INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

Course No: **EE206P**

Name of the Lab: **Digital Systems Laboratory**

Academic Year/Semester: **2024/B.Tech 3^{rd} Sem**

ASSIGNMENT NUMBER: **02**

Verilog HDL codes to perform some operations

MADHU SAI NAIK POOJARI
EE23B039
PRATHAM CHINTAMANI
EE23B041

Instructor:
Dr. Vikramkumar Pudi
Lab Assistant:
Mr. Kumar Bellikatti

19-Aug-2024

Contents

1 Problem Statement

Assignment-02

Write Verilog HDL code to perform the following operations?

S	Function	
000	C = A + B	A and B are N-bit signed
001	C = A - B	A and B are N-bit signed
010	C = A > B	A and B are N-bit unsigned
011	C = A < B	A and B are N-bit unsigned
100	C = (A = B)	A and B are N-bit unsigned
101	C = A x B (Unsigned)	A and B are N-bit unsigned
110	C = A x B (Signed)	A and B are N-bit signed
111	C = AxB + D (Signed)	A,B and D are N-bit signed
A, B, C and D are N-bit binary signed or Unsigned numbers. S is 3-bit binary number. The Default N size is 32-bits		

Deadline
Sept 19th , 2024

भारतीय प्रौद्योगिकी संस्थान तिरुपति
TIRUPATI

2 Objectives of the Experiment

To implement a logic code with logic gate modules, using verilog and to verify the output.

3 Design Approach

- Create code for different functions. Initially performing the code for 2 bits and to 4 bits. Finally to N bit.
- Create the final N bit code for the following operations.
- Create the test-bench for the following codes and test for some sample input.

4 Implementation

Full Adder as a building block

```
module Full_Adder(a,b,cin,sum,cout);
input a,b,cin;
output sum,cout;

assign sum = a ^ b ^ cin;
assign cout = (a&b) | (b&cin) |(cin&a);

endmodule
```

N-Bit Signed Ripple Carry Adder

```
module Signed_Adder_Nbit #(parameter N = 32)(a,b,cin,sum);
input [N-1:0]a,b;
wire [2*N-1:0]a1,b1;
input cin;
output [2*N-1:0]sum;
wire cout;
wire [2*N:0]w;
assign w[0] = cin;
assign a1 = {{N{a[N-1]}},a[N-1:0]};
assign b1 = {{N{b[N-1]}},b[N-1:0]};
genvar i;
generate
for (i = 0; i < 2*N; i = i + 1) begin
Full_Adder fa(.a(a1[i]),.b(b1[i]),.cin(w[i]),.cout(w[i+1]),.sum(sum[i])
   );
end
endgenerate
endmodule
```

Full Subtractor as a building block

```
module Sub_1bit( input a, input b ,input bin ,output y,output bout);
assign y = a^b^bin;
assign bout= bin&(~(a^b))|(~a)&b;
endmodule
```

N-Bit Signed Subtractor

```
module Sub_Nbit#(parameter N=8) (a,b,bin,c);
input [N-1:0] a,b;
input bin;
output [2*N-1:0]c;
wire [2*N-1:0]a1,b1;

assign a1 = {{N{a[N-1]}},a[N-1:0]};
assign b1 = {{N{b[N-1]}},b[N-1:0]};

wire [2*N-1:0] bout;

genvar j;
generate

for (j = 0; j < 2*N; j = j + 1)
begin

if (j == 0)
begin
Sub_1bit s1 (.a(a1[j]),.b(b1[j]),.bin(bin),.bout(bout[j]),.y(c[j]));
end

else
begin
Sub_1bit s1 (.a(a1[j]),.b(b1[j]),.bin(bout[j-1]),.bout(bout[j]),.y(c[j
   ]));
end

end
endgenerate

endmodule
```

N-Bit Greater-than Comparator

```
module Greater_Nbit #(parameter N =32) (a,b,y);
input [N-1:0] a,b;
output [2*N-1:0]y;

wire [N-1:0]e;
wire [N-1:0]g;
wire [N:0]f ;

assign e[N-1:0] = ~(a^b)  ;
assign g[N-1:0] = a & ~b    ;
assign f[0]= 0;

genvar  i;
generate

for (i=0 ; i<N ;i=i+1 ) begin
assign f[i+1]= g[i] | e[i]&f[i] ;
end

endgenerate

assign y = {{2*N-1{1'b0}} ,f[N]};
endmodule
```

