FACULTY OF ENGINEERING AND TECHNOLOGY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ADVANCED DIGITAL DESIGN ENCS533

COURSE PROJECT

Name: Yousef Ghanem Number: 1172333

Instructor: Dr. Abdallatif Abuissa

Date: 10-7-2020

# 1. Abstract:

In this project I’ll try to build a M\*N bit Multiplier using two kinds of adders , first one is a carry ripple adder and the second is a carry look ahead adder. This operation was done by some rules and used several types of gates in it all has a specific delay time , and at last the multiplier was tested by an analyzer to test if the output is valid or not.

Contents

[1. Abstract: 2](#_Toc45286712)

[2. **Brief Introduction:** 4](#_Toc45286713)

[2.1 **Full Adder:** 4](#_Toc45286714)

[2.2 **Ripple carry adder:** 5](#_Toc45286715)

[2.3 **Carry Look-Ahead Adder:** 6](#_Toc45286716)

[3. **Brief theoretical overview**: 7](#_Toc45286717)

[4. **Design philosophy:** 9](#_Toc45286718)

[5. **Simulation Results**: 10](#_Toc45286719)

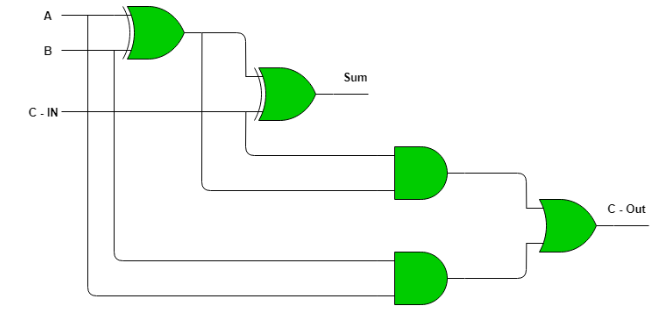
[6. **Conclusion**: 11](#_Toc45286720)

[References: 11](#_Toc45286721)

[**VHDL Code:** 12](#_Toc45286722)

# 2. **Brief Introduction:**

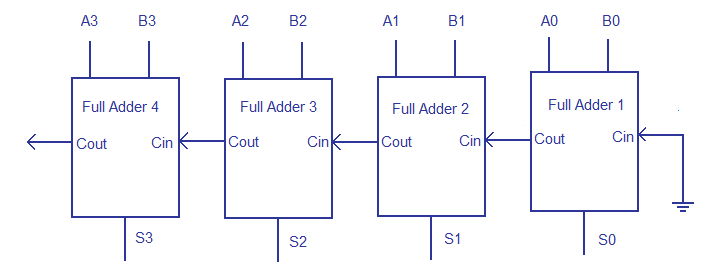
## 2.1 **Full Adder:**

 Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.  
A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to the another.

*Full Adder logic circuit*

## 2.2 **Ripple carry 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. Consider a NOT gate, When the input is “0” the output will be “1” and vice versa. The time taken for the NOT gate’s output to become “0” after the application of logic “1” to the NOT gate’s input is the propagation delay here. Similarly, the carry propagation delay is the time elapsed between the application of the carry in signal and the occurrence of the carry out (Cout) signal. Circuit diagram of a 4-bit ripple carry adder is shown below.

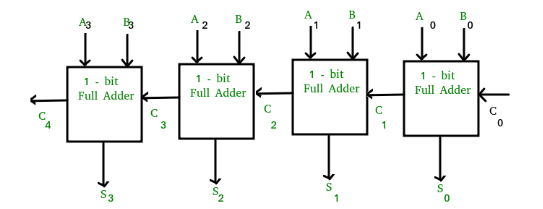


*Ripple carry adder*

Sum out S0 and carry out Cout of the Full Adder 1 is valid only after the propagation delay of Full Adder 1. In the same way, Sum out S3 of the Full Adder 4 is valid only after the joint propagation delays of Full Adder 1 to Full Adder 4. In simple words, the final result of the ripple carry adder is valid only after the joint propagation delays of all full adder circuits inside it.

## 2.3 **Carry Look-Ahead Adder:**

In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block. So, it is not possible to generate the sum and carry of any block until the input carry is known. The ith block waits for the i – 1th block to produce its carry. So, there will be a considerable time delay which is carry propagation delay.



*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.

**Advantages and Disadvantages of Carry Look-Ahead Adder :**  
**Advantages –**

* The propagation delay is reduced.
* It provides the fastest addition logic.

**Disadvantages –**

* The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
* The circuit is costlier as it involves a greater number of hardware.

