Design and Comparative Analysis of Conventional Adders and Parallel Prefix Adders

K. Madhavi¹, Kuppam N Chandrasekar²

¹M.Tech scholar, GVIC, Madhanapally, A.P, India
²Assistant Professor, Dept. of ECE, GVIC, Madhanapally, A.P, India

Abstract — The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. In VLSI implementations, parallel-prefix adders are known to have the best performance. This paper investigates three types of PPA’s (Kogge Stone Adder (KSA), Spanning Tree Adder (STA) and Sparse Kogge Stone Adder (SKA)). Additionally Ripple Carry Adder (RCA), Carry look ahead adder (CLA), Carry select Adder (CSLA) and Carry Skip Adder (CSA) are also investigated. These adders are implemented in verilog Hardware Description Language (HDL) synthesis results are carried out using Xilinx Integrated Software Environment (ISE) Design Suite and simulation results are carried out using modelsim.

Keywords — Ripple carry adder; carry select adder; carry skip adder; parallel prefix adders; block cell; gray cell; Xilinx; modelsim.

I. INTRODUCTION

Binary adders are one of the most essential logic elements within a digital system. In addition, binary adders are also helpful in units other than Arithmetic Logic Units (ALU), such as multipliers, dividers and memory addressing. 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 have better performance. The delays of the adders are discussed [1].

In this paper, above mentioned PPA’s and RCA and CSA are implemented and characterized on a Xilinx Spartan3e FPGA. Finally, delay, power and area for the designed adders are presented and compared.

II. DRAWBACKS OF CONVENTIONAL ADDERS

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. Figure 1 demonstrates an example of an 8-bit binary add operation and how the carry chain is affected. 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.

Fig.1: Binary Adder Example.

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.

\[ P = A \oplus B \]  
\[ G = A \cdot B \]

Equations (3),(4),(5) and (6) are observed that, the carry complexity increases by increasing the adder bit width. So designing higher bit of 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.

III. DIFFERENCE BETWEEN PARALLEL-PREFIX ADDERS AND OTHERS

Parallel-prefix adders, also known as carry-tree adders, pre-compute the propagate and generate
signals. These signals are variously combined using the fundamental carry operator (fco).

\[(g_k, p_i) \oplus (g_k, p_k) = (g_k + p_i \cdot g_k, p_i \cdot p_k) \]  

Due to associative property of the fco, these operators can be combined in different ways to form various adder structures. For example, the four-bit carry-look ahead generator is given by:

\[c_4 = (g_2, p_2) \oplus \left[ (g_2, p_2) \oplus (g_1, p_1) \right] \]  

A simple rearrangement of the order of operations allows parallel operation, resulting in a more efficient tree structure for this four-bit example:

\[c_4 = [(g_2, p_2) \oplus (g_1, p_1)] \oplus [(g_2, p_2) \oplus (g_1, p_1)] \]  

Equations (8) and (9) are observed that, the carry look ahead adder takes 3 steps to generate the carry, but the bit PPA takes 2 steps to generate the carry.

IV. PARALLEL-PREFIX ADDER STRUCTURE

To resolve the delay of carry-look ahead adders, the scheme of multilevel-look ahead adders or parallel-prefix adders can be employed. The idea is to compute small group of intermediate prefixes and then find large group prefixes, until all the carry bits are computed. These adders have tree structures within a carry-computing stage similar to the carry propagate adder. However, the other two stages for these adders are called pre-computation and post-computation stages. In pre-computation stage, each bit computes its carry generate/propagate and a temporary sum. In the prefix stage, the group carry generate/propagate signals are computed to form the carry chain and provide the carry in for the adder below.

\[G_i:k = G_i:j + P_i:j \cdot G_j:1:k \]  
\[P_i:k = P_i:j \cdot P_j:1:k \]  

In the post-computation stage, the sum and carry-out are finally produced. The carry-out can be omitted if only a sum needs to be produced.

\[s_i = t_i \oplus G_i:1 \]  
\[cout = gn+1 + p_n+1 \oplus G_{n-2}:-1 \]

where \(G_i:1 = c_i\) with the assumption \(g-1 = c_i\). The general diagram of parallel-prefix structures is shown in Figure 2, where an 8-bit case is illustrated.

All parallel-prefix structures can be implemented with the equations above; however, Equation can be interpreted in various ways, which leads to different types of parallel-prefix trees. There are several design factors that can impact the performance of prefix structures.

- Radix/Valency
- Logic Levels
- Fan-out
- Wire tracks

![Fig.2: 8-bit Parallel-Prefix Structure with carry save notation.](image)

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. Such structures can usually be divided into three stages, pre-computation, prefix tree and post-computation. In the prefix tree, group generate/propagate are the only signals used. The group generate/propagate equations are based on single bit generate/propagate, which are computed in the pre-computation stage.

\[g_i = a_i \cdot b_i \]
\[p_i = a_i \oplus b_i \]

where \(0 < i < n\). \(g_{-1} = c_i\) and \(p_{-1} = 0\). Sometimes, \(p_i\) can be computed with OR logic instead of an XOR gate. The OR logic is mandatory especially when Ling’s scheme is applied. Here, the XOR logic is utilized to save a gate for temporary sum \(t_i\).

