

## 4. Question 4

### 4.1. Design code

```
1 module fifo_memory #(
2     /*----- Parameters -----*/
3     parameter FIFO_WIDTH = 16,          /* Data in/out and memory word width */
4     parameter FIFO_DEPTH = 512,         /* Memory depth */
5     parameter ADDR_SIZE = $clog2(FIFO_DEPTH) /* Address size based on the memory depth */
6 );
7     /*-----Inputs-----*/
8     input [FIFO_WIDTH-1:0] din_a,        /* Write Data: The input data bus used when writing the FIFO */
9     input wen_a,                          /* Write Enable. If the FIFO is not full, make it high to write into FIFO */
10    input ren_b,                           /* Read Enable. If the FIFO is not empty, make it high to read from FIFO */
11    input clk_a,                           /* Clock for port a, used in the writing operation */
12    input clk_b,                           /* Clock for port b, used in the reading operation */
13    input rst,                             /* Active high synchronous reset. It resets dout_b and internal counter */
14    /*-----outputs-----*/
15    output reg [FIFO_WIDTH-1:0] dout_b,    /* Read Data: The output data bus used when reading from the FIFO */
16    output reg full,                       /* Full Flag: Indicates FIFO is full */
17    output reg empty                       /* Empty Flag: Indicates FIFO is empty */
18 );
19 /*-----FIFO MEMORY-----*/
20 reg [FIFO_WIDTH-1:0] fifo_mem [FIFO_DEPTH-1:0];
21 /*-----Write and Read Pointer-----*/
22 reg [ADDR_SIZE-1:0] wr_ptr, rd_ptr;
23 /*-----Count of elements in FIFO-----*/
24 reg [ADDR_SIZE:0] fifo_count;
25
26 /*-----Write into the FIFO-----*/
27 always @(posedge clk_a ) begin
28     if (rst) begin
29         wr_ptr <= 0;
30         fifo_count <= 0;
31         full <= 0;
32     end else if (wen_a && !full) begin
33         fifo_mem[wr_ptr] <= din_a;
34         wr_ptr <= wr_ptr + 1;
35         fifo_count <= fifo_count + 1;
36         empty <= 0;
37         if (fifo_count + 1 == FIFO_DEPTH)
38             full <= 1;
39     end
40 end
41
42 /*-----Read from the FIFO-----*/
43 always @(posedge clk_b) begin
44     if (rst) begin
45         rd_ptr <= 0;
46         dout_b <= 0;
47         empty <= 1;
48     end else if (ren_b && !empty) begin
49         dout_b <= fifo_mem[rd_ptr];
50         rd_ptr <= rd_ptr + 1;
51         fifo_count <= fifo_count - 1;
52         full <= 0;
53         if (fifo_count - 1 == 0)
54             empty <= 1;
55     end
56 end
57
58 endmodule
```

