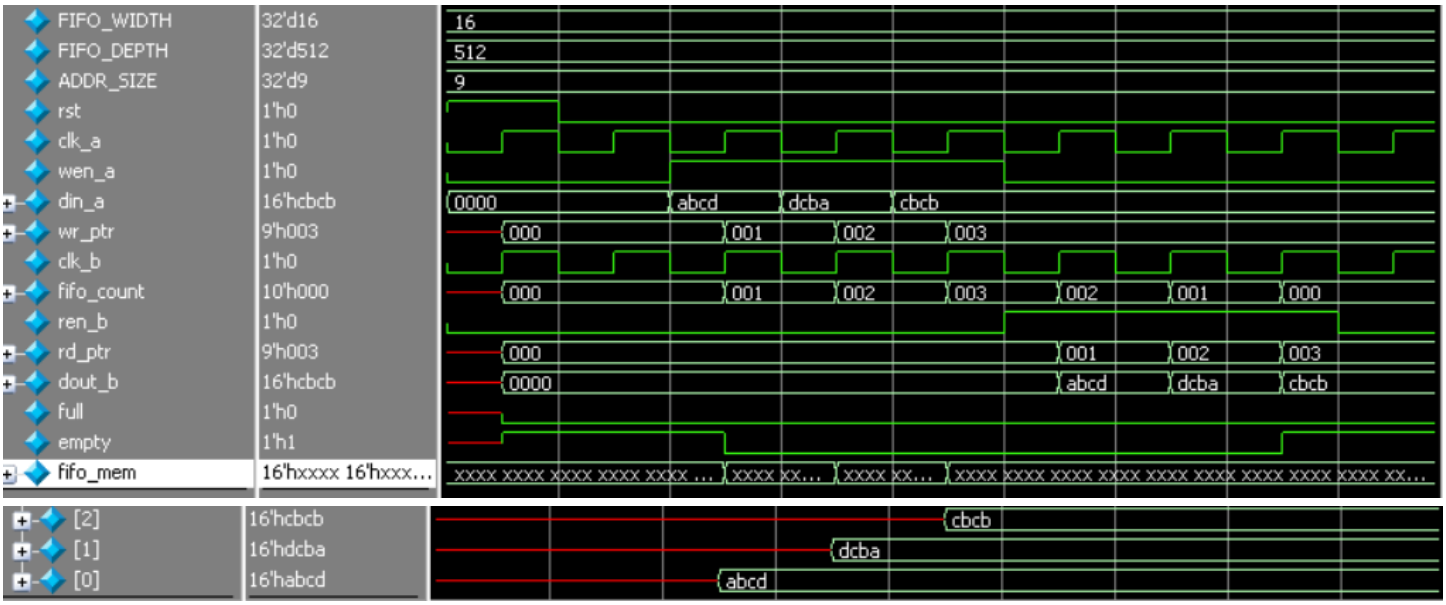
4.2. Testbench

```
module fifo_memory_tb();
    /*----------- Parameters -----------*/
    localparam FIFO_WIDTH = 16;
    localparam FIFO_DEPTH = 512;
    localparam ADDR_SIZE = $clog2(FIFO_DEPTH);
    /*--------------Inputs--------------*/
    reg [FIFO_WIDTH-1:0] din_a;
    reg wen_a;
    reg ren_b;
    reg clk_a;
    reg clk_b;
    reg rst;
    /*--------------outputs-------------*/
    wire [FIFO_WIDTH-1:0] dout_b;
    wire full;
    wire empty;

    /*------Instantiation of DUT--------*/
    fifo_memory #(
        .FIFO_WIDTH(FIFO_WIDTH),
        .FIFO_DEPTH(FIFO_DEPTH),
        .ADDR_SIZE(ADDR_SIZE)
    ) DUT (.*);

    /*-----clock Port-A generation------*/
    initial begin
        clk_a = 0;
        forever #5 clk_a = ~clk_a; // 10ns period
    end
    /*-----clock Port-B generation------*/
    initial begin
        clk_b = 0;
        forever #5 clk_b = ~clk_b; // 10ns period
    end
    /*--------Stimulus process---------*/
    initial begin
        $display("---START SIMULATION---");
        /* intialize the inputs */
        din_a = 0;
        wen_a = 0;
        ren_b = 0;
        rst   = 0;

        /* check reset signal */
        rst = 1;
        @(negedge clk_a);
        rst = 0;
        @(negedge clk_a);
        /* Write to FIFO */
        wen_a = 1;
        din_a = 16'hABCD;
        @(negedge clk_a);
        din_a = 16'hDCBA;
        @(negedge clk_a);
        din_a = 16'hCBCB;
        @(negedge clk_a);
        /* disable write enable */
        wen_a = 0;
        /* Read from FIFO */
        ren_b = 1;
        @(negedge clk_b);
        @(negedge clk_b);
        @(negedge clk_b);
        ren_b = 0;
        @(negedge clk_b);
        $display("---END SIMULATION---");
        $stop;
    end

    // Monitor signals
    initial begin
        $monitor("rst=%b, wen_a=%b, ren_b=%b, din_a=%h, dout_b=%h, full=%b, empty=%b", rst, wen_a, ren_b, din_a, dout_b, full, empty);
    end

endmodule
```

