Research Article Impact Factor: 0.621 ISSN: 2319-507X
Vishal Naik, IJPRET, 2014; Volume 2 (9): 413-424 IJPRET

INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

DESIGN OF A CARRY TREE ADDER

VISHAL R. NAIK[1], SONIA KUWELKAR[2]

1. Microelectronics (ETC dept.), Goa College of Engineering, Goa University, India.
2. Assistant Professor (ETC dept.), Goa College of Engineering, Goa University, India

Accepted Date: 27/02/2014 ; Published Date: 01/05/2014

Abstract: The core of every microprocessor and digital signal processor is its data path. The heart of data-path and addressing units in turn are arithmetic units which include adders. Parallel-prefix adders offer a highly efficient solution to the binary addition problem and are well suited for VLSI implementations. This paper investigates four types of carry-tree adders the Brent-Kung, Kogge-Stone, the Slansky and Han-Carlson adder) and compares them to the simple Ripple Carry Adder (RCA). Due to the presence of a fast carry-chain, the RCA designs exhibit better delay performance up to 128 bits. The carry-tree adders are expected to have a speed advantage over the RCA as bit widths approach 256.

Keywords: Power-Delay Performance, Area, Carry Tree Adders

PAPER-QR CODE

Corresponding Author: MR. VISHAL R. NAIK

Access Online On:

www.ijpret.com

How to Cite This Article:

Vishal Naik, IJPRET, 2014; Volume 2 (9): 413-424

INTRODUCTION

Addition is one of the simplest and commonly used operations and is in most cases a speed determining factor for arithmetic operations. The addition of two binary numbers is the fundamental arithmetic operation in microprocessors, digital signal processors, and data-processing application-specific integrated circuits. Several algorithms have been presented for high speed parallel addition, and there is generally a trade off between speed and area. Hence, binary adders are crucial building blocks in very large-scale integrated circuits.

For the optimization of speed in adders, the most important factor is carry generation. For the implementation of a fast adder, the generated carry should be driven to the output as fast as possible, thereby reducing the worst path delay which determines the ultimate speed of the

digital structure. In the design for timing optimization, a network can be optimized either at circuit or logic level. Logic level optimization is done by manipulating boolean equations, whereas circuit level optimization can be carried out by manipulating transistor sizes and circuit topologies. In the optimization for area, care should be taken in the design of the building blocks of the structure, which determine the area occupied by the architecture and, finally, also effect the speed.

However, because of the structure of the configurable logic and routing resources in FPGAs, parallel-prefix adders will have a different performance than VLSI implementations.

In particular, most modern FPGAs employ a fast-carry chain which optimizes the carry path for the simple Ripple Carry Adder (RCA)[4]. In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are

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

I. Carry tree adder designs

2.2) Brent Kung adder architecture[3]

In order to approach the structure known as the Brent Kung Structure, which uses the logarithmic concept, the entire architecture is easily understood by dividing the system into three separate stages:

1. Generate/Propagate Generation

2. The Dot (•) Operation

3. Sum generation

Generate/Propagate Generation

If the inputs to the adder are given by the signals A and B, then the generate and propagate signals are obtained according to the following equations.

G = A.B (2.1)

P = A⊕B (2.2)

The Dot (•) Operation

The most important building block in the Brent Kung Structure is the dot (•) operator. The basic inputs to this structure are the generate and propagate signals generated in the previous stage. The •operator is a function that takes in two sets of inputs-- (g, p) and (g', p')-- and generates a set of outputs-- (g + pg', pp').

These building blocks are used for the generation of the carry signals in the structure. For the generation of the carry signals, the carry for the kth bit from the carry look-ahead concept is given by

Co,k = Gk +Pk(Gk-1 +Pk-1 +P k-1 (...+P1(G0 +P0 Ci,0))) (2.3)

Using the dot operator explained above the Equation 2.3 can be written for the different carry signals as

Co,0 = G0 +P Ci,0 = • (G0,P0)

Co,1 = G1 + G0 P1 = • ((G1 , P1)•• (G0, P0))

.....

