INTERNATIONAL JOURNAL OF TECHNO-ENGINEERING
IJTE

ISSN: 2057-5688

Reducing the Hardware Complexity of a Parallel Prefix Adder

K.SIVAKUMAR[1], G.LAKSHMI BHARATH[2], B.UMAKANTH[3]
[1]PG Student, Dept of ECE, SITS, Kadapa, AP, India.
[2]Assistant Professor, Dept of ECE, SITS, Kadapa, AP, India.
[3]Assistant Professor, Dept of ECE, SV College of Engineering, AP, India.

Abstract— Currently, parallel prefix adders (PPA) are considered effective combinational circuits for performing the binary addition of two multi-bit numbers. These adders are widely used in arithmetic-logic units, which are parts of modern processors, such as microprocessors, digital signal processors, etc. This paper deals with Kogge-Stone adder, which is one of the fastest PPA. When performing the schematic implementation, this adder has a large hardware complexity. Therefore, in this work for reducing its hardware complexity the scheme of modified PPA has been developed. The performance parameters considered for the comparative analysis of the presented adders are: the number of logic gates, Quine-complexity and maximum delay obtained when schematic modeling in CAD environment Quartus II based on FPGA Altera EP2C15AF484C6. As a result, when simulation of 32-bit adder, Kogge-Stone adder and modified PPA have similar maximum delay. However modified PPA has reduced hardware complexity by 22.5% compared to Kogge-Stone adder.

1. INTRODUCTION

Basically there are two types of adders, Serial adders(which performs addition bit by bit and if speed is not the constraint, a cost effective option is to use serial adders). Another type which is parallel adders performs operation parallel that is adding bits simultaneously. The basic adders like Ripple carry adders, in which full adders are connected serially, due to which carry propagation forms the longest path leading to large delay and Carry Lookahead adders in which delay is less than that of ripple carry adders as carry is generated Before hand, but they consumes a lot of area as circuit sharing is not there. The performance of adder and thus overall circuit is very important for the design as technology keeps on advancing. So, by changing the carry generation structure of carry Lookahead adder by parallel prefix trees we can improve the performance. In this paper Kogge-Stone adder is investigated, which is one of the known effective fastest PPA. Kogge-Stone is widely and efficiently used. Such an adder has minimum delay while performing the binary addition. However, for estimation of hardware costs this adder has a great number of logic gates and Quine-complexity used in the schematic implementation. Therefore, in the present work for reducing its hardware complexity a modified parallel prefix adder is developed. Then, the comparison of the two presented adders is made by the following parameters: the number of logic gates, Quinecomplexity, as well as the delay obtained by simulation in Quartus II CAD environment based on FPGA Altera EP2C15AF484C6. Perspective architecture is proposed for schematic implementation of various PPA. And derivation of the formulas is also described for computing the hardware characteristics which are dependent on the bit width of input operands of the presented adders.

2. EXISTING SYSTEM

Carry Save Adder

A carry save adder is so called because the carry is saved at the individual stages and latter computed in the end. In fact the result from each addition issplit into two parts i.e half-sum bit and a carry bit. The half sum bits and carry bit are not combined until very end. In the end a ripple carry adder is used totake care of all the carry bits. Carry save adders are commonly used for high speed and less delay, where they generally are able to operate faster than "ripple carry" adders because a carry save adder does not completely perform the relatively time-exhaustive process of combining carries with sum bits between successive additions in the multiplication process but instead defer it until the final cycle of the operation. The whole motivation lies in the fact that the carry is delayed until the very end and the signals don't have to move farther. This helps in a smaller delay in comparison to ripple carry adder. n-bit carry save adder can be implemented by using n full adders by using the following
Techniques:
1. Use a ripple carry adder.
2. Add 0 at the beginning (MSB) of the sum array after first stage.
3. Shift the carry array left by one bit.
We have utilized the ripple carry adder designed in the previous chapter to design carry save

INTERNATIONAL JOURNAL OF TECHNO-ENGINEERING
IJTE
ISSN: 2057-5688

adder. Note, that there are three inputs to the carry save adder. This is the reason for area and delay to be in comparison with ripple carry adder. Refer to block diagram in Fig.1. The idea can be scaled to n-bit.

