



ADVANCED DIGITAL SYSTEMS DESIGN ENCS3310

Course Project

Name: Faris Abufarha

Student ID: 1200546

Instructor: Dr. **Abdallatif Abuissa**

Section: 2

Date: 22/8/2022

Table of contents

1. introduction and background	3
2. Design philosophy	3
Stage I	5
2.1 MUX 4X1	5
2.2 Full adder	5
2.3 4-bit ripple adder	6
Stage II	7
2.5 Carry look ahead adder :	7
3. Results	7
3.1 MUX 4X1	8
3.2 Full adder	8
3.3 : 4-bit ripple adder	8
3.4 Full test bench	9
Stage II	9
3.5 CLA ADDER	9
3.6 Full test bench 2	10
3.7 Errors	10
3.8 Solutions	11
4. Conclusion and Future works	11
5. Resources and tools	12
5.1 Tools used in this project	12
5.2 Resources	13
APPENDIX	14

1. introduction and background

The aim of this project is to design an Arithmetic Unit (AU), that does multiple arithmetic operations according to the inputs (Fig. 1) using Verilog. The project is split into 2 stages. Stage I which consists of the regular 4-bit ripple adder. Stage II consists of the Carry look ahead adder, inplace of the previous adder; the reason for this is to make the AU faster.

	Select			Input Y	Output $D = A + Y + C_{in}$	Microoperations
	S 1	S 0	Ci n			
1	0	0	0	B	$D = A + B$	Add
2	0	0	1	B	$D = A + B + 1$	Add with carry
3	0	1	0	B^-	$D = A + B^-$	Sub. With borrow
4	0	1	1	B^-	$D = A + B^- + 1$	Sub
5	1	0	0	0	$D = A$	Transfer A
6	1	0	1	0	$D = A + 1$	Increment
7	1	1	0	1	$D = A - 1$	Decrement
8	1	1	1	1	$D = A$	Transfer A

Figure 1: figure 1.1

2. Design philosophy

We were assigned to build the circuit as shown in (Fig. 2). First of all, I started by building the combinational circuit as (4X1 MUX, Full adder, 4-bit ripple adder) structurally using basic gates each with its own given delay as given in (Fig. 3). After building each component I tested each of them with its own test bench, thus the whole system, and so Stage II. Latency of every component was calculated either by trying different cases with correct answers that takes the longest time, or by checking which basic gates working together that counts as one delay, and the ones which depend on the latter added to them (will be explained more in MUX example)

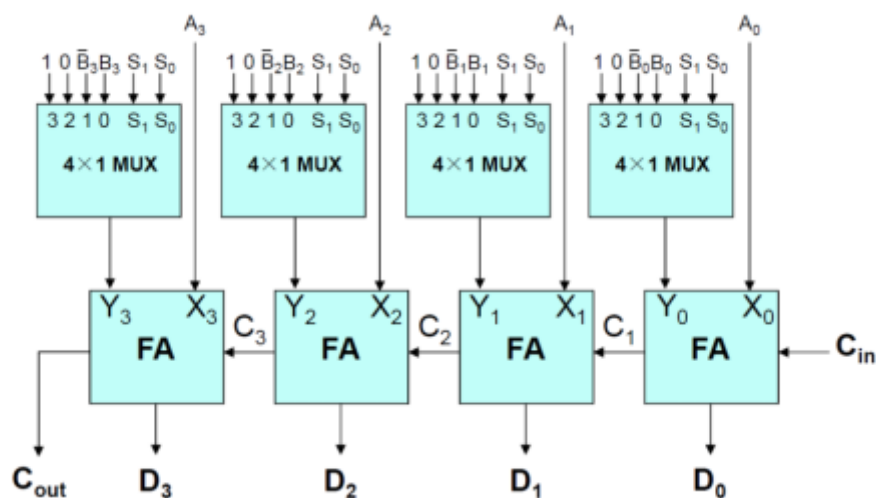


Figure 2: 1.2

Gate	Delay
Inverter	3 ns
NAND	5 ns
NOR	5 ns
AND	7 ns
OR	7 ns
XNOR	9 ns
XOR	11 ns

Figure 3: 1.3

NOTE : instead of typing 'ns' every time i used the compiler directive " `timescale " in the header file

NOTE: in Stage II, i used a direct common clock to the system, TG, and RA. as was said to some student instead of using flip-flops

