VLSI Engineering Lab (Digital)

Experiment 1 – 8:1 MUX and 4-bit Ripple Carry Adder

Utkarsh Patel (18EC30048)

Objective

Designing 8:1 MUX module from 2:1 MUX module where the latter is designed using behavioural and structural modelling

Designing 4-bit ripple carry adder module using four stages of full adder module

Circuit Diagram

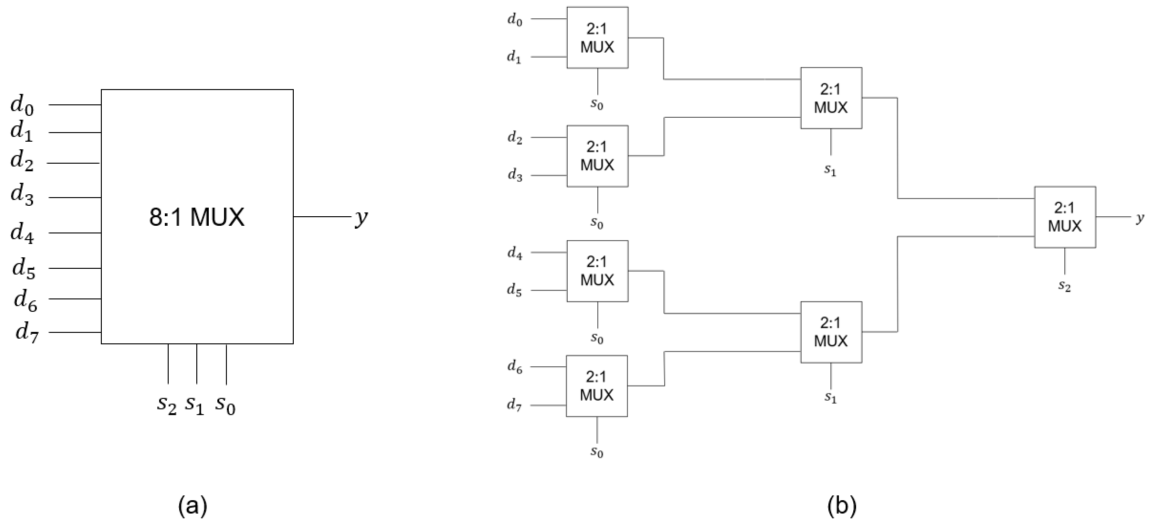

Fig. 1. (a) Schematic of 8:1 MUX, **(b)** Design of 8:1 MUX using 2:1 MUX

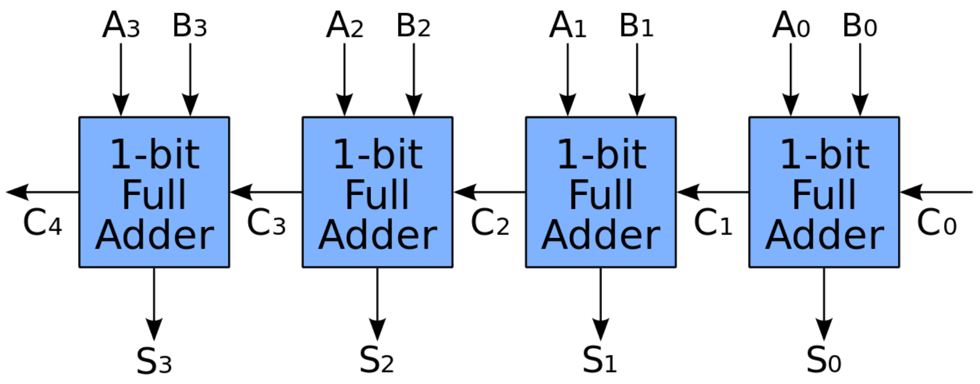

Fig. 2. Schematic of 4-bit ripple carry adder (designed using four 1-bit full adder)

Theory

8:1 MUX

Multiplexer is a combinational circuit which have many data inputs and single output depending on control or select inputs. For N input lines, $\lceil log_2 N \rceil$ selection lines, or we can say that for 2^n input lines, n selection lines are required. Multiplexers are also known as “Data n selector, parallel to serial convertor, many to one circuit, universal logic circuit”. Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.