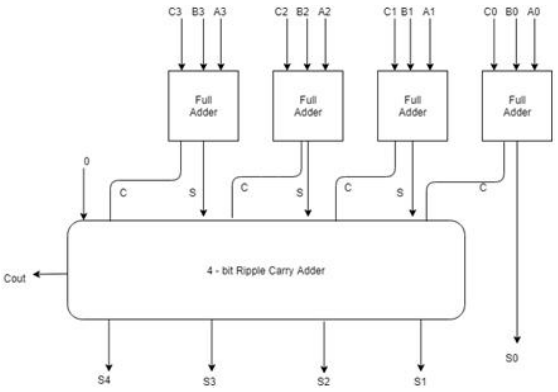

Figure 1: 4-bit Carry Save Adder

The test-bench used are same for all the adders in the library. Care must be taken to incorporate right data types when using more than 128 bit adders. Statistics for 8, 16, 32 and 64 bits are shown in the table below.

Table 1: Statistics: Carry Save Adder

Statistics: Carry Save Adder				
	8-bits	16-bits	32-bits	64-bits
Delay (ps)	61.22	86.66	127.05	248.21
AND2_X1	8	12	29	61
AND3_X2	0	1	0	0
AOI21_X1	5	21	49	98
AOI21_X2	0	0	15	
INV_X1	4	37	78	155
INV_X2	0	0	2	5
NAND2_X1	24	18	50	82
NOR2_X1	18	57	119	247
NOR3_X1	0	1	6	6
OAI21_X1	0	8	39	71
XNOR2_X1	0	1	0	0
OR2_X1	0	0	3	3
XOR2_X1	22	34	58	122
Number of cells	81	190	448	880

3. PROPOSED DESIGN

Parallel Prefix Adders

Parallel prefix adders are most important because of the speed at which they operate. The sum of n-bit number can be computed in time O(log n). This reduction in time is achieved due to its use of a tree network known as prefix operation graph. The reduction in time helps in addition of wider word lengths. A block diagram for parallel prefix adder is shown in 2. Every parallel prefix adder can be designed using three stages as described in the figure 6.1. The first stage is simple half adder. The core of the parallel prefix adder is the prefix graph that propagates the carry to the final stages. An example of the graph is show in Fig. 6.2. In the prefix operation graph, each node is a basic logical circuit described as prefix operation.

The goal of addition is to compute the sum, S, of two operands A and B,both of which are binary words of length n. For n-bit addition, the first stage of the adder computes the generate (G) and propagate (P) terms for each bit of the operands according to the following equations:

Gi = Ai AND BiPi = Ai XOR Bi

Stage 2 consists of the basic prefix operation, pref, is defined as follows:

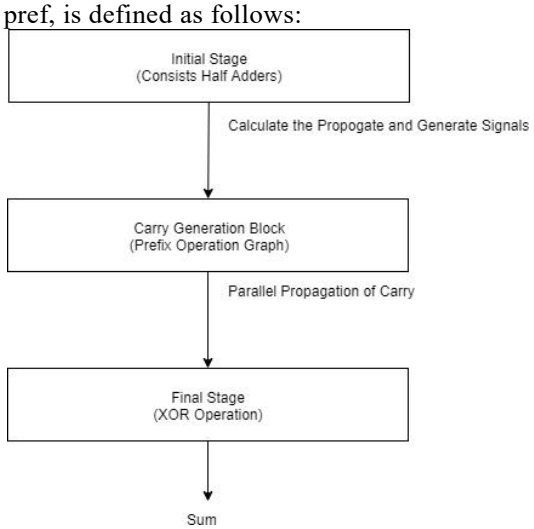

Figure 2: Parallel Prefix Block Diagram

(Gi,Pi) pref (Gj,Pj) = (Gi + Pi . Gj , Pi.Pj)

In the above equation, + refers to logical OR and . refers to logical AND.

