Stage 2 Edits (Verilog) Report

Project members:

Mahmoud Marzouk

Mohammed Abdelsalam

Ahmed Alkot

Ahmed Nasser

Instructor:

Dr. Amr Bayoumi

I. Work division across members.

Mahmoud Marzouk:

- Multiplication, XOR operations: hardware implementation and SystemVerilog code.
- Software circuit schematic.
- **Multiplication testbench**
- **Multiplication Edits**
- **Multiplication Schematic**

Ahmed Alkot:

- Logic operations (OR, NAND, and XOR): hardware implementation and SystemVerilog code.
- Output/Input circuit, and Multiplexers' design: hardware implementation.
- Gantt Chart preparation.
- Project report preparation.
- **New Mux Code**
- **Logic Operations Testbench**
- **Inputs and Logic Operations Schematic**

Mohammed Abdelsalam:

- Addition, subtraction operations: hardware implementation and SystemVerilog code.
- SystemVerilog code integration and testbench.
- Hand-Drawn Schematic
- Purchasing components.
- **Addition and Subtraction Testbench**
- **Addition and Subtraction Edits**
- **New ALU Code**
- **Addition, Subtraction, INC, Shifter, and outputs Schematic**

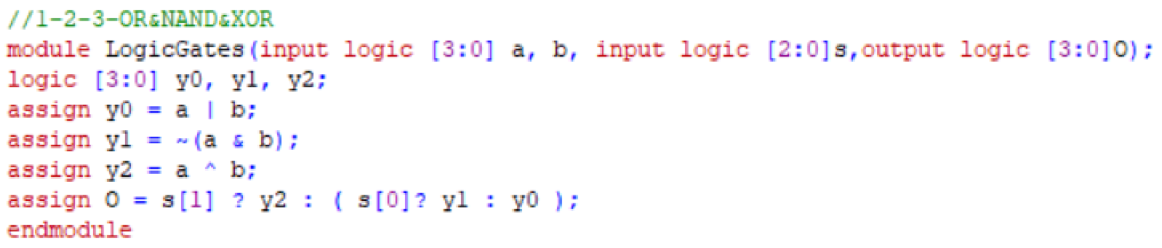
Ahmed Nasser:

- Right shift, incrementation operations: hardware implementation and SystemVerilog code.
- Handling power input and supply, Input/Output circuit, and LEDs.
- Devising a method for inputs’ control via an Arduino chip and the power supply.
- Purchasing components.
- **Shifter and INC Edits**
- **Shifter and INC testbench**
- **Complete PCB Design**

Code:

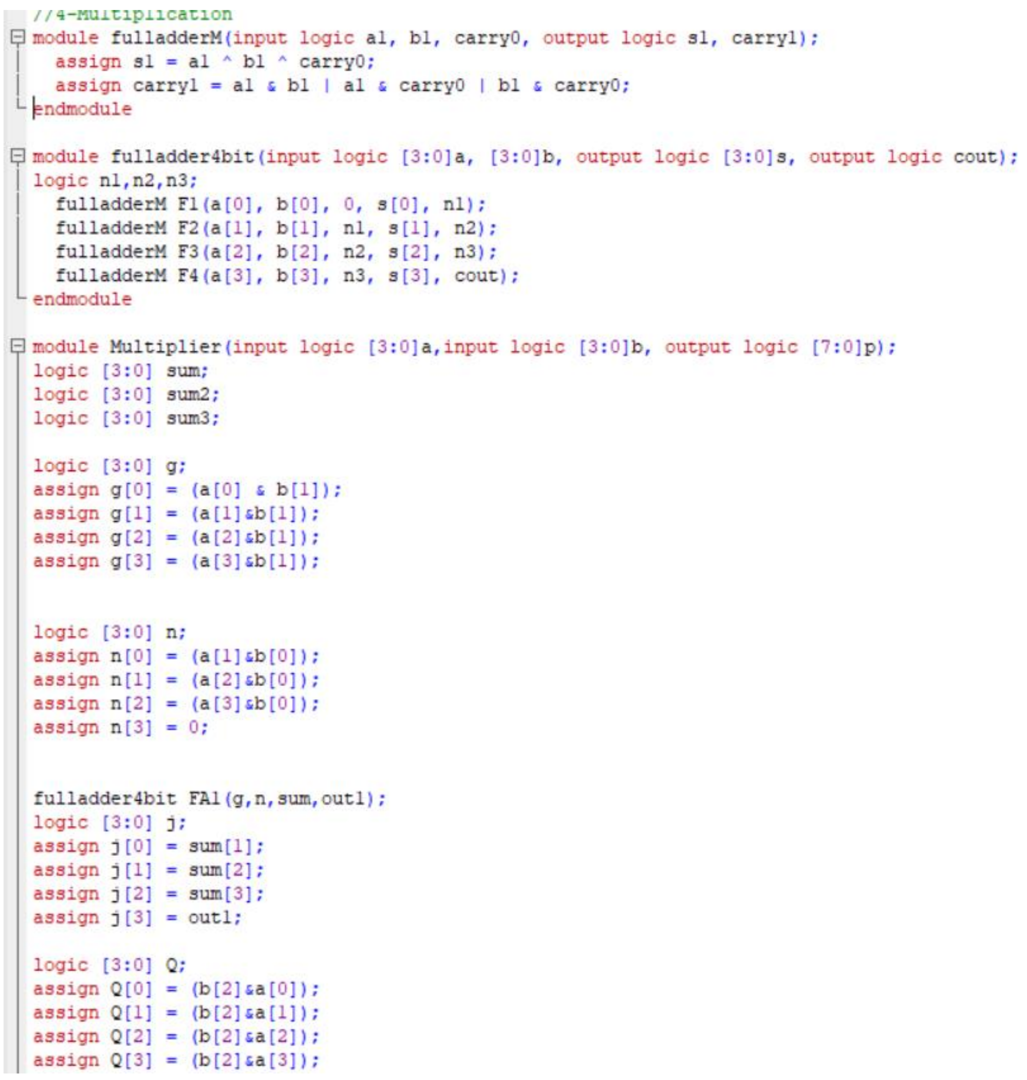
Bottom Level:

Logic Operations:

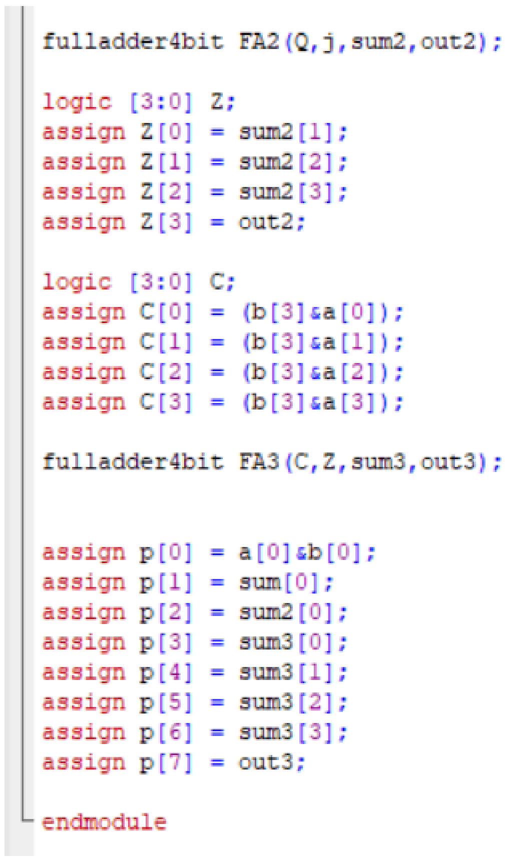


```
//1-2-3-OR&NAND&XOR
module LogicGates(input logic [3:0] a, b, input logic [2:0]s,output logic [3:0]O);
logic [3:0] y0, y1, y2;
assign y0 = a | b;
assign y1 = ~(a & b);
assign y2 = a ^ b;
assign O = s[1] ? y2 : ( s[0]? y1 : y0 );
endmodule
```

Multiplication:

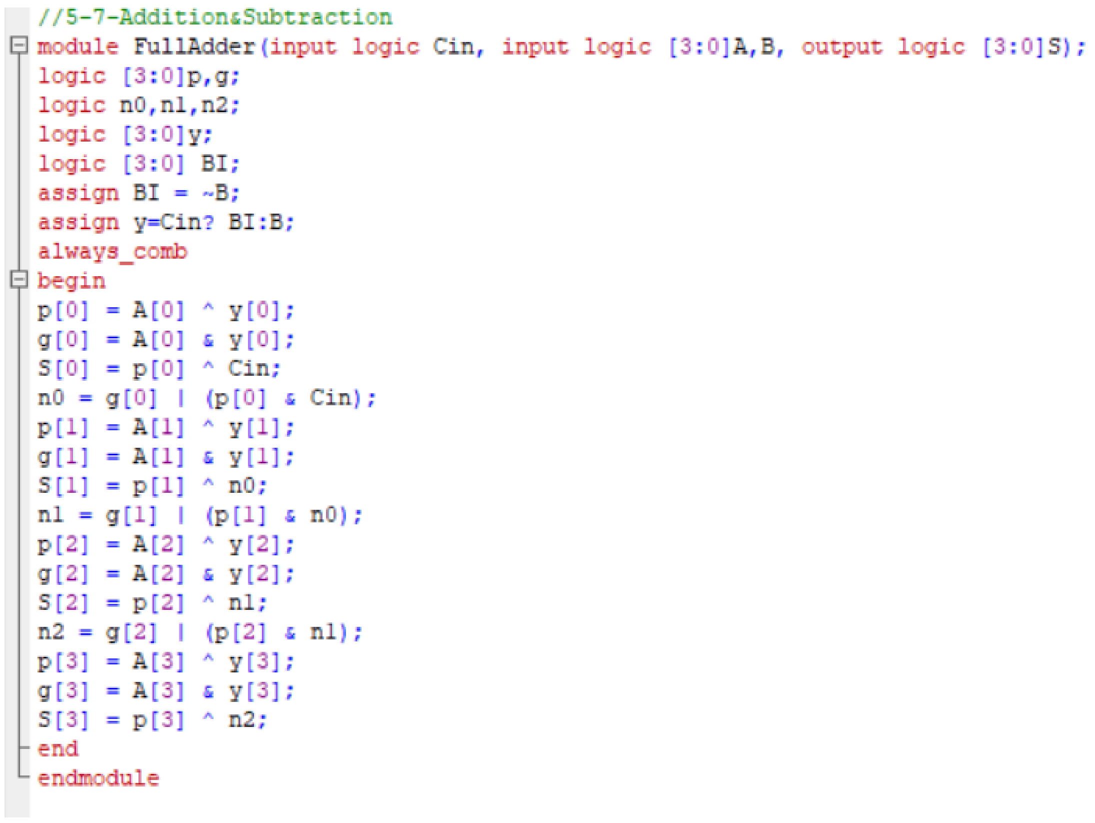


```
//4-Multiplication
module fulladderM(input logic a1, b1, carry0, output logic s1, carry1);
  assign s1 = a1 ^ b1 ^ carry0;
  assign carry1 = a1 & b1 | a1 & carry0 | b1 & carry0;
endmodule

module fulladder4bit(input logic [3:0]a, [3:0]b, output logic [3:0]s, output logic cout);
logic n1,n2,n3;
  fulladderM F1(a[0], b[0], 0, s[0], n1);
  fulladderM F2(a[1], b[1], n1, s[1], n2);
  fulladderM F3(a[2], b[2], n2, s[2], n3);
  fulladderM F4(a[3], b[3], n3, s[3], cout);
endmodule

module Multiplier(input logic [3:0]a,input logic [3:0]b, output logic [7:0]p);
logic [3:0] sum;
logic [3:0] sum2;
logic [3:0] sum3;

logic [3:0] g;
assign g[0] = (a[0] & b[1]);
assign g[1] = (a[1]&b[1]);
assign g[2] = (a[2]&b[1]);
assign g[3] = (a[3]&b[1]);


logic [3:0] n;
assign n[0] = (a[1]&b[0]);
assign n[1] = (a[2]&b[0]);
assign n[2] = (a[3]&b[0]);
assign n[3] = 0;


fulladder4bit FA1(g,n,sum,out1);
logic [3:0] j;
assign j[0] = sum[1];
assign j[1] = sum[2];
assign j[2] = sum[3];
assign j[3] = out1;

logic [3:0] Q;
assign Q[0] = (b[2]&a[0]);
assign Q[1] = (b[2]&a[1]);
assign Q[2] = (b[2]&a[2]);
assign Q[3] = (b[2]&a[3]);
```



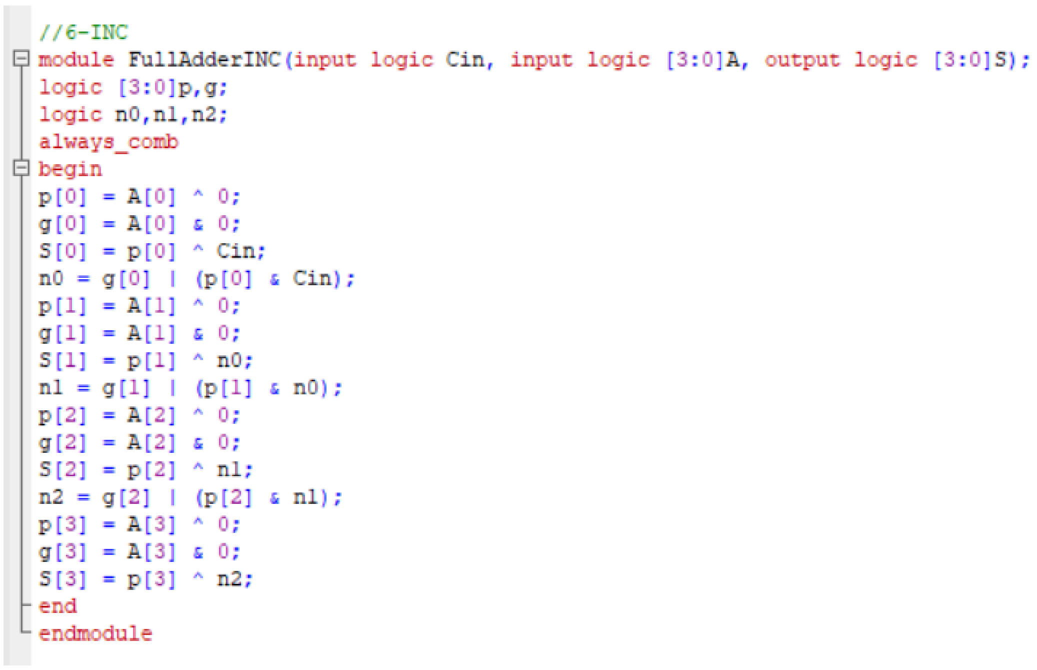
```
fulladder4bit FA2(Q,j,sum2,out2);

logic [3:0] Z;
assign Z[0] = sum2[1];
assign Z[1] = sum2[2];
assign Z[2] = sum2[3];
assign Z[3] = out2;

logic [3:0] C;
assign C[0] = (b[3]&a[0]);
assign C[1] = (b[3]&a[1]);
assign C[2] = (b[3]&a[2]);
assign C[3] = (b[3]&a[3]);

fulladder4bit FA3(C,Z,sum3,out3);


assign p[0] = a[0]&b[0];
assign p[1] = sum[0];
assign p[2] = sum2[0];
assign p[3] = sum3[0];
assign p[4] = sum3[1];
assign p[5] = sum3[2];
assign p[6] = sum3[3];
assign p[7] = out3;

endmodule
```