C0,k = •((Gk,Pk)•• (Gk-1,Pk-1)••...•• (G0,P0) – (2.4)

where ••is a function defined in order to access all the tuples. The 8-bit Brent Kung Structure is shown in Figure 2.1. This figure shows all the carry signals generated at different stages in the structure. In the structure, two binary tree structure are represented -- the forward and the reverse trees. The forward binary tree alone is not sufficient for the generation of all the carry

signals. It can only generate the signals shown as Co,0,Co,1,Co,3 and Co,7. The remaining carry signals are generated by the reverse binary tree.

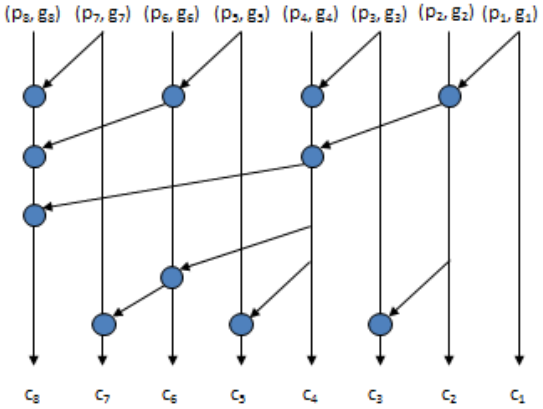

Fig.2.1. 8-bit Brent-Kung adder tructure[8]

Sum generation

The final stage in this architecture is the sum generation stage. The sum is given by

$S = A \oplus B \oplus C$ (2.5)

where A and B are the input signals, and C is the carry signal. The carry is obtained from the dot operator stage discussed earlier, and the exclusive of A and B is actually the propagate signal itself. Hence the sum 'S' can finally be represented and realized as

$S = P \oplus C$ (2.6)

2.2) Kogge-Stone adder architecture

The Kogge-Stone structure has a more optimal implementation, as its fan-out is greatly reduced to 2 at the expense of larger "o" (circle) cells. It is obtained by copying the of the most significant bit position. Figure 2.2 shows this carry tree for 4-bit operands. The 8-bit Kogge-Stone structure is shown in Figure 2.2. Recursive division of the blocks can construct full adder using such a tree for the implementation. The number of "o" cells required for the implementation is $n(\log_2 n-1)+1$ and the delay is $\log_2 n$,where n is the adders width. It is expected that Kogge-Stone adder should consume more resources than Any other adder. [5]

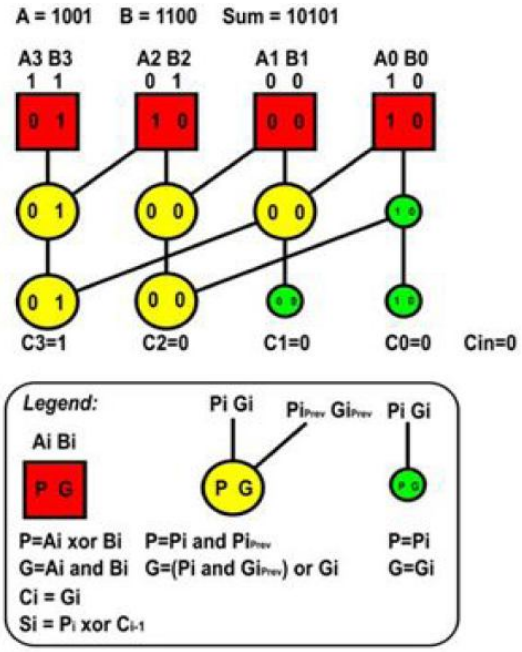

Fig.2.2. 8-bit Kogge-Stone adder structure.[9]

2.3) J. Sklansky-conditional adder architecture

Fig 2.3 shows the J.Sklansky conditional sum addition of two binary coded numbers.

We represent by $\tau_0, \tau_1, \tau_2, \tau_3, \tau_4$, the successive time intervals during which conditional sums and carries are computed. During τ_0 two sum and carry pairs for each column are computed. One pair under the assumption (assumption A_0) that the carry brought to each column is 0, and the other pair under the assumption (assumption B_0) that the carry brought to each column is 1. An exception to this is at column 0, where only A_0 is assumed, since the carry brought to that column is known to be 0.