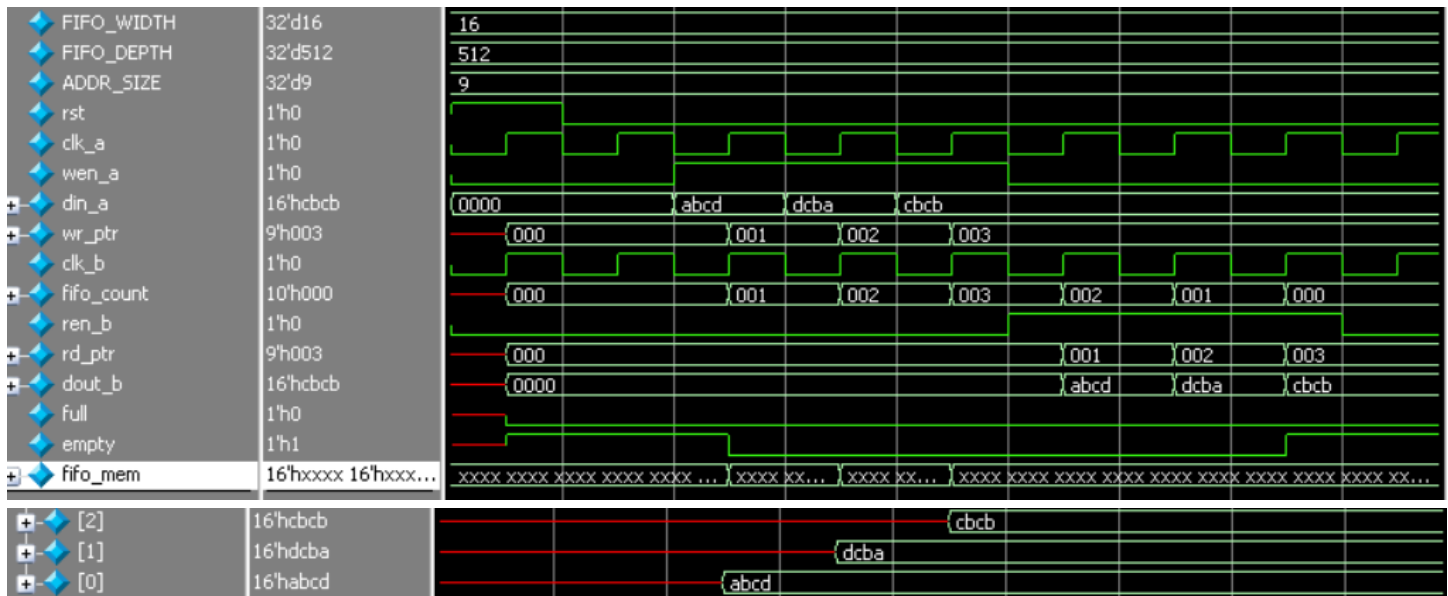
## 4.2. Testbench

```
1 module fifo_memory_tb();
2   /*----- Parameters -----*/
3   localparam FIFO_WIDTH = 16;
4   localparam FIFO_DEPTH = 512;
5   localparam ADDR_SIZE = $clog2(FIFO_DEPTH);
6   /*-----Inputs-----*/
7   reg [FIFO_WIDTH-1:0] din_a;
8   reg wen_a;
9   reg ren_b;
10  reg clk_a;
11  reg clk_b;
12  reg rst;
13  /*-----outputs-----*/
14  wire [FIFO_WIDTH-1:0] dout_b;
15  wire full;
16  wire empty;
17
18  /*-----Instantiation of DUT-----*/
19  fifo_memory #(
20    .FIFO_WIDTH(FIFO_WIDTH),
21    .FIFO_DEPTH(FIFO_DEPTH),
22    .ADDR_SIZE(ADDR_SIZE)
23  ) DUT (.);
24
25  /*-----clock Port-A generation-----*/
26  initial begin
27    clk_a = 0;
28    forever #5 clk_a = ~clk_a; // 10ns period
29  end
30  /*-----clock Port-B generation-----*/
31  initial begin
32    clk_b = 0;
33    forever #5 clk_b = ~clk_b; // 10ns period
34  end
35  /*-----Stimulus process-----*/
36  initial begin
37    $display("---START SIMULATION---");
38    /* initialize the inputs */
39    din_a = 0;
40    wen_a = 0;
41    ren_b = 0;
42    rst = 0;
43
44    /* check reset signal */
45    rst = 1;
46    @(negedge clk_a);
47    rst = 0;
48    @(negedge clk_a);
49    /* Write to FIFO */
50    wen_a = 1;
51    din_a = 16'hABCD;
52    @(negedge clk_a);
53    din_a = 16'hDCBA;
54    @(negedge clk_a);
55    din_a = 16'hC8CB;
56    @(negedge clk_a);
57    /* disable write enable */
58    wen_a = 0;
59    /* Read from FIFO */
60    ren_b = 1;
61    @(negedge clk_b);
62    @(negedge clk_b);
63    @(negedge clk_b);
64    ren_b = 0;
65    @(negedge clk_b);
66    $display("---END SIMULATION---");
67    $stop;
68  end
69
70  // Monitor signals
71  initial begin
72    $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);
73  end
74
75 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=cbbb, dout_b=0000, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbbb, dout_b=0000, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbbb, dout_b=abcd, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbbb, dout_b=dcba, full=0, empty=0
# rst=0, wen_a=0, ren_b=1, din_a=cbbb, dout_b=cbbb, full=0, empty=1
# rst=0, wen_a=0, ren_b=0, din_a=cbbb, dout_b=cbbb, full=0, empty=1
# ---END SIMULATION---
```

