IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 12, Issue 2, Ver. II (Mar.-Apr. 2017), PP 44-49
www.iosrjournals.org

High –Speed Implementation of Design and Analysis by Using Parallel Prefix Adders

K. MADHAVI, P. DIVYA,
Asst. Professor, Dept of E.C.E, LENDI INSTITUTE OF ENGINEERING AND TECHNOLOGY
E.mail id:kkmadhu432@gmail.com
Asst. Professor, Dept of E.C.E, LENDI INSTITUTE OF ENGINEERING AND TECHNOLOGY
E.mail id:divyapenugurthi@gmail.com

Abstract: *The binary adder is the critical element in most digital circuit designs including the digital signal processors (DSP) and microprocessor data unit path. As such as extensive research continues to be focused on improving the power, delay, improvement of the adder. The design and analysis of the parallel prefix adders (carry select adders) is to be implemented by using Verilog. In VLSI implementations, parallel prefix adders are very high speed performance. Binary adders are one of the most essential logic elements within a digital system. Therefore, binary addition is essential that any improvement in binary addition can result in a performance boost for any computing system and hence, help improve the performance of the entire system. Parallel-prefix adders (also known as carry-tree adders) are known to have the best performance in VLSI designs. This paper investigates (the Kogge-Stone, sparse Kogge-Stone, Ladner fischer adder, Brent-Kung adder) and compares them to the simple Ripple Carry Adder (RCA) for high number of binary bits.*

Keywords: *Parallel-Prefix Adders, Xilinx ISE 14.7i, Verilog, Power, Delay.*

I. Introduction

To humans, decimal numbers are easy to comprehend and implement for performing arithmetic. However, in digital systems, such as a microprocessor, DSP (Digital Signal Processor) or ASIC (Application-Specific Integrated Circuit), binary numbers are more pragmatic for a given computation. This occurs because binary values are optimally efficient at representing many values. Binary adders are the most important to calculate any type of Arithmetic operation in digital design. The major problem for binary addition is the carry chain. As the width of the input operand increases, the length of the carry chain increases. Fig.1.If the length of the bit is increased the sum is bit is also increased. Demonstrates an example of an 8- bit binary add operation and how the carry chain is affected. When we implement any design the no luts are present. depends on luts the is depends .As the no binary bits increases the no of look up table(lut) is increased. As the no of binary bits increases the carry has to travel for longer paths. This example shows that the worst case occurs when the carry travels the longest possible path, from the least significant bit (LSB) to the most significant bit (MSB). In order to improve the performance of carry-propagate adders, it is possible to accelerate the carry chain, but not eliminate it. Consequently, most digital designers often resort to building faster adders when optimizing computer architecture, because they tend to set the critical path for most computations. In VLSI implementations, parallel-prefix adders are known to have the best performance. Reconfigurable logic such as Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved.

performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs involving mobile DSP and telecommunications applications and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs.

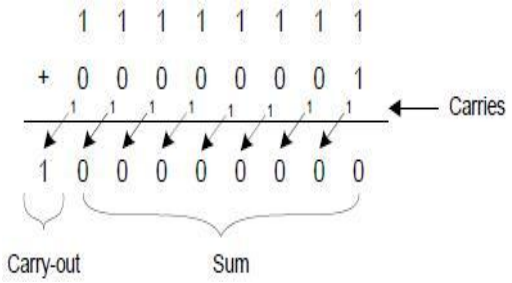

Fig.1. Binary Adder Example.

The power advantage is especially important with the growing popularity of mobile and portable

electronics, which make extensive use of DSP functions. However, because of the structure of the configurable logic and routing resources in FPGAs, parallel-prefix adders will have a different performance than VLSI implementations. In particular, most modern FPGAs employ a fast-carry chain which optimizes the carry path for the simple Ripple Carry Adder (RCA). In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are described.As the no of bits has to travel power plays an important role.

Several tree-based adder structures are implemented and characterized on a FPGA and compared with the Ripple Carry Adder (RCA) and the Carry Skip Adder (CSA). Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given.

II. Binaryadder Schemes

Adders are one of the most essential components in digital building blocks however, the performance of adders become more critical as the technology advances. The problem of addition involves algorithms in Boolean algebra and their respective circuit implementation. Algorithmically, there are linear-delay,and more no of luts are present in adders like ripple-carry adders (RCA), which are the most straightforward but slowest. Adders like carry-skip adders (CSKA), carry-select adders (CSEA) and carry-increment adders (CINA) are linear-based adders with optimized carry-chain and improve upon the linear chain within a ripple-carry adder. Carry-look ahead adders (CLA) have logarithmic delay and currently have evolved to parallel-prefix structures. Other schemes, like Ling adders, NAND/NOR adders and carry-save adders can help improve performance as well.

A. Binary Adder Notations and Operations

As mentioned previously, adders in VLSI digital systems use binary notation. In that case, add is done bit by bit using Boolean equations. Consider a simple binary add with two n-bit inputs A,B and one-bit carry-in $c_{in,}$ and with n-bit output is denoted as Sum.

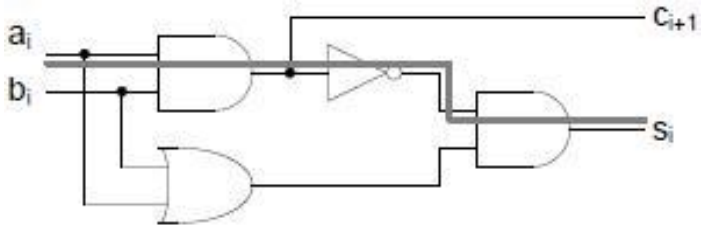

Fig.2. 1-bit Half Adder.

$$S = A + B + c_{in} \quad \textbf{(1)}$$

where $A = a_{n-1}, a_{n-2}\ldots\ldots a_0$; $B = b_{n-1}, b_{n-2}\ldots\ldots b_0$.

The + in the above equation is the regular add operation. However, in the binary world, only Boolean algebra works. For add related operations, AND, OR and Exclusive-OR (XOR) are required. In the following documentation, a dot between two variables (each with single bit), e.g. a _ b denotes 'a AND b'. Similarly, a + b denotes 'a OR b' and a _ b denotes 'a XOR b'. Considering the situation of adding two bits, the sum s and carry c can be expressed using Boolean operations mentioned above.suppose if we add 4 bits the output is only 2 variables like sum and carry.

$$s_i = a_i \hat{} b_i \quad \textbf{(2)}$$

$$c_{i+1} = a_i . b_i \quad \textbf{(3)}$$

The Equation of c_{i+1} can be implemented as shown in Fig.2. In the figure, there is a half adder, which takes only 2 input bits. The solid line highlights the critical path, which indicates the longest path from the input to the output. Equation of ci+1 can be extended to perform full add operation, where there is a carry input.

B. Ripple-Carry Adders (RCA)

The simplest way of doing binary addition is to connect the carry-out from the previous bit to the next bit's carry-in. Each bit takes carry-in as one of the inputs and outputs sum and carry-out bit and hence the name ripple-carry adder. This type of adders is built by cascading 1-bit full adders. A 4-bit ripple-carry adder is shown in Fig.3. Each trapezoidal symbol represents a single-bit full adder. At the top of the figure, the carry is rippled through the adder from cin to cout. It can be observed in Fig.3 that the critical path, highlighted with a solid line, is from the least significant bit (LSB) of the input (a0 or b0) to the most significant bit (MSB) of sum (sn-1).

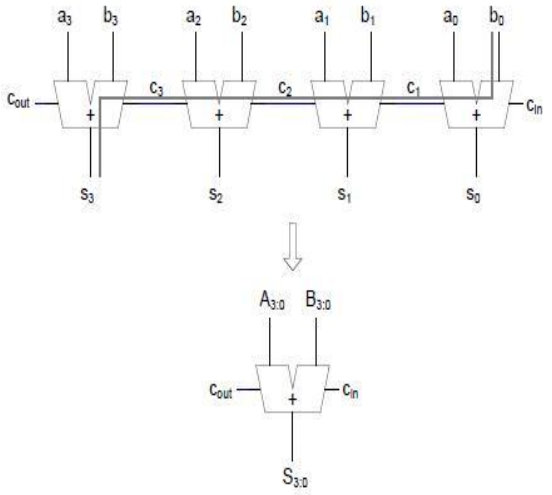

Fig.3. Ripple-Carry Adder.

C. Carry-Skip Adders (CSKA)

There is an alternative way of reducing the delay in the carry-chain of a RCA by checking if a carry will propagate through to the next block. This is called carry-skip adders.

$$c_{i+1} = P_{i:j} \_ G_{i:j} + P_{i:j} \cdot c_j \quad \textbf{(4)}$$

Fig. 4 shows an example of 16-bit carry-skip adder.

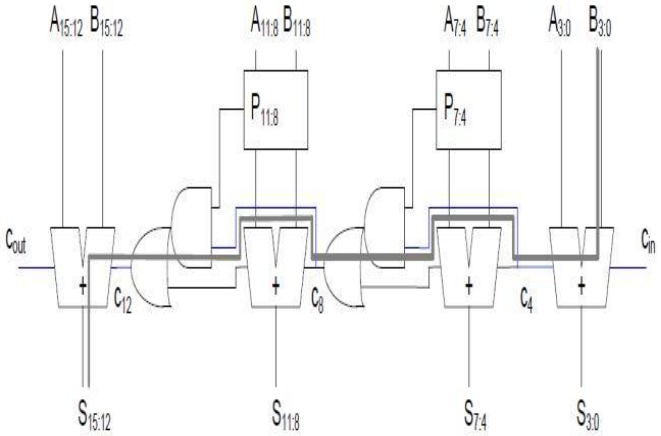

Fig. 4. Carry-Skip Adder.

The carry-out of each block is determined by selecting the
carry-in and $G_{i:j}$ using $P_{i:j.}$ When $P_{i:j}$ = 1, the carry-in c_j is allowed to get through the block immediately. Otherwise, the
carry-out is determined by Gi:j. The CSKA has less delay in
the carry-chain with only a little additional extra logic. Further improvement can be achieved generally by making the central block sizes larger and the two-end block sizes smaller.

III. Parallel Prefix Adder

The PPA is like a Carry Look Ahead Adder. The production of the carriers the prefix adders can be designed in many different ways based on the different requirements. We use tree structure form to increase the speed of arithmetic operation. If we add parallel the time constrained of the design is less.ThereforeParallel prefix adders are faster adders and these are faster adders and used for high performance arithmetic structures in industries. Prefix means the outcome of the operation depends on the initial inputs. Parallel means involves the execution of an operation in parallel. This is done by segmentation into smaller pieces that are computed in parallel. Operation means any arbitrary primitive operator "○" that is associative is parallelizable. It is fast because the processing is accomplished in a parallel fashion.

- An example shown that Associative operations are parallelizable.

Consider the logical OR operation: a + b this operation is associative.

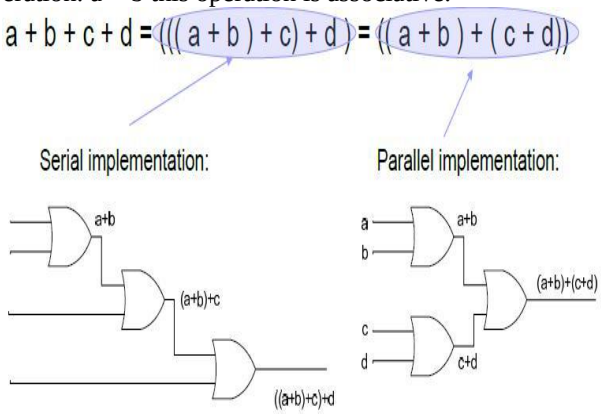

Fig.5.serial and parallel implementation

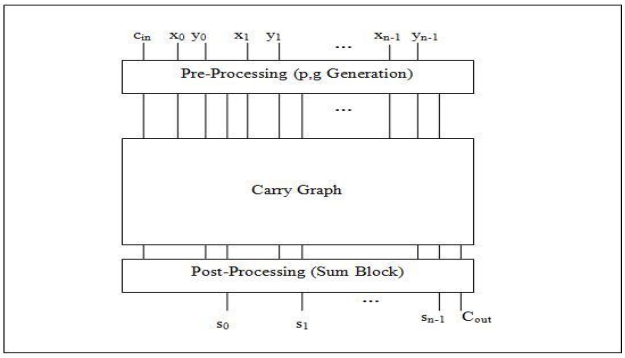

Fig 6. PPA structured diagram

In the following discussion about prefix trees, the radix is assumed to be 2 (i.e. the number of inputs to the logic gates is always 2). The more aggressive prefix schemes have logic levels [log2(n)], where n is the width of the inputs. However, these schemes require higher fan out, or many wire-tracks or dense logic gates, which will compromise the performance e.g. speed or power. Some other schemes have relieved fan-out and wire tracks at the cost of more logic levels. When radix is fixed, The design trade-off is made among the logic levels, fan-out and wire tracks.

B. Building Prefix Structures

Parallel-prefix structures are found to be common in high performance adders because of the delay is logarithmically proportional to the adder width. The parallel prefix addition is done in 3 steps.

- Pre-processing stage
- Carry generation network
- Post processing stage

The parallel prefix adder operation has the following 3-stage structure is briefly explained as, in the first stage Pre-calculation of p_i, g_i for each stage, in the second stage Calculation of carry c_i for each stage, in the third stage Combine c_i and pi of each stage to generate the sum bits si final sum. The group generate/propagate equations are based on single bit generate/propagate, which are computed in the pre-computation stage.

Pre-processing Stage: In this stage we compute, the generate and propagate signals are used to generate carry input of each adder. A and B are inputs. These signals are given by the equation 1&2.

$$P_i=A_i\ 0B_i \quad \textbf{(5)}$$

$G_I=A_i.B_i$ **(6) Carry Generation Stage:** In this stage we compute carries corresponding to each bit. Execution is done in parallel form [4].After the computation of carries in parallel they are divided into smaller pieces. carry operator contain two AND gates , one OR gate. It uses propagate and generate as intermediate signals which are given by the equations 3&4.

Post Processing Stage: This is the final stage to compute the summation of input bits. it is same for all adders and sum bit equation given

C. Parallel Prefix Operations

Here we designate BC as the black cell which generates the ordered pair in equation the gray cell (GC) generate signal only. The interconnect area is known to be high, but for an FPGA with large routing overhead to begin with, this is not as important as in a VLSI implementation.

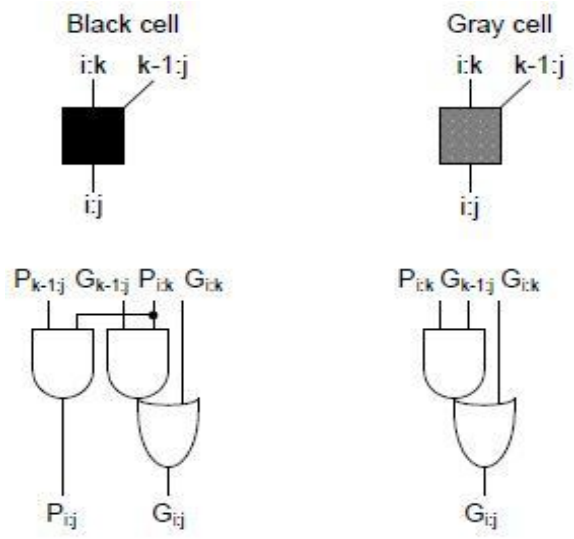

Fig.7. Cell Definitions.

The black operator receives two sets of generate and propagate signals (g_i, p_i), (g_{i-1}, p_{i-1}), computes one set of generate and propagate signals (g_o , p_o) and The gray operator receives two sets of generate and propagate signals (g_i, p_i),(g_{i-1} , p_{i-1}), computes only one generate signal with the same.

$$s_i = p_i \,.\, G_{i-1}\text{:-}1 \tag{7}$$

$$c_{out} = G_n\text{:-}1 \tag{8}$$

where "-1" is the position of carry-input.The generate/ propagate signals can be grouped in different fashion to get the same correct carries. Based on different ways of grouping the generate/propagate signals, different prefix architectures can be created. Fig.6 shows the definitions of cells that are used in prefix structures, including black cell and gray cell. Black/gray cells implement the above two equations, which will be heavily used in the following discussion on prefix trees.

IV. Different Types Of Parallel Prefix

ADDERS A. Kogge-Stone Prefix Tree

Kogge-Stone prefix tree is among the type of prefix trees that use the fewest logic levels. A 16-bit example is shown in Fig.7. In fact, Kogge-Stone is a member of Knowles prefix tree. The 16-bit prefix tree can be viewed as Knowels [1,1,1,1]. The numbers in the brackets represent the maximum branch fan-out at each logic level. The maximum fan-out is 2 in all logic levels for all width Kogge-Stone prefix trees. The key of building a prefix tree is how to implement Equation according to the specific features of that type of prefix tree and apply the rules described in the previous section. Gray cells are inserted similar to black cells except that the gray cells final output carry outs instead of intermediate G/P group. The reason of starting with Kogge-Stone prefix tree is that it is the easiest to build in terms of using a program.

concept. The example in Fig.7 is 16-bit (a power of 2) prefix tree. It is not difficult extend the structure to any width if the basics are strictly followed.

B. Brent-kung Adder

The Brent kung adder consists of several smaller ripple carry adders(RCA) on its lower half, a carry tree on its upper half. It terminates with RCAs. The number of carries generated is less in a sparse kogge stone adder compared to the regular kogge-stone adder. the functionality of the GP block, black cell, gray has been reduced. The sparse kogge-stone adder ,this design terminates with the 4 bit RCA.

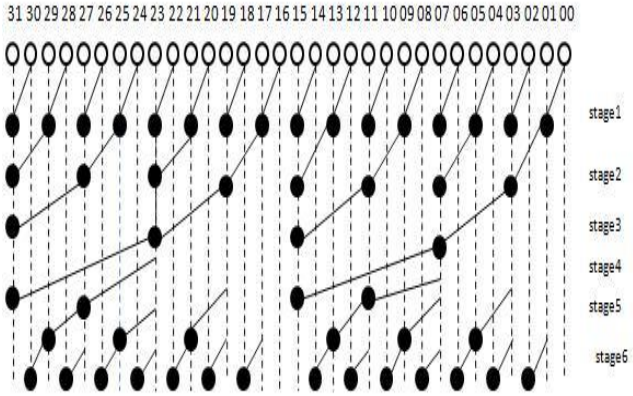

Fig.8. 32-bit Brent Kung adder.

C. Sparse kogge-stone adder

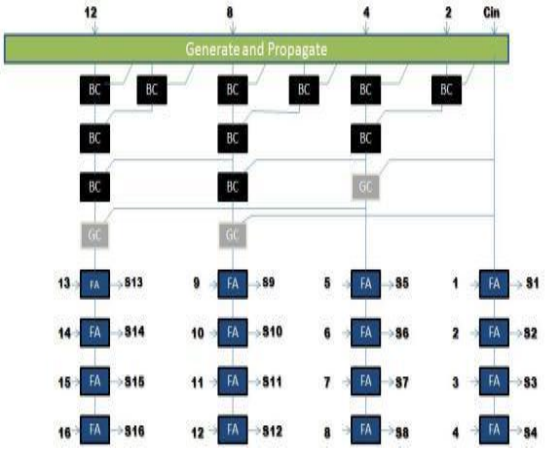

Fig.9: Sparse 32 bit Kogge-Stone adder.

Results and analysis of PPA

S.no	Adder name		Delay	Power	No of Luts
1	RCA	53.16	0.663	73	
2	KSA	24.13	1.78	204	
3	Brent kogge stone adder		22.12	0.456	39
4	Ladner	28.12	0.723	90	

V. Conclusion

In terms of area or cost between the two PPAs, the BKA proves to be a better choice. Even though the BKA's area rises as the bit size increase, the taken of Brent-Kung Adder is very less less compared to remaining designs it does not rise as drastically as KSA. Propagation delay (t_{pd}), KSA is a better choice. Although BKA has lower t_{pd} for bit size of 8 bits, the KSA has very low t_{pd} compared to BKA when the bit size is more than 32 bits.As the bit size increases for critical power applications. Also Brent-kung adder is sufficient compared to other adders. Therefore, only at bit size less than 32 bits the KSA has longer t_{pd}. The KSA is widely-known as a PPA that performs fast logical addition.

References:

[1]. R. P Brent & H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers*, Vol C-31, pp 260-264, 1982.
[2]. P. M. Kogge & H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans.*
[3]. *Computers*, Vol. C-22, pp 786-793, 1973. R. E. Ladner & M. J. Fischer, "Parallel Prefix Computation,"
[4]. *JACM*, Vol. 27-4, pp 831-838, 1980.
[5]. T. Hans & D. A Carlson, "Fast Area-Efficient VLSI Adders," *Proc. 8th IEEE Symposium on Computer Arithmetic*, pp 49-56, 1987.
[6]. S. Knowles, "A Family of Adders," *Proc. 15th IEEE Symposium on Computer Arithmetic*, pp 277-281, 2001.