Analysis & Comparison Different Adders

Mr. Pradeep Kumar Sharma[#1], Ms anamika Singh[#2], Mr. Nityanand Sharma[#3]

#1 RCERT, Sitapura, Jaipur

#2, Suresh Gyan Vihar University,Jaipur

#3, Jagan nath University, Jaipur

[1]pks011279@gmail.com, [4]agarwal_ksh@yahoo.com

Abstract

This paper is primarily deals the construction of high speed adder circuit using Hardware Description Language (HDL) in the platform Xilinx ISE 9.2 iandimplement the monField Programmable Gate Arrays (FPGAs) to analyze the design parameters.Them otivation behind this investigation is that anadder is a very basic building block of Arithmetic Logic Unit (ALU) and would be alimiting factor in performance of Central Processing Unit(CPU) .Design of a high speedsing lecoreprocessor is the future goal of this paper. Single coreprocessor would have many advantages over amultiple-coreapproach.Task execution on asing lecore is a well unders too the process, while executionon many cores is a problem that has not yet been solved. There a reals ocomputational tasks which parallelize very badly, where asinglehigh clock rate processor would suitthem very well. Such a high speed processor needs certain components that should support high speed.The two main components of processors are the ALU and the register file. The one of the critical path with in an ALU may be the carry-chainin addition operation.

INTRODUCTION

The saying goes that if you can count, you can control. Addition is a fundamental operation for any digital system, digital signal processing or control system. A fast and accurate operation of a digital system is greatly influenced by the performance of the resident adders. Adders are also very important component in digital systems because of their extensive use in other basic digital operations such as subtraction, multiplication and division. Hence, improving performance of the digital adder would greatly advance the execution of binary operations inside a circuit compromised of such blocks. The performance of a digital circuit block is gauged by analyzing its power dissipation, layout area and its operating speed. Comparing the performance metrics for the 16-bit adders implemented on Xilinx FPGA board, using Synopsys synthesis tools, the tradeoffs becomes apparent. As can be seen there exist an inverse relationship between time delays, operating speed, and circuit area, in this case the number of CLBs (measure of the area). The ripple carry adder, the most basic of flavors, is at the one extreme of this spectrum with the least amount of CLBs but the highest delay. The carry select adder on the other hand, is at the opposite corner since it has the lowest delay (half that of the ripple carry's) but with a larger area required to compensate for this time gain. Finally, the carry look-ahead is middle ground. Power dissipation, for this case study, is in direct proportion to the number of CLBs.

For more information on different adders, please see Appendix

DESIGN OBJECTIVE

To Design an optimized Gate level Logic for the following adders:

1. Ripple Carry Adder
2. Bit Serial Adder
3. Carry Look Ahead Adder
4. Carry Select Adder

To compare the above Adder Architectures on the basis of their performance in terms of Area, Timing and Power.

Benefits of Using Flow HDL. The flow HDL design software provides top down graphical front-end tools for easy analysis of complex designs in a comprehensive format. This uses flow Diagrams instead of hardware development language (HDL) or schematic representation. FlowHDL allows one to concentrate on the more abstract aspects of the design by correcting common errors. Through static checking and simulation, we can easily capture graphical design specifications.

It also increases productivity by reducing the time it takes to move from an initial idea to a testable design.

TYPES OF ADDER

In this lecture we will review the implementation technique of several types of adders and study their characteristics and performance. These are

- Ripple carry adder, or carry propagate adder,
- Carry look-ahead adder
- Carry skip adder,
- Manchester chain adder,
- Carry select adders
- Pre-Fix Adders
- Multi-operand adder
- Carry save Adder
- Pipelined parallel adder

For the same length of binary number, each of the above adders has different performance in terms of Delay, Area, and Power. All designs are assumed to be CMOS static circuits and they are viewed from architectural point of view.

BASIC ADDER UNIT

The most basic arithmetic operation is the addition of two binary digits, i.e. bits. A combinational circuit that adds two bits, according the scheme outlined below, is called a half adder. A full adder is one that adds three bits, the third produced from a previous addition operation. One way of implementing a full adder is to utilizes two half adders in its implementation. The full adder is the basic unit of addition employed in all the adders studied here

HALF ADDER

A half adder is used to add two binary digits together, A and B. It produces S, the sum of A and B, and the corresponding carry out Co. Although by itself, a half adder is not extremely useful, it can be used as a building block for larger adding circuits (FA).One possible implementation is using two AND gates, two inverters, and an OR gate instead of a XOR gate as shown in Fig. 1.

