30 Days of RTL Coding

-By T Tushar Shenoy

Day 13

Problem Statement: Implementing a 4 to 2 Encoder**,** Octal Binary Encoder (8 to 3 Encoder)**,** Decimal to BCD Encoder**,** Priority Encoder using Structural Implementation.

Theory:

An encoder is a digital circuit that converts a set of binary inputs into a unique binary code. The binary code represents the position of the input and is used to identify the specific input that is active. Encoders are commonly used in digital systems to convert a parallel set of inputs into a serial code.

The basic principle of an encoder is to assign a unique binary code to each possible input. For example, a 4-to-2 line encoder has 4 input lines and 2 output lines and assigns a unique 2-bit binary code to each of the 2^2 = 4 possible input combinations.There are different types of encoders, including priority encoders, which assign a priority to each input, and binary-weighted encoders, which use a binary weighting system to assign binary codes to inputs. In summary, an encoder is a digital circuit that converts a set of binary inputs into a unique binary code that represents the position of the input. Encoders are widely used in digital systems to convert parallel inputs into serial codes.

An Encoder is a **combinational circuit** that performs the reverse operation of a Decoder. It has a maximum of **2^n input lines** and **‘n’ output lines**, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with ‘n’ bits.

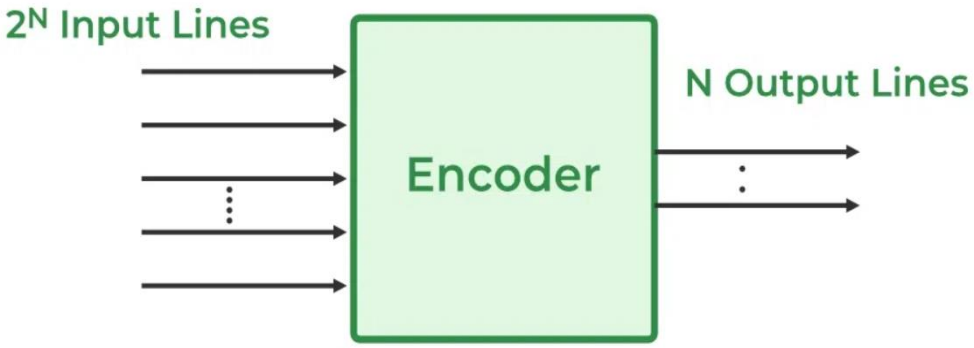


FIG: Encoder Block Diagram

1. 4 to 2 Encoder

The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0**. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.

FIG: 4x2 Encoder Block Diagram

INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

FIG: 4x2 Encoder Truth Table

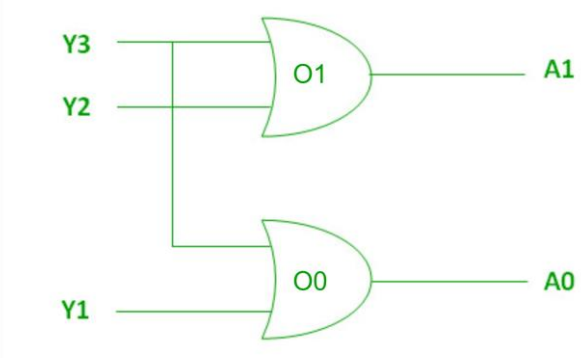


FIG: 4x2 Encoder Logic Diagram

Verilog Code:

```
// Verilog Code for 4x2 Encoder
module Encoder4x2(Y,A);

input [3:0]Y;
output [1:0]A;

or O1(A[1],Y[3],Y[2]);
or O0(A[0],Y[3],Y[1]);

endmodule
```

Testbench Code:

```
//Testbench for 4x2 Encoder
module Encoder4x2_tb();

reg [3:0]Y;
wire [1:0]A;

Encoder4x2 dut(.Y(Y),.A(A));

initial begin
   Y=4'b0001;
#5 Y=4'b0010;
#5 Y=4'b0100;
#5 Y=4'b1000;
#5 $finish;

end
endmodule
```

Schematic:

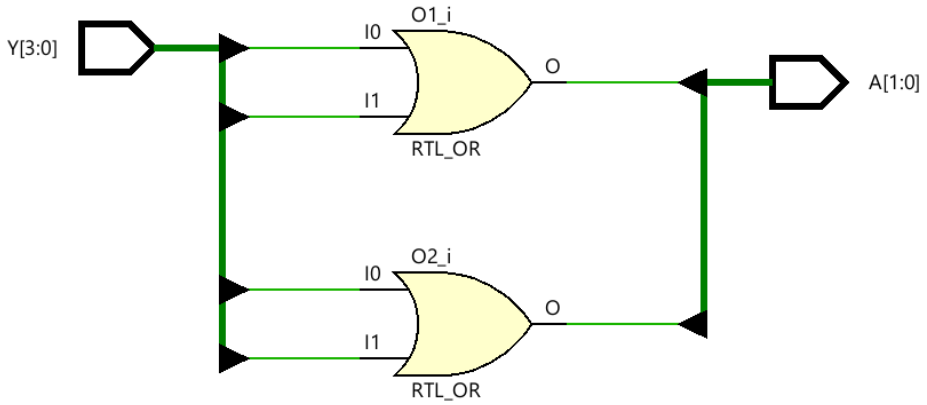
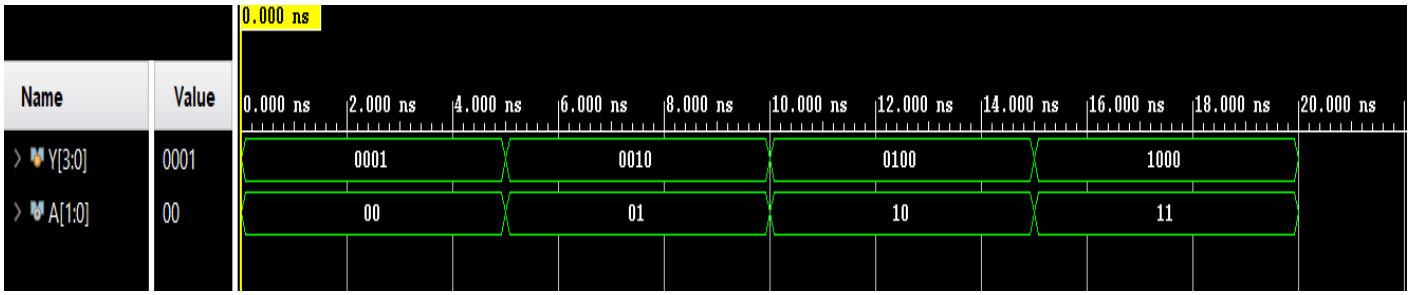


FIG: 4x2 Encoder Schematic

Simulation Output:



2. Octal to Binary Encoder (8 to 3 Encoder)

The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: Y7 to Y0 and 3 outputs: A2, A1 & A0. Each input line corresponds to each octal digit and three outputs generate corresponding binary code.

