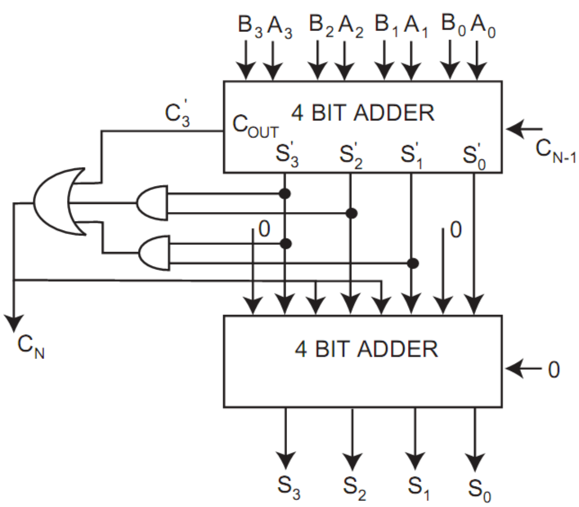
LAB REPORT-B20EE016

Digital Design - EEL2020

Dhruv

B20EE016

BCD-Adder



Binary Sum						BCD Sum						Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1		
0	0	0	0	0	S	0	0	0	0	0		0
0	0	0	0	1	A	0	0	0	0	1		1
0	0	0	1	0	M	0	0	0	1	0		2
. . . .	. . . .	. . . .	. . . .	. . . .	E C O	. . . .	. . . .	. . . .	. . . .	. . . .		. . . .
0	1	0	0	0	D	0	1	0	0	0		8
0	1	0	0	1	E	0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

