Ho Chi Minh City National University

University of Information Technology

Computer Engineering

Digital System Design With Verilog

Subject: FIFO AND SRA

Class: CE213.P21.2

Instructor: Lam Duc Khai

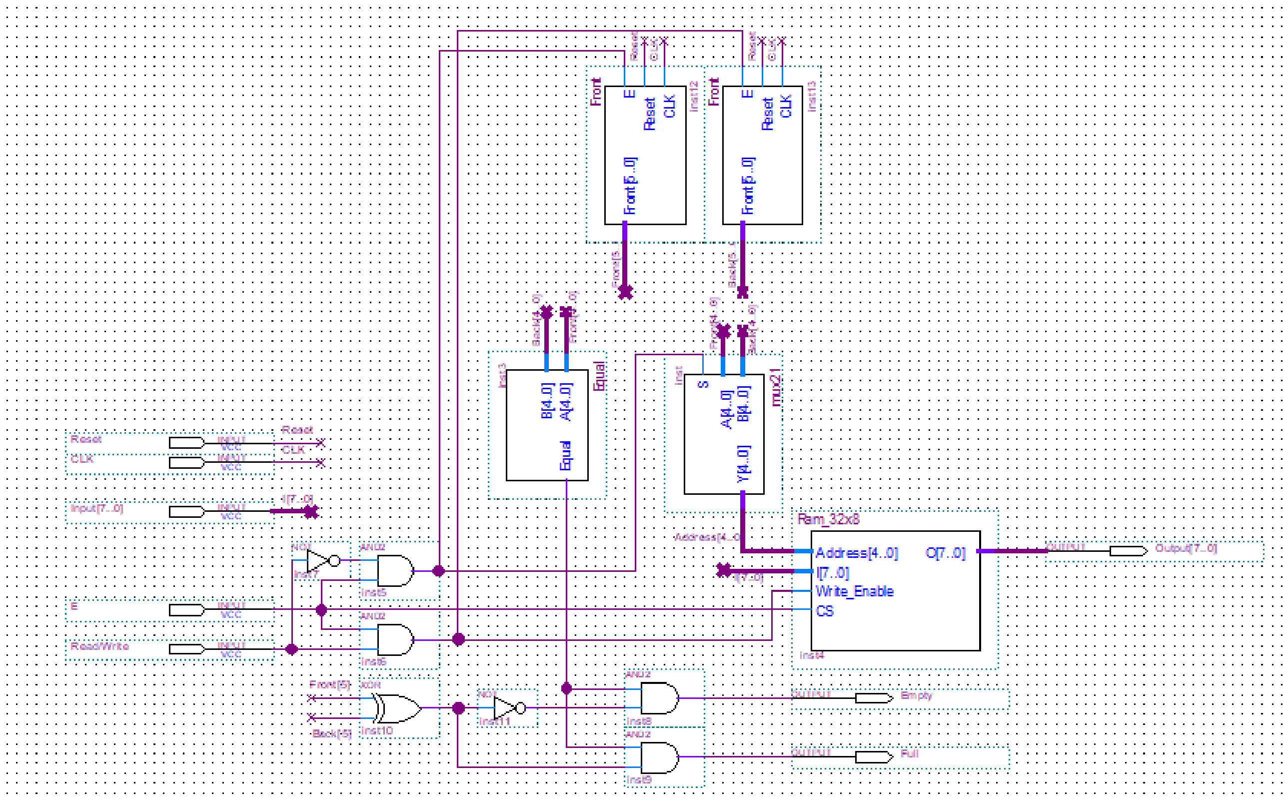
Performed by students: Trương Thiên Quý

Ho Chi Minh City, 4/2025

FIFO_STRUCTURAL:

FIFO is a memory in which data entered first will be output first. It includes 5 blocks with important function: Counter (Front and Back), Ram32x8, Equal, and Mux2_1.

a. Architecture:



- According to the designed architecture, FIFO uses "RAM32X8" memory. With the address selected from the set "Mux_2_1" with the selected data being 2 numbers representing the 2 addresses "Front" and "Back". When "Write" we will write data to the "Back" address and "Back" will add 1 to indicate the next free memory cell, conversely, when "Read" we will read data from the "Front" address and Front will also add to point to the cell with the next data.
- The input signals are as follows:
  - CLK: When positive edge CLK, "Output" will change according to "Input". The "Reset" signal alone can reset FIFO at any time.
  - Input: input data.
  - Enable:
    - When “Enable” equal 0, FIFO will not write or read and "Output" is now equal Z.
    - In contrast, FIFO works.
  - Read/Write:
    - When this signal is equal to 0, the FIFO will perform the function of "Read" data from the "Front" address, if FIFO not empty, "Output" will be equal to that data.
    - When this signal is equal to 1, if FIFO not full, it will perform the function of "Write" data to the "Back" and "Output" addresses at this time equal to Z.

- Reset:
  - When “Reset” equal 0, FIFO works normally.
  - On the contrary, “Reset” equal 1, FIFO will return to its original state, that is the "Front" and "Back" addresses are both equal to 0, FIFO is empty.

- Output signals are as follows:
  - Output: output data.
  - Empty: When "Front" is equal to "Back", then "Empty" equals 1, which means the FIFO is empty and can write data to the "Back" address. Otherwise, "Empty" equals 0.
  - Full: When all memory cells have data, meaning "Back" and "Front" point to the same address cell, but with different sign bits, it can be said that "Back" just points to the last empty memory cell and after increasing by 1 unit, it is at the correct address "Front".

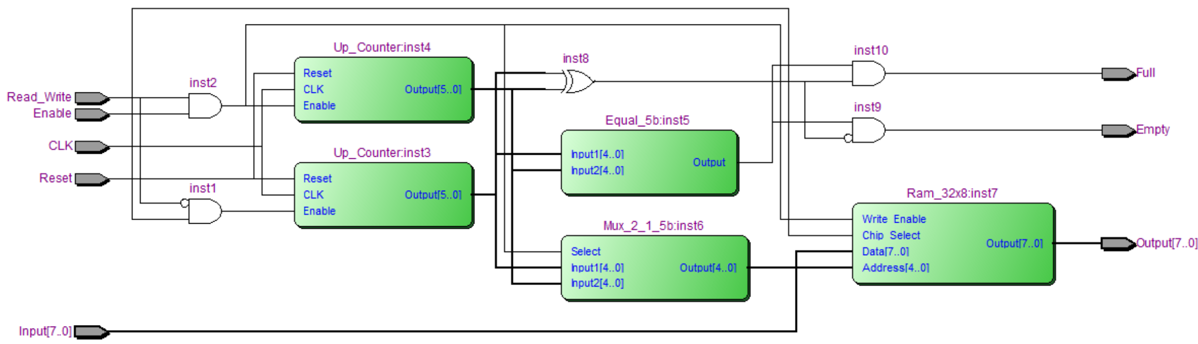
In this design, "Empty" and "Full" have 5 bits but only the first 4 bits are used as the address, the last bit is the sign bit to identify "Full".

b. Verilog code:

```
module FIFO (Output, Empty, Full, Input, Reset, Read_Write, Enable, CLK);
//define_IO
   input [7:0] Input;
   input Reset, Read_Write, Enable, CLK;
   output [7:0] Output;
   output Empty, Full;

//define_wire/reg
   wire Front_E, Back_E, A, Equal;
   wire [5:0] Front, Back;
   wire [4:0] Address;
//body
   and inst1 (Front_E, ~Read_Write, Enable);
   and inst2 (Back_E, Read_Write, Enable);

   Up_Counter inst3 (Front, Reset, CLK, Front_E);
   Up_Counter inst4 (Back, Reset, CLK, Back_E);

   Equal_5b inst5 (Equal, Front[4:0], Back[4:0]);
   Mux_2_1_5b inst6 (Address, Front[4:0], Back[4:0], Back_E);
   Ram_32x8 inst7 (Output, Input, Address, Back_E, Enable);

   xor inst8 (A, Front[5], Back[5]);
   and inst9 (Empty, Equal, ~A);
   and inst10 (Full, Equal, A);

endmodule
```

In Verilog code we also design modules, and input/output signals like the above architecture.