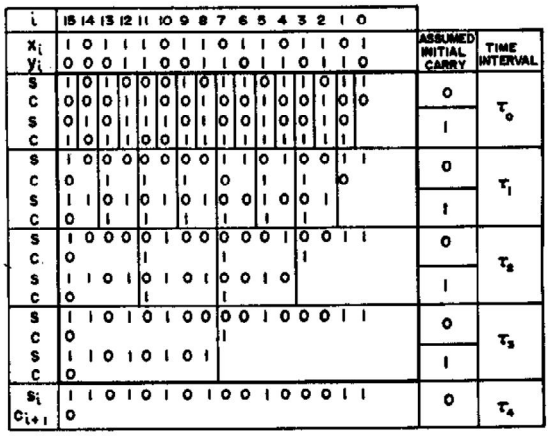

Fig.2.3. Conditional sum addition. [7]

The first row belonging to τ_0 in Fig.2.3 contains the conditional sum under the assumption A_0, the second row has the conditional carries under the same assumption, and the third and the fourth rows contain the conditional sums and carries under assumption B_0. In particular, consider column 0. Here we have, using the well-known Boolean relations for sum and carry bits,

$S_0^0 = X_0 \oplus Y_0 = 0 \oplus 1 = 1$

$C_1^0 = X_0 . Y_0 = 0.1 = 0$

$S_1^0 = X_1 \oplus Y_1 = 1 \oplus 0 = 1$

$C_2^0 = X_1 . Y_1 = 1.0 = 0$

$S_1^1 = S_1^0 \oplus 1 = 0$

$C_2^1 = X_0 V Y_0 = 1 V 0 = 1$

In a similar manner we find the other entries in the τ_0 columns.

During τ_1 conditional sums and carries for pairs of columns (namely, column numbers 0 and 1, 2 and 3, ..., and 14 and 15)are computed simultaneously, under the assumption (A_1) that carry brought to each pair of columns is 0, and (B_1) that this carry is 1.

The rows belonging to τ_1 are arranged in a manner similar to those of τ_0.

Continuing this process. The conditional sums and final carries are computed for tetrads of columns during τ_3, and for the entire sixteen column group during τ_4 the sums and final carry produced are no longer conditional but are equal to the sum bits and the final carry for the two summands, Xi and Yi. The 8-bit J. Sklansky-conditional adder structure is shown in Figure 2.4

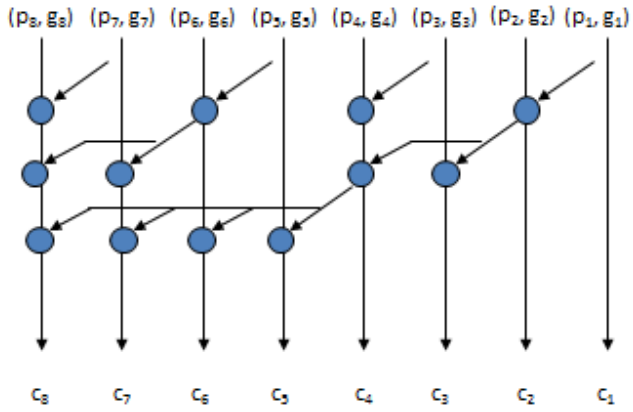

Fig.2.4.8-bit J.Sklansky adder structure. [8]

2.4) Han - Carlson adder architecture.

Like Brent and Kung's scheme, Han and Carlson also proposed a scheme to reduce the complexity of prefix tree[18]. This scheme is different from Kogge-Stone scheme in the sense that these performs carry-merge

operations on even bits and generate/propagate operation on odd bits. At the end, these odd bits recombine with even bits carry signals to produce the true carry bits. This adder has five stages in which the middle three stages are resembles with the Kogge-Stone structure. The advantage of this adder is its shorter span wires than the Kogge-Stone adder and thus there is a reduction in complexity at the cost of an additional stage for carry-merge path. [10]

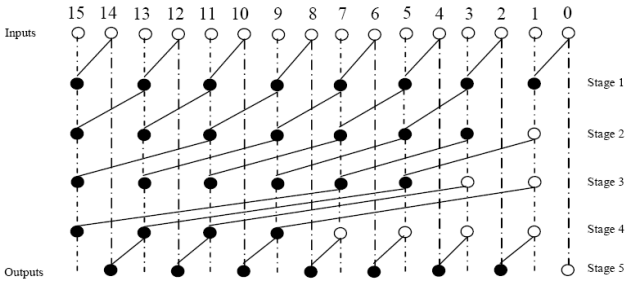

Fig.2.5. 16-bit Han-Carlson adder structure.

