****

**Faculty of Engineering & Technology**

**Computer Systems Engineering Department**

**ADVANCED DIGITAL**

**“Ripple Adder, Look-ahead Adder, Arithmetic Unit”**

**Prepared By:**

**Farah Abu Lebdeh - 1171456**

**Instructor : Dr . Abdallatif Abuissa**

**Date: 12/12/2019**

* **Table of contents:**

**Introduction…………………………………………………………. (3)**

**Background……………………………...………………………....... (3)**

* **VHDL…………………………………………………...….(3)**
* **Adder…………………………………………………...…..(3)**
* **Full Adder……………………………………………...…..(3)**
* **Ripple Adder…………………………………...…………..(3)**
* **Carry Look Ahead Adder………………………..……….(4)**
* **Multiplexer…………………………………………………(4)**
* **Test Generator……………………………………………..(4)**

**Design Philosophy ……………………………………...……………(5)**

**Arithmetic Unit ………………………………………………………(6)**

**Conclusion……………………………………………..………….…. (7)**

**Simulation…………………………………………………………….(8)**

**References……………………………………………….…………....(8)**

**Appendix ……………………………………………….………….... (9)**

* **Introduction :**

The aim of this project is to design an Arithmetic Unit, and write a code for functional verification. Using design Ripple Adder, Look-ahead Adder . Arithmetic Unit in VHDL language and simulation tools .The Arithmetic Unit and Ripple Adder, Look-ahead Adder is to be built structurally from a library of gates basic gate with different delays .A CPU consists of three main sections: memory for variables (registers), control circuitry (microcode), and the ALU. The ALU (Arithmetic Logic Unit) is the part of a CPU that actually does calculations and condition testing in this project we want only Arithmetic unit without logic .For example, if you wish to add two binary numbers, it is the AU that is responsible for producing the result. If your program needs to execute some code if two values are equal it is the test Generator that performs the comparison between the values.

* **Background​ :**

**VHDL** : is an acronym for Very high speed integrated circuit (VHSIC) Hardware Description Language which is a programming language that describes a logic circuit by function, data flow behavior, and/or structure. This hardware description is used to configure a programmable logic device (PLD), such as a field programmable gate array (FPGA), with a custom logic design.

**The Circuit :**In my project use the basic gate : xor , and , or , invertor

**Adder:** A binary adder is an electronic circuit used in digital electronics, such as a computer, to adder two binary numbers. It is built using :

* **Full adder :** is a logic circuit that adds two input operand bits plus a Carry in bit and outputs a Carry out bit and a sum bit.. The Sum out  (Sout) of a full adder is the XOR of input operand bits A, B and the Carry in (Cin) bit.  Truth table and schematic of a 1 bit Full adder is shown below:

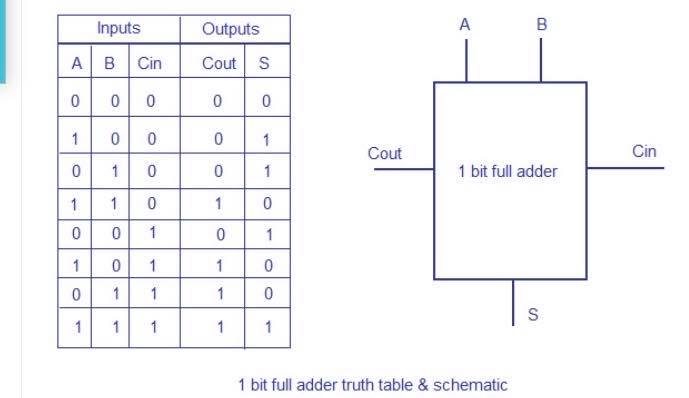


Fig.1-(Full Adder)

* **Ripple Adder :** Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next  stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs. Propagation delays inside the logic circuitry is the reason behind this. Propagation delay is time elapsed between the application of an input and occurrence of the corresponding output.

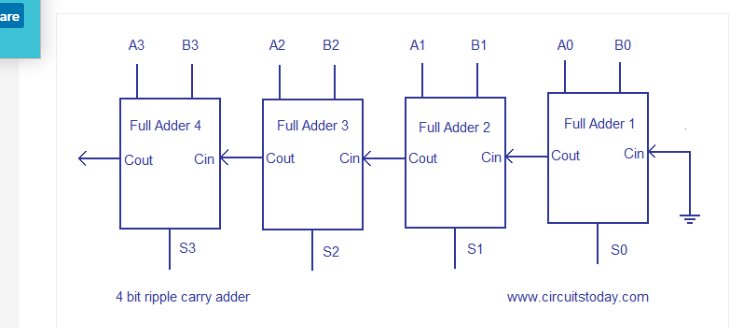


Fig.2-(Ripple Adder)

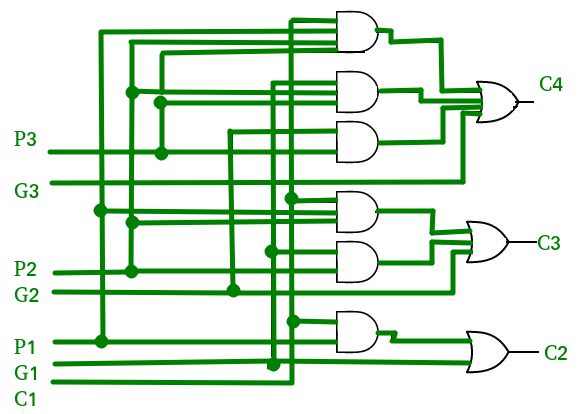
* **Carry Look-ahead Adder :**A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail . This method makes use of logic gates so as to look at the lower order bits of the augend and addend to see whether a higher order carry is to be generated or not.

Fig3-(CLA)

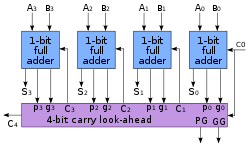


Fig4-(CLA)

If we define two new binary variables

Pi = Ai XOR Bi ,,,,,,,,Gi = Ai AND Bi

The output sum and carry can respectively be expressed as

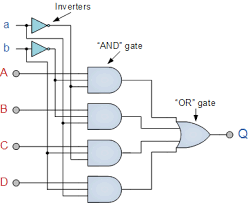
Si = Pi XOR Ci ,,,,,,,,, Ci+1 = Gi + PiCi

Gi is called a carry generate, and it produces a carry of 1 when both Ai and Bi are 1, regardless of the input carry Ci. Pi is called a carry propagate, because it determines whether a carry into stage i will propagate into stage i + 1 (i.e., whether an assertion of Ci will propagate to an assertion of Ci+1). => C0 = input carry ,C1 = G0 + P0C0

Continuing this way, we can find each carry of next stage adders in terms of Gi, Pi, and C0. Hence, each next carry will depend on input carry.

**What is a Multiplexer(Mux)?**

The multiplexer or MUX is a digital switch, also called as data selector. It is a combinational circuit with more than one input line, one output line and more than one select line. It allows the binary information from several input lines or sources and depending on the set of select lines , particular input line , is routed onto a single output line

* **Mux 4\*1:** A  4-to-1 multiplexer consists four data input lines as D0 to D3, two select lines as S0 and S1 and a single output line Y. The select lines S1 and S2 select one of the four input lines to connect the output line. The particular input combination on select lines selects one of input (D0 through D3) to the output . The figure below (Fig.5)shows the block diagram of a 4-to-1 multiplexer using basic logic gates. The below figure shows the logic circuit of 4:1 MUX which is implemented by four 3-inputs AND gates, two 1-input NOT gates, and one 4-inputs OR gate . In this circuit, each data input line is connected as input to an AND gate and two select lines are connected as other two inputs to it. The AND gate output is connected to with inputs of OR gate so as to produce the output Y.

**Fig.5(Mux 4\*1)**

* **Test Generator using table Arithmetic unit: (main arithmetic operation)**

The inputs and the output of the Arithmetic Unit are fed through/to flip-flops (registers). The Arithmetic unit works as shown in the following table: (Fig.6)

1. [**Add**](https://en.wikipedia.org/wiki/Binary_number#Addition): A and B are summed and the sum

appears at Y and carry-out.

1. **Add with carry**: A, B and carry-in are summed and the sum appears at Y and carry-out.
2. **Subtract with borrow**: B is subtracted from A (or vice versa) with borrow (carry-in) and the difference appears at Y and carry-out (borrow out).
3. [**Subtract**](https://en.wikipedia.org/wiki/Binary_number#Subtraction): B is subtracted from A (or vice versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator. This operation may also be used to compare the magnitudes of A and B; in such cases the Y output may be ignored by the processor, which is only interested in the status bits (particularly zero and negative) that result from the operation.
4. **Increment**: A (or B) is increased by one and the resulting value appears at Y.
5. **Decrement**: A (or B) is decreased by one and the resulting value appears at Y.
6. **Transfer(8,5**): all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative, or to load the operand into a processor register.

**Design philosophy :**

In my project first I implement all the gates using the basic gates given with delay , then build structurally full adder using entity of basic gates(xor , and , or) . after that I use Full Adder to implement Ripple Adder (4-bit Full Adder) . In Carry Look Ahead Adder I implement different than Ripple because in look ahead we need to calculate more than sum and carry we need also (G & P) where (G = A . B) And (P = A ⊕ B) After built adders , I went to implement my Arithmetic Adder .

**Design** : VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a test bench.

**Arithmetic Unit:**

A basic AU has input  (A , B , selection,1,0,cin, ) and a result output (D, cout).

**First of all** I implemented all the logic gates with the required delay.a Full Adder(2xor ,2and ,or) the input for 1 bit full adder(A,B,cin) ,A xor B out n1,n1 xor cin out sum ,n1 and cin out n3,A and B out n4,n3 or n4 out cout ,The output (sum,Cout). and mux 4\*1 (4and ,invertor ,or )

Y=S1(not) .S2(not).A.B +S1(not) .S2.A.B+ S1.S2(not).A.B+ S1 .S2.A.B

**Second:** I built the 1-bit Full Adder then Built 4-bit Full Adder structurally , full Adder operation (A, B, Cin) input and (Sum, Cout) output.  The 4-bit Ripple Carry Adder VHDL Code can be Easily Constructed by Port Mapping 4 Full Adder . **Third** : to implement 4 bit Ripple Carry Adder VHDL Code, using code full adder .Now declare full adder entity as component in 4-bit Ripple Carry Adder VHDL Code and do Port Map operation.

**Fourthly** : to implement Carry Look Ahead Adder, first implement Full Adder and then Carry logic using Propagation and generation Block. The inputs (A, B, Cin) and Outputs (S, P, G) where P is Propagate Output and G is Generate output. carry look ahead adder can be implemented by first constructing full adder block and port map them to four times and also implementing carry generation .

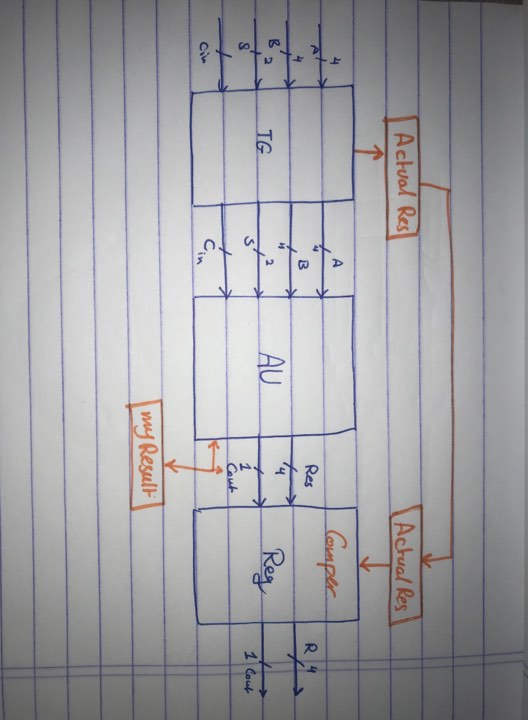
* To implement Arithmetic unit using Ripple Carry Adder do Port Map operation for ripple Adder and Mux . For Verification call Port map Arithmetic unit use ripple adder and arithmetic system must built previously.
* To implement Arithmetic unit using Carry Look Ahead Adder do Port Map operation for Carry Look Ahead Adder and Mux . For Verification call Port map Arithmetic unit use Carry Look Ahead Adder and arithmetic system must built previously.

When Arithmetic Unit with ripple adder I enter the input (clk, A,B,S, Cin) and the output ( D, Cout),With call 4 port map of mux input (S0,S1,B,Bnot)and out and (Y) and Y enter to ripple adder Like input.

When Arithmetic Unit with CLHA I enter the input (clk, A,B, Cin) and the output for one block (P,G,D,Cout). With call 4 port map of mux input (S0,S1,B,Bnot)and out and (Y) .

**Signals** :An AU has a variety of input and output [nets](https://en.wikipedia.org/wiki/Net_(electronics)), which are the [electrical conductors](https://en.wikipedia.org/wiki/Electrical_conductors) used to convey [digital signals](https://en.wikipedia.org/wiki/Digital_signal_(electronics)) between the AU and Test Generator and register . When an AU is operating, external circuits apply signals to the AU inputs and, in response, the AU produces and conveys signals to external circuitry via its outputs.

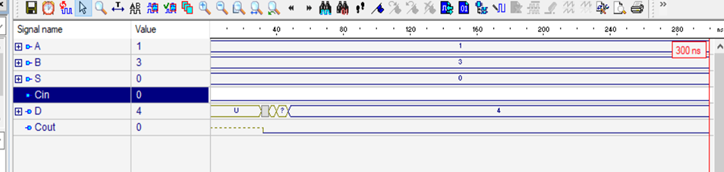
When enter to binary number A,B and Cin With value for two selection test Generator calculate actual result depending the table in (Fig.6) ,pass value of input to AU system calculate my result and pass to register ,then compere the actual result with my result Finally print in file if the actual result equal my result . (Fig .7)



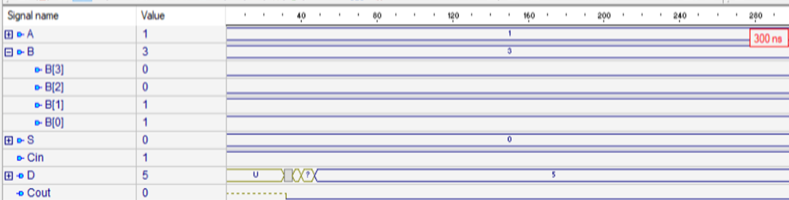
**Conclusion:**

In my project I conclude that the clock rate is very important and any gate can make a error of result, such as in full adder there is two different implementation, one take 16 ns delay and the other take 12 but more hardware, we conclude that more speed we need more hardware, less hardware we must wait for delay more than first. The same thing in ripple adder and look Ahead. On other hand the calculation is by hand in ripple adder is 350 (each adder take 35 \* 5 adder \* 2 ) and in practical ~ 300 note I don’t generate all possibilities but a range. In look ahead adder by hand is 115, but in run time max is 88 in my attempt it close.

* **Results ( Simulation Results) ​:**
* **Simulation results for AU:**

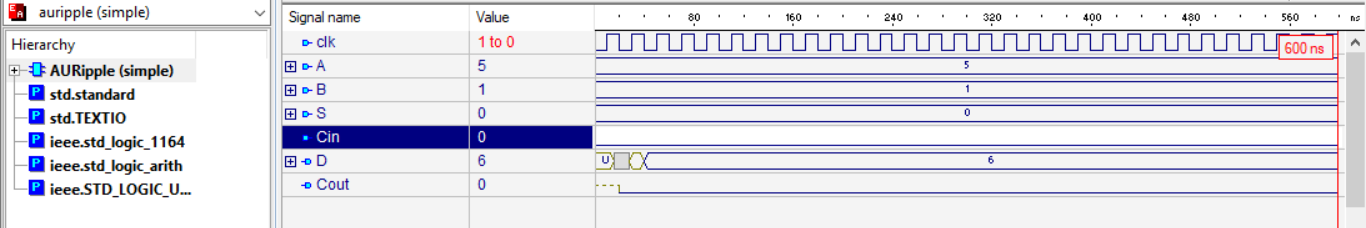
****

**Fig.result(1)**



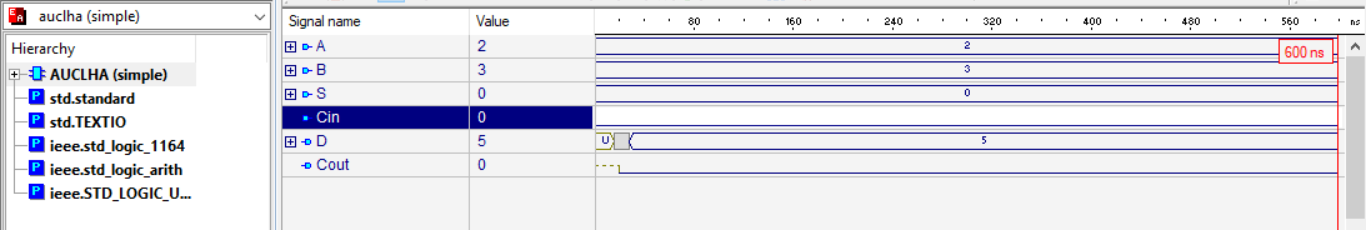
**Fig.result(2)**

* **Simulation results for AU from ripple adder:**



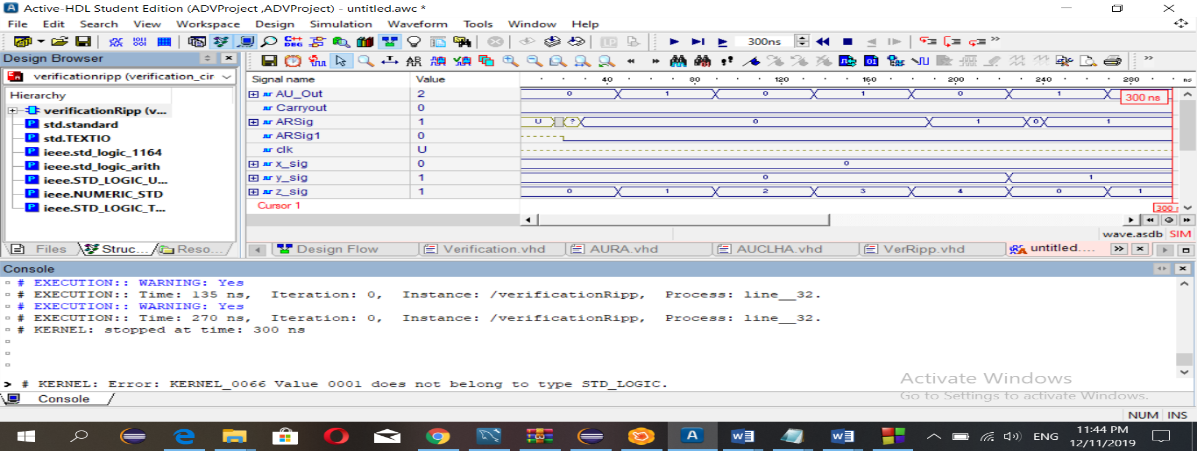
**Fig.result(3)**

* **Simulation results for AU from Carry Look Ahead Adder:**

****

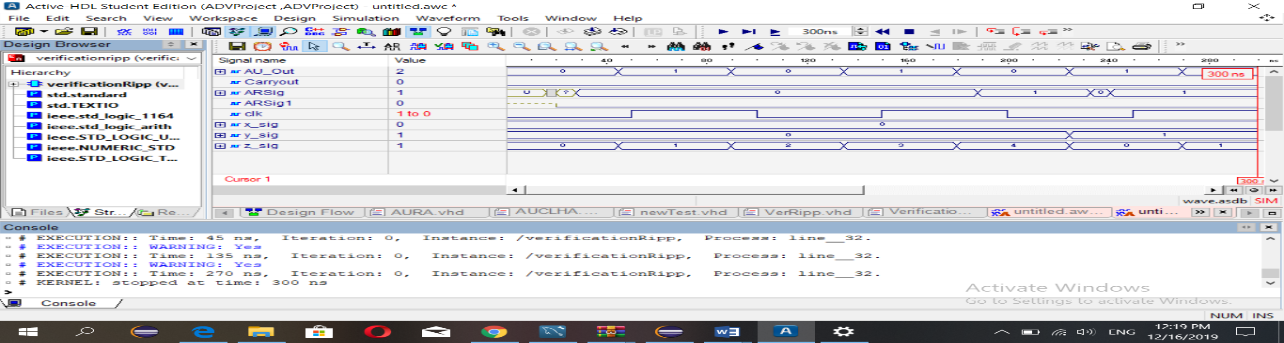
**Fig.result(4)**

* **Simulation result for Verification Ripple Adder:**



**Fig.result(5)**

* **Simulation result for AU from Carry Look Head Adder:**



**Fig.result(6)**

References:

* <https://www.geeksforgeeks.org/carry-look-ahead-adder/> access in 1/12/2019
* <https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl/v_cl_addr.html> access in 27/11/2019
* Book : Digital System Design With VHDL-Mark Zwolinski –second Edition
* **Appendix:**
* **First : The Basic gate I use in my project:**

---------------------"Arithmetic Unit"-------------------------------

-----------------XOR GATE for full adder-----------------------------

-----------------XOR GATE number (1)---------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

Entity fxorgate is

Port ( a,b : in std\_logic;

y : out std\_logic);

end Entity fxorgate;

Architecture beh\_xor of fxorgate is

begin

y <= a xor b after 8 ns;

End Architecture beh\_xor;

-----------------XOR GATE number (2)-----------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

Entity fxorgate2 is

Port ( y,cin : in std\_logic;

sum : out std\_logic);

end Entity fxorgate2;

Architecture beh\_xor2 of fxorgate2 is

begin

sum <= y xor cin after 8 ns;

End Architecture beh\_xor2;

----------------------------------------AND(2-bit) GATE ------------------------------

----------------------------------------AND GATE number (1)------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity fandgate is

port(y,cin : in std\_logic ;

x:out std\_logic);

End entity fandgate;

Architecture behfandgate of fandgate is

begin

x <= y and cin after 6 ns;

end architecture behfandgate;

----------------------------------------AND GATE number (2)------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity fandgate2 is

port(a,b : in std\_logic ;

x:out std\_logic);

End entity fandgate2;

Architecture behfandgate2 of fandgate2 is

begin

x <= a and b after 6 ns;

end architecture behfandgate2;

----------------------------------------AND GATE number (4)------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity fandgate4 is

port(a,b,c,d : in std\_logic ;

x:out std\_logic);

End entity fandgate4;

Architecture behfandgate4 of fandgate4 is

begin

x <= a and b and c and d after 6 ns;

end architecture behfandgate4;

----------------------------------------AND GATE number (5)------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity fandgate5 is

port(a,b,c,d,e : in std\_logic ;

x:out std\_logic);

End entity fandgate5;

Architecture behfandgate5 of fandgate5 is

begin

x <= a and b and c and d and e after 6 ns;

end architecture behfandgate5;

----------------------3bit------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity and3 is

port(A, B, C : in std\_logic ;

y:out std\_logic);

End entity and3;

Architecture behand3 of and3 is

begin

y <= A and B and C after 6 ns;

end architecture behand3;

--------------------------4bit----------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity and4 is

port(A, B, C ,D: in std\_logic ;

y:out std\_logic);

End entity and4;

Architecture behand4 of and4 is

begin

y <= A and B and C and D after 6 ns;

end architecture behand4;

------------------------------------OR GATE (2bit)-------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity fORgate is

port(x,z: in std\_logic ;

cout:out std\_logic);

end entity fORgate;

architecture behfORgate of fORgate is

begin

cout <= x or z after 6 ns;

end architecture behfORgate;

------------------------------------OR GATE (4bit)-------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity fOR4 is

port(x,y,n,z: in std\_logic ;

cout:out std\_logic);

end entity fOR4;

architecture behfOR4 of fOR4 is

begin

cout <= x or z or n or y after 6 ns;

end architecture behfOR4;

------------------------------------OR GATE (3bit)-------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity fORgate3 is

port(x,z,y: in std\_logic ;

cout:out std\_logic);

end entity fORgate3;

architecture behfORgate3 of fORgate3 is

begin

cout <= x or z or y after 6 ns;

end architecture behfORgate3;

------------------------------------OR GATE (5bit)-------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity fORgate5 is

port(x,z,y,a,b: in std\_logic ;

cout:out std\_logic);

end entity fORgate5;

architecture behfORgate5 of fORgate5 is

begin

cout <= x or z or y or a or b after 6 ns;

end architecture behfORgate5;

-------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity inv1 is port (Sel0: in std\_logic ; selbar0 :out std\_logic);

end entity;

architecture invv1 of inv1 is

begin

selbar0<= not Sel0 after 3 ns;

end architecture;

----------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity inv2 is port (Sel1: in std\_logic ; selbar1 :out std\_logic);

end entity;

architecture invv2 of inv2 is

begin

selbar1<= not Sel1 after 3 ns;

end architecture;

----------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux is

port(A, selbar0, selbar1 : in std\_logic ;

t1:out std\_logic);

End entity andmux;

Architecture behandmux of andmux is

begin

t1 <= A and selbar0 and selbar1 after 6 ns;

end architecture behandmux;

-------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux2 is

port(B, Sel0, selbar1 : in std\_logic ;

t2:out std\_logic);

End entity andmux2;

Architecture behandmux2 of andmux2 is

begin

t2 <= B and sel0 and selbar1 after 6 ns;

end architecture behandmux2;

--------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux3 is

port(C, selbar0, Sel1 : in std\_logic ;

t3:out std\_logic);

End entity andmux3;

Architecture behandmux3 of andmux3 is

begin

t3 <= C and selbar0 and Sel1 after 6 ns;

end architecture behandmux3;

------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux4 is

port(D, Sel0, Sel1 : in std\_logic ;

t4:out std\_logic);

End entity andmux4;

Architecture behandmux4 of andmux4 is

begin

t4 <= D and Sel0 and Sel1 after 6 ns;

end architecture behandmux4;

------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux5 is

port(a,b,c,d : in std\_logic ;

z:out std\_logic);

End entity andmux5;

Architecture behandmux5 of andmux5 is

begin

z <= a and b and c and d after 6 ns;

end architecture behandmux5;

------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

Entity andmux6 is

port(a,b,c,d,y : in std\_logic ;

z:out std\_logic);

End entity andmux6;

Architecture behandmux6 of andmux6 is

begin

z <= a and b and c and d and y after 6 ns;

end architecture behandmux6;

-------------------------------------------------------------------------------------------

library ieee ;

use ieee.std\_logic\_1164.all ;

entity or1 is port (t1, t2, t3, t4: in std\_logic ; Y :out std\_logic);

end entity;

architecture orr of or1 is

begin

Y<= t1 or t2 or t3 or t4 after 6 ns;

end architecture;

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY dflipFlop IS

PORT(d, clk, reset : IN STD\_LOGIC;

q: OUT STD\_LOGIC);

END ENTITY dflipFlop;

ARCHITECTURE Dff\_circuit OF dflipFlop IS

BEGIN

PROCESS ( clk )

BEGIN

IF( reset = '1' ) THEN

q <= '0';

ELSIF( RISING\_EDGE(clk) ) THEN

q <= d;

END IF;

END PROCESS;

END ARCHITECTURE Dff\_circuit;

-----------------------------------------------------------------------

----------------------------Register-----------------------------------

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY n\_bit\_register IS

GENERIC(n : positive := 11);

PORT(clk, reset : IN STD\_LOGIC;

d : IN std\_logic\_vector(n-1 DOWNTO 0);

q: OUT STD\_LOGIC\_vector(n-1 DOWNTO 0));

END ENTITY n\_bit\_register;

ARCHITECTURE register\_circuit of n\_bit\_register is

BEGIN

Dffloop:FOR i IN 0 TO n-1 GENERATE

DFF: ENTITY WORK.dflipFlop(Dff\_circuit)

PORT MAP(d(i), clk, reset,q(i));

END GENERATE Dffloop;

END ARCHITECTURE register\_circuit;

* Code Ripple Adder:

------------------------------------------------------------------------------------------

---------------\*Built 1-bit full adder from the basic gates\*------------------------------

library IEEE;

USE ieee.std\_logic\_1164.ALL;

ENTITY fFulladder IS --Entity Full Adder

PORT ( A,B, CIN: IN STD\_LOGIC; --input Full Adder

D, COUT: OUT STD\_LOGIC); --the output

END ENTITY fFulladder;

ARCHITECTURE dataflow OF fFulladder IS --arch

signal y,x,z: STD\_LOGIC;

BEGIN

g1: ENTITY work.fxorgate(beh\_xor) PORT MAP (A,B,y); -- xor gate 2 bit input

g2: ENTITY work.fxorgate2(beh\_xor2) PORT MAP (y,CIN,D); -- xor gate 2 bit input

g3: ENTITY work.fandgate(behfandgate) PORT MAP (y,CIN,x); -- and gate 2 bit input

g4: ENTITY work.fandgate2(behfandgate2) PORT MAP (A,B,z); -- and gate 2 bit input

g5: ENTITY work.fORgate(behfORgate) PORT MAP (x,z,COUT); -- or gate 2 bit input

END ARCHITECTURE dataflow;

----------------------\*build 4-bit adder"Ripple\_Adder"\*-----------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ripple\_Adder is --Ripple adder is 4 bit full adder

Port ( X,Y : in STD\_LOGIC\_VECTOR (3 downto 0); --entity 3 input(X,Y,Cin) and 2-output (D,Cout)

Cin : in STD\_LOGIC;

D : out STD\_LOGIC\_VECTOR (3 downto 0);

Cout : out STD\_LOGIC);

end Ripple\_Adder;

ARCHITECTURE structural OF Ripple\_Adder IS

SIGNAL C1,C2,C3: STD\_LOGIC;

SIGNAL A : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

BEGIN

A(0) <= X(0);

A(1) <= X(1);

A(2) <= X(2);

A(3) <= X(3);

g0: entity work.fFulladder(dataflow) --call 4 full adder to implement Ripple Adder

PORT MAP ( X(0), Y(0), Cin, D(0), C1);

g1: entity work.fFulladder(dataflow)

PORT MAP ( X(1), Y(1), C1, D(1), C2);

g2: entity work.fFulladder(dataflow)

PORT MAP ( X(2), Y(2), C2, D(2), C3);

g3: entity work.fFulladder(dataflow)

PORT MAP ( X(3), Y(3), C3, D(3), Cout);

END ARCHITECTURE structural;

* Code Carry Look Ahead Adder:

-- ----------------------------------------------------

-- Carry Look Ahead Adder -----------------------------

----------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY clhaadder IS --entity

PORT(clk:in std\_logic;

x, y : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

cin : IN STD\_LOGIC;

s : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

cout : OUT STD\_LOGIC);

END ENTITY clhaadder;

ARCHITECTURE Arc OF clhaadder IS --Arch

SIGNAL p : STD\_LOGIC\_VECTOR(3 DOWNTO 0); -- carryProp

SIGNAL g : STD\_LOGIC\_VECTOR(3 DOWNTO 0); -- carryGene

SIGNAL c : STD\_LOGIC\_VECTOR(4 DOWNTO 0); -- CLAGenerator

SIGNAL p0c0 : STD\_logic; --Signal for Equation CLHA

SIGNAL p1g0 : STD\_logic;

SIGNAL p1p0c0 : STD\_logic;

SIGNAL p2g1 : STD\_logic;

SIGNAL p2p1g0 : STD\_logic;

SIGNAL p2p1p0c0 : STD\_logic;

SIGNAL p3g2 : STD\_logic;

SIGNAL p3p2g1 : STD\_logic;

SIGNAL p3p2p1g0 : STD\_logic;

SIGNAL p3p2p1p0c0 : STD\_logic;

BEGIN

carryPropagate: FOR i IN 3 DOWNTO 0 GENERATE

carryPropLoop: ENTITY WORK.fxorgate(beh\_xor)

PORT MAP(x(i), y(i), p(i));

END GENERATE carryPropagate;

--behfandgate2 of fandgate2

carryGenerate: FOR i IN 3 DOWNTO 0 GENERATE

carryGeneLoop: ENTITY WORK.fandgate2(behfandgate2)

PORT MAP(x(i), y(i), g(i));

END GENERATE carryGenerate;

c(0) <= cin;

--behfandgate2 of fandgate2

p0c0\_label: ENTITY WORK.fandgate2(behfandgate2)

PORT MAP(p(0), c(0), p0c0);

c1: ENTITY WORK.fORgate(behfORgate)

PORT MAP(p0c0, g(0), c(1));

--behfandgate2 of fandgate2

-- behfORgate of fORgate

--behfORgate3 of fORgate3

p1g0\_label: ENTITY WORK.fandgate2(behfandgate2)

PORT MAP(p(1), g(0), p1g0);

--behand3 of and3

--behand3 of and3

p1p0c0\_label: ENTITY WORK.and3(behand3)

PORT MAP(p(1), p(0), c(0), p1p0c0);

c2: ENTITY WORK.fORgate3(behfORgate3)

PORT MAP(g(1), p1g0, p1p0c0, c(2));

p2g1\_label: ENTITY WORK.fandgate2(behfandgate2)

PORT MAP(p(2), g(1), p2g1);

p2p1g0\_label: ENTITY WORK.and3(behand3)

PORT MAP(p(2), p(1), g(0), p2p1g0);

p2p1p0c0\_label: ENTITY WORK.and4(behand4)

PORT MAP(p(2), p(1), p(0), c(0), p2p1p0c0);

--behfOR4 of fOR4

c3: ENTITY WORK.fOR4(behfOR4)

PORT MAP(g(2), p2g1, p2p1g0, p2p1p0c0, c(3));

--behfandgate2 of fandgate2

p3g2\_label: ENTITY WORK.fandgate2(behfandgate2)

PORT MAP(p(3), g(2), p3g2);

-- behfORgate of fORgate

--behfORgate3 of fORgate3

--behand3 of and3

p3p2g1\_label: ENTITY WORK.and3(behand3)

PORT MAP(p(3), p(2), g(1), p3p2g1);

--behand4 of and4

p3p2p1g0\_label: ENTITY WORK.and4(behand4)

PORT MAP(p(3), p(2), p(1), g(0), p3p2p1g0);

--behandmux6 of andmux6

p3p2p1p0c0\_label: ENTITY WORK.andmux6(behandmux6)

PORT MAP(p(3), p(2), p(1), p(0), c(0), p3p2p1p0c0);

-- behfORgate5 of fORgate5

c4: ENTITY WORK.fORgate5(behfORgate5)

PORT MAP(g(3), p3g2, p3p2g1, p3p2p1g0, p3p2p1p0c0, c(4));

cout <= c(4); -- output carry

sumf : FOR i IN 0 TO 3 GENERATE

mygate : ENTITY WORK.fxorgate(beh\_xor)

PORT MAP (p(i),c(i),s(i));

END GENERATE sumf;

END ARCHITECTURE Arc;

* **Arithmetic System**:

-------------------------AU----------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity AUsystem is -- entity

port(clk:in std\_logic;

A,B: in std\_logic\_vector(3 downto 0);

S:in std\_logic\_vector(1 downto 0);

Cin:in std\_logic;

D:out std\_logic\_vector(3 downto 0);

Cout:out std\_logic);

end entity AUsystem;

architecture simple of AUsystem is

signal n1,n2,n3,n4,n5,n6,n7,n8: STD\_logic;

begin --- Arch using mux and full adder

Cout <= n8;

G1: entity work.MUX4to1(structural) port map (S(1),S(0),B(0),n1);

G2: entity work.MUX4to1(structural)port map (S(1),S(0),B(1),n2);

G3: entity work.MUX4to1(structural) port map (S(1),S(0),B(2),n3);

G4: entity work.MUX4to1(structural) port map (S(1),S(0),B(3),n4);

G5: entity work.fFulladder(dataflow) port map (A(0),n1,Cin,D(0),n5);

G6: entity work.fFulladder(dataflow) port map (A(1),n2,n5,D(1),n6);

G7: entity work.fFulladder(dataflow) port map (A(2),n3,n6,D(2),n7);

G8: entity work.fFulladder(dataflow) port map (A(3),n4,n7,D(3),n8);

end architecture simple;

* **Arithmetic Unit using Ripple Adder :**

-------------AU using ripple adder--------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity AURipple is

port(clk:in std\_logic;

A,B: in std\_logic\_vector(3 downto 0);

S:in std\_logic\_vector(1 downto 0);

Cin:in std\_logic;

D:out std\_logic\_vector(3 downto 0);

Cout:out std\_logic);

end entity AURipple;

architecture simpleRipple of AURipple is

signal n1,n2,n3,n4: STD\_logic;

begin

G1: entity work.MUX4to1(structural) port map (S(1),S(0),B(0),n1);

G2: entity work.MUX4to1(structural)port map (S(1),S(0),B(1),n2);

G3: entity work.MUX4to1(structural) port map (S(1),S(0),B(2),n3);

G4: entity work.MUX4to1(structural) port map (S(1),S(0),B(3),n4);

G5: entity work.Ripple\_Adder(structural) port map (A,B,Cin,D,cout);

end architecture simpleRipple;

* **Arithmetic unit using CLHA:**

-------------------------AUCLHA----------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity AUCLHA is

--clk:in std\_logic;

port( clk:in std\_logic;

A,B: in std\_logic\_vector(3 downto 0);

S:in std\_logic\_vector(1 downto 0);

Cin:in std\_logic;

D:out std\_logic\_vector(3 downto 0);

Cout:out std\_logic);

end entity AUCLHA;

architecture simple of AUCLHA is

signal n1,n2,n3,n4: STD\_logic;

begin

-- Arc OF clhaadder

G1: entity work.MUX4to1(structural) port map (S(1),S(0),B(0),n1);

G2: entity work.MUX4to1(structural)port map (S(1),S(0),B(1),n2);

G3: entity work.MUX4to1(structural) port map (S(1),S(0),B(2),n3);

G4: entity work.MUX4to1(structural) port map (S(1),S(0),B(3),n4);

G5:entity work.clhaadder(Arc)port map (clk,A,B,Cin,D,Cout);

end architecture simple;

* **Test Generator:**

----------------------------------------------------------------------------

-------------------- TestGenerator Rigester----------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use ieee.NUMERIC\_STD.all;

entity AU is

Port ( clk : in std\_logic;

A, B : in STD\_LOGIC\_VECTOR(3 downto 0); -- 2 inputs 4-bit

AU\_Sel : in STD\_LOGIC\_VECTOR(2 downto 0); -- 1 input 3-bit for selecting function

AU\_Out : out STD\_LOGIC\_VECTOR(3 downto 0); -- 1 output 5-bit

Carryout : out std\_logic -- Carryout flag

);

end entity AU;

architecture Behavioral of AU is

signal AU\_Result : std\_logic\_vector (3 downto 0);

signal tmp: std\_logic\_vector (4 downto 0);

begin

process(A,B,AU\_Sel,clk)

begin

case(AU\_Sel) is

when "000" => -- Addition

AU\_Result <= A + B ;

when "001" => -- Addition with carry

AU\_Result <= A + B + 1 ;

when "010" => -- Addition with Borrow

AU\_Result <= A - B ;

when "011" => -- Sub

AU\_Result <= A - B + 1 ;

when "100" => -- Transfer A

AU\_Result <= A + 0;

when "101" => -- incr

AU\_Result <= A + 1 ;

when "110" => -- Dec

AU\_Result <= A - 1 ;

when "111" => -- Transfer A

AU\_Result <= A + 0 ;

when others => AU\_Result <= A + B ;

end case;

end process;

AU\_Out <= AU\_Result; -- AU out

tmp <= ('0' & A) + ('0' & B);

Carryout <= tmp(4); -- Carryout flag

end Behavioral;

* Verification Arithmetic Unit with Ripple Adder:

---------------------------------------------------

------------------Verification RA--------------

---------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.NUMERIC\_STD.ALL;

USE STD.TEXTIO.ALL;

use IEEE.std\_logic\_textio.all;

ENTITY verificationRipp IS ---empty entity ,I dependent signal instance input and output

END ENTITY verificationRipp;

ARCHITECTURE verification\_circuitRipp OF verificationRipp IS

SIGNAL AU\_Out: STD\_LOGIC\_VECTOR(3 downto 0);

SIGNAL Carryout: std\_logic ;

SIGNAL ARSig: STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL ARSig1: STD\_LOGIC;

signal clk :std\_logic;

SIGNAL x\_sig : STD\_LOGIC\_VECTOR(3 DOWNTO 0);-- input A

SIGNAL y\_sig : STD\_LOGIC\_VECTOR(3 DOWNTO 0);--input B

SIGNAL z\_sig : STD\_LOGIC\_VECTOR(2 DOWNTO 0);--selection and cin

BEGIN

Circuit: ENTITY WORK.AURipple(simpleRipple)port map (clk, x\_sig, y\_sig,z\_sig(1 downto 0),z\_sig(2),ARSig(3 downto 0),ARSig1); --AU with ripple

Act:entity work.AU(Behavioral) port map (clk, x\_sig, y\_sig,z\_sig(2 downto 0),AU\_Out(3 downto 0),Carryout); --main circuit AU

PROCESS

--FILE outputFile : TEXT; -- File / not use

-- VARIABLE file\_status : FILE\_OPEN\_STATUS;

-- VARIABLE buff : LINE;

VARIABLE x\_var : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

VARIABLE y\_var : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

VARIABLE z\_var : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

BEGIN

--FILE\_OPEN(file\_status, outputFile, "dataFile.txt", WRITE\_MODE);

FOR i IN 0 TO 15 LOOP --A

FOR j IN 0 TO 15 LOOP --B

FOR k IN 0 TO 4 LOOP --sel

x\_var := CONV\_STD\_LOGIC\_VECTOR(i, 4);

y\_var := CONV\_STD\_LOGIC\_VECTOR(j, 4);

z\_var := CONV\_STD\_LOGIC\_VECTOR(k, 3);

x\_sig <= x\_var;

y\_sig <= y\_var;

z\_sig <= z\_var;

if(AU\_Out = ARSig and Carryout = ARSig1)then

ASSERT FALSE

REPORT ("Yes")

SEVERITY WARNING;

end if;

-- WRITE(buff, x\_sig);

-- WRITE(buff, " + ");

-- WRITE(buff, y\_sig);

-- WRITE(buff, " = ");

-- WRITE(buff, " | ");

-- writeline(outputFile, buff);

IF (i = 15 and j = 15 and k=7 ) THEN

ASSERT FALSE

REPORT ("CONGRATS! ALL MATCHED")

SEVERITY WARNING;

END IF;

WAIT FOR 45ns ;

END LOOP;

END LOOP;

END LOOP;

END PROCESS;

END ARCHITECTURE verification\_circuitRipp;

* Verification Arithmetic Unit with Carry Look Ahead Adder:

-------------------------------------------------------------------

------------------Verification CLHA-------------------------

-------------------------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.NUMERIC\_STD.ALL;

USE STD.TEXTIO.ALL;

use IEEE.std\_logic\_textio.all;

ENTITY myverification IS

END ENTITY myverification;

ARCHITECTURE myverification\_circuit OF myverification IS

SIGNAL AU\_Out: STD\_LOGIC\_VECTOR(3 downto 0);

SIGNAL Carryout: std\_logic ;

SIGNAL ARSig: STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL ARSig1: STD\_LOGIC;

signal clk :std\_logic;

SIGNAL x\_sig : STD\_LOGIC\_VECTOR(3 DOWNTO 0);--A

SIGNAL y\_sig : STD\_LOGIC\_VECTOR(3 DOWNTO 0);--B

SIGNAL z\_sig : STD\_LOGIC\_VECTOR(2 DOWNTO 0);--sel

BEGIN

Act:entity work.AU(Behavioral) port map (clk, x\_sig, y\_sig,z\_sig(2 downto 0),AU\_Out(3 downto 0),Carryout);

Circuit:entity work.AUCLHA(simple) port map (clk, x\_sig, y\_sig,z\_sig(1 downto 0),z\_sig(2),AU\_Out(3 downto 0),Carryout);

PROCESS

FILE outputFile : TEXT;

VARIABLE file\_status : FILE\_OPEN\_STATUS;

VARIABLE buff : LINE;

VARIABLE x\_var : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

VARIABLE y\_var : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

VARIABLE z\_var : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

BEGIN

--FILE\_OPEN(file\_status, outputFile, "dataFile.txt", WRITE\_MODE);

FOR i IN 0 TO 15 LOOP --A

FOR j IN 0 TO 15 LOOP --B

FOR k IN 0 TO 4 LOOP --sel

x\_var := CONV\_STD\_LOGIC\_VECTOR(i, 4);

y\_var := CONV\_STD\_LOGIC\_VECTOR(j, 4);

z\_var := CONV\_STD\_LOGIC\_VECTOR(k, 3);

x\_sig <= x\_var;

y\_sig <= y\_var;

z\_sig <= z\_var;

if(AU\_Out = ARSig and Carryout = ARSig1)then

ASSERT FALSE

REPORT ("Yes")

SEVERITY WARNING;

end if;

-- WRITE(buff, x\_sig);

--WRITE(buff, " + ");

--WRITE(buff, y\_sig);

--WRITE(buff, " = ");

--WRITE(buff, " | ");

--writeline(outputFile, buff);

IF (i = 15 and j = 15 and k=7 ) THEN

ASSERT FALSE

REPORT ("CONGRATS! ALL MATCHED")

SEVERITY WARNING;

END IF;

WAIT FOR 45ns ;

END LOOP;

END LOOP;

END LOOP;

END PROCESS;

END ARCHITECTURE myverification\_circuit;