Digital Design Lab Report

Name	Dev Goel
Roll No.	B20CS090
Experiment Number	**IV**

Objective

(i) Write a Program to implement a.
  - **a.** BCD Adder Unit
  - **b.** BCD Subtractor Unit

(ii) Write a Program to implement a Binary Multiplier (3-bit X 3-bit).
  - use input A = A0, A1, A2
    use input B = B0, B1, B2

EXPERIMENT (i)a: Write a Program to implement a BCD Adder Unit.

(All required codes are also present in the zipped folder containing .v files)

BCD Adder Unit:

```
`timescale 1ns / 1ps
module bcd_adder(inp_A, inp_B, inp_cin, out_ans, output_carry);

input [3:0] inp_A, inp_B;
input inp_cin;
output [3:0] out_ans;
output output_carry;

wire [3:0] out_S1;
wire out_carry;
wire [3:0] inp_A2;
wire c_out;
```

```
four_bit_adder fba1(inp_A, inp_B, inp_cin, out_S1, out_carry);
assign output_carry = (out_carry) | (out_S1[3]&out_S1[2]) |
(out_S1[3]&out_S1[1]);
assign inp_A2[3] = 0;
assign inp_A2[2] = output_carry;
assign inp_A2[1] = output_carry;
assign inp_A2[0] = 0;

four_bit_adder fba2(inp_A2, out_S1, 0, out_ans, c_out);
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_bcd_adder;

reg [3:0] inp_A, inp_B;
reg inp_cin;
wire [3:0] out_ans;
wire output_carry;
integer i, j;
bcd_adder bcd1(inp_A, inp_B, inp_cin, out_ans, output_carry);

initial
   begin
      assign inp_cin = 0;
      for(i = 0; i<10; i=i+1)
      begin
         for(j = 0; j<10 ; j=j+1)
         begin
            assign inp_A = i;
            assign inp_B = j; #10;
         end
      end
   end
initial #1000 $finish;
endmodule
```

Waveform:

EXPERIMENT (i)b: Write a Program to implement a BCD Subtractor Unit.

BCD Subtractor Unit:

```
`timescale 1ns / 1ps
module bcd_subtractor(inp_A, inp_B, carry_out, out_S);

input [3:0] inp_A, inp_B;
output carry_out;
output [3:0] out_S;

wire [3:0] out_B;

TensComplement c1(inp_B, out_B);

wire [3:0] out_ans;
bcd_adder bcd1(inp_A, out_B, 0, out_ans, carry_out);

wire [3:0] out_ans2;
TensComplement c2(out_ans, out_ans2);
assign out_S = (carry_out)? out_ans:out_ans2;
endmodule
```

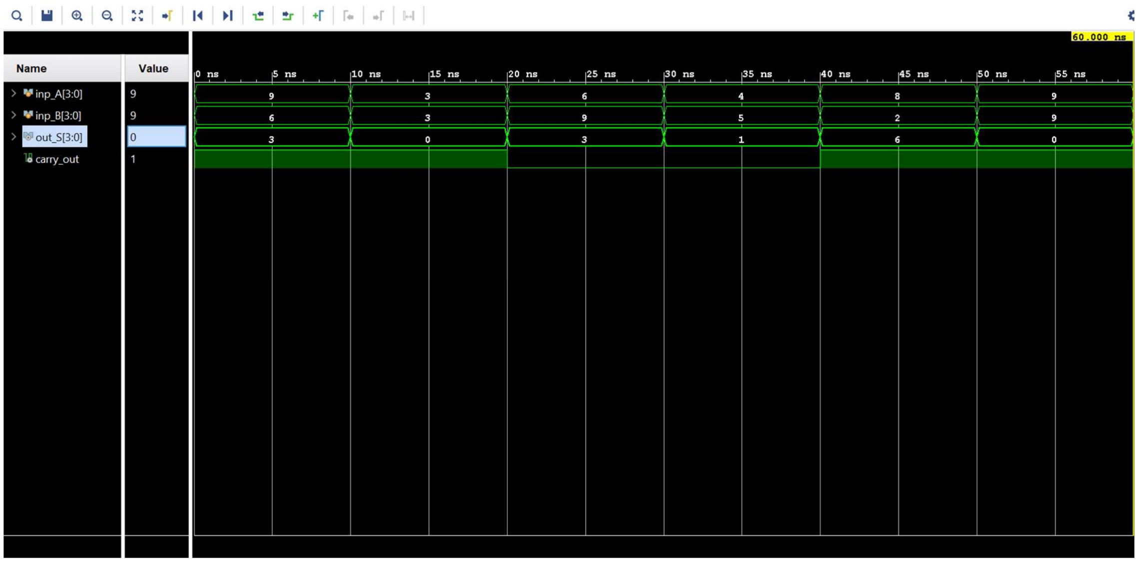
10s Complement:

```
`timescale 1ns / 1ps
module TensComplement(A,complement);
input [3:0] A;
output [3:0] complement;
assign compslement[0] = A[0];
assign complement[1] = A[1]&A[0]|A[3]&(~A[0])|A[2]&(~A[1])&(~A[0]);
assign complement[2] = A[2]&(~A[1])|A[2]&(~A[0])|(~A[2])&A[1]&A[0];
assign complement[3] =
(~A[2])&A[1]&(~A[0])|(~A[3])&(~A[2])&(~A[1])&A[0];
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_bcd_subtractor;

reg [3:0] inp_A, inp_B;
wire [3:0] out_S;
wire carry_out;

bcd_subtractor bcds1(inp_A, inp_B, carry_out, out_S);

initial
   begin
      inp_A = 9;  inp_B = 6;  #10;
      inp_A = 3;  inp_B = 3;  #10;
      inp_A = 6;  inp_B = 9;  #10;
      inp_A = 4;  inp_B = 5;  #10;
      inp_A = 8;  inp_B = 2;  #10;
      inp_A = 9;  inp_B = 9;  #10;
   end

initial #60 $finish;
endmodule
```