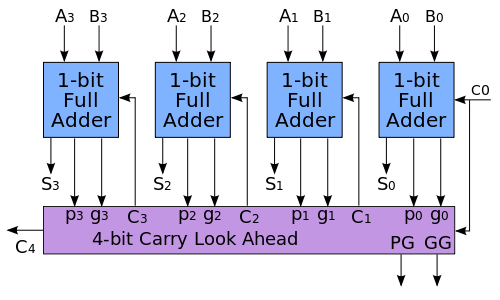
# 3. **Brief theoretical overview**:

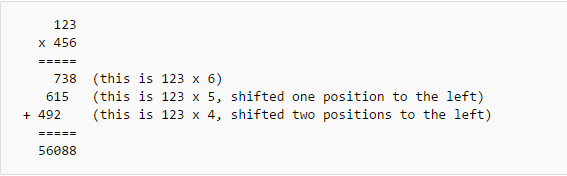
Binary Multiplier:

A binary multiplier is an [electronic circuit](https://en.wikipedia.org/wiki/Electronic_circuit) used in [digital electronics](https://en.wikipedia.org/wiki/Digital_electronics), such as a [computer](https://en.wikipedia.org/wiki/Computer), to [multiply](https://en.wikipedia.org/wiki/Multiplication) two [binary numbers](https://en.wikipedia.org/wiki/Binary_number). It is built using [binary adders](https://en.wikipedia.org/wiki/Binary_adder). A variety of [computer arithmetic](https://en.wikipedia.org/wiki/Category:Computer_arithmetic) techniques can be used to implement a digital multiplier. Most techniques involve computing a set of *partial products*, and then summing the partial products together. This process is similar to the method taught to primary schoolchildren for conducting long multiplication on base-10 integers, but has been modified here for application to a base-2 ([binary](https://en.wikipedia.org/wiki/Binary_numeral_system)) [numeral system](https://en.wikipedia.org/wiki/Numeral_system).

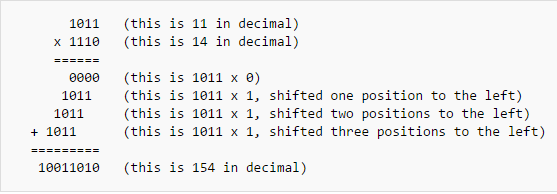
Multiplication basics:

The method taught in school for multiplying decimal numbers is based on calculating partial products, shifting them to the left and then adding them together. The most difficult part is to obtain the partial products, as that involves multiplying a long number by one digit (from 0 to 9):





A binary computer does exactly the same, but with binary numbers. In binary encoding each long number is multiplied by one digit (either 0 or 1), and that is much easier than in decimal, as the product by 0 or 1 is just 0 or the same number. Therefore, the multiplication of two binary numbers comes down to calculating partial products (which are 0 or the first number), shifting them left, and then adding them together (a binary addition, of course):



This is much simpler than in the decimal system, as there is no table of multiplication to remember: just shifts and add.

