TABLE OF CONTENTS

ACKNOWLEDGEMENT

I wish to place on record my deep sense of gratitude for the instructor Dr. James Leffew for his constant guidance throughout the course.

OBJECTIVE

The purpose of this project is design and simulation of a sixteen bit Conditional Sum Adder using four bit slice components with Look ahead carry. The Look ahead carry is incorporated in the 4 bit slices.

INTRODUCTION

High performance microprocessors demand faster arithmetic operations. The addition of two operations is the most frequent operation in almost any microprocessor arithmetic unit. A two-operand adder is used not only when performing additions and subtractions but also employed in some more complex operations such as multiplication, division, and other functions. Propagation delay has been one of the major problems facing engineers working to implement high-speed circuits. High propagation delays in binary addition will result in a highly amplified propagation delay at the output of the circuit.

There are many ways of formulating the process of binary addition. Each different way provides different insight and thus suggests different implementations. Examples are Weinberger & Smith's carry-look ahead (CLA) adder, Nadler's pyramid adder ,Sklansky's conditional sum adder , Bedrij's carry-select adder, and Ladner & Fischer's prefix adder.

The following table compares the various binary addition techniques with Conventional ripple Adder(1).

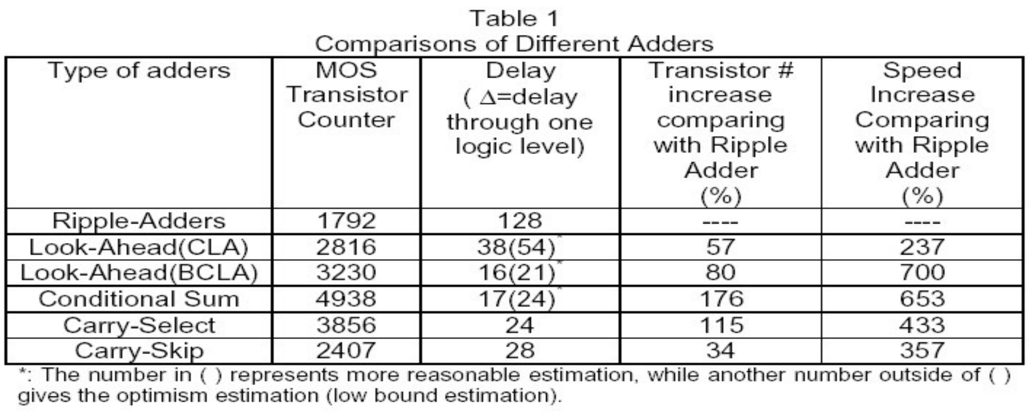


Table 1
Comparisons of Different Adders

Type of adders	MOS Transistor Counter	Delay (Δ=delay through one logic level)	Transistor # increase comparing with Ripple Adder (%)	Speed Increase Comparing with Ripple Adder (%)
Ripple-Adders	1792	128	----	----
Look-Ahead(CLA)	2816	38(54)*	57	237
Look-Ahead(BCLA)	3230	16(21)*	80	700
Conditional Sum	4938	17(24)*	176	653
Carry-Select	3856	24	115	433
Carry-Skip	2407	28	34	357