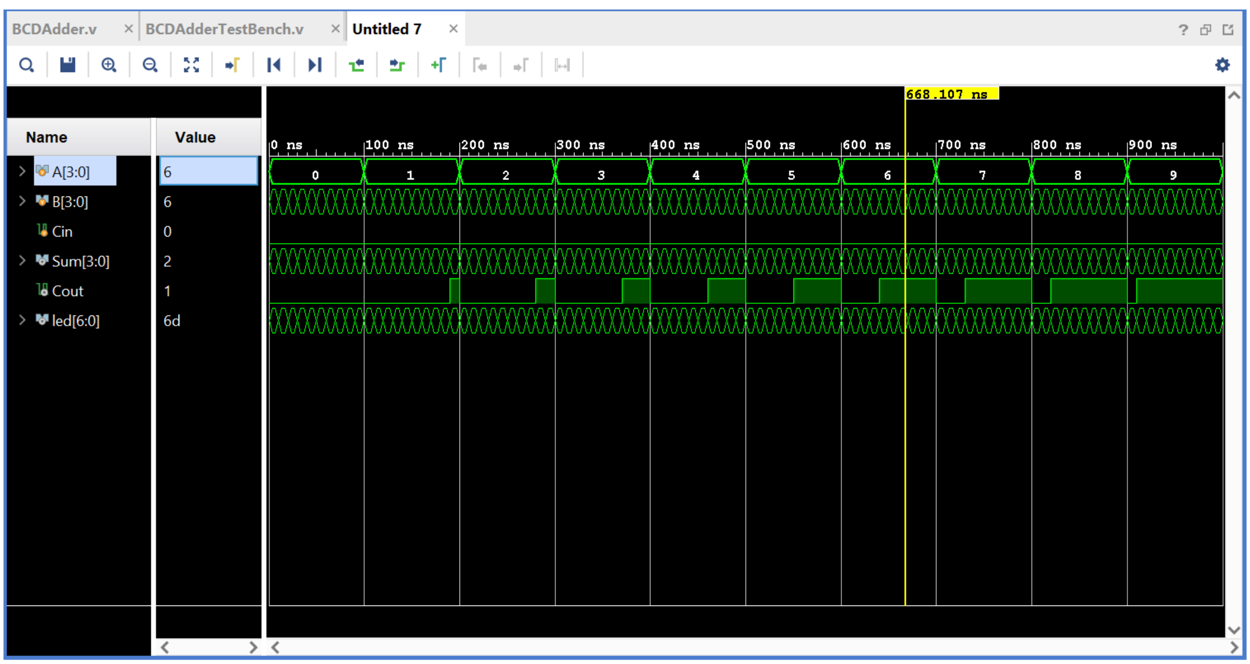
Code

```
`timescale 1ns / 1ps
module Half_Adder(a,b,sum,carry );
    input a,b;
    output sum ,carry;
    assign sum=a^b;
    assign carry=a&b;
endmodule

module Full_Adder(a,b,c,sum,carry);
    input a,b,c;
    output sum,carry;
    wire sum1,carry1,carry2;
```

```
    Half_Adder H1(b,c,sum1,carry1);
    Half_Adder H2(a,sum1,sum,carry2);
    assign carry=carry1|carry2;
endmodule




module SevenSegmentDisplay(Hex,Led);
    input [0:3] Hex;
    output reg [1:7] Led;
    always@(Hex)
        begin
            case(Hex)
                0 : Led = 7'b1111110;
                1 :  Led = 7'b0110000;
                2 :  Led  = 7'b1101101;
                3 :  Led = 7'b1111001;
                4 :  Led  = 7'b0110011;
                5 :  Led  = 7'b1011011;
                6 :  Led   = 7'b1011111;
                7 :  Led  = 7'b1110000;
                8 :  Led   = 7'b1111111;
                9 :  Led   = 7'b1111011;
                default: Led = 7'b0000000;
            endcase
        end
endmodule


module FourBitAdder(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0]Sum,
    output Cout
);
    wire [3:0] carry;

    assign carry[0]=Cin;
    Full_Adder F1(A[0],B[0],carry[0],Sum[0],carry[1]);
    Full_Adder F2(A[1],B[1],carry[1],Sum[1],carry[2]);
    Full_Adder F3(A[2],B[2],carry[2],Sum[2],carry[3]);
    Full_Adder F4(A[3],B[3],carry[3],Sum[3],Cout);
//    SevenSegmentDisplay s1(Sum,led);
endmodule


module BCDAdder(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0]Sum,
    output Cout,
```

```
    output [6:0] led
);
    reg [3:0] SixOrZero;
    wire temp1,temp2,temp3,temp4,temp5;
    wire [3:0] sum1 ;
    FourBitAdder F11(A,B,Cin,sum1,temp1);
    assign temp2=sum1[3]&sum1[2];
    assign temp3=sum1[3]&sum1[1];
    assign temp4=temp1|temp2|temp3;
    assign Cout=temp4;
    always @(temp4)
    begin
        if (temp4==1) SixOrZero=4'b0110;
        else SixOrZero=4'b0000;
    end
    FourBitAdder F12(sum1,SixOrZero,0,Sum,temp5);


    SevenSegmentDisplay s1(Sum,led);



endmodule
```

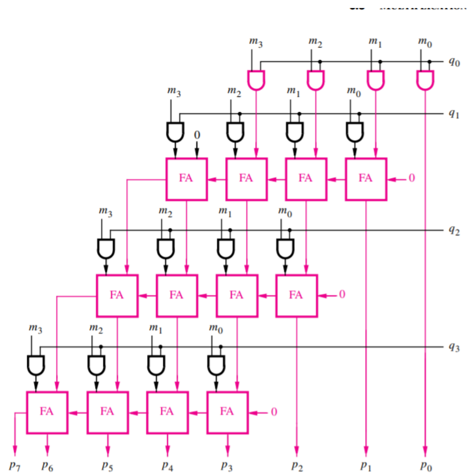
Simulation



TestBench

```
`timescale 1ns / 1ps
module BCDAdderTestBench();
    reg[3:0]A,B;
    reg Cin;
    wire [3:0]Sum;
    wire Cout;
    wire [6:0] led;
    integer i,j;

     BCDAdder B11(A,B,Cin,Sum,Cout,led);
    initial

    begin
        A = 0;
        B = 0;
        Cin=1'b0;
//        A=4'b0000;B=4'b0000;
//        #10 A=4'b0000;B=4'b0000;
//        #10 A=4'b0001;B=4'b0101;
//        #10 A=4'b1000;B=4'b1000;
            for(i=0;i<10;i=i+1) begin
            for(j=0;j<10;j=j+1) begin
            A = i;
            B = j;
            #10;
            end end
     end
//initial # $finish;


endmodule
```