Waveform:



EXPERIMENT (ii): Write a Program to implement a Binary Multiplier (3-bit X 3-bit).

Binary Multiplier (3-bit X 3-bit):

```
`timescale 1ns / 1ps
module binary_multiplier(inp_A, inp_B, out_ans, ans_unit, ans_ten);

input [2:0] inp_A, inp_B;
output [5:0] out_ans;
output [3:0] ans_unit;
output reg [3:0] ans_ten;

wire [2:0] inp1a;
assign inp1a[0] = inp_A[0]&inp_B[1];
assign inp1a[1] = inp_A[1]&inp_B[1];
assign inp1a[2] = inp_A[2]&inp_B[1];
```

```
wire [2:0] inp1b;
assign inp1b[0] = inp_A[1]&inp_B[0];
assign inp1b[1] = inp_A[2]&inp_B[0];
assign inp1b[2] = 0;
wire [2:0] out1s;
wire out_carry1;
three_bit_adder tba1(inp1a, inp1b, 0, out1s, out_carry1);

wire [2:0] inp2a;
assign inp2a[0] = inp_A[0]&inp_B[2];
assign inp2a[1] = inp_A[1]&inp_B[2];
assign inp2a[2] = inp_A[2]&inp_B[2];
wire [2:0] inp2b;
assign inp2b[0] = out1s[1];
assign inp2b[1] = out1s[2];
assign inp2b[2] = out_carry1;
wire [2:0] out2s;
wire out_carry2;
three_bit_adder tba2(inp2a, inp2b, 0, out2s, out_carry2);

assign out_ans[0] = inp_A[0]&inp_B[0];
assign out_ans[1] = out1s[0];
assign out_ans[2] = out2s[0];
assign out_ans[3] = out2s[1];
assign out_ans[4] = out2s[2];
assign out_ans[5] = out_carry2;

assign ans_unit = out_ans%10;
always @(out_ans)
   begin
      if (out_ans < 10)
      begin
         assign ans_ten = 4'b0000;
      end
      else if (out_ans < 20)
      begin
         assign ans_ten = 4'b0001;
```

```
        end
        else if (out_ans < 30)
        begin
            assign ans_ten = 4'b0010;
        end
        else if (out_ans < 40)
        begin
            assign ans_ten = 4'b0011;
        end
        else if (out_ans < 50)
        begin
            assign ans_ten = 4'b0100;
        end
    end
 endmodule
```

Three-bit adder:

```
`timescale 1ns / 1ps
module three_bit_adder(inp_A, inp_B, inp_cin, out_S, out_cout);

input [2:0] inp_A, inp_B;
input inp_cin;
output [2:0] out_S;
output out_cout;
wire cout1, cout2;

full_adder fa1(inp_A[0], inp_B[0], inp_cin, out_S[0], cout1);
full_adder fa2(inp_A[1], inp_B[1], cout1, out_S[1], cout2);
full_adder fa3(inp_A[2], inp_B[2], cout2, out_S[2], out_cout);

endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_binary_multiplier;
```

```
reg [2:0] inp_A, inp_B;
wire [5:0] out_ans;
wire [3:0] ans_unit, ans_ten;
integer i, j;

binary_multiplier binMul1(inp_A, inp_B, out_ans, ans_unit, ans_ten);

initial
   begin
      for(i = 0; i<10; i=i+1)
      begin
         for(j = 0; j<10 ; j=j+1)
         begin
            assign inp_A = i;
            assign inp_B = j; #10;
         end
      end
   end
initial #1000 $finish;
endmodule
```

Waveform: