



Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS533

Project Report

Prepared by : Tareq Shannak

ID Number : 1181404

Instructor : Dr. Abdallatif Abuissa

Section : 2

Date : 15/12/2020

Table of Contents

Brief Introduction and Background	III
Design philosophy	IV
Basic Gates.....	IV
One-Bit Full Adder	IV
Stage 1	IV
Stage 2	V
Four-Bit Full Adder	V
Stage 1	V
Stage 2	V
BCD Adder.....	V
System (2-Digit BCD Adder)	VI
Test Generator	VI
Result Analyzer	VII
Built In Self Test	VII
Results	VIII
Stage 1	VIII
Stage 2	IX
Conclusion and Future works	X
Appendix	XI

Brief Introduction and Background

In this project, we will implement a 2-Digit BCD Adder in two stages. This system will be constructed from little entities which also will be constructed from a minimum entities. We will use the basic gates with certain delays as shown in Table 1. In some cases, we can use NAND gates instead of another one (like XOR gates) because we can reach to our function with small delays using NAND gates, we will use them in 1-bit full adder. The basic gates will have the same delay regardless of the number of inputs.

Gate	Delay
Inverter	3 NS
NAND	6 NS
NOR	6 NS
AND	8 NS
OR	8 NS
XNOR	10 NS
XOR	12 NS

Table 1

From the basic gates given above, we will built 1-bit full adder, hence, we can build 4-bit full adder which can be used in implement 1-BCD adder, and we can implement a system from the BCD adder to make a 2-Digit BCD Adder.

The two stages that we will build the system: first, using a ripple full adder which we had learned in digital course, the second stage is using an adder with carry look ahead generator. We will calculate the duration of delay for each stage to use in testing the outputs.

There will be a Built In Self-Test (BIST) for each stage that has two registers: Test generator which sends the inputs to our system and send the outputs to the second register and it has a clock input, the second register is the Result analyzer which receives the behavioral output from the test generator and receives the output of the system, it make sure that the two outputs are correct.

We will use Aldec Active-HDL Student Edition to simulate our system, testing it and resulting the outputs correctly.

Design philosophy

Basic Gates

First, we implemented the basic gates as shown in Figure 1, there was an idea to make one entity for each basic gates and make it generic with N variable which specifies the number of inputs, however it is difficult to implement and call the entity of the gate in other entities.

```
7  -----AND2-----
8  LIBRARY ieee;
9  USE ieee.std_logic_1164.ALL;
10
11  ENTITY AND2 IS
12    PORT(A,B: IN STD_LOGIC:= '0';
13         F: OUT STD_LOGIC:= '0');
14  END AND2;
15
16  ARCHITECTURE structural OF AND2 IS
17  BEGIN
18    F <= A AND B AFTER 8 NS;
19  END;
20
21  -----AND3-----
22  LIBRARY ieee;
23  USE ieee.std_logic_1164.ALL;
24
25  ENTITY AND3 IS
26    PORT(A,B,C: IN STD_LOGIC:= '0';
27         F: OUT STD_LOGIC:= '0');
28  END AND3;
29
30  ARCHITECTURE structural OF AND3 IS
31  BEGIN
32    F <= A AND B AND C AFTER 8 NS;
33  END;
34
35  -----AND4-----
36  LIBRARY ieee;
37  USE ieee.std_logic_1164.ALL;
38
39  ENTITY AND4 IS
40    PORT(A,B,C,D: IN STD_LOGIC:= '0';
41         F: OUT STD_LOGIC:= '0');
42  END AND4;
43
44  ARCHITECTURE structural OF AND4 IS
45  BEGIN
46    F <= A AND B AND C AND D AFTER 8 NS;
47  END;
48
49  -----AND5-----
50  LIBRARY ieee;
51  USE ieee.std_logic_1164.ALL;
52
53  ENTITY AND5 IS
54    PORT(A,B,C,D,E: IN STD_LOGIC:= '0';
55         F: OUT STD_LOGIC:= '0');
56  END AND5;
57
58  ARCHITECTURE structural OF AND5 IS
59  BEGIN
60    F <= A AND B AND C AND D AND E AFTER 8 NS;
61  END;
62
63  -----OR2-----
64  LIBRARY ieee;
65  USE ieee.std_logic_1164.ALL;
66
67  ENTITY OR2 IS
68    PORT(A,B: IN STD_LOGIC:= '0';
69         F: OUT STD_LOGIC:= '0');
70  END OR2;
71
72  ARCHITECTURE structural OF OR2 IS
73  BEGIN
74    F <= A OR B AFTER 8 NS;
75  END;
76
77  -----OR3-----
78  LIBRARY ieee;
79  USE ieee.std_logic_1164.ALL;
80
81  ENTITY OR3 IS
82    PORT(A,B,C: IN STD_LOGIC:= '0';
83         F: OUT STD_LOGIC:= '0');
84  END OR3;
85
86  ARCHITECTURE structural OF OR3 IS
87  BEGIN
88    F <= A OR B OR C AFTER 8 NS;
89  END;
90
91  -----OR4-----
92  LIBRARY ieee;
93  USE ieee.std_logic_1164.ALL;
94
95  ENTITY OR4 IS
96    PORT(A,B,C,D: IN STD_LOGIC:= '0';
97         F: OUT STD_LOGIC:= '0');
98  END OR4;
99
100  ARCHITECTURE structural OF OR4 IS
101  BEGIN
102    F <= A OR B OR C OR D AFTER 8 NS;
103  END;
104
105  -----OR5-----
106  LIBRARY ieee;
107  USE ieee.std_logic_1164.ALL;
108
109  ENTITY OR5 IS
110    PORT(A,B,C,D,E: IN STD_LOGIC:= '0';
111         F: OUT STD_LOGIC:= '0');
112  END OR5;
113
114  ARCHITECTURE structural OF OR5 IS
115  BEGIN
116    F <= A OR B OR C OR D OR E AFTER 8 NS;
117  END;
118
119  -----XOR2-----
120  LIBRARY ieee;
121  USE ieee.std_logic_1164.ALL;
122
123  ENTITY XOR2 IS
124    PORT(A,B: IN STD_LOGIC:= '0';
125         F: OUT STD_LOGIC:= '0');
126  END XOR2;
127
128  ARCHITECTURE structural OF XOR2 IS
129  BEGIN
130    F <= A XOR B AFTER 12 NS;
131  END;
132
133  -----NAND2-----
134  LIBRARY ieee;
135  USE ieee.std_logic_1164.ALL;
136
137  ENTITY NAND2 IS
138    PORT(A,B: IN STD_LOGIC:= '0';
139         F: OUT STD_LOGIC:= '0');
140  END NAND2;
141
142  ARCHITECTURE structural OF NAND2 IS
143  BEGIN
144    F <= A NAND B AFTER 6 NS;
145  END;
```