BCD-Multiplier



Code

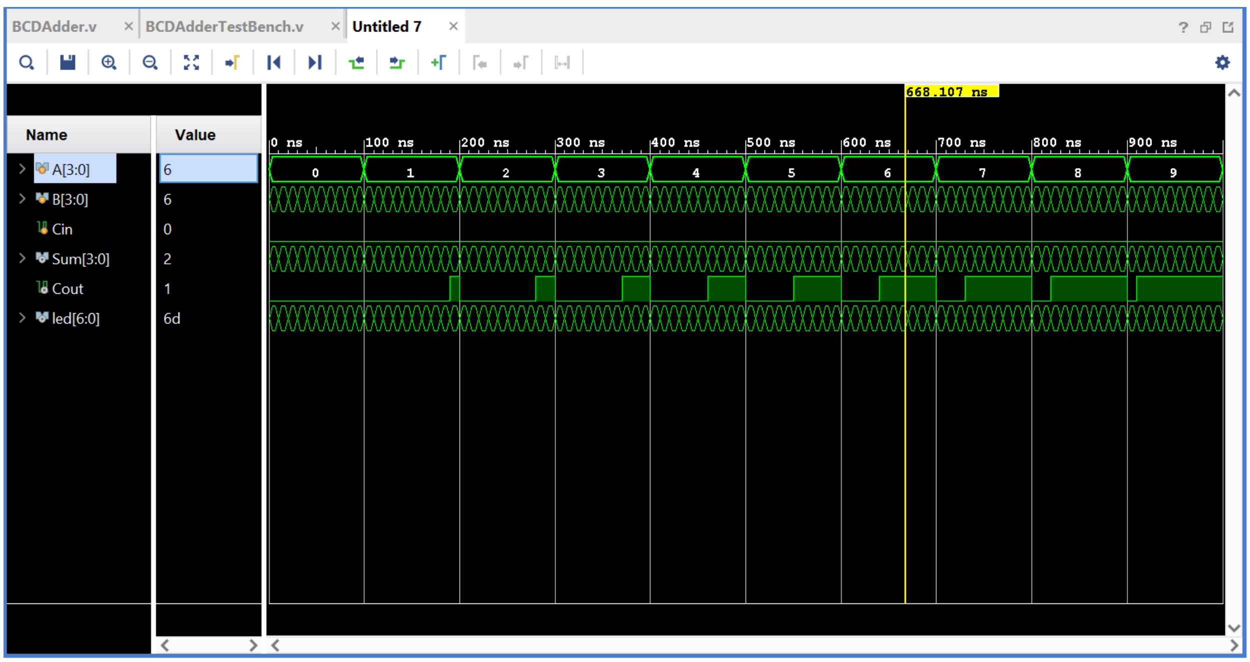
```
`timescale 1ns / 1ps

module Half_Adder(a,b,sum,carry );
    input a,b;
    output sum ,carry;
    assign sum=a^b;
    assign carry=a&b;
endmodule

module Full_Adder(a,b,c,sum,carry);
    input a,b,c;
    output sum,carry;
    wire sum1,carry1,carry2;
    Half_Adder H1(b,c,sum1,carry1);
    Half_Adder H2(a,sum1,sum,carry2);
    assign carry=carry1|carry2;
endmodule




module SevenSegmentDisplay(Hex,Led);
    input [0:3] Hex;
    output reg [1:7] Led;
    always@(Hex)
        begin
            case(Hex)
                0 : Led = 7'b1111110;
                1 :  Led = 7'b0110000;
```

```
                2 :  Led  = 7'b1101101;
                3 :  Led = 7'b1111001;
                4 :  Led  = 7'b0110011;
                5 :  Led  = 7'b1011011;
                6 :  Led   = 7'b1011111;
                7 :  Led  = 7'b1110000;
                8 :  Led   = 7'b1111111;
                9 :  Led   = 7'b1111011;
                default: Led = 7'b0000000;
            endcase
        end
endmodule







module FourBitAdder(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0]Sum,
    output Cout
);
    wire [3:0] carry;

    assign carry[0]=Cin;
    Full_Adder F1(A[0],B[0],carry[0],Sum[0],carry[1]);
    Full_Adder F2(A[1],B[1],carry[1],Sum[1],carry[2]);
    Full_Adder F3(A[2],B[2],carry[2],Sum[2],carry[3]);
    Full_Adder F4(A[3],B[3],carry[3],Sum[3],Cout);
//    SevenSegmentDisplay s1(Sum,led);
endmodule

module bcd_conv(
input [5:0] Product,
output reg [3:0] Tens,Ones);
wire [3:0] k;
wire [5:0]c;
wire [3:0]temp;

always @(Product)
    begin
        if ((Product <= 6'b001001 )==1) begin
        assign Tens=4'b0000;
        end
        else if ((Product >= 6'b001010 & Product < 6'b010100)==1) begin