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

\[G_i:k = G_i:j + P_i:j \cdot G_j:1:k \]  
\[P_i:k = P_i:j \cdot P_j:1:k \]  

More practically, the above equation can be expressed using a symbol “\(\odot\)“ denoted by Brent and Kung. Its function is exactly the same as that of a black cell. That is

\[(G_i:k; P_i:k) = (G_i:j; P_i:j) \odot (G_j:1:k; P_j:1:k)\]

![Fig.3: Cell Definitions.](image)
Gi:k = (gi; pi) o (gi-1; pi-1) o …………o (gk; pk)
Pi:k = pi . pi-1 . …… . pk
The “o” operation will help make the rules of building prefix structures. In the post-computation, the sum and carry-out are the final output.
si = pi . Gi-1:
cout = Gn-1
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 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.

PREFIX TREE FAMILY
Parallel-prefix trees have various architectures. These prefix trees can be distinguished by four major factors. 1) Radix/Valency 2) Logic Levels 3) Fan-out 4) Wire Tracks 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 fanout, 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.

It is readily apparent that a key advantage of the tree structured adder is that the critical path due to the carry delay is on the order of \[\log2N\] for an N-bit wide adder. The arrangement of the prefix network gives rise to various families of adders. For this study, the focus is on the Kogge-Stone adder, known for having minimal logic depth and fanout (see Figure 4). Here we designate BC as the black cell which generates the ordered pair in equation (1); the gray cell (GC) generates the left 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. The regularity of the Kogge-Stone prefix network has built in redundancy which has implications for fault-tolerant designs

The sparse Kogge-Stone adder, shown in Figure 5, is also studied. This hybrid design completes the summation process with a 4 bit RCA allowing the carry prefix network to be simplified.

Another carry-tree adder known as the spanning tree carry-lookahead (CLA) adder is also examined [6]. Like the sparse Kogge-Stone adder, this design terminates with a 4-bit RCA. As the FPGA uses a fast carry-chain for the RCA, it is interesting to compare the performance of this adder with the sparse Kogge-Stone and regular Kogge-Stone adders. Also of interest for the spanning-tree CLA is its testability features [7].
These adders are implemented in verilog HDL in Xilinx ISE design suite and then verified using Xilinx Spartan3e FPGA through chip scope analyzer and simulation results are carried out using modelsim, the corresponding simulation results of the adders are shown below figures.

V. DISCUSSION OF RESULTS

The delays observed for adder designs from synthesis reports in Xilinx ISE synthesis reports are shown in Figure 13.
The area of the adder designs is measured in terms of look up tables (LUT) and input output blocks (IOB) taken for Xilinx Spartan3e FPGA is plotted in the figure 14.

![Image of LUTs and IOBs in Xilinx ISE](image_url)

**Fig.14:** LUT’s and IOB’s observed in Xilinx ISE

From the table 1, the second column gives the designed adder names for sixteen bit. The third column represents the delay observed in Xilinx ISE tool. The fourth and fifth columns represents area in terms of LUTs and IOBs observed in Xilinx ISE tool. The sixth column represents the power in milliwatts (mw) observed in Xpower analyzer in Xilinx ISE tool.

**TABLE I. Comparison of delay, LUTs and IOBs for adders**

<table>
<thead>
<tr>
<th>SL.NO</th>
<th>Adder Name (16 bit)</th>
<th>Delay (ns)</th>
<th>LUTs Out of 9312</th>
<th>IOBs Out of 232</th>
<th>Power (mw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ripple carry adder</td>
<td>24.686</td>
<td>32</td>
<td>50</td>
<td>80.98</td>
</tr>
<tr>
<td>2</td>
<td>carry select adder</td>
<td>17.501</td>
<td>41</td>
<td>50</td>
<td>80.98</td>
</tr>
<tr>
<td>3</td>
<td>carry skip addr</td>
<td>24.841</td>
<td>40</td>
<td>50</td>
<td>80.98</td>
</tr>
<tr>
<td>4</td>
<td>kogge stone adder</td>
<td>14.041</td>
<td>94</td>
<td>50</td>
<td>80.98</td>
</tr>
<tr>
<td>5</td>
<td>spheres kogge adder</td>
<td>17.527</td>
<td>57</td>
<td>65</td>
<td>80.98</td>
</tr>
<tr>
<td>6</td>
<td>spanning tree adder</td>
<td>21.72</td>
<td>39</td>
<td>65</td>
<td>80.98</td>
</tr>
</tbody>
</table>

The fourth and fifth columns give the area of the adder designs in terms of LUT’s and IOB’s. Out of all values, RCA has taken less number of LUT’s. Out of four PPA’s, KSA has less delay and has taken more LUT’s. Out of four PPA’s, STA has taken less area in terms of LUT’s. Out of all adders (mentioned), RCA has taken less area and CSA has more delay. The power for all mentioned adders is approximately 80.98 mill watts (mw)

VI. CONCLUSION

From the study of analysis done on area and power, we have concluded that the efficiency is improved by 56.87% in ours delay for RCA, when compared to KSA. The implementations that have been developed in this dissertation help to improve the design of parallel prefix adders and their associated computing architectures. This has the potential of impacting many application specific and general purpose computer architectures. Consequently, this work can impact the designs of many computing systems, as well as impacting many areas of engineers and science.

**REFERENCES**


