Analysis of Delay, Power and Area for Parallel Prefix Adders

SIKILIGIRI SHAHIN¹, G. NAVEENA LAKSHMI², K. UPENDRA RAJU³

¹PG Scholar, Dept of ECE, KLM Collage of Engineering for Women, Kadapa, YSR (Dt), AP, India.
²Asst Prof, Dept of ECE, KLM Collage of Engineering for Women, Kadapa, YSR (Dt), AP, India.
³HOD, Dept of ECE, KLM Collage of Engineering for Women, Kadapa, YSR (Dt), AP, India.

Abstract: High performance microprocessor units require high performance adders and other arithmetic units. Modern microprocessors are however 32-bits or 64-bits as that is the minimum required for floating point arithmetic as per the IEEE 754 Standard. 8-bit and 16-bit arithmetic processors are normally found in micro-controller applications for embedded systems where high speed is important but low power constraints dominate system design. A good metric of performance on such designs would be the power-delay product (or equivalently energy per bit.) Many designs give a high speed at the cost of more power or low power at the cost of low speed. In Very Large Scale Integration (VLSI) designs, Parallel prefix adders (PPA) have better delay performance. At present, the research continues on increasing the adder’s delay performance. In many practical applications like mobile and telecommunications, the Speed and power performance improved in FPGAs is better than microprocessor and DSP’s based solutions. Additionally, power is also an important aspect in growing trend of mobile electronics, which makes large-scale use of DSP functions. Because of the Programmability, structure of configurable logic blocks (CLB) and programming interconnects in FPGAs, Parallel prefix adders have better performance. This paper investigates two types of PPA’s, Brent Kung Adder (BKA) and Sparse Kogge Stone Adder (SKA). Additionally Ripple Carry Adder (RCA) are also investigated. These adders are implemented in Verilog Hardware Description Language (HDL) using Xilinx Integrated Software Environment (ISE) 14.1 Design Suite. These designs are implemented in Xilinx Vertex 5 Field Programmable Gate Arrays (FPGA) and all these adder’s delay, power and area are investigated and compared finally.

Keywords: Parallel Prefix Adders; Carry Tree Adders; FPGA; Delay; Power.

I. INTRODUCTION

The addition of two binary numbers is one of the most fundamental and important arithmetic function in modern digital systems such as microprocessors and digital signal processors. In these systems binary adders are used in arithmetic logic units (ALU), multipliers, dividers and memory address generation. The requirements of adders are that they should be fast and efficient in terms of power and chip area. Most often, the maximum operating speed of most of the modern digital systems depend on how fast adders can process the data and hence responsible for setting the minimum clock cycle time in processors. The major problem for binary addition is the propagation delay in the carry chain. As the width of the input operand increases, the length of the carry chain increases. To address the carry propagation problem, most of the modern adder architectures are represented as a parallel prefix adder (PPA) structure consisting of pre-processing, carry look-ahead and post processing sections. Parallel Prefix Adders have been established as the most efficient circuits for binary addition in digital systems. Their regular structure and fast performance makes them particularly attractive for VLSI implementation. The delay of a parallel prefix adder is directly proportional to the number of levels in the carry propagation stage. The performance metrics considered for the analysis of the adders are: power, delay and area. For performance comparison, the adders were realized using various prefix tree algorithms. Using simulation studies, delay, area and power performance of the various adder modules were obtained.

II. LIMITATIONS OF RIPPLE CARRY AND CARRY LOOKAHEAD ADDER

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. RCA is used to perform any number of additions. In this RCA is serial adder and it has propagation delay problem. With increase in h₉ & f₉ circuits, delay also increases simultaneously. In fig.1, 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. A fast method of adding numbers is called carry-look ahead adder. This method doesn't require the carry signal to propagate stage by stage, causing a bottleneck. Instead it uses additional logic to expedite the propagation and generation of carry information, allowing fast addition at the expense of more hardware requirements.

Copyright © 2015 IJVDCS. All rights reserved.
Rather than waiting for carry signals to ripple from the least significant to the most significant bit, CLA adders divided the inputs into groups of r bits and implement the logic equations to determine if each group will generate or propagate carry. Basically this adder works on two operations called propagate and generate and propagate equations are given by:

\[ P_i = A_i \oplus B_i \]  

\[ G_i = A_i \cdot B_i \]  

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

\[ C_1 = G_0 + P_2 C_0 \]  

\[ C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0 \]  

\[ C_3 = G_2 + P_2 G_1 + P_1 P_2 G_0 + P_2 P_1 P_0 C_0 \]  

\[ C_4 = G_3 + P_3 G_2 + P_3 P_1 G_1 + P_2 P_1 P_2 G_0 + P_1 P_2 P_1 P_0 C_0 \]  