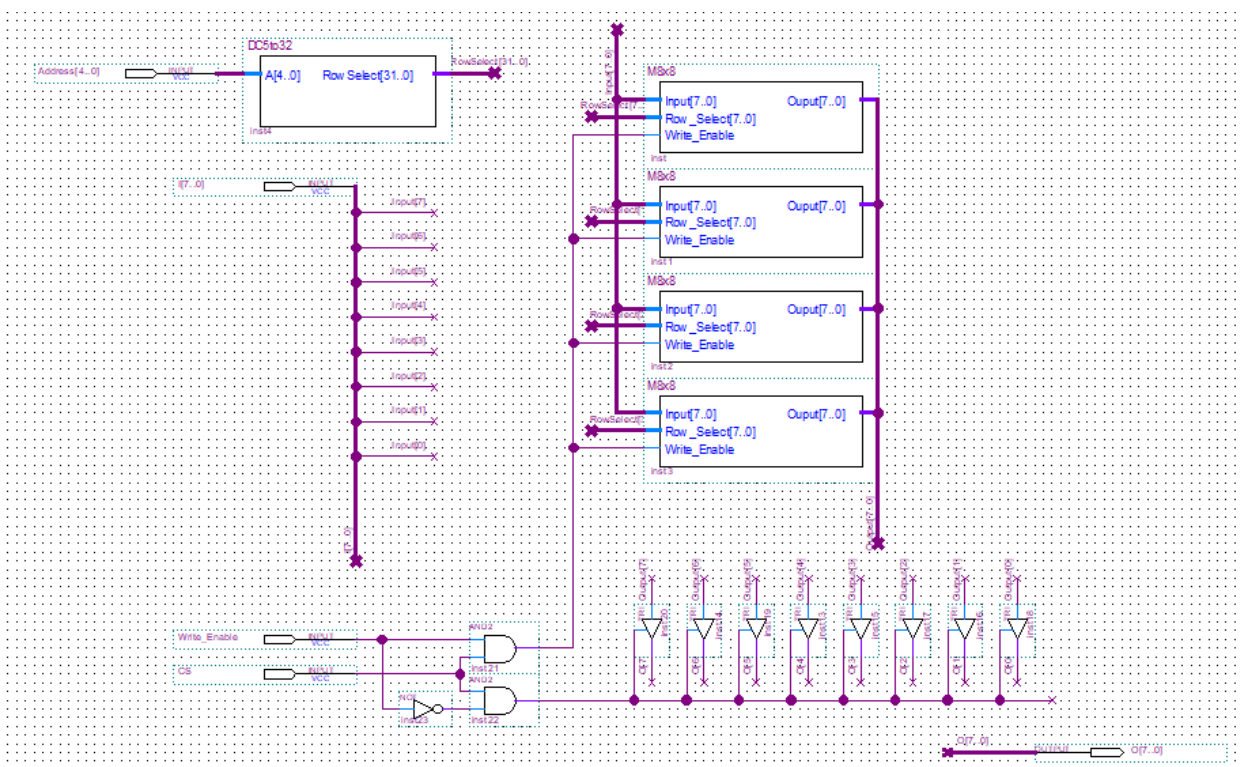
c. RTL after synthesis:



We see it is similar to the previously designed architecture.

1. Ram_32x8:

a. Architecture:



- Connecting the 4 blocks of 8x8 memory cell, we get a 32x8 block of memory cells.
- The "Output" signals are in turn connected to the tri-state, relying on the CS x (RWS)' signal to control the "Output" in read and write states.

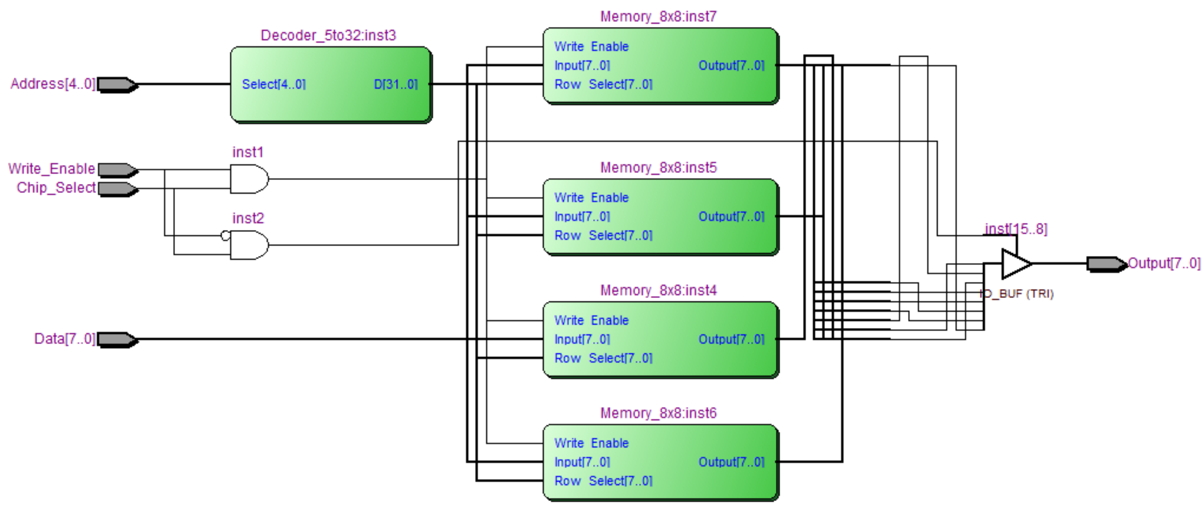
- The Row_Select pins connect to the 5 to 32 decoder to access the address of each row (register).
- RWS x CS input to generate the Write_Enable signal.

b. Verilog code:

```
module Ram_32x8(Output, Data, Address, Write_Enable, Chip_Select);
//define_IO
   input [4:0] Address;
   input [7:0] Data;
   input Write_Enable, Chip_Select;
   output [7:0] Output;

//define_wire/reg
   wire WE, CS;
   wire [32:0] RS;
   wire [7:0] Out;

//body
   and inst1 (WE, Write_Enable, Chip_Select);
   and inst2 (CS, ~Write_Enable, Chip_Select);

   Decoder_5to32 inst3 (Address, RS);
   Memory_8x8 inst4 (Out, Data, RS[7:0], WE);
   Memory_8x8 inst5 (Out, Data, RS[15:8], WE);
   Memory_8x8 inst6 (Out, Data, RS[23:16], WE);
   Memory_8x8 inst7 (Out, Data, RS[31:24], WE);


   bufif1 inst8 (Output[0], Out[0], CS);
   bufif1 inst9 (Output[1], Out[1], CS);
   bufif1 inst10 (Output[2], Out[2], CS);
   bufif1 inst11 (Output[3], Out[3], CS);
   bufif1 inst12 (Output[4], Out[4], CS);
   bufif1 inst13 (Output[5], Out[5], CS);
   bufif1 inst14 (Output[6], Out[6], CS);
   bufif1 inst15 (Output[7], Out[7], CS);

endmodule
```

In Verilog code we also design modules, and input/output signals like the above architecture.

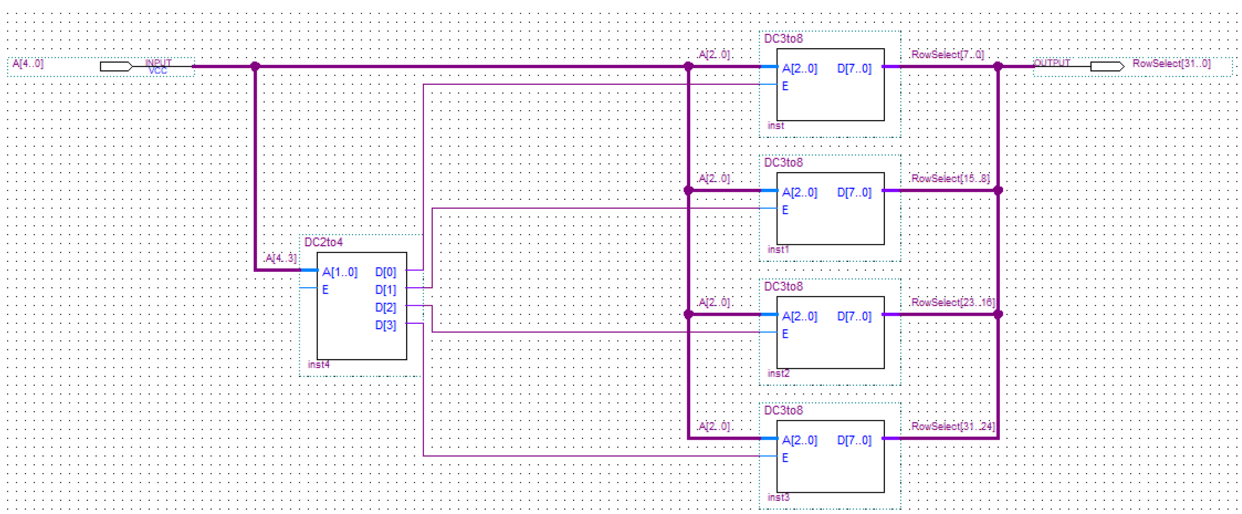
c. RTL after synthesis:



We see it is similar to the previously designed architecture.

- **Decoder_5to32:**

a. Architecture:



- To control the registers through the Row_select signal, we use the decoder 5 to 32. Thus, the Ram bar will now have 32 addresses represented by 5 bits.