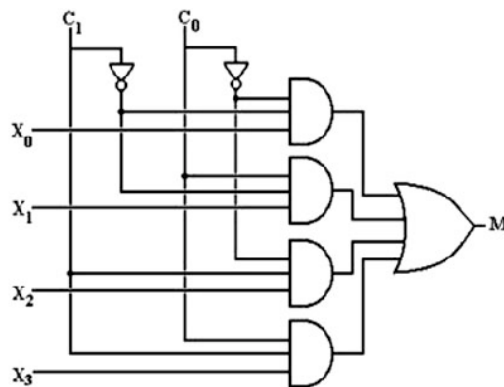
Stage I

Maximum frequency = $1 / \text{clk time} = 1 / 70 \text{ ns} = 0.014285714 * 10^9 \text{ Hz}$

2.1 MUX 4X1

Latency explained: both not gates are working concurrently so counted as one delay (3ns), the 4 and gates working concurrently as well, so (7ns), plus the OR gate. In total = $3 + 7 + 7 = 17\text{ns}$, some few seconds were added for more secure calculations. And so on the left components.

Circuit:



Code :

```
module MUX4X1(f,s, b);
    // TIME NEEDED 3 + 7 + 7 >= 17
    input [0:3] b;
    input [1: 0] s;
    output f;

    wire [0:3] w;
    wire ns1,ns0; // not s1, not s0

    not #3 (ns1, s[1]);
    not #3 (ns0, s[0]);

    and #7 (w[0] , b[0], ns1 ,ns0) , (w[1] , b[1], s[0] ,ns1) , (w[2] , b[2], s[1] ,ns0) , (w[3] , b[3],s[1], s[0])

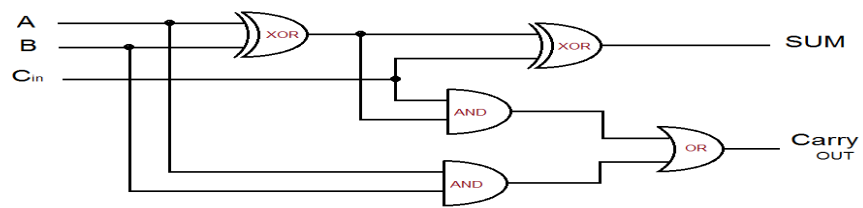
    or #7 (f, w[0], w[1], w[2], w[3]);

endmodule
```

Figure 4: MUX 4X1

2.2 Full adder

Circuit:



Code:

```
// FULL ADDER
module FULLADDER(sum, cout , a, b, cin);
    input a, b, cin;
    output sum, cout;

    wire z1,z2,z3;

    xor #11 (sum,a,b,cin);

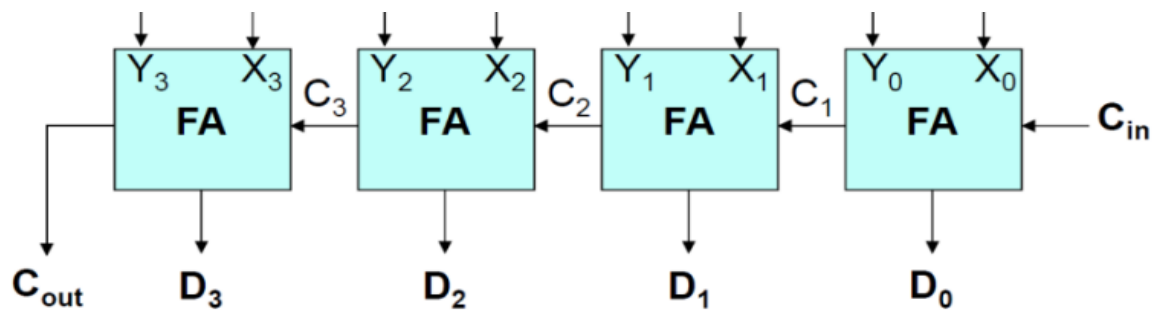
    and #7 (z1, a, b);
    and #7 (z2, a, cin);
    and #7 (z3, b, cin);

    or #7 (cout, z1,z2,z3);

endmodule
```