From the above equations it is observed that, the carry complexity increases by increasing the adder bit width. The carry look ahead shows good performance for the lower order bits that is it shows better performance for sum of less number of bits. The 32-bit carry look ahead adder shows better performance than 64-bit carry look ahead adder. Even the 64-bit adder shows better performance than 128 bit adder. As the bit size increases the performance decreases in the carry look ahead adder. 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. 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 employs 3- stage structure of carry look ahead adder. The improvement is in the carry generation stage which is the most intensive one. The biggest difference between the full adder and parallel prefix adder is that in the full adder, summation and carry calculation is done in the same one bit block but in the prefix adder, summation and carry calculation are separated from the bit block and all calculation is treated as a whole in the carry graph. The carry graph uses the prefix circuit and this is the origin of the name, “Prefix Adder”. 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”.

\[ (g_L \cdot p_L) \cdot o (g_R \cdot p_R) = (g_L + p_L \cdot g_R \cdot p_R) \cdot o (g_R \cdot p_R) \]  

For example, 4 bit CLA carry equation is given by

\[ C_4 = (g_4 \cdot p_4) \cdot o [(g_3 \cdot p_3) \cdot o [(g_2 \cdot p_2) \cdot o (g_3 \cdot p_3)]] \]  

For example, 4 bit PPA carry equation is given by

\[ C_4 = [(g_4 \cdot p_4) \cdot o (g_3 \cdot p_3)] \cdot o [(g_4 \cdot p_4) \cdot o (g_3 \cdot p_3)] \]  

From the above equation it is 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

Parallel-prefix structures are found to be common in high performance adders because of the delay is logarithmically proportional to the adder width. PPA’s basically consists of 3 stages

- Pre computation
- Prefix stage
- Final computation

The Parallel-Prefix Structure is shown in fig.2.

![Parallel-Prefix Structure with carry save notation.](image)
Analysis of Delay, Power and Area for Parallel Prefix Adders

\[ P_{t-k} = P_{r-j} \cdot P_{j-1-k} \] \hspace{1cm} (11)

More practically, the equations above can be expressed using a symbol “o”. Its function is exactly the same as that of a black cell i.e.

\[ G_{i,k} : P_{i,k} = (G_{i,j} : P_{i,j}) \circ (G_{j-1,k} : P_{j-1,k}) \] \hspace{1cm} (12)

\[ S_i = P_i \cdot G_{i-1} \] \hspace{1cm} (13)

\[ C_{out} = G_{n-1} \] \hspace{1cm} (14)

Where “-1” is the position of carry-input.

C. Final Computation

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

For analysis of various parallel prefix structures, see [2], [3] & [4]. The sparse Kogge-Stone adder(SKAn) consists of several smaller ripple carry adders (RCAs) 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 and the gray cell remains exactly the same as in the regular Kogge-Stone adder. The 16 bit SKA uses black cells and gray cells as well as full adder blocks too. This adder computes the carry inputs using the BC’s and GC’s and terminates with 4 bit RCA’s. Totally it uses 16 full adders. The 16 bit SKA is shown in fig.4. In this adder, the input bits (a, b) are converted as propagate and generate (p, g). Then propagate and generate terms are given to BC’s and GC’s. The carries are propagated in advance using these cells. Later these are given to full adder blocks.

Fig. 3. Black and Gray Cell logic Definitions.

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

Fig. 4. 16-bit sparse kogge-Stone adder.

The 16 bit BKA is shown in the below fig.5.

Fig. 5. 16-bit brent kung adder.
Brent-Kung adder is a very well-known logarithmic adder architecture that gives an optimal number of stages from input to all outputs but with asymmetric loading on all intermediate stages. Thus rise and fall times at all nodes are not the same. To overcome this drawback, buffer stages were inserted between each stage of the adder and the logic between the buffer stages was optimized. The input pre-computation blocks generate Generate ($G_i$) and Propagate ($P_i$) signals from adder inputs $A_i$ and $B_i$. The intermediate dot product blocks take 2 inputs ($G_i$, $P_i$) and generate 2 primed $G_{i}^{\prime}$ and $P_{i}^{\prime}$ outputs equivalent to generate and propagate signals for individual blocks. BKA occupies less area than SKA. It uses limited number of propagate and generate cells than the other 3 adders. It takes less area to implement than the KSA and has less wiring congestion. The operation of the 16 bit brent kung adder is given above in fig.6. This adder uses less BC’s and GC’s than kogge stone adder and has the better delay performance.

![Fig.6. Simulation result of adder design in Xilinx ISE.](image)

V. CONCLUSION

From the study of analysis done on area and power, we have concluded that the efficiency is improved by 5.77 % in ours delay for RCA, when compared to [1] and for KSA it is improved by 19.28 % when compared with [1].

VI. REFERENCES