**Performance Analysis of Parallel Prefix Adders Using Zynq-7000 SoC**

**Neeraj Kumar Cheryala1, Nikhil Madhavaneni2, Rajesh Odela\*3**

*1,2Student, Dept of ECE, Vasavi College of Engineering, Hyderabad, Telangana State, India*

*3\*Assistant Professor, Dept of ECE, Vasavi College of Engineering, Hyderabad, Telangana State, India*

*Email : 1neerajcher@gmail.com1, 2nikhil.madhavaneni@gmail.com, 3rajesh.odela2012@gmail.com\**

**ABSTRACT**

*The heart of every processor is the Arithmetic Logic Unit (ALU) and the heart of every ALU is the Adder circuit. Adder does subtraction (via 2's complement arithmetic) and is the core part of multiplier circuits. Since addition is the most basic arithmetic operation; and adder is the most fundamental arithmetic component of the processor, over the years the field of VLSI arithmetic has been intriguing for many digital VLSI researchers around the world. Several traditional and novel approaches have been introduced as years passed by to make the addition as efficient as possible in terms of delay, power consumption, area and loading of gate outputs. This paper discusses a chosen few traditional approaches to addition, analyses their performance and then evaluates and compares the performance of several Parallel prefix adders. Parallel Prefix adders are the most efficient approach for DSP chips till date. The goal is to decide the best adder out of those discussed for a given application with low on-chip power, resource consumption and higher speed of calculation. The adders are implemented in Verilog Hardware Description Language using Vivado Design Suite wherein the designs are implemented on Xilinx ZYNQ XC7Z020 (7000 series) SoC. Subsequently, the power, delay and hardware utilization of the adders are investigated.*

*Keywords— Ripple-carry adder, Parallel prefix adders, Prefix tree, Kogge-Stone adder, Brent-Kung adder, Han-Carlson adder, Ladner-Fischer adder, Vivado design suite, Zynq-7000.*

**1. INTRODUCTION**

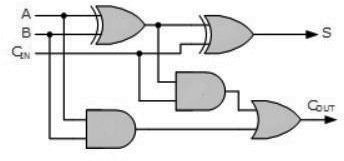
The VLSI chips highly rely on high speed, power efficient adder circuits for computations. Conventional adders succeed to fulfill the requirement till certain length of the input bits. As the input bit width increases, some of these adders slow down in calculations while consuming lesser power and area and some other variants are fast with comparatively higher power consumption. Parallel Prefix adders offer a good trade-off between speed and power consumption [1]. This paper is intended to discuss and analyse a few conventional adders and Parallel Prefix adders. It also compares their performance in terms of power consumption and delay while highlighting the drawbacks of traditional approaches. Finally, the power consumption, area and delay aspects of each Parallel Prefix adder discussed in this paper are presented and compared with one another.

The fundamental unit of conventional adders i.e. Full adder [2] is discussed later in this section. Section-II discusses a chosen few conventional multibit adders including Ripple-carry adder, Carry-bypass adder and Carry-lookahead adder. The section also highlights the drawbacks of conventional approaches and presents the need to go for Parallel prefix adders. Then, Section-III discusses the foundations, prefix trees and concepts behind Parallel Prefix Adders namely Kogge-Stone adder, Brent-Kung adder, Han-Carlson and Ladner-Fischer adders. Next section i.e. Section-IV is exigent in the sense that this section compares the performance of all the parallel prefix adders discussed in the previous section evaluating them theoretically and subsquently using the Vivado design suite with Verilog code written and implemented on Xilinx Zynq-7000SoC.

As mentioned earlier, let's first discuss the fundamental 1-bit full-adder which is the building block of the traditional approaches to adder circuits. This shouldn't take a dime.

***Full Adder***

A full adder is an addition to the half adder, which can compute the addition of two inputs and the previous carry. The delay in the circuit is 2 time units. A full adder cannot fulfill the requirement of an adder in anALU.



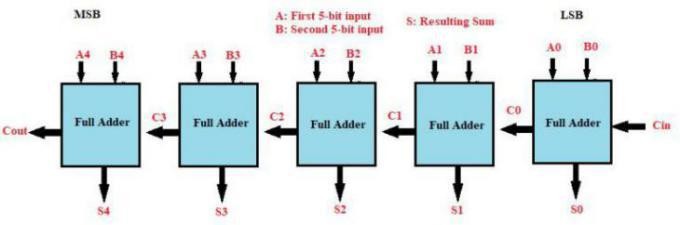
**Fig. 1: Full-adder logic circuit.**

Till now, we have seen the addition for one-bit inputs, but in real-time, the inputs are of multiple bits. Now let's look into the circuits which are capable of performing addition on multiplebits.

**2. CONVENTIONAL MULTI-BITADDERS**

With all arithmetic circuits, the delay is the most important parameter for measuring performance. Arithmetic circuits become part of the datapath of the circuit. The datapath needs to be as fast as possible, with power consumption being a second metric in terms of importance. The critical path is the longest possible delay between any two input and output registers (Where the inputs and outputs are stored). This is important because the adders other than parallel prefix adders can be easily analysed and evaluated with the critical path.

*Ripple-Carry adder*

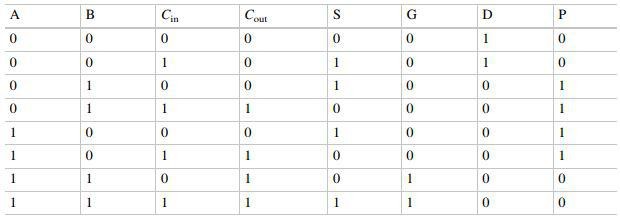


**Fig. 2: Block diagram of a 5-bit Ripple-carry adder.**

It is a binary adder that is capable of producing an arithmetic sum of two binary numbers. From the name of the circuit, we have an understanding that a ripple is generated for the carry. It is constructed by the combination of ‘n’ full adders for n-bit adder in a cascaded fashion, with the output carry of each full adder connected to input carry of the next full adder. The delay in the circuit is ‘n’ time constants for an ‘n’ bit adder, wherein the delay of one full adder is considered as one time constant. Here the critical path is from starting bit and it propagates to the last bit, as each full adder has to wait for the previous full adder carry to perform the addition of individual bits.

This adder isn't generally preferred because of the large delay associated with it as the number of input bitsincreases.

*Generate - Propagate Logic*



**Fig. 3: Truth table for a full adder with a different perspective including the generate and Propagate bits.**

Above is the truth table for a full adder but with a different perspective. This new approach is the first step to design faster adders. Here we express sum and cout in terms of G (Generate), P (Propagate) and D (Delete). These three bits define the carry-out andare introduced because Cinistightly coupled to them. The variables G, P and D are mutually-exclusive and are defined as follows:

Generate: Cout is 1 regardless of what Cin is. *G=A&B* Propagate: Cin is copied to Cout (Propagated). *P=A^B* Delete: Cout is 0 regardless of what Cin is. *D=(~A)&(~B)* Coutcanbeexpressedas:*Cout = G/(P&Cin)* (Ddoesnotappear in the expression because Cout is 0 when both G and P are 0. This is when D = 1) and *Sum=A^B^Cin*Also, note that the G, P and D are independent of Cin. This means that these bits can be calculated directly with given operand bits A and B and hence our adder is no more dependent on previous stage's Cout i.e. Cin for the presentstage.

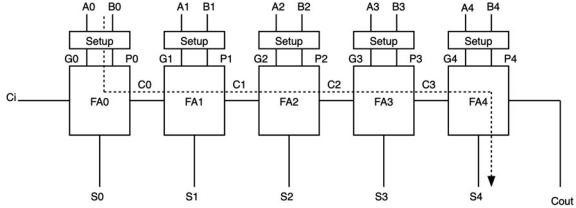
In fact, all G’s and P’s are calculated in parallel within Tgp (Time to calculate G and P) of the start of operation. Note that the coefficient of Tgp (Delay due to generate-propagate logic) is 1 since the generate-propagate bits are calculated simultaneously for all the bit positions. Thereafter, the operation is similar to that of a Ripple Carry Adder. Thus, the propagation delay is calculated as follows:

*Tpd(Propagation delay) =Tgp+(N-1)Tcarry+Tsum*

# where Tcarry = Time for calculating Cout of each bit

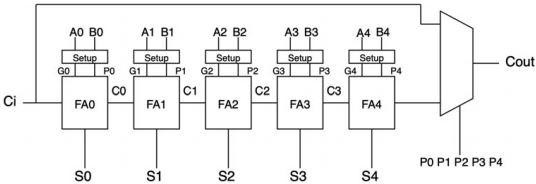
# Tsum = Time for calculating the last bit's Sum

The equation shows that this has no considerable effect on larger word lengths since then Tcarry term would be dominant.



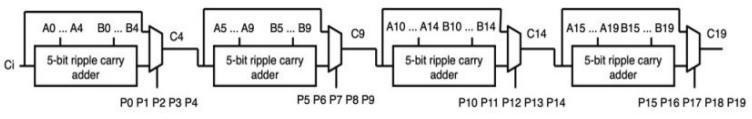
**Fig. 4: RCA with generate-propagate bits. [3]**

*Carry-skip (or) Carry-bypass Adder*

**

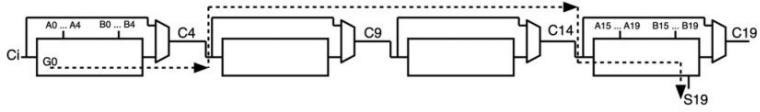
**Fig. 5: Five-bit RCA with G and P bits and a 2x1 Mux. [3]**

The above figure is a five bit ripple carry adder [3] butwith a generate and propagate logic (as discussed in the previous section) at each bit. Besides, the final carry-out of the chain is multiplexed between two cases. If all five-bit position propagates are true, then the entire five-bit adder is propagating. In this case, the carry-out is chosen to be the carry-in. In any other case, the carry-out is chosen to be the output of the five-bit ripple carry adder. This approach doesn't really seem to reduce the critical path delay. Firstly, the last output to come out of the five-bit adder is S4 rather than Cout. Thus, even if the bypass adder manages to produce Cout earlier, we would still have to wait for the sum bit. Additionally, the case where all bits are propagating is a special case. There are 32 possibilities for P0 P1 P2 P3 P4 and “11111” is just one of them. Thus, even if Cout is considered our ultimate output, we would get an improvement in the delay of this output only 1 out of 32 times. Although, the best case delay is improved, the worst case is still 4Tcarry +Tsum.

****

**Fig. 6: 20-bit adder from 5-bit carry-bypass adders. [3]**

The above figure shows a 20-bit adder formed by four 5-bit carry bypass adders. If the entire 20-bit adder were made as a simple ripple carry adder, the critical path delay would be 19Tcarry + Tsum. However, a block-wise bypass as shown in figure does lead to a significant improvement. To measure this improvement, we have to determine the critical path. The critical path is shown in next figure. This path is a little counterintuitive to understand, so we will examine it in detail. The critical path [4] is observed when the following G-P values are present; G0 or D0 = 1, P1 through P18 = 1, and G-P for bit position 19 being don’t cares. This critical path indicates that the first-bit position generates a carry. This carry is then propagated by bits 1-4 where the ripple-carry path is chosen at the first multiplexer. In the second block, the propagating bits 5-9 and the bypass path are multiplexed. The same happens for the third block for bits 10–14. In the last block, we are interested in S19 rather than Cout. Thus, regardless of the conditions on propagate in the last block, we still have to go through the ripplepath.

**Fig. 7: Critical path of a 20-bit carry-bypass adder. [3]**

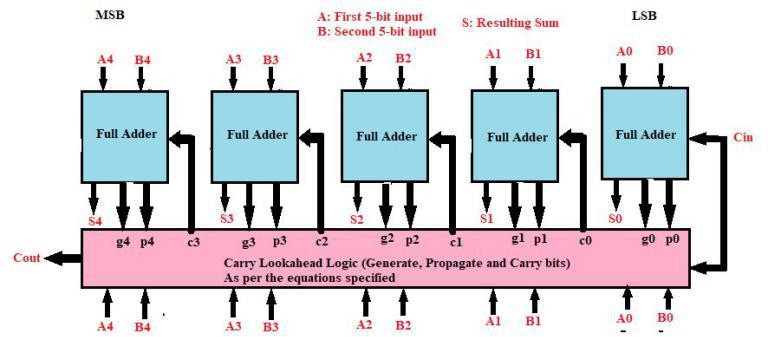
If we consider an N-bit adder divided into P blocks, then each block contains K = N/P bits. We can write the critical path delay in terms of N and K asfollows:

*Tpd=KTcarry+(N/K-1)Tmux+(K-1)Tcarry+Tsum*

The delay is still linearly dependent on the number of bits N. But the slope of graph between propagation delay and Tcarry is reduced to N/K from N. What we are missing in our discussion is the intercept term. The intercept can only be calculated at N >= K. Below N = K we have an undefined area where the number of bits in a block is larger than the total numberofbitsintheadder.TheinterceptisTsum+(2K−

1) Tcarry. Thus, as K increases the intercept increases while the slope decreases. This means that for a certain N, there will be an opposing perspective to choose different K. For very large N, it is best to choose higher K because the lower slope means there is less accumulated delay for the large number of bits. For smaller N, it is better to choose a lower K so as to reduce the initial cost of they-intercept.

*Carry Lookahead Adder*



**Fig. 8: Block diagram of 5-bit Carry-lookahead adder.**

As we have seen that propagate and generate logic is used to design faster adders, the carry-lookahead adder [2] makes use of them to calculate one or more carry bits before the sum. This reduces the wait time to calculate the result of the larger-value bits of the adder. Once the carry for all bits is generated, the sum bit is calculated by ex-or operation of propagate and carry. Here the delay is constant for all n-bit adders, which is 3-time constants, here 3 levels indicate propagate and generate, carry-lookahead generator block, and the final stage sum. The major disadvantage here is that area consumed by the adder is vast and as the number of bits increases, the area increases exponentially which may inturn require more power to drive thecircuit.

To summarize, conventional approaches to adder circuits fail to offer good trade-offs between delay, power and area. While the Ripple-carry adder requires nominal hardware, it is fundamentally slow as the input carry can be propagated all the way to the output. Hence the delay increases linearly with input bit-width. On the other side Carry lookahead adder performs very fast addition and the delay does not increase with the input bit-width. However, the hardware utilization increases exponentially and hence the power consumed for addition. Interestingly, the Carry-Skip adder has an improved speed compared to Ripple-Carry adder but only when we choose optimal K and N as discussed previously. Else, this adder offers similar delay as the Ripple-Carry adder. This may not be aligned to the design patterns while manufacturing the VLSI chips as all the bit combinations are not optimal to improve the delay using Carry-Skip adder. For instance, we can improve the performance only when 5-bit ripple carry adders are cascaded to form a 20-bit adder as discussed previously. Moving forward, we discuss the concepts of Parallel Prefix Adders in the next section which offer a good-tradeoffs between delay, power and area etc.

**3. PARALLEL PREFIXADDERS**

*Group Propagate, generate and the Dot Operator*

In the previous sections, we have analysed the traditional adders out of which Carry-lookahead adder proved to be the fastest while consuming comaparatively large amounts of logic and power. However, the best part of carry-lookahead addition is that if performed in a hierarchical fashion, this allows us to design some of the fastest adders possible with lesser logic and power consumed when compared to the traditional Carry-lookahead adder.

Consider the carry expression for the third bit;

*C2=G2 | P2&G1 | P1&P2&G0 | P0&P1&P2&Cin*

We can observe a structure for these expressions as if in the above carry expression, the first term represents the current position generating a carry. The second term represents the previous position generating a carry and the current position propagating it. The third term, the bit position two bits removed generating and the following two positions propagating and the last term considers the possibility of a carry-into the very first-bit position that all subsequent positions then propagate. The last term is called as the 'group-propagate' term as it corresponds to all the bit positions propagating the very input carry to the output. All the other terms in any of these expressions are collectively called as 'group-generate' term as they represent the carry-generation at each corresponding bit position and then propagated to the output. Group-propagates and generates can be defined for any range of bit positions say 2-4 for instance but should be contiguous. In this paper, group-propagate and generate for x:y range are represented as Pxy andGxy.

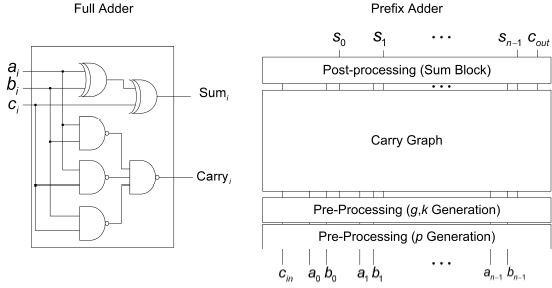
The concept of Group-generate and propagate along with the 'dot' operator (which is discussed in this section) leads to the design of very efficient adders [5] called as 'Parallel-prefix adders'. The dot operator is used in the design of parallel adders to derive a longer group from two groups of generates and propagates which are contiguous again. Thus it is defined as,

*(Gxz, Pxz)=(Gxy,pxy) o [G(y+1)z, P(y+1)Z]=*

*[G(y+1)z| Gxy&P(y+1)z&Pxy].*

Note that the dot operator is not commutative and it will combine continuous ranges while ignoring any overlaps in the input ranges.

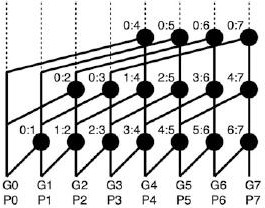
Now using the definitions of carry-lookahead addition, group generates and propagates, we can design and analyse a new class of adders as mentioned in the previous section called as Parallel Prefix Adders [6]. As the name indicates, these adders achieve their speed through 'parallelism' and hierarchical approach to calculate the carry of each bit position. However, they occupy more area when compared to the ripple-carry adder and also suffer from disadvantages of loading on gate outputs (Fan-out) and difficult routing. There have been five major parallel prefix adders which have their own advantages and disadvantages when compared with each other. They are Kogge-Stone adder, Brent-Kung adder, Han-Carlson adder, Ladner-Fischer adder and Knowles adders. Note that the design of Knowles adders is not discussed in this paper since it includes classes and would increase the length of paper. The Kogge-Stone adder which is the simplest of all the parallel prefix adders is discussed in the followingsection.



**Fig. 9: Full adder vs Prefix adder [8]**

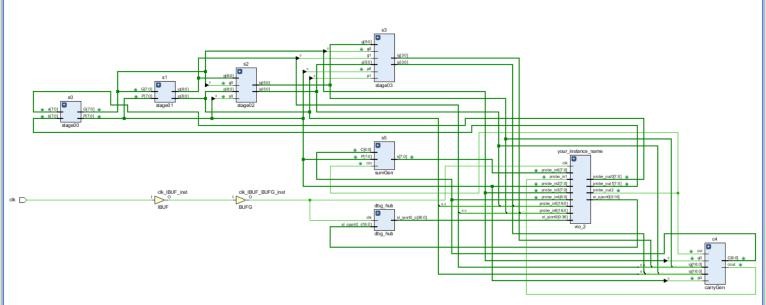
*Kogge-Stone Adder*

It's one of the most popular parallel prefix adders. It has a faster design which makes the computation speed high, but the area of the circuit is more. It has three stages [7] for generating the addition as any parallel prefix adder consists: Pre-processing, Carry Graph and Post-processing (Sum block). Below is the image of the carry graphnetwork.



**Fig. 10: Prefix tree for 8-bit KS adders. [3]**

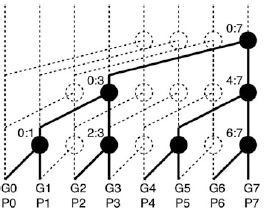
At the 0th stage, generate (gi) and propagate (pi) are generated for each bit from the inputs A and B correspondingly. Now in the next stage which is carry generate stage, with the help of group propagate and generate, carry is derived. Here the number of steps in this stage varies as the number of input bits varies. The last stage is responsible for the generation of sum, here the propagate (pi) is XORed with carry which is generated from the carry network (ci). The advantages of the adder are that the fan-out is minimum, the addition operation is fast, and the delay in the circuit is small which is given by log2(n) where n is the number of the bits in the input. The disadvantage is that the area taken by the adder is large. Figure 11 shows the schematic of an 8-bit Kogge-Stone adder obtained using Vivado Design Suite (Discussed in IV).



**Fig. 11: Synthesized schematic of 8-bit Kogge-Stone adder.**

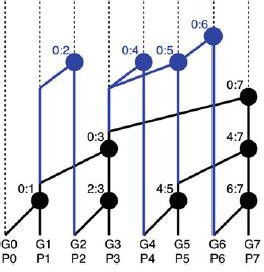
*Brent-Kung Adder*

The disadvantages of the KS adder as we have seen in the carry graph network are the number of wires. This number is high and grows very fast with the size of the inputs. Also, the wires need to cross over many other wires which makes the routing difficult. Let's look at the 8-bit KS adder in a way that only the final carry matters. In this case, most of the dot operations can be dropped and the tree diagram would look as follows:



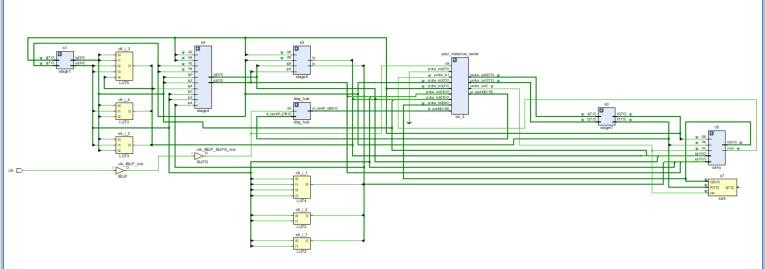
**Fig. 12: Pruned tree diagram for 8-bit KS adder. [3]**

This way, the routing congestion is reduced but this is not a proper 8-bit adder since all the internal carries are also needed in order to calculate S0 through S6. So, we make modifications to this pruned tree to derive a new adder called as Brent-Kung adder [8]. An 8-bit BK adder requires four additional dot operators at the top of the last-bit carry. These are used to calculate the missing carries using the minimum possible number of additional hardware units. Figure 13 shows an 8-bit BK adder prefix treediagram.



**Fig. 13: Prefix tree for 8-bit BK adder. [3]**

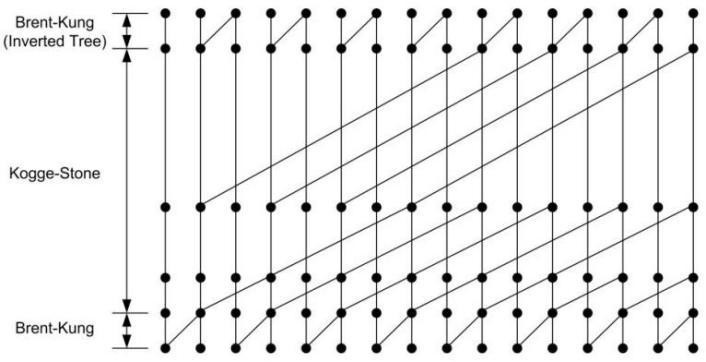
As you can see, BK adders are much smaller than KS adders. The 8-bit BK adder has 11 dot operations compared to 17 in KS. The 16-bit BK adder has 26 dot operators compared to 49 in 16-bit KS adder. Thus the area is the advantage of BK adder and becomes more highlighted as the input size grows. The BK adder also has wiring which is easily routable when compared to the KS adder. However, the BK adder has two major disadvantages. Firstly, the number of stages has now increased for all the internal bit carries to be calculated. Secondly, many dot operations have fan out of more than two units. Figure 14 shows the synthesized schematic of an 8-bit Brent-Kung adder obtained using Vivado Design Suite.



**Fig. 14: Synthesized Schematic of 8-bit Brent-Kung adder.**

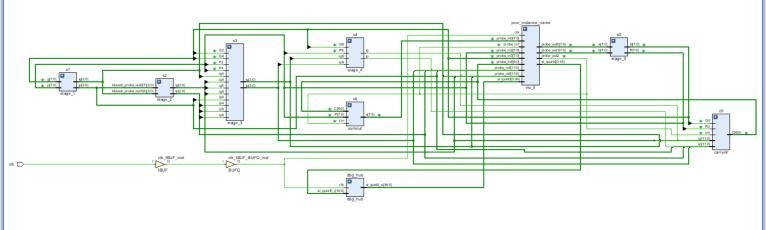
*Han-Carlson Adder (Hybrid Prefix Adder)*

Han-Carlson adder is a hyrid approach [9] which leverages both the Kogge-Stone and Brent-Kung adders (hence hybrid). It uses the BK approach (slow but small) in the first and last stages of the carry generation and the KS approach (fast but large) in the middle stages. The idea of Han-Carlson prefix tree is similar to Kogge-Stone's prefix trees as it has a maximum fan-out of 2. The difference is that Han-Carlson prefix tree uses much less hardware units and wirings than Kogge-Stone adder. The cost is one extra logic level. As a result, this adder offers a good trade-off between fan-out and number of logic units. Hence it also consumes lower power. Figure 15 shows the prefix tree for 16-bit Han-Carlsonadder.



**Fig. 15: Prefix tree for 16-bit HC adder.**

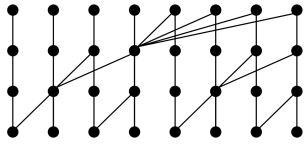
The Han-Carlson adder in the above figure uses a single Brent-Kung level at the beginning and at the end of the graph, and the Kogge-Stone approach in the middle. Simply put, Han-Carlson adder achieves higher speeds with lower power consumption and area. Figure 16 shows the synthesized schematic of an 8-bit Han-Carlson adder obtained using Vivado Design Suite.

**Fig. 16: Synthesized schematic of 8-bit Han-Carlson adder.**

*Ladner-Fischer Adder*

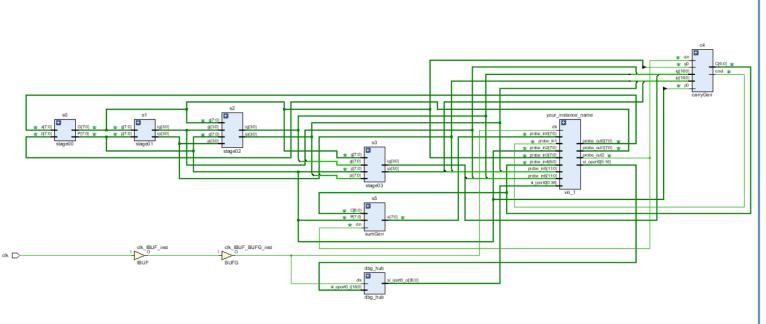
Here is another adder i.e. Ladner-Fischer Adder [10], it is also the fastest adder with focus on the design time and the most common choice of high-speed performance adders alongside the Kogge-Stone adder. It has three stages for generating the addition as any parallel prefix adder consists; Pre-processing, Carry graph network and Post-processing (Sum block).

At the 0th stage, generate (gi) and propagate (pi) are generated for each bit from the inputs A and B correspondingly. Now in the next stage which is carry generate stage, with the help of group propagate and generate carry is derived. Here the number of steps in this stage varies as the number of input bits varies. Below is the carry network carry graph, which is the logic carried out in generating carry.



**Fig. 17: Prefix tree for 8-bit Ladner-Fischer adder**.

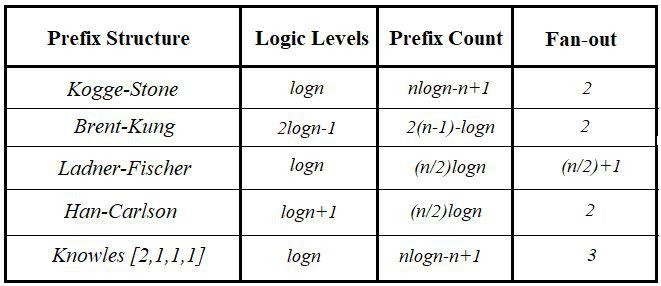
The last stage is responsible for the generation of sum. Here the propagate (pi) is XORed with carry which is generated form the carry network(ci). The advantages are that the logic level is simple that makes the circuit fast, which makes the circuit small. The disadvantage is that it has large fanout compared to other parallel prefix adders. Figure 18 shows the synthesized schematic of an 8-bit Ladner-Fischer adder obtained using Vivado Design Suite.



**Fig. 18: Synthesized schematic of 8-bit Ladner-Fischer adder.**

**4. PERFORMANCE COMPARISON OF PARALLELPREFIX ADDERS**

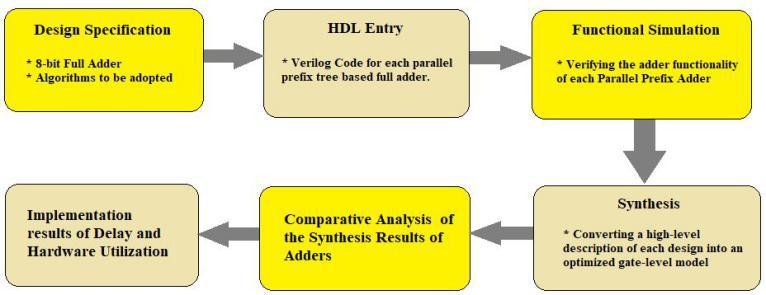
Now that we have understood several parallel prefix adders and the corresponding prefix tree diagrams, it’s time for evaluating them [11] for the best pick for a particular application. The below table gives a glance of performance attributes of all the PPAs discussed in this paper in theory as a function of input bit width. Note that the table also includes a class of Knowles parallel adders [12] to consolidate the evaluation.



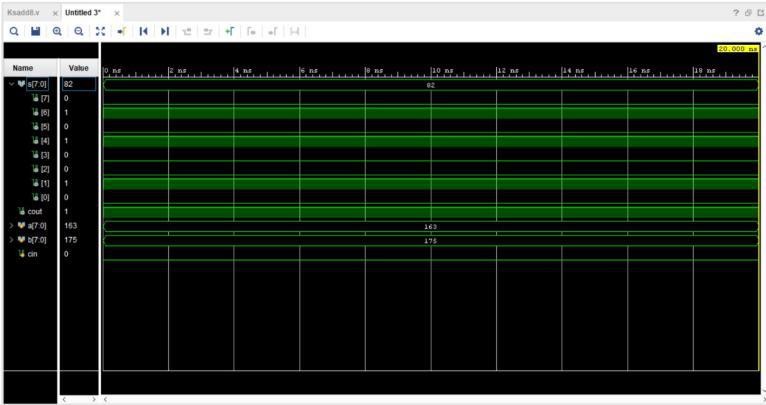
**Fig. 19: Theoretical comparison of PPAs discussed in this paper (includes Knowles [2,1,1,1] adder).**

*Performance Comparison Using Vivado Design Suite From Xilinx*

FPGAs and development boards from Xilinx are accompanied by Hardware development tools like the Vivado Design Suite [13]. The suite fills the requirement of design flows and provides a proven methodology that maximizes productivity from concept through implementation and debug. Figure 20 depicts the design flow and methodology adopted for the implementation and comparison of thedesigns.

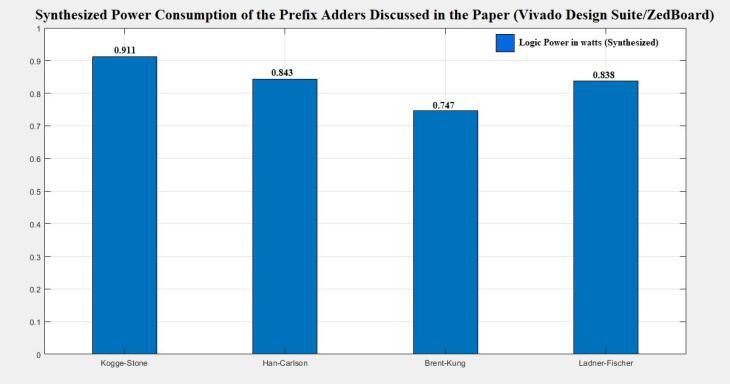


**Fig. 20: Design flow for implementation and Performance Comparison.**



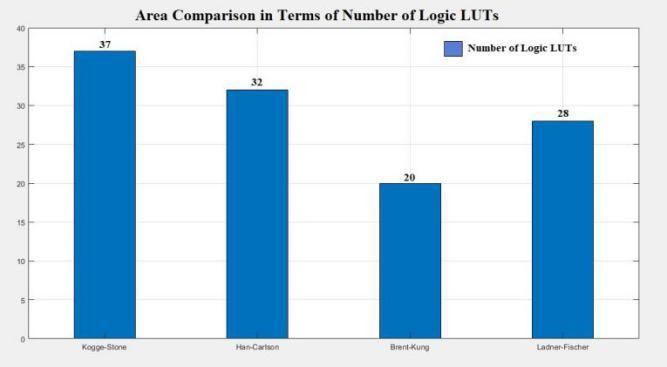
**Fig. 21: Simulation results for 8-bit Kogge-Stone adder (Similar results were obtained for other adder circuits).**

The designs are implemented on Zynq Evaluation and Development Kit (Zedboard) [14] which features an APSoC named XC-7Z020 CLG484 (Zynq-7000 series). It combines a dual Cortex A9 Processing System (PS) with 85,000 Programmable Logic cells (PL or FPGA). Figure 20 shows the bar graph for comparison of power consumption of 8-bit variants after synthesizing the designs. It includes dynamic logic power for each adder circuit.



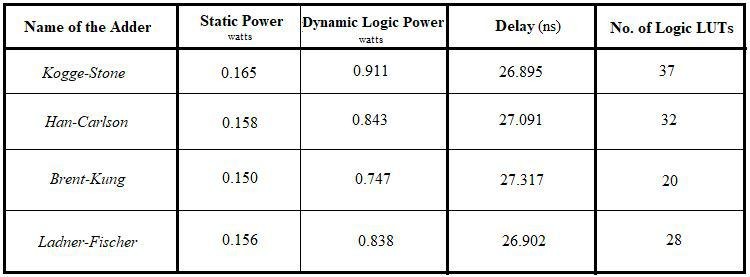
**Fig. 22: Synthesized Power Comparison of the Adder circuits discussed in the paper.**

The bar graph in Figure 22 shows that Brent-Kung adder has a lower dynamic power consumption when compared to other adder circuits. On the other side, Kogge-Stone adder has the highest dynamic power consumption complemented by higher hardware utilization which is obtained after implementation as shown by Figure 23. Lastly, we observe that there is a little difference between Han-Carlson and Ladner-Fischer adders with former consuming slightly more than the latter. Han-Carlson proves to be a hybrid approach as its power consumption lies between those of Kogge-Stone and Brent-Kung adders.



**Fig. 23: Comparison of number of LUTs for each Adder.**

We obtain the area for each adder circuit after implementing the designs in terms of number of Logic LUTs (Look-up tables) as specified by the tool. Figure 23 shows that Brent-Kung adder is impressive in terms of area and number of logic LUTs consumed. Its hardware utilization is almost half of that of Kogge-Stone adder. Han-Carlson adder with 32 LUTs stands between Kogge-Stone and Brent-Kung adders. From the above two comparisons, we can conclude that Brent-Kung has better performance in terms of power and area. Han Carlson is slightly better than Ladner-Fischer in area but falls behind when it comes to the dynamic power consumption. While Kogge-Stone is not so good for low power and resource-critical applications, it proves to be the best choice for high-speed applications like DSPs since it has the lowest delay as summarized in the Figure 24. The table summarizes all the performance aspects discussed till now along with delay and static powerconsumption.



**Fig. 24: Table Summarizing the Performance Comparison of Prefix Adders discussed in the paper.**

The table also provides a good intuition that Kogge-Stone adder has much lower delays compared to those of other prefix adders when the bit-width of the inputsincreases.

1. **5. CONCLUSION AND FUTURE WORK**

In this paper, we have discussed the disadvantages of conventional adders and the concepts behind Parallel Prefix adders. Implementation of the designs of Parallel Prefix adders using Vivado Design Suite consolidated the understanding of the trade-offs between these adders. Asper the results obtained, Kogge-Stone Adder proves to be the fastest adder while consuming higher on-chip power and area when compared to others. This can serve as an ideal complement to a multiplier in a MAC (Multiply-and-Accumulate) unit in signal processing applications required to be operated at higher speeds. Whereas Brent-Kung adder proves to be the best choice for low-power VLSI designs (where delay is not of primary concern) and is not so resource-intensive. Han-Carlson adder proves to be the hybrid approach with delay, area and power lying between those of Kogge-Stone and Brent-Kung adders. Finally, Ladner-Fischer adder outperforms Han-Carlson adder in resource and power consumption but is attributed with high fan-out. As part of our future work, we would like to extend the bit-width to 32 and 64 bits and compare the results at different operating temperatures. We would also like to optimize the static and dynamic power consumption of the adder circuits using low power VLSI design techniques. In our future papers, we intend to compare the fan-outs of the adder circuits which is not discussed in this paper. Lastly, we would like to design an FIR filter using Kogge-Stone adder for high speeds or Brent-Kung adder for low power design. The most interesting aspect is to leverage the reconfigurable FPGAs alongside a processing system (as present in Zynq-7000 Soc) to get reconfigured with appropriate adder in the field as required by the application. The ARM core can be programmed to reconfigure the PL part with adder suitable for a given application as discussed in this paper.

**REFERENCES**

1. Vojin G. Oklobdzija, VLSI Arithmetic, University of California, [Online] Available: http://www.ece.ucdavis.edu/acsel
2. BehroozParhami, .Computer Arithmetic, Oxford University Press, ISBN978-0-19-532848-6.
3. Karim Abbas, .Handbook of Digital CMOS Technology, Circuits and Systems, Springer, ISBN978-3-030-37194-4.
4. S. Majerski, .On determination of optimal distributions of carry skips in adders, IEEE Trans. Electron. Comput., pp. 45.58, Feb.1967.
5. SreehariVeeramachaneni, .Design of Efficient VLSI Arithmetic Circuits, International Institute of Information Technology, Hyderabad, June 2015. [Online] Available:<https://shodhganga.inflibnet.ac.in/handle/10603/45300>
6. T. Kilburn, D. B. G. Edwards, and D. Aspinall, .Parallel addition in digital computers: a new fast carry circuit,. in Proc. IEE, vol. 106, pt. B, Sept. 1959, p.464.
7. P. Kogge and H. Stone, .A parallel algorithm for the efficient solution of a general class of recurrencerelations, IEEE Trans. Computers, vol. C-22, no. 8, pp. 786.793, Aug. 1973.
8. R. P. Brent and H. T. Kung, .A regular layout for parallel adders,. IEEE Trans. Computers, vol. C-31, no. 3, pp. 260.264, Mar.1982.119
9. T. Han and D. Carlson, .Fast area-efficient VLSI Adders, in Proc. 8th Symp. Comp. Arith., Sept. 1987, pp.49.56.
10. R. Ladner and M. Fischer, .Parallel prefix Computation,. J. ACM, vol. 27, no. 4, pp. 831.838, Oct.1980.
11. Choi Youngmoon, .Prallel Prefix Adder Design, The University of Texas at Austin, 2004. [Online] Available:<https://repositories.lib.utexas.edu/handle/2152/1300>
12. S. Knowles, .A family of adders,. in Proc. 15th IEEE Symp. Comp. Arith., June 2001, pp.277.281.
13. Xilinx, .Vivado Design Suite User Guide, UG904 (v2020.1), August 2020. [Online] Available: [https://www.xilinx.com/support/documentation/sw\_manua](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug904-vivado-implementation.pdf) ls/xilinx2020\_1/ug904-vivado-implementation.pdf
14. AVNET, .ZedBoard Hardware User’s Guide, v2.2, January 2014. [Online] Available: http://zedboard.org/sites/default/files/documentations/Zed Board\_HW\_UG\_v2\_2.pdf