2.3 4-bit ripple adder

Circuit



Code :

```

module IV_BIT_ADDER(sum,cout,a,b,cin);

    input cin;
    input [3:0] a,b;
    output [3:0] sum;
    output cout;
    wire [2:0] w;

    FULLADDER FA0(.sum(sum[0]), .cout(w[0]), .a(a[0]), .b(b[0]), .cin(cin));
    FULLADDER FA1(.sum(sum[1]), .cout(w[1]), .a(a[1]), .b(b[1]), .cin(w[0]));
    FULLADDER FA2(.sum(sum[2]), .cout(w[2]), .a(a[2]), .b(b[2]), .cin(w[1]));
    FULLADDER FA3(.sum(sum[3]), .cout(cout), .a(a[3]), .b(b[3]), .cin(w[2]));

endmodule

```

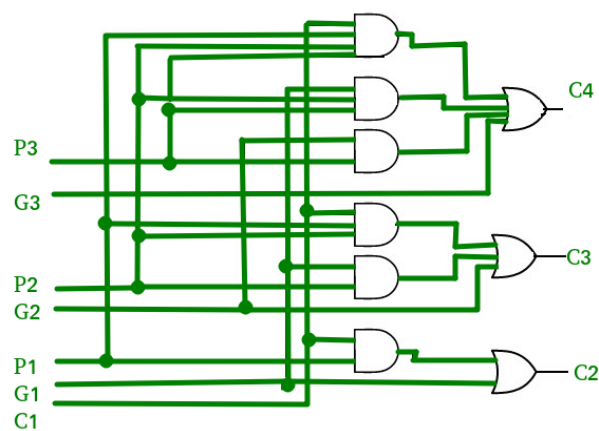
Figure 6: ripple adder

Stage II

Maximum frequency = $1 / \text{clk time} = 1 / 55 \text{ ns} = 0.018181818 \times 10^9 \text{ Hz}$

2.5 Carry look ahead adder :

Circuit & equations



$$\begin{aligned}
 S_i &= P_i \oplus C_i \\
 C_{i+1} &= G_i + P_i C_i
 \end{aligned}$$

Code: -in appendix

-

3. Results

NOTE: for more precision and time saving; i relied on the console outputs not the waveforms

Stage I:

3.1 MUX 4X1

```
# KERNEL: MUX TEST BENCH
# KERNEL: Time 20 select= 00 input = 1010 OUT= 1
# KERNEL:
# KERNEL: Time 40 select= 01 input = 1010 OUT= 0
# KERNEL:
# KERNEL: Time 60 select= 10 input = 1010 OUT= 1
# KERNEL:
# KERNEL: Time 80 select= 11 input = 1010 OUT= 0
# KERNEL:
# RUNTIME: Info: RUNTIME_0068 testbench.sv (29): $finish called.
# KERNEL: Time: 85 ns, Iteration: 0, Instance: /TSTMUX4, Process: @ALWAYS#29_2@.
# KERNEL: stopped at time: 85 ns
```

Figure 8: MUX results

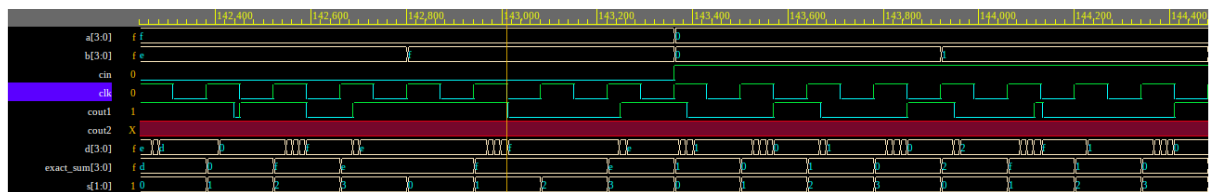
3.2 Full adder

```
# KERNEL: FULL ADDER TST BENCHMARK
# KERNEL: Time 25 input = 0 0 0 SUM= 0 CARRY = 0
# KERNEL:
# KERNEL: Time 50 input = 0 1 0 SUM= 1 CARRY = 0
# KERNEL:
# KERNEL: Time 75 input = 1 0 0 SUM= 1 CARRY = 0
# KERNEL:
# KERNEL: Time 100 input = 1 1 0 SUM= 0 CARRY = 1
# KERNEL:
# KERNEL: Time 125 input = 0 0 1 SUM= 1 CARRY = 0
# KERNEL:
# KERNEL: Time 150 input = 0 1 1 SUM= 0 CARRY = 1
# KERNEL:
# KERNEL: Time 175 input = 1 0 1 SUM= 0 CARRY = 1
# KERNEL:
# KERNEL: Time 200 input = 1 1 1 SUM= 1 CARRY = 1
.. .....
```