Addition & Subtraction:



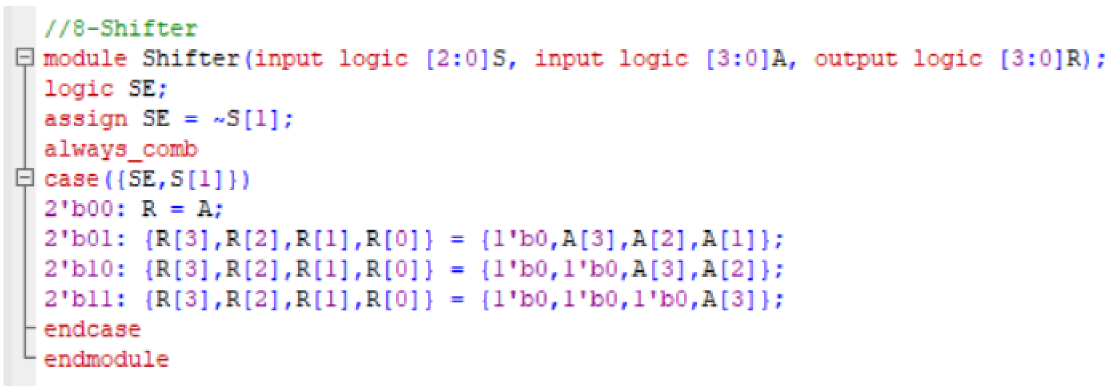
```
//5-7-Addition&Subtraction
module FullAdder(input logic Cin, input logic [3:0]A,B, output logic [3:0]S);
logic [3:0]p,g;
logic n0,n1,n2;
logic [3:0]y;
logic [3:0] BI;
assign BI = ~B;
assign y=Cin? BI:B;
always_comb
begin
p[0] = A[0] ^ y[0];
g[0] = A[0] & y[0];
S[0] = p[0] ^ Cin;
n0 = g[0] | (p[0] & Cin);
p[1] = A[1] ^ y[1];
g[1] = A[1] & y[1];
S[1] = p[1] ^ n0;
n1 = g[1] | (p[1] & n0);
p[2] = A[2] ^ y[2];
g[2] = A[2] & y[2];
S[2] = p[2] ^ n1;
n2 = g[2] | (p[2] & n1);
p[3] = A[3] ^ y[3];
g[3] = A[3] & y[3];
S[3] = p[3] ^ n2;
end
endmodule
```

INC:



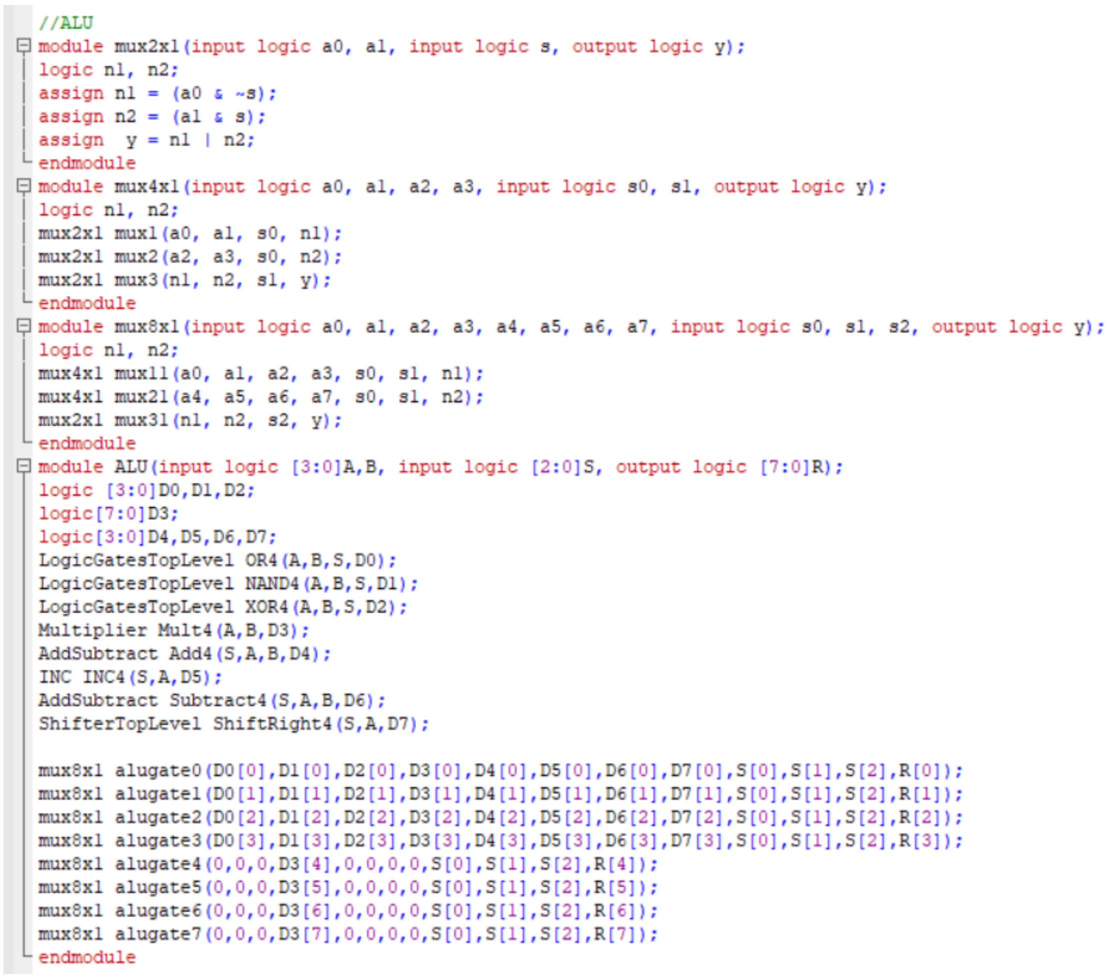
```
//6-INC
module FullAdderINC(input logic Cin, input logic [3:0]A, output logic [3:0]S);
logic [3:0]p,g;
logic n0,n1,n2;
always_comb
begin
p[0] = A[0] ^ 0;
g[0] = A[0] & 0;
S[0] = p[0] ^ Cin;
n0 = g[0] | (p[0] & Cin);
p[1] = A[1] ^ 0;
g[1] = A[1] & 0;
S[1] = p[1] ^ n0;
n1 = g[1] | (p[1] & n0);
p[2] = A[2] ^ 0;
g[2] = A[2] & 0;
S[2] = p[2] ^ n1;
n2 = g[2] | (p[2] & n1);
p[3] = A[3] ^ 0;
g[3] = A[3] & 0;
S[3] = p[3] ^ n2;
end
endmodule
```

Shifter:



```
//8-Shifter
module Shifter(input logic [2:0]S, input logic [3:0]A, output logic [3:0]R);
logic SE;
assign SE = ~S[1];
always_comb
case({SE,S[1]})
2'b00: R = A;
2'b01: {R[3],R[2],R[1],R[0]} = {1'b0,A[3],A[2],A[1]};
2'b10: {R[3],R[2],R[1],R[0]} = {1'b0,1'b0,A[3],A[2]};
2'b11: {R[3],R[2],R[1],R[0]} = {1'b0,1'b0,1'b0,A[3]};
endcase
endmodule
```