### 4.4. QuestaSim Waveform



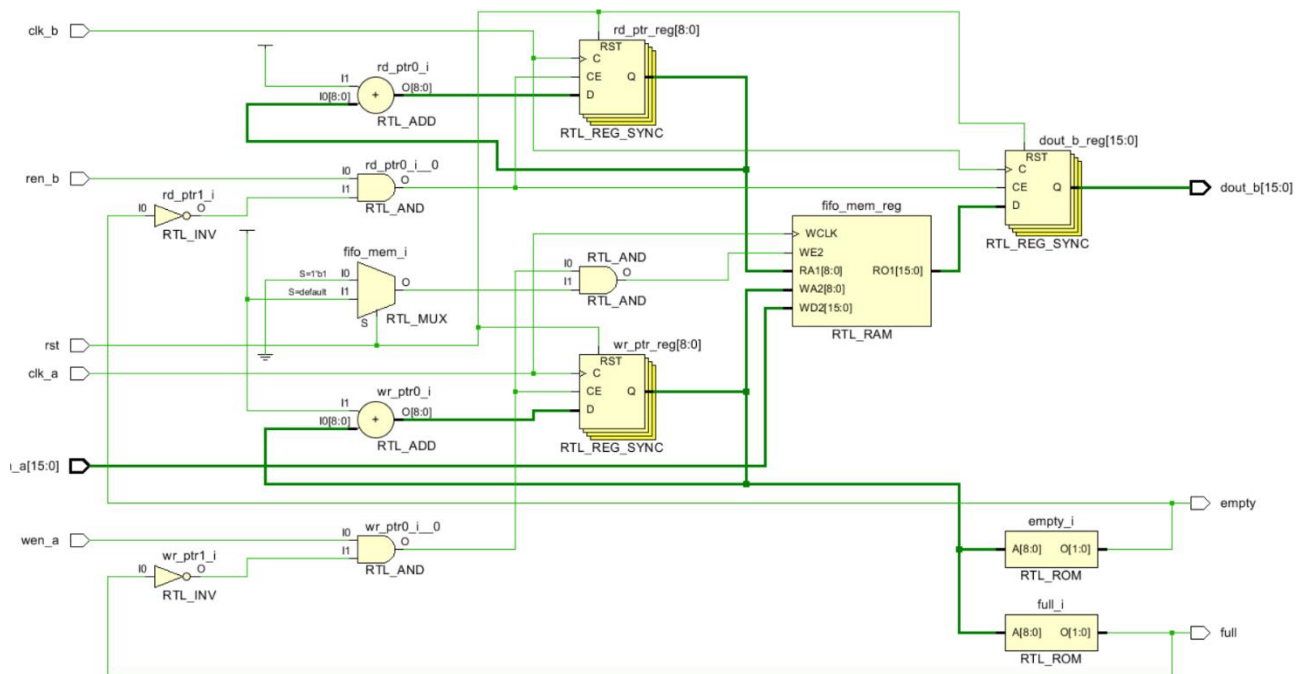
## 4.5. Elaboration

### 4.5.1. Messages Tab

Search, Filter, Sort, Info (13), Status (15), Show All

- Vivado Commands (7 infos)
  - General Messages (7 infos)
- Elaborated Design (6 infos)
  - General Messages (6 infos)
    - [Synth 8-6157] synthesizing module 'fifo\_memory' [FIFO\_memory.v:1]
    - [Synth 8-6155] done synthesizing module 'fifo\_memory' (1#1) [FIFO\_memory.v:1]
    - [Device 21-403] Loading part xc7a35ticpg236-1L
    - [Project 1-570] Preparing netlist for logic optimization
    - [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
    - [Project 1-111] Unisim Transformation Summary:  
No Unisim elements were transformed.

### 4.5.2. Schematic



## 4.6. Synthesis

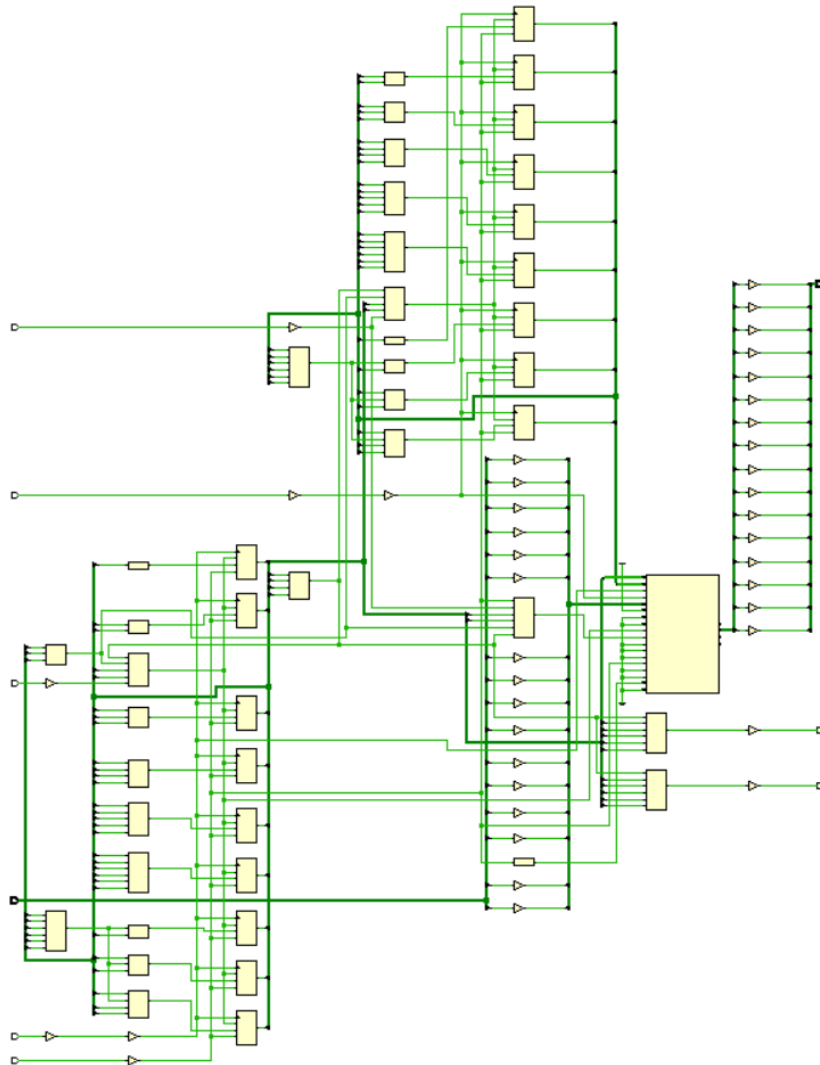
### 4.6.1. Messages Tab

Search | Filter | Sort | View | [X] Warning (1) [X] Info (39) [X] Status (16) [Show All]

▼ Synthesis (1 warning)

[Warning] [Constraints 18-5210] No constraint will be written out.