An 8:1 MUX has 8 data input lines, 3 select lines and 1 output line (Fig. 1). The truth table of 8:1 MUX is given as follows

Table 1. ***Truth Table of 8:1 MUX***

	Selection Inputs		**Output**
S2	***S1***	***S0***	***Y***
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

This can be easily implemented using 2:1 MUX (Fig. 1(b)). The core idea is to select the index of that data input whose index has binary representation given by the select lines. In the first stage, select S0 is used to filter out the possible candidates, further filtering is done by select S1 and S2. This algorithm works because for a given select inputs, there exists only one index in data inputs with which it is associated.

4-bit Ripple Carry Adder

Multiple full adder circuits can be cascaded in series to add an N-bit number. For an N-bit serial adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs. Propagation delays inside the logic circuitry is the reason behind this.

For 4-bit ripple carry adder, only four full adder modules are required. Consider we want to add two numbers $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$. This addition is performed from LSB to MSB. Initially the input carry is set to 0. In the first stage, A_0 and B_0 are added using 1-bit full adder modules, the sum is stored at 0-th bit of ripple carry adder’ sum and the output carry is propagated to next stage as input carry. This is done until we have added every bits in the two integers.

Verilog codes

1 8:1 MUX Design

```
// Behavioral modelling of 2:1 MUX
module mux_2to1_beh(input [1:0]in,input sel,output reg out);
  always@(*)
  if(sel==1'b0)
    out=in[0];
  else
    out=in[1];
endmodule

// Structural modelling of 2:1 MUX
module mux_2to1_str(input [1:0]in,input sel,output out);
  wire w1,w2,sel_bar;
  not N1(sel_bar,sel);
  and a1(w1,sel,in[1]);
  and a2(w2,sel_bar,in[0]);
  or g1(out,w1,w2);
endmodule

// Designing 8:1 MUX using 2:1 MUX
module mux_8to1(input [7:0]in,input [2:0]sel,output out);
  wire [5:0]x;
  mux_2to1_str m1(in[1:0],sel[0],x[0]);
  mux_2to1_str m2(in[3:2],sel[0],x[1]);
  mux_2to1_str m3(in[5:4],sel[0],x[2]);
  mux_2to1_str m4(in[7:6],sel[0],x[3]);
  mux_2to1_str m5(x[1:0],sel[1],x[4]);
  mux_2to1_str m6(x[3:2],sel[1],x[5]);
  mux_2to1_str m7(x[5:4],sel[2],out);
endmodule

// Setting test-bench
module mux_8to1_tb();
  reg [7:0]in;
  reg [2:0]sel;
  wire out;
  mux_8to1 m1(in,sel,out);
  initial begin
    in=8'b10100111;
    sel=3'b000;
    $display("Time       Input    Select  Output");
    $monitor("%3g  %12b %7b %5b",$time,in,sel,out);
    #39 $finish;
  end
  always #5 sel=sel+1;

  initial begin
    $dumpfile("mux_8to1.vcd");
    $dumpvars;
  end
endmodule
```

2 4-bit Ripple Carry Adder

```
// 1-bit full adder module
module fadder(input a,b,c,output s, cout);
  assign s=a^b^c;
  assign cout=(a&b)|(b&c)|(a&c);
endmodule

// 4-bit ripple carry adder module
module ripper4(input [3:0]a,input [3:0]b, output [3:0]s,input c0,output c4);
  wire c1,c2,c3;
  fadder f1(a[0],b[0],c0,s[0],c1);
  fadder f2(a[1],b[1],c1,s[1],c2);
  fadder f3(a[2],b[2],c2,s[2],c3);
  fadder f4(a[3],b[3],c3,s[3],c4);
endmodule

// setting up the test bench
module ripple_tb;
  reg [3:0]a;
  reg [3:0]b;
  wire cout;
  wire [3:0]sum;
  assign c0=0;
  ripper4 instant(.a(a),.b(b),.s(sum),.c0(c0),.c4(cout));
  initial begin
    $monitor("time=%2d, a=%b%b%b%b, b=%b%b%b%b, s=%b%b%b%b, cout=%b\n",$time,a[3],a[2],a[1],a[0],b[3],
b[2],b[1],b[0],sum[3],sum[2],sum[1],sum[0],cout);
    #0;a=4'b0100;b=4'b0001;
    #5;a=4'b0110;b=4'b0001;
    #5;a=4'b0100;b=4'b1011;
    #5;a=4'b0100;b=4'b1111;
    #5;a=4'b1100;b=4'b1011;
    #5;$finish;
  end
  initial begin
    $dumpfile("ripple4_inst.vcd");
    $dumpvars;
  end
endmodule
```