4.3. QuestaSim Transcript

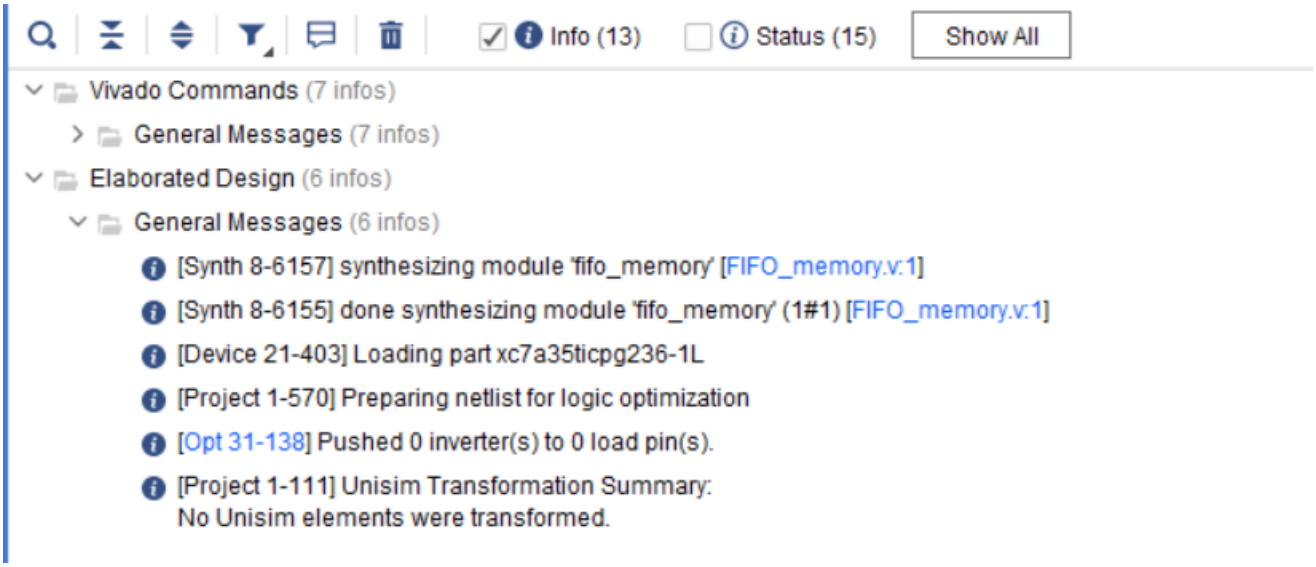
```
# ---START SIMULATION---
# rst=1, wen_a=0, ren_b=0, din_a=0000, dout_b=xxxx, full=x, empty=x
# rst=1, wen_a=0, ren_b=0, din_a=0000, dout_b=0000, full=0, empty=1
# rst=0, wen_a=0, ren_b=0, din_a=0000, dout_b=0000, full=0, empty=1
# rst=0, wen_a=1, ren_b=0, din_a=abcd, dout_b=0000, full=0, empty=1
# rst=0, wen_a=1, ren_b=0, din_a=abcd, dout_b=0000, full=0, empty=0
# rst=0, wen_a=1, ren_b=0, din_a=dcba, dout_b=0000, full=0, empty=0
# rst=0, wen_a=1, ren_b=0, din_a=cbcb, dout_b=0000, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbcb, dout_b=0000, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbcb, dout_b=abcd, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbcb, dout_b=dcba, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbcb, dout_b=cbcb, full=0, empty=1
# rst=0, wen_a=0, ren_b=0, din_a=cbcb, dout_b=cbcb, full=0, empty=1
# ---END SIMULATION---
```

