IJECEC INTERNATIONAL JOURNAL OF ELECTRICAL AND COMPUTER COMMUNICATION ENGINEERING
ISSN (Online):2533-8945
VOLUME 3 ISSUE 6

IMPLEMENTATION OF SPECULATIVE AND KOGGE-STONE ADDER USING HAN-CARLSON TOPOLOGY

[#1]V. SOUBHAGYA SREE
M.Tech(VLSI&ES),
Department of ECE,
Vasireddy Venkatadri Institute of Technology.

[#2]G. NAVEEN KUMAR
Assistant Professor(PH.D),
Department of ECE,
Vasireddy Venkatadri Institute of Technology.

Abstract—Variable latency adders have been recently proposed in literature. A variable latency adder employs speculation: the exact arithmetic function is replaced with an approximated one that is faster and gives the correct result most of the time, but not always. The approximated adder is augmented with an error detection network that asserts an error signal when speculation fails. Speculative variable latency adders have attracted strong interest thanks to their capability to reduce average delay compared to traditional architectures. This paper proposes a novel variable latency speculative adder based on Han-Carlson parallel-prefix topology that resulted more effective than variable latency Kogge-Stone topology. The paper describes the stages in which variable latency speculative prefix adders can be subdivided and presents a novel error detection network that reduces error probability compared to previous approaches. Several variable latency speculative adders, for various operand lengths, using both Han-Carlson and Kogge-Stone topology. Obtained results show that proposed variable latency Han-Carlson adder outperforms both previously proposed speculative Kogge-Stone architectures and non-speculative adders, when high-speed is required. It is also shown that non-speculative adders remain the best choice when the speed constraint is relaxed.

Index Terms—Addition, digital arithmetic, parallel-prefix adders, speculative adders, speculative functional units, variable latency adders.

I. INTRODUCTION

To humans, decimal numbers are easy to comprehend and implement for performing arithmetic. However, in digital systems, such as a microprocessor, DSP (Digital Signal Processor)or ASIC (Application-Specific Integrated Circuit), binary numbers are more pragmatic for a given computation. This occurs because binary values are optimally efficient at representing many values.

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.

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 a computer architecture, because they tend to set the critical path for most computations.

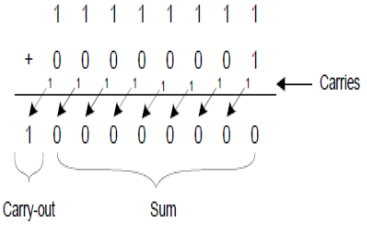

Figure 1: Binary Adder Example.

The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. As such, extensive research continues to be focused on improving the power delay performance of the adder. In VLSI implementations, parallel-prefix adders are known to have the best performance. Reconfigurable logic such as Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs involving mobile DSP and telecommunications applications and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs.

The power advantage is especially important with the growing popularity of mobile and portable electronics, which make extensive use of DSP

ISSN (Online):2533-8945
VOLUME 3 ISSUE 6

functions. However, because of the structure of the configurable logic androuting 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).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) and the Carry Skip Adder (CSA). Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given.

Carry-Propagate Adders

Binary carry-propagate adders have been extensively published, heavily attacking problems related to carry chain problem. Binary adders evolve from linear adders, which have a delay approximately proportional to the width of the adder, e.g. ripple-carry adder (RCA),to logarithmic-delay adder, such as the carry-lookahead adder (CLA). There are some additional performance enhancing schemes, including the carry-increment adder and the Ling adder that can further enhance the carry chain, however, in Very Large Scale Integration (VLSI) digital systems, the most efficient way of offering binary addition involves utilizing parallel-prefix trees, this occurs because they have the regular structures that exhibit logarithmic delay.

II. ADDERS

Half Adder: As mentioned previously, adders in VLSI digital systems use binary notation. In that case, add is done bit by bit using Boolean equations. Consider a simple binary add with two n-bit inputs A;B and a one-bit carry-in cin along with n-bit output S.

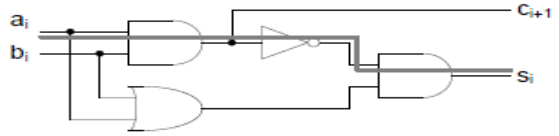

Figure 2: 1-bit Half Adder.