N-Bit Less-than Comparator

```
module Lesser_Nbit #(parameter N =32) (a,b,y);
input [N-1:0] a,b;
output [2*N-1:0]y;

wire [N-1:0]e;
wire [N-1:0]g;
wire [N:0]f ;

assign e[N-1:0] = ~(a^b)  ;
assign g[N-1:0] = ~a& b    ;
assign f[0]= 0;

genvar  i;
generate

for (i=0 ; i<N ;i=i+1 ) begin
assign f[i+1]= g[i] | e[i]&f[i] ;
end

endgenerate

assign y = {{2*N-1{1'b0}} ,f[N]};
endmodule
```

N-Bit Equal to Comparator

```
module Equal_Nbit #(parameter N = 32) (a,b,y);

input [N-1:0]a,b;
wire [N-1:0] c;
output [2*N-1:0]y;

assign c = ~(a^b);
assign y = {{2*N-1{1'b0}} ,{&c}};

endmodule
```

N-Bit Multiplier Unsigned

```
module Multiplier_Nbit #(parameter N = 32) (a,b,y);

input [N-1:0]a,b;
output [2*N-1:0]y;

wire [2*N - 1:0] m [N-1:0];
wire [2*N - 1:0] s [N-1:0];
wire [N:0]c;

assign c[0] = 0;

generate
genvar i;
for (i = 0; i < N; i = i +1) begin : gen_m
assign m[i] = {{N{1'b0}}, a[N-1:0] & {N{b[i]}}, {i{1'b0}} };
end
endgenerate

Full_Adder_Nbit #(.N(2*N)) f1 (.a(m[0]), .b(m[1]), .cin(c[0]), .cout(c
   [1]), .sum(s[0]));

generate
genvar j;
for (j = 1; j < N-1; j = j+1)
begin : gen_adders
Full_Adder_Nbit #(.N(2*N)) f3 (.a(s[j-1]), .b(m[j+1]), .cin(c[j]), .
   cout(c[j+1]), .sum(s[j]));
end
endgenerate

assign y = s[N-2];

endmodule
```

N-bit Multiplier Signed

```
module Signed_Multiplier #(parameter N=32) (a,b,y);

input [N-1:0] a,b;
output [2*N-1:0] y;
wire [2*(N*N)-1:0] m;
wire [2*(N*N)-1:0] l;
wire [2*(N*N)-1:0] s;
wire [N-1:0] cin;

assign cin[0] =0;

genvar i ;
generate

for (i=0;i<N;i=i+1) begin
if (i==0) begin
assign l[(2*N*(i+1))-1 : 2*N*i] = {{{(N-i-1){1'b0}}, 1'b1 }, a&{N{b[i
   ]}}, {i{1'b0}}};
end
else begin
assign l[(2*N*(i+1))-1 : 2*N*i] = {{(N-i){1'b0}}, a&{N{b[i]}}, {i{1'b0
   }}};
end

end
endgenerate


genvar j ;
generate

for (j=0;j<N;j=j+1) begin
if (j==0) begin
assign m [(2*N*(j+1))-1 : 2*N*j] = {l[ 2*N-1 : N+1], l[N], ~l[N-1], l[N
   -2 : 0]};
end
if (j==N-1) begin
```

```
35 assign m [(2*N*(j+1))-1 : 2*N*j] = {1'b1,l[(2*N*(j+1))-2], ~l[ (2*N*(j
      +1))-3 : 2*N*j+j], l[2*N*j+j-1 : 2*N*j]};
36 end
37 else begin
38 assign m [(2*N*(j+1))-1 : 2*N*j] = {l[(2*N*(j+1))-1 : (2*N*(j+1))-1-N+j
      +1], ~l[(2*N*(j+1))-1-N+j], l[(2*N*(j+1) )-1-N+j-1 : 2*N*j]};
39 end
40
41 end
42 endgenerate
43
44 assign s[(2*N )-1:0] =  m[2*N-1:0];
45 genvar k;
46 generate
47
48 for (k=0;k<N-1;k=k+1) begin
49 Full_Adder_Nbit #(.N(2*N)) f1  (.a( s[2*N*(k+1)-1:2*N*k]),
50             .b( m[2*N*(k+2)-1:2*N*(k+1)]),
51             .sum(s[2*N*(k+2)-1:2*N*(k+1)]),
52                   .cin(cin[k]),
53             .cout(cin[k+1])
54 );
55
56 end
57 endgenerate
58
59
60 assign y[2*N-1:0] = s [2*(N*N)-1:2*N*(N-1)];
61
62 endmodule
```

N bit Multiplier Followed by 2N-bit adder

```
module AB_C #(parameter N = 32) (a,b,c,d);

input [N-1:0] a,b;
input [2*N-1:0] d;
output [2*N-1:0] c;
wire [2*N-1:0] e;
wire cout;

Signed_Multiplier #(N) f1(.a(a),.b(b),.y(e));
Full_Adder_Nbit #(2*N) f2 (.a(e), .b(d), .cin(1'b0), .cout(cout), .sum(
   c));

endmodule
```