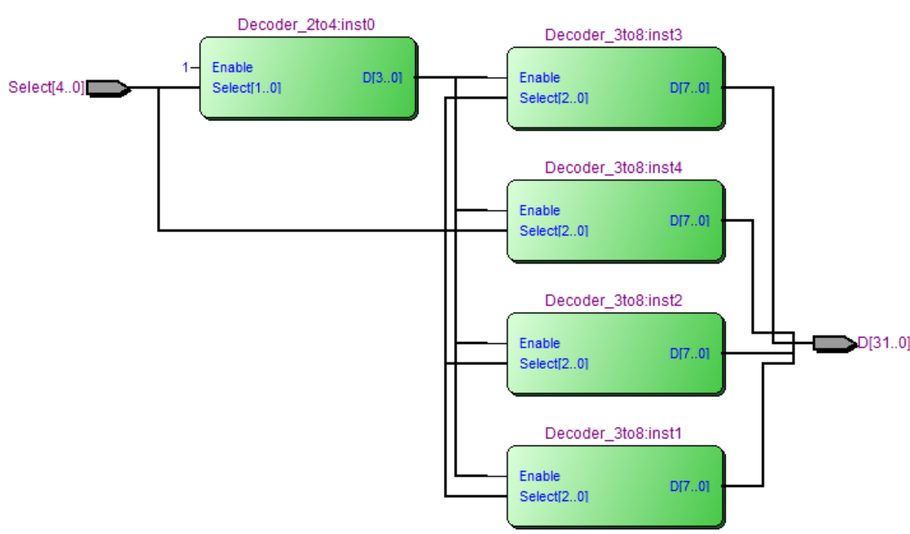
- Components of decoder 5 to 32: We design decoder 5 to 32 with 4 decoders 3 to 8 and will control them with 1 decoder 2 to 4 as shown above:

b. Verilog code:

```
module Decoder_5to32(Select, D);
//define_IO
   input  [4:0] Select;
   output [31:0] D;

//define_wire/reg
   wire [3:0] D24;

//body
   Decoder_2to4 inst0 (Select[4:3], 1, D24[3:0]);
   Decoder_3to8 inst1 (Select[2:0], D24[0], D[7:0]);
   Decoder_3to8 inst2 (Select[2:0], D24[1], D[15:8]);
   Decoder_3to8 inst3 (Select[2:0], D24[2], D[23:16]);
   Decoder_3to8 inst4 (Select[2:0], D24[3], D[31:24]);

endmodule

```

In Verilog code we also design modules, and input/output signals like the above architecture.

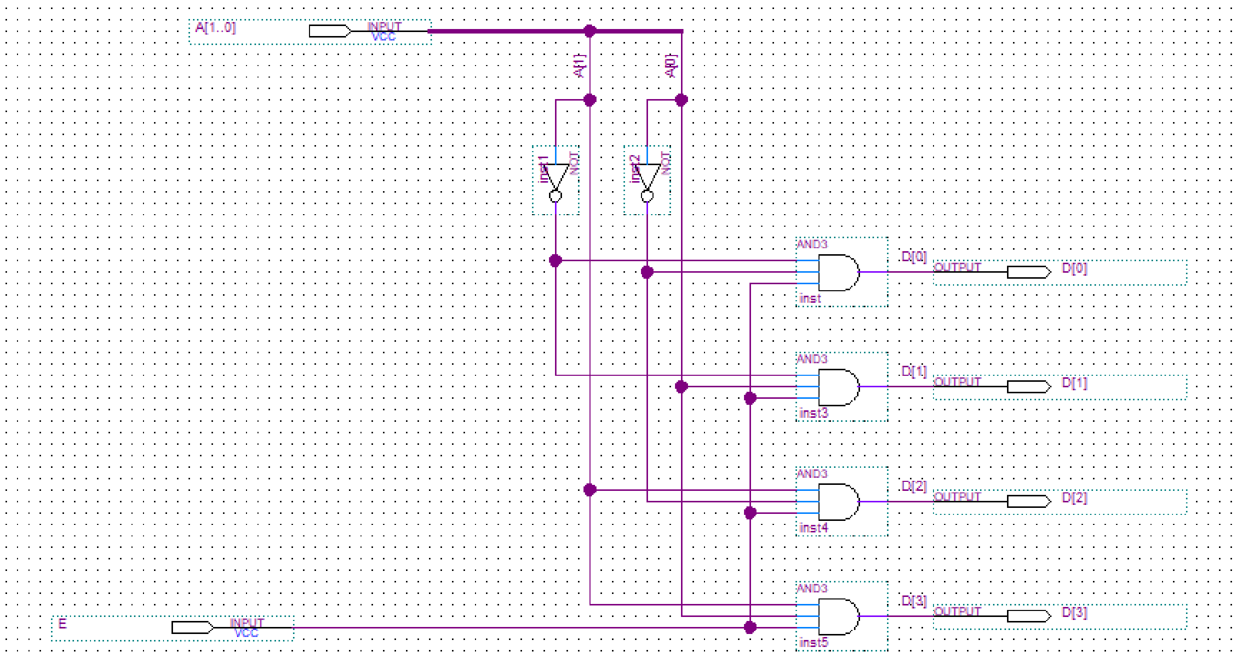
c. RTL:



We see it is similar to the previously designed architecture.

- ***Decoder_2to4:***

a. Architecture:



Truth table:

Input				Output			
E	A1	A0		D3	D2	D1	D0
0	x	x		0	0	0	0
1	0	0		0	0	0	1
1	0	1		0	0	1	0
1	1	1		0	1	0	0
1	1	0		1	0	0	0

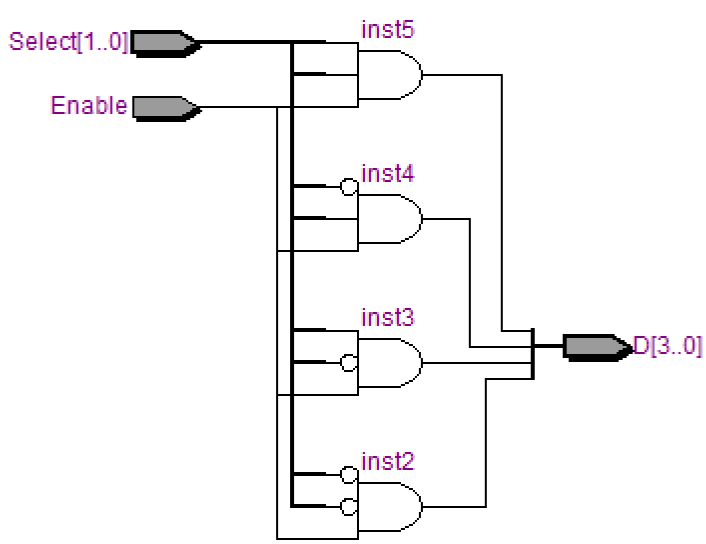
D0 = E.A1’.A0’; D1 = E.A1’.A0; D2 = E.A1.A0; D3 = E.A1.A0’;

b. Verilog code:

```
module Decoder_2to4(Select, Enable, D);
//define_IO
   input [1:0] Select;
   input Enable;
   output [3:0] D;

//define_wire/reg

//body
   and inst2 (D[0], ~Select[0], ~Select[1], Enable);
   and inst3 (D[1], Select[0], ~Select[1], Enable);
   and inst4 (D[2], ~Select[0], Select[1], Enable);
   and inst5 (D[3], Select[0], Select[1], Enable);

endmodule

```

In Verilog code we also design logic gates and input/output signals like the above architecture.

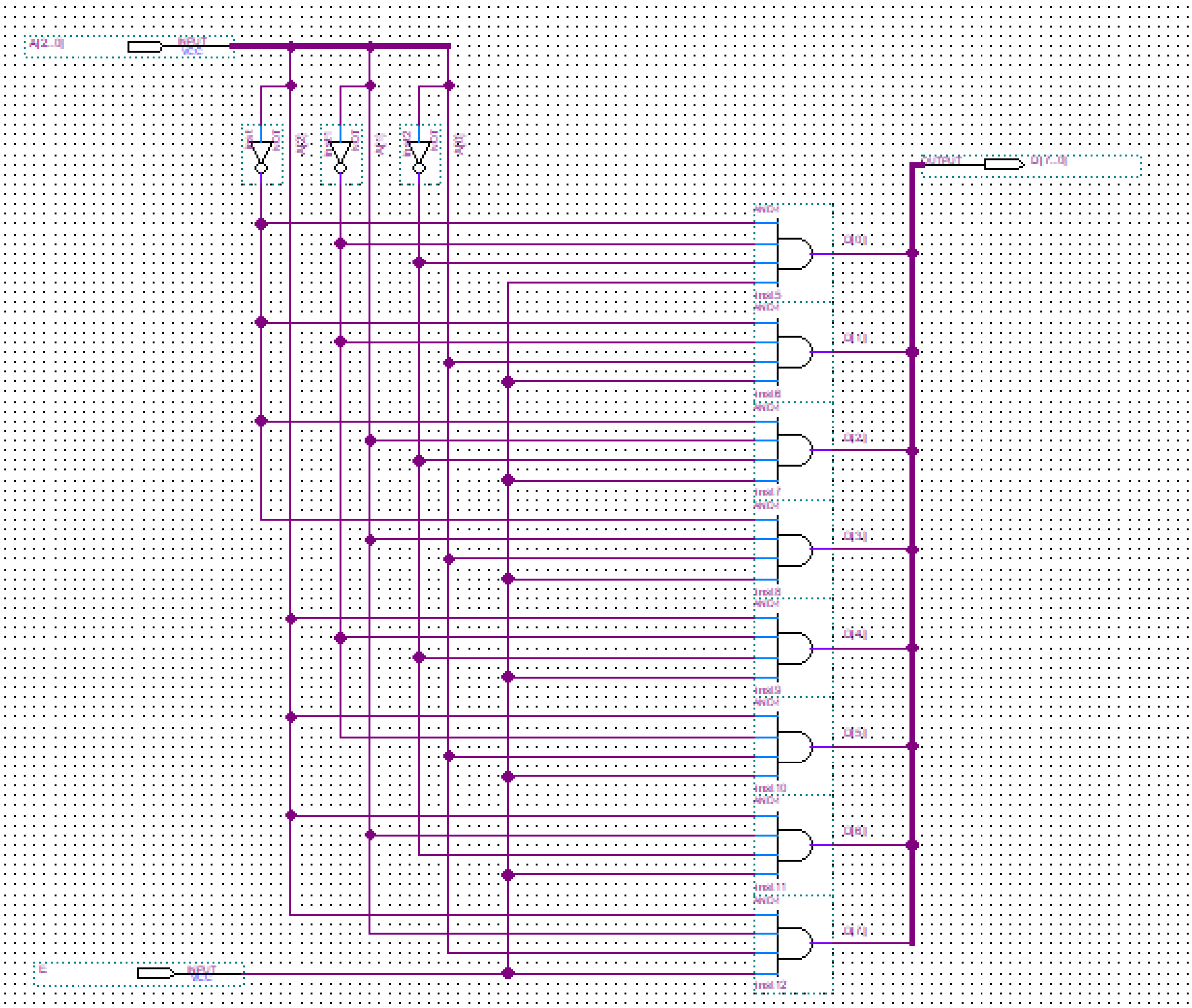
c. RTL:



We see it is similar to the previously designed architecture.