In the end, the carry is equal to Gi's and sum is calculated by XOR with initial propagate which is the final stage.

We have designed two parallel prefix adders:

1. Kogge-Stone Adder
2. Han-Carlson Adder

P.M. Kogge and H.S. Stone were he first to use the property of commutativityand design parallel prefix adders where the computation of the prefixes is considered to be a recurrence that can be performed in parallel. The Kogge-Stone computation uses log2n stages, where n is the number of bits in the operands. Han-Carlson adder is a hybrid of Kogge-Stone and another parallel prefix adder i.e Brent-Kung. Kogge-Stone takes log2n stages and the Brent-Kung construction takes 2log2n-1 stages. Han-Carlson adder takes less area for the combinational circuits as compared to Kogge-Stone design. Each prefix tree consist of the some basic building blocks such as prefix_op (Bigger Circle), Square Box, Buffer and Diamond (Last stage XOR). Prefix tree graph for 16-bit Kogge stone is shown in Fig.3 .

INTERNATIONAL JOURNAL OF TECHNO-ENGINEERING
IJTE
ISSN: 2057-5688

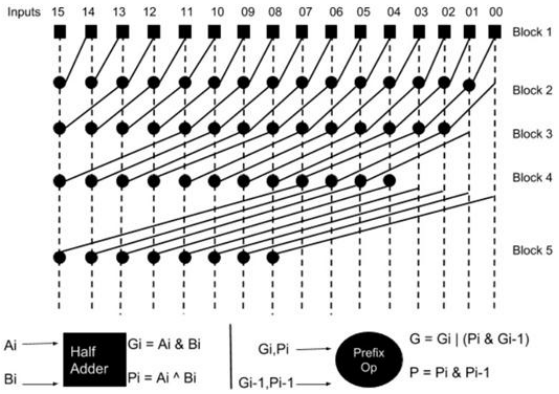

Figure 3: 16-bit Kogge Stone Prefix Graph
ABC Results: First Stage Square Operation
Gates = 3 Cap = 0.9 ff Area = 0.69 Delay = 4.70 ps

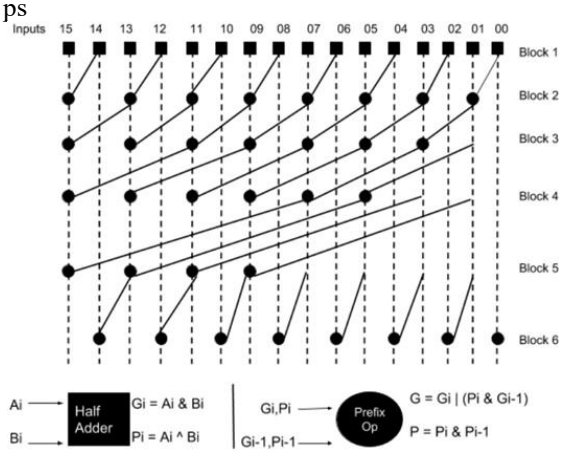

Figure 4: 16-bit Han Carlson Prefix Graph

Table 2: ABC Results: Carry Save Adder

ABC Results: Building Blocks for Prefix Adder			
	Diamond	Prefix Operation	Square Opera-tion
Delay (ps)	4.28	3.08	4.70
NAND2_X1 cells:	0	2	0
AND2_X1 cells:	0	1	1
BUF_X2 cells:	0	1	1
INV_X1 cells:	0	1	0
XOR2_X1 cells:	1	0	0
NOR2_X1 cells:	0	0	2
internal signals:	0	1	0
input signals:	2	4	2
output signals:	1	2	2

