ISSN: 2454-1907
DOI: https://doi.org/10.29121/ijetmr.v6.i6.2019.392

IJETMR

International Journal of Engineering Technologies and Management Research
A Knowledge Repository

VERIFICATION OF CARRY LOOK AHEAD ADDER USING CONSTRAINED RANDOMIZED LAYERED TEST BENCH

Dr. Anuradha M. Sandi [*1]
[*1] Department of Electronics and Communication Engineering, Guru Nanak Dev Engineering Collage, India

Abstract:
In processors and in digital circuit designs, adder is an important component. As a result, adder is the main area of research in VLSI system design for improving the performance of a digital system. The performance depends on power consumption and delay. Adders are not only used for arithmetic operations, but also for calculating addresses and indices. In digital design we have half adder and full adder, by using these adders we can implement ripple carry adder (RCA). RCA is used to perform any number of additions. In this RCA is serial adder and it has propagation delay problem. With increase in hard & fast circuits, delay also increases simultaneously. That's the reason these Carry look ahead adders (CLA) are used. The carry look ahead adder speeds up the addition by reducing the amount of time required to determine carry bits. It uses two blocks, carry generator (Gi) and carry propagator (Pi) which finds the carry bit in advance for each bit position from the nearest LSB, if the carry is 1 then that position is going to propagate a carry to next adder.

Keywords: VLSI Design; Propagation Delay; Carry Generator; Carry Propagator; Ripple Carry Adder.

Cite This Article: Dr. Anuradha M. Sandi. (2019). "VERIFICATION OF CARRY LOOK AHEAD ADDER USING CONSTRAINED RANDOMIZED LAYERED TEST BENCH." *International Journal of Engineering Technologies and Management Research*, 6(6), 40-50. DOI: https://doi.org/10.29121/ijetmr.v6.i6.2019.392.

1. Introduction

A ripple-carry adder works in the same way as pencil-and-paper methods of addition. Starting at the rightmost (least significant) digit position, the two corresponding digits are added and a result obtained. It is also possible that there may be a carry out of this digit position (for example, in pencil-and-paper methods, "6 + 6 = 2, carry 1"). Accordingly, all digit positions other than the rightmost one need to take into account the possibility of having to add an extra 1 from a carry that has come in from the next position to the right.
Carry-look ahead depends on two things:

1) Calculating for each digit position whether that position is going to propagate a carry if one comes in from the right.
2) Combining these calculated values to be able to deduce quickly whether, for each group of digits, that group is going to propagate a carry that comes in from the right.

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

Supposing that groups of four digits are chosen the sequence of events goes something like this:

1) All 1-bit adders calculate their results. Simultaneously, the look ahead units perform their calculations.
2) Assuming that a carry arises in a particular group, that carry will emerge at the left-hand end of the group within at most five gate delays and start propagating through the group to its left.
3) If that carry is going to propagate all the way through the next group, the look ahead unit will already have deduced this. Accordingly, before the carry emerges from the next group, the look ahead unit is immediately (within one gate delay) able to tell the next group to the left that it is going to receive a carry – and, at the same time, to tell the next look ahead unit to the left that a carry is on its way.