FIG: Octal to Binary Encoder (8 to 3 Encoder) Block Diagram

INPUTS								OUTPUTS		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

FIG: Octal to Binary Encoder (8 to 3 Encoder) Truth Table

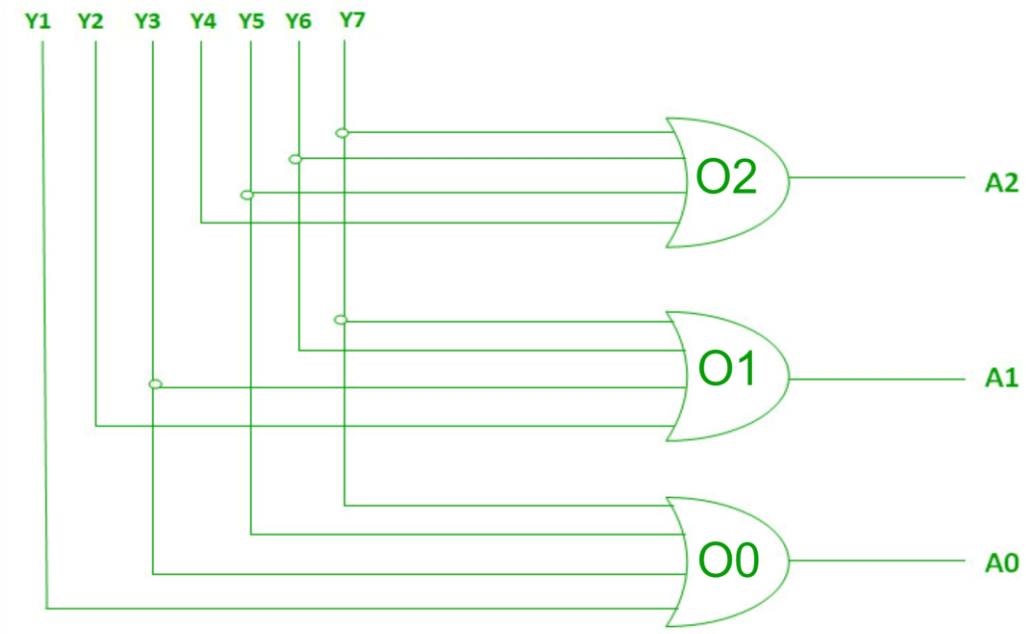


FIG: Octal to Binary Encoder (8 to 3 Encoder) Logic Diagram

Verilog Code:

```
// Verilog Code for 8x3 Encoder
module Encoder8x3(Y,A);

input [7:0]Y;
output [2:0]A;

or O2(A[2],Y[7],Y[6],Y[5],Y[4]);
or O1(A[1],Y[7],Y[6],Y[3],Y[2]);
or O0(A[0],Y[7],Y[5],Y[3],Y[1]);

endmodule
```

Testbench Code:

```
//Testbench for 8x3 Encoder
module Encoder8x3_tb();

reg [7:0]Y;
wire [2:0]A;

Encoder8x3 dut(.Y(Y),.A(A));

initial begin
   Y=8'b00000001;
#5 Y=8'b00000010;
#5 Y=8'b00000100;
#5 Y=8'b00001000;
#5 Y=8'b00010000;
#5 Y=8'b00100000;
#5 Y=8'b01000000;
#5 Y=8'b10000000;
#5 $finish;

end
endmodule
```

Schematic:

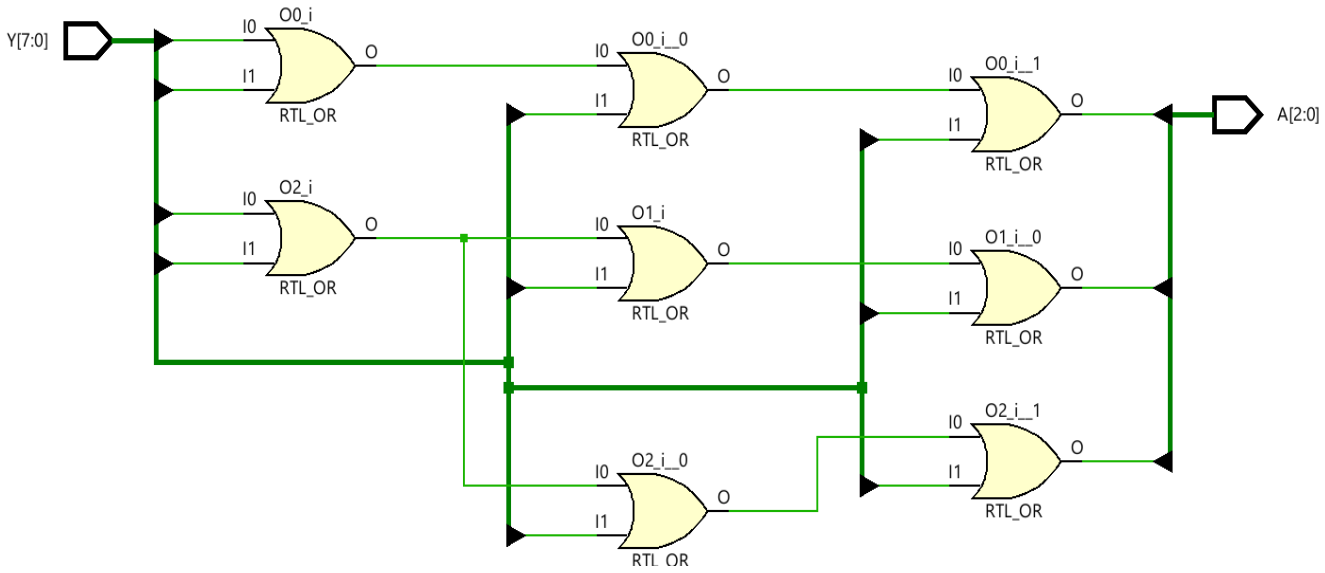
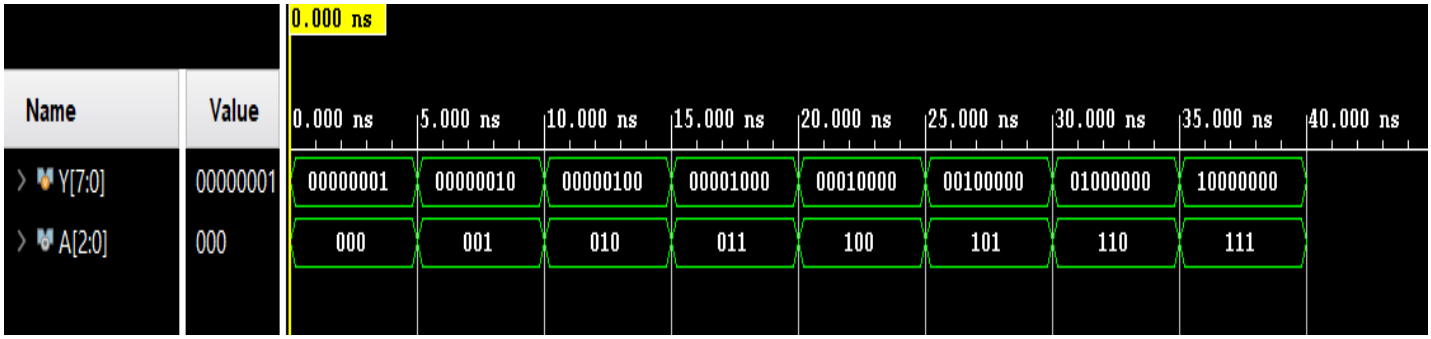


FIG: Octal to Binary Encoder (8 to 3 Encoder) Schematic

Simulation Output:



3. Decimal to BCD Encoder

The decimal-to-binary encoder usually consists of **10 input lines** and **4 output lines**. Each input line corresponds to each decimal digit and 4 outputs correspond to the BCD code. This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines.

