Lab Assignment – 10

Name – Md Junaid Mahmood

Enrolment Number – 19116040

Department – Computer Science & Engineering

Year – 2nd Year

Part-1: Design an ALU using Verilog. The ALU takes two inputs A and B. Each of these are 4-bit numbers. A third input called SEL determines the operation to be performed on A and B. The output OUT gives the results (including carry, if any).

The Verilog Code for above ALU is given below:

```
// Code your design here
module alu(A, B, SEL, OUT);
  input [3:0] A, B;
  input [2:0] SEL;

  output [7:0] OUT;

  reg [7:0] OUT;

  always @* begin
    case(SEL)
      3'b000: begin
        OUT <= 8'b00000000;
      end
      3'b001: begin
        OUT <= A + B;
      end
```

```
   3'b010: begin
    OUT <= A * B;
   end
   3'b011: begin
    OUT <= 8'b00000000;
    OUT[3:3] <= A[2:2];
    OUT[2:2] <= A[1:1];
    OUT[1:1] <= A[0:0];
    OUT[0:0] <= A[3:3];
   end
   3'b100: begin
    OUT <= 8'b00000000;
    OUT[3:3] <= A[0:0];
    OUT[2:2] <= A[3:3];
    OUT[1:1] <= A[2:2];
    OUT[0:0] <= A[1:1];
   end
   3'b101: begin
    OUT <= (A > B)?8'b00000001:8'b00000000;
   end
   3'b110: begin
    OUT <= (A == B)?8'b00000001:8'b00000000;
   end
   3'b111: begin
    OUT <= 8'b00000001;
   end
  endcase
 end
endmodule
```

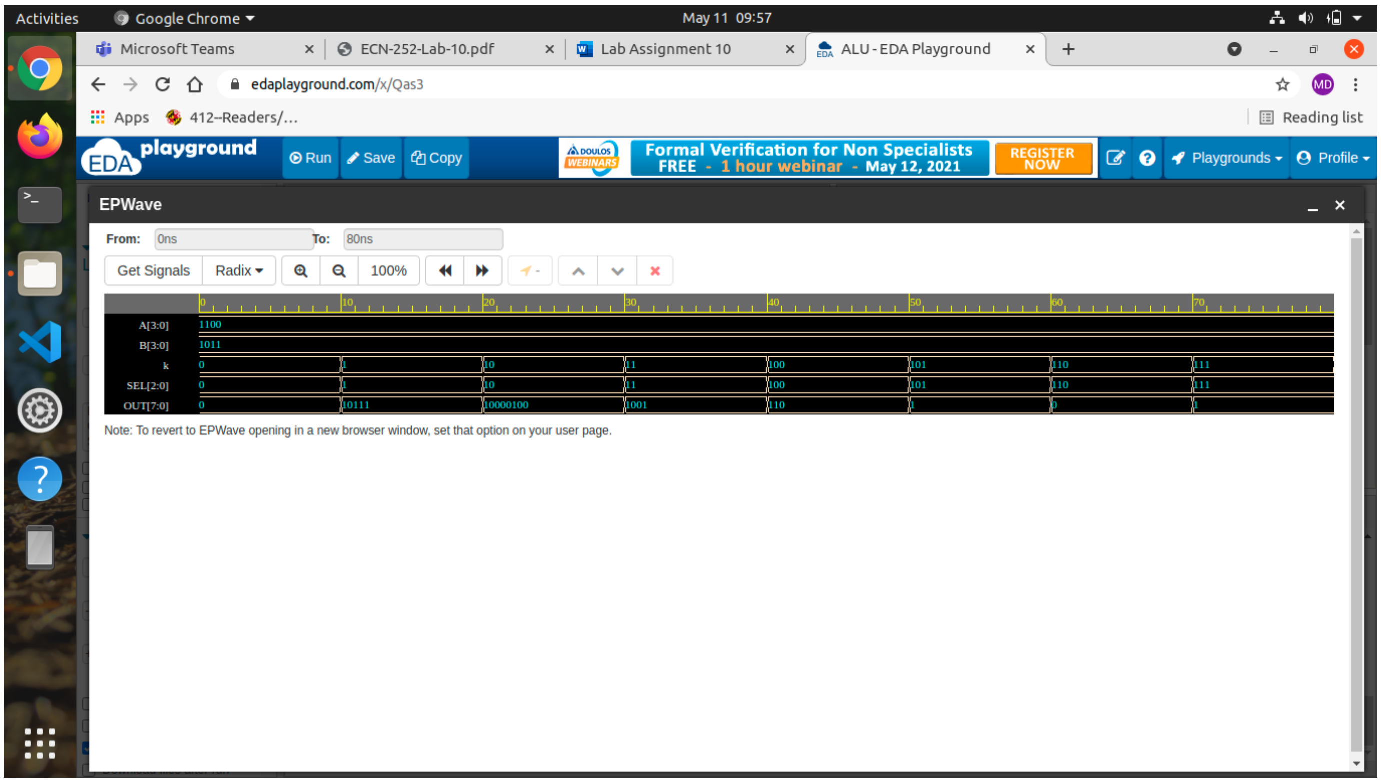
<u>Testbench designed for testing above code:</u>

```
// Code your testbench here
// or browse Examples
module main();
  reg [3:0] A, B;
  reg [2:0] SEL;
  wire [7:0] OUT;
  integer k;

  alu dut(.A(A), .B(B), .SEL(SEL), .OUT(OUT));

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

  initial begin
    A = 12;
    B = 11;
    SEL = -1;

    for (k=0;k<=7;k=k+1) begin
           SEL = SEL + 3'b001;
           #10;
    end
    $finish();
  end
endmodule
```

Now in above code we have adopted following conventions:

- Input A and B are 4-bit numbers.
- Input SEL is a 3-bit number.
- Output OUT is defined to be a 8-bit number. However, this 8-bits is only complete utilised in case of multiplications. In case of addition only last 5-bits are needed. In case of left rotation and right rotation only last 4-bits are needed. In remaining four cases only least significant bit of OUT is important.

Screenshot of waveform with A = $(12)_{10}$ and B = $(11)_{10}$ is given below:



```
A[3:0]    1100
B[3:0]    1011
k         0   1      10        11    100  101  110  111
SEL[2:0]  0   1      10        11    100  101  110  111
OUT[7:0]  0   10111  10000100  1001  110  1    0    1
```

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Now in the testbench designed, A = $(1100)_2$ and B = $(1011)_2$. Based on above snapshot, we have following result:

- When SEL = $(000)_2$, OUT = $(0)_2$.
- When SEL = $(001)_2$, OUT = $(10111)_2$.
- When SEL = $(010)_2$, OUT = $(10000100)_2$.
- When SEL = $(011)_2$, OUT = $(1001)_2$.
- When SEL = $(100)_2$, OUT = $(110)_2$.
- When SEL = $(101)_2$, OUT = $(1)_2$.
- When SEL = $(110)_2$, OUT = $(0)_2$.
- When SEL = $(111)_2$, OUT = $(1)_2$.

It can be observed that obtained value is same as the expected value. Thus, the corresponding Verilog Code is functioning correctly.