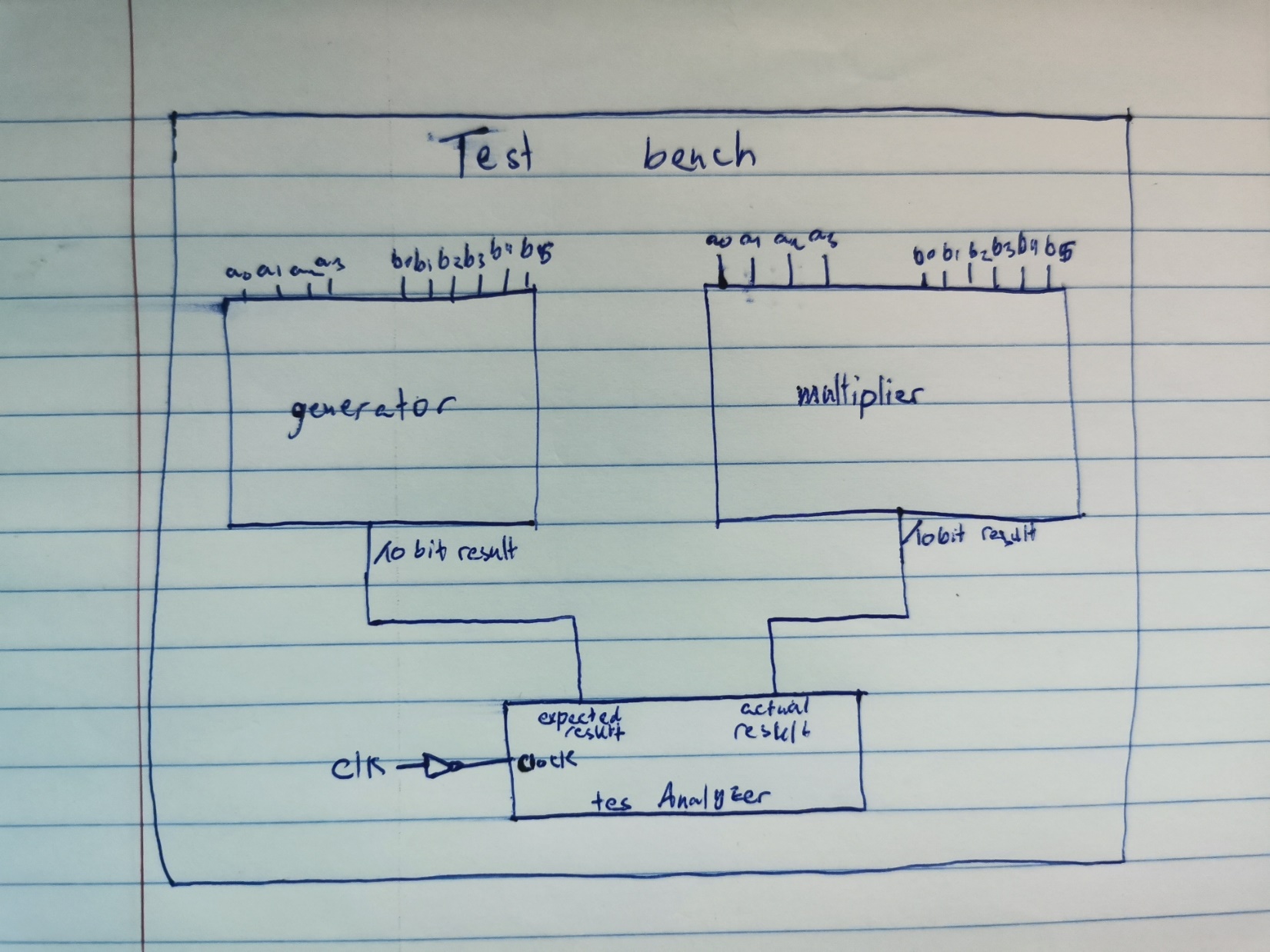
This method is mathematically correct and has the advantage that a small CPU may perform the multiplication by using the shift and add features of its arithmetic logic unit rather than a specialized circuit. The method is slow, however, as it involves many intermediate additions. These additions take a lot of time. Faster multipliers may be engineered in order to do fewer additions; a modern processor can multiply two 64-bit numbers with 6 additions (rather than 64) and can do several steps in parallel.

# 4. **Design philosophy:**

I implemented the code in several VHDL files (Gates Library , Adders Library , Multiplier , Verfication , Test).

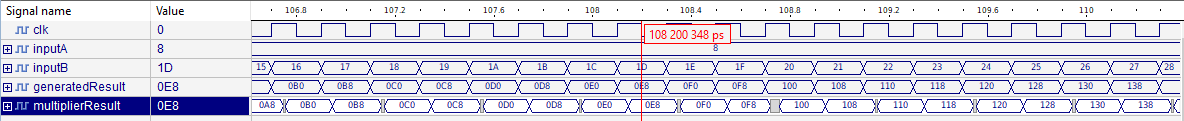
First, I implemented the logic gates with the required delay , I implemented a 1-bit full adder using those gates and then I implemented a carry ripple 4-bit adder and a carry look ahead 4-bit adder , I also implemented a j\*k multiplier , a test generator and a result analyzer and those all work in one system in a test bench to determine the accuracy of the multiplier.

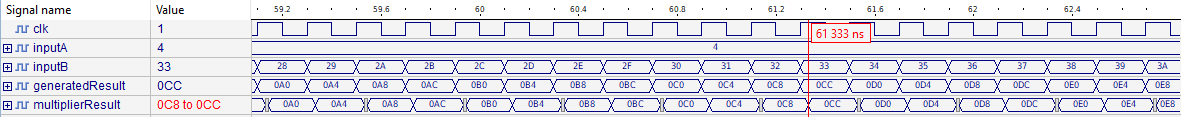
In the test we used j = 4 and k = 6 for the multiplier and generated certain inputs using the clocks and then compared the result from the multiplier to the expected result from the generator , the comparison was done by assert and report commands in VHDL which the report displays a warning message when the both results are not equal.