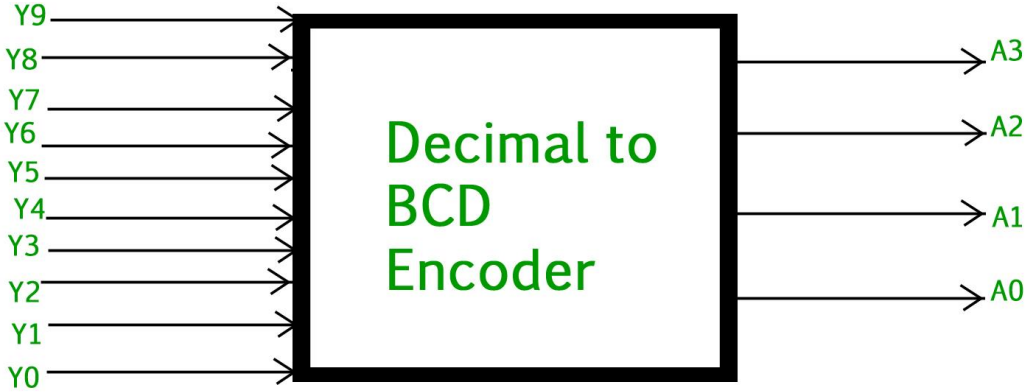


FIG: Decimal to BCD Encoder Block Diagram

INPUTS										OUTPUTS			
Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

FIG: Decimal to BCD Encoder Truth Table

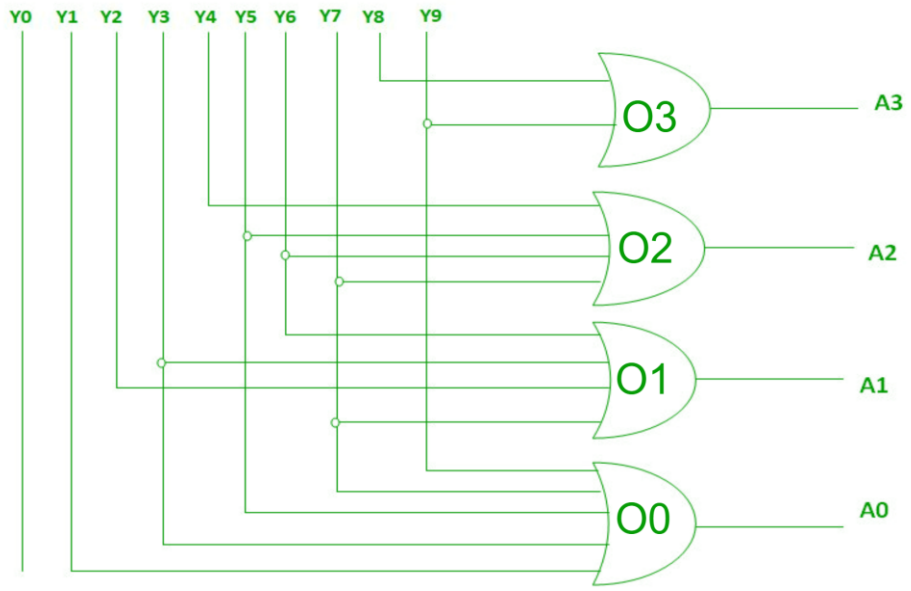


FIG: Decimal to BCD Encoder Logic Diagram

Verilog Code:

```
// Verilog Code for BCD Encoder
module Encoder_BCD(Y,A);

input [9:0]Y;
output [3:0]A;

or O3(A[3],Y[9],Y[8]);
or O2(A[2],Y[7],Y[6],Y[5],Y[4]);
or O1(A[1],Y[7],Y[6],Y[3],Y[2]);
or O0(A[0],Y[9],Y[7],Y[5],Y[3],Y[1]);

endmodule
```

Testbench Code:

```
//Testbench for BCD Encoder
module Encoder_BCD_tb();

reg [9:0]Y;
wire [3:0]A;

Encoder_BCD dut(.Y(Y),.A(A));

initial begin
   Y=10'b0000000001;
#5 Y=10'b0000000010;
#5 Y=10'b0000000100;
#5 Y=10'b0000001000;
#5 Y=10'b0000010000;
#5 Y=10'b0000100000;
#5 Y=10'b0001000000;
#5 Y=10'b0010000000;
#5 Y=10'b0100000000;
#5 Y=10'b1000000000;
#5 $finish;

end

endmodule
```

Schematic:

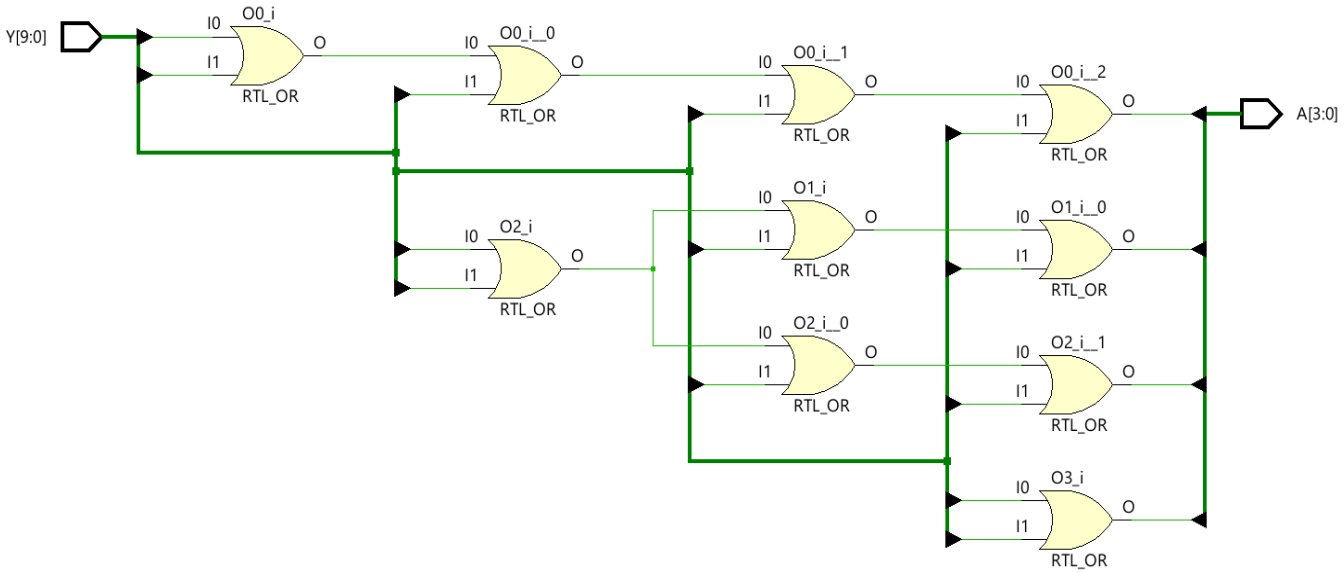
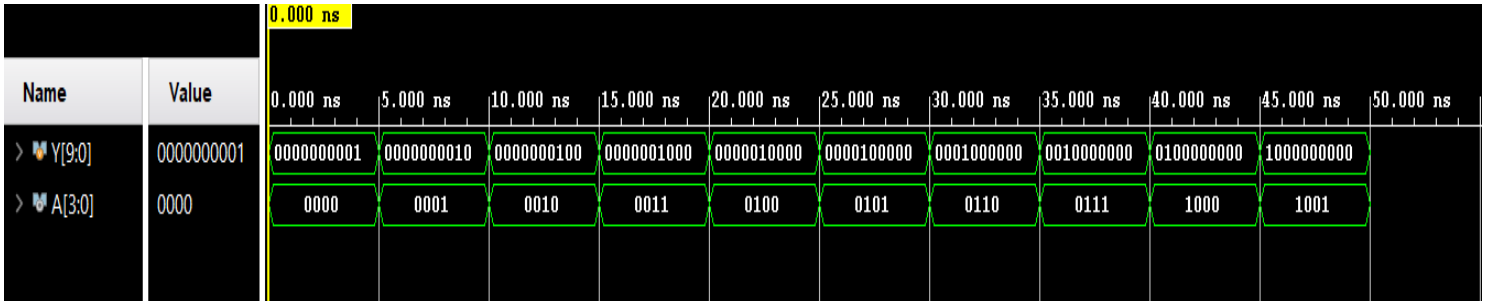


FIG: Decimal to BCD Encoder Schematic

Simulation Output:



4. Priority Encoder

A 4 to 2 priority encoder has 4 inputs: Y3, Y2, Y1 & Y0, and 2 outputs: A1 & A0. Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having higher priority.

INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

FIG: Priority Encoder Truth Table

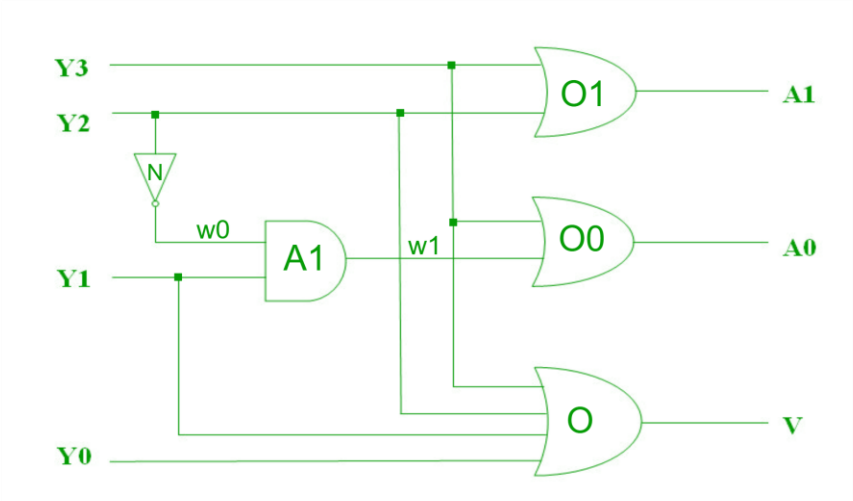


FIG: Priority Encoder Logic Diagram

Verilog Code:

```
// Verilog Code for Priority Encoder
module Priority_Encoder(Y,A,V);

input [3:0]Y;
output [1:0]A;
output V;

wire w0,w1;
not N(w0,Y[2]);

and A1(w1,w0,Y[1]);

or O1(A[1],Y[3],Y[2]);
or O0(A[0],w1,Y[3]);
or O(V,Y[3],Y[2],Y[1],Y[0]);

endmodule
```

Testbench Code:

```
//Testbench for Priority Encoderr
module Priority_Encoder_tb();

reg [3:0]Y;
wire [1:0]A;
wire V;

Priority_Encoder dut(.Y(Y),.A(A),.V(V));

initial begin
   Y=4'b0000;
#5 Y=4'b0001;
#5 Y=4'b001x;
#5 Y=4'b01xx;
#5 Y=4'b1xxx;
#5 $finish;

end

endmodule
```

Schematic:

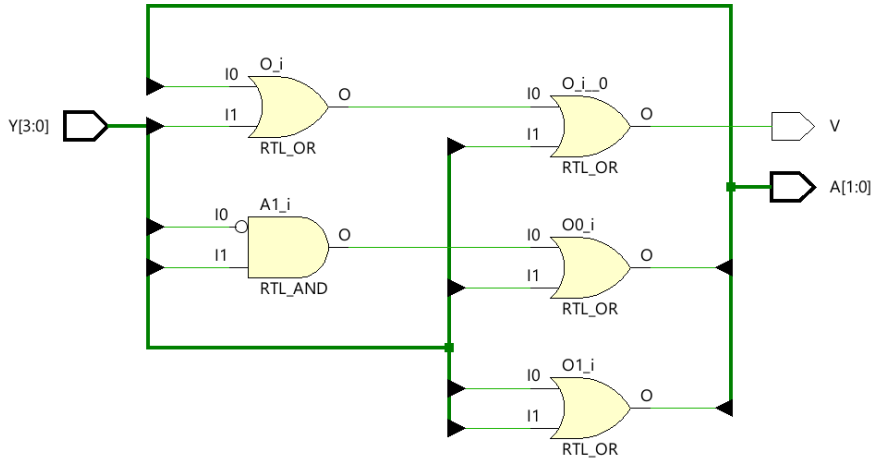
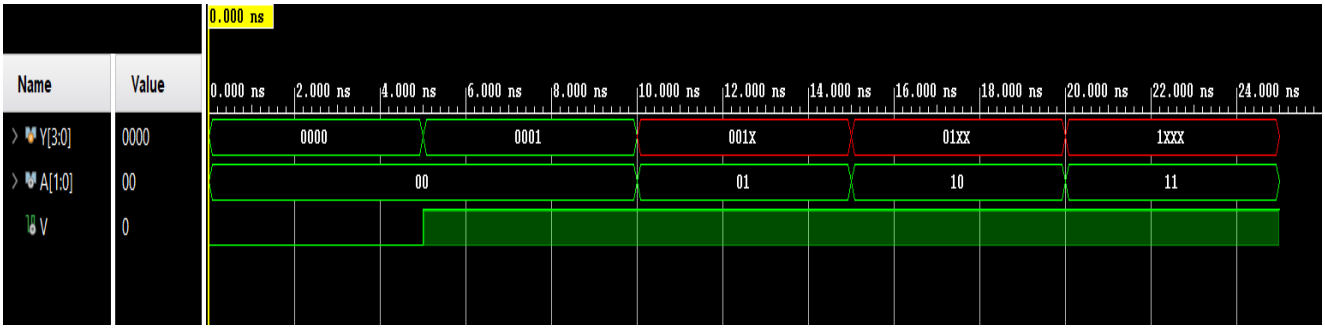


FIG: Priority Encoder Schematic

Simulation Output:



GitHub Repository URL: https://github.com/tusharshenoy/RTL-Day-13-Encoder