3.3 : 4-bit ripple adder

```
# KERNEL: 4 Bit ADDER ISI BENCHMARK2
# KERNEL: Time 69 input = 0000 0000 0 SUM= 0000 CARRY = 0
# KERNEL:
# KERNEL: Time 138 input = 1111 0000 0 SUM= 1111 CARRY = 0
# KERNEL:
# KERNEL: Time 207 input = 1111 1001 0 SUM= 1000 CARRY = 1
# KERNEL:
# KERNEL: Time 276 input = 1111 1001 1 SUM= 1001 CARRY = 1
# KERNEL:
# RUNTIME: Info: RUNTIME_0068 testbench.sv (34): $finish called.
```

3.4 Full test bench

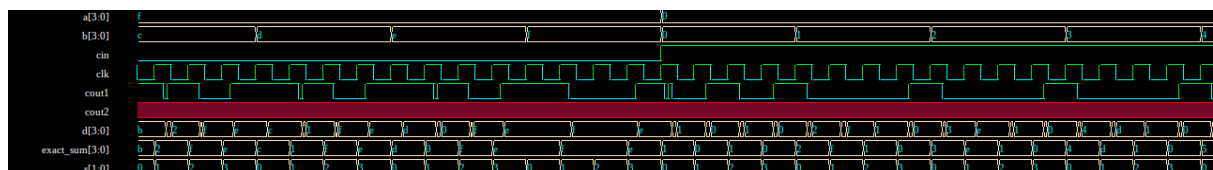
[illegible]

Stage II

3.5 CLA ADDER

```
# KERNEL: Time 44 A = 0000 B = 0000 Cin = 0 Cout = 0 S = 0000
# KERNEL:
# KERNEL: Time 88 A = 0000 B = 0001 Cin = 0 Cout = 0 S = 0001
# KERNEL:
# KERNEL: Time 132 A = 0000 B = 1000 Cin = 0 Cout = 0 S = 1000
# KERNEL:
# KERNEL: Time 176 A = 1110 B = 1000 Cin = 0 Cout = 1 S = 0110
# KERNEL:
# KERNEL: Time 220 A = 1110 B = 0010 Cin = 0 Cout = 1 S = 0000
# KERNEL:
# KERNEL: Time 264 A = 1110 B = 0010 Cin = 1 Cout = 1 S = 0001
# KERNEL:
# KERNEL: Time 308 A = 1110 B = 1111 Cin = 1 Cout = 1 S = 1110
# KERNEL:
# KERNEL: Time 352 A = 1000 B = 1111 Cin = 1 Cout = 1 S = 1000
# KERNEL:
# KERNEL: Time 396 A = 1000 B = 1111 Cin = 1 Cout = 1 S = 1000
# KERNEL:
# KERNEL: Time 440 A = 1000 B = 1111 Cin = 1 Cout = 1 S = 1000
# KERNEL:
# KERNEL: Time 484 A = 1000 B = 1111 Cin = 1 Cout = 1 S = 1000
# KERNEL:
# RUNTIME: Info: RUNTIME_0068 testbench.sv (34): $finish called.
```

3.6 Full test bench 2

[illegible]

3.7 Errors

Some added intended errors:

Stage I:

Lower clock cycle time:



The screenshot shows a Verilog IDE with a code editor on the left and a console on the right. The code editor contains the following Verilog code:

```
455 end
456
457 endmodule
458
459
460 module ANALYZER(exact_sum, prob_sum, clk );
461
462     input clk;
463     inout [4:0] exact_sum;
464
```