        assign Tens=4'b0001;
        end
        else if ((Product >= 6'b010100& Product < 6'b011110)==1) begin
        assign Tens=4'b0010;
        end
```

```
        else if ((Product >= 6'b011110& Product < 6'b101000)==1) begin
        assign Tens=4'b0011;
        end
        else if ((Product >= 6'b101000& Product < 6'b110010)==1) begin
        assign Tens=4'b0100;
        end
        assign Ones=Product%10;

end

endmodule







module BCDMultiplier(
input [2:0] A,B,
output[5:0] Product,
output [6:0] LedTens,
output [6:0] LedOnes,
output[3:0] Ones,
output[3:0] Tens
);

wire[3:0] FirstProduct;
assign FirstProduct[0]=A[0]&B[0];
assign FirstProduct[1]=A[1]&B[0];
assign FirstProduct[2]=A[2]&B[0];
assign FirstProduct[3]=0;

wire[3:0] SecondProduct;
assign SecondProduct[0]=0;
assign SecondProduct[1]=A[0]&B[1];
assign SecondProduct[2]=A[1]&B[1];
assign SecondProduct[3]=A[2]&B[1];

wire [3:0] FirstSum;
wire FirstCarry;
FourBitAdder F1(FirstProduct,SecondProduct,0,FirstSum,FirstCarry);

wire[3:0] FirstProduct2;
assign FirstProduct2[0]=FirstSum[1];
assign FirstProduct2[1]=FirstSum[2];
assign FirstProduct2[2]=FirstSum[3];
assign FirstProduct2[3]=FirstCarry;

wire[3:0] SecondProduct2;
assign SecondProduct2[0]=0;
assign SecondProduct2[1]=A[0]&B[2];
assign SecondProduct2[2]=A[1]&B[2];
assign SecondProduct2[3]=A[2]&B[2];
```

```
wire [3:0] FirstSum2;
wire FirstCarry2;
FourBitAdder F2(FirstProduct2,SecondProduct2,0,FirstSum2,FirstCarry2);

assign Product[0]=FirstSum[0];
assign Product[1]=FirstSum2[0];
assign Product[2]=FirstSum2[1];
assign Product[3]=FirstSum2[2];
assign Product[4]=FirstSum2[3];
assign Product[5]=FirstCarry2;

//wire [3:0] Tens,Ones;
bcd_conv cnv(Product,Tens,Ones);
SevenSegmentDisplay s1(Tens,LedTens);
SevenSegmentDisplay s2(Ones,LedOnes);
endmodule
```