4.4. QuestaSim Waveform

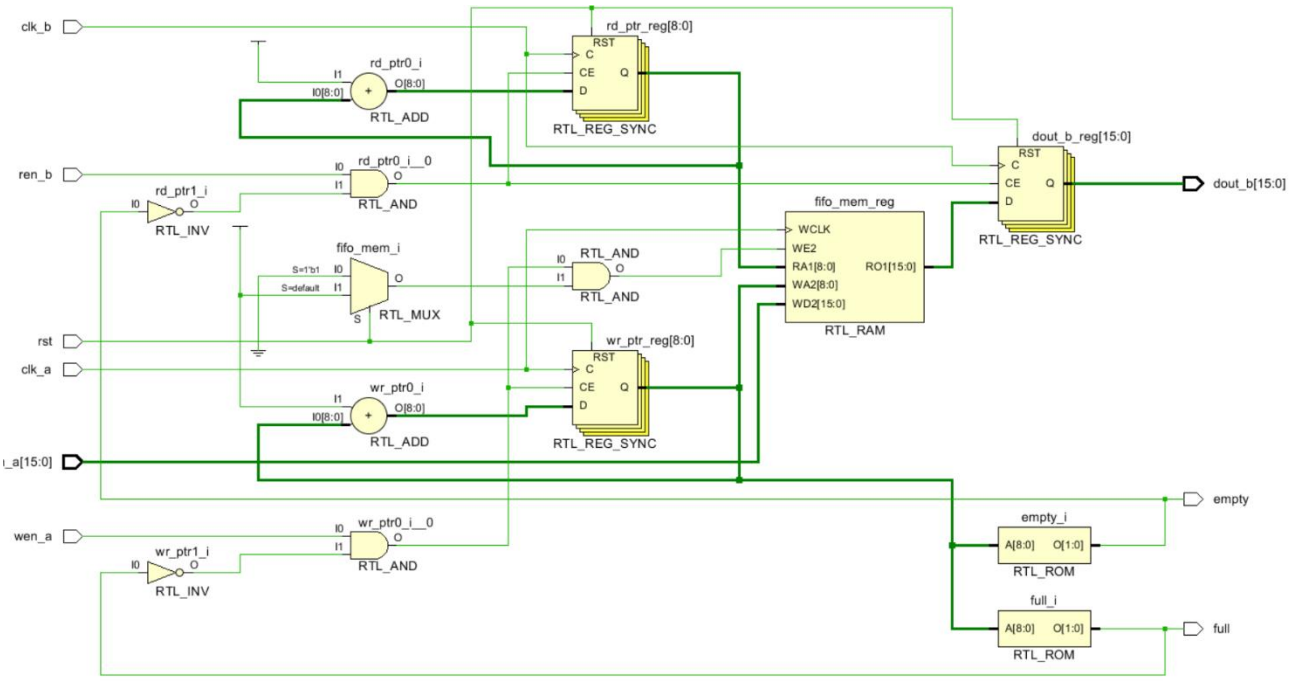


4.5. Elaboration

4.5.1. Messages Tab

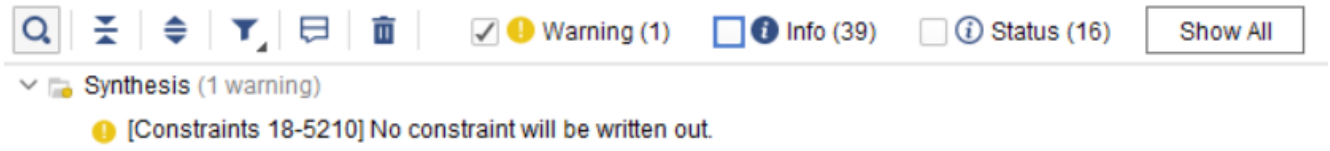


4.5.2. Schematic

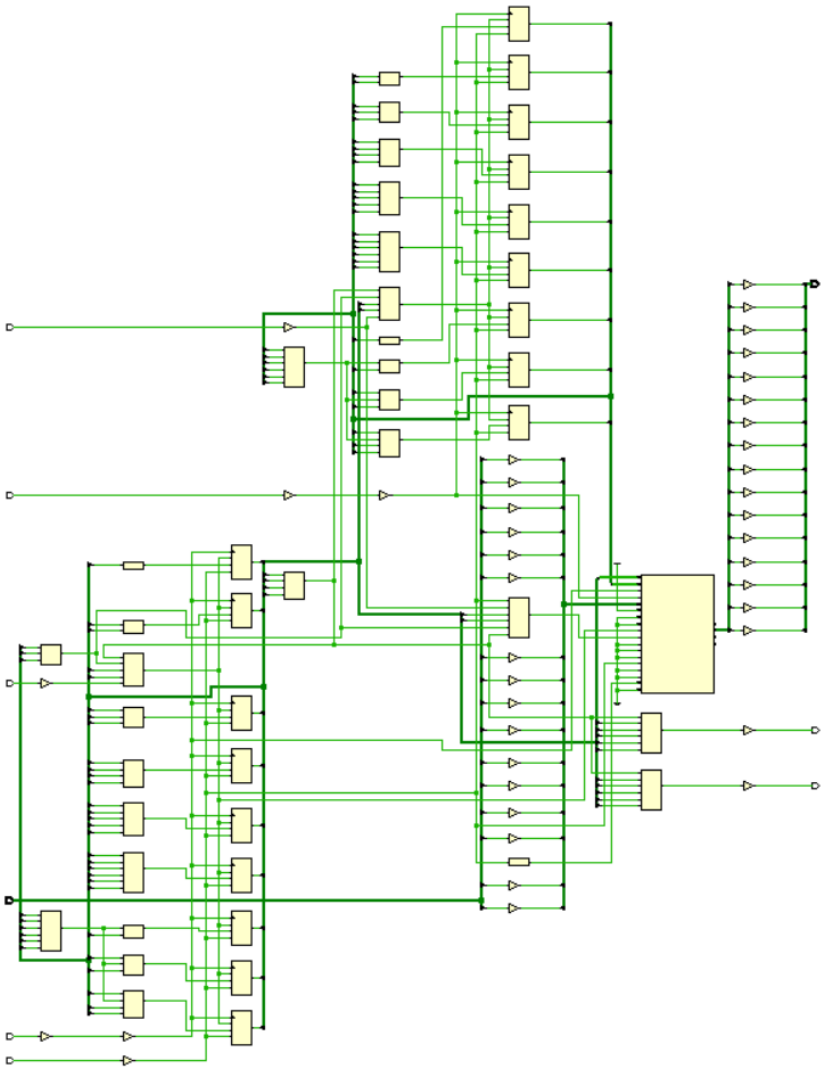


4.6. Synthesis

4.6.1. Messages Tab



4.6.2. Schematic



4.7. Implementation

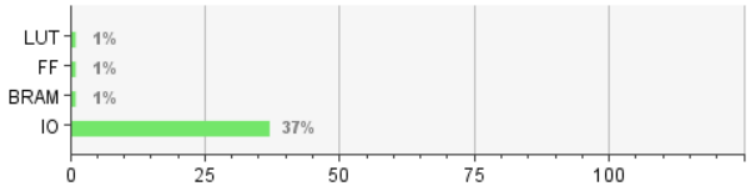
4.7.1. Utilization Report

Hierarchy

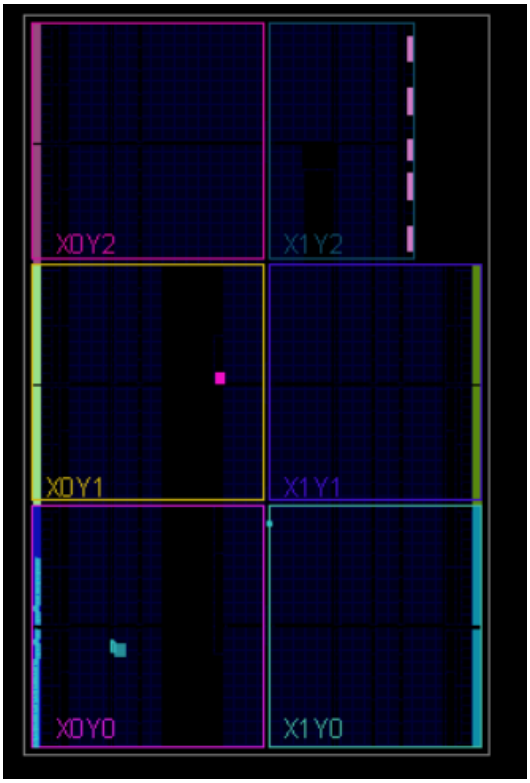
Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N fifo_memory	20	18	5	20	10	0.5	39	2

Summary

Resource	Utilization	Available	Utilization %
LUT	20	20800	0.10
FF	18	41600	0.04
BRAM	0.50	50	1.00
IO	39	106	36.79



4.7.2. FPGA Device



4.8. Snippet from Netlist file

```
/ Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
// ------------------------------------------------------------------------------
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018
// Date        : Mon Aug  5 12:13:30 2024