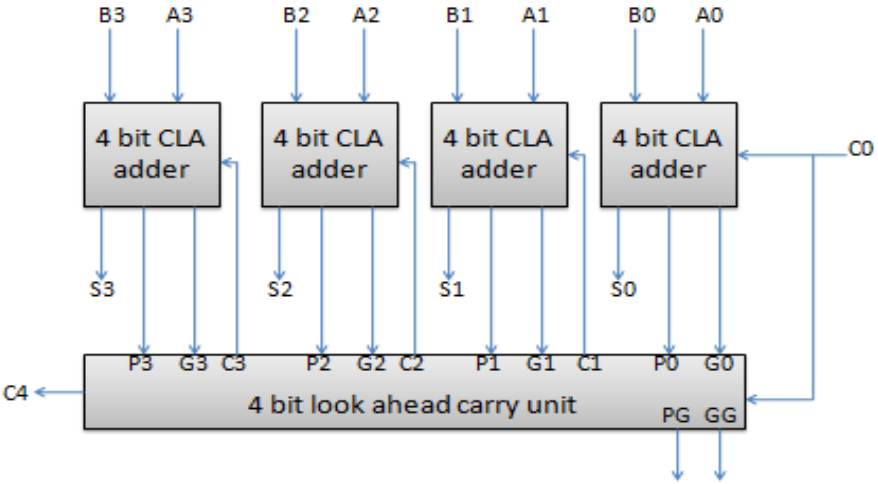


Figure 1: 4-bit carry look ahead adder

The net effect is that the carries start by propagating slowly through each 4-bit group, just as in a ripple-carry system, but then move four times as fast, leaping from one look ahead-carry unit to the next. Finally, within each group that receives a carry, the carry propagates slowly within the digits in that group.

The more bits in a group, the more complex the look ahead carry logic becomes, and the more time is spent on the "slow roads" in each group rather than on the "fast road" between the groups (provided by the look ahead carry logic). On the other hand, the fewer bits there are in a group, the more groups have to be traversed to get from one end of a number to the other, and the less acceleration is obtained as a result.

It is possible to have more than one level of look ahead-carry logic, and this is in fact usually done. Each look ahead-carry unit already produces a signal saying "if a carry comes in from the right, I will propagate it to the left", and those signals can be combined so that each group of, say, four look ahead-carry units becomes part of a "super group" governing a total of 16 bits of the numbers being added. The "super group" look ahead-carry logic will be able to say whether a carry entering the super group will be propagated all the way through it, and using this information, it is able to propagate carries from right to left 16 times as fast as a naive ripple carry. With this kind of two-level implementation, a carry may first propagate through the "slow road" of individual adders,

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

then, on reaching the left-hand end of its group, propagate through the "fast road" of 4-bit look ahead-carry logic, then, on reaching the left-hand end of its super group, propagate through the "superfast road" of 16-bit look ahead-carry logic.

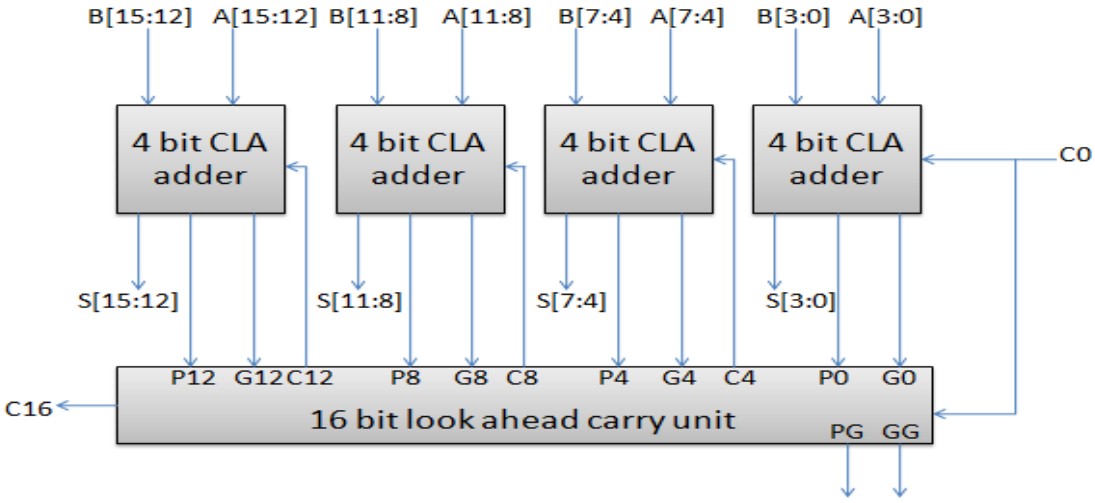


Figure 2: 16-bit carry look ahead adder

2. Materials and Methods

2.1. Carry Look Ahead Method

Carry-look ahead logic uses the concepts of generating and propagating carries. Although in the context of a carry-look ahead adder, it is most natural to think of generating and propagating in the context of binary addition, the concepts can be used more generally than this. In the descriptions below, the word digit can be replaced by bit when referring to binary addition of 2.