- *Decoder_3to8:*

a. Architecture:



Truth table:

Input					Output							
E	A2	A1	A0		D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	x		0	0	0	0	0	0	0	0
1	0	0	0		0	0	0	0	0	0	0	1
1	0	0	1		0	0	0	0	0	0	1	0
1	0	1	1		0	0	0	0	0	1	0	0
1	0	1	0		0	0	0	0	1	0	0	0
1	1	0	0		0	0	0	1	0	0	0	0
1	1	0	1		0	0	1	0	0	0	0	0
1	1	1	1		0	1	0	0	0	0	0	0
1	1	1	0		1	0	0	0	0	0	0	0

D0 = E.A2’.A1’.A0’; D1 = E.A2’.A1’.A0; D2 = E.A2’.A1.A0’; D3 = E.A2’.A1.A0;

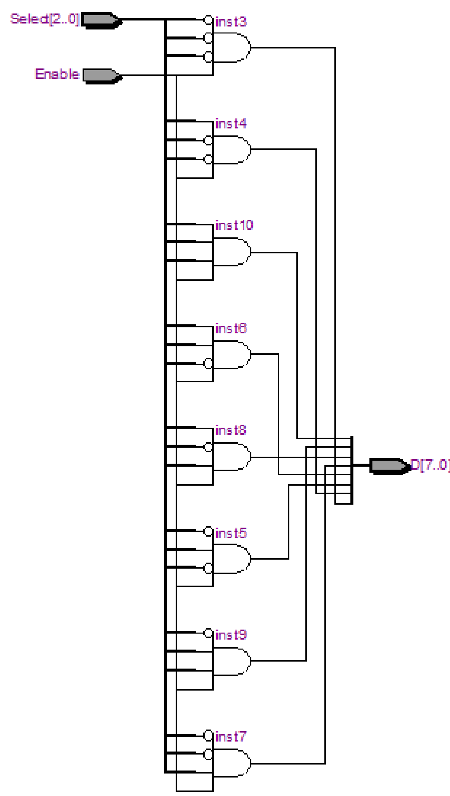
D4 = E.A2.A1’.A0’; D5 = E.A2.A1’.A0; D6 = E.A2.A1.A0’; D7 = E.A2.A1.A0;

b. Verilog code:

```
module Decoder_3to8(Select, Enable, D);
//define_IO
   input [2:0] Select;
   input Enable;
   output [7:0] D;

//define_wire/reg

//body
   and inst3 (D[0], ~Select[0], ~Select[1], ~Select[2], Enable);
   and inst4 (D[1], Select[0], ~Select[1], ~Select[2], Enable);
   and inst5 (D[2], ~Select[0], Select[1], ~Select[2], Enable);
   and inst6 (D[3], Select[0], Select[1], ~Select[2], Enable);
   and inst7 (D[4], ~Select[0], ~Select[1], Select[2], Enable);
   and inst8 (D[5], Select[0], ~Select[1], Select[2], Enable);
   and inst9 (D[6], ~Select[0], Select[1], Select[2], Enable);
   and inst10 (D[7], Select[0], Select[1], Select[2], Enable);

endmodule
```

In Verilog code we also design logic gates and input/output signals like the above architecture.

c. RTL:



We see it is similar to the previously designed architecture.

- Memory_8x8:

a. Architecture:

b. Verilog code:

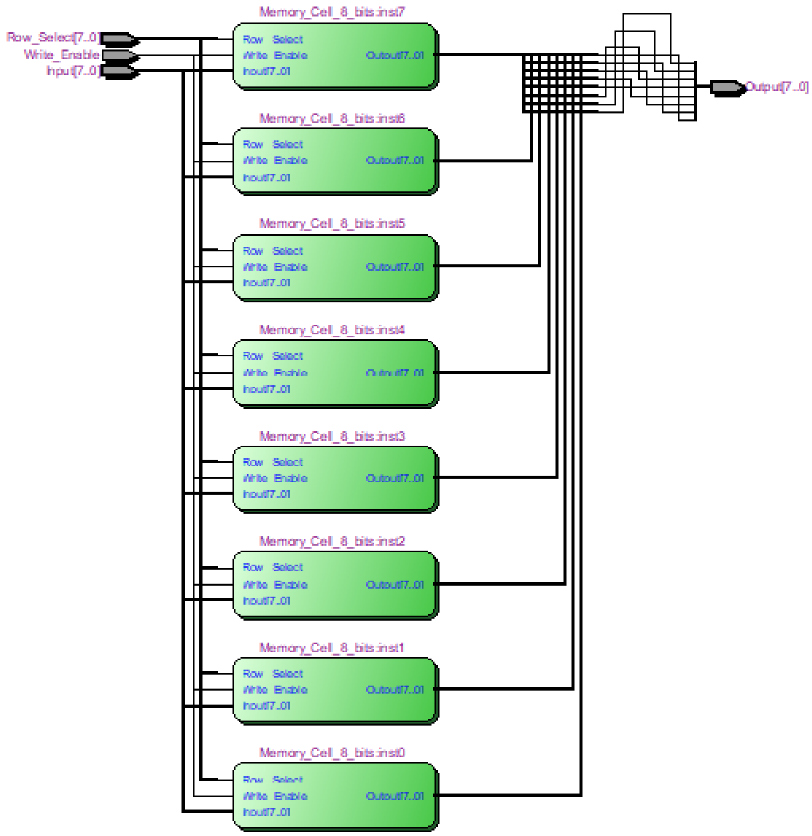
For design convenience, we first design Memory_Cell_8_bits, then combine the 8 newly designed registers to form Memory_8x8.

- Memory_8x8: it includes 8 Memory_Cell_8_bit:

a. Code:

```
module Memory_8x8(Output, Input, Row_Select, Write_Enable);
//define_IO
   input [7:0] Input, Row_Select;
   input Write_Enable;
   output [7:0] Output;

//define_wire/reg

//body
   Memory_Cell_8_bits inst0 (Output, Input, Row_Select[0], Write_Enable);
   Memory_Cell_8_bits inst1 (Output, Input, Row_Select[1], Write_Enable);
   Memory_Cell_8_bits inst2 (Output, Input, Row_Select[2], Write_Enable);
   Memory_Cell_8_bits inst3 (Output, Input, Row_Select[3], Write_Enable);
   Memory_Cell_8_bits inst4 (Output, Input, Row_Select[4], Write_Enable);
   Memory_Cell_8_bits inst5 (Output, Input, Row_Select[5], Write_Enable);
   Memory_Cell_8_bits inst6 (Output, Input, Row_Select[6], Write_Enable);
   Memory_Cell_8_bits inst7 (Output, Input, Row_Select[7], Write_Enable);
endmodule
```

b. RTL:



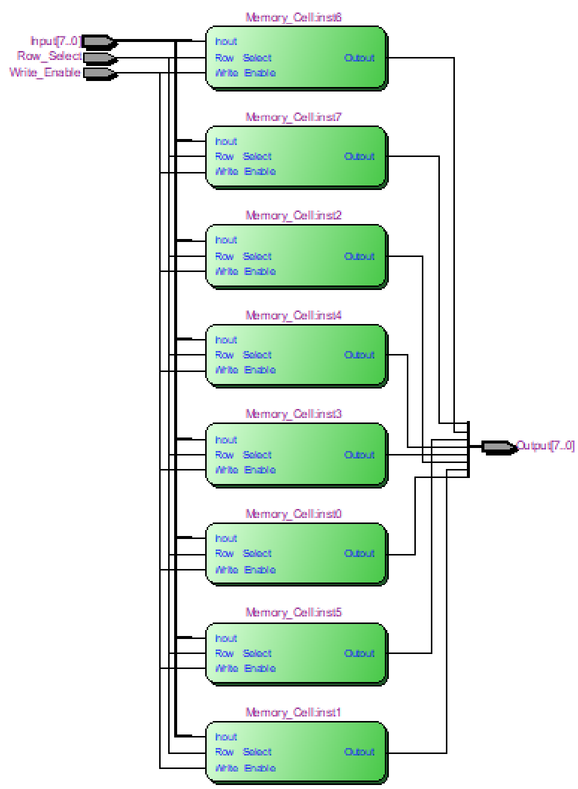
We see it is similar to the previously designed architecture.