Considering the situation of adding two bits, the sum s and carry c can be expressed using Boolean operations mentioned above.

$$si = ai \wedge bi$$
$$ci+1 = ai.bi$$

Full Adder: Equation of ci+1 can be extended to perform full add operation, where there is a carry input.

$$si = ai \wedge bi \wedge ci$$
$$ci+1 = ai .bi + ai. ci + bi . ci$$

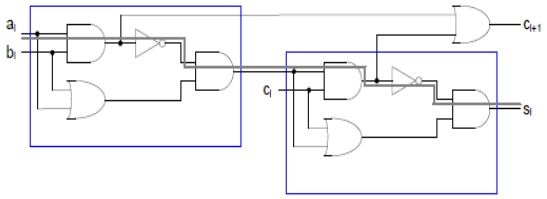

Figure 3: 1-bit Full Adder.

A full adder can be built based on Equation above. The block diagram of a 1-bit full adder is shown in Figure 3. The full adder is composed of 2 half adders and an OR gate for computing carry-out.

Ripple-Carry Adders (RCA)

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. This type of adders is built by cascading 1-bit full adders. A 4-bit ripple-carry adder is shown in Figure 4. Each trapezoidal symbol represents a single-bit full adder. At the top of the figure, the carry is rippled through the adder from cin to cout.

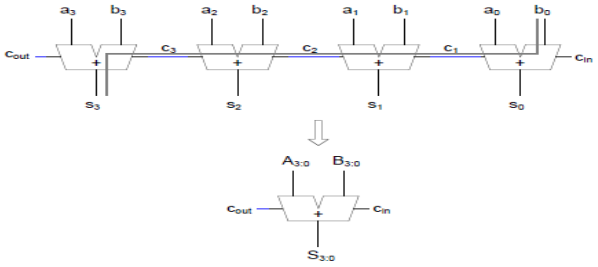

Figure 4: Ripple-Carry Adder.

Carry-Select Adders (CSLA)

Simple adders, like ripple-carry adders, are slow since the carry has to to travel through every full adder block. There is a way to improve the speed by duplicating the hardware due to the fact that the carry can only be either 0 or 1. The method is based on the conditional sum adder and extended to a carry-select adder. With two RCA, each computing the case of the one polarity of the carry-in, the sum can be obtained with a 2x1 multiplexer with the carry-in as the select signal. An example of 16-bit carry-select adder is shown in Figure 5. In the figure, the adder is grouped into four 4-bit blocks.

ISSN (Online):2533-8945
VOLUME 3 ISSUE 6

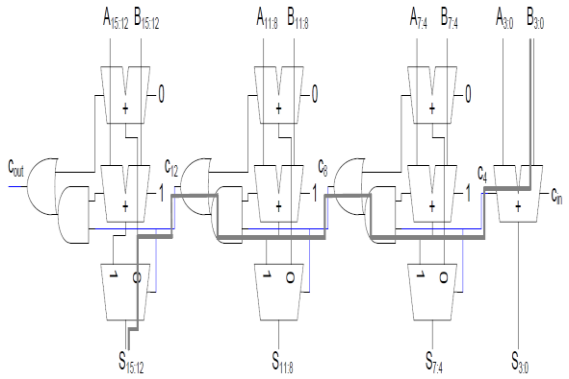

Figure 5: Carry-Select Adder.

Carry-Skip Adders (CSKA)

There is an alternative way of reducing the delay in the carry-chain of a RCA by checking if a carry will propagate through to the next block. This is called carry-skip adders.

ci+1 = Pi:j _ Gi:j + Pi:j.cj

Figure 6 shows an example of 16-bit carry-skip adder.

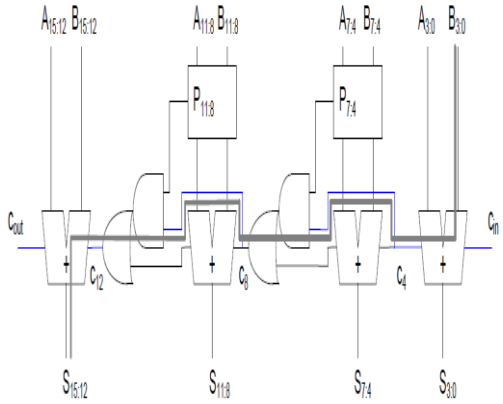

Figure 6: Carry-Skip Adder.