### 4.6.2. Schematic



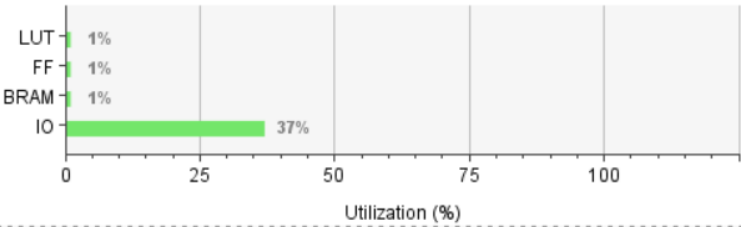
## 4.7. Implementation

### 4.7.1. Utilization Report

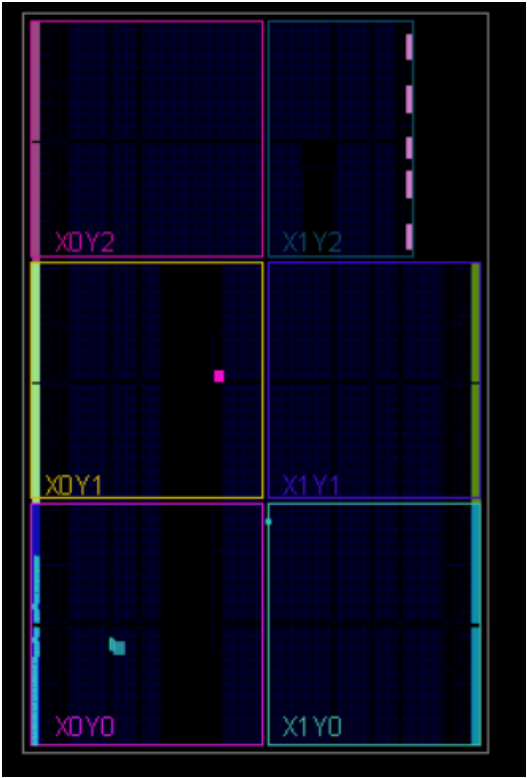
Hierarchy								
Name	1	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)
N fifo_memory		20	18	5	20	10	0.5	39

#### 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

```
1 / Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
2 // -----
3 // Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018
4 // Date       : Mon Aug  5 12:13:30 2024
5 // Host       : Youssif running 64-bit major release (build 9200)
6 // Command    : write_verilog {D:/CUFE/Diplomas/Digital
7 //             : Diploma/Coding/SPI_Slave_single_port_Ram/FPGA/project_FIFO_memory.v}
8 // Design     : fifo_memory
9 // Purpose    : This is a Verilog netlist of the current design or from a specific cell of the design. The output is an
10 //            : IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input
11 //            : design files.
12 // Device     : xc7a35ticpg236-1l
13 // -----
14 `timescale 1 ps / 1 ps
15
16 (* ADDR_SIZE = "9" *) (* FIFO_DEPTH = "512" *) (* FIFO_WIDTH = "16" *)
17 (* STRUCTURAL_NETLIST = "yes" *)
18 module fifo_memory
19     (din_a,
20     wen_a,
21     ren_b,
22     clk_a,
23     clk_b,
24     rst,
25     dout_b,
26     full,
27     empty);
28     input [15:0]din_a;
29     input wen_a;
30     input ren_b;
31     input clk_a;
32     input clk_b;
33     input rst;
34     output [15:0]dout_b;
35     output full;
36     output empty;
37
38     wire \<const0> ;
39     wire \<const1> ;
40     wire clk_a;
41     wire clk_a_IBUF;
42     wire clk_a_IBUF_BUFG;
43     wire clk_b;
44     wire clk_b_IBUF;
45     wire clk_b_IBUF_BUFG;
46     wire [15:0]din_a;
47     wire [15:0]din_a_IBUF;
48     wire [15:0]dout_b;
49     wire [15:0]dout_b_OBUF;
50     wire empty;
51     wire empty_OBUF;
52     wire fifo_mem_reg_i_2_n_0;
53     wire fifo_mem_reg_i_3_n_0;
54     wire fifo_mem_reg_i_4_n_0;
55     wire full;
56     wire full_OBUF;
57     wire full_OBUF_inst_i_2_n_0;
58     wire [8:0]p_0_in;
59     wire [8:0]rd_ptr;
60     wire rd_ptr0;
61     wire \rd_ptr[0]_i_2_n_0 ;
62     wire \rd_ptr[1]_i_1_n_0 ;
63     wire \rd_ptr[2]_i_1_n_0 ;
64     wire \rd_ptr[3]_i_1_n_0 ;
65     wire \rd_ptr[4]_i_1_n_0 ;
66     wire \rd_ptr[5]_i_1_n_0 ;
67     wire \rd_ptr[6]_i_1_n_0 ;
68     wire \rd_ptr[6]_i_2_n_0 ;
69     wire \rd_ptr[7]_i_1_n_0 ;
70     wire \rd_ptr[8]_i_1_n_0 ;
71     wire ren_b;
72     wire ren_b_IBUF;
73     wire rst;
74     wire rst_IBUF;
75     wire wen_a;
76     wire wen_a_IBUF;
77     wire wr_ptr0;
78     wire \wr_ptr[8]_i_2_n_0 ;
79     wire [8:0]wr_ptr_reg_0;
80
81     GND GND
82     (.G(\<const0> ))
```

