

# Static Power Comparison at Various Temperatures of Parallel Prefix Adder Based on FPGA

<sup>1</sup>Avinash shrivastava, <sup>2</sup>chandrahas sahu  
<sup>1</sup>Student, M.E. (VLSI Design), <sup>2</sup>Assistant Professor,  
<sup>1</sup>Department of Electronics & Communication,  
<sup>1</sup>SSCET (CSVTU), Bhilai, (CG), India

**Abstract**— Parallel-prefix structures (also known as carry tree) are found to be common in high performance adders in very large scale integration (VLSI) designs because of the delay is logarithmically proportional to the adder width. Such structures can usually be classified into three basic stages which are pre-computation, prefix tree and post-computation. However, this performance advantage does not translate directly into FPGA implementations due to constraints on logic block configurations and routing overhead. In this paper six types of carry-tree adders (the Kogge-Stone, Brent Kung, Han Carlson, Ladner, Fischer, Sklansky and Harris adder) investigates and compares them to the simple Ripple Carry Adder (RCA). These implementations have been successfully done in verilog hardware descriptive language using Xilinx Integrated Software Environment (ISE) 13.2 design suit. These designs are implemented in Xilinx Spartan 6 ,Spartan 6 low power, virtex 6, virtex 6 low power Field Programmable Gate Arrays (FPGA) and delays, area and power are measured using xpower analyzers 13.2 and all these adder’s Comparison of Slice utilization, No. of logic levels required & Delay are investigated and compared finally.

**Keywords :** Parallel prefix adders; carry tree adders; FPGA; logic analyzer; delay; power.

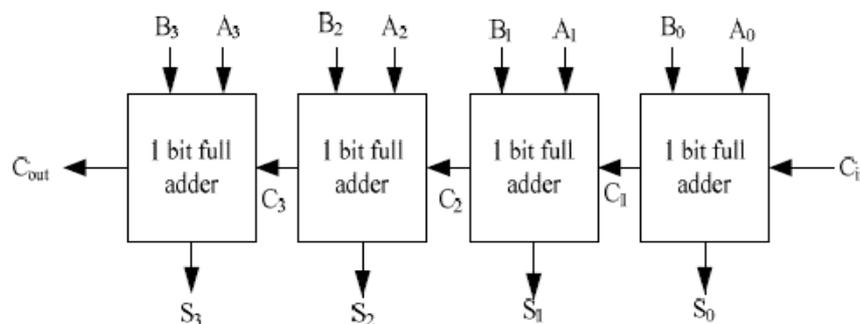
## INTRODUCTION

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.

Parallel Prefix Adder (PPA) is very useful in today’s world of technology because of its implementation in Very Large Scale Integration (VLSI) chips. The VLSI chips rely heavily on fast and reliable arithmetic computation. These contributions can be provided by PPA. There are many types of PPA such as Kogge Stone [1], Brent Kung [2], Ladner Fisher [3], Hans Carlson [4] and Knowles [5], Harris. For the purpose of this research, only Brent Kung and Kogge Stone adders will be investigated. Fig. 1 shows the structured diagram of a PPA. PPA can be divided into three main parts, namely the pre-processing, carry graph and post-processing. The pre-processing part will generate the propagate (*p*) and generate (*g*) bits. The acquirement of the PPA carry bit is differentiates PPA from other type of adders. It is a parallel form of obtaining the carry bit that makes it performs addition arithmetic faster.

In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are described. An efficient testing strategy for evaluating the performance of these adders is discussed. Several tree-based adder structures are implemented and characterized on a FPGA and compared with the Ripple Carry Adder (RCA)

## DRAWBACKS OF RIPPLE CARRY AND CARRY LOOKAHEAD ADDER



**Fig.2: 4 bit ripple carry adder**

In fig.2, the first sum bit should wait until input carry is given, the second sum bit should wait until previous carry is propagated and so on. Finally the output sum should wait until all previous carries are generated. So it results in delay. In order to reduce the delay in RCA (or) to propagate the carry in advance, we go for carry look ahead adder .Basically this adder works on two operations called propagate and generate The propagate and generate equations are given by.

$$P_i = A_i \oplus B_i \quad (1)$$

$$G_i = A_i \cdot B_i \quad (2)$$

For 4 bit CLA, the propagated carry equations are given as

$$C_1 = G_0 + P_0 C_0 \quad (3)$$

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0 \quad (4)$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \quad (5)$$

$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0 \quad (6)$$

Equations (3), (4), (5) and (6) are observed that, the carry complexity increases by increasing the adder bit width. So designing higher bit CLA becomes complexity. In this way, for the higher bit of CLA's, the carry complexity increases by increasing the width of the adder. So results in bounded fan-in rather than unbounded fan-in, when designing wide width adders. In order to compute the carries in advance without delay and complexity, there is a concept called Parallel prefix approach.

### DIFFERENCE BETWEEN PARALLEL-PREFIX ADDERS AND OTHERS

The PPA's pre-computes generate and propagate signals are presented in [2]. Using the fundamental carry operator (fco), these computed signals are combined in [3]. The fundamental carry operator is denoted by the symbol "o",

### PARALLEL PREFIX ADDER STRUCTURE

Parallel-prefix structures are found to be common in high performance adders because of the delay is logarithmically proportional to the adder width [2].

PPA's basically consists of 3 stages

- Pre computation
- Prefix stage
- Final computation

#### Pre computation

In pre-computation stage, propagates and generates are computed for the given inputs using the given equations (1) and (2).

#### Prefix stage

In the prefix stage, group generate/propagate signals are computed at each bit using the given equations. The black cell (BC) generates the ordered pair in equation (7), the gray cell (GC) generates only left signal, following [2].

$$G_{i:j} = G_{i:j} + G_{i:j} \cdot P_{j-1:k} \quad (10)$$

$$P_{i:k} = P_{i:j} \cdot P_{j-1:k} \quad (11)$$

More practically, the equations (10) and (11) can be expressed using a symbol "o" denoted by Brent and Kung. Its function is exactly the same as that of a black cell i.e.

$$G_{i:k} : P_{i:k} = (G_{i:j} : P_{i:j}) o (G_{j-1:k} : P_{j-1:k}) \quad (12)$$

The "o" operation will help make the rules of building prefix structures.