Figure 1 - Basic Gates

One-Bit Full Adder

Stage 1

This full adder is implemented as shown in Figure 2 to use later in building ripple Full Adder, we used NAND gates to decrease the time of delay.

```
138  -----One Bit Full Adder-----
139  LIBRARY ieee;
140  USE ieee.std_logic_1164.ALL;
141
142  ENTITY FA IS
143    PORT(A,B,Carryin: IN STD_LOGIC:= '0';
144         Sum,Carryout: OUT STD_LOGIC:= '0');
145  END FA;
146
147  ARCHITECTURE structural OF FA IS
148  SIGNAL T,R,E,Q,S,H,K: STD_LOGIC := '0';
149  BEGIN
150    -- Full Adder Using NAND gates (Faster)
151    G1: ENTITY WORK.NAND2(structural) PORT MAP(A,B,T);
152    G2: ENTITY WORK.NAND2(structural) PORT MAP(T,A,R);
153    G3: ENTITY WORK.NAND2(structural) PORT MAP(T,B,E);
154    G4: ENTITY WORK.NAND2(structural) PORT MAP(R,E,Q);
155    G5: ENTITY WORK.NAND2(structural) PORT MAP(Q,Carryin,S);
156    G6: ENTITY WORK.NAND2(structural) PORT MAP(S,T,Carryout);
157    G7: ENTITY WORK.NAND2(structural) PORT MAP(S,Carryin,H);
158    G8: ENTITY WORK.NAND2(structural) PORT MAP(S,Q,K);
159    G9: ENTITY WORK.NAND2(structural) PORT MAP(H,K,Sum);
160    -- Another way to implement Full Adder (Slower)
161    --G1: ENTITY WORK.XOR2(structural) PORT MAP(A,B,X);
162    --G2: ENTITY WORK.XOR2(structural) PORT MAP(X,Carryin,Sum);
163    --G3: ENTITY WORK.AND2(structural) PORT MAP(X,Carryin,Y);
164    --G4: ENTITY WORK.AND2(structural) PORT MAP(A,B,Z);
165    --G5: ENTITY WORK.OR2(structural) PORT MAP(Y,Z,Carryout);
166  END;
```

Figure 2 - 1-Bit FA

Stage 2

In stage 2, we didn't use one-bit Full Adder, we just build a carry lookahead generator as shown in Figure 3 to use it in building a Full Adder with lookahead carry in Four-Bit. This generator generates a carry for each one-bit Full Adder concurrently without depending on the previous carry, just depend on the first carry (Carryin) using two signals: G called carry generate and P called carry propagate.

```
170
171 -----Carry Lookahead Generator-----
172
173 LIBRARY ieee;
174 USE ieee.std_logic_1164.ALL;
175
176 ENTITY carryGenerator IS
177   PORT(P,G: IN STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
178        Carryin: IN STD_LOGIC:='0';
179        Carryout: OUT STD_LOGIC_VECTOR(4 DOWNTO 0):="00000");
180 END carryGenerator;
181
182 ARCHITECTURE carryLookahead OF carryGenerator IS
183   SIGNAL C,GC: STD_LOGIC_VECTOR(4 DOWNTO 0):="00000";
184   SIGNAL PG: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
185   SIGNAL PPG: STD_LOGIC_VECTOR(3 DOWNTO 1):="0000";
186   SIGNAL PPPG: STD_LOGIC_VECTOR(3 DOWNTO 2):="00";
187   SIGNAL PPPPG: STD_LOGIC:='0';
188   BEGIN
189     GC <= G(3 DOWNTO 0)&C(0);
190     C(0) <= Carryin;
191     Carryout <= C;
192
193     gen1: FOR i IN 0 TO 3 GENERATE
194     BEGIN
195       G: ENTITY WORK.AND2(structural) PORT MAP(P(i),GC(i),PG(i));
196     END GENERATE;
197
198     gen2: FOR i IN 1 TO 3 GENERATE
199     BEGIN
200       G: ENTITY WORK.AND3(structural) PORT MAP(P(i),P(i-1),GC(i-1),PPG(i));
201     END GENERATE;
202
203     gen3: FOR i IN 2 TO 3 GENERATE
204     BEGIN
205       G: ENTITY WORK.AND4(structural) PORT MAP(P(i),P(i-1),P(i-2),GC(i-2),PPPG(i));
206     END GENERATE;
207
208     A1: ENTITY WORK.OR2(structural) PORT MAP(G(0), PG(0), C(1));
209     A2: ENTITY WORK.OR3(structural) PORT MAP(GC(2), PG(1), PPG(1), C(2));
210     A3: ENTITY WORK.OR4(structural) PORT MAP(GC(3), PG(2), PPG(2), PPPG(2), C(3));
211     A4: ENTITY WORK.AND5(structural) PORT MAP(P(3), P(2), P(1), P(0), GC(0), PPPPG);
212     A44: ENTITY WORK.OR5(structural) PORT MAP(GC(4), PG(3), PPG(3), PPPG(3), PPPPG, C(4));
213 END;
```

Figure 3 - Carry Lookahead Generator

Four-Bit Full Adder

Figure 4 shows the structure of 4-Bit adder entity with 2 architectures (stages).

Stage 1

We built the 4-bit ripple full adder using 1-bit full adder entity.

Stage 2

We built 4-bit lookahead carry full adder using the carry generator and for loop to calculate the sum in each bit individually.

```
214
215 -----Four Bit Full Adder-----
216
217 LIBRARY ieee;
218 USE ieee.std_logic_1164.ALL;
219
220 ENTITY FA4 IS
221   PORT(A,B: IN STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
222        Carryin: IN STD_LOGIC:='0';
223        Sum: OUT STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
224        Carryout: OUT STD_LOGIC:='0');
225 END FA4;
226
227 ----- 4 Bit Ripple FA -----
228
229 ARCHITECTURE ripple OF FA4 IS
230   SIGNAL C: STD_LOGIC_VECTOR(4 DOWNTO 0);
231   BEGIN
232     C(0) <= Carryin;
233     Carryout <= C(4);
234     gen1: FOR i IN 0 TO 3 GENERATE
235     BEGIN
236       G: ENTITY WORK.FA(structural) PORT MAP(A(i),B(i),C(i),Sum(i),C(i+1));
237     END GENERATE;
238   END;
239
240 ----- 4 Bit Adder With Carry Lookahead -----
241
242 ARCHITECTURE lookahead OF FA4 IS
243   SIGNAL C: STD_LOGIC_VECTOR(4 DOWNTO 0):="00000";
244   SIGNAL P,G: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
245   BEGIN
246     C(0) <= Carryin;
247     Carryout <= C(4);
248     GG: ENTITY WORK.carryGenerator(carryLookahead) PORT MAP(P,G,Carryin,C);
249     gen1: FOR i IN 0 TO 3 GENERATE
250     BEGIN
251       G1: ENTITY WORK.XOR2(structural) PORT MAP(A(i),B(i),P(i));
252       G2: ENTITY WORK.AND2(structural) PORT MAP(A(i),B(i),G(i));
253       G3: ENTITY WORK.XOR2(structural) PORT MAP(P(i),C(i),Sum(i));
254     END GENERATE;
255   END;
```

Figure 4 - 4-Bit FA

BCD Adder

Figure 5 shows how we built the BCD adder in the two stages, in the two stages they are the same idea in writing the code.

```

257 -----BCD Adder-----
258
259 LIBRARY ieee;
260 USE ieee.std_logic_1164.ALL;
261
262 ENTITY BCD IS
263   PORT(A,B: IN STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
264         Carryin: IN STD_LOGIC:= '0';
265         Sum: OUT STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
266         Carryout: OUT STD_LOGIC:= '0');
267 END BCD;
268
269 -----Ripple BCD Adder-----
270
271 ARCHITECTURE ripple OF BCD IS
272   SIGNAL AA,BB: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
273   SIGNAL X,Y,Z,R: STD_LOGIC:= '0';
274 BEGIN
275   BB(1) <= BB(2);
276   Carryout <= BB(2);
277   G1: ENTITY WORK.FA4(ripple) PORT MAP(A,B,Carryin,AA,X);
278   G2: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(2),Y);
279   G3: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(1),Z);
280   G4: ENTITY WORK.OR3(structural) PORT MAP(X,Y,Z,BB(2));
281   G5: ENTITY WORK.FA4(ripple) PORT MAP(AA,BB,'0',SUM,R);
282 END;
283
284 -----BCD Adder With Carry Lookahead-----
285
286 ARCHITECTURE lookahead OF BCD IS
287   SIGNAL AA,BB: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
288   SIGNAL X,Y,Z,R: STD_LOGIC:= '0';
289 BEGIN
290   BB(1) <= BB(2);
291   Carryout <= BB(2);
292   G1: ENTITY WORK.FA4(lookahead) PORT MAP(A,B,Carryin,AA,X);
293   G2: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(2),Y);
294   G3: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(1),Z);
295   G4: ENTITY WORK.OR3(structural) PORT MAP(X,Y,Z,BB(2));
296   G5: ENTITY WORK.FA4(lookahead) PORT MAP(AA,BB,'0',SUM,R);
297 END;
300 ----- System -----
301
302 LIBRARY ieee;
303 USE ieee.std_logic_1164.ALL;
304
305 ENTITY System IS
306   PORT(A,B: IN STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
307         Sum: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
308         Carryout: OUT STD_LOGIC:= '0');
309 END System;
310
311 ----- System With Ripple Adder -----
312
313 ARCHITECTURE ripple OF System IS
314   SIGNAL X: STD_LOGIC;
315 BEGIN
316   G1: ENTITY WORK.BCD(ripple) PORT MAP(A(3 DOWNTO 0),B(3 DOWNTO 0),'0',Sum(3 DOWNTO 0),X);
317   G2: ENTITY WORK.BCD(ripple) PORT MAP(A(7 DOWNTO 4),B(7 DOWNTO 4),X,Sum(7 DOWNTO 4),Carryout);
318 END;
319
320 ----- System With Carry Lookahead Adder -----
321
322 ARCHITECTURE lookahead OF System IS
323   SIGNAL X: STD_LOGIC;
324 BEGIN
325   G1: ENTITY WORK.BCD(lookahead) PORT MAP(A(3 DOWNTO 0),B(3 DOWNTO 0),'0',Sum(3 DOWNTO 0),X);
326   G2: ENTITY WORK.BCD(lookahead) PORT MAP(A(7 DOWNTO 4),B(7 DOWNTO 4),X,Sum(7 DOWNTO 4),Carryout);
327 END;

```

Figure 5 - BCD Adder (1-Digit and 2-Digit)

System (2-Digit BCD Adder)

Figure 5 shows how we built the system in the two stages, in the two stages they are the same idea in writing the code.

Test Generator

This generator as shown on Figure 6 has a clock input, A, B inputs and the correct output, there are 2 processes in its generator's architecture: the first process resulting the correct output in behavioral logic, and the second one changes (increment) the values of A and B when the clock input rises to reach all possible inputs.

```

329 ----- Test Generator -----
330
331 LIBRARY ieee;
332 USE ieee.std_logic_1164.ALL;
333 USE ieee.std_logic_ARITH.ALL;
334 USE ieee.std_logic_UNSIGNED.ALL;
335
336 ENTITY TestGenerator IS
337   PORT(CLK: IN STD_LOGIC:= '0';
338         A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
339         Correct: OUT STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000");
340 END TestGenerator;
341
342 ARCHITECTURE generator OF TestGenerator IS
343   SIGNAL AA,BB: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
344 BEGIN
345   A<=AA;
346   B<=BB;
347   -- The Process Below shows how to implement the system using
348   PROCESS (AA,BB)
349     VARIABLE X: STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000";
350     BEGIN
351       X(4 DOWNTO 0) := ('0'&AA(3 DOWNTO 0)) + ('0'&BB(3 DOWNTO 0));
352       if(X(4 DOWNTO 0) > "01001") then
353         X(3 DOWNTO 0) := X(3 DOWNTO 0)+ "0110";
354         X(4) := '1';
355       end if;
356       X(8 DOWNTO 4) := ("0000"&X(4)) + ('0'&AA(7 DOWNTO 4)) + ('0'&BB(7 DOWNTO 4));
357       if(X(8 DOWNTO 4) > "01001") then
358         X(7 DOWNTO 4) := X(7 DOWNTO 4)+ "0110";
359         X(8) := '1';
360       end if;
361       CORRECT<=X;
362     END PROCESS;
363
364 -- The Process below changes the values of AA and BB(=> A and B) when
365 -- (A = 0, B = 0) => (99, 99)
366 PROCESS
367 BEGIN
368   FOR i IN 0 TO 9 LOOP
369     FOR j IN 0 TO 9 LOOP
370       FOR K IN 0 TO 9 LOOP
371         FOR L IN 0 TO 9 LOOP
372           AA(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(i,4);
373           AA(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,4);
374           BB(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(K,4);
375           BB(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(L,4);
376           WAIT UNTIL rising_edge(CLK);
377         END LOOP;
378       END LOOP;
379     END LOOP;
380   END LOOP;
381   WAIT;
382 END PROCESS;
383 END;

```

Figure 6 - Test Generator

Result Analyzer

This analyzer ensures that the outputs are correct when the clock input rises as shown on Figure 7.

```
384 ----- Result Analyser -----
385 ----- Result Analyser -----
386
387 LIBRARY ieee;
388 USE ieee.std_logic_1164.ALL;
389 USE ieee.std_logic_ARITH.ALL;
390
391 ENTITY ResultAnalyser IS
392   PORT(CLK: IN STD_LOGIC:= '0';
393         Correct,myResult: IN STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000");
394 END ResultAnalyser;
395
396 ARCHITECTURE analyser OF ResultAnalyser IS
397 BEGIN
398   -- The Process below make sure that the resulting output from our system equals to the correct one
399   -- otherwise, print an error when the outputs are not equal to each other
400   PROCESS
401   BEGIN
402     assert (myResult = Correct)
403     report "The results that were obtained don't agree with the theoretical results"
404     severity ERROR;
405     WAIT UNTIL rising_edge(CLK);
406   END PROCESS;
407 END;
```

Figure 7 - Result Analyzer

Built In Self Test

This entity as shown on Figure 8 - in the two stages - has the whole system with a test generator and result analyzer, the clock signal inverses after a certain time and the test generator changes A and B signals and send the correct output to the result analyzer. The outputs A and B send to the system that will resulting output, this output will go to the result analyzer that assert if this output is correct or not depending on the correct result of the test generator. The generator and analyzer will have the same clock signal. The difference between the two stages is the delay.

```
408 ----- Built In Self Test -----
409 ----- Built In Self Test -----
410
411 LIBRARY ieee;
412 USE ieee.std_logic_1164.ALL;
413 USE ieee.std_logic_ARITH.ALL;
414
415 ENTITY BIST IS
416 END BIST;
417
418 ----- Test For The Ripple System -----
419 ARCHITECTURE ripple OF BIST IS
420   SIGNAL CLK: STD_LOGIC:= '0';
421   SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
422   SIGNAL Correct,myResult: STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000";
423 BEGIN
424   -- 85*2 = 170 ns is the minimum delay we should have to have a correct output
425   CLK <= NOT CLK AFTER 85 NS;
426   G1: ENTITY WORK.TestGenerator(generator) PORT MAP(CLK, A, B, Correct);
427   G2: ENTITY WORK.System(ripple) PORT MAP(A, B, myResult(7 DOWNTO 0), myResult(8));
428   G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(CLK, Correct, myResult);
429 END;
430
431 ----- Test For The Carry Lookahead System -----
432 ARCHITECTURE lookahead OF BIST IS
433   SIGNAL CLK: STD_LOGIC:= '0';
434   SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
435   SIGNAL Correct,myResult: STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000";
436 BEGIN
437   -- 68*2 = 136 ns is the minimum delay we should have to have a correct output
438   CLK <= NOT CLK AFTER 68 NS;
439   G1: ENTITY WORK.TestGenerator(generator) PORT MAP(CLK, A, B, Correct);
440   G2: ENTITY WORK.System(lookahead) PORT MAP(A, B, myResult(7 DOWNTO 0), myResult(8));
441   G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(CLK, Correct, myResult);
442 END;
```

Figure 8 – BIST

Results

The whole code is in Appendix.

Stage 1

We can notice that the minimum possible time to prevent the delay problems is 170 nanoseconds, it will not print an error.

```
----- Test For The Ripple System -----  
ARCHITECTURE ripple OF BIST IS  
SIGNAL CLK: STD_LOGIC:= '0';  
SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0);  
SIGNAL Correct,myResult: STD_LOGIC_VECTOR  
BEGIN  
    -- 85*2 = 170 ns is the minimum delay  
    CLK <= NOT CLK AFTER 85 NS;  
    G1: ENTITY WORK.TestGenerator(generat  
    G2: ENTITY WORK.System(ripple) PORT M  
    G3: ENTITY WORK.ResultAnalyser(analys  
END;
```

These are some screenshots for the simulation shows the results and the difference between the behavioral output and the actual output.

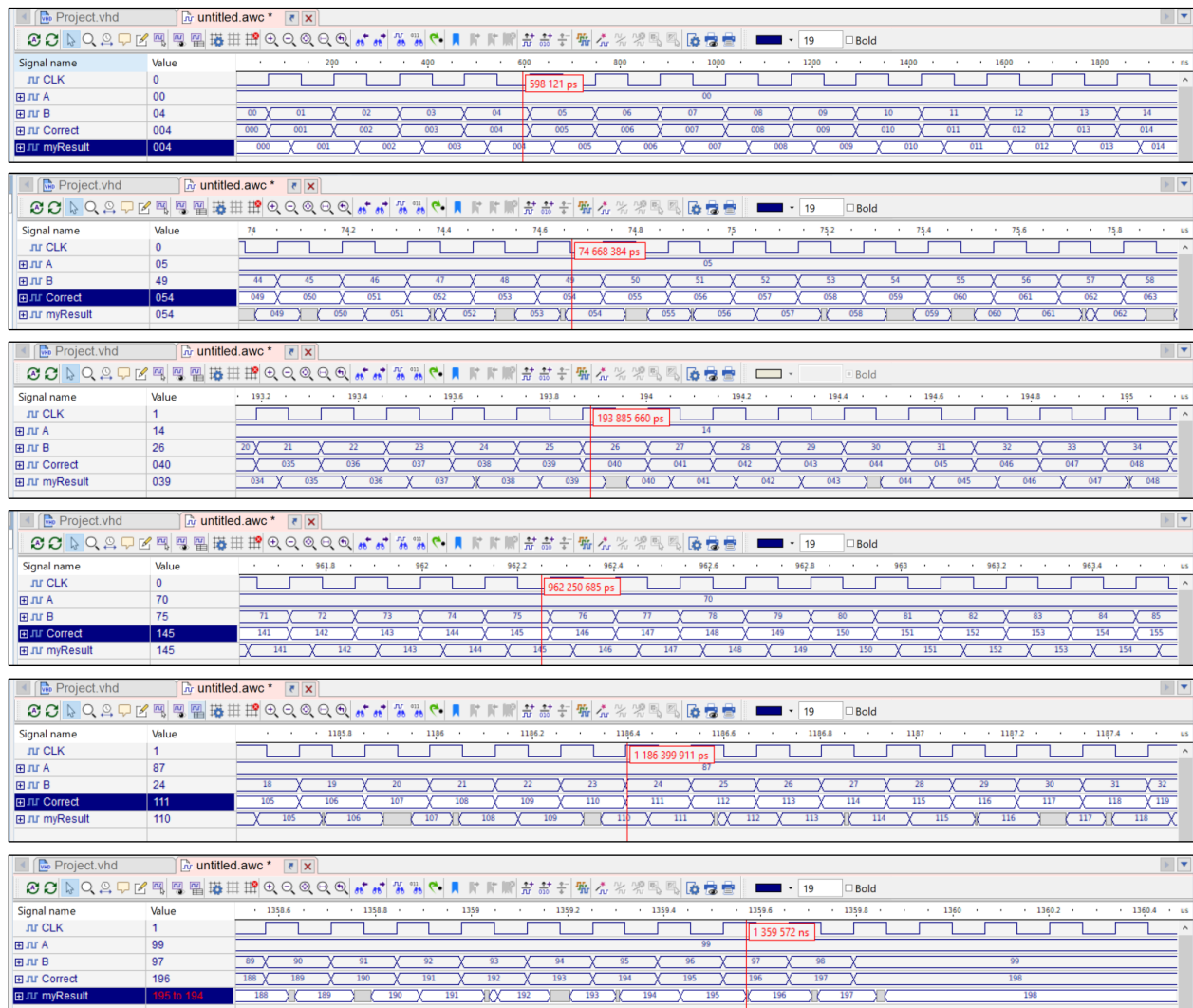


Stage 2

We can notice that the minimum possible time to prevent the delay problems is 136 nanoseconds, it will not print an error.

```
----- Test For The Carry Lookahead S
ARCHITECTURE lookahead OF BIST IS
SIGNAL CLK: STD_LOGIC:= '0';
SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Correct,myResult: STD_LOGIC_V
BEGIN
-- 68*2 = 136 ns is the minimum
CLK <= NOT CLK AFTER 68 NS;
G1: ENTITY WORK.TestGenerator(ge
G2: ENTITY WORK.System(lookahead
G3: ENTITY WORK.ResultAnalyser(a
END;
```

These are some screenshots for the simulation shows the results and the difference between the behavioral output and the actual output.



Conclusion and Future works

The results that were obtained from the previous procedures agree with the theoretical results. Moreover, we conclude that we can do huge system constructing with smaller ones.

We successfully design a 2-digit BCD adder (8 bits), and then complete a code for functional verification. We noticed how the built in test is useful in check the accuracy of the outputs.

We learned a lot about types of adders, also we noticed the difference between the ripple full adder and the carry lookahead adder in our system. The adder with lookahead carry more faster than the ripple one because the carry of each 1-bit full adder does not depend on the previous carries except the first one, so for example the second full adder of the second bit need to wait until the input carry results from the first full adder, but in the adder with lookahead carry doesn't need to wait, all full adders work in parallel.

We became more familiar with VHDL and how to write commands like how to print an error, make a delay, testing the systems and building entities in behavioral and structural logics, also we used Aldec HDL to simulate our project and see the signals of the entity that we worked on it.

Appendix

TAREQ SHANNAK 1181404
PROJECT

-----BASIC GATES WITH DELAY-----

-----AND2-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY AND2 IS
    PORT(A,B: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END AND2;
```

```
ARCHITECTURE structural OF AND2 IS
BEGIN
```

```
    F <= A AND B AFTER 8 NS;
```

```
END;
```

-----AND3-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY AND3 IS
    PORT(A,B,C: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END AND3;
```

```
ARCHITECTURE structural OF AND3 IS
BEGIN
```

```
    F <= A AND B AND C AFTER 8 NS;
```

```
END;
```

-----AND4-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY AND4 IS
    PORT(A,B,C,D: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END AND4;
```

```
ARCHITECTURE structural OF AND4 IS
BEGIN
```

```
    F <= A AND B AND C AND D AFTER 8 NS;
```

```
END;
```

-----AND5-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY AND5 IS
    PORT(A,B,C,D,E: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END AND5;
```

```
ARCHITECTURE structural OF AND5 IS
BEGIN
```

```
    F <= A AND B AND C AND D AND E AFTER 8 NS;
```

```
END;
```

-----OR2-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY OR2 IS
```

```

        PORT(A,B: IN STD_LOGIC:= '0';
              F: OUT STD_LOGIC:= '0');
END OR2;

ARCHITECTURE structural OF OR2 IS
BEGIN
    F <= A OR B AFTER 8 NS;
END;

-----OR3-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY OR3 IS
    PORT(A,B,C: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END OR3;

ARCHITECTURE structural OF OR3 IS
BEGIN
    F <= A OR B OR C AFTER 8 NS;
END;

-----OR4-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY OR4 IS
    PORT(A,B,C,D: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END OR4;

ARCHITECTURE structural OF OR4 IS
BEGIN
    F <= A OR B OR C OR D AFTER 8 NS;
END;

-----OR5-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY OR5 IS
    PORT(A,B,C,D,E: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END OR5;

ARCHITECTURE structural OF OR5 IS
BEGIN
    F <= A OR B OR C OR D OR E AFTER 8 NS;
END;

-----XOR2-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY XOR2 IS
    PORT(A,B: IN STD_LOGIC:= '0';
          F: OUT STD_LOGIC:= '0');
END XOR2;

ARCHITECTURE structural OF XOR2 IS
BEGIN
    F <= A XOR B AFTER 12 NS;
END;

-----NAND2-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY NAND2 IS

```

```

        PORT(A,B: IN STD_LOGIC:= '0';
              F: OUT STD_LOGIC:= '0');
END NAND2;

ARCHITECTURE structural OF NAND2 IS
BEGIN
    F <= A NAND B AFTER 6 NS;
END;

-----
-----One Bit Full Adder-----
-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY FA IS
    PORT(A,B,Carryin: IN STD_LOGIC:= '0';
          Sum,Carryout: OUT STD_LOGIC:= '0');
END FA;

ARCHITECTURE structural OF FA IS
SIGNAL T,R,E,Q,S,H,K: STD_LOGIC := '0';
BEGIN
    -- Full Adder Using NAND gates (Faster)
    G1: ENTITY WORK.NAND2(structural) PORT MAP(A,B,T);
    G2: ENTITY WORK.NAND2(structural) PORT MAP(T,A,R);
    G3: ENTITY WORK.NAND2(structural) PORT MAP(T,B,E);
    G4: ENTITY WORK.NAND2(structural) PORT MAP(R,E,Q);
    G5: ENTITY WORK.NAND2(structural) PORT MAP(Q,Carryin,S);
    G6: ENTITY WORK.NAND2(structural) PORT MAP(S,T,Carryout);
    G7: ENTITY WORK.NAND2(structural) PORT MAP(S,Carryin,H);
    G8: ENTITY WORK.NAND2(structural) PORT MAP(S,Q,K);
    G9: ENTITY WORK.NAND2(structural) PORT MAP(H,K,Sum);
    -- Another way to implement Full Adder (Slower)
    --G1: ENTITY WORK.XOR2(structural) PORT MAP(A,B,X);
    --G2: ENTITY WORK.XOR2(structural) PORT MAP(X,Carryin,Sum);
    --G3: ENTITY WORK.AND2(structural) PORT MAP(X,Carryin,Y);
    --G4: ENTITY WORK.AND2(structural) PORT MAP(A,B,Z);
    --G5: ENTITY WORK.OR2(structural) PORT MAP(Y,Z,Carryout);

END;

-----
-----Carry Lookahead Generator-----
-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY carryGenerator IS
    PORT(P,G: IN STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
          Carryin: IN STD_LOGIC:= '0';
          Carryout: OUT STD_LOGIC_VECTOR(4 DOWNTO 0):= "00000");
END carryGenerator;

ARCHITECTURE carryLookahead OF carryGenerator IS
SIGNAL C,GC: STD_LOGIC_VECTOR(4 DOWNTO 0):= "00000";
SIGNAL PG: STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
SIGNAL PPG: STD_LOGIC_VECTOR(3 DOWNTO 1):= "000";
SIGNAL PPPG: STD_LOGIC_VECTOR(3 DOWNTO 2):= "00";
SIGNAL PPPPG: STD_LOGIC:= '0';
BEGIN
    GC <= G(3 DOWNTO 0) & C(0);
    C(0) <= Carryin;
    Carryout <= C;

```

```

gen1: FOR i IN 0 TO 3 GENERATE
BEGIN
    G: ENTITY WORK.AND2(structural) PORT MAP(P(i),GC(i),PG(i));
END GENERATE;

gen2: FOR i IN 1 TO 3 GENERATE
BEGIN
    G: ENTITY WORK.AND3(structural) PORT MAP(P(i),P(i-1),GC(i-1),PPG(i));
END GENERATE;

gen3: FOR i IN 2 TO 3 GENERATE
BEGIN
    G: ENTITY WORK.AND4(structural) PORT MAP(P(i),P(i-1),P(i-2),GC(i-2),PPPG(i));
END GENERATE;

A1: ENTITY WORK.OR2(structural) PORT MAP(G(0), PG(0), C(1));
A2: ENTITY WORK.OR3(structural) PORT MAP(GC(2), PG(1), PPG(1), C(2));

A3: ENTITY WORK.OR4(structural) PORT MAP(GC(3), PG(2), PPG(2), PPPG(2), C(3));

A4: ENTITY WORK.AND5(structural) PORT MAP(P(3), P(2), P(1), P(0), GC(0), PPPPG);
A44: ENTITY WORK.OR5(structural) PORT MAP(GC(4), PG(3), PPG(3), PPPG(3), PPPPG, C(4));

END;

-----
-----Four Bit Full Adder-----
-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY FA4 IS
    PORT(A,B: IN STD_LOGIC_VECTOR(3 DOWNT0 0):="0000";
        Carryin: IN STD_LOGIC:='0';
        Sum: OUT STD_LOGIC_VECTOR(3 DOWNT0 0):="0000";
        Carryout: OUT STD_LOGIC:='0');
END FA4;

-----
----- 4 Bit Ripple FA -----
ARCHITECTURE ripple OF FA4 IS
    SIGNAL C: STD_LOGIC_VECTOR(4 DOWNT0 0);
    BEGIN
        C(0) <= Carryin;
        Carryout <= C(4);
        gen1: FOR i IN 0 TO 3 GENERATE
        BEGIN
            G: ENTITY WORK.FA(structural) PORT MAP(A(i),B(i),C(i),Sum(i),C(i+1));
        END GENERATE;
    END;

-----
----- 4 Bit Adder With Carry Lookahead -----
ARCHITECTURE lookahead OF FA4 IS
    SIGNAL C: STD_LOGIC_VECTOR(4 DOWNT0 0):="00000";
    SIGNAL P,G: STD_LOGIC_VECTOR(3 DOWNT0 0):="0000";
    BEGIN
        C(0) <= Carryin;
        Carryout <= C(4);
        GG: ENTITY WORK.carryGenerator(carryLookahead) PORT MAP(P,G,Carryin,C);
        gen1: FOR i IN 0 TO 3 GENERATE
        BEGIN
            G1: ENTITY WORK.XOR2(structural) PORT MAP(A(i),B(i),P(i));
            G2: ENTITY WORK.AND2(structural) PORT MAP(A(i),B(i),G(i));
            G3: ENTITY WORK.XOR2(structural) PORT MAP(P(i),C(i),Sum(i));
        END GENERATE;
    END;

```



```

END;

-----BCD Adder-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY BCD IS
    PORT(A,B: IN STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
          Carryin: IN STD_LOGIC:='0';
          Sum: OUT STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
          Carryout: OUT STD_LOGIC:='0');
END BCD;

----- Ripple BCD Adder -----
ARCHITECTURE ripple OF BCD IS
    SIGNAL AA,BB: STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
    SIGNAL X,Y,Z,R: STD_LOGIC:='0';
    BEGIN
        BB(1) <= BB(2);
        Carryout <= BB(2);
        G1: ENTITY WORK.FA4(ripple) PORT MAP(A,B,Carryin,AA,X);
        G2: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(2),Y);
        G3: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(1),Z);
        G4: ENTITY WORK.OR3(structural) PORT MAP(X,Y,Z,BB(2));
        G5: ENTITY WORK.FA4(ripple) PORT MAP(AA,BB,'0',SUM,R);
    END;

----- BCD Adder With Carry Lookahead -----
ARCHITECTURE lookahead OF BCD IS
    SIGNAL AA,BB: STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
    SIGNAL X,Y,Z,R: STD_LOGIC:='0';
    BEGIN
        BB(1) <= BB(2);
        Carryout <= BB(2);
        G1: ENTITY WORK.FA4(lookahead) PORT MAP(A,B,Carryin,AA,X);
        G2: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(2),Y);
        G3: ENTITY WORK.AND2(structural) PORT MAP(AA(3),AA(1),Z);
        G4: ENTITY WORK.OR3(structural) PORT MAP(X,Y,Z,BB(2));
        G5: ENTITY WORK.FA4(lookahead) PORT MAP(AA,BB,'0',SUM,R);
    END;

----- System -----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY System IS
    PORT(A,B: IN STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
          Sum: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
          Carryout: OUT STD_LOGIC:='0');
END System;

----- System With Ripple Adder -----
ARCHITECTURE ripple OF System IS
    SIGNAL X: STD_LOGIC;
    BEGIN
        G1: ENTITY WORK.BCD(ripple) PORT MAP(A(3 DOWNTO 0),B(3 DOWNTO 0),'0',Sum(3 DOWNTO 0),X);
        G2: ENTITY WORK.BCD(ripple) PORT MAP(A(7 DOWNTO 4),B(7 DOWNTO 4),X,Sum(7 DOWNTO 4),Carryout);
    END;

```

```
----- System With Carry Lookahead Adder -----
```

```
ARCHITECTURE lookahead OF System IS
```

```
SIGNAL X: STD_LOGIC;
```

```
BEGIN
```

```
    G1: ENTITY WORK.BCD(lookahead) PORT MAP(A(3 DOWNTO 0),B(3 DOWNTO 0),'0',Sum(3 DOWNTO 0),X);
```

```
    G2: ENTITY WORK.BCD(lookahead) PORT MAP(A(7 DOWNTO 4),B(7 DOWNTO 4),X,Sum(7 DOWNTO 4),Carryout);
```

```
END;
```

```
----- Test Generator -----
```

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.ALL;
```

```
USE ieee.std_logic_ARITH.ALL;
```

```
USE ieee.std_logic_UNSIGNED.ALL;
```

```
ENTITY TestGenerator IS
```

```
    PORT(CLK: IN STD_LOGIC:= '0';
```

```
         A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";
```

```
         Correct: OUT STD_LOGIC_VECTOR(8 DOWNTO 0):= "000000000");
```

```
END TestGenerator;
```

```
ARCHITECTURE generator OF TestGenerator IS
```

```
SIGNAL AA,BB: STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";
```

```
BEGIN
```

```
    A<=AA;
```

```
    B<=BB;
```

```
    -- The Process Below shows how to implement the system using behavioural logic
```

```
    PROCESS (AA,BB)
```

```
        VARIABLE X: STD_LOGIC_VECTOR(8 DOWNTO 0):= "000000000";
```

```
        BEGIN
```

```
            X(4 DOWNTO 0) := ('0'&AA(3 DOWNTO 0)) + ('0'&BB(3 DOWNTO 0));
```

```
            if(X(4 DOWNTO 0) > "01001") then
```

```
                X(3 DOWNTO 0) := X(3 DOWNTO 0)+"0110";
```

```
                X(4) := '1';
```

```
            end if;
```

```
            X(8 DOWNTO 4) := ('0000'&X(4)) + ('0'&AA(7 DOWNTO 4)) + ('0'&BB(7 DOWNTO 4));
```

```
            if(X(8 DOWNTO 4) > "01001") then
```

```
                X(7 DOWNTO 4) := X(7 DOWNTO 4)+"0110";
```

```
                X(8) := '1';
```

```
            end if;
```

```
            CORRECT<=X;
```

```
        END PROCESS;
```

```
        -- The Process below changes the values of AA and BB(= A and B) when the clock has a rising edge
```

```
        -- (A = 0, B = 0) => (99, 99)
```

```
        PROCESS
```

```
            BEGIN
```

```
                FOR i IN 0 TO 9 LOOP
```

```
                    FOR j IN 0 TO 9 LOOP
```

```
                        FOR K IN 0 TO 9 LOOP
```

```
                            FOR L IN 0 TO 9 LOOP
```

```
                                AA(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(i,4);
```

```
                                AA(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,4);
```

```
                                BB(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(K,4);
```

```
                                BB(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(L,4);
```

```
                                WAIT UNTIL rising_edge(CLK);
```

```
                            END LOOP;
```

```
                        END LOOP;
```

```
                    END LOOP;
```

```
                END LOOP;
```

```
                WAIT;
```

```
            END PROCESS;
```

```
END;
```

```

----- Result Analyser -----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_ARITH.ALL;

ENTITY ResultAnalyser IS
    PORT(CLK: IN STD_LOGIC:= '0';
          Correct,myResult: IN STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000");
END ResultAnalyser;

ARCHITECTURE analyser OF ResultAnalyser IS
BEGIN
    -- The Process below make sure that the resulting output from our system equals to the correct one
    -- otherwise, print an error when the outputs are not equal to each other
    PROCESS
    BEGIN
        assert (myResult = Correct)
        report "The results that were obtained don't agree with the theoretical results"
        severity ERROR;
        WAIT UNTIL rising_edge(CLK);
    END PROCESS;
END;

----- Built In Self Test -----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_ARITH.ALL;

ENTITY BIST IS
END BIST;

----- Test For The Ripple System -----
ARCHITECTURE ripple OF BIST IS
    SIGNAL CLK: STD_LOGIC:= '0';
    SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
    SIGNAL Correct,myResult: STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000";
    BEGIN
        -- 85*2 = 170 ns is the minimum delay we should have to have a correct output
        CLK <= NOT CLK AFTER 85 NS;
        G1: ENTITY WORK.TestGenerator(generator) PORT MAP(CLK, A, B, Correct);
        G2: ENTITY WORK.System(ripple) PORT MAP(A, B, myResult(7 DOWNTO 0), myResult(8));
        G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(CLK, Correct, myResult);
    END;

----- Test For The Carry Lookahead System -----
ARCHITECTURE lookahead OF BIST IS
    SIGNAL CLK: STD_LOGIC:= '0';
    SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
    SIGNAL Correct,myResult: STD_LOGIC_VECTOR(8 DOWNTO 0):="000000000";
    BEGIN
        -- 68*2 = 136 ns is the minimum delay we should have to have a correct output
        CLK <= NOT CLK AFTER 68 NS;
        G1: ENTITY WORK.TestGenerator(generator) PORT MAP(CLK, A, B, Correct);
        G2: ENTITY WORK.System(lookahead) PORT MAP(A, B, myResult(7 DOWNTO 0), myResult(8));
        G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(CLK, Correct, myResult);
    END;

```