The addition of two 1-digit inputs A and B is said to generate if the addition will always carry, regardless of whether there is an input-carry (equivalently, regardless of whether any less significant digits in the sum carry). For example, in the decimal addition 52 + 67, the addition of the tens digits 5 and 6 generates because the result carries to the hundreds digit regardless of whether the ones digit carries (in the example, the ones digit does not carry (2 + 7 = 9).

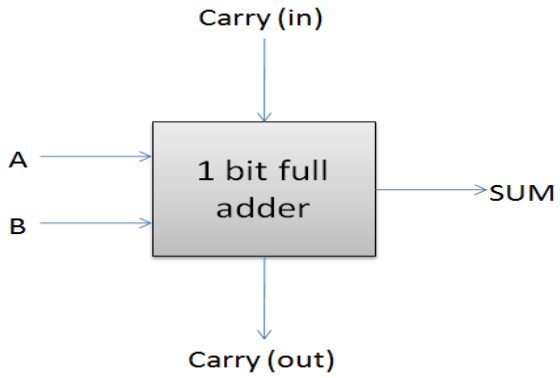


Figure 3: 1-bit full adder

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

Table 1: 1-Bit Full Adder Truth Table

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the case of binary addition A, B generates if and only if both A and B are 1. If we write G (A, B) to represent the binary predicate that is true if and only if A.B generates, we have
G (A, B) = A.B

The addition of two 1-digit inputs A and B is said to propagate if the addition will carry whenever there is an input carry (equivalently, when the next less significant digit in the sum carries). For example, in the decimal addition 37 + 62, the addition of the tens digits 3 and 6 propagate because the result would carry to the hundreds digit if the ones were to carry (which in this example, it does not). Note that propagate and generate are defined with respect to a single digit of addition and do not depend on any other digits in the sum.

In the case of binary addition, A+B propagates if and only if at least one of A or B is 1. If P(A,B) is written to represent the binary predicate that is true if and only if A+B propagates, one has

P(A+B) =A+B

Sometimes a slightly different definition of propagate is used. By this definition A + B is said to propagate if the addition will carry whenever there is an input carry, but will not carry if there is no input carry. Due to the way generate and propagate bits are used by the carry-lookahead logic, it doesn't matter which definition is used. In the case of binary addition, this definition is expressed by

P`(A, B)=A^B

For binary arithmetic, or is faster than xor and takes fewer transistors to implement. However, for a multiple-level carry-lookahead adder, it is simpler to use P`(A,B) Given these concepts of generate and propagate, a digit of addition carries precisely when either the addition generates or the next less significant bit carries and the addition propagates. Written in boolean algebra, with Ci the carry bit of digit *i*, and Pi and Gi the propagate and generate bits of digit *i* respectively,

Ci+1= Gi+(Pi.Ci).

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

2.2. Implementation Details

For each bit in a binary sequence to be added, the carry-look ahead logic will determine whether that bit pair will generate a carry or propagate a carry. This allows the circuit to "pre-process" the two numbers being added to determine the carry ahead of time. Then, when the actual addition is performed, there is no delay from waiting for the ripple-carry effect (or time it takes for the carry from the first full adder to be passed down to the last full adder).

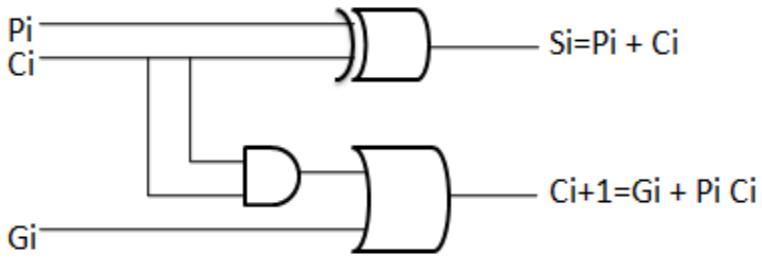


Figure 4: Sum and Carry generation using CLA

Equations:
C1=G0+P0C0
C2=G1+P1C1
C3=G2+P2C2
C4=G3+P3C3

Substituting then into yields the expanded equations:
C1 = G0 + P0C0
C2 = G1 + P1C1
C2= G1 + P1 (G0 + P0C0) = G1 + P1G0 + P1P0C0
C3 = G2 + P2C2 = G2 + P2G1 + P2P1G0 + P2P1P0C0
C4 = G3 + P3C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0
The calculation of the gate delay of a 16-bit adder (using 4 CLAs and 1 LCU) is not as straight forward as the ripple carry adder.

Starting at time of zero:

- calculation of Pi and Gi is done at time 1,
- calculation of Ci is done at time 3,
- calculation of the PG is done at time 2,
- calculation of the GG is done at time 3,
- calculation of the inputs for the CLAs from the LCU are done at:
- time 0 for the first CLA,

1) time 5 for the second, third and fourth CLA,

calculation of the Si are done at:

2) time 4 for the first CLA,
3) time 8 for the second, third & fourth CLA,

- calculation of the final carry bit (C16) is done at time 5.

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

The maximal time is 8 gate delays (for S[8:15]).
A standard 16-bit ripple-carry adder would take 16 × 3 − 1 = 47 gate delays.

2.3. Cadence Design Systems

1) American multinational design automation (EDA) software and engineering services company, founded in 1988 by the merger of SDA Systems and ECDA inc.
2) The company produces software, hardware and silicon structures for designing integrated circuits, systems on chips (SoCs) and printed circuit boards.

Overview

- Cadence Design Systems, headquartered in san jose california, in the North San Jose Innovation District, is a supplier of electronic design technologies and engineering services in the electronic design automation (EDA) industry.
- Cadence products primarily target SoC design engineers, and are used to move a design into packaged silicon, with products for custom and analog design, digital design, mixed-signal design, verification, and package/PCB design, as well as a broad selection of IP, and also hardware for emulation and FPGA prototyping.
- The company also provides products that assist with the development of complete hardware and software platforms that support end applications.

Features of Cadence

- Easy-to-use interactive simulation environment.
- Built-in waveform display and signal analysis capabilities.
- Integral part of the Virtuoso custom design platform

3. Results and Discussions

Carry Look Ahead (CLA) design is based on the principle of looking at lower adder bits of argument and addend if higher orders carry generated. This adder reduces the carry delay by reducing the number of gates through which a carry signal must propagate. As shown in figure in the generation and propagation stage, the generation values, propagation values are computed. Internal carry generation is calculated in second stage. And in final stage, the sum is calculated. The architecture of CLA is given in fig 5 and flow chart of CLA is given in *fig* 6.

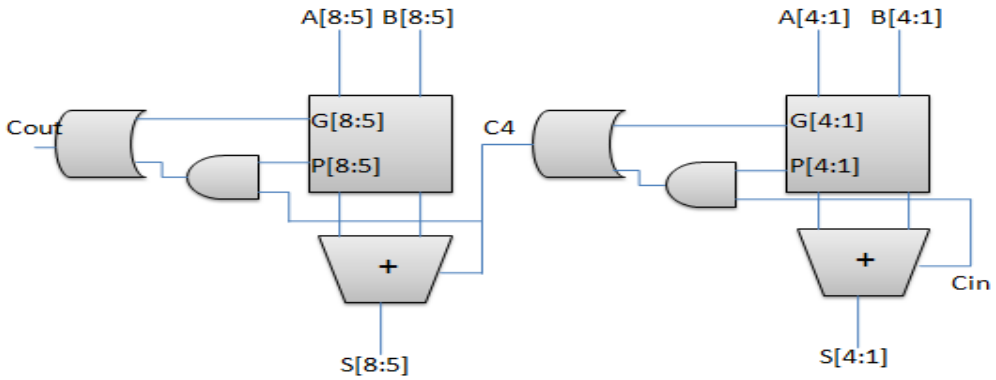


Figure 5: architecture of CLA

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

The fact when the carry will be generated is when both bits A and B are 1, and when one of the two bits is 1 and the carry-in (carry of the previous stage) is 1.

Table 2: Truth Table Of CLA

A	B	Cin	Cout	Condition
0	0	0	0	No carry generation
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

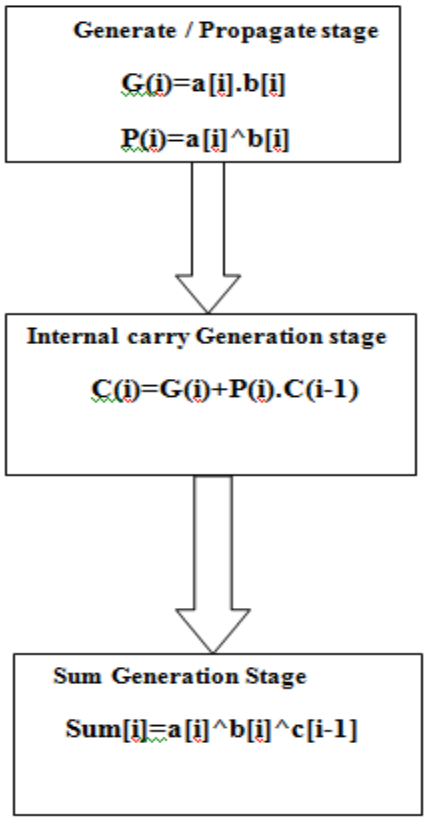
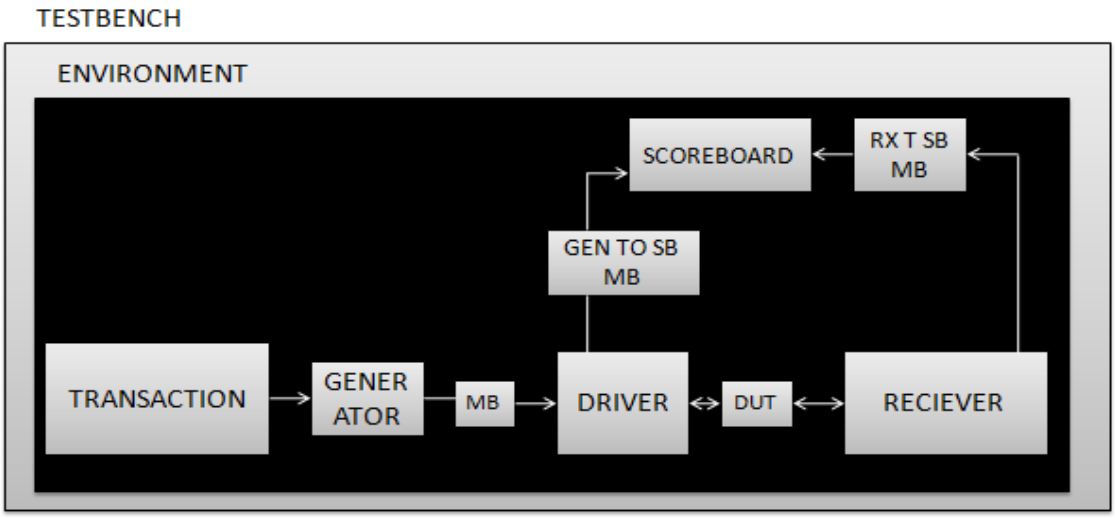


Figure 6: flow chart of CLA

3.1. Layered Testbench

A key concept for any modern verification methodology is the Layered test bench. Although this process may seem to make the testbench more complex, it actually helps to make your task easier by dividing the code into smaller pieces that can be developed separately. A single routine cannot randomly generate all types of stimulus and it becomes complex and not maintainable.

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207



Transaction Class

Transaction class is the base class for all other components in the LTB. The transaction class specifies the variables that are to be randomized. It also holds the constraints for randomization. All the other classes access the variables from the transaction class.

Generator Class

The generator class generates the stimulus required for verification. The stimulus generation will be mainly based on randomization of variables, based on the variables from base/ transaction class. The generated variables are sent through mailbox to driver as object level.

Mailbox

A mailbox is a mechanism to exchange messages between processes. Data can be sent to a mailbox by one process and retrieved by another. Data can be any valid system Verilog data types, including class data types. Mailboxes are created with the new () method. The number of messages in a mailbox can be obtained via the num () method. The num () method returns the number of messages currently in the mailbox.

The returned value should be used with care because it is valid only until the next get () or put () is executed on the mailbox. The put () method places a message in a mailbox. The put () method stores a message in the mailbox in strict FIFO order. The get () method retrieves a message from a mailbox. The get () method removes one message from queue.

Driver

The driver drives signal from object level to signal level and forces to the DUT connected. Here, in this LTB the generated data send through mailbox is collected by driver and converted into signal level and forced to DUT. Before sending data to DUT, through another mailbox driver sends set of values generated to scoreboard also.

Receiver & Scoreboard

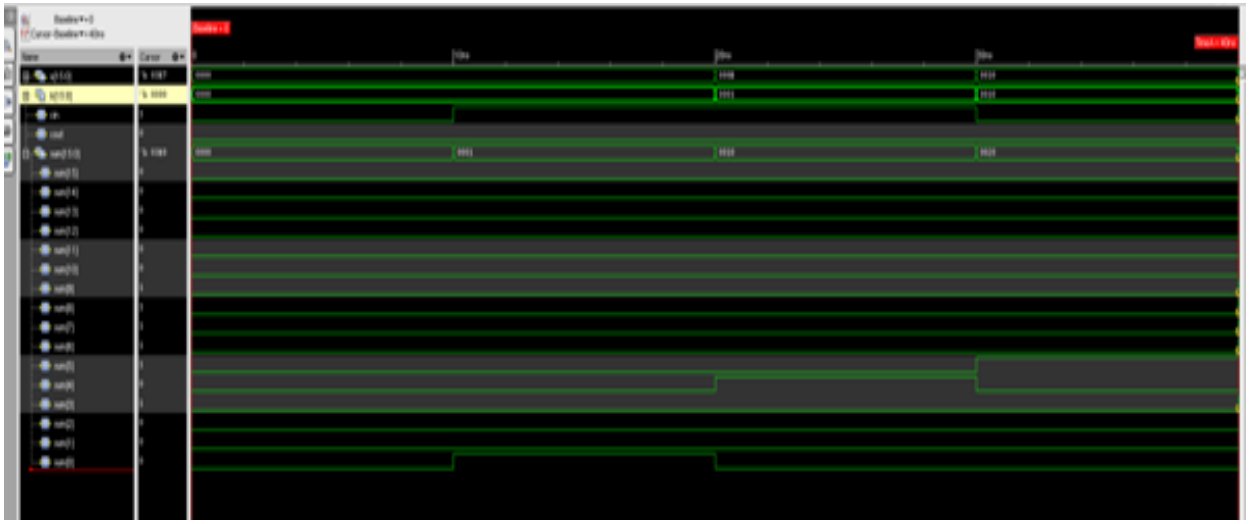
The receiver receives the data from DUT i.e., its outputs. It collects from signal level to object level and sends it back to scoreboard through a mailbox. Scoreboard usually checks the data collected from driver to that of data collected from receiver for their functional correctness.

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

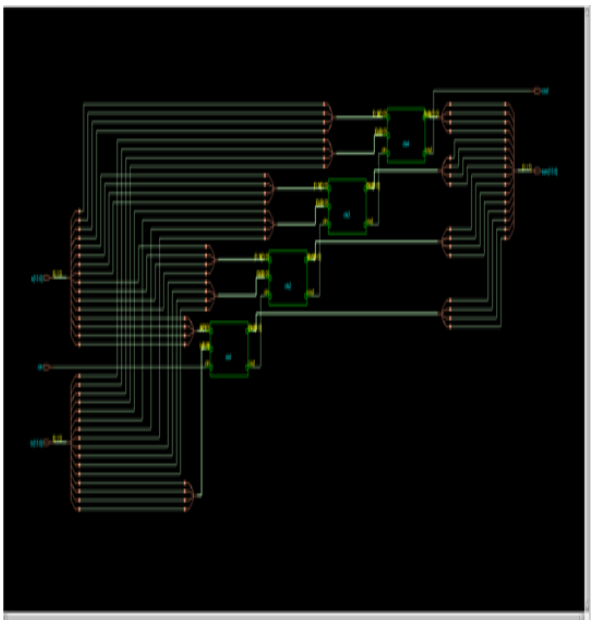
Interfaces

Interfaces are used to define physical existence of the signals. Virtual interfaces are used so as to apply OOPs concepts on interfaces, which we can't perform on real interface signals. All the LTB components are considered to work under Environment and environment is invoked under the test bench.

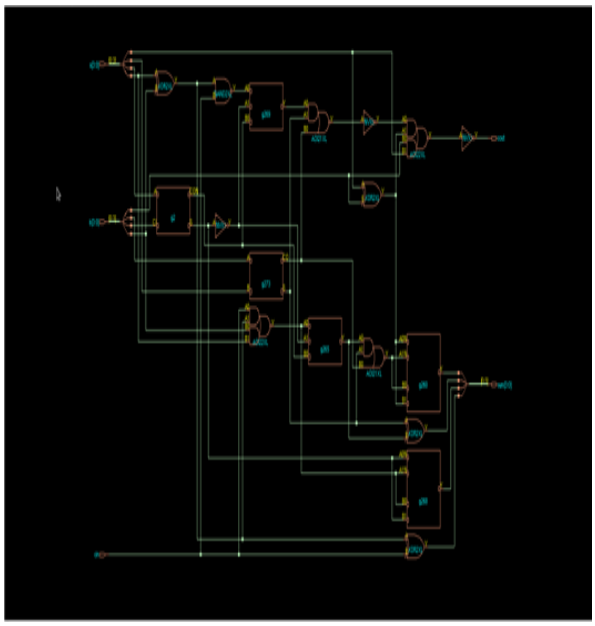
3.2. Simulation Result



(a)



(b)



(c)

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
carry_look_ahead_16bit	72	48.693	77639.358	77688.051
cla1	18	12.179	17273.826	17286.005
cla2	18	12.179	17841.233	17853.411
cla3	18	12.179	18331.322	18343.500
cla4	18	12.157	16516.607	16528.764

(d)

Instance	Cells	Cell Area	Net Area	Total Area	Wireload
carry_look_ahead_16bit	72	1370	0	1370	<none> (D)
cla4	18	343	0	343	<none> (D)
cla3	18	343	0	343	<none> (D)
cla2	18	343	0	343	<none> (D)
cla1	18	343	0	343	<none> (D)

(D) = wireload is default in technology library

(e)

Figure 7 (a): 16-bit CLA simulation waveform (b)16-bit CLA synthesized diagram (c) 4-bit CLA synthesized diagram (d) 16-bit CLA Power analysis (e) 16-bit CLA Area analysis

4. Conclusions and Recommendations

The selected adder circuit with minimum area , power and delay is carry increment adder for 4-bit, in case of an 8-bit, there is a competence between Carry Look ahead and Carry increment adder and in case of 16-bit , carry skip adder has minimum area and delay which proves to be easy solution in improving the speed of the adder circuit over other conventional adder circuits in discussion suffering from disadvantage of either occupying more number of slices or look up tables per unit of cell or have highest minimum propagation delay owing to their critical carry path for same power . The selected adder circuit is also found to have comparatively less power consumption in comparison to other adder circuits. Hence it can be concluded that the above respective adder circuits on the basis of different word size can be used to speed up the final addition in parallel multiplier circuits and other architectures which uses adder circuits, exhibiting maximum efficiency.

Acknowledgements

It gives me immense pleasure to express my deep sense of gratitude to Principal and Management, Guru Nanak Dev Engineering College Bidar, Karnataka, INDIA for their inspiration and academic support by providing good facilities.

ISSN: 2454-1907
DOI: 10.5281/zenodo.3245207

References

[1] Y.Choi, "Parallel Prefix Adder Design" Proc. 17th IEEE Symposium on Computer Arithmetic, pp 90-98, 27th June 2005.
[2] A. Beaumont-Smith, C.C. Lim: Parallel Prefix Adder Design, 15th Symposium on Computer Arithmetic, p.218-225, 200.
[3] G. Yang, S.-O. Jung, K.-H. Baek, S.H. Kim, S. Kim, S.-M. Kang, "A 32-bit Carry Lookahead Adder using Dual-path all-N logic," IEEE Trans. VLSI Systems, vol. 13, no. 8, pp. 992-996, 2005.
[4] P. Kogge and H. Stone, - A Parallel Algorithm for the Efficient Solution of a General class of Recurrence Relation, IEEE Transactions on computers, vol, C-22, no.8, pp.786-793, Aug1973
[5] Swaroop Ghosh, Patrick Ndai, Kaushik Roy. "A Novel Low Overhead Fault Tolerant Kogge-Stone Adder using Adaptive Clocking", date 2008.
[6] Jin-Fa Lin, Yin- Tsung Hwang and Ming – Hwa Sheu, -, "Low Power 10-Transistor Full adder Design Based on Degenerate pass Transistor Logic." IEEE Trans. VLSI Systems, vol. 13, No.6, pp.686-695, Jun 2012.
[7] Pakkiraiah Chakali, Madhu Kumar Patnala, "Design of High speed Kogge-Stone based Carry Select Adder,"International Journal of Emerging Science and Engineering (IJESE), Volume-1, Issue-4, February 2013
[8] Chakali, Pakkiraiah, and Madhu Kumar Patnala. "Design of High Speed Kogge-Stone Based Carry Select Adder." International Journal of Emerging Science and Engineering (IJESE), vol. 1, no. 4, pp. 34-37, Feb. 2013.
[9] Mala, T. Ratna, R. Vinay Kumar, and T. Chandra Kala. "Design and Verification of Area Efficient High-Speed Carry Select Adder." IJRCCT, vol. 1, no. 6, pp. 345-349, Nov. 2012.
[10] Sunil M, Ankit R D, Manjunatha G D and Premananda B S "Design And Implementation of Fast Parallel Prefix Kogge Stone Adder." In International Journal of Electrical and Electronics Engineering & Telecommunications, vol. 3, no. 1, January 2014.
[11] Begum, Mohammed Haseena, and V. Vamsi Mohana Krishna. "Design and Verification Of Low Power And Area Efficient Kogge-Stone Carry Select Adder." In International Journal of Engineering Research and Technology, ESRSA Publications, vol. 2, no. 8, pp. 462-467, Aug 2013.
[12] B. Ramkumar, Harish M Kittur, "Low –Power and Area-Efficient Carry Select Adder", IEEE transaction on very large scale integration (VLSI) systems, vol.20, no.2, pp.371-375, Feb 2012.
[13] Adilakshmi Siliveru, M.Bharathi, "Design of Kogge-Stone and Brent-Kung adders using Degenerate Pass Transistor Logic." IJESE, ISSN: 2319–6378, Volume-1, Issue-4, February 2013.
[14] K-H.Cheng, W-S. Lee and Y-C.Huang, " A 1.2 V 500 MHZ 32-bit Carry Lookahead Adder," 8th IEEE International Conference on Electronics, Circuits and Systems, Vol 2, September 2-5, 2001, pp. 765-768.
[15] Y. Kim and L-S Kim, '64-bit Carry-Select Adder with Reduced Area," Electronics letters, Vol. 37, Issue 10, May 10, 2001, pp. 614-615.

*Corresponding author.
E-mail address: anu29975@ gmail.com