Carry-Look-ahead Adders (CLA)

The carry-chain can also be accelerated with carry generate/propagate logic. Carry-lookahead adders employ the carry generate/propagate in groups to generate carry for the next block. In other words, digital logic is used to calculate all the carries at once. When building a CLA, a reduced version of full adder, which is called a reducedfull adder (RFA) is utilized. Figure 7 shows the block diagram for an RFA. The carrygenerate/propagate signals gi/pi feed to carry-lookahead generator (CLG) for carry inputs to RFA.

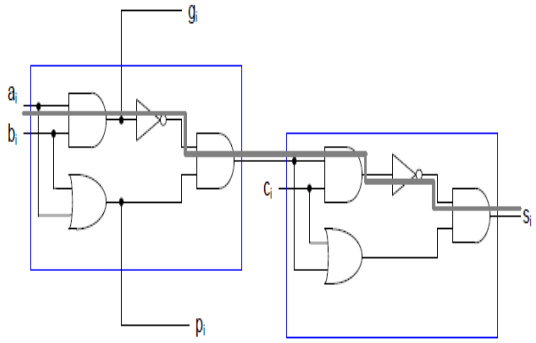

Figure 7: Reduced Full Adder.

The theory of the CLA is based on next Equations. Figure 8 shows an example of 16-bit carry-lookaheadadder. In the figure, each block is fixed at 4-bit. BCLG stands for Block Carry Lookahead Carry Generator, which generates generate/propagate signals in group form. For the 4-bit BCLG, the following equations are created.

Gi+3:i = gi+3 + pi+3 .gi+2 + pi+3 .pi+2 .gi+1 + pi+3 .pi+2 .pi+1 .gi

Pi+3:i = pi+3 .pi+2 .pi+1 .pi

The group generate takes a delay of 4/\, which is an OR after an AND, therefore, the
carry-out can be computed, as follows.

ci+3 = Gi+3:i + Pi+3:i .ci

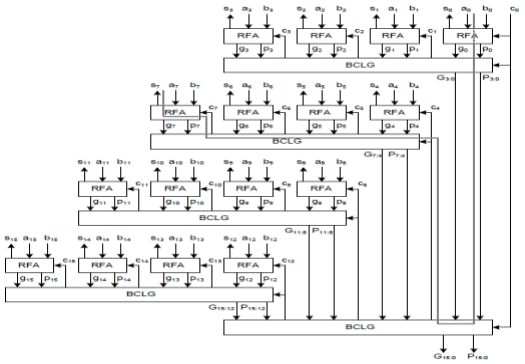

Figure 8: Carry-Lookahead Adder.

III. PROPOSED ADDER

ADDERS ARE basic functional units in computer arithmetic. Binary adders are used in microprocessor for addition and subtraction operations as well as for floating point multiplication and division. Therefore adders are fundamental components and improving their performance is one of the major challenges in digital designs. Theoretical research [1] has established lower bounds on area and delay of -bit adders: the former varies linearly with adder size, the latter has O($\log_2$)n an behavior. High speed adders are based on well establishedparallelprefix

ISSN (Online):2533-8945
VOLUME 3 ISSUE 6

architectures [1], [2], including Brent-Kung [3], KoggeStone [4], Sklansky [5], Han-Carlson [6], Ladner-Fischer [7], Knowles [8]. These standard architectures operate with fixed latency. Better average performances can be achieved by using variable latency adders, that have been recently proposed in literature [9]. A variable latency adder employs speculation: the exact arithmetic function is replaced with an approximated one that is faster and gives the correct result most of the time, but not always. The approximated adder is augmented with an error detection network that asserts an output signal when speculation fails. In this case (misprediction), another clock cycle is needed to obtain the correct result with the help of a correction stage. Since the addition time is one clock cycle when no error occurs and two clock cycles when the speculation fails, the average addition time can be computed as

$$T_{avg} = P_{Err} \cdot 2 \cdot T_{clk} + (1 - P_{Err}) \cdot T_{clk} = T_{clk}(1 + P_{Err}) \quad (1)$$

Where T_{clk} is the clock period and is the error probability of the speculative adder.

Speculative adders are built upon the observation that the critical path is rarely activated in traditional adders [9]–[13]. In particular, in traditional adders each output depends on all previous bits, so the most significant output depends on all the input bits. Instead, in speculative adders each output depends only on the previous K bits, where K goes as $O(\log_2)n$ [12]–[15].

A. Han-Carlson and Kogge-Stone

Parallel-Prefix Adder Topologies The pre-processing and post-processing stages of