## 5. Question 5

### 5.1. Snippet after instantiation of IP modules

```
1 wire [5:0]adder_out,multiplier_out;
2 /* Adder IP Instantiation */
3 generate
4     if(FULL_ADDER = "ON")
5         c_addsub_0 adder_mod (
6             .A(A_r),          // input wire [2 : 0] A
7             .B(B_r),          // input wire [2 : 0] B
8             .C_IN(cin_r),      // input wire C_IN
9             .S(adder_out)      // output wire [3 : 0] S
10        );
11     else if(FULL_ADDER = "OFF")
12         c_addsub_0 adder_mod (
13             .A(A_r),          // input wire [2 : 0] A
14             .B(B_r),          // input wire [2 : 0] B
15             .C_IN(cin_r),      // input wire C_IN
16             .S(adder_out)      // output wire [3 : 0] S
17        );
18 endgenerate
19
20 /* Multiplication IP Instantiation */
21 mult_gen_0 your_instance_name (
22     .A(A_r), // input wire [2 : 0] A
23     .B(B_r), // input wire [2 : 0] B
24     .P(multiplier_out) // output wire [5 : 0] P
25 );
26
27 /* always block for storing the inputs value to D-FF with rising edge of clock if rst is LOW */
28 always @(posedge clk or posedge rst) begin : INPUTS_REGISTRATION
29     if(rst)begin
30         A_r <= 0;
31         B_r <= 0;
32         opcode_r <=0;
33         cin_r <= 0;
34         serial_in_r <= 0;
35         red_op_A_r <= 0;
36         red_op_B_r <= 0;
37         bypass_A_r <= 0;
38         bypass_B_r <= 0;
39         direction_r <= 0;
40         out_r <= 0;
41     end
42     else begin
43         A_r <= A;
44         B_r <= B;
45         opcode_r <= opcode;
46         cin_r <= cin;
47         serial_in_r <= serial_in;
48         red_op_A_r <= red_op_A;
49         red_op_B_r <= red_op_B;
50         bypass_A_r <= bypass_A;
51         bypass_B_r <= bypass_B;
52         direction_r <= direction;
53     end
54     if(bypass_A_r && bypass_B_r)
55         out_r <= First_priority;
56     else if (bypass_A_r)
57         out_r <= A_r;
58     else if (bypass_B_r)
59         out_r <= B_r;
60     else if(~invalid_case)begin
61         case (opcode_r)
62             3'b000 : begin
63                 /* AND operation */
64                 if(red_op_A_r && red_op_B_r) out_r <= &First_priority;
65                 else if(red_op_A_r) out_r <= &A_r;
66                 else if(red_op_B_r) out_r <= &B_r;
67                 else out_r <= A_r & B_r;
68             end
69             3'b001 : begin
70                 /* XOR operation */
71                 if(red_op_A_r && red_op_B_r) out_r <= ^First_priority;
72                 else if(red_op_A_r) out_r <= ^A_r;
73                 else if(red_op_B_r) out_r <= ^B_r;
74                 else out_r <= A_r ^ B_r;
75             end
76             3'b010 :begin
77                 /* addition */
78                 out_r <= adder_out;
79             end
80         endcase
81     end
82 end
```