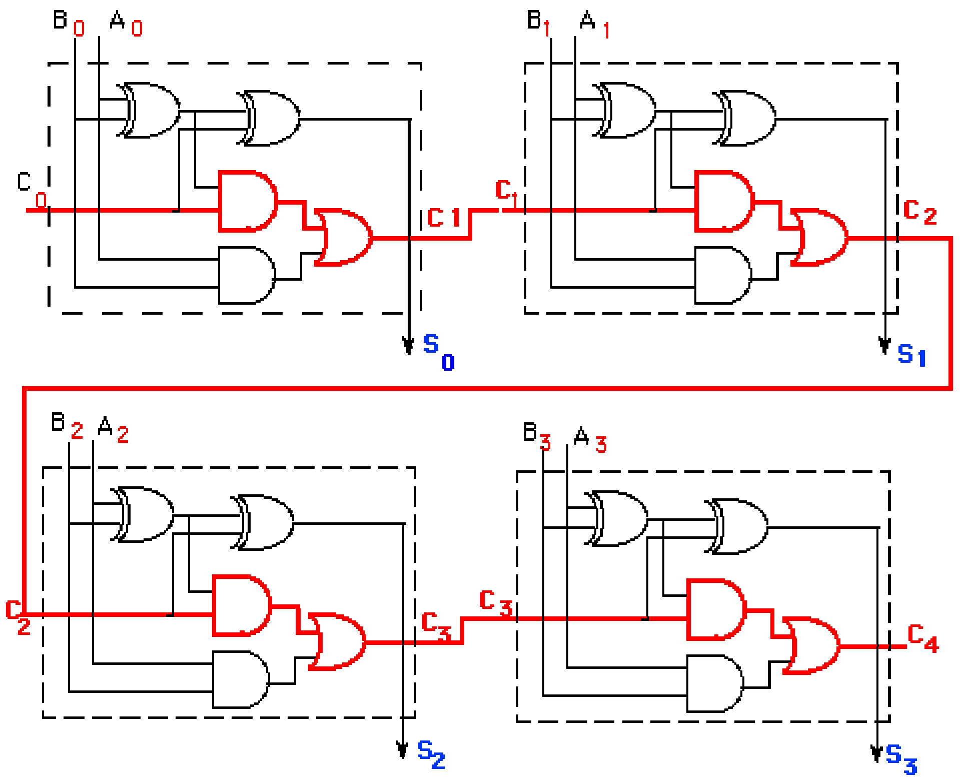
*: The number in ( ) represents more reasonable estimation, while another number outside of ( ) gives the optimism estimation (low bound estimation).

TOOL USED

MAX PLUS II (Version 10.1) from ALTERA has been used in this project. MAX+PLUS II software is a fully integrated, architecture-independent package for designing logic with Altera programmable logic devices. MAX+PLUS II offers a full spectrum of logic design capabilities: three design entry methods for hierarchical designs; floorplan editing; powerful logic synthesis; design partitioning; functional, timing, and board-level-type linked simulation; detailed timing analysis; automatic error location; and device programming and verification.

ADDER STRUCTURE

Carry Look Ahead Adder(CLA) Technique is used to speed up carry propagation in adder complex. Hence the carries entering all the bit positions of a "parallel" adder are generated simultaneously by additional logic circuitry. This results in a constant addition delay independent of the length of the adder.



Conventional Ripple carry Adder

For the conventional ripple carry adder shown above, the Sum of the most significant stage will be valid after 2(N-1) + 1 gate delays, in which N is the number of bits . The carry-out bit will be valid after 2N gate delays. This delay may be in addition to any delays associated with interconnections. It should be mentioned that in case one implements the circuit in a FPGA, the delays may be different from the above expression depending on how the logic has been placed in the look up tables and how it has been divided among different CLBs (Configurable Logic Block) .For instance, for a

32-bit adder, the delay would be about 63 ns if one assumes a gate delay of 1 ns. That would imply that the maximum frequency one can operate this adder would be only 16 MHz! For fast applications, a better design is required.

The carry-look-ahead adder solves this problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases: (1) when both bits A_i and B_i are 1, or (2) when one of the two bits is 1 and the carry-in (carry of the previous stage) is 1. Thus, one can write,

$$C_{OUT} = C_{i+1} = A_i.B_i + (A_i \text{ XOR } B_i).C_{i.} \qquad (1)$$

Which can be also written as,

$$C_{i+1} = G_i + P_i.C_i \qquad (2)$$

in which

$$G_i = A_i.B_i \qquad (3)$$

$$Pi = (A_i \text{ XOR } B_i) \qquad (4)$$

are called the Generate ($G_{i)}$ and Propagate (Pi) term.