Memoy_Cell_8_bit: It includes 8 Memory Cell:

a. Code:

```verilog
module Memory_Cell_8_bits(Output, Input, Row_Select, Write_Enable);
//define_IO
   input [7:0] Input;
   input Row_Select, Write_Enable;
   output [7:0] Output;

//define_wire/reg

//body
   Memory_Cell inst0 (Output[0], Input[0], Row_Select, Write_Enable);
   Memory_Cell inst1 (Output[1], Input[1], Row_Select, Write_Enable);
   Memory_Cell inst2 (Output[2], Input[2], Row_Select, Write_Enable);
   Memory_Cell inst3 (Output[3], Input[3], Row_Select, Write_Enable);
   Memory_Cell inst4 (Output[4], Input[4], Row_Select, Write_Enable);
   Memory_Cell inst5 (Output[5], Input[5], Row_Select, Write_Enable);
   Memory_Cell inst6 (Output[6], Input[6], Row_Select, Write_Enable);
   Memory_Cell inst7 (Output[7], Input[7], Row_Select, Write_Enable);
endmodule
```

b. RTL:

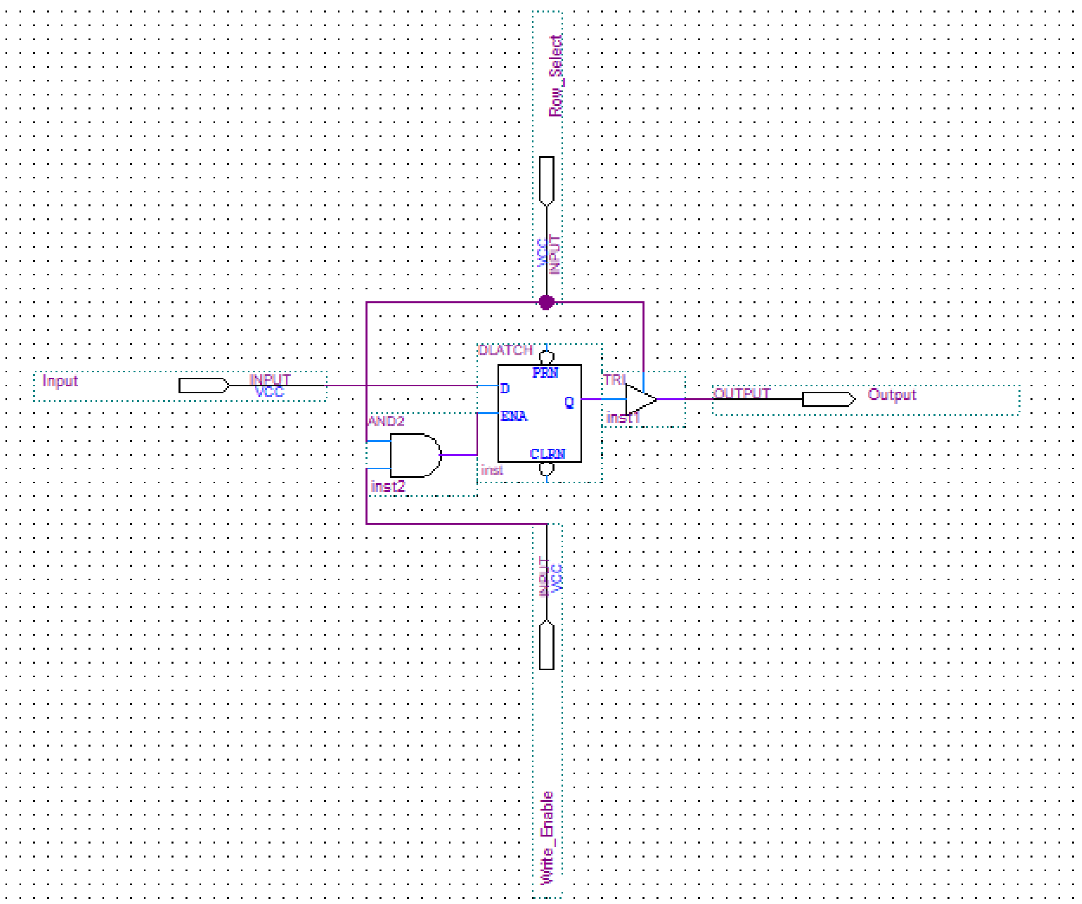


We see it is similar to the previously designed architecture.

Memory_Cell:

- *Memory cell (MC) is the smallest unit used to store data in RAM.*
- *In this design use D-latch. Buffer tri is used to control the output of MC.*
- *The input Row_Select connected to the tri-state controls the output of the MC. Combined with Write_Enable to control read and write state.*

a. Architecture:

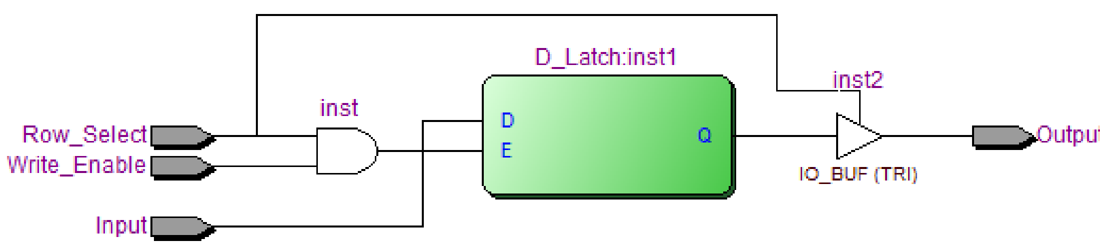


b. Verilog code:

```
module Memory_Cell(Output, Input, Row_Select, Write_Enable);
//define_IO
   input Input, Row_Select, Write_Enable;
   output Output;

//define_wire/reg
   wire Q, ENA;

//body
   and inst (ENA, Row_Select, Write_Enable);
   D_Latch inst1 (Q, Input, ENA);
   bufif1 inst2 (Output, Q, Row_Select);
endmodule
```

In Verilog code we also design logic gates and input/output signals like the above architecture.

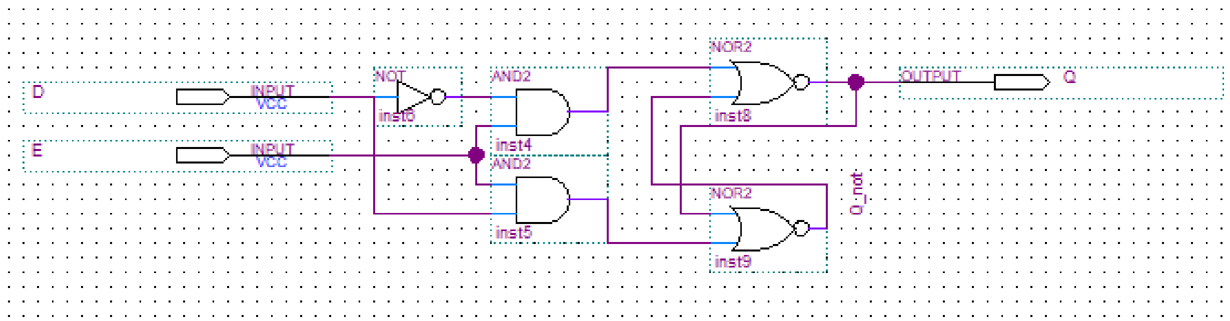
c. RTL:



We see it is similar to the previously designed architecture.

D_Latch:

a. Architecture:

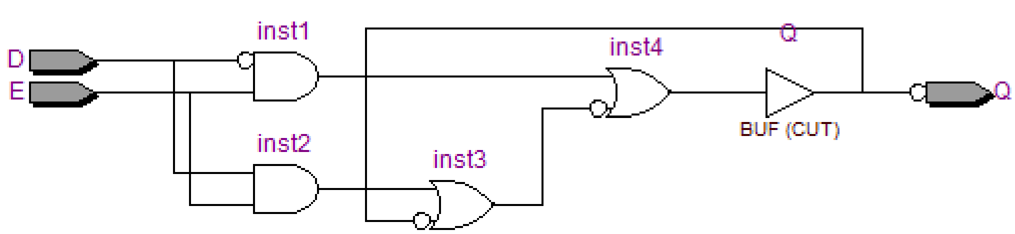


b. Verilog code:

```
module D_Latch(Q, D, E);
//define_IO
   input D, E;
   output Q;

//define_wire/reg
   wire A1, A2, Q_not;

//body
   and inst1 (A1, ~D, E);
   and inst2 (A2, D, E);
   nor inst3 (Q_not, A2, Q);
   nor inst4 (Q, A1, Q_not);
endmodule
```

In Verilog code we also design logic gates and input/output signals like the above architecture.

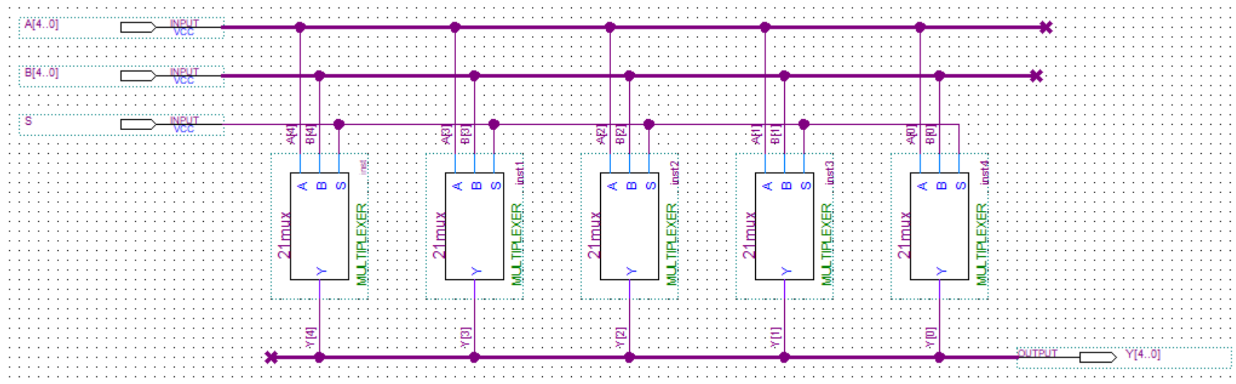
c. RTL:



We see it is similar to the previously designed architecture.