#### C. Final computation

In the final computation, the sum and carryout are the final output.

$$S_i = P_i \cdot G_{i-1:-1} \quad (12)$$

$$C_{out} = G_{n:-1} \quad (13)$$

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. Figure 3 shows the definitions of cells that are used in prefix structures, including BC and GC. For analysis of various parallel prefix structures, see [2], [3] & [4].

In the prefix tree, group generate/propagate signals are computed at each bit.

$$G_{i:j} = G_{i:j} + P_{i:j} \cdot G_{j-1:k}$$

$$P_{i:j} = P_{i:j} \cdot P_{j-1:k} \tag{15}$$

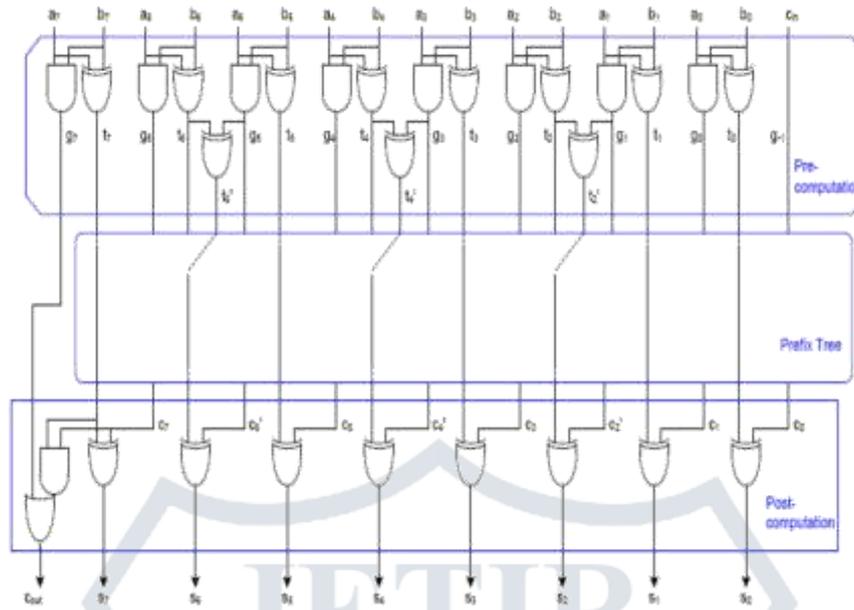


Fig. 3. Parallel-Prefix Structure with carry save notation

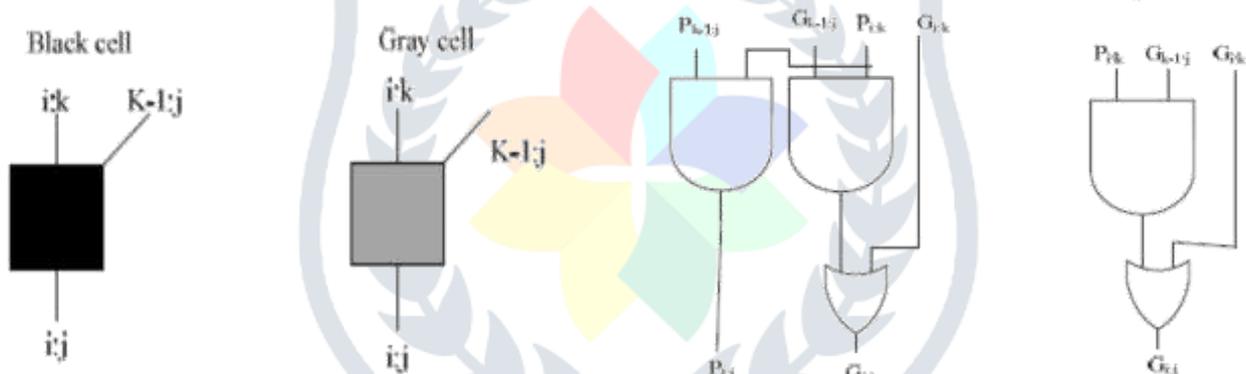


Fig. 4. Black and Gray Cell logic Definitions

**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 Figure 5. In fact, Kogge-Stone is a member of Knowles prefix tree. The 16-bit prefix tree can be viewed as Knowles [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 (3.2) according to the special 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 Figure 5 is 16-bit (a power of 2) prefix tree. It is not difficult to extend the structure to any width if the basics are strictly followed.

For the Kogge-Stone prefix tree, at the logic level 1, the inputs span is 1 bit (e.g. group (4:3) takes the inputs at bit 4 and bit 3). Group (4:3) will be taken as inputs and combined with group (6:5) to generate group (6:3) at logic level 2. Group (6:3) will be taken as inputs and combined with group (10:7) to generate group (10:3) at logic level 3, and so on so forth. With this inspection, the structure can be described with the Algorithm 6.1 listed below.

In Algorithm 6.1, the number of logic levels is calculated first. At each logic level, the maximum input bit span and maximum output bit span are computed. Equation (15) is applied in the inner loop, where bit goes from bit v-1 though bit n-1. If any of the subscript goes less than -1, the value stays at -1. This means there is no crossing over bit

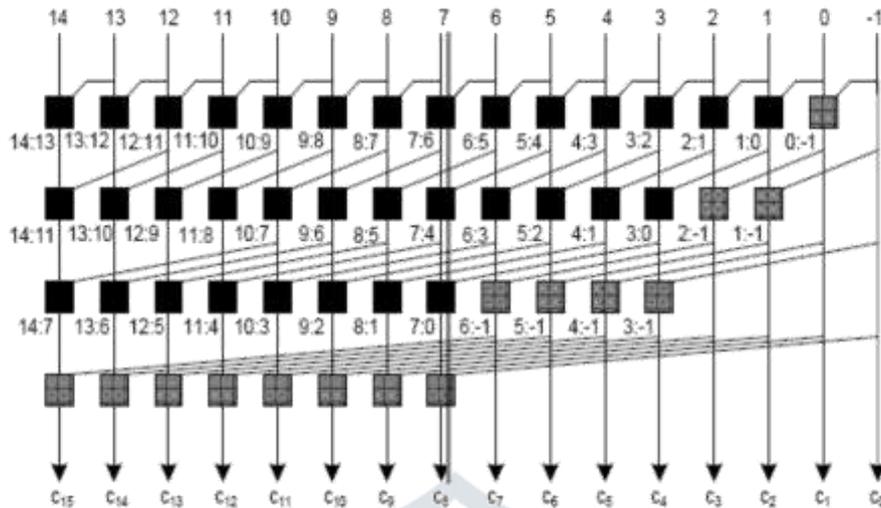


Figure 5: 16-bit Kogge-Stone Prefix Tree.