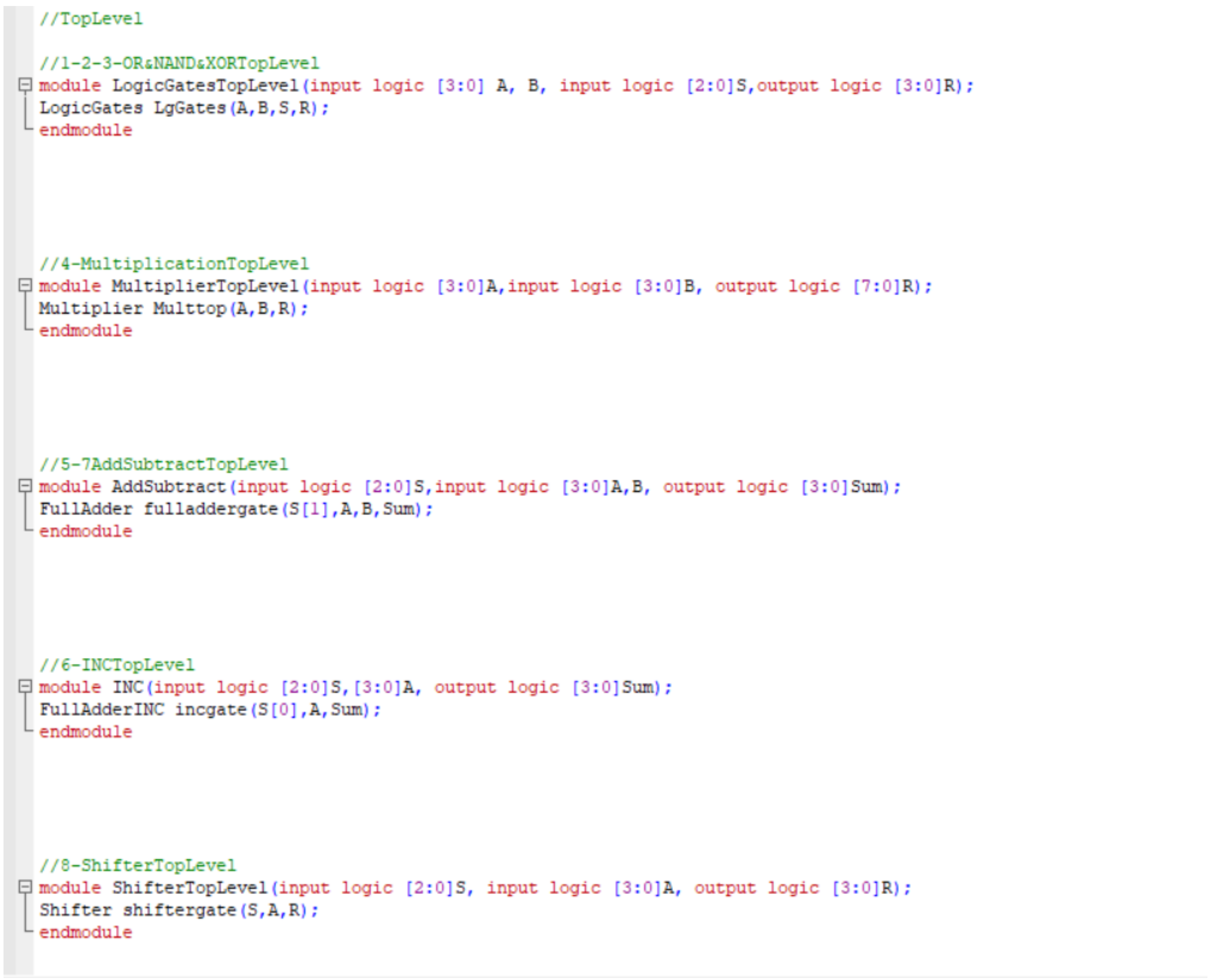
ALU:



```
//ALU
module mux2x1(input logic a0, a1, input logic s, output logic y);
logic n1, n2;
assign n1 = (a0 & ~s);
assign n2 = (a1 & s);
assign  y = n1 | n2;
endmodule
module mux4x1(input logic a0, a1, a2, a3, input logic s0, s1, output logic y);
logic n1, n2;
mux2x1 mux1(a0, a1, s0, n1);
mux2x1 mux2(a2, a3, s0, n2);
mux2x1 mux3(n1, n2, s1, y);
endmodule
module mux8x1(input logic a0, a1, a2, a3, a4, a5, a6, a7, input logic s0, s1, s2, output logic y);
logic n1, n2;
mux4x1 mux11(a0, a1, a2, a3, s0, s1, n1);
mux4x1 mux21(a4, a5, a6, a7, s0, s1, n2);
mux2x1 mux31(n1, n2, s2, y);
endmodule
module ALU(input logic [3:0]A,B, input logic [2:0]S, output logic [7:0]R);
logic [3:0]D0,D1,D2;
logic[7:0]D3;
logic[3:0]D4,D5,D6,D7;
LogicGatesTopLevel OR4(A,B,S,D0);
LogicGatesTopLevel NAND4(A,B,S,D1);
LogicGatesTopLevel XOR4(A,B,S,D2);
Multiplier Mult4(A,B,D3);
AddSubtract Add4(S,A,B,D4);
INC INC4(S,A,D5);
AddSubtract Subtract4(S,A,B,D6);
ShifterTopLevel ShiftRight4(S,A,D7);

mux8x1 alugate0(D0[0],D1[0],D2[0],D3[0],D4[0],D5[0],D6[0],D7[0],S[0],S[1],S[2],R[0]);
mux8x1 alugate1(D0[1],D1[1],D2[1],D3[1],D4[1],D5[1],D6[1],D7[1],S[0],S[1],S[2],R[1]);
mux8x1 alugate2(D0[2],D1[2],D2[2],D3[2],D4[2],D5[2],D6[2],D7[2],S[0],S[1],S[2],R[2]);
mux8x1 alugate3(D0[3],D1[3],D2[3],D3[3],D4[3],D5[3],D6[3],D7[3],S[0],S[1],S[2],R[3]);
mux8x1 alugate4(0,0,0,D3[4],0,0,0,0,S[0],S[1],S[2],R[4]);
mux8x1 alugate5(0,0,0,D3[5],0,0,0,0,S[0],S[1],S[2],R[5]);
mux8x1 alugate6(0,0,0,D3[6],0,0,0,0,S[0],S[1],S[2],R[6]);
mux8x1 alugate7(0,0,0,D3[7],0,0,0,0,S[0],S[1],S[2],R[7]);
endmodule
```