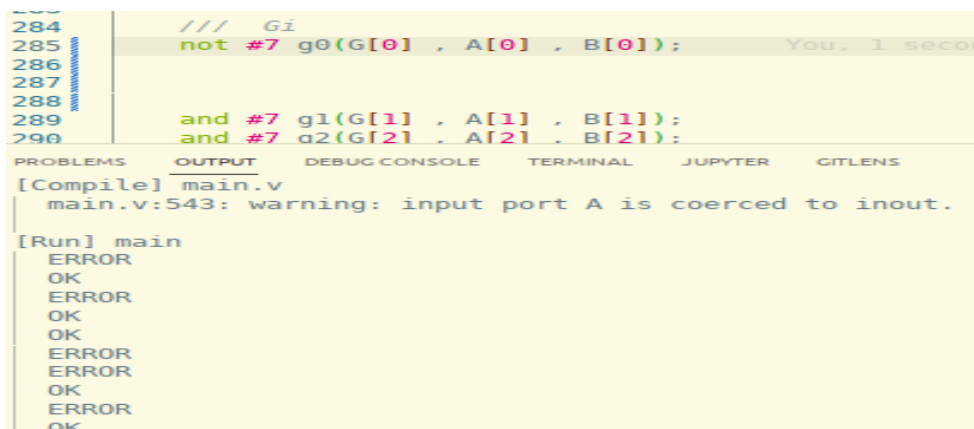
The console on the right shows a list of errors, all of which are "ERROR" messages. The first error is "ERROR" and the last is "ERROR".

For example: here i put the same clock for the ripple adder in Stage I, it gave wrong outputs of course because CLA adder much faster than ripple adder

Stage II:

Intended error: messing with CLA equations

Instead of and, i put not, results:



The screenshot shows a Verilog IDE with a code editor on the left and a console on the right. The code editor contains the following Verilog code:

```
284 /// Gi
285 not #7 g0(G[0] , A[0] , B[0]);
286
287
288 and #7 g1(G[1] , A[1] , B[1]);
289 and #7 g2(G[2] , A[2] , B[2]);
290
```

The console on the right shows a list of errors, all of which are "ERROR" messages. The first error is "ERROR" and the last is "ERROR".

3.8 Solutions

Stage I

Increasing the clock cycle time to its original time, which was 70 ns

Stage II:

Sticking to the original equations

4. Conclusion and Future works

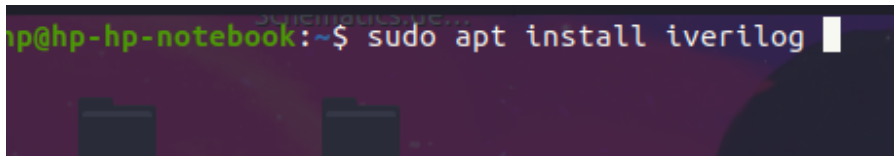
- We used two types of adders, one is a ripple full adder and the other is a carry look ahead adder and we noticed the difference between them in structure, and in speed; which we noticed that CLA adder was about 27.5 % faster than the ripple
- We became more familiar with the Hardware description languages like Verilog, and the concept of design verification (DV)
- We noticed the large difference between hardware speed and other High level programming languages i.e (Python, Java, Js)
- The design can be uploaded on FPGA and work as expected

5. Resources and tools

5.1 Tools used in this project

1. Visual studio code: used as a text editor
https://en.wikipedia.org/wiki/Visual_Studio_Code
2. Git & Github : version control system.
Repo link :
https://github.com/faris771/VERILOG_PROJECT
3. EDA playground: online simulator for verilog
<https://www.edaplayground.com/>
4. Iverilog compiler (verilog compiler for linux machines):

https://iverilog.fandom.com/wiki/Getting_Started



command:

```
iverilog -o hello hello.v
```

5.2 Resources

1. Geeks for Geeks: <https://www.geeksforgeeks.org/carry-look-ahead-adder/>
2. Stack overflow (for solving bugs) : <https://stackoverflow.com/>
3. Some Youtube channels videos like :
<https://www.youtube.com/watch?v=bJ53XErKY8&t=434s>

APPENDIX

Source code:

```
/*  
*   AUTHOR : FARIS ABUFARHA  
*   ID : 1200546  
*   SECTION : 2  
*
```

```
*
*/

`timescale 1ns/1ns // to make ns is the default time unit


/*
    Gate
Delay

Inverter
3 ns

NOR
5 ns

AND
7 ns

OR
7 ns

XNOR
9 ns

XOR
11 ns

*/
```

```

module MUX4X1(f,s, b);

    // TIME NEEDED 3 + 7 +7 >= 17
    input  [0:3] b;
    input  [1: 0] s;
    output  f;

    wire [0:3] w;
    wire ns1,ns0; // not s1, not s0

    not #3 (ns1, s[1]);
    not #3 (ns0, s[0]);

    and #7 (w[0] , b[0], ns1 ,ns0) , (w[1] , b[1], s[0] ,ns1), (w[2] ,
b[2], s[1] ,ns0), (w[3] , b[3],s[1], s[0]);

    or #7  (f, w[0], w[1], w[2], w[3]);

endmodule

module TSTMUX4;

    reg [0:3] b;
    reg [1: 0] s;
    wire f;
    MUX4X1 tst (.b(b), .s(s), .f(f)) ; // order doesn't matter

    initial begin

        $display(" MUX TEST BENCH");

        // $monitor("Time %0d select= %b input = %b OUT=
%b\n",$time,s,b,f);
        {s} = 2'b00;
        b = 4'b1010;

        repeat(4) begin

            #21 {s} = {s} + 2'b01; // 1 ns diff between each iteration
printing the value