2. Mux_2_1_5b: connect 5 block Mux_2_1:

a. Architecture:

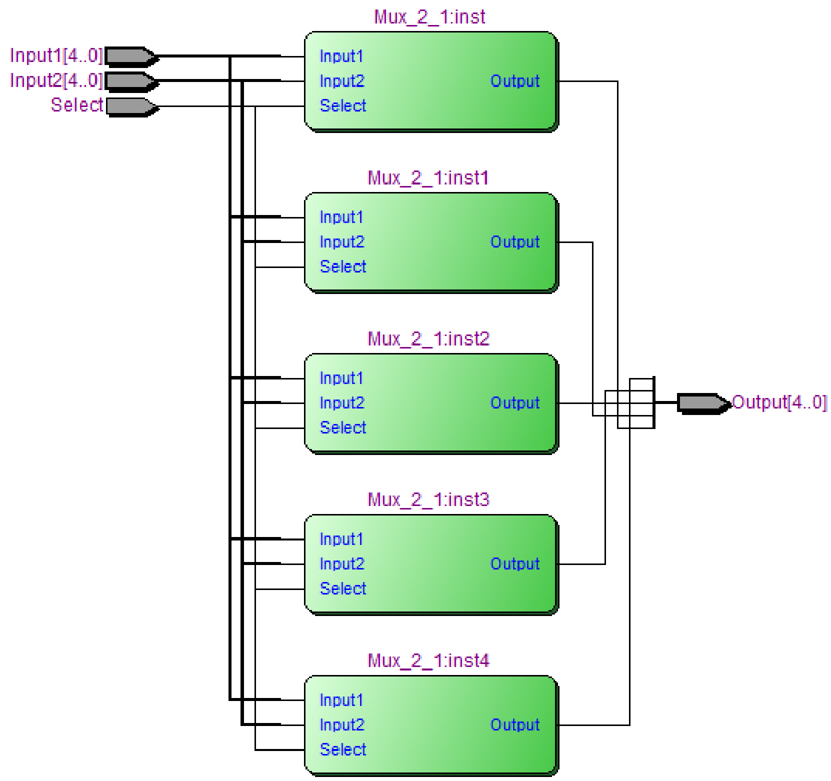


b. Verilog code

```
module Mux_2_1_5b(Output, Input1, Input2, Select);
//define_IO
   input [4:0] Input1, Input2;
   input Select;
   output [4:0] Output;

//define_wire/reg

//body
   Mux_2_1 inst  (Output[0], Input1[0], Input2[0], Select);
   Mux_2_1 inst1 (Output[1], Input1[1], Input2[1], Select);
   Mux_2_1 inst2 (Output[2], Input1[2], Input2[2], Select);
   Mux_2_1 inst3 (Output[3], Input1[3], Input2[3], Select);
   Mux_2_1 inst4 (Output[4], Input1[4], Input2[4], Select);

endmodule

```

In Verilog code we also design modules, and input/output signals like the above architecture.

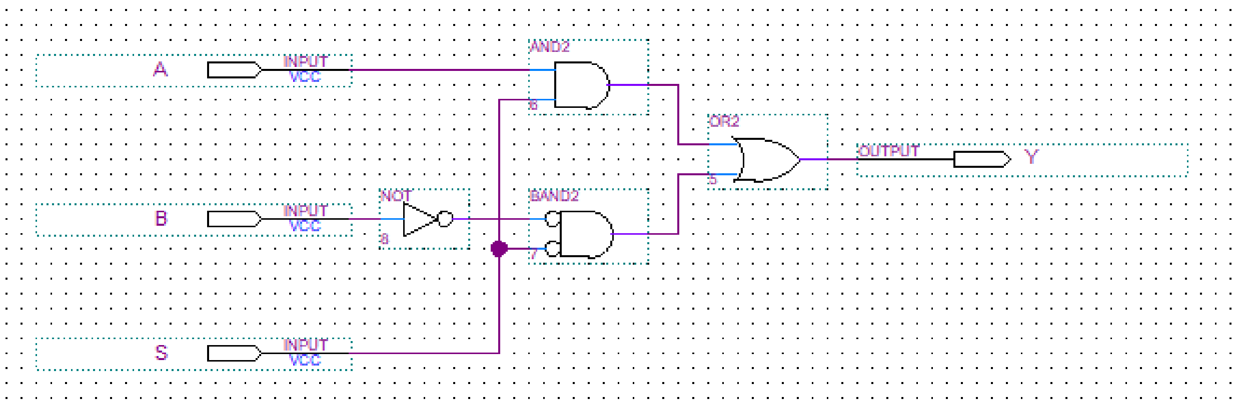
c. RTL:



We see it is similar to the previously designed architecture.

Mux_2_1_1b:

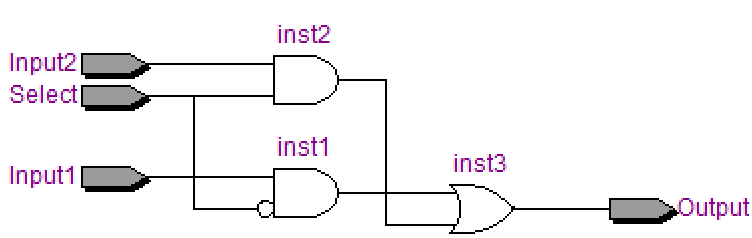
a. Architecture:



b. Verilog code:

```
module Mux_2_1(Output, Input1, Input2, Select);
//define_IO
   input Input1, Input2, Select;
   output Output;

//define_wire/reg
   wire A1, A2;

//body
   and inst1 (A1, ~Select, Input1);
   and inst2 (A2, Select, Input2);

   or inst3 (Output, A1, A2);

endmodule
```

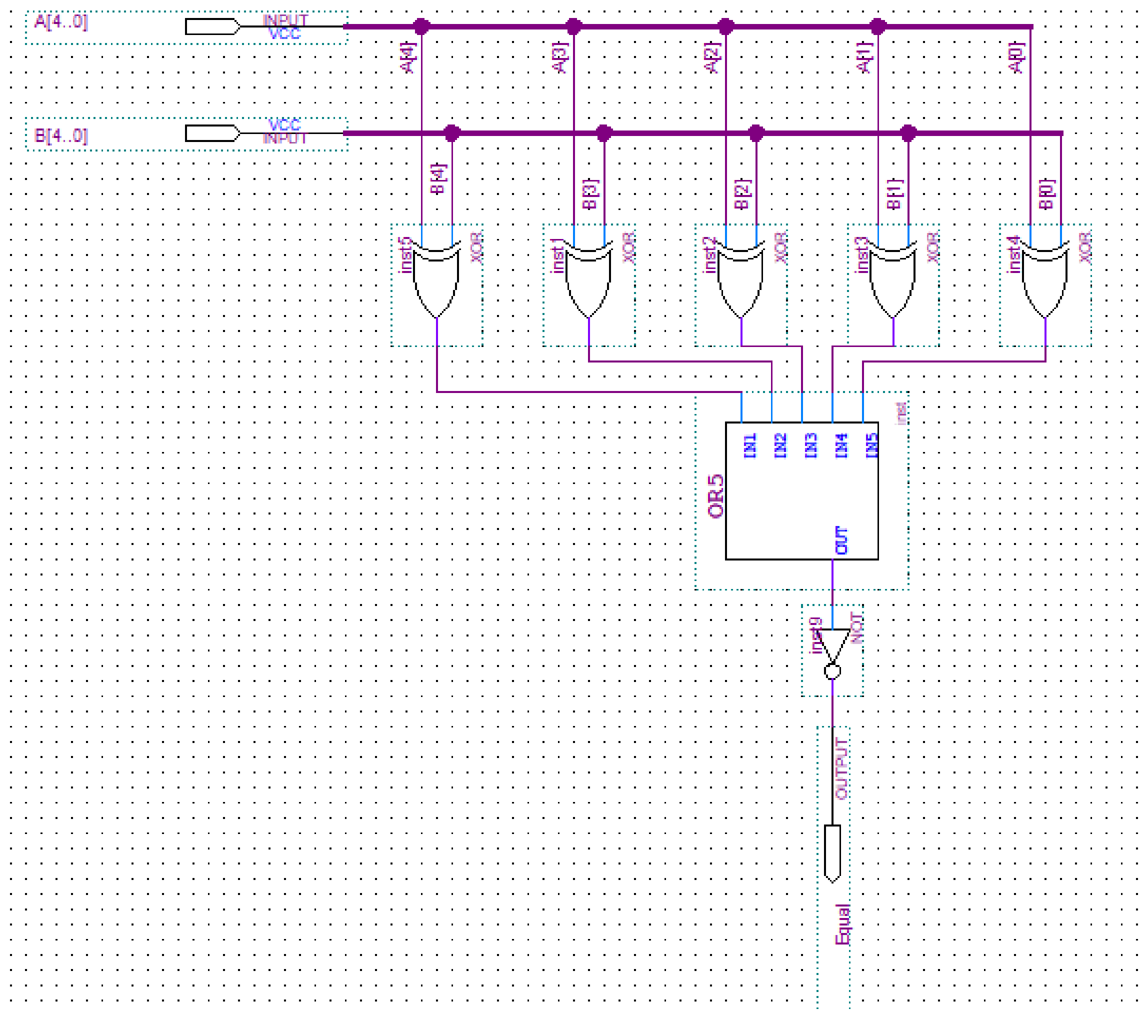
c. RTL:



We see it is similar to the previously designed architecture.