# 5. **Simulation Results**:

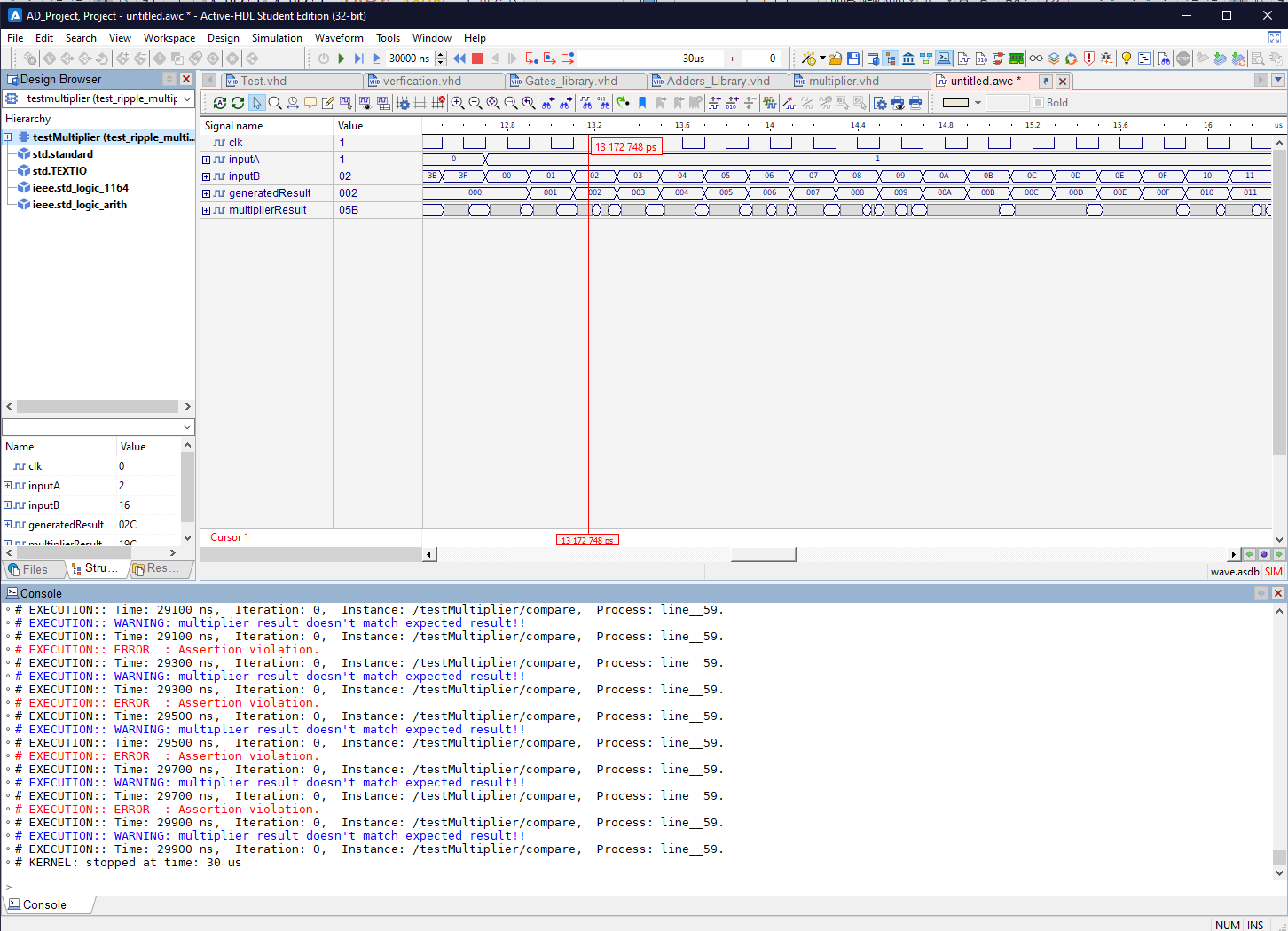
* 1. Correct Result of a 6\*4 multiplier:





* 1. Result of a buggy multiplier:

In the multiplier implementation I replaced the and gates in the inputs of the adders with or gates and the result was as follows:



# 6. **Conclusion**:

The carry look ahead adder is faster than the ripple adder in most cases, and the same for the full adder, because A design with alternating carry polarities and optimized AND-OR-Invert gates can be about twice as fast, that will optimize the delay, since that the NAND Gates has less Transistors than AND gates, as we learned in Integrated Circuits course. The advantage of look ahead adder is that it takes less speed than ripple, but its implementation is complex than ripple. So, ripple adder is simple to implementation, but it takes high delay.