```



```

        end

    end

    always #20 $display("Time %0d select= %b input = %b OUT=
    %b\n", $time, s, b, f); // 20 > 17(mux total delation )
    always #85 $finish; // 4 cases each case 20 ns = 80 ns, 85 ns > 80
    ns

endmodule

//===== FULL ADDER =====
module FULLADDER(sum, cout , a, b, cin);

    input a, b, cin;
    output sum, cout;

    wire z1,z2,z3;

    xor #11 (sum,a,b,cin);

    and #7 (z1, a, b);
    and #7 (z2, a, cin);
    and #7 (z3, b, cin);

    or #7 (cout, z1,z2,z3);

endmodule

module FA_TEST;

    reg Tcin,Ta,Tb;
    wire Tsum,Tcout;

```

```

    FULLADDER DUMMY(.sum(Tsum), .cout(Tcout), .a(Ta), .b(Tb),
.cin(Tcin));

    initial begin
        $display("FULL ADDER TST BENCHMARK");
        // total time must be > 25 ns
        {Tcin,Ta,Tb} = 3'b000;

        repeat(7) begin
            #26 {Tcin,Ta,Tb} = {Tcin,Ta,Tb} + 3'b001;
        end

    end

    always #25 $display("Time %0d input = %b %b %b SUM= %b CARRY =
%b\n",$time,Ta,Tb,Tcin,Tsum,Tcout); // 1 sec diff between changing of
Ta and Tb and Tcin and printing the value
    always #500 $finish;

endmodule

//===== 4 bit ADDER =====

module IV_BIT_ADDER(sum,cout,a,b,cin);

    input cin;
    input [3:0] a,b;
    output [3:0] sum;
    output cout;
    wire [2:0] w;

    FULLADDER FA0(.sum(sum[0]), .cout(w[0]), .a(a[0]), .b(b[0]),
.cin(cin));
    FULLADDER FA1(.sum(sum[1]), .cout(w[1]), .a(a[1]), .b(b[1]),
.cin(w[0]));
    FULLADDER FA2(.sum(sum[2]), .cout(w[2]), .a(a[2]), .b(b[2]),
.cin(w[1]));

```

```

    FULLADDER FA3(.sum(sum[3]), .cout(cout), .a(a[3]), .b(b[3]),
.cin(w[2]));

endmodule

module TST_IV_BIT_ADDER; // CHECK FOR BETTER DELAY

    reg cin;
    reg [3:0] a,b;
    wire [3:0] sum;
    wire cout;

    IV_BIT_ADDER DUMMY(.sum(sum), .cout(cout), .a(a), .b(b), .cin(cin));
    initial begin
        $dumpfile("dump.vcd");// for online waveform stuff
        $dumpvars(1);

    end

    initial begin

        $display("4 BIT ADDER TST BENCHMARK2");

        a = 4'b0000;
        b = 4'b0000;
        cin = 1'b0;

        #70 a = 4'b1111;
        #70 b = 4'b1001;
        #70 cin = 1'b1;

    end

    always #69 $display("Time %0d input = %b %b %b SUM= %b CARRY =
%b\n",$time,a,b,cin,sum,cout); // 1 sec diff between changing of Ta
and Tb and Tcin and printing the value
    always #300 $finish;

endmodule