Output Log

1 8:1 MUX

```
Time        Input     Select  Output

VCD info: dumpfile mux_8to1.vcd opened for output.

  0      10100111     000     1

  5      10100111     001     1

 10      10100111     010     1

 15      10100111     011     0

 20      10100111     100     0

 25      10100111     101     1

 30      10100111     110     0

 35      10100111     111     1

mux_8_1.v:39: $finish called at 39 (1s)
```

2 4-bit Ripple Carry Adder

```
VCD info: dumpfile ripple4_inst.vcd opened for output.

time= 0, a=0100, b=0001, s=0101, cout=0

time= 5, a=0110, b=0001, s=0111, cout=0

time=10, a=0100, b=1011, s=1111, cout=0

time=15, a=0100, b=1111, s=0011, cout=1

time=20, a=1100, b=1011, s=0111, cout=1

adder_4bit.v:29: $finish called at 25 (1s)
```

Waveforms

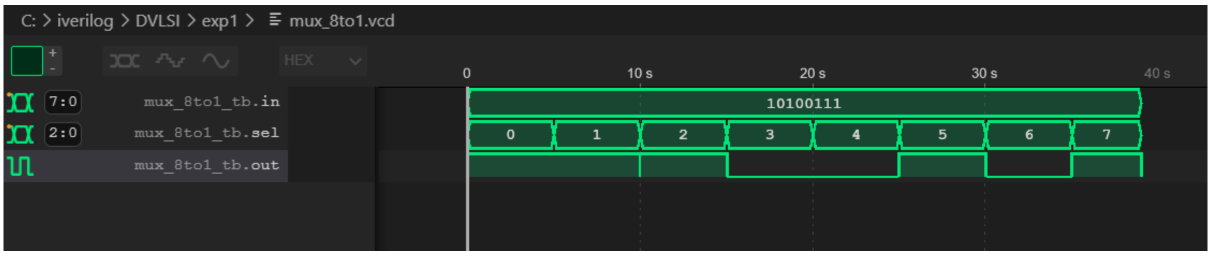

Fig. 3. ***Waveform observed in 8:1 MUX:*** In this figure, **`in`** represents the input 8-bit binary number and **`sel`** represents the 3-bit select input line and **`out`** represents the bit of in as dictated by **`sel`**.

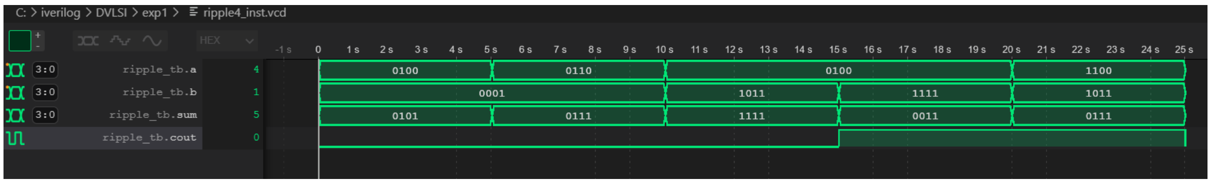

Fig. 4. ***Waveform observed in 4-bit ripple carry adder:*** In this figure, **`a`** and **`b`** represents 4-bit integers that we have to add using the module, **`sum`** represents sum of **`a`** and **`b`**, and **`cout`** represents the output carry after the addition.

Conclusion

- The first part of this experiment dealt with design of 8:1 MUX using only 2:1 MUX as per behavioural and structural modelling.
- Multiplexer is a combinational circuit which have many data inputs and single output depending on control or select inputs.
- From Fig. 1b, it can be observed that minimum of seven 2:1 MUX are required for such design.
- The same algorithm was implemented in Verilog code.
- The test bench was setup, where there is a fixed 8-bit binary number, and select inputs are used to extract a given bit of this number. During simulation (Fig. 3), it was observed that the code runs correctly.
- In the second experiment, 4-bit ripple carry adder is designed using four 1-bit full adder module.
- A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage.
- The design illustrated in Fig. 2 is implement in Verilog.
- The test bench was setup, where for some period of time, the two 4-bit integers **`a`** and **`b`** were fixed, and their sum and output carry are printed.
- During simulation (Fig. 4), we can observe that the implementation is correct as correct sum and output carry is observed for all the cases.