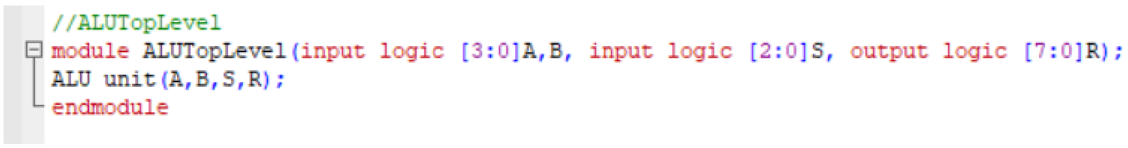
Top Level:

Everything:



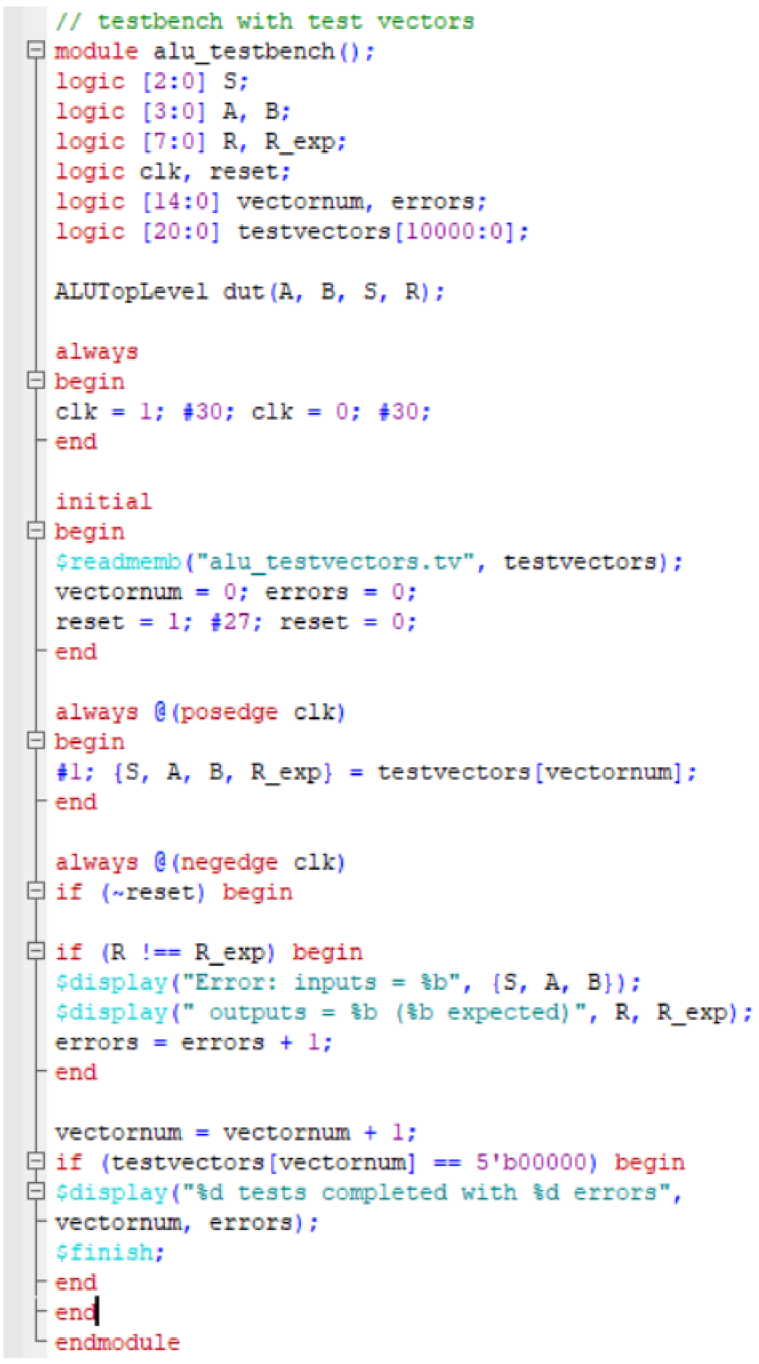
```
//TopLevel

//1-2-3-OR&NAND&XORTopLevel
module LogicGatesTopLevel(input logic [3:0] A, B, input logic [2:0]S,output logic [3:0]R);
LogicGates LgGates(A,B,S,R);
endmodule




//4-MultiplicationTopLevel
module MultiplierTopLevel(input logic [3:0]A,input logic [3:0]B, output logic [7:0]R);
Multiplier Multtop(A,B,R);
endmodule




//5-7AddSubtractTopLevel
module AddSubtract(input logic [2:0]S,input logic [3:0]A,B, output logic [3:0]Sum);
FullAdder fulladdergate(S[1],A,B,Sum);
endmodule




//6-INCTopLevel
module INC(input logic [2:0]S,[3:0]A, output logic [3:0]Sum);
FullAdderINC incgate(S[0],A,Sum);
endmodule




//8-ShifterTopLevel
module ShifterTopLevel(input logic [2:0]S, input logic [3:0]A, output logic [3:0]R);
Shifter shiftergate(S,A,R);
endmodule
```

