PART 1

Pmos #(5,6 ,7)

Nmos #(3,4,5)

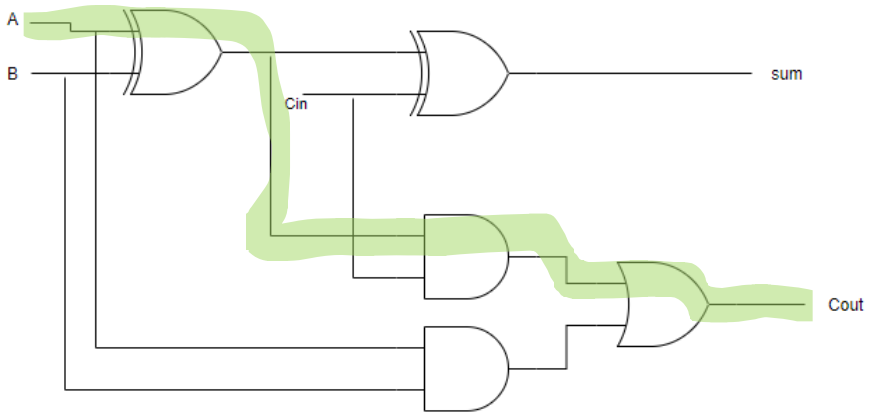
Not #(5,7)

Nand #(10,8) - > and#(17,13)

Nor #(10,14) -> or #(17,19)

Xor #(17,19) ((without considering inbuilt not delay))

Diagram:

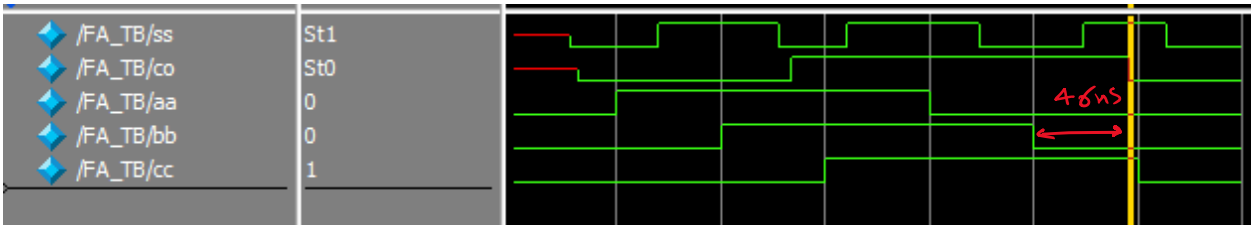


Worst case delay is the selected path with all delays to0

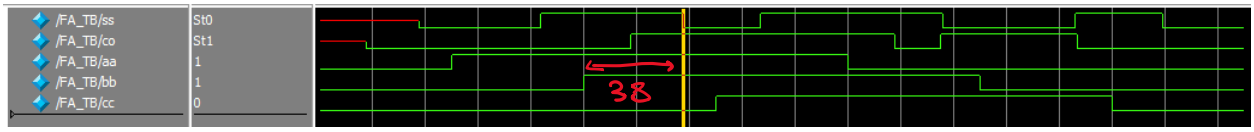
The result is 19 + 13 + 14 = 46ns

For all nand circuit the worst case is: 2*xor To0 = 38ns

Wave form(with functionality):



All nand for majority: (we use this circuit for the rest of project)



Verilog and testbench:

```
`timescale lns/lns

module Full_adder_G(input a,b,ci ,output s , co);
    xor #(17,19) x1(s0,a,b);
    xor #(17,19) x2(s,s0,ci);
    nand #(10,8) a1(c0,a,b);
    nand #(10,8) a2(c1,ci,s0);
    nand  #(10,8)o1(co,c1,c0);
endmodule
```

```
`timescale lns/lns
module FA_TB();
    wire ss,co;
    logic aa = 0,bb = 0,cc = 0;
    Full_adder_G F1(aa,bb,cc,ss,co);
    initial begin
        #50 aa=1;
        #50 bb = 1;
        #50 cc = 1;
        #50 aa = 0;
        #50 bb = 0;
        #50 cc = 0;
        #50 $stop;
    end
endmodule
```

PART 2 & 3

Adder verilog:

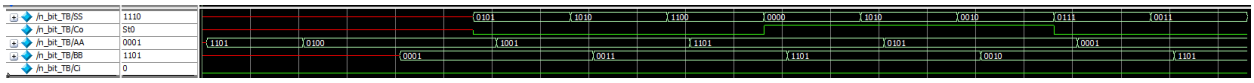
```
`timescale 1ns/1ns
module n_bit_adder #(parameter n = 1)(input [n-1:0] A,[n-1:0] B,input Ci, output [n-1:0] S,output Co);
    assign #(n*38) {Co,S} = A + B + Ci;
endmodule
```

Part 3 test bench:

```
`timescale 1ns/1ns


module n_bit_TB();
    wire [3:0]SS;
    wire Co;
    logic [3:0]AA;
    logic [3:0]BB;
    logic Ci = 0;
    n_bit_adder #(.n(4)) ADD1(.A(AA),.B(BB),.Ci(Ci),.S(SS),.Co(Co));
    initial begin
        #10 AA = 4'b1101;
        repeat (5) begin
            #200 AA = $random;
            #200 BB = $random;
        end
    end
endmodule
```