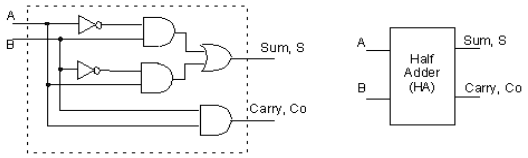


Figure.1: Half-Adder logic and block diagrams

A	B	S	C_O
O	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

Table:1 Half-Adder truth table

Boolean Equation:

S=A'B+AB'

C_O=AB

FULL ADDER

A full adder is a combinational circuit that performs the arithmetic sum of three bits: A, B and a carry in, C, from a previous addition, Fig. 2a. Also, as in the case of the half adder, the full adder produces the corresponding sum, S, and a carry out Co. As mentioned previously a full adder maybe designed by two half adders in series as shown below in Figure2b.The sum of A and B are fed to a second half adder, which then adds it to the carry in C (from a previous addition operation) to generate the final sum S. The carry out, Co, is the result of an OR operation taken from the carry outs of both half adders. There are a variety of adders in the literature both at the gate level and transistor level each giving different performances.

BOOLEN EQUATIONS AND FULL ADDER BLOCK DIAGRAM

$$S = C \oplus (A \oplus B)$$

$$C_o = AB + C(A \oplus B)$$

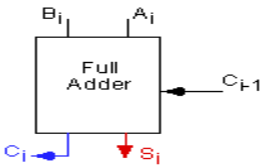


Table 2: FA Truth Table

A	B	C	S	Co
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

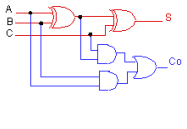


Figure 2a: Full adder

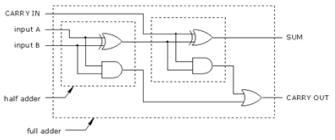


Figure 2b:Full adder constructed from Half Adders

PARALLEL ADDERS

Parallel adders are digital circuits that compute the addition of variable binary strings of equivalent or different size in parallel. The schematic diagram of a parallel adder is shown below in Fig. 3.

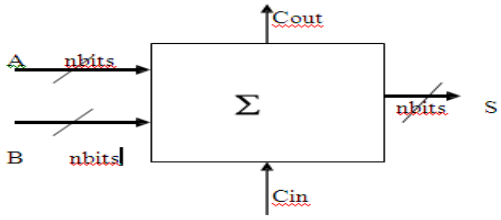


Fig. 3 Parallel Adder

FUNCTIONAL DESCRIPTION

RIPPLE CARRY ADDER

The ripple carry adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage. A number of full adders may be added to the ripple carry adder or ripple carry adders of different sizes may be cascaded in order to accommodate binary vector strings of larger sizes. For an n-bit parallel adder, it requires n computational elements (FA). Figure 4 shows an example of a parallel adder: a 4-bit ripple-carry adder. It is composed of four full adders. The augend‟s bits of x are added to the addend bits of y respectfully of their binary position. Each bit addition creates a sum and a carry out. The carry out is then transmitted to the carry in of the next higher-order bit. The final result creates a sum of four bits plus a carry out (c4).

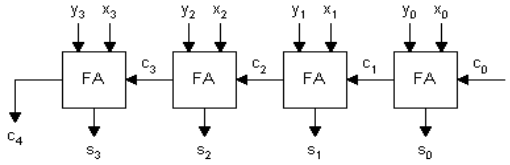


Figure 4: Parallel Adder: 4-bit Ripple-Carry Adder Block Diagram

Even though this is a simple adder and can be used to add unrestricted bit length numbers, it is however not very efficient when large bit numbers are used. One of the most serious drawbacks of this adder is that the delay increases linearly with the bit length. As mentioned before, each full adder has to wait for the carry out of the previous stage to output steady-state result. Therefore even if the adder has a value at its output terminal, it has to wait for the propagation of the carry before the output reaches a correct value as shown in Fig. 5.

Taking again the example in figure 4, the addition of x_4 and y_4 cannot reach steady state until c_4 becomes available. In turn, c_4 has to wait for c_3, and so on down to c_1. If one full adder takes T_{fa} seconds to complete its operation, the final result will reach its steady-state value only after $4.T_{fa}$ seconds. Its area is n A_{fa} A (very) small improvement in area consumption can be achieved if it is known in advance that the first carry in (c_0) will always be zero. (If so, the first full adder can be replace by a half adder). In general, assuming all gates have the same delay and area of NAND-2 then this circuit has 3n T_{gate} delay and $5nA_{gate}$. (One must be aware that in Static CMOs, this assumption is not true). Gate delays depend on intrinsic delay + f_{anin} delay+f_{anout} delay.

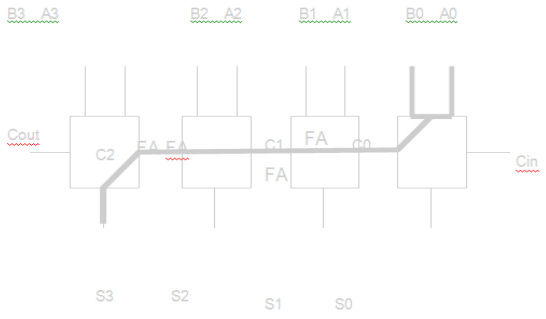


Figure 5: Carry Propagation in Carry Ripple Adder

Generally speaking, the worst-case delay of the RCA is when a carry signal transition ripples through all stages of adder chain from the least significant bit to the most significant bit, which is approximated by:

$$t=(n-1)t_c+t_s$$

where tc is the delay through the carry stage of a full adder, and ts is the delay to compute the sum of the last stage. The delay of ripple carry adder is linearly proportional to n, the number of bits, therefore the performance of the RCA is limited when n grows bigger. The advantages of the RCA are lower power consumption as well as a compact layout giving smaller chip area.

To design a larger adder ripple carry adders are cascaded. An example of 37 bit carry propagate adder is shown in Fig. 6

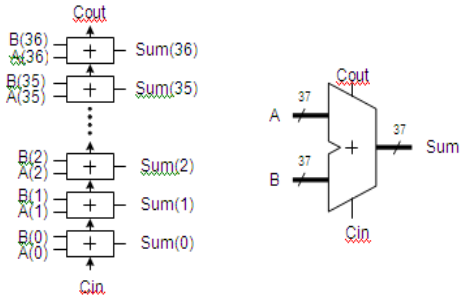


Figure 6: The structure and schematic diagram of a 37-bit Adder

CARRY SKIP ADDER

A carry-skip adder consists of a simple ripple carry-adder with a special speed up carry chain called a **skip chain**. This chain defines the distribution of ripple carry blocks, which compose the skip adder.

BOOLEAN EQUATIONS OF A FULL ADDER

$P_i = A_i \oplus B_i$ Equ. 1 --carry propagate of i^{th} stage

$S_i = P_i \oplus C_i$ Equ. 2 --sum of i^{th} stage

$C_{i+1} = A_iB_i + P_iC_i$ Equ. 3 --carry out of i^{th} stage

Supposing that $A_i = B_i$, then P_i in equation 1 would become zero (equation 4). This would make C_{i+1} to depend only on the inputs A_i and B_i, without needing to know the value of C_i.

$A_i = B_i \rightarrow P_i = 0$ Equ. 4 --from #Equation 1

If $A_i = B_i = 0 \rightarrow C_{i+1} = A_iB_i = 0$ --from equation 3

If $A_i = B_i = 1 \rightarrow C_{i+1} = A_iB_i = 1$ --from equation 3

Therefore, if Equation 4 is true then the carry out, C_{i+1}, will be one if $A_i = B_i = 1$ or zero if $A_i = B_i = 0$. Hence we can compute the carry out at any stage of the addition provided equation 4 holds. These findings would enable us to build an adder whose average time of computation would be proportional to the longest chains of zeros and of different digits of A and B. Alternatively, given two binary strings of numbers, such as the example below, it is very likely that we may encounter large chains of consecutive bits (Block 2) where A_i B_i. In order to deal with this scenario we must reanalyze equation 3 carefully.

$A_i \neq B_i \rightarrow P_i = 1$ Equ. 5 --from Equation 1

If $A_i \neq B_i \rightarrow C_{i+1} = C_i$ --from Equation 3

In the case of comparing two bits of opposite value, the carry out at that particular stage, will simply be equivalent to the carry in. Hence we can simply propagate the carry to the next stage without having to wait for the sum to be calculated.

TWO RANDOM BIT STRINGS

A	10100	01011	10100	01011
B	01101	10100	01010	01100
	block 3	block 2	block 1	block 0

In order to take advantage of the last property, we can design an adder that is divided into blocks, as shown in Fig. 7, where a special purpose circuit can compare the two binary strings inside each block and determine if they are equal or not. In the latter case the carry entering the block will simply be propagated to the next block and if this is the case all the carry inputs to the bit positions in that block are

all either 0"s or 1"s depending on the carry in into the block. Should only one pair of bits (A_i and B_i) inside a block be equal then the carry skip mechanism would be unable to skip the block. In the extreme case, although still likely, that there exist one such case, where $A_i = B_i$, in each block, then no block is skipped but a carry would be generated in each block instead.

CARRY SKIP CHAIN

In summary the **carry skip chain** mechanism (Figure 7) works as follows: Two strings of binary numbers to be added are divided into blocks of equal length. In each cell within a block both bits are compared for un-equivalence. This is done by Exclusive ORing each individual cell (parallel operation and already present in the full adder) producing a comparison

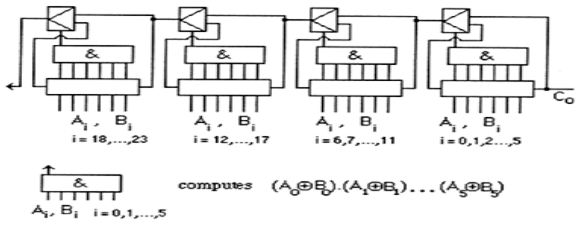
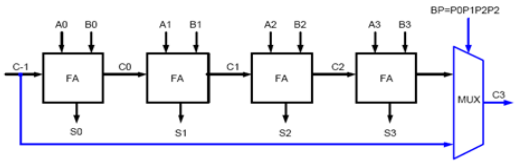


Figure 7: Carry skip chain mechanism

String. Next the comparison string is ANDed within itself in a domino fashion. This process ensures that the comparison of each and all cells was indeed unequal and we can therefore proceed to propagate the carry to the next block. A MUX is responsible for selecting a **generated carry** or a **propagated (**previous) **carry** with its selection line being the output of the comparison circuit just described. If for each cell in the block $A_i \neq B_i$ then we say that a carry can skip over the block otherwise if $A_i = B_i$ we shall say that the carry must be generated in the block.

When studying carry skip adders the main purpose is to find a configuration of blocks that minimizes the longest life of a carry, i.e. from the time of its generation to the time of the generation of the next carry Many models have been suggested: the first with blocks of equal size and the second with blocks of different sizes according to some heuristic.



CARRY BYPASS CIRCUIT ARCHITECTURE

The delay of n-bit adder based on m-bit blocks of Carry Bypass Adder, CBA rippled together can be given by:

$t=t_{setup}+mt_{carry}+(n/m-1)\ t_{carry}+t_{sum}$

where tc is the delay through the carry stage of a full adder, and ts is the delay to compute the sum of the last stage. The delay of ripple carry adder is linearly proportional to n, the number of bits, therefore the performance of the RCA is limited when n grows bigger. The advantages of the RCA are lower power consumption as well as a compact layout giving smaller chip area.

To design a larger adder ripple carry adders are cascaded. An example of 37 bit carry propagate adder is shown in Fig. 6

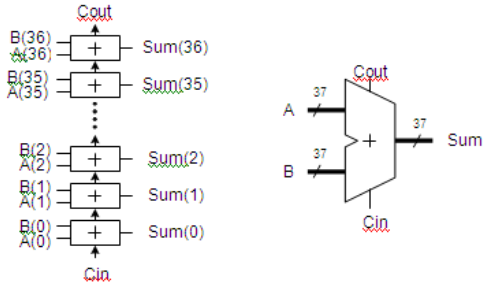


Figure 6: The structure and schematic diagram of a 37-bit Adder

CARRY SKIP ADDER

A carry-skip adder consists of a simple ripple carry-adder with a special speed up carry chain called a **skip chain**. This chain defines the distribution of ripple carry blocks, which compose the skip adder.

Carry Skip Mechanics

The addition of two binary digits at stage i, where i ≠ 0, of the ripple carry adder depends on the carry in, C_i , which in reality is the carry out, C_{i-1}, of the previous stage. Therefore, in order to calculate the sum and the carry out, C_{i+1} , of stage i, it is imperative that the carry in, C_i, be

known in advance. It is interesting to note that in some cases C_{i+1} can be calculated without knowledge of C_i.

BOOLEAN EQUATIONS OF A FULL ADDER

$P_i = A_i \oplus B_i$ Equ. 1 --carry propagate of i^{th} stage
$S_i = P_i \oplus C_i$ Equ. 2 --sum of i^{th} stage
$C_{i+1} = A_iB_i + P_iC_i$ Equ. 3 --carry out of i^{th} stage

Supposing that $A_i = B_i$, then P_i in equation 1 would become zero (equation 4). This would make C_{i+1} to depend only on the inputs A_i and B_i, without needing to know the value of C_i.

$A_i = B_i \rightarrow P_i = 0$ Equ. 4 --from #Equation

If $A_i = B_i = 0 \rightarrow C_{i+1} = A_iB_i = 0$ --from equation 3
If $A_i = B_i = 1 \rightarrow C_{i+1} = A_iB_i = 1$ --from equation 3

Therefore, if Equation 4 is true then the carry out, C_{i+1}, will be one if $A_i = B_i = 1$ or zero if $A_i = B_i = 0$. Hence we can compute the carry out at any stage of the addition provided equation 4 holds. These findings would enable us to build an adder whose average time of computation would be proportional to the longest chains of zeros and of different digits of A and B. Alternatively, given two binary strings of numbers, such as the example below, it is very likely that we may encounter large chains of consecutive bits (Block 2) where A_i B_i. In order to deal with this scenario we must reanalyze equation 3 carefully.

$A_i \neq B_i \rightarrow P_i = 1$ Equ. 5 --from Equation 1

If $A_i \neq B_i \rightarrow C_{i+1} = C_i$ --from Equation 3

In the case of comparing two bits of opposite value, the carry out at that particular stage, will simply be equivalent to the carry in. Hence we can simply propagate the carry to the next stage without having to wait for the sum to be calculated.

TWO RANDOM BIT STRINGS

A	10100	01011	10100	01011
B	01101	10100	01010	01100
	block 3	block 2	block 1	block 0

In order to take advantage of the last property, we can design an adder that is divided into blocks, as shown in Fig. 7, where a special purpose circuit can compare the two binary strings inside each block and determine if they are equal or not. In the latter case the carry entering the block will simply be propagated to the next block and if this is the case all the carry inputs to the bit positions in that block are all either 0"s or 1"s depending on the carry in into the block. Should only one pair of bits (A_i and B_i) inside a block be equal then the carry skip mechanism would be unable to skip the block. In the extreme case, although still likely, that there exist one such case, where $A_i = B_i$, in each block, then no block is skipped but a carry would be generated in each block instead.

CARRY SKIP CHAIN

In summary the **carry skip chain** mechanism (Figure 7) works as follows:

Two strings of binary numbers to be added are divided into blocks of equal length. In each cell within a block both bits are compared for un-equivalence. This is done by Exclusive ORing each individual cell (parallel operation and already present in the full adder) producing a comparison String.

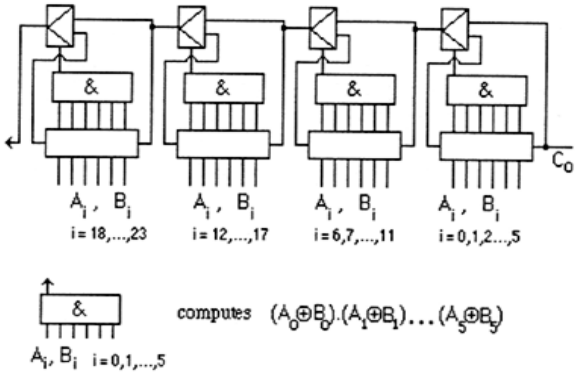
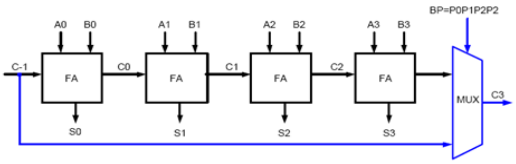


Figure 7: Carry skip chain mechanism

Next the comparison string is ANDed within itself in a domino fashion. This process ensures that the comparison of each and all cells was indeed unequal and we can therefore proceed to propagate the carry to the next block. A MUX is responsible for selecting a **generated carry** or a **propagated (**previous) **carry** with its selection line being the output of the comparison circuit just described. If for each cell in the block $A_i \neq B_i$ then we say that a carry can skip over the block otherwise if $A_i = B_i$ we shall say that the carry must be generated in the block. When studying carry skip adders the main purpose is to find a configuration of blocks that minimizes the longest life of a carry, i.e. from the time of its generation to the time of the generation of the next carry. Many models have been suggested: the first with blocks of equal size and the second with blocks of different sizes according to some heuristic.



CARRY BYPASS CIRCUIT ARCHITECTURE

The delay of n-bit adder based on m-bit blocks of Carry Bypass Adder, CBA rippled together can be given by:

$t=t_{setup}+mt_{carry}+(n/m-1)\ t_{carry}+t_{sum}$ (7)

n is the adder length and m is the length of the blocks Comparing to the RCA, the CBA has slightly improved speed for wider-bit adders (still linear to n), but with higher active capacitance and the area overhead because of the extra bypass circuit.

THE ADDER

A Manchester carry adder consists of cascaded stages of Manchester propagation cells, shown above. The optimum amount of cascaded stages may be calculated for a technology by simulation. For a 16 bit adder example a 4-bit adder made up of four static stage cells, shown in figure 9, is chosen in order to reduce the number of series-propagate transistors, which greatly improves speed. In the case of a four-bit adder, the maximum number of transistors that are in series with the gate, when all propagate signals and Ci is true, is only five.

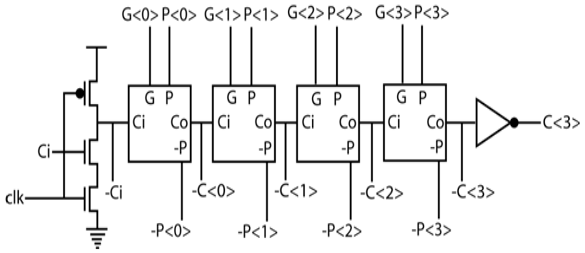


Fig 9: 4 BIT MANCESTER CARRY SECTION

In addition to the cascaded Manchester propagation cells the adder requires carry propagation and carry generation logic, also called a PG generator shown in Figure 10. Finally to complete the design four XNOR blocks each of which produces the SUM at each particular stage is required.

To further reduce the worst-case propagation time of the Manchester carry adder in the case where Ai $\neq$ B i, for all i, an additional bypass circuit is introduced in order to bypass the four stages. The circuit is illustrated in Figure 11.

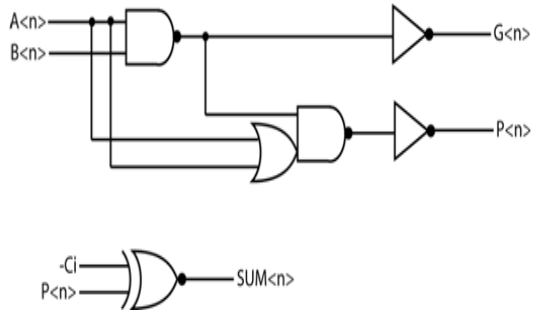


FIGURE10. PG LOGIC AND SUM LOGIC

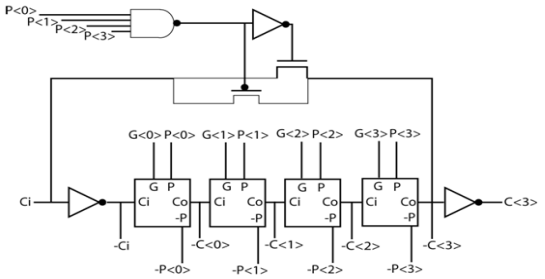


Fig11. MANCESTER CARRY ADDER WITH CARRY BEPASS

Other Manchester adders' implementations are possible. One such adder is based on MUXes called a conflict free Manchester Adder. Although this version reduces even further the propagation time of the adder, it still embodies the core of a Manchester adder whose ultimate goal is to achieve the reduction of the worst-case time propagation by employing a Manchester cell. A Manchester Adder can be constructed by designing a cell and cascading it as shown in the Figures 12.

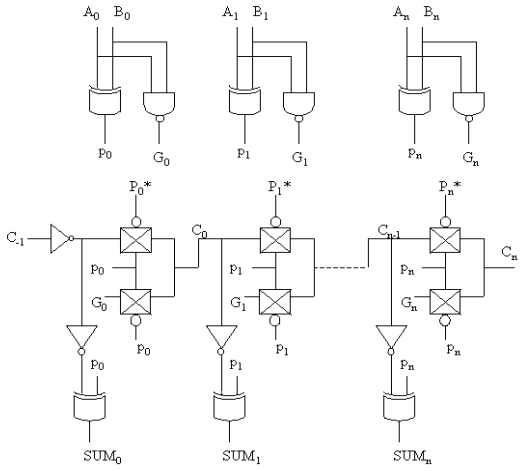
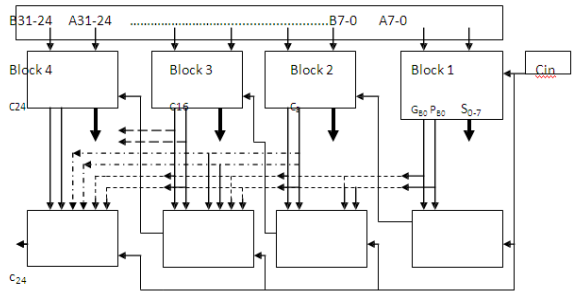


Fig 12. THE CONFIGURATION OF THE MANCESTER ADDER/SUBTRACTOR

LARGE ADDER DESIGN

Large adders require a special design. Most standard adders are modified in a way or other to be able to use them for larger designs. For example Carry Look Ahead adders are modified as hierarchical 2 level circuits. This is because as n increases, the block size has to be limited as

well as ripple through delay accumulates. It is no longer practical to use standard look-ahead method. The hierarchical CLA has two levels. In this design, the first level of CLAs generates the sums as well as the second level ,,generate and propagate signals. These signals then are fedto the 2nd level CLA with carryout of each level to produce the carryout signal. Each Block CLA has a special design. For more details one can refer to: Assume that you want to design a 32 bit CLA adder. One way is to divide the adder into four 8- bit CLA with carry ripple between them. Other method would be to design a 2- level hierarchical adder as shown below.



In the above diagram

PB0= P7P6P5P4P3P2P1P0

And

GBo=g7+p7g6+p7P6G5+……P7P6P5P4P3P2P1G0

Other carrys then can be obtained using CLA methodology as

c8 = GB0 + PB0 cin

c16 =GB1 + PB1 c8

c24= GB2 + PB2 c16

c32 = GB3 + PB3 c24

Another method is to use a Block CLA, without going into details an example a large 53 bit CLA is shown in Fig 14.

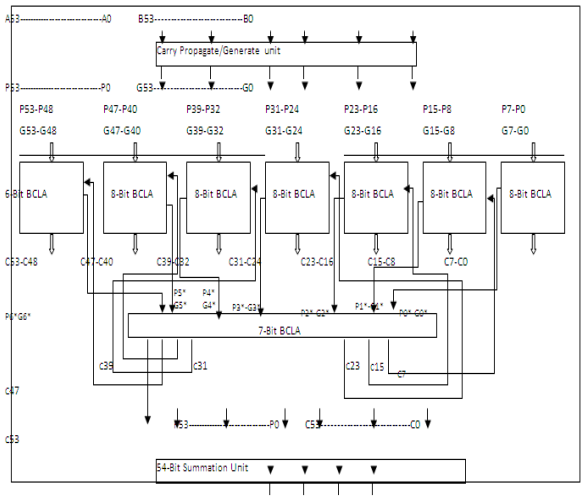


FIGURE14. A 53 BIT CARRY LOCK AHEAD ADDER

PERFORMANCE STATIISTICS

3.1 Ripple Carry Adder:

RCA	AREA (μm2)	Data Arrival Time	POWER
8 BIT	50	4.19	41.02
16 BIT	98	5.5	76.77
32 BIT	194	5.5	150.4
64 BIT	386	5.5	297.64

3.2 Carry Look Ahead Adder:

CLA	AREA (μm2)	Data Arrival Time	POWER
8 BIT	87	13.67	103.83
16 BIT	165	22.26	201.12
32 BIT	315	37.99	381.09
64 BIT	594	53.35	704.51

3.3 Carry Select Adder:

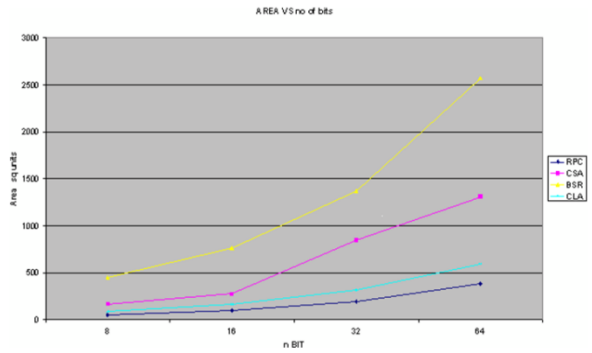
CARRYSEL	AREA (µm2)	Data Arrival Time	POWER
8 BIT	166	14.75	231.7
16 BIT	276	23.26	369.61
32 BIT	846	36.01	990.37
64 BIT	1307	56.21	1971

3.4 Bit Serial Adder:

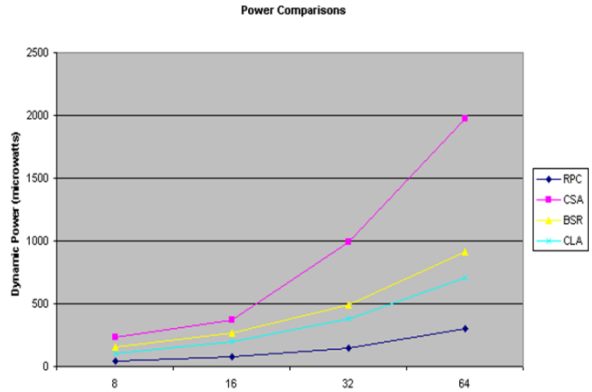
BIT SERIAL	AREA (µm2)	Data Arrival Time	POWER
8 BIT	448	1.37	151.38
16 BIT	763	1.37	269.75
32 BIT	1369	1.37	487.95
64 BIT	2569	1.37	916.69

COMPARATIVE RESULTS

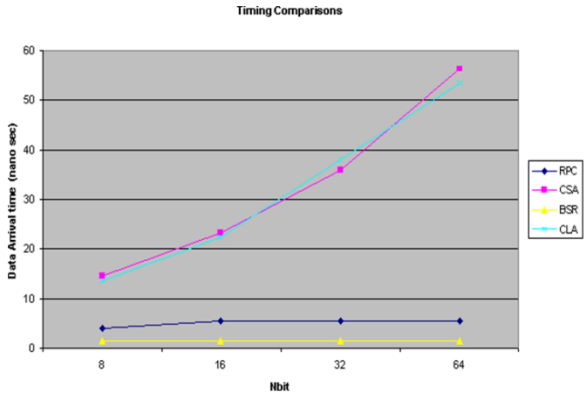
AREA COMPARISON CHART



POWER CONSUMPTION COMPARISON CHART



DATA ARRIVAL TIME COMPARISON CHART



CONCLUSION

Implemented the basic binary architectures of the following adders:

1. Bit Serial Adder,
2. Ripple Carry Adder,
3. Carry Look Ahead Adder, and
4. Carry Select Adder.

We have implemented these adders for 8, 16, 32 and 64 bits and analyzed their performance in terms of Area, Power and Timing requirements. This study enabled us to select a particular type of Adder for optimum performance.

The type of adder to be selected depends on three factors:

1. Area of the layout that influences the cost.

2.Timing and power that influences the performance of the adder. So, the selecting an adder is a trade off between all the above factors. As area increases the speed of the circuit also increases, resulting in high costs. So, there should be a kind of balance between these factors.

In our implementation of adders we prefer the carry select adder because of its speed. However it occupies larger area when compared to other architectures.

REFRENCES

[1] Stefan Sjoholm and Lennart Lind, VHDL for designers, Prentice Hall, 1997

[2] Vitit Kantabutra, "Designing optimum One-Level Carry-Skip Adders" IEEE
Transactions on Computers, Vol.42, No.6, June 1993

[3] Luigi Dadda and Vincenzo Piuri, "Pipelined adders" IEEE Transactions on
Computers, Vol.45, No.3, March 1996

[4] M. Morris Mano, Digital Design second edition, Prentice Hall, 1991

[5] Carver Mead and Lynn Conway, Introduction to VLSI design, Addison-Wesley
Company, 1980

[6] Jien-Chung Lo, "A fast binary adder with conditional carry generation" IEEE Transactions on Computers, Vol.46, No.2, February 1997

[7] N.H.E. Weste and K. Eshraghian, Principle of CMOS VLSI Design, Addison- Wesley Company, 1992

[8] Peter Pirsch, Architectures for digital signal processing, John Wiley & Sons,
1998

[9] A. Guyot, B. Hochet and J.M. Muller, "A way to build efficient Carry-Skip
adders", IEEE Transactions on Computers, pp.1144-1152, October 1987

[10] S. Brown, Z. Verasenic, "Fundamentals of Digital Logic with VHDL," Mc. Graw
Hill, 2nd edition, 2004.