Mux_8x1_Nbit

```
module Mux_8x1_Nbit #(parameter N = 32) (s,a,b,c,d,e,f,g,h,o);
input [2:0]s;
input [2*N-1:0] a,b,c,d,e,f,g,h;

output [2*N-1:0]o;

assign o = (s == 3'b000) ? a:
     (s == 3'b001) ? b:
     (s == 3'b010) ? c:
     (s == 3'b011) ? d:
     (s == 3'b100) ? e:
     (s == 3'b101) ? f:
     (s == 3'b110) ? g:
      h;

endmodule
```

TOP LEVEL

```
module Top_assign2 #(parameter N =32) (a,b,d,c,s);
input [N-1:0]a,b;
input [2*N-1:0]d;
input [2:0]s;
output [2*N-1:0]c;

wire [2*N-1:0] e,f,g,h,i,j,k,l;

Signed_Adder_Nbit #(N) fe (.a(a),.b(b),.cin(1'b0),.sum(e));
Sub_Nbit #(N) ff (.a(a),.b(b),.y(f),.bin(1'b0));
Greater_Nbit #(N) fg (.a(a),.b(b),.y(g));
Greater_Nbit #(N) fh (.a(a),.b(b),.y(h));
Lesser_Nbit #(N) fi (.a(a),.b(b),.y(i));
Multiplier_Nbit #(N) fj (.a(a),.b(b),.y(j));
Signed_Multiplier #(N) fk (.a(a),.b(b),.y(k));
AB_C #(N) fl (.a(a),.b(b),.d(d),.c(c));

Mux_8x1_Nbit #(N) m (.s(s),.a(e),.b(f),.c(g),.d(h),.e(i),.f(j),.g(k),.h
   (l),.o(c));

endmodule
```

TEST BENCH

```
module TBTop_assign();

  parameter M = 4;

  reg [2:0]s;
  reg [M-1:0]a;
  reg [M-1:0]b;
  reg [2*M-1:0]d;
  wire [M-1:0]c;

  Top_assign1 #(.N(M)) DUT(.a(a),.s(s),.b(b),.c(c).d(d));
  repeat(20)
begin
s = $random();
a = $random();
b = $random();
d = $random();
#10;
#5 $display("%b, %b, %b, %b,%b", s, a, b, d,c );
end
$finish;
end
initial



begin
$monitor("s = %b,a = %b,b = %b,d = %b,c = %b", s, a, b, d, c );
end
endmodule
```

All the modules have been constructed to handle 2N bit output to accommodate the multipliers in the mux

5 Simulation Results and Discussions

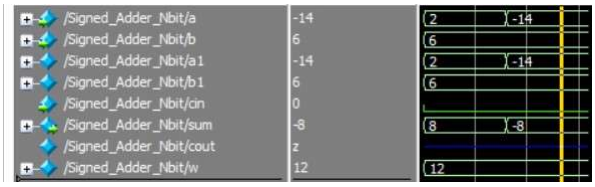

Figure 1: signed adder

Figure 2: C = AB+D

- All the modules were found to be working correctly.
- The output for each individual module was obtained.
- However, due to the size discrepancies in the top module, the final output was not obtained.

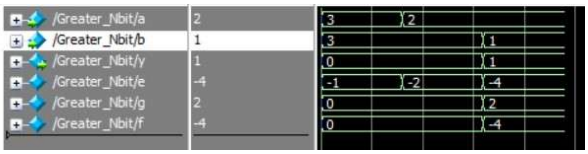

Figure 3: N-bit Greater Than comaparator

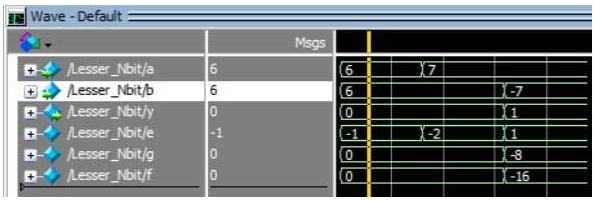

Figure 4: N-bit Lesser than comaparator

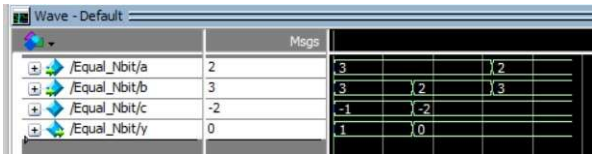

Figure 5: N-bit equal to comaparator

6 Conclusion

- Logic for the operations were implemented using verilog HDL code.
- These codes were connected to each other, like the full adder was used in both subtractor and multiplier.
- A test bench was made to verify the functionality and correctness of the verilog code with random inputs.
- The simulation was successful with zero errors.