3. Equal_5b: we use xor2 gate, if 2 input of xor2 gate is the same, the output will equal 0, so we have a not gate after output of xor2 gate, the purpose is if 2 input is the same, the final output is 1.

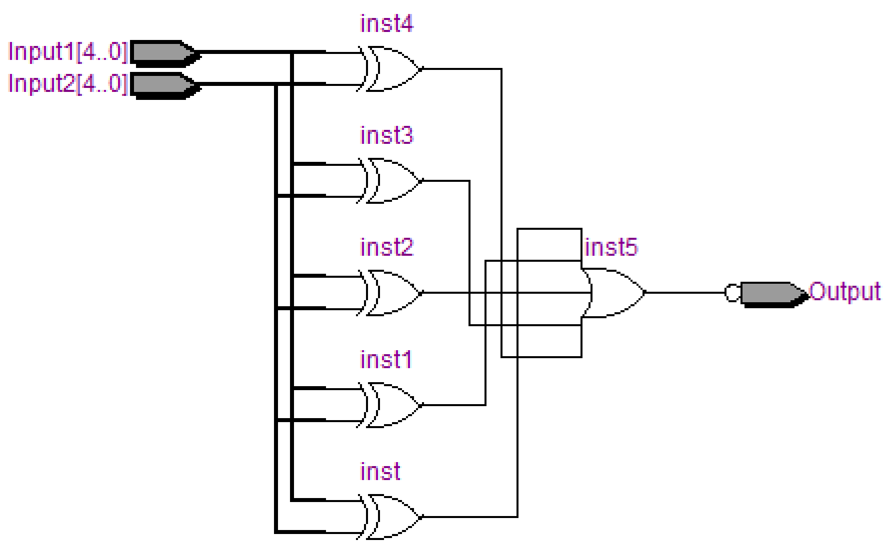
a. *Architecture:*



b. Verilog code:

```
module Equal_5b (Output, Input1, Input2); //if Input1 = Input2 => Output = 1
//define_IO
   input  [4:0] Input1, Input2;
   output Output;

//define_wire/reg
   wire [4:0] XOR;

//body
   xor inst (XOR[0], Input1[0], Input2[0]);
   xor inst1 (XOR[1], Input1[1], Input2[1]);
   xor inst2 (XOR[2], Input1[2], Input2[2]);
   xor inst3 (XOR[3], Input1[3], Input2[3]);
   xor inst4 (XOR[4], Input1[4], Input2[4]);

   nor inst5 (Output, XOR[0], XOR[1], XOR[2], XOR[3], XOR[4]);
endmodule
```

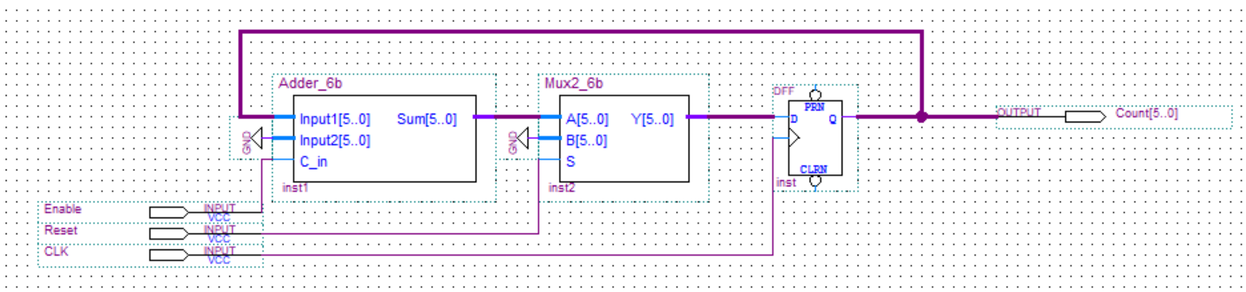
c. RTL:



We see it is similar to the previously designed architecture.

4. Up_Counter:

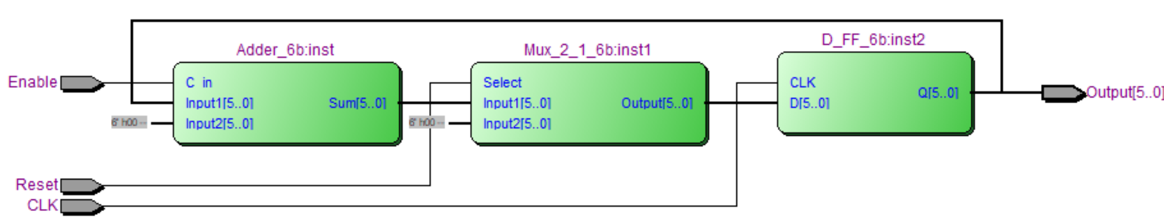
a. Architecture:



b. Verilog code:

```verilog
module Up_Counter (Output, Reset, CLK, Enable);
//define_IO
   input CLK, Reset, Enable;
   output [5:0] Output;

//define_wire/reg
   wire [5:0] Count, A;

//body
   Adder_6b inst (Count, Output, 0, Enable);
   Mux_2_1_6b inst1 (A, Count, 0, Reset);
   D_FF_6b inst2 (Output, A, CLK);

endmodule

```

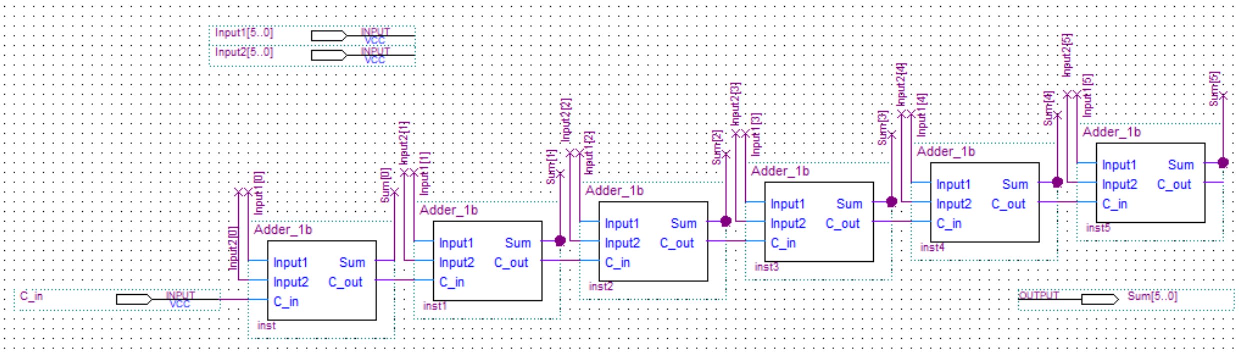
c. RTL:



We see it is similar to the previously designed architecture.

I. Add_6b:

a. Architecture:

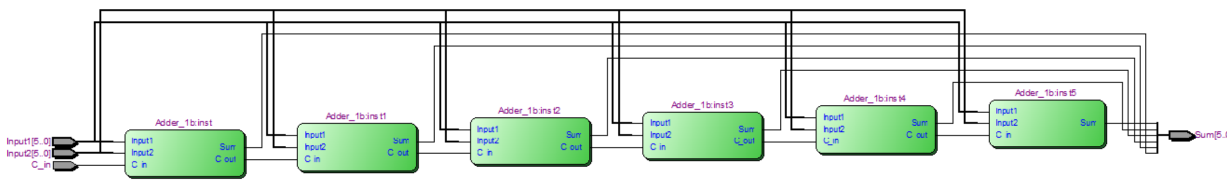


b. Verilog code:

```
module Adder_6b(Sum, Input1, Input2, C_in);
//define_IO
   input [5:0] Input1, Input2;
   input C_in;
   output [5:0] Sum;

//define_wire/reg
   wire [5:0] C;

//body
   Adder_1b inst (Sum[0], C[0], Input1[0], Input2[0], C_in);
   Adder_1b inst1 (Sum[1], C[1], Input1[1], Input2[1], C[0]);
   Adder_1b inst2 (Sum[2], C[2], Input1[2], Input2[2], C[1]);
   Adder_1b inst3 (Sum[3], C[3], Input1[3], Input2[3], C[2]);
   Adder_1b inst4 (Sum[4], C[4], Input1[4], Input2[4], C[3]);
   Adder_1b inst5 (Sum[5], C[5], Input1[5], Input2[5], C[4]);

endmodule
```

In Verilog code we also design modules, and input/output signals like the above architecture.