Simulation



TestBench

```
`timescale 1ns / 1ps
module BCDMultiplierTestBench();
    reg [2:0] A,B;
```

```
    wire [5:0] Product;
    wire [6:0] LedOnes;
    wire [6:0] LedTens;
    wire [3:0] Ones;
    wire [3:0] Tens;
    integer i, j;
    BCDMultiplier M1(A,B,Product,LedOnes,LedTens,Ones,Tens);
    initial
    begin
    for(i = 0; i<=7 ; i=i+1)begin
    for(j = 0 ; j<=7 ; j=j+1)begin
    A = i;
    B = j;
    #10;
    end
    end
    end
initial #800 $finish;
endmodule
```

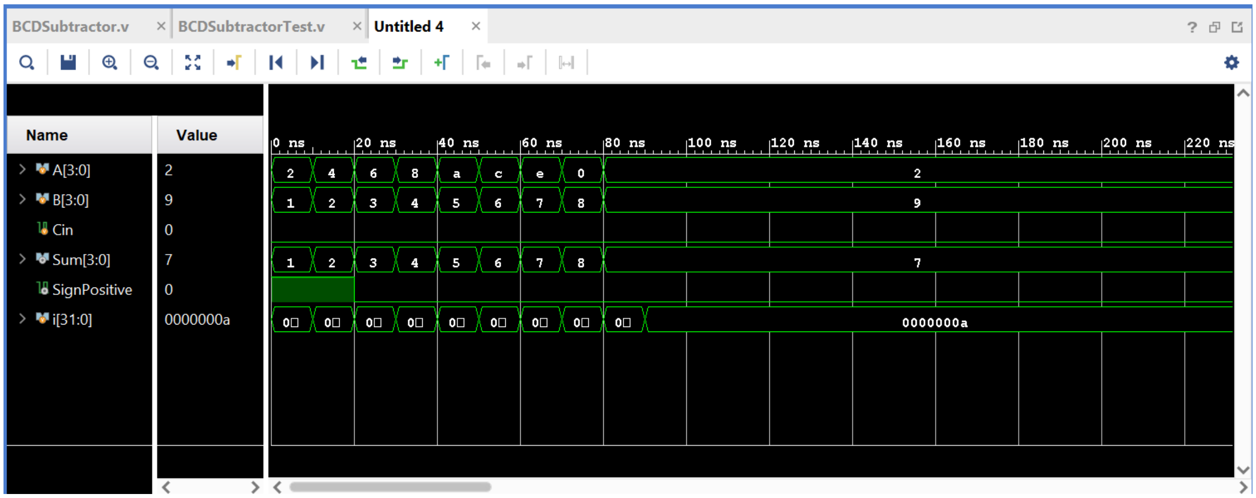
BCD-Subtractor

Code

```
`timescale 1ns / 1ps
module TensComplement(A,complement);
input [3:0] A;
output [3:0] complement;
assign complement[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

module SevenSegmentDisplay(Hex,Led);
    input [0:3] Hex;
    output reg [1:7] Led;
    always@(Hex)
        begin
            case(Hex)
                0 : Led = 7'b1111110;
                1 :  Led = 7'b0110000;
                2 :  Led  = 7'b1101101;
                3 :  Led = 7'b1111001;
                4 :  Led  = 7'b0110011;
                5 :  Led  = 7'b1011011;
                6 :  Led   = 7'b1011111;
```

```
                7 :  Led  = 7'b1110000;
                8 :  Led   = 7'b1111111;
                9 :  Led   = 7'b1111011;
                default: Led = 7'b0000000;
            endcase
        end
endmodule

module BCDSubtractor(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output reg [3:0]Sum,
    output reg SignPositive
);
wire temp4;
wire [3:0] Output,NB,NSum;
wire [6:0] templed;

TensComplement  A1(B,NB);
BCDAdder BCD1(A,NB,Cin,Output,temp4,templed);
TensComplement A22(Output,NSum);
always @(temp4)
    begin
        if (temp4==1) begin
        SignPositive=1'b1;
        assign Sum=Output;
        end
        else begin
        assign SignPositive=1'b0;
        assign Sum=NSum;
        end
    end


endmodule
```

Simulation



;

TestBench

```
`timescale 1ns / 1ps
module BCDSubtractorTest();
    reg[3:0]A,B;
    reg Cin;
    wire [3:0]Sum;
    wire SignPositive;
    integer i;
    BCDSubtractor B11(A,B,Cin,Sum,SignPositive);
    initial
    begin

    for (i = 1; i<10;i=i+1)begin
        A=2*i;
        B=i;
        Cin=0;
        #10;
    end
    end

endmodule
```

CONCLUSION

- In Bcd Adder we take 4bit numbers hence we can represnt only 0-15 numbers but on 7 segment display we can display 0-9 only so when our answer after addition comes out to be greater than 9 we add 6 to it whihch inturn make us skip 10-15 numbers (including 10-15) which in order helps to display it on two 7 segment display

- In order to calculate the BCD subtraction we take 10s compliment of the number