# References:

1. <https://www.geeksforgeeks.org/full-adder-in-digital-logic/> (Full adder)
2. <https://www.circuitstoday.com/ripple-carry-adder#:~:text=A%20ripple%20carry%20adder%20is%20a%20logic%20circuit%20in%20which,rippled%20into%20the%20next%20stage.> (ripple carry adder)
3. <https://www.geeksforgeeks.org/carry-look-ahead-adder/> (carry look ahead adder)
4. <https://en.wikipedia.org/wiki/Binary_multiplier> (Binary multiplier)

# **VHDL Code:**

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

-- Gates Library

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

-- NOT INVERTER GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** NOTGATE **IS**

**PORT(**x **:** **IN** STD\_LOGIC**;**

y **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** NOTGATE**;**

**ARCHITECTURE** not\_arc **OF** NOTGATE **IS**

**BEGIN**

y **<=** **NOT** x **AFTER** 4 NS**;**

**END** **ARCHITECTURE** not\_arc**;**

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

--2 inputs AND Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ANDGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ANDGATE2**;**

**ARCHITECTURE** and\_arc2 **OF** ANDGATE2 **IS**

**BEGIN**

z **<=** x **AND** y **AFTER** 9 NS**;**

**END** **ARCHITECTURE** and\_arc2**;**

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

-- 2 inputs OR Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ORGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ORGATE2**;**

**ARCHITECTURE** or\_arc2 **OF** ORGATE2 **IS**

**BEGIN**

z **<=** x **OR** y **AFTER** 9 NS**;**

**END** **ARCHITECTURE** or\_arc2**;**

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

-- 2 inputs NAND GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** NANDGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** NANDGATE2**;**

**ARCHITECTURE** nand\_arc **OF** NANDGATE2 **IS**

**BEGIN**

z **<=** x **NAND** y **AFTER** 6 NS**;**

**END** **ARCHITECTURE** nand\_arc**;**

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

-- 2 inputs NOR GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** NORGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** NORGATE2**;**

**ARCHITECTURE** nor\_arc **OF** NORGATE2 **IS**

**BEGIN**

z **<=** x **NOR** y **AFTER** 6 NS**;**

**END** **ARCHITECTURE** nor\_arc**;**

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

-- 2 inputs XOR GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** XORGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** XORGATE2**;**

**ARCHITECTURE** xor\_arc2 **OF** XORGATE2 **IS**

**BEGIN**

z **<=** x **XOR** y **AFTER** 12 NS**;**

**END** **ARCHITECTURE** xor\_arc2**;**

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

-- 2 inputs XNOR GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** XNORGATE2 **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** XNORGATE2**;**

**ARCHITECTURE** xnor\_arc **OF** XNORGATE2 **IS**

**BEGIN**

z **<=** x **XNOR** y **AFTER** 12 NS**;**

**END** **ARCHITECTURE** xnor\_arc**;**

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

--3 inputs AND Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ANDGATE3 **IS**

**PORT(**x**,** y**,** q **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ANDGATE3**;**

**ARCHITECTURE** and\_arc3 **OF** ANDGATE3 **IS**

**BEGIN**

z **<=** **((**x **AND** y**)** **AND** q**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** and\_arc3**;**

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

--4 inputs AND Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ANDGATE4 **IS**

**PORT(**x**,** y**,** q**,** w **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ANDGATE4**;**

**ARCHITECTURE** and\_arc4 **OF** ANDGATE4 **IS**

**BEGIN**

z **<=** **(((**x **AND** y**)** **AND** q**)** **AND** w**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** and\_arc4**;**

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

--5 inputs AND Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ANDGATE5 **IS**

**PORT(**x**,** y**,** q**,** w**,** e **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ANDGATE5**;**

**ARCHITECTURE** and\_arc5 **OF** ANDGATE5 **IS**

**BEGIN**

z **<=** **((((**x **AND** y**)** **AND** q**)** **AND** w**)** **AND** e**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** and\_arc5**;**

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

--6 inputs AND Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ANDGATE6 **IS**

**PORT(**x**,** y**,** q**,** w**,** e**,** r **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ANDGATE6**;**

**ARCHITECTURE** and\_arc6 **OF** ANDGATE6 **IS**

**BEGIN**

z **<=** **(((((**x **AND** y**)** **AND** q**)** **AND** w**)** **AND** e**)** **AND** r**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** and\_arc6**;**

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

-- 3 inputs OR Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ORGATE3 **IS**

**PORT(**x**,** y**,** q **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ORGATE3**;**

**ARCHITECTURE** or\_arc3 **OF** ORGATE3 **IS**

**BEGIN**

z **<=** **((**x **OR** y**)** **OR** q**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** or\_arc3**;**

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

-- 4 inputs OR Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ORGATE4 **IS**

**PORT(**x**,** y**,** q**,** w **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ORGATE4**;**

**ARCHITECTURE** or\_arc4 **OF** ORGATE4 **IS**

**BEGIN**

z **<=** **(((**x **OR** y**)** **OR** q**)** **OR** w**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** or\_arc4**;**