// Host        : Youssif running 64-bit major release  (build 9200)
// Command     : write_verilog {D:/CUFE/Diplomas/Digital
//               Diploma/Coding/SPI_Slave_single_port_Ram/FPGA/project_FIFO_memory.v}
// Design      : fifo_memory
// Purpose     : This is a Verilog netlist of the current design or from a specific cell of the design. The output is an
//               IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input
//               design files.
// Device      : xc7a35ticpg236-1L
// ------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

(* ADDR_SIZE = "9" *) (* FIFO_DEPTH = "512" *) (* FIFO_WIDTH = "16" *)
(* STRUCTURAL_NETLIST = "yes" *)
module fifo_memory
   (din_a,
    wen_a,
    ren_b,
    clk_a,
    clk_b,
    rst,
    dout_b,
    full,
    empty);
  input [15:0]din_a;
  input wen_a;
  input ren_b;
  input clk_a;
  input clk_b;
  input rst;
  output [15:0]dout_b;
  output full;
  output empty;

  wire \<const0> ;
  wire \<const1> ;
  wire clk_a;
  wire clk_a_IBUF;
  wire clk_a_IBUF_BUFG;
  wire clk_b;
  wire clk_b_IBUF;
  wire clk_b_IBUF_BUFG;
  wire [15:0]din_a;
  wire [15:0]din_a_IBUF;
  wire [15:0]dout_b;
  wire [15:0]dout_b_OBUF;
  wire empty;
  wire empty_OBUF;
  wire fifo_mem_reg_i_2_n_0;
  wire fifo_mem_reg_i_3_n_0;
  wire fifo_mem_reg_i_4_n_0;
  wire full;
  wire full_OBUF;
  wire full_OBUF_inst_i_2_n_0;
  wire [8:0]p_0_in;
  wire [8:0]rd_ptr;
  wire rd_ptr0;
  wire \rd_ptr[0]_i_2_n_0 ;
  wire \rd_ptr[1]_i_1_n_0 ;
  wire \rd_ptr[2]_i_1_n_0 ;
  wire \rd_ptr[3]_i_1_n_0 ;
  wire \rd_ptr[4]_i_1_n_0 ;
  wire \rd_ptr[5]_i_1_n_0 ;
  wire \rd_ptr[6]_i_1_n_0 ;
  wire \rd_ptr[6]_i_2_n_0 ;
  wire \rd_ptr[7]_i_1_n_0 ;
  wire \rd_ptr[8]_i_1_n_0 ;
  wire ren_b;
  wire ren_b_IBUF;
  wire rst;
  wire rst_IBUF;
  wire wen_a;
  wire wen_a_IBUF;
  wire wr_ptr0;
  wire \wr_ptr[8]_i_2_n_0 ;
  wire [8:0]wr_ptr_reg__0;

  GND GND
       (.G(\<const0> ))
```