Wave form:



PART 4&5

Verilog:

(The monitor output is to get all the initial wires to monitor the full functionality)

```
`timescale 1ns/1ns

module counter_127(input [126:0]A,output [6:0]S,[6:0][63:0]Monitor);
    wire [6:0][63:0]g;
    wire [5:0][31:0]C;
    assign C[0] = A[95:64];
    assign C[1][15:0] = A[111:96];
    assign C[2][7:0] = A[119:112];
    assign C[3][3:0] = A[123:120];
    assign C[4][1:0] = A[125:124];
    assign C[5][0] = A[126];
    assign g[0]= A[63:0];

    genvar i;
    genvar j;
    generate
        for (i = 0 ;i<6;i = i+1) begin: OUTER
            for (j = 0 ;j<64/2**(i +1);j = j+1) begin: INNER
            n_bit_adder #(i+1) inst(
                .A(g[i][2*(i+1)*j+ i:2*(i+ 1)*j]),
                .B(g[i][2*(i+1)*j + (2*i+1): 2*(i+ 1)*j+ i+1]),
                .Ci(C[i][j]),
                .S(g[i+1][(i+2)*j+ i:(i+2)*j]),
                .Co(g[i+1][(i+2)*j+ i+1]));
            end
        end
    endgenerate
    assign S = g[6][6:0];
    assign Monitor = g;
endmodule
```

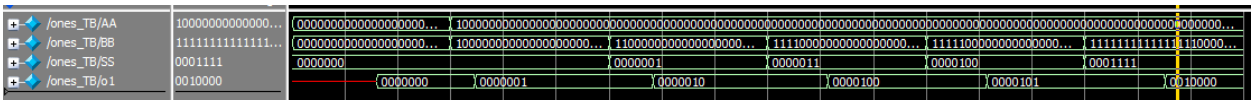
TestBench

(This is for part 4,5,6)

```
`timescale 1ns/1ns

module ones_TB();
    logic [126:0] AA;
    wire [126:0] BB;
    wire [6:0][63:0]M;
    logic [6:0] SS = 7'b0;
    wire [6:0] o1;
    wire [6:0] o2;
    counter_127 O1(.A(BB),.S(o1),.Monitor(M));
    ones_synth O2(BB,o2);
    shifter S1(AA,SS,BB);
    initial begin
        #0 AA = 127'b0;
        #1500 AA[126] = 1;
        #1500 SS = 7'b0000001;
        #1500 SS = 7'b0000011;
        #1500 SS = 7'b0000100;
        #1500 SS = 7'b0001111;
        #1500 $stop;
    end
endmodule
```

Output:



For this part an arithmetic shifter is used to make marching 1 input.

Number of ones is equal to SS+1(initial left most bit)

So the output should always be SS +1 which is satisfied in the wave form.

The worst case delay is 798ns.

PART 6

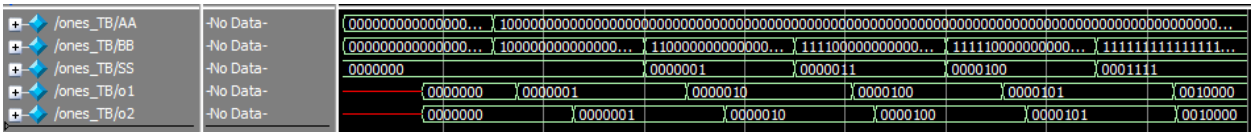
Verilog:

```
`timescale 1ns/1ns

module ones_synth(input [126:0]A,output logic [6:0]S);
    logic [6:0] C;
    int i;
    always @ (A) begin
        C = 7'b0;
        for (i = 0;i<128;i = i+1) begin
            if (A[i] == 1'b1) C = C+1;
        end
    end
    assign #798 S = C;
endmodule
```

Output:

(o2 is for always statement and o1 is for generate(part4))



As it is obvious although all the outputs are the same, but always statement couldn't handle delays itself and to implement the delays we shough use an assign statement.

But it doesn't work precisely because we should put the worst case delay for the output and this causes most of the transitions to be different it terms of delay.

PART 7

The output for behaivioral structure is:

```
ABC RESULTS:              NAND cells:     1318
ABC RESULTS:               NOR cells:     1810
ABC RESULTS:               NOT cells:      534
ABC RESULTS:        internal signals:     3187
ABC RESULTS:           input signals:      127
ABC RESULTS:          output signals:        7
```

```
Number of cells:                3194
  $_AND_                         244
  $_AOI3_                        592
  $_MUX_                         496
  $_NAND_                        134
  $_NOR_                         837
  $_NOT_                          13
  $_OAI3_                          9
  $_OR_                            9
  $_XNOR_                        620
  $_XOR_                         240
```

Modular output:

```
  Number of cells:                646
    $_AND_                         23
    $_AOI3_                        19
    $_NAND_                       195
    $_NOR_                         15
    $_NOT_                         38
    $_OAI3_                       101
    $_OR_                          15
    $_XNOR_                       229
    $_XOR_                         11
```

It can be dedicated that modular output is more efficient that the behavioral output and uses less gates to implement the functionality.