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

-- 5 inputs OR Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ORGATE5 **IS**

**PORT(**x**,** y**,** q**,** w**,** e **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ORGATE5**;**

**ARCHITECTURE** or\_arc5 **OF** ORGATE5 **IS**

**BEGIN**

z **<=** **((((**x **OR** y**)** **OR** q**)** **OR** w**)** **OR** e**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** or\_arc5**;**

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

-- 6 inputs OR Gate:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** ORGATE6 **IS**

**PORT(**x**,** y**,** q**,** w**,** e**,** r **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** ORGATE6**;**

**ARCHITECTURE** or\_arc6 **OF** ORGATE6 **IS**

**BEGIN**

z **<=** **(((((**x **OR** y**)** **OR** q**)** **OR** w**)** **OR** e**)** **OR** r**)** **AFTER** 9 NS**;**

**END** **ARCHITECTURE** or\_arc6**;**

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

-- 3 inputs XOR GATE:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** XORGATE3 **IS**

**PORT(**x**,** y**,** q **:** **IN** STD\_LOGIC**;**

z **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** XORGATE3**;**

**ARCHITECTURE** xor\_arc3 **OF** XORGATE3 **IS**

**BEGIN**

z **<=** **((**x **XOR** y**)** **XOR** q**)** **AFTER** 12 NS**;**

**END** **ARCHITECTURE** xor\_arc3**;**

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

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

-- ADDERS LIBRARY

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

-- FULL ADDER:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** fullAdder **IS**

**PORT(**x**,** y **:** **IN** STD\_LOGIC**;**

cin **:** **IN** STD\_LOGIC**;**

cout**,** s **:** **OUT** STD\_LOGIC**);**

**END** **ENTITY** fullAdder**;**

**ARCHITECTURE** fullAdder\_structural **OF** fullAdder **IS**

**SIGNAL** c1**,** c2**,** c3 **:** STD\_LOGIC**;**

**BEGIN**

gate1 **:** **ENTITY** work**.**XORGATE3 **PORT** **MAP(**x**,** y**,** cin**,** s**);**

gate2 **:** **ENTITY** work**.**ANDGATE2 **PORT** **MAP(**x**,** y**,** c1**);**

gate3 **:** **ENTITY** work**.**ANDGATE2 **PORT** **MAP(**x**,** cin**,** c2**);**

gate4 **:** **ENTITY** work**.**ANDGATE2 **PORT** **MAP(**cin**,** y**,** c3**);**

gate5 **:** **ENTITY** work**.**ORGATE3 **PORT** **MAP(**c1**,** c2**,** c3**,** cout**);**

**END** **ARCHITECTURE** fullAdder\_structural**;**

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

-- CARRY RIPPLE ADDER:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** nbit\_rippleAdder **IS**

**PORT(**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** nbit\_rippleAdder**;**

**ARCHITECTURE** nbit\_rippleAdder\_structural **OF** nbit\_rippleAdder **IS**

**SIGNAL** carry **:** STD\_LOGIC\_VECTOR**(**4 **DOWNTO** 0**);**

**BEGIN**

carry**(**0**)** **<=** cin**;**

cout **<=** carry**(**4**);**

gnr**:** **FOR** i **IN** 0 **TO** 3 **GENERATE**

f **:** **ENTITY** work**.**fullAdder**(**fullAdder\_structural**)**

**PORT** **MAP** **(**x**(**i**),**y**(**i**),**carry**(**i**),**carry**(**i**+**1**),**s**(**i**));**

**END** **GENERATE** gnr**;**

**END** **ARCHITECTURE** nbit\_rippleAdder\_structural**;**

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

-- CARRY LOOK AHEAD ADDER:

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**ENTITY** CLA\_adder **IS**