5. Question 5

5.1. Snippet after instantiation of IP modules

```
wire [5:0]adder_out,multiplier_out;
    /* Adder IP Instantiotion */
    generate
        if(FULL_ADDER = "ON")
           c_addsub_0 adder_mod (
             .A(A_r),             // input wire [2 : 0] A
             .B(B_r),             // input wire [2 : 0] B
             .C_IN(cin_r),        // input wire C_IN
             .S(adder_out)        // output wire [3 : 0] S
            );
        else if(FULL_ADDER = "OFF")
           c_addsub_0 adder_mod (
             .A(A_r),             // input wire [2 : 0] A
             .B(B_r),             // input wire [2 : 0] B
             .C_IN(cin_r),        // input wire C_IN
             .S(adder_out)        // output wire [3 : 0] S
            );
    endgenerate

    /* Multiplication IP Instantiotion */
    mult_gen_0 your_instance_name (
        .A(A_r),  // input wire [2 : 0] A
        .B(B_r),  // input wire [2 : 0] B
        .P(multiplier_out)  // output wire [5 : 0] P
    );

    /* always block for storing the inputs value to D-FF with rising edge of clock if rst is LOW */
    always @(posedge clk or posedge rst) begin : INPUTS_REGISTERATION
        if(rst)begin
            A_r <= 0;
            B_r <= 0;
            opcode_r <=0;
            cin_r    <= 0;
            serial_in_r <= 0;
            red_op_A_r  <= 0;
            red_op_B_r  <= 0;
            bypass_A_r  <= 0;
            bypass_B_r  <= 0;
            direction_r <= 0;
            out_r <= 0;
        end
        else begin
            A_r <= A;
            B_r <= B;
            opcode_r <= opcode;
            cin_r    <= cin;
            serial_in_r <= serial_in;
            red_op_A_r  <= red_op_A;
            red_op_B_r  <= red_op_B;
            bypass_A_r  <= bypass_A;
            bypass_B_r  <= bypass_B;
            direction_r <= direction;

        if(bypass_A_r && bypass_B_r)
            out_r <= First_priority;
        else if (bypass_A_r)
            out_r <= A_r;
        else if (bypass_B_r)
            out_r <= B_r;
        else if(~invalid_case)begin
        case (opcode_r)
            3'b000 : begin
                /* AND operation */
                if(red_op_A_r && red_op_B_r)    out_r <= &First_priority;
                else if(red_op_A_r)     out_r <= & A_r;
                else if(red_op_B_r)     out_r <= & B_r;
                else    out_r <= A_r & B_r;
            end
            3'b001 : begin
                /* XOR operation */
                if(red_op_A_r && red_op_B_r)    out_r <= ^First_priority;
                else if(red_op_A_r)     out_r <= ^A_r;
                else if(red_op_B_r)     out_r <= ^B_r;
                else out_r <= A_r ^ B_r;
            end
            3'b010 :begin
                /* addition */
                 out_r <= adder_out;
            end
```