III) Related work

It is noted that delay models and cost analysis for adder designs developed for VLSI technology do not map directly to FPGA designs. This study focuses on carry-tree adders implemented on a Xilinx Spartan 3 FPGA. The distinctive contributions of this paper are two-fold. First, we consider tree-based adders and a hybrid form which combines a tree structure with a ripple-carry design. The adders which are chosen for analysing are Brent-kung adder, Kogge-Stone adder, J.Slansky adder and Han-Carlson adder. Second, this paper considers the practical issues involved in testing the adders and provides actual measurement data to compare with simulation results.

IV.RESULTS AND DISCUSSION

The various adder designs are simulated using ModelSim simulator. Simulation results for Ripple carry adder, Brent-Kung adder, Kogge-Stone adder and Sklansky adder are as follows

4.1) Ripple carry adder

Description-

a=8bit input (00111011); b=8bit input (11001010)

sum =8bit output (11110101); c=0; cout=1

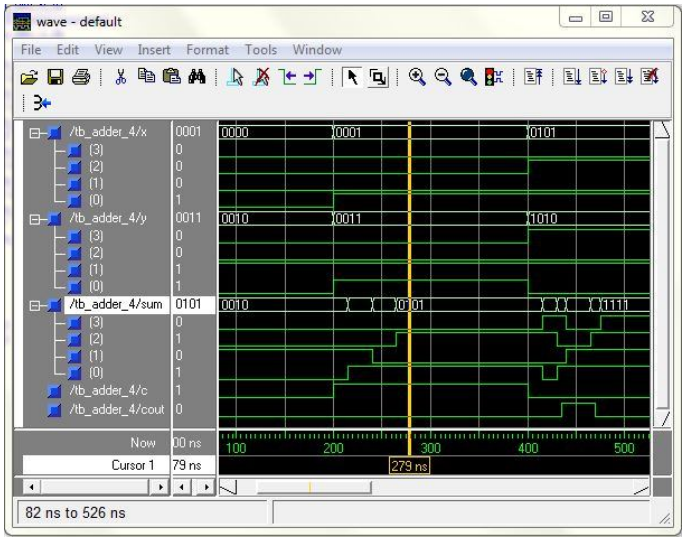

Fig.4.1. Simulation result of Ripple carry adder

4.2) Brent-Kung adder

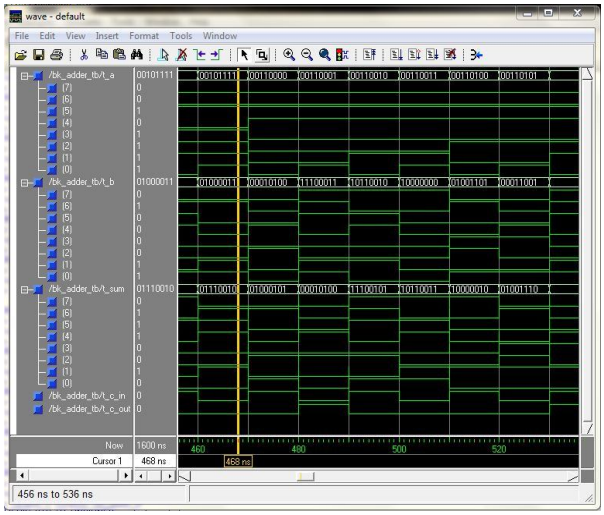

Fig.4.2. Simulation results of Brent-Kung adder

Description-

a=8bit input (00101111); b=8bit input (01000011)

sum =8bit output (01110010); c=0; cout=0

4.3) Kogge-Stone adder

Description-

a=8bit input (00111011); b=8bit input (11001010)

sum =8bit output (11110101); c=0; cout=1

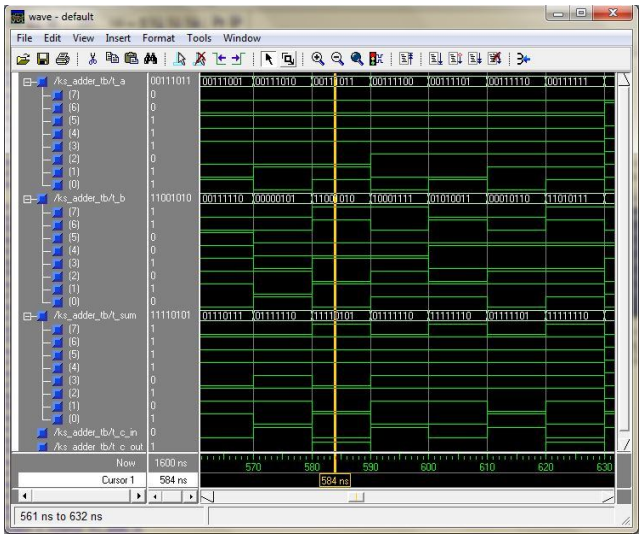

Fig.4.3. Simulation result of kogge-Stone adder

4.4) J. Sklansky adder

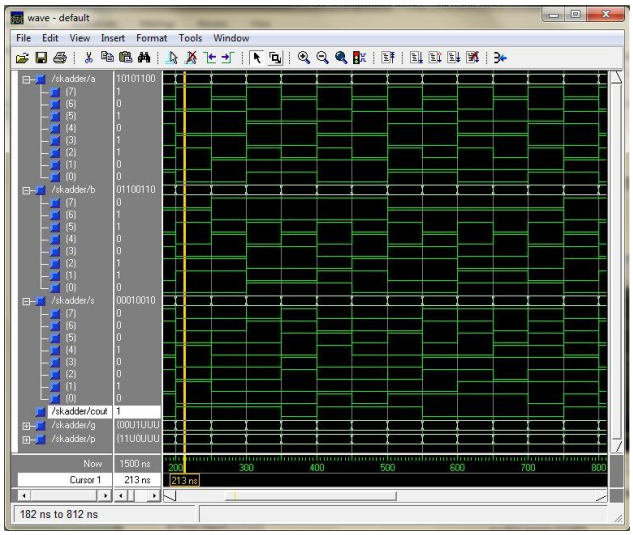

Fig.4.4. Simulation result of 8-bit J.Slansky adder

Description-

a=8bit input (10101100); b=8bit input (01100110)

sum =8bit output (00010010); cout=1

Fig.4.1, fig.4.2, fig.4.3 and fig.4.4 shows simulation results of carry tree adders which are implemented in generic. All the codes are written in VHDL language and simulated in ModelSim simulator successfully.

Further work:

These designs are implementing on FPGA and finding path delay, cell area and power on Xilinx FPGA. Because the adder is often the critical element which determines to a large part the cycle time and power dissipation for many digital signal processing and cryptographical implementations, it would be worthwhile for future FPGA designs to include an optimized carry path to enable tree-based adder designs to be optimized for place and routing. This would improve their performance similar to what is found for the RCA. We plan to explore possible FPGA architectures that could implement a "fast-tree chain" and investigate the possible trade-offs involved.

REFERENCES

1. Nurdiani Zamhari, Peter Voon, Kuryati Kipli, Kho Lee Chin, Maimun Huja Husin ,"Comparison of Parallel Prefix Adder (PPA)", Proceedings of the World Congress on Engineering 2012 Vol II

2. M.Venkata Subbaiah, M. Ravi Kishore," Design and Characterization of Parallel Prefix Adders" National Conference On Electrical Sciences -2012 (NCES-12)

3. R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.,* vol. C-31, pp. 260-264, 1982.

4. Neil H.E.Weste,David Harris and Ayan Banerjee ,CMOS VLSI Design(pearson education Asia, third edition, 2006.)

5. P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. on Computers*, Vol. C-22, No 8, August 1973.

6. Harris, "A Taxonomy of Parallel Prefix Networks", IEEE, 2003

7. J. Sklansky, "Conditional-Sum Addition Logic", IRE transactions on computers, 1960.

8. Parallel prefix adders, KostasVitoroulis, 2006. Presented to Dr. A. J. Al-Khalili. Concordia University.

9. Wikimedia Commons

10. Han, Carlson, "Fast Area-Efficient VLSI Adders, IEEE, 1987.

11. K. Vitoroulis and A. J. Al-Khalili, "Performance of Parallel Prefix Adders Implemented with FPGA technology," *IEEE Northeast Workshop on Circuits and Systems*, pp. 498-501, Aug. 2007.

12. G.Jyoshna, P.Murali Krishna, B.Doss, "Parallel-Prefix Adder Architecture With Efficient Timing-Area Characteristic", UACEE International journal of