**Algorithm 6.1** Building Kogge-Stone Prefix Tree

```

L = log2(n);
for llevel = 1; llevel ≤ L; llevel ++ do
u = 2llevel; {output bit span}
v = 2llevel-1; {input bit span}
For i = v-1; i < n-1; i ++ do
G . Pi:i-u+1 = (G . Pi:i-v+1) ∘ (G . Pi-v:i-u+1);
end for
end for
    
```

**BRENT-KUNG PREFIX TREE**

Brent-Kung prefix tree is a well-known structure with relatively sparse network. The fan out is among the minimum as  $f = 0$ . So is the wire tracks where  $t = 0$ . The cost is the extra Level 1 logic levels. A 16-bit example is shown in Figure 6. The critical path is shown in the figure with a thick gray line. Brent-Kung prefix tree is a bit complex to build because it has the most logic levels. To build such a structure, the pseudo-code can be composed as Algorithm 7.1.

**Algorithm 7.1** Building Brent-Kung Prefix Tree

```

L = log2(n);
for llevel=1; llevel ≤ L; llevel++ do
u = 2llevel; {output bit span}
v = 2llevel-1; {input bit span}
for i = u- 2; i < n-1; i + = u do
GPi:i-u+1 = (GPi:i-v+1) ∘ (GPi-v:i-u+1);
end for
end for
for llevel = L-1, llevel ≥ 1; llevel – do
u = 2llevel; {output bit span}
v = 2llevel-1; {input bit span}
for i = u+v- 2; i < n-1; i + = u do
GPi:-1 = (GPi:i-v+1) ∘ (GPi-v:i-1);
end for
end for
    
```

-1 or the LSB boundary.

The statement in the inner for loop is applying Equation (3.2). The validity of this implementation can be verified by looking at Table 3.1. In the table, one group operation is randomly selected at each logic level. Other operations can be verified by inserting the numbers as listed in Table 3.1. The term  $GP_{i:i} = GP_i$  and LSB boundary of the inputs/outputs is bit 1. Table 3.1 can also be matched against Figure 3.8 to see the correspondence.

The pseudo-code is a simplified version of the exact program. In the real program, the code should tell where the black cells and gray cells are. The program also needs control so that the LSB never goes beyond 1 and utilizes optional buffers. In Figure 3.8, there are fan-outs more than 2 because the structure is not buffered. Figure 3.9 shows a buffered 16-bit pre x tree, however, the

exact number of buffers is based on the capacitance and resistance of the interconnect network [46]. Both figures indicate a wire track of 8.

**Table 1: Verifying the Pseudo-Code of Building a Kogge-Stone Prefix Tree**

Logic Levels	u	v	Output (i:i-u+1)	Input1 (i:i-v+1)	Input2 (i-v:i-u+1)	Equation Mapping
1	2	1	7:6	7:7	6:6	$GP_{7:6} = GP_7 \circ GP_6$
2	4	2	11:8	11:10	9:8	$GP_{11:8} = GP_{11:10} \circ GP_{9:8}$
3	8	4	14:7	14:11	10:7	$GP_{14:7} = GP_{14:11} \circ GP_{10:7}$
4	16	8	7:-1	7:0	-1:-1	$GP_{7:-1} = GP_{7:0} \circ GP_{-1}$

The algorithmic delay is simply the number of logic levels. The area can be estimated as the number of cells in the prefix tree. To simply the calculation, all cells are counted as black cells. To understand this structure, remember that the number of gray cells always equals to n - 1 since the prefix tree only outputs n - 1 carries. A black cell has one more AND gate than a gray cell, and therefore, a more accurate area estimation will just subtract that n 1 AND gates. The number cells for a Kogge-Stone prefix tree can be counted as follows. Each logic level has n m cells, where  $m = 2^{llevel-1}$ . That is, each logic level is missing m cells. That number is the sum of a geometric series starting from 1 to  $n/2$  which totals to n - 1. The total number of cells will be  $n \log_2 n$  subtracting the total number of cells missing at each logic level, which winds up with  $n \log_2(n) - n + 1$ . When n = 16, the area is estimated as 49.

**BRENT-KUNG PREFIX TREE**

Brent-Kung prefix tree is a well-known structure with relatively sparse network. The fan-out is among the minimum as f = 0. So is the wire tracks where t = 0. The cost is the extra L-1 logic levels. A 16-bit example is shown in Figure 3.13. The critical path is shown in the figure with a thick gray line. Brent-Kung prefix tree is a bit complex to build because it has the most logic levels. To build such a structure, the pseudo-code can be composed as Algorithm 8.1.

**Algorithm 8.1: Building Brent-Kung Prefix Tree**

```

L = log2(n);
for llevel=1; llevel ≤ L; llevel++ do
u = 2llevel; {output bit span}
v = 2llevel-1; {input bit span}
for i = u- 2; i < n-1; i + = u do
GPi:i-u+1 = (GPi:i-v+1) ∘ (GPi-v:i-u+1);
end for
end for
for llevel = L-1, llevel ≥ 1; llevel - - do
u = 2llevel; {output bit span}
v = 2llevel-1; {input bit span}
for i = u+v- 2; i < n-1; i + = u do
GPi:-1 = (GPi:i-v+1) ∘ (GPi-v:i-1);
end for
end for
    
```

The algorithm deals with this prefix tree in 2 major f or loops. The first f or loop handles logic level by logic level from 1 up to L with the second f or loop handling the rest L- 1 logic levels in a decremental fashion. Figure 6 can be divided as the top 4 logic levels and the bottom 3 logic levels. The structure starts with cells every 2 bits. The input span is 1 bit and the output span is 2 bits. At logic level 2 and 3, the distance between each cell is 4 and 8 bits, respectively. The input/output span is 2=4 bits at logic level 2 and 4=8 bits at logic level 3. At logic level 4, the only cell is at MSB (bit 14 from input) with input spanning 8 bits and output spanning 16 bits. By logic level 4, some carries are already generated. At logic level 5 through 7, the input bit span is decremented instead of being incremented as in the previous cases. The input bit spans at logic level 5 through 6 and 7 are 4, 2 and 1 bit, respectively. The term output span no longer applies to these L-1 levels since all the outputs are the final carries with

the form  $G_{i:-1}$ .

The delay is estimated as the number of logic levels (i.e. L). The total number of cells can be calculated in the following way. In the first  $\log_2 n$  logic levels, the number of cells is a geometric series. For example, in the 16-bit prefix tree, at logic level 1 through 4, there are 8, 4, 2, 1 cell at each level. The sum of this series is n-1. For the rest of the logic levels, there only exist gray cells. The total number of gray cells is n-1 for any prefix tree as mentioned before. However, in the previous  $\log_2 n$  logic levels, the pre x tree contains  $\log_2 n$  gray cells. The sum of cells is  $2(n-1) \log_2 n$ . When  $n = 16$ , the number of cells required is 26.

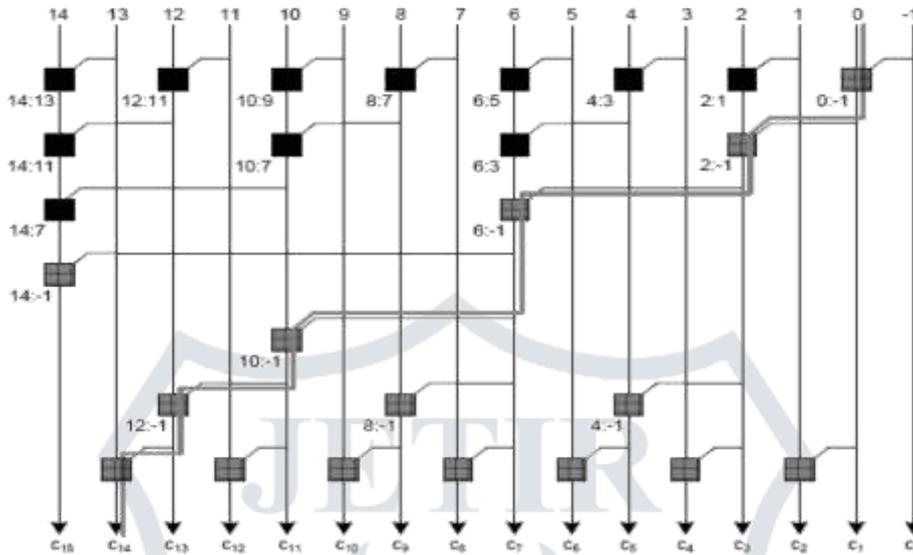


Figure 6: 16-bit Brent-Kung Prefix Tree.

**SKLANSKY PREFIX TREE**

Sklansky prefix tree takes the least logic levels to compute the carries. Plus, it uses less cells than Knowles [2,1,1,1] and Kogge-Stone structure at the cost of higher fan-out. Figure 7 shows the 16-bit example of Sklansky prefix tree with critical path in solid line. For a 16-bit Sklansky prefix tree, the maximum fan-out is 9 (i.e.  $f = 3$ ). The structure can be viewed as a compacted version of Brent-kung's, where logic levels are reduced and fan-out increased. A similar pseudo-code listed for Brent-Kung prefix tree can be used to generate a Sklansky prefix tree. However, the maximum input span is still a power of 2, relating with the number of logic levels. The difference is that one more f or loop is required to account for the multiple fan-out (e.g. at logic level 2 through 4 in Figure 7, where the cells are placed in group of 2,4 and 8, respectively).

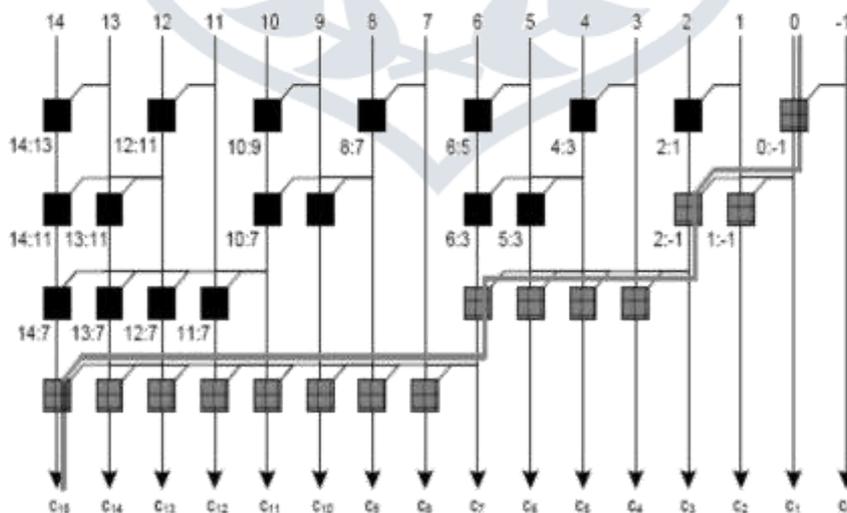
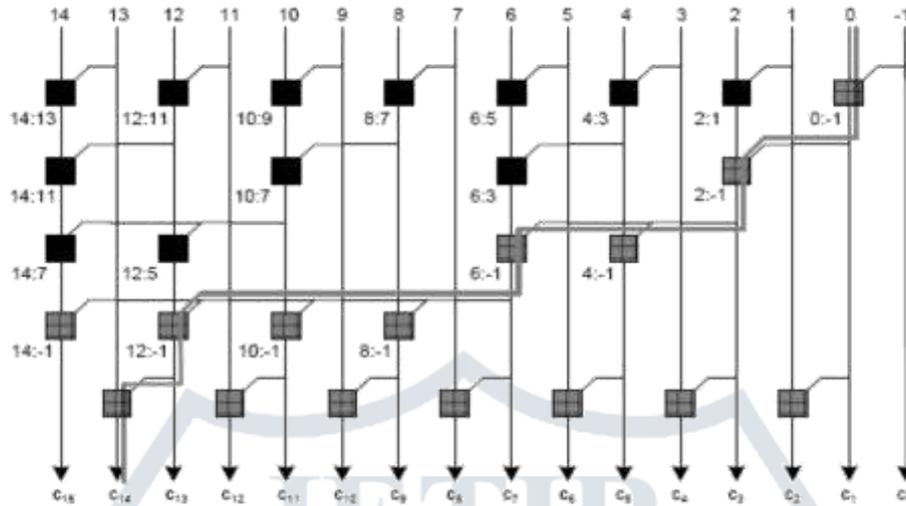


Figure 7: 16-bit Sklansky Prefix Tree.

The number of logic levels is  $\log_2 n$ . Each logic level has  $n/2$  cells as can be observed in Figure 7. The area is estimated as  $(n/2) \log_2 n$ . When  $n = 16$ , 32 cells are required.

**LADNER-FISCHER PREFIX TREE**

Sklansky prefix tree has the minimum logic levels, and uses fewer cells than Kogge-Stone and Knowles prefix trees. The major problem of Sklansky prefix tree is its high fan-out. Ladner-Fischer prefix tree is proposed to relieve this problem. To reduce fan-out without adding extra cells, more logic levels have to be added. Figure 8 shows a 16-bit example of Ladner-Fischer prefix tree.



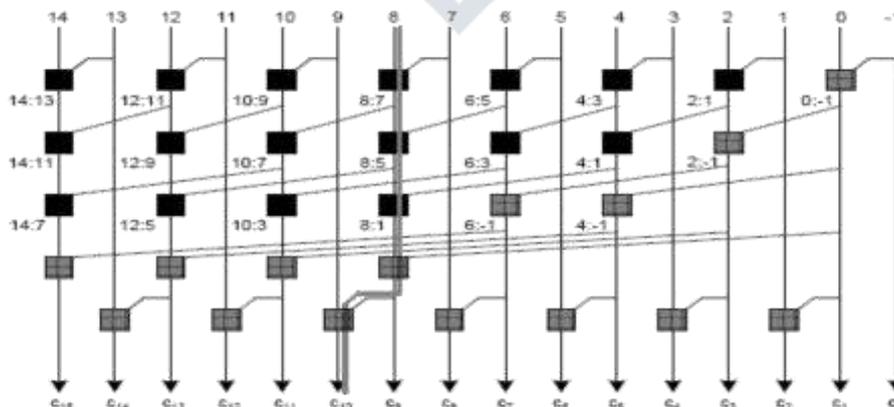
**Figure 8:11-bit Ladner-Fischer Prefix Tree Synthesis.**

Ladner-Fischer prefix tree is a structure that sits between Brent-Kung and Sklansky prefix tree. It can be observed that in Figure 8, the first two logic levels of the structure are exactly the same as Brent-Kung's. Starting from logic level 3, fan-out more than 2 is allowed (i.e.  $f > 0$ ). Comparing the fan-out of Ladner-Fischer's and Sklansky's, the number is reduced by a factor of 2 since Ladner-Fischer prefix tree allows more fan-out one logic level later than Sklansky prefix tree. Building a Ladner-Fischer prefix tree can be seen as a relieved version of Sklansky prefix tree. For a structure like Figure 8, a extra row of cells are required to generate the missing carries.

The delay for the type of Ladner-Fischer prefix tree is  $\log_2(n)$ . The first and last logic level takes  $n/2$  and  $n/2 - 1$  cells. In between, there are  $\log_2(n) - 1$  logic levels, each having  $n/4$  cells. Summing up the cells,  $n/2 + n/2 - 1 + (n/4)(\log_2(n) - 1)$  which is equal to  $(n/4)\log_2(n) + 3n/4 - 1$ . When  $n = 16$ , total cells required is 27.

**HAN-CARLSON PREFIX TREE**

The idea of Han-Carlson prefix tree is similar to Kogge-Stone's structure since it has a maximum fan-out of 2 or  $f = 0$ . The difference is that Han-Carlson prefix tree uses much less cells and wire tracks than Kogge-Stone. The cost is one extra logic level. Han-Carlson prefix tree can be viewed as a sparse version of Kogge-Stone prefix tree. In fact, the fan-out at all logic levels is the same (i.e. 2). The pseudo-code for Kogge-Stone's structure can be easily modified to build a Han-Carlson prefix tree. The major difference is that in each logic level, Han-Carlson prefix tree places cells every other bit and the last logic level accounts for the missing carries. Figure 9 shows a 16-bit Han-Carlson prefix tree, ignoring the buffers. The critical path is shown with thick solid line.



**Figure 9: 16-bit Han-Carlson Prefix Tree.**

This type of Han-Carlson prefix tree has  $\log_2 n + 1$  logic levels. It happens to have the same number cells as Sklansky prefix tree since the cells in the extra logic level can be move up to make the each of the previous logic levels all have  $n=2$  cells. The area is

estimated as  $(n=2) \log_2 n$ . When  $n = 16$ , the number is 32.

**HARRIS PREFIX TREE**

The idea from Harris about prefix tree is to try to balance the logic levels, fan-out and wire tracks. Harris proposed a cube to show the taxonomy for prefix trees in Figure 10, which illustrates the idea for 16-bit prefix trees [1]. In the figure, all the prefix trees mentioned above are on the cube, with Sklansky prefix tree standing at the fan-out extreme, Brent-Kung at the logic levels extreme, and Kogge-Stone at the wire track extreme.

The balanced prefix structure is close to the center of cube (i.e. when  $n = 16, l = 1, f = 1$  and  $t = 1$  or represented in short by  $(1; 1; 1)$ ). The logic levels is  $2^4 + 1 = 5$ , maximum fan-out is  $2^1 + 1 = 3$  and wire track is  $2^1 = 2$ . The diagram is shown in Figure 10 with critical path in solid line. Observation can be made that there is bit overlap in logic level 4 similar to Knowles [2; 1; 1; 1]. The overlap is valid for producing correct carries as it has been proven for Knowles [2; 1; 1; 1]. For  $n \geq 16$ ,  $(1; 1; 1)$  will not be sufficient to build a prefix tree. More logic levels, or fan-out, or wire tracks need to be added. For example, when  $n = 32$ , the prefix tree can be in the form of  $(1; 1; 2)$ ,  $(1; 2; 1)$  and  $(2; 1; 1)$ . Like Ladner-Fischer and Han-Carlson prefix tree illustrated in the previous sections, Harris prefix tree has  $\log_2(n) + 1$  logic levels. It needs the same number of cells required for Han-Carlson and Sklansky prefix tree, which is  $n/2 \log_2(n)$ .

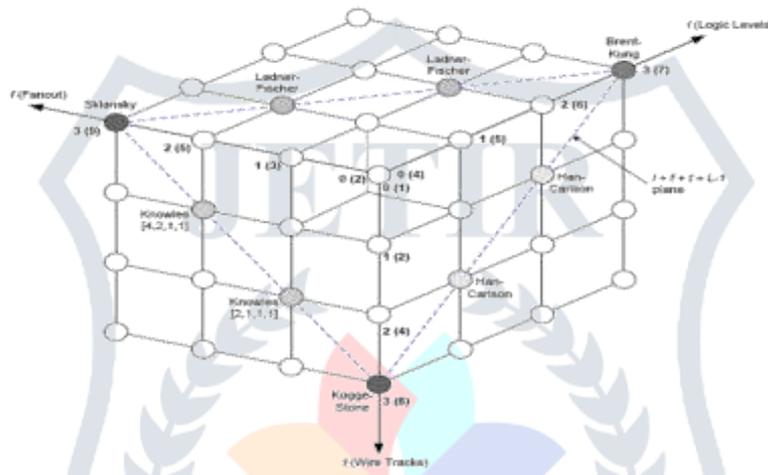


Figure10: Taxonomy of 16-bit Prefix Tree (Adapted from [1]).

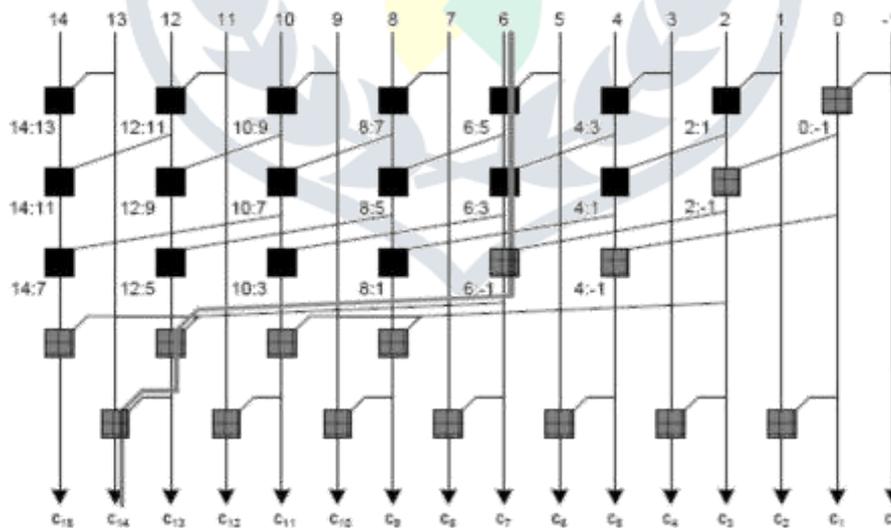
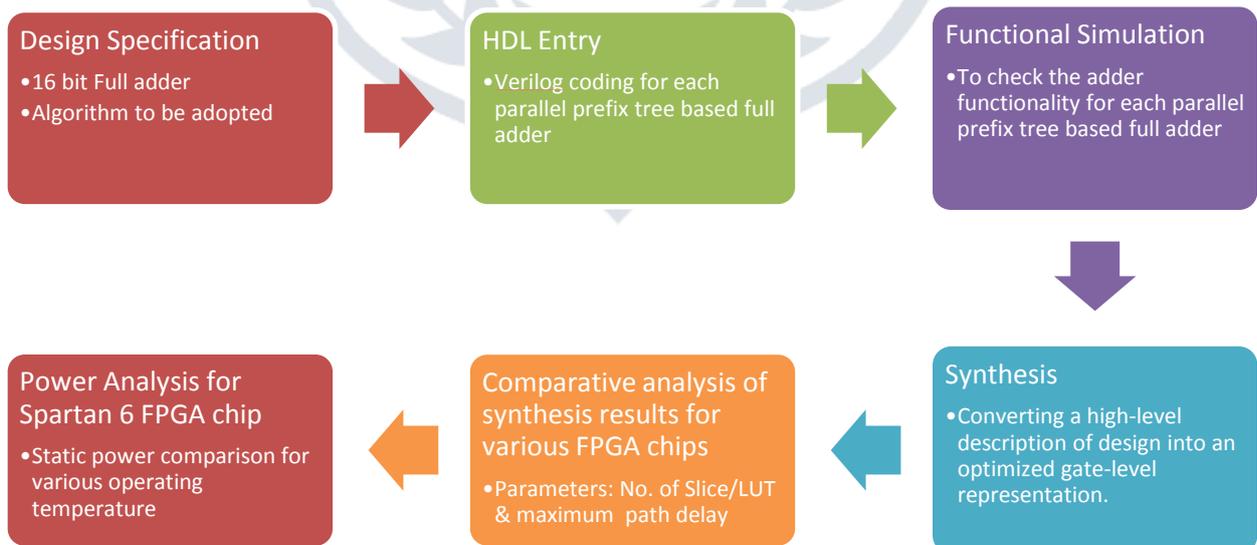
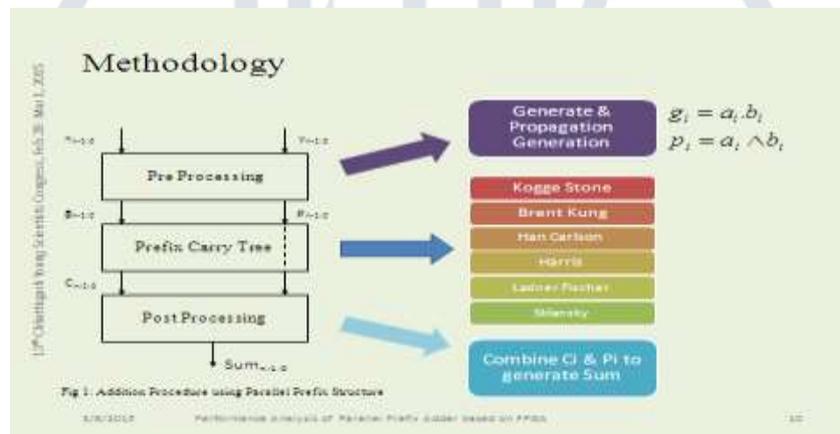


Figure 11: 16-bit Harris Prefix Tree

**TABLE 2: ALGORITHMIC ANALYSIS**

Types	Logic Levels	Area	Fan-out	Wire tracks
Brent-Kung	$2\log_2(n)$	$2n - \log_2 n - 2$	2	1
Kogge-Stone	$\log_2 n$	$n\log_2 n - n + 1$	2	$n/2$
Ladner-Fischer	$\log_2 n + 1$	$n/4 \log_2 n + 3n/4 + 1$	$n/4 + 1$	1
Sklansky	$\log_2 n$	$\left(\frac{n}{2}\right) \log_2 n$	$n/2 + 1$	1
Han-Carlson	$\log_2 n$	$\left(\frac{n}{2}\right) \log_2 n$	2	$n/4$
Harris	$\log_2 n$	$\left(\frac{n}{2}\right) \log_2 n$	3	$n/6$

**METHODOLOGY**



DISCUSSION OF RESULT

TABLE 3: COMPARISON OF SLICE UTILIZATION, NO. OF LOGIC LEVELS REQUIRED & DELAY

Adder name	Delay (ns)					
	No. of slice LUT	No. of logic level	Spartan 6	Spartan 6 low power	Vertex 6	Vertex 6 low power
Ripple carry	24	10	12.244	18.772	4.599	4.984
Kogge stone	43	10	11.935	18.316	4.629	4.995
Brent kung	26	9	11.447	17.506	4.240	4.564
Han-Carlson	26	10	12.145	18.445	4.469	4.871
Harris	39	8	10.306	15.248	3.512	3.821
Sklansky	28	10	12.555	19.404	4.939	5.279
Ladner Fischer	31	9	11.545	17.638	4.258	4.585

TABLE 4: STATIC POWER COMPARISON AT VARIOUS TEMPERATURES

Device: Spartan 6 XC6SLX45, Temp Grade: C-Grade,  $V_{ccint} = 1.2V$ ,  $V_{ccaux} = 2.5v$

Adder Name	Process Amb. Temp. 0°C		Process Amb. Temp. 25°C		Process Amb. Temp. 50°C		Process Amb. Temp. 75°C	
	Typical (Watt)	Max. (Watt)	Typical (Watt)	Max. (Watt)	Typical (Watt)	Max. (Watt)	Typical (Watt)	Max. (Watt)
Ripple Carry	0.022	0.049	0.037	0.088	0.068	0.175	0.137	0.404
Kogge Stone	0.022	0.049	0.037	0.088	0.068	0.175	0.137	0.405
Brent Kung	0.022	0.049	0.037	0.088	0.068	0.175	0.137	0.405
Han Carlson	0.022	0.049	0.037	0.121	0.068	0.175	0.137	0.405
Harris	0.022	0.050	0.038	0.090	0.069	0.180	0.406	0.483
Ladner Fischer	0.022	0.049	0.037	0.088	0.068	0.175	0.137	0.404
Sklansky	0.022	0.049	0.037	0.088	0.068	0.175	0.137	0.405

The delay observed for adder design from synthesis reports in Xilinx ISE 13.2 synthesis reports are shown in figure.

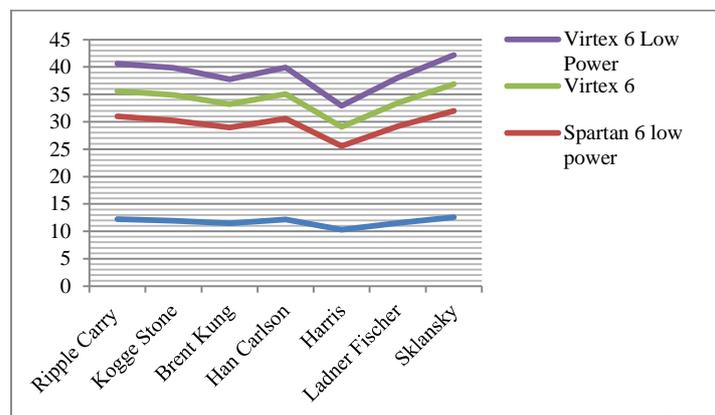
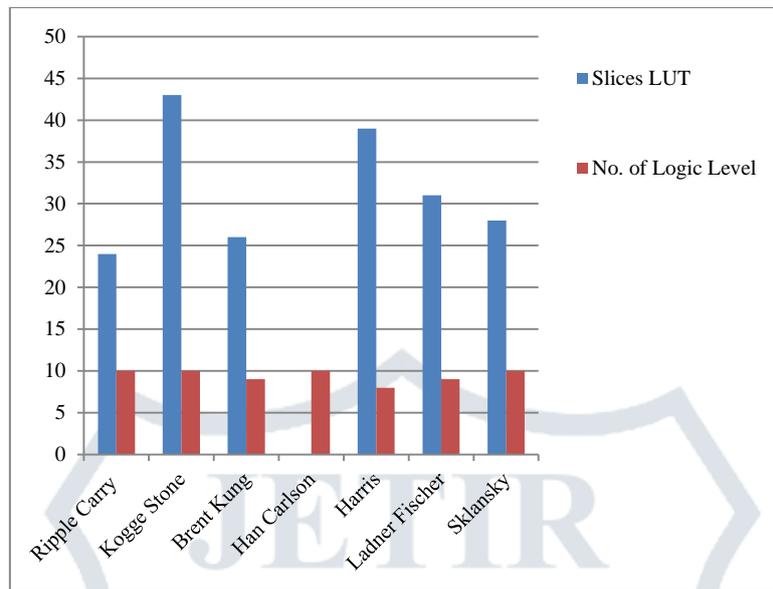


Fig. 11. Simulation results for the adder designs

The no. of slices LUT and no. of logic level observed for adder designs from synthesis reports in Xilinx ISE 13.2 are compared and shown in figure 12. The area of the adder designs is measured in terms of look up tables (LUT) and input output blocks (IOB) taken for Xilinx ISE 13.2 in Spartan 6 FPGA chip is plotted in the figure. As per reference [1], ISE software doesn't give exact delay of the adders because it is not able to analyze the critical path over the adder [1]. From the comparison table it is clear that Out of all adders, Harris adder has less delay. KSA adder and BKA have about the same delay.



**Fig: Comparative chart for Slice utilization**

According to the synthesis reports, out of four parallel prefix adders, Harris adder has better delay because of taking least logic level for compaction where as on the basis of delay and area (slice utilization or the no. of LUT required) Brent Kung is the best on an average.

## CONCLUSIONS

From the study of analysis done on delay, area and power, we have concluded that the efficiency is improved by 6.50 % in ours delay for RCA, when compared to Brent Kung [2], and for KSA it is improved by 2.53 % when compared with [1], and for Harris adder it is improved by 18.80 % in Spartan 6 FPGA chip. So we can say that Harris is the best because of taking least logic level for compaction where as on the basis of delay and area (slice utilization or the no. of LUT required) Brent Kung is the best on an average where as power analysis report shows that almost all of the parallel prefix adders takes more or less the same power.