ALU:



```
//ALUTopLevel
module ALUTopLevel(input logic [3:0]A,B, input logic [2:0]S, output logic [7:0]R);
ALU unit(A,B,S,R);
endmodule
```

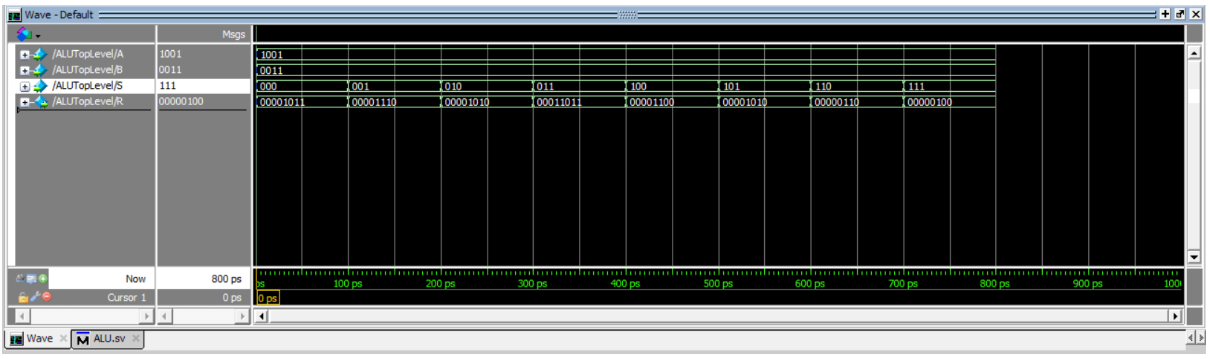
Testbench:



```
// testbench with test vectors
module alu_testbench();
logic [2:0] S;
logic [3:0] A, B;
logic [7:0] R, R_exp;
logic clk, reset;
logic [14:0] vectornum, errors;
logic [20:0] testvectors[10000:0];

ALUTopLevel dut(A, B, S, R);

always
begin
clk = 1; #30; clk = 0; #30;
end

initial
begin
$readmemb("alu_testvectors.tv", testvectors);
vectornum = 0; errors = 0;
reset = 1; #27; reset = 0;
end

always @(posedge clk)
begin
#1; {S, A, B, R_exp} = testvectors[vectornum];
end

always @(negedge clk)
if (~reset) begin

if (R !== R_exp) begin
$display("Error: inputs = %b", {S, A, B});
$display(" outputs = %b (%b expected)", R, R_exp);
errors = errors + 1;
end

vectornum = vectornum + 1;
if (testvectors[vectornum] == 5'b00000) begin
$display("%d tests completed with %d errors",
vectornum, errors);
$finish;
end
end
endmodule
```

Sample Simulation:

A sample of testbench with the same inputs but different selections is shown. You can check the whole 1568 operations using the testvector testbench.


Wave - Default
Msgs
/ALUTopLevel/A 1001
/ALUTopLevel/B 0011
/ALUTopLevel/S 111
/ALUTopLevel/R 00000100
1001
0011
000 001 010 011 100 101 110 111
00001011 00001110 00001010 00011011 00001100 00001010 00000110 00000100
Now 800 ps
Cursor 1 0 ps
ps 100 ps 200 ps 300 ps 400 ps 500 ps 600 ps 700 ps 800 ps 900 ps 100
0 ps
Wave
ALU.sv