Notice that both the Propagate and Generate terms only depend on the input bits and thus will be valid after one gate delay.Let's apply this to a 4-bit adder.

$$C_1 = G_0 + P_0.C_0 \qquad (5)$$

$$C_2 = G_1 + P_1.C_1 = G_1 + P1.G_0 + P_1.P_0.C_0 \qquad (6)$$

$$C_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0 \qquad (7)$$

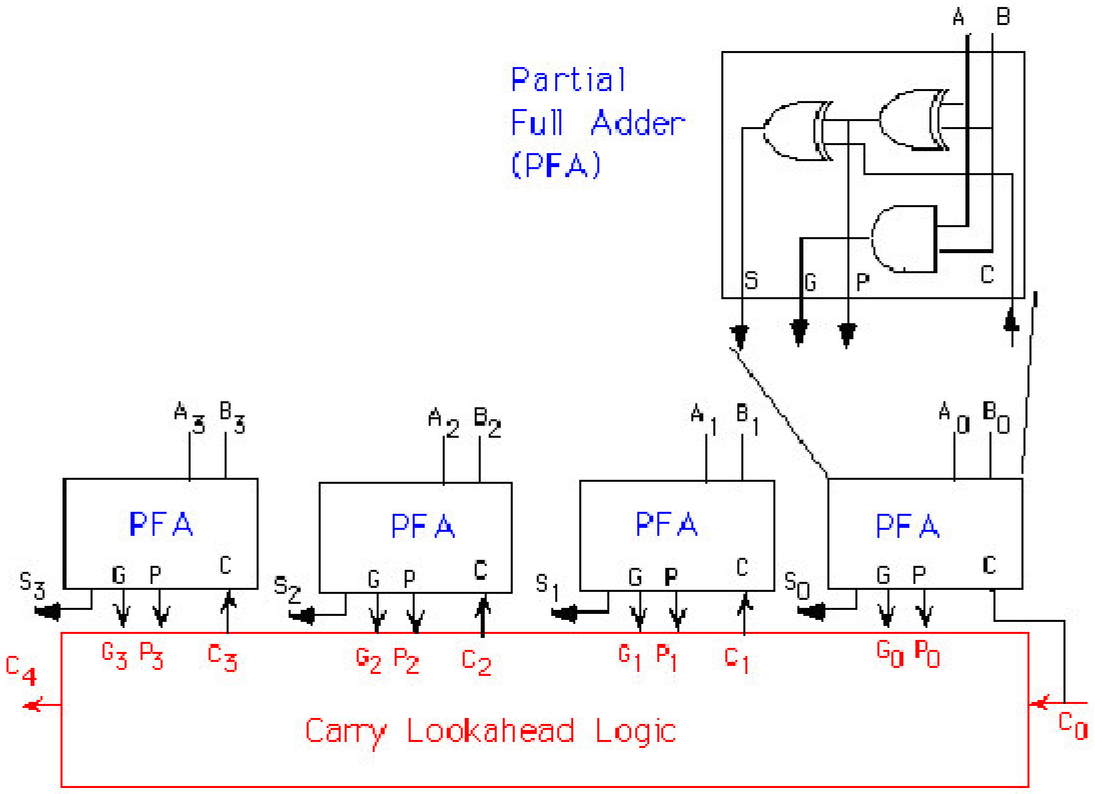
$$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3P_2.P_1.G_0 + P_3P_2.P_1.P_0.C_0 \qquad (8)$$

Notice that the carry-out bit, C_{i+1}, of the last stage will be available after **three** delays (one delay to calculate the Propagate signal and two delays as a result of the AND and OR gate). The Sum signal can be calculated as follows,

$$S_i = A_i \text{ XOR } B_i \text{ XOR } C_i = P_i \text{ XOR } C_i. \qquad (9)$$

The Sum bit will thus be available after one additional gate delay. The advantage is that these delays will be the same independent of the number of bits one needs to add, in contrast to the ripple counter.

The carry-look ahead adder can be broken up in two modules: (1) the Partial Full Adder, PFA, which generates Si, Pi and Gi as defined by equations 3, 4 and 9 above; and (2) the Carry Look-ahead Logic, which generates the carry-out bits according to equations 5 to 8. The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead logic block as shown in Figure .



4 bit CARRY LOOK AHEAD LOGIC ADDER

A 16-Bit CLA Adder could be constructed continuing along in the same logic pattern, with the MSB carry-out resulting from the OR of 16 AND gates. This would make the 16-Bit CLA Adder as fast as the 1-Bit Ripple Carry Adder.