## REFERENCES

- [1] D. Harris, .A taxonomy of parallel prefix networks,. in *Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, Nov. 2003, pp. 2213.2217.
- [2] D. Goldberg, .What every computer scientist should know about floating point arithmetic, . *ACM Computing surveys*, vol. 23, no. 1, pp. 5.48, 1991.
- [3] J. Chen and J. E. Stine, .Optimization of bipartite memory systems for multiplicative divide and square root,. *48th IEEE International Midwest Symp. Circuits and Systems*, vol. 2, pp. 1458.1461, 2005.
- [4] S. Winograd, .On the time required to perform addition, *J. ACM*, vol. 12, no. 2, pp. 277.285, 1965.
- [5] R. K. Richards, *Arithmetic Operations in Digital Computers*. D. Van Nostrand Co., Princeton, N.J., 1955.
- [6] A. Weinberger and J. Smith, .A logic for high-speed addition,. *National Bureau of Standards*, no. Circulation 591, pp. 3.12, 1958.
- [7] A. Tyagi, .A reduced area scheme for carry-select adders,. *IEEE Trans. Computers*, vol. 42, no. 10, pp. 1163.1170, Oct. 1993.
- [8] H. Ling, .High speed binary adder,. *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156.166, 1981.
- [9] 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.[10] P. Kogge and H. Stone, .A parallel algorithm for the efficient solution of a general class of recurrence relations,. *IEEE Trans. Computers*, vol. C-22, no. 8, pp. 786.793, Aug. 1973.
- [11] S. Knowles, .A family of adders, in *Proc. 15th IEEE Symp. Comp. Arith.*, June 2001, pp. 277.281.
- [12] J. Sklansky, .Conditional-sum addition logic,. *IRE Trans. Electronic Computers*, vol. EC-9, pp. 226.231, June 1960.
- [13] R. Ladner and M. Fischer, .Parallel prefix Computation, *J. ACM*, vol. 27, no. 4, pp. 831.838, Oct. 1980.
- [14] T. Han and D. Carlson, .Fast area-efficient VLSI Adders, in *Proc. 8th Symp. Comp. Arith.*, Sept. 1987, pp. 49.56.
- [15] A. Naini, D. Bearden, and W. Anderson, .A 4.5ns 96b CMOS adder design, in *Proc. IEEE Custom Integrate Circuits Conference*, vol. 38, no. 8, Apr. 1965, pp. 114.117.

- [16] 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.
- [17] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill, 1967.
- [18] W. K. Jenkins and B. J. Leon, .The use of residue number systems in the design of finite impulse response digital filters., *IEEE Trans. Circuits and Systems*, vol. 24, no. 4, pp. 171.201, Apr. 1977.
- [19] X. Lai and J. L. Massey, .A proposal for a new block encryption standard., in *Advances in Cryptology - EUROCRYPT'90, Berlin, Germany: Springer-Verilog*, 1990, pp. 389.404.
- [20] S. S.-S. Yau and Y.-C. Liu, .Error correction in redundant residue number systems., *IEEE Trans. Computers*, vol. C-22, no. 1, pp. 5.11, Jan. 1973.
- [21] F. Halsall, *Data Communications, Computer Networks and Open Systems*. Addison Wesley, 1996.
- [22] C. Efstathiou, D. Nikolos, and J. Kalamatianos, .Area-time efficient modulo  $2^n - 1$  adder design., *IEEE Trans. Circuits and System-II*, vol. 41, no. 7, pp. 463.467, 1994.
- [23] L. Kalamboukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. kalamatianos, .High-speed parallel-prefix modulo  $2n \pm 1$  adders., *IEEE Trans. Computers*, vol. 49, no. 7, special is sure on computer arithmetic, pp. 673.680, July 2000.
- [24] C. Efstathiou, H. T. Vergos, and D. Nikolos, .Fast parallel-prefix modulo  $2n + 1$  adders., *IEEE Trans. Computers*, vol. 53, no. 9, pp. 1211.1216, Sept. 2004.
- [25] H. T. Vergos, C. Efstathiou, and D. Nikolos, .Modulo  $2n - 1$  adder design using select-prefix blocks., *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1399.1406, Nov. 2003.
- [26] V. Paliouras and T. Stouraitis, .Novel high-radix residue number system multipliers and adders., in *Proc. 1999 IEEE Int'l Symp. Circuits and Systems VLSI (ISCAS '99)*, 1999, pp. 451.454.
- [27] S. Bi, W. J. Gross, W. Wang, A. Al-khalili, and M. N. S. Swamy, .An area-reduced scheme for modulo  $2^n - 1$  addition/subtraction., in *Proc. 9th International Database Engineering & Application Symp.*, 2005, pp. 396.399.
- [28] R. Zimmermann, Efficient VLSI implementation of modulo  $(2^n \pm 1)$  addition and multiplication, in *Proc. 14th IEEE Symp. Computer Arithmetic*, 1999, pp. p.158.167.
- [29] J. Chen and J. E. Stine, .Enhancing parallel-prefix structures using carry-save notation, *51st Midwest Symp. Circuits and Systems*, pp. 354.357, 2008.
- [30] G. E. Moore, .Cramming more components onto integrated circuits., in *Electronics*, May 1965, pp. 25.5.1.25.5.4.
- [31] M. Lehman and N. Burla, .Skip techniques for high-speed carry propagation in binary arithmetic units., *IRE Trans. Electron. Comput.* pp. 691.698, Dec. 1961.
- [32] O. J. Bedrij, .Carry-select adder., *IRE Trans. Electron. Comput.*, pp. 340.346, June 1962.
- [33] J. Sklansky, .Conditional sum addition logic., *IRE Trans. Electron. Comput.*, pp. 226. 231, June 1960.
- [34] V. G. Oklobdzija, B. Zeydel, H. Dao, S. Mathew, and R. Krishnamurthy, .Energy delay estimation technique for high-performance microprocessor VLSI adders., *Proc. 16th IEEE Symp. Computer Arithmetic (ARITH-16'03)*, p. 272, June 2003.
- [35] R. Zimmermann, .Binary adder architectures for cell-based VLSI and their synthesis., Ph.D. dissertation, ETH Dissertation 12480, Swiss Federal Institute of Technology, 1997.