```

```
//===== SYSTEM=====

module SYSTEM(d,cout,a,b,s,cin);

    input cin;
    input [3:0] a,b;
    input [1:0] s;
    output [3:0] d;
    output cout;
    // wire [2:0] c;
    wire [3:0] y;

    MUX4X1 mux0(.b({b[0],~b[0],1'b0,1'b1}), .s(s), .f(y[0]));
    MUX4X1 mux1(.b({b[1],~b[1],1'b0,1'b1}), .s(s), .f(y[1]));
    MUX4X1 mux2(.b({b[2],~b[2],1'b0,1'b1}), .s(s), .f(y[2]));
    MUX4X1 mux3(.b({b[3],~b[3],1'b0,1'b1}), .s(s), .f(y[3]));

    IV_BIT_ADDER ADDER4(.sum(d), .cout(cout), .a(a), .b(y), .cin(cin));

endmodule

// ===== Carry look ahead adder
=====

module CLA_ADDER(S , Cout , A , B , Cin); // SOURCE :
https://www.geeksforgeeks.org/carry-look-ahead-adder/

    input [3:0]A;
    input [3:0]B;
    input Cin;

    output Cout;
    output[3:0]S; // sum

    wire [3:1]C;// C[i + 1 ] = G[i] + P[i]& c[i] , c[0] = cin

    wire [0:3]P; // Pi = Ai xor Bi
    wire [0:3]G; // Gi = Ai and Bi

```

```

/// Gi
and #7 g0(G[0] , A[0] , B[0]);

and #7 g1(G[1] , A[1] , B[1]);
and #7 g2(G[2] , A[2] , B[2]);
and #7 g3(G[3] , A[3] , B[3]);

/// Pi
xor #11 p0(P[0] , A[0] , B[0]);
xor #11 p1(P[1] , A[1] , B[1]);
xor #11 p2(P[2] , A[2] , B[2]);
xor #11 p3(P[3] , A[3] , B[3]);

/// C1
wire tmp1;
and #7 c11(tmp1 , P[0] , Cin);
or #7 c12(C[1] , G[0] , tmp1); // C[1] = G[0] + P[0]& cin

/// C2
wire tmp2; // tmp2 = P[1]& cin
wire tmp3; // tmp3 = P[1]& cin & G[1]
and #7 c21(tmp2 , P[1] , G[0]);
and #7 c22(tmp3 , P[1] , P[0] , Cin);
or #7 c23(C[2] , tmp2 , tmp3 , G[1]);

/// C3
wire tmp4; // tmp4 = P[2]& G[1]
wire tmp5; // tmp5 = P[2]& P[1] & G[0]
wire tmp6; // tmp6 = P[2]& P[1] & P[0] & cin

and #7 c31(tmp4 , P[2] , G[1]);
and #7 c32(tmp5 , P[2] , P[1] , G[0]);
and #7 c33(tmp6 , P[2] , P[1] , P[0] , Cin);
or #7 c34(C[3] , tmp4 , tmp5 , tmp6 , G[2]);

/// Cout (C4)
wire tmp7;
wire tmp8;
wire tmp9;
wire tmp10;

```

```

and #7 c41(tmp7 , P[3] , G[2]);
and #7 c42(tmp8 , P[3] , P[2] , G[1]);
and #7 c43(tmp9 , P[3] , P[2] , P[1] , G[0]);
and #7 c44(tmp10 , P[3] , P[2] , P[1] , P[0] , Cin);
or #7 c45(Cout , tmp7 , tmp8 , tmp9 , tmp10 , G[3]);

/// Sums // SUM = Pi XOR Ci

xor #11 s0(S[0] , P[0] , Cin);
xor #11 s1(S[1] , P[1] , C[1]);
xor #11 s2(S[2] , P[2] , C[2]);
xor #11 s3(S[3] , P[3] , C[3]);

endmodule

module TST_CLA; // SEEMS like 45 ns is the suitable delay
reg [3:0]A;
reg [3:0]B;
reg Cin;

wire Cout;
wire [3:0]S; // sum

CLA_ADDER DUMMY(.S(S), .Cout(Cout), .A(A), .B(B), .Cin(Cin));

initial begin
    $display("CLA TST BENCHMARK");
    A = 4'b0000;
    B = 4'b0000;
    Cin = 1'b0;

    // #34 A = 4'b0001;
    #45 B = 4'b0001;
    // #34 A = 4'b0010;
    #45 B = 4'b1000;
    #45 A = 4'b1110;
    #45 B = 4'b0010;
    #45 Cin = 1'b1;
    #45 B = 4'b1111;
    #45 A = 4'b1000;

end