**PORT(**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** CLA\_adder**;**

**ARCHITECTURE** CLA\_adder\_structural **OF** CLA\_adder **IS**

-- carry propagate

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

-- carry generate

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

-- CLA Generator

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

**SIGNAL** p0c0 **:** STD\_logic**;**

**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**

carry\_Propagate**:** **FOR** i **IN** 3 **DOWNTO** 0 **GENERATE**

carry\_Prop\_Loop**:** **ENTITY** WORK**.**XORGATE2**(**xor\_arc2**)**

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

**END** **GENERATE** carry\_Propagate**;**

carry\_Generate**:** **FOR** i **IN** 3 **DOWNTO** 0 **GENERATE**

carry\_Gene\_Loop**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

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

**END** **GENERATE** carry\_Generate**;**

c**(**0**)** **<=** cin**;**

p0c0\_out**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

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

c1**:** **ENTITY** WORK**.**ORGATE2**(**or\_arc2**)**

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

p1g0\_out**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

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

p1p0c0\_out**:** **ENTITY** WORK**.**ANDGATE3**(**and\_arc3**)**

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

c2**:** **ENTITY** WORK**.**ORGATE3**(**or\_arc3**)**

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

p2g1\_out**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

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

p2p1g0\_out**:** **ENTITY** WORK**.**ANDGATE3**(**and\_arc3**)**

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

p2p1p0c0\_out**:** **ENTITY** WORK**.**ANDGATE4**(**and\_arc4**)**

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

c3**:** **ENTITY** WORK**.**ORGATE4**(**or\_arc4**)**

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

p3g2\_out**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

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

p3p2g1\_out**:** **ENTITY** WORK**.**ANDGATE3**(**and\_arc3**)**

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

p3p2p1g0\_out**:** **ENTITY** WORK**.**ANDGATE4**(**and\_arc4**)**

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

p3p2p1p0c0\_out**:** **ENTITY** WORK**.**ANDGATE5**(**and\_arc5**)**

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

c4**:** **ENTITY** WORK**.**ORGATE5**(**or\_arc5 **)**

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

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

sum **:** **FOR** i **IN** 0 **TO** 3 **GENERATE**

gate **:** **ENTITY** WORK**.**XORGATE2**(**xor\_arc2**)**

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

**END** **GENERATE** sum**;**

**END** **ARCHITECTURE** CLA\_adder\_structural**;**

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

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

-- Multiplier

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

**LIBRARY** ieee**;**

**USE** ieee**.**std\_logic\_1164**.ALL;**

**ENTITY** multiplier **IS**

-- j \* k => (j-1) \* k-bit adders

**GENERIC(**j**:** positive **:=** 6 **;** k**:** positive **:=** 4**);**

**PORT(**x **:** **IN** std\_logic\_vector**((**j **-** 1**)** **DOWNTO** 0**);**

y **:** **IN** std\_logic\_vector**((**k **-** 1**)** **DOWNTO** 0**);**

res **:** **OUT** std\_logic\_vector**((**j **+** k **-** 1**)** **DOWNTO** 0**));**

**END** **ENTITY** multiplier**;**

--Stage one multiplier:

-- Multiplier built by ripple adders:

**ARCHITECTURE** multiplier\_ripple\_struct **OF** multiplier **IS**

**TYPE** d2 **IS** **ARRAY** **(**j**-**1 **DOWNTO** 0 **)** **OF** std\_logic\_vector **(**k**-**1 **DOWNTO** 0**);**

**SIGNAL** ands**,** sum**,** shiftedsum**:** d2**;**

**SIGNAL** cout **:** std\_logic\_vector **(**j**-**1 **DOWNTO** 0**);**

**BEGIN**

x\_input**:**

**FOR** i **IN** 0 **TO** j **-** 1 **GENERATE**

y\_input**:**

**FOR** m **IN** 0 **TO** k **-** 1 **GENERATE**

andLoop**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

**PORT** **MAP(**x**(**i**),** y**(**m**),** ands**(**i**)(**m**));**

**END** **GENERATE** y\_input**;**

**END** **GENERATE** x\_input**;**

sum**(**0**)** **<=** ands**(**0**);**

cout**(**0**)** **<=** '0'**;**

muliplication**:**

**FOR** i **IN** 0 **TO** j**-**2 **GENERATE**

res**(**i**)** **<=** sum**(**i**)(**0**);**

shiftedsum**(**i**)** **<=** cout**(**i**)** **&** sum**(**i**)(**k**-**1 **DOWNTO** 1**);**

adder**:** **ENTITY** work**.**nbit\_rippleadder**(**nbit\_rippleadder\_structural**)**

**PORT** **MAP** **(**shiftedsum**(**i**),** ands**(**i**+**1**),** '0'**,** sum**(**i**+**1**),** cout**(**i**+**1**));**

**END** **GENERATE** muliplication**;**

res**(**j**+**k**-**1 **DOWNTO** j**-**1**)** **<=** cout**(**j**-**1**)** **&** sum**(**j**-**1**);**

**END** **ARCHITECTURE** multiplier\_ripple\_struct**;**

--Stage Two multiplier:

-- Multiplier built by look ahead adders:

**ARCHITECTURE** multiplier\_cla\_struct **OF** multiplier **IS**

**TYPE** d2 **IS** **ARRAY** **(**j**-**1 **DOWNTO** 0 **)** **OF** std\_logic\_vector **(**k**-**1 **DOWNTO** 0**);**

**SIGNAL** ands**,** sum**,** shiftedsum**:** d2**;**

**SIGNAL** cout **:** std\_logic\_vector **(**j**-**1 **DOWNTO** 0**);**

**BEGIN**

x\_input**:**

**FOR** i **IN** 0 **TO** j **-** 1 **GENERATE**

y\_input**:**

**FOR** m **IN** 0 **TO** k **-** 1 **GENERATE**

andLoop**:** **ENTITY** WORK**.**ANDGATE2**(**and\_arc2**)**

**PORT** **MAP(**x**(**i**),** y**(**m**),** ands**(**i**)(**m**));**

**END** **GENERATE** y\_input**;**

**END** **GENERATE** x\_input**;**

sum**(**0**)** **<=** ands**(**0**);**

cout**(**0**)** **<=** '0'**;**

muliplication**:**

**FOR** i **IN** 0 **TO** j**-**2 **GENERATE**

res**(**i**)** **<=** sum**(**i**)(**0**);**

shiftedsum**(**i**)** **<=** cout**(**i**)** **&** sum**(**i**)(**k**-**1 **DOWNTO** 1**);**

adder**:** **ENTITY** work**.**cla\_adder**(**cla\_adder\_structural**)**

**PORT** **MAP** **(**shiftedsum**(**i**),** ands**(**i**+**1**),** '0'**,** sum**(**i**+**1**),** cout**(**i**+**1**));**

**END** **GENERATE** muliplication**;**

res**(**j**+**k**-**1 **DOWNTO** j**-**1**)** **<=** cout**(**j**-**1**)** **&** sum**(**j**-**1**);**

**END** **ARCHITECTURE** multiplier\_cla\_struct**;**

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

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

-- Test benches

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

**LIBRARY** ieee**;**

**USE** ieee**.**std\_logic\_1164**.ALL;**

**ENTITY** testMultiplier **IS**

**END** testMultiplier**;**

-- test bench for ripple based multiplier:

**ARCHITECTURE** test\_ripple\_multiplier **OF** testMultiplier **IS**

**SIGNAL** clk **:** std\_logic **:=**'0'**;**

**SIGNAL** inputA **:** std\_logic\_vector **(**3 **DOWNTO** 0**);**

**SIGNAL** inputB **:** std\_logic\_vector **(**5 **DOWNTO** 0**);**

**SIGNAL** generatedResult **:** std\_logic\_vector **(**9 **DOWNTO** 0**);**

**SIGNAL** multiplierResult **:** std\_logic\_vector **(**9 **DOWNTO** 0**);**

**BEGIN**

clk **<=** **NOT** clk **AFTER** 100 ns**;**

exp**:** **ENTITY** work**.**Gen**(**Gen\_struct**)** **PORT** **MAP** **(**clk**,**inputA**,**inputB**,**generatedResult**);**

ripple **:** **ENTITY** work**.**multiplier**(**multiplier\_ripple\_struct**)** **PORT** **MAP** **(**inputB**,**inputA**,**multiplierResult**);**

compare **:** **ENTITY** work**.**TestAnalyzer**(**analyzer\_struct**)** **PORT** **MAP** **(**clk**,**generatedResult**,**multiplierResult**);**

**END** **ARCHITECTURE** test\_ripple\_multiplier**;**

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

-- test bench for carry look ahead based multiplier:

**ARCHITECTURE** test\_lookAhead\_multiplier **OF** testMultiplier **IS**

**SIGNAL** clk **:** std\_logic **:=**'0'**;**

**SIGNAL** inputA **:** std\_logic\_vector **(**3 **DOWNTO** 0**);**

**SIGNAL** inputB **:** std\_logic\_vector **(**5 **DOWNTO** 0**);**

**SIGNAL** generatedResult **:** std\_logic\_vector **(**9 **DOWNTO** 0**);**

**SIGNAL** multiplierResult **:** std\_logic\_vector **(**9 **DOWNTO** 0**);**

**BEGIN**

clk **<=** **NOT** clk **AFTER** 120 ns**;**

exp**:** **ENTITY** work**.**Gen**(**Gen\_struct**)** **PORT** **MAP** **(**clk**,**inputA**,**inputB**,**generatedResult**);**

cla0 **:** **ENTITY** work**.**multiplier**(**multiplier\_cla\_struct**)** **PORT** **MAP** **(**inputB**,**inputA**,**multiplierResult**);**

compare **:** **ENTITY** work**.**TestAnalyzer**(**analyzer\_struct**)** **PORT** **MAP** **(**clk**,**generatedResult**,**multiplierResult**);**

**END** **ARCHITECTURE** test\_lookAhead\_multiplier**;**

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