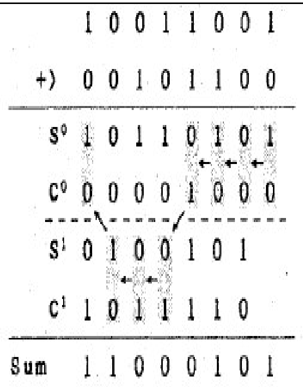
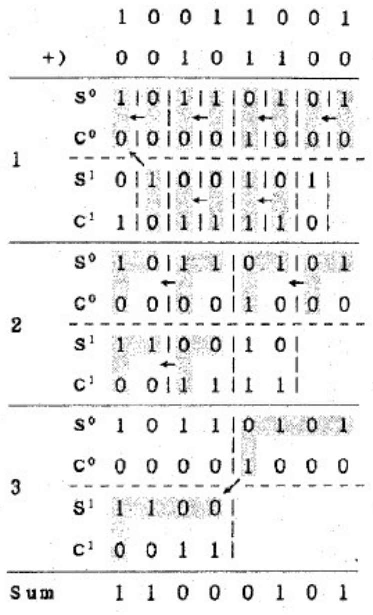
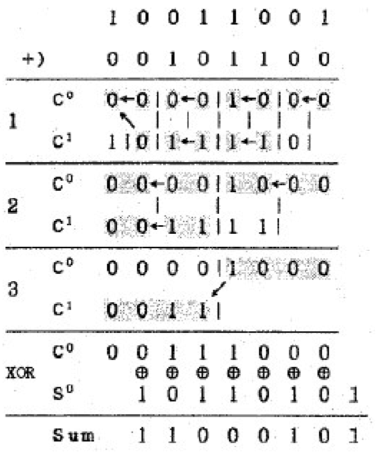
However, another plausible method to create a 16-Bit CLA Adder would be to ripple the carry-out of a 4-Bit CLA Adder to the carry-in of another 4-Bit CLA Adder, using four 4-Bit modules total. This would make the 16-Bit CLA Adder as fast as the 4-Bit Ripple Carry Adder. Each 4 bit slices will have a Partial full adder or conditional sum adder, 2 to 1 Multiplexer and look-ahead carry logic.

An algorithm for fast addition -conditional sum addition(CSA) was presented by J.Sklansky early in 1960.It is possible to design a adder with up to five or six times the ripple adder performance by using CSA algorithm, but it needs larger size of area. It is shown that the conditional sum adder has a better power-delay product than other adders for high speed applications(2) . The Conditional sum addition rules can overcome the carry propagation problems. It generates distant carriers and using these carriers to select the true sum outputs from two simultaneously generated provisional sums under different carry input conditions. The following table shows the 8-bit addition, where the arrows show the actual carries generated between sections. It is seen that simultaneous additions are performed on all sections independently. The addition process of a n bit adder is completed in t steps, where

$$t = \log_2 n \tag{10}$$

where n is number of input bits.

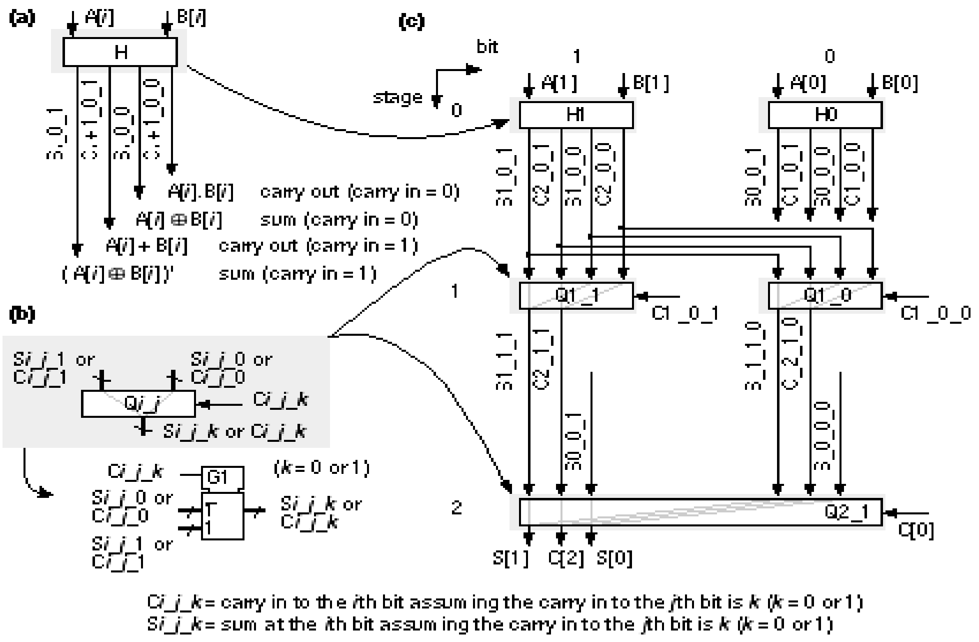
Ripple Carry Addition	**Conditional Sum addition**	**Conditional carry Addition**