```

```
    always #44 $display("Time %0d A = %b B = %b Cin = %b Cout = %b S = %b\n", $time, A, B, Cin, Cout, S); // 1 sec diff between changing and printing the value
```

```
    always #500 $finish;
```

```
endmodule
```

```
module TEST_GENERATOR (d,cout,a,b,s,cin,clk);
```

```
    output reg cin;
```

```
    output reg [3:0] a,b;
```

```
    output reg [1:0] s;
```

```
    input clk; // HERE ??
```

```
    integer counter;
```

```
    output reg [3:0] d; // extra bit for carry
```

```
    output reg cout;
```

```
    initial begin
```

```
        counter = 0;
```

```
    end
```

```
    always @(posedge clk) begin
```

```
        if( counter == 2049) // 2048 + 1 extra bit for last case  
            $finish;
```

```
        {cin, a, b,s} = counter;
```

```
        counter = counter + 1;
```

```
        // if ({s,cin} == 3'b000 ) begin
```

```
        // end
```

```
        case({s,cin})
```

```
            4'b0000 : begin d = a + b; cout = d[4];end
```

```
            4'b0001 : begin d = a + b + 1;cout = d[4];end
```

```
            4'b0010 : begin d = a + ~b; cout = d[4];end
```

```

        4'b0011 : begin d = a + ~b + 1 ;cout = d[4];end
        4'b0100 : begin d = a;cout = d[4];end
        4'b0101 : begin d = a +1;cout = d[4];end
        4'b0110 : begin d = a - 1 ;cout = d[4];end
        4'b0111 : begin d = a;cout = d[4];end

    endcase

end

endmodule

module ANALYZER(exact_sum, prob_sum, clk );

    input clk;
    input [4:0] exact_sum;
    input [4:0] prob_sum; // extra bit for carry

    always @( negedge clk) begin // generator is on positive edge,
analyzer must be on negative edge

        if( exact_sum != prob_sum ) begin
            // $display("\033[1m"); bold
            $display("ERROR");
            // $display("\033[0m");
        end
        else begin
            $display("OK");
        end

    end

end

endmodule

module FULL_TEST_R;

    wire cin;
    wire [3:0] a,b;
    wire [1:0] s;

```



```

    reg clk;

    initial begin
        clk = 1;
    end

    // =====

    wire [3:0] d; // prob sum
    wire cout1;

    wire [3:0] exact_sum; //
    wire cout2;

    TEST_GENERATOR TG(.cin(cin), .a(a), .b(b), .s(s), .d(exact_sum),
.cout(cout2), .clk(clk));

    SYSTEM DUMMY(.d(d), .cout(cout1), .a(a), .b(b), .s(s), .cin(cin));
    ANALYZER ANLZ(.exact_sum({cout2,exact_sum}),.prob_sum({cout1,d})
,.clk(clk));

    always #70 clk = ~clk; // By trial : 70 ns is a suitable latency for
the system to finish correctly

endmodule

module SYSTEM_CLA(d,cout,a,b,s,cin);

    input cin;
    input [3:0] a,b;
    input [1:0] s;
    output [3:0] d;
    output cout;
    // wire [2:0] c;
    wire [3:0] y;

```

```

MUX4X1 mux0(.b({b[0],~b[0],1'b0,1'b1}), .s(s), .f(y[0]));
MUX4X1 mux1(.b({b[1],~b[1],1'b0,1'b1}), .s(s), .f(y[1]));
MUX4X1 mux2(.b({b[2],~b[2],1'b0,1'b1}), .s(s), .f(y[2]));
MUX4X1 mux3(.b({b[3],~b[3],1'b0,1'b1}), .s(s), .f(y[3]));

CLA_ADDER ADDER4(.S(d), .Cout(cout), .A(a), .B(y), .Cin(cin));

endmodule

module FULL_TEST_CLA;

    wire cin;
    wire [3:0] a,b;
    wire [1:0] s;

    reg clk;

    initial begin
        clk = 1;
    end

    // =====

    wire [3:0] d; // prob sum
    wire cout1;

    wire [3:0] exact_sum; //
    wire cout2;

    TEST_GENERATOR TG(.cin(cin), .a(a), .b(b), .s(s), .d(exact_sum),
.cout(cout2),.clk(clk));

    SYSTEM_CLA DUMMY(.d(d), .cout(cout1), .a(a), .b(b), .s(s),
.cin(cin));
    ANALYZER ANLZ(.exact_sum({cout2,exact_sum}),.prob_sum({cout1,d})
,.clk(clk));

    always #55 clk = ~clk; //55 GOOD

```

```
endmodule
```

My gratitude to you for all efforts you have done during the semester, and I hope you liked this humble report.