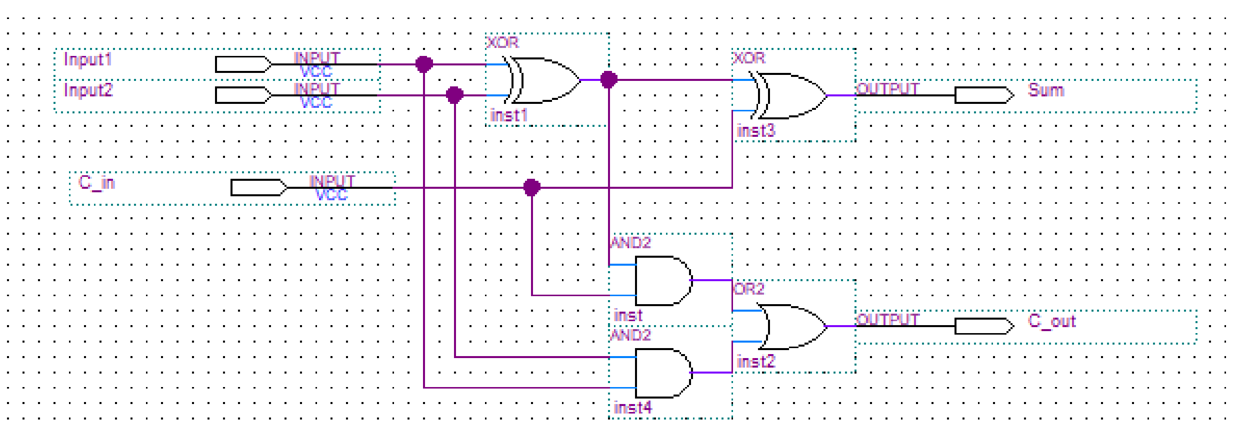
c. RTL:



We see it is similar to the previously designed architecture.

Add_1b:

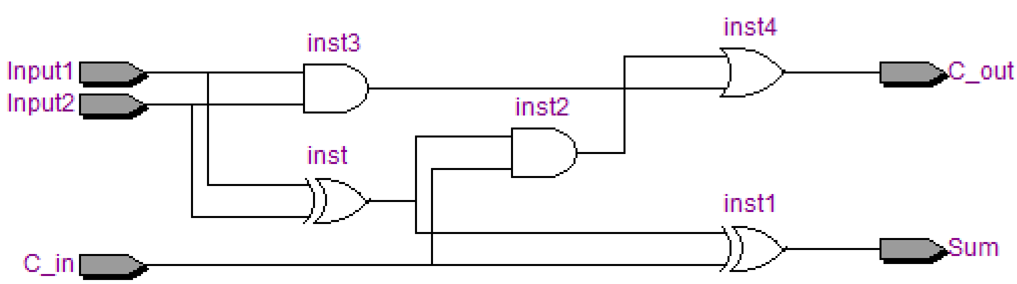
a. Architecture:



b. Verilog code:

```
module Adder_1b (Sum, C_out, Input1, Input2, C_in);
//define_IO
   input Input1, Input2, C_in;
   output Sum, C_out;

//define_wire/reg
   wire A, B, C;

//body
   xor inst (A, Input1, Input2);
   xor inst1 (Sum, A, C_in);

   and inst2 (B, A, C_in);
   and inst3 (C, Input1, Input2);
   or inst4 (C_out, B, C);

endmodule
```

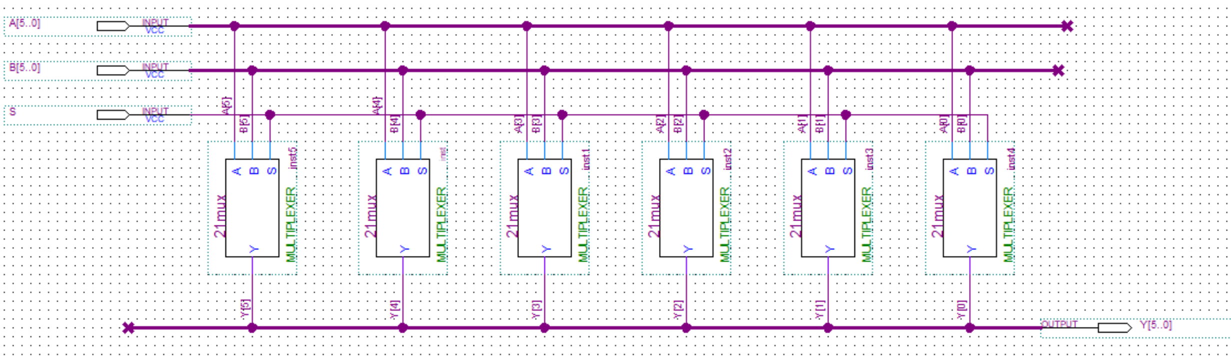
c. RTL:



We see it is similar to the previously designed architecture.

II. Mux_2_1_6b:

a. Architecture:

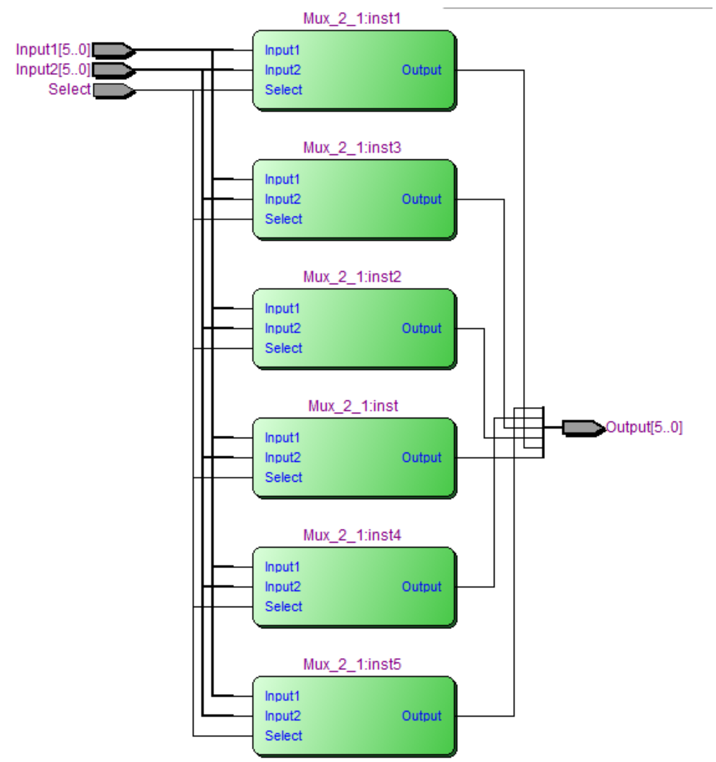


b. Verilog code:

```
module Mux_2_1_6b(Output, Input1, Input2, Select);
//define_IO
   input [5:0] Input1, Input2;
   input Select;
   output [5:0] Output;

//define_wire/reg

//body
   Mux_2_1 inst  (Output[0], Input1[0], Input2[0], Select);
   Mux_2_1 inst1 (Output[1], Input1[1], Input2[1], Select);
   Mux_2_1 inst2 (Output[2], Input1[2], Input2[2], Select);
   Mux_2_1 inst3 (Output[3], Input1[3], Input2[3], Select);
   Mux_2_1 inst4 (Output[4], Input1[4], Input2[4], Select);
   Mux_2_1 inst5 (Output[5], Input1[5], Input2[5], Select);

endmodule
```

In Verilog code we also design modules, and input/output signals like the above architecture.

c. RTL:



We see it is similar to the previously designed architecture

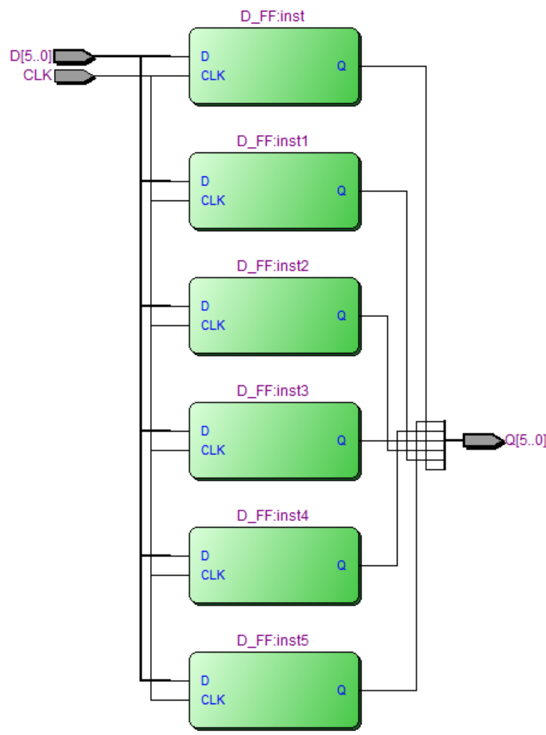
III. D_FF_6b:

a. Verilog code:

```
module D_FF_6b(Q, D, CLK);
//define_IO
   input [5:0] D;
   input CLK;
   output [5:0] Q;

//define_wire/reg

//body
   D_FF inst  (Q[0], D[0], CLK);
   D_FF inst1 (Q[1], D[1], CLK);
   D_FF inst2 (Q[2], D[2], CLK);
   D_FF inst3 (Q[3], D[3], CLK);
   D_FF inst4 (Q[4], D[4], CLK);
   D_FF inst5 (Q[5], D[5], CLK);
endmodule
```

In Verilog code we also design modules, and input/output signals like the above architecture.

b. *RTL:*



We see it is similar to the previously designed architecture.