International Journal of Research
Available at https://edupediapublications.org/journals

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

Energy-Efficient Reconfigurable Approximate Carry Look-Ahead Adder (Rap-Cla) Using Xilinx

Dalu.Rakesh & Sk.Fairoze
[1]Pg Scholar , [2] Assistant Professor

Department Of Electronics & Communication Engineering

Sri Vani Educational Society Group Of Institutions Chevuturu, Krishna (Dt)

ABSTRACT

In this paper, we propose a fast yet energy-efficient reconfigurable approximate carry look-ahead adder (RAP-CLA). This adder has the ability of switching between the approximate and exact operating modes making it suitable for both error-resilient and exact applications. The structure, which is more area and power efficient than state-of-the-art reconfigurable approximate adders, is achieved by some modifications to the conventional carry look ahead adder (CLA). The efficacy of the proposed RAP-CLA adder is evaluated by comparing its characteristics to those of two state-of-the-art reconfigurable approximate adders as well as the conventional (exact) CLA. The results reveal that, in the approximate operating mode, the proposed 32-bit adder provides better delay and power reductions compared to those of the exact CLA, respectively, at the cost of low error rate. It also provides lower delay and power consumption, respectively, compared to other approximate adders considered in this work. Finally, the effectiveness of the proposed adder on two image processing applications of smoothing and sharpening is demonstrated. The proposed architecture of this paper analysis the delay and area using Xilinx 14.3.

I. .INTRODUCTION

The challenge of the verifying a large design is growing exponentially. There is a need to define new methods that makes functional verification easy. Several strategies in the recent years have been proposed to achieve good functional verification with less effort. Recent advancement towards this goal is methodologies. The methodology defines a skeleton over which one can add flesh and skin to their requirements to achieve functional verification.

The report is organized as two major portions; first part is brief introduction and history of the functional verification of regular Carry select adder which tells about different advantages Carry select adder and RCA architecture and in this Regular Ckt one drawback is there overcome that complexicity problem we go for modified architecture of CSLA .

Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA. Based on this modification 8-, 16- CSLA architecture have been developed and compared with the regular CSLA architecture. The proposed design has reduced area and power as compared with the regular CSLA with only a slight increase in the delay.

This work evaluates the performance of the proposed designs in terms of delay, area, power, and their products by hand with logical effort and through custom design and layout in Xilinx XC series devices CMOS process technology. The results analysis shows that the proposed

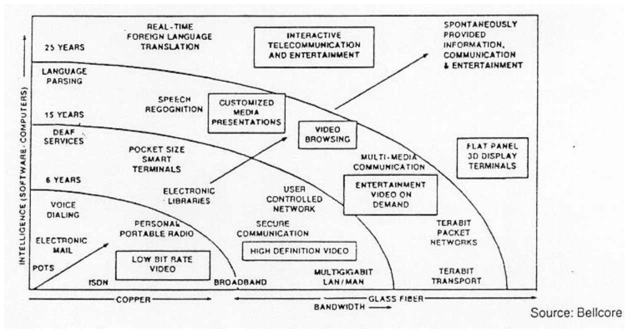
Second part is Design and verification of the Architecture of CSLA circuits. Architecture of the test bench gives complete description about the components and sub components used to achieve the verification goals and also explain about improvements made in the design of the usb-i2c bridge, test plan identifies all the test case required to meet the goals and finally results of the project

Historical Perspective

The electronics industry has achieved a phenomenal growth over the last two decades, mainly due to the rapid advances in integration technologies, large-scale systems design - in short, due to the advent of VLSI. The number of applications of integrated circuits in high-performance computing, telecommunications, and consumer electronics has been rising steadily, and at a very fast pace. Typically, the required computational power (or, in other words, the intelligence) of these applications is the driving force for the fast development of this field. Figure 1.1 gives an overview of the prominent trends in information technologies over the next few

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

decades. The current leading-edge technologies (such as low bit-rate video and cellular communications) already provide the end-users a certain amount of processing power and portability.



This trend is expected to continue, with very important implications on VLSI and systems design. One of the most important characteristics of information services is their increasing need for very high processing power and bandwidth (in order to handle real-time video, for example). The other important characteristic is that the information services tend to become more and more personalized (as opposed to collective services such as broadcasting), which means that the devices must be more intelligent to answer individual demands, and at the same time they must be portable to allow more flexibility/mobility

As more and more complex functions are required in various data processing and telecommunications devices, the need to integrate these functions in a small system/package is also increasing. The level of integration as measured by the number of logic gates in a monolithic chip has been steadily rising for almost three decades, mainly due to the rapid progress in processing technology and interconnect technology. Table 1.1 shows the evolution of logic complexity in integrated circuits over the last three decades, and marks the milestones of each era. Here, the numbers for circuit complexity should be interpreted only as representative examples to show the order-of-magnitude. A logic block can contain anywhere from 10 to 100 transistors, depending on the function. State-of-the-art examples of ULSI chips, such as the DEC Alpha or the INTEL Pentium contain 3 to 6 million transistors.

The most important message here is that the logic complexity per chip has been (and still is) increasing exponentially. The monolithic integration of a large number of functions on a single chip usually provides:

- Less area/volume and therefore, compactness
- Less power consumption
- Less testing requirements at system level
- Higher reliability, mainly due to improved on-chip interconnects
- Higher speed, due to significantly reduced interconnection length
- Significant cost savings

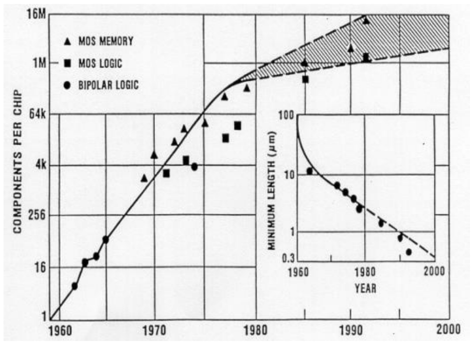


Figure-1. Evolution of integration density and minimum feature size, as seen in the early 1980s.

Therefore, the current trend of integration will also continue in the foreseeable future. Advances in device manufacturing technology, and especially the steady reduction of minimum feature size (minimum length of a transistor or an interconnect realizable on chip) support this trend. Figure 1.2 shows the history and forecast of chip complexity - and minimum feature size - over time, as seen in the early 1980s. At that time, a minimum feature size of 0.3 microns was expected around the year 2000. The actual development of the technology, however, has far exceeded these expectations. A minimum size of 0.25 microns was readily achievable by the year 1995. As a direct result of this, the integration density has also exceeded previous expectations - the first 64 Mbit DRAM, and the INTEL Pentium microprocessor chip containing more than 3 million transistors were already available by 1994, pushing the envelope of integration density.

When comparing the integration density of integrated circuits, a clear distinction must be made between the memory chips and logic chips. Figure 1.3 shows the level of integration over time for memory and logic chips, starting in 1970. It can be observed that in terms of transistor count, logic chips contain significantly fewer transistors in any given year mainly due to large consumption of chip area for complex interconnects. Memory circuits are highly regular and thus more cells can be integrated with much less area for interconnects.

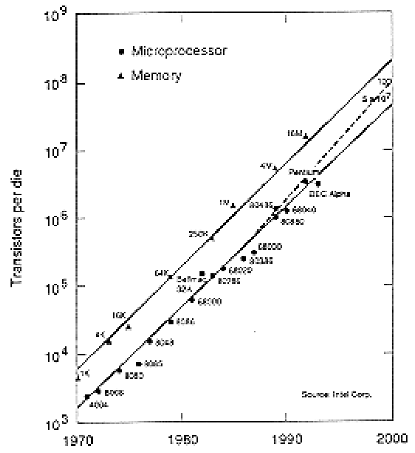


Figure-2. Level of integration over time, for memory chips and logic chips.

Generally speaking, logic chips such as microprocessor chips and digital signal processing (DSP) chips contain not only large arrays of memory (SRAM) cells, but also many different functional units. As a result, their design complexity is considered much higher than that of memory chips, although advanced memory chips contain some sophisticated logic functions. The design complexity of logic chips increases almost exponentially with the number of transistors to be integrated. This is translated into the increase in the design cycle time, which is the time period from the start of the chip development until the mask-tape delivery time. However, in order to make the best use of the current technology, the chip development time has to be short enough to allow the maturing of chip manufacturing and timely delivery to customers. As a result, the level of actual logic integration tends to fall short of the integration level achievable with the current processing technology. Sophisticated computer-aided design (CAD) tools and methodologies are developed and applied in order to manage the rapidly increasing design complexity.

VLSI Design Flow

The design process, at various levels, is usually evolutionary in nature. It starts with a given set of requirements. Initial design is developed and tested against the requirements. When requirements are not met, the design has to be improved. If such improvement is either not possible or too costly, then the revision of requirements and its impact analysis must be considered. The Y-chart (first introduced by D. Gajski) shown in Fig. 1.4 illustrates a design flow for most logic chips, using design activities on three different axes (domains) which resemble the letter Y.

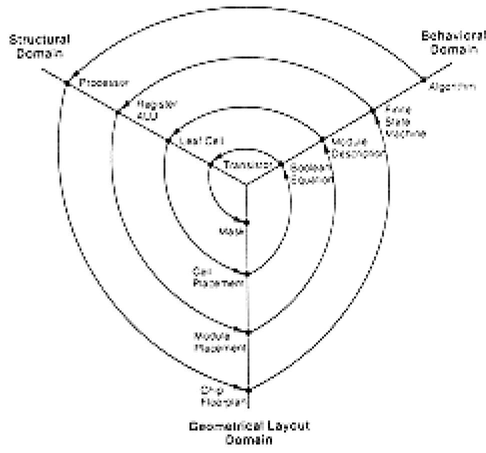


Figure3: Typical VLSI design flow in three domains (Y-chart representation).

The Y-chart consists of three major domains, namely:

- behavioral domain,
- structural domain,
- geometrical layout domain.

The design flow starts from the algorithm that describes the behavior of the target chip. The corresponding architecture of the processor is first defined. It is mapped onto the chip surface by floorplanning. The next design evolution in the behavioral domain defines finite state machines (FSMs) which are structurally implemented with functional modules such as registers and arithmetic logic units (ALUs).

These modules are then geometrically placed onto the chip surface using CAD tools for automatic module placement followed by routing, with a goal of minimizing the interconnects area and signal delays. The third evolution starts with a behavioral module description. Individual modules are then implemented with leaf cells. At this stage the chip is described in terms of logic gates (leaf cells), which can be placed and interconnected by using a cell placement & routing program. The last evolution involves a detailed Boolean description of leaf cells followed by a transistor level implementation of leaf cells and mask generation. In standard-cell based design, leaf cells are already pre-designed and stored in a library for logic design use.

provides a more simplified view of the VLSI design flow, taking into account the various representations, or abstractions of design - behavioral, logic, circuit and mask layout. Note that the verification of design plays a very important role in every step during this process. The failure to properly verify a design in its early phases typically causes significant and expensive re-design at a later stage, which ultimately increases the time-to-market.

Although the design process has been described in linear fashion for simplicity, in reality there are many iterations back and forth, especially between any two neighboring steps, and occasionally even remotely separated pairs. Although top-down design flow provides an excellent design process control, in reality, there is no truly unidirectional top-down design flow. Both top-down and bottom-up approaches have to be combined. For instance, if a chip designer defined an architecture without close estimation of the corresponding chip area, then it is very likely that the resulting chip layout exceeds the area limit of the available technology. In such a case, in order to fit the architecture into the allowable chip area, some functions may have to be removed and the design process must be repeated. Such changes may require significant modification of the original requirements. Thus, it is very important to feed forward low-level information to higher levels (bottom up) as early as possible.

In the following, we will examine design methodologies and structured approaches which have been developed over the years to deal with both complex hardware and software projects. Regardless of the actual size of the project, the basic principles of structured design will improve the prospects of success. Some of the classical techniques for reducing the complexity of IC design are: Hierarchy, regularity, modularity and locality.

II. LITERATURE REVIEW

The electronics of a general biomedical device consist of energy delivery, analog-to-digital conversion, signal processing, and communication subsystems. Each of these blocks must be designed for minimum energy consumption. Specific design techniques, such as aggressive voltage scaling, dynamic power-performance management, and energy-efficient signaling, must be employed to adhere to the stringent energy constraint. The constraint itself is set by the energy source, so energy harvesting holds tremendous promise toward enabling sophisticated systems without straining user lifestyle. Further, once harvested, efficient delivery of the low-energy levels, as well as robust operation in the aggressive low-power modes, requires careful understanding and treatment of the specific design limitations that dominate this realm. We outline the performance and power constraints of biomedical devices, and present circuit techniques to achieve complete systems operating down to power levels of microwatts. In all cases, approaches that leverage advanced technology trends are emphasized.

Power dissipation is one of the most important design objectives in integrated circuits, after speed. As adders are the most widely used components in such circuits, design of efficient adder is of much concern for researchers. This paper presents performance analysis of different Fast Adders. The comparison is done on the basis of three performance parameters i.e. Area, Speed and Power consumption. We present a modified carry select adder designed in different stages. Results obtained from modified carry select adders are better in area and power consumption.

Addition is the heart of arithmetic unit and the arithmetic unit is often the work horse of a computational circuit. So adders play a key role in designing an arithmetic unit and also many digital integrated circuits. Carry Select Adder (CSLA) is one of the fastest adders used in many data processors and in digital circuits to perform arithmetic operations. But CSLA is area-consuming because it consists of dual ripple carry adder (RCA) in the structure. To reduce the area of CSLA, a CSLA with Binary to Excess-1 Converter is already designed which reduces the area of adder. But there are other techniques to design a CSLA to reduce its area. One of such technique is using an add one circuit technique. This paper proposes the design of square root CSLA (SQRT CSLA) using add one circuit with significant reduction in area. The proposed design is synthesized using Leonardo Spectrum to get area (number of gates) and delay (ns). The performance in terms of area and delay are evaluated for square root CSLA using add one circuit and are compared with existing SQRT CSLA and SQRT CSLA using Binary to Excess-1 Converter (BEC). The results analysis shows that the proposed SQRT CSLA using add one circuit is better than the existing SQRT CSLA and SQRT CSLA using BEC.

III. EXISTING SYSTEM

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit(s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar.

Although adders can be constructed for many numerical representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder–subtractor. Other signed number representations require a more complex adder.

Half adder

The **half adder** adds two one-bit binary numbers *A* and *B*. It has two outputs, *S* and *C* (the value theoretically carried on to the next addition); the final sum is 2*C* + *S*. The simplest half-adder design, pictured on the right, incorporates an XOR gate for *S* and an AND gate for *C*. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder

A **full adder** adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as *A*, *B*, and C_{in}; *A* and *B* are the operands, and C_{in} is a bit carried in from the next less significant stage. The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates. One example implementation is with and

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)).$$

In this implementation, the final OR gate before the carry-out output may be replaced by an XOR gate without altering the resulting logic. Using only two types of gates is convenient if the circuit is being implemented using simple IC chips which contain only one gate type per chip. In this light, C_{out} can be implemented as

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)).$$

A full adder can be constructed from two half adders by connecting *A* and *B* to the input of one half adder, connecting the sum from that to an input to the second adder, connecting C_i to the other input and OR the two carry outputs. Equivalently, *S* could be made the three-bit XOR of *A*, *B*, and C_i, and C_{out} could be made the three-bit majority function of *A*, *B*, and C_i.

Ripple carry adder

It is possible to create a logical circuit using multiple full adders to add *N*-bit numbers. Each full adder inputs a C_{in}, which is the C_{out} of the previous adder. This kind of adder is a *ripple carry adder*, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

The layout of a ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Each full adder requires three levels of logic. In a 32-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is 3 (from input to carry in first adder) + 31 * 2 (for carry propagation in later adders) = 65 gate delays. A design with alternating carry polarities and optimized AND-OR-Invert gates can be about twice as fas

Carry-lookahead adders

To reduce the computation time, engineers devised faster ways to add two binary numbers by using carry-lookahead adders. They work by creating two signals (*P* and *G*) for each bit position, based on if a carry is propagated through from a less significant bit position (at least one input is a '1'), a carry is generated in that bit position (both inputs are '1'), or if a carry is killed in that bit position (both inputs are '0'). In most cases, *P* is simply the sum output of a half-adder and *G* is the carry output of the same adder. After *P* and *G* are generated the carries for every bit position are created. Some advanced carry-lookahead architectures are the Manchester carry chain, Brent–Kung adder, and the Kogge–Stone adder.

Some other multi-bit adder architectures break the adder into blocks. It is possible to vary the length of these blocks based on the propagation delay of the circuits to optimize computation time. These block based adders include the carry bypass adder which will determine *P* and *G* values for each block rather than each bit, and the carry select adder which pre-generates sum and carry values for either possible carry input to the block.

IV. PROPOSED SYSTEM

RECONFIGURABLE APPROXIMATE CLA

Supposing that we have two bits of storage per digit, we can use a redundant binary representation, storing the values 0, 1, 2, or 3 in each digit position. It is therefore obvious that one more binary number can be added to our carry-save result without overflowing our storage capacity: but then what?

The key to success is that at the moment of each partial addition we add three bits:

- 0 or 1, from the number we are adding.
- 0 if the digit in our store is 0 or 2, or 1 if it is 1 or 3.
- 0 if the digit to its right is 0 or 1, or 1 if it is 2 or 3.

To put it another way, we are taking a carry digit from the position on our right, and passing a carry digit to the left, just as in conventional addition; but the carry digit we pass to the left is the result of the *previous* calculation and not the current one. In each clock cycle, carries only have to move one step along, and not *n* steps as in conventional addition.

Because signals don't have to move as far, the clock can tick much faster.

There is still a need to convert the result to binary at the end of a calculation, which effectively just means letting the carries travel all the way through the number just as in a conventional adder. But if we have done 512 additions in the process of performing a 512-bit multiplication, the cost of that final conversion is effectively split across those 512 additions, so each addition bears 1/512 of the cost of that final "conventional" addition.

Drawbacks

At each stage of a carry-save addition,

1. We know the result of the addition at once.
2. We *still do not know* whether the result of the addition is larger or smaller than a given number (for instance, we do not know whether it is positive or negative).

This latter point is a drawback when using carry-save adders to implement modular multiplication (multiplication followed by division, keeping the remainder only). If we cannot know whether the intermediate result is greater or less than the modulus, how can we know whether to subtract the modulus or not?

Montgomery multiplication, which depends on the rightmost digit of the result, is one solution; though rather like carry-save addition itself, it carries a fixed overhead, so that a sequence of Montgomery multiplications saves time but a single one does not. Fortunately exponentiation, which is effectively a sequence of multiplications, is the most common operation in public-key cryptography.

Carry Select Adder:

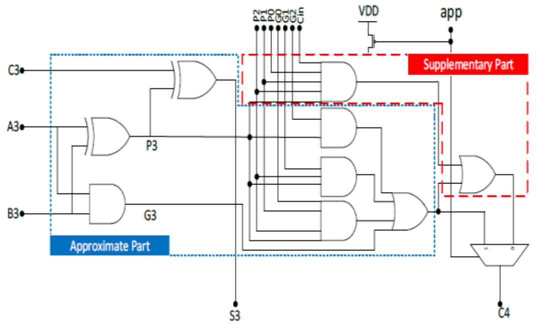
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

In electronics, a **carry-select adder** is a particular way to implement an adder, which is a logic element that computes the $(n+1)$-bit sum of two n-bit numbers. The carry-select adder is simple but rather fast, having a gate level depth of $O(\sqrt{n})$.

The carry-select adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known.

The number of bits in each carry select block can be uniform, or variable. In the uniform case, the optimal delay occurs for a block size of $\lfloor\sqrt{n}\rfloor$. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The $O(\sqrt{n})$ delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays.

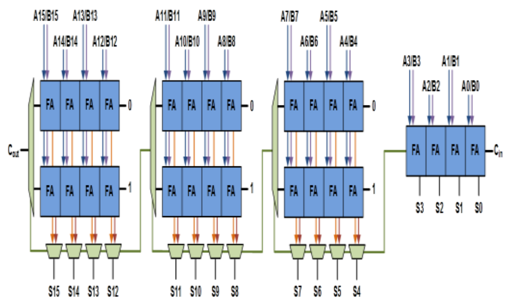
Basic building block



Above is the basic building block of a carry-select adder, where the block size is 4. Two 4-bit ripple carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result.

Uniform-sized adder

A 16-bit carry-select adder with a uniform block size of 4 can be created with three of these blocks and a 4-bit ripple carry adder. Since carry-in is known at the beginning of computation, a carry select block is not needed for the first four bits. The delay of this adder will be four full adder delays, plus three MUX delays.



MODIFIED 16-B SQRT CSLA

The structure of the proposed 16-b SQRT CSLA using BEC for RCAwith Cin =1 to optimize the area and power is shown in Fig. 6. We again split the structure into five groups. The delay and area estimation of each group are shown in Fig. . The steps leading to the evaluation are given here.

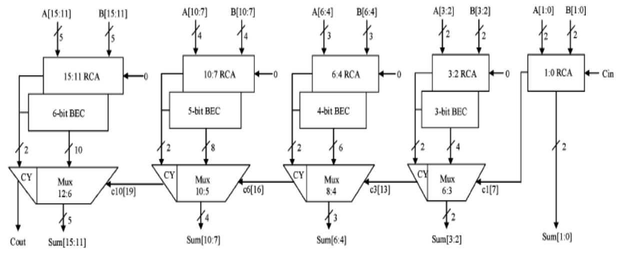
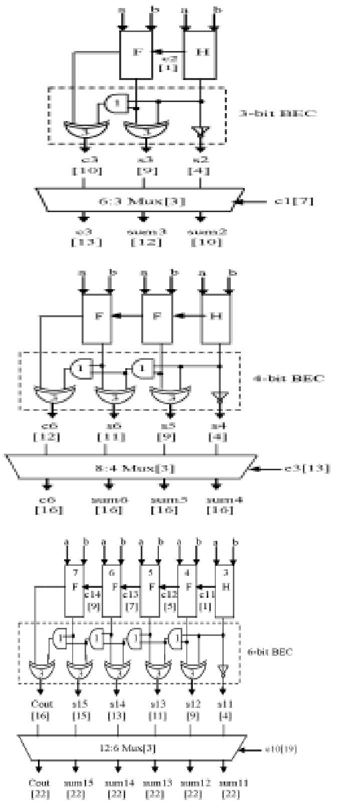


TABLE III
DELAY AND AREA COUNT OF REGULAR SQRT CSLA GROUPS

Group	Delay	Area
Group2	11	57
Group3	13	87
Group4	16	117
Group5	19	147



1) The group2 [see Fig. 7(a)] has one 2-b RCA which has 1 FA and 1 HA for Cin=0. Instead of another 2-b RCA with Cin=1 a 3-b BEC is used which adds one to the output from 2-b RCA. Based on the consideration of delay values of Table I, the arrival time of selection input c1[time(t)=7] of 6:3 mux is earlier than the s3[t=9] and c3[t=10] and later than the s2[t=4]. Thus,
the sum3 and final c3 (output from mux) are depending on s3 and mux and partial c3 (input to mux) and mux, respectively. The sum2 depends on c1 and mux.

2) For the remaining group's the arrival time of mux selection input is always greater than the arrival time of data inputs from the BEC's. Thus, the delay of the remaining groups depends on the arrival time of mux selection input and the mux delay.
The area count of group2 is determined as follows

$$\begin{aligned}
&\text{Gate count} = 43\ (\text{FA} + \text{HA} + \text{Mux} + \text{BEC})\\
&\text{FA} = 13(1*13)\\
&\text{HA} = 6(1*6)\\
&\text{AND} = 1\\
&\text{NOT} = 1\\
&\text{XOR} = 10(2*5)\\
&\text{Mux} = 12(3*4).
\end{aligned}$$

3) Similarly, the estimated maximum delay and area of the other groups of the modified SQRT CSLA are evaluated and listed in Table IV.

Comparing Tables III and IV, it is clear that the proposed modified SQRT CSLA saves 113 gate areas than the regular SQRT CSLA, with only 11 increases in gate delays. To further evaluate the performance, we have resorted to ASIC implementation and simulation

V. SIMULATION RESULTS

Simulation Results

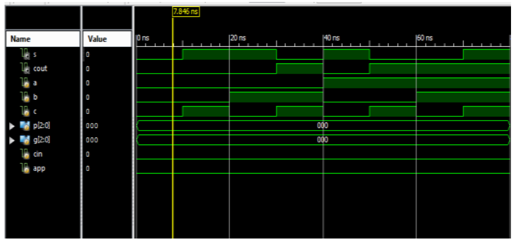


Fig 4.Simulation Wave Form For RAP CLA

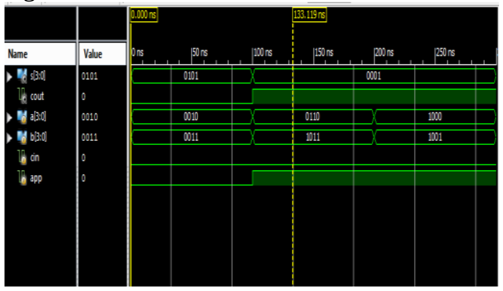


Fig .5 Simulation For 4bit RAP CLA adder

VI. CONCLUSION

In this project, a high-speed yet energy-efficient reconfigurable approximate carry look-ahead adder was suggested. The adder enjoyed the ability of switching between the approximate and exact operating modes making it suitable for both error-resilient and exact applications. The structure of the proposed adder was based on some modifications to the structure of t conventional CLA. To assess the efficacy of the proposed structure, its design parameters were compared to those of some suggested reconfigurable approximate adders. The parameters which included delay, power, energy energy-delay-product, and area were evaluated at a 15nm technology. The results showed up to 49% and 19% lower delay and power consumption, respectively, compared to the those of the approximate adders at the price of up to an error rate of 35.16%. Also, the effectiveness of the proposed adder on two image processing applications is studied as well.

REFERENCES

[1] K. K. Parhi, VLSI Digital Signal Processing. New York, NY, USA: Wiley, 1998.
[2] A. P. Chandrakasan, N. Verma, and D. C. Daly, "Ultralow-power electronics for biomedical applications," Annu. Rev. Biomed. Eng., vol. 10, pp. 247– 274, Aug. 2008.
[3] O. J. Bedrij, "Carry-select adder," IRE Trans. Electron. Comput., vol. EC-11, no. 3, pp. 340–344, Jun. 1962.
[4] Y. Kim and L.-S. Kim, "64-bit carry-select adder with reduced area," Electron. Lett., vol. 37, no. 10, pp. 614–615, May 2001.
[5] Y. He, C. H. Chang, and J. Gu, "An area-efficient 64-bit square root carryselect adder for low power application," in Proc. IEEE Int. Symp. Circuits Syst., 2005, vol. 4, pp. 4082–4085.
[6] B. Ramkumar and H. M. Kittur, "Low-power and area-efficient carry-select adder," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 2, pp. 371–375, Feb. 2012.
[7] I.-C. Wey, C.-C. Ho, Y.-S. Lin, and C. C. Peng, "An area-efficient carry select adder design by sharing the common Boolean logic term," in Proc. IMECS, 2012, pp. 1–4.
[8] S. Manju and V. Sornagopal, "An efficient SQRT architecture of carry select adder design by common Boolean logic," in Proc. VLSI ICEVENT, 2013, pp. 1–5.
[9] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, 2nd ed. New York, NY, USA: Oxford Univ. Press, 2010.