Table 3: Statistics: Kogge Stone Adder

Statistics: Kogge Stone Adder				
	8-bit	16-bit	32-bit	64-bit
Delay (ps)	31.97	39.49	48.07	57.01
Number of wires	14	16	18	20
Number of wire bits	98	226	514	1154
Number of public wires	14	16	18	20
Number of public wirebits	98	226	514	1154
Number of memories	0	0	0	0
Number of memory bits	0	0	0	0
Number of processes	0	0	0	0
Number of cells	37	92	219	506
Buffer	7	15	31	63
Diamond	8	16	32	64
Pref_Operation	14	45	124	315
Square_Operation	8	16	32	64

Table 4: Statistics: Han Carlson Adder

Statistics: Han Carlson Adder		
	16-bit	32-bit
Delay (ps)	77.28	62.67
Number of wires	18	20
Number of wire bits	258	578
Number of public wires	18	20
Number of public wirebits	258	578
Number of memories	0	0
Number of memory bits	0	0
Number of processes	0	0
Number of cells	107	250
Buffer	47	111
Diamond	16	32
Pref_Operation	28	75
Square_Operation	16	32

4. RESULTS

Results of all the modules are tabulated below along with the bits, delay and number of cells for each.

Table 5: Results: Adders

Results: Adders			
		Delay(ps)	Number of Cells
Ripple Carry Adder	8-bit	48.59	40
	16-bit	86.82	76
	32-bit	120.21	150
	64-bit	240.48	294
Carry Save Adder 3 - Inputs	8-bit	61.22	81
	16-bit	86.66	190
	32-bit	127.05	448
	64-bit	248.21	880
Kogge-Stone Adder	8-bit	31.97	59
	16-bit	39.49	168
	32-bit	48.07	454
	64-bit	57.01	1100
Han-Carlson Adder	16-bit	77.28	96
	32-bit	62.67	286

Table 6: Results

ISSN: 2057-5688

Results			
		Delay(ps)	Number of Cells
Bit-Shift Left	8-bit	32.46	89
	16-bit	45.39	310
	32-bit	58.38	1259
	64-bit	67.98	4739
Bit-Shift Right	8-bit	33.75	89
	16-bit	50.90	358
	32-bit	59.06	1308
	64-bit	72.56	4935
Bit-Shift Rotate Left	8-bit	32.15	135
	16-bit	42.94	572
	32-bit	57.19	2354
	64-bit	64.64	9214
Bit-Shift Rotate Right	8-bit	32.15	135
	16-bit	42.94	572
	32-bit	57.19	2354
	64-bit	64.64	9214
Radix-4 Booth Multi-plier	8-bit	118.16	665
	16-bit	258.49	2624
	32-bit	561.12	11093
	64-bit	1171.70	45050

CONCLUSION

A base has been created by creating this library where the modules can be used as per the required number of bits. An engineer would not need to create this modules redundantly and can be directly instantiated while designing complex modules. Current modules include Binary Adders, Parallel Prefix Adder, Shifters and Radix-4 Booth Multiplier. All the modules are designed successfully and are synthesizable. Extensive synthesis results are attached and also available at the repository. In future, we can continue to grow our library as per required by adding more modules to it.

References

1) R. P. Brent and H. T. Kung. A regular layout for parallel adders. IEEE Trans. Comput., 31(3):260–264, March 1982.
2) Leininger Joel Calvin and Taylor George Phillips. Carry save adder.
3) T. Han and D. A. Carlson. Fast area-efficient vlsi adders. In 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH), pages 49–56, May 1987.
4) P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Transactions on Computers, C-22(8):786–793, Aug 1973.
5) M. Morris Mano. Digital Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 2001.
6) S. Muthyala Sudhakar, K. P. Chidambaram, and E. E. Swartzlander. Hybrid han-carlson adder. In 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS), pages 818–821, Aug 2012.
7) Neil Weste and David Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Addison-Wesley Publishing Company, USA, 4th edition, 2010.