 1 0 0 1 1 0 0 1 +) 0 0 1 0 1 1 0 0 S⁰ 1 0 1 1 0 1 0 1 C⁰ 0 0 0 0 1 0 0 0 S¹ 0 1 0 0 1 0 1 C¹ 1 0 1 1 1 1 0 Sum 1 1 0 0 0 1 0 1	C^0=A and B C^1= A OR B S^0=A XOR B S^1= NOT (A XOR B)  1 0 0 1 1 0 0 1 +) 0 0 1 0 1 1 0 0 1 2 3 S⁰ C⁰ S¹ C¹ Sum 1 1 0 0 0 1 0 1	C^0=A and B C^1= A OR B  1 0 0 1 1 0 0 1 +) 0 0 1 0 1 1 0 0 1 2 3 C⁰ C¹ XOR S⁰ 1 0 1 1 0 1 0 1 Sum 1 1 0 0 0 1 0 1

An improvement of the conditional sum addition is conditional carry addition, which is shown in table. It also has no carry propagation problem. The generated distant carriers are used to select the true carry inputs from two simultaneously generated provisional carriers under different carry input conditions. The arrows show the actual carriers generated between sections The simultaneous carry generations are performed on all section independently. The conditional carry addition of an 8 bit addition is completed in 3 steps. An extra XOR function of the cout and S^0 is required to generate the final sum outputs, the final results.

Suppose we have an n -bit adder that generates two sums: One sum assumes a carry-in condition of ' 0' , the other sum assumes a carry-in condition of ' 1' . We can split this n -bit adder into an i -bit adder for the i LSBs and an ($n - i$)-bit adder for the $n - i$ MSBs. Both of the smaller adders generate two conditional sums as well as true and complement carry signals. The two (true and complement) carry signals from the LSB adder are used to select between the two ($n - i + 1$)-bit conditional sums from the MSB adder using 2($n - i + 1$) two-input MUXes. This is a **conditional-sum adder** (also often abbreviated to CSA) [Sklansky, 1960]. We can recursively apply this technique. For example, we can split a 16-bit adder using i = 8 and n = 8; then we can split one or both 8–bit adders again—and so on.

Figure shows the simplest form of an n -bit conditional-sum adder that uses n single-bit conditional adders, H (each with four outputs: two conditional sums, true carry, and complement carry), together with a tree of 2:1 MUXes (Qi_j). The conditional-sum adder is usually the fastest of all the adders we have discussed (it is the fastest when logic cell delay increases with the number of inputs—this is true for all ASICs except FPGAs).



(a)
A[i]
B[i]
H
A[i].B[i] carry out (carry in = 0)
A[i] ⊕ B[i] sum (carry in = 0)
A[i] + B[i] carry out (carry in = 1)
(A[i] ⊕ B[i])' sum (carry in = 1)
(b)
Si_j_1 or Ci_j_1
Si_j_0 or Ci_j_0
Qi_j
Ci_j_k
Si_j_k or Ci_j_k
G1
(k = 0 or 1)
(c)
bit
stage
H1
H0
Q1_1
Q1_0
Q2_1
C1_0_1
C1_0_0
C[0]
S[1] C[2] S[0]
Ci_j_k = carry in to the ith bit assuming the carry in to the jth bit is k (k = 0 or 1)
Si_j_k = sum at the ith bit assuming the carry in to the jth bit is k (k = 0 or 1)

FIGURE : The conditional-sum adder. (a) A 1-bit conditional adder that calculates the sum and carry out assuming the carry in is either ' 1' or ' 0' The (b) multiplexer that selects between sums and carries. (c) A 4-bit conditional-sum adder with carry input, C[0].

PROJECT DESCRIPTION

The project objective is met by dividing 16 bit Addition into 4-bit slices.Each slice contains conditional sum generator ,Look Ahead Carry generator and multiplexer components. All the components are stored in user library. Main program is able to call/access each component and utilize its functionality.

The descriptions of each component are given below.

Components Used

4 bit Conditional Sum Generator(CSG):

The inputs A and B, each of 4 bits, are used in this component to produce Sum when when carry is 0 and Sum when carry is 1.It also produces C^0 and C^1.

This module makes use of propagating(P), generating (G) and transfer(T) functions.

Where P,G and T are given by,

P_i = A_i XOR B_i

G_i = A_i AND B_i

T_i = A_i OR B_i

The VHDL code for this component uses behavioral model.

Look-ahead carry (LAC) generator:

It produces carry out using C^0 and C^1 from Conditional sum generator and Carry in.

VHDL Code for LAC uses behavioral model where Carry Out is defined by

Cout = C^0 or (C^1 and Cin).

4 bit 2:1Multiplexer:

It produces output either S^0 or S^1 depending on carry signal from Look Ahead carry unit.Here carry acts as select signal for multiplexer.

VHDL code for 4 bit 2:1 multiplexer uses behavioral model with wait statement.It also incorporates timing Model.

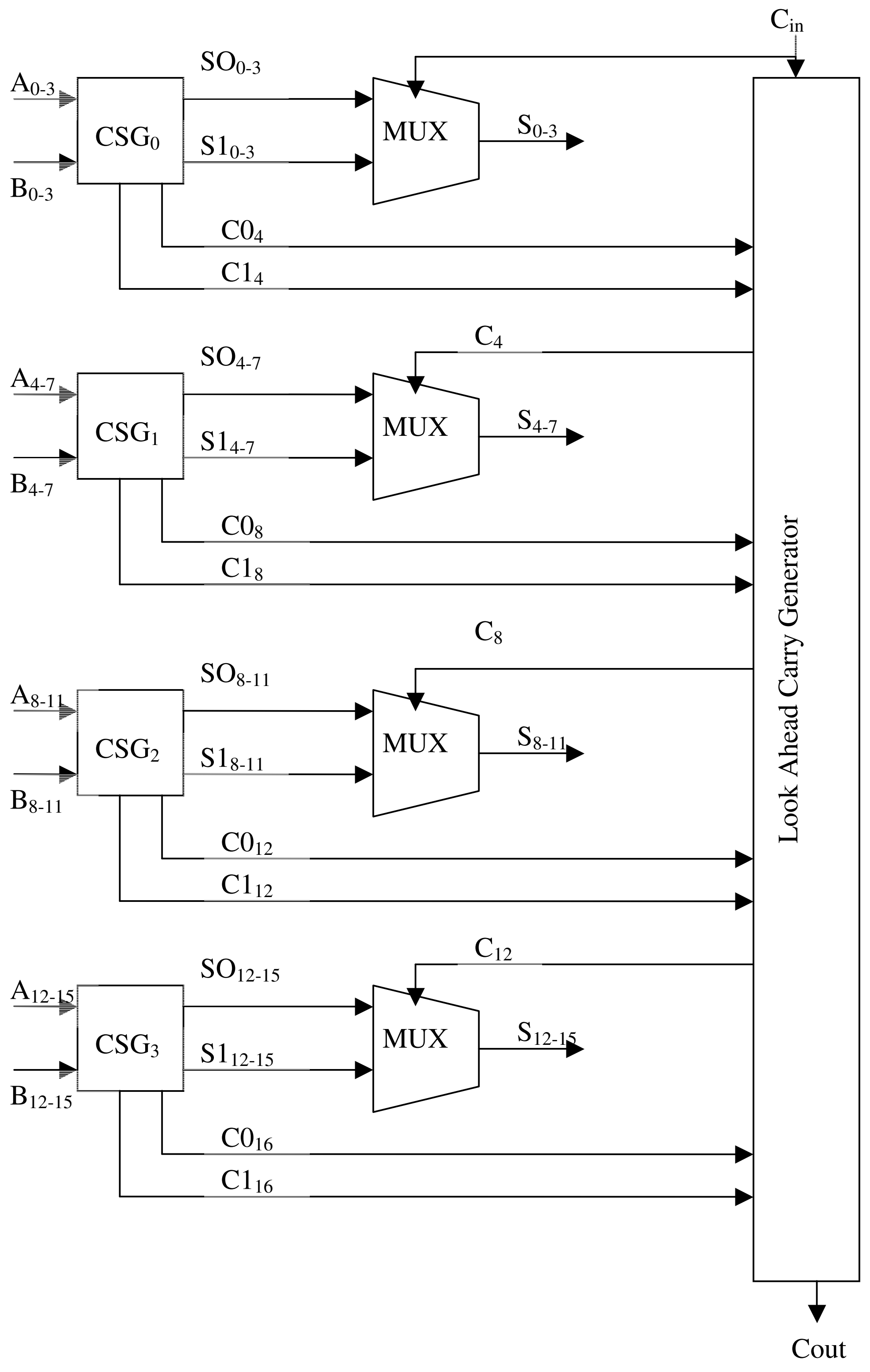
Package

All the components mentioned above are included in separate package called CSA4.It contains port definitions of all the components.VHDL codes for all components and packages are stored in separate folder mylib and this folder is defined as user library when compiling 16 bit adder using Altera MAX II.

16 bit Adder using 4 bit CSA slices

16 Bit Adder is developed by slicing 16 bits into four portions. Addition mechanism for each portion is then developed by using the library mylib. Where mylib contains components CSG, LAC, 2:1 mux .The inputs are *A*, *B* 16-bit vectors and a *Cin* bit with outputs being 16-bit sum and a *Cout* bit. The three components CSG,LAC and 2:1 mux are called in each bit slice and the carry out from each bit slice is passed as carry in to next bit slice. Carry out of the last bit slice is *Cout*. Thus the complete operation of the 16 bit addition is performed by using the components in the package stored in the library.

The 16 bit Adder using the components in the package CSA4 is shown below.



16 BIT CONDITIONAL SUM ADDER

Test bench

Simulation can also be done by providing data needed using a test bench. In simulation, ports of the program being tested are mapped with test bench. Here Test bench provides inputs A, *B* and *Cin* .Outputs *Sum* and *Cout* are generated when main module is called from the test bench. Test bench uses configuration specifications to bind values.

The test bench could be written in such a way that it provides random input values for A and B. The test bench can also be written to detect errors in the circuit and thus aborting the simulation when incorrect results are obtained.

ANALYSIS OF SIMULATION RESULTS:

The following table shows the inputs and corresponding results executed from the 16 bit adder mentioned above.

	INPUT			OUTPUT	
	A	B	Cin	Sum	Cout
Hex	FFFF	0001		0001	
Binary	1111 1111 1111 1111	0000 0000 0000 0001	1	0000 0000 0000 0001	1

The waveforms resulted by simulation are given in appendix.

CONCLUSION

The 16 bit addition is performed by using four 4 bit slices which use conditional sum adder and look ahead carry logic.

This leads to production of faster adder than conventional ripple carry Adder due to decrease in carry propagation delay.

Reference

1. Weihua Chen,"*Implementation and Comparison of 16-bit Look Ahead Adders*", project report for EE8053 Computer Arithmetic Algorithm.

2. Jungang Han and Glen Stone,"*Implementation and verification of conditional sum adder*", 1988-311-23,July 1, 1988,Department of computer science Reports.

3.Anantha Chandrakasan,Robert W.Broersen,"*Minimizing Power consumption in digital CMOS circuit*",in proceedings of IEEE,Vol 83,No 4,pp 498-523,April,1995.

4. M. Horauer and D. Loy, "*Adder Synthesis*", Proceedings of Austrochip '95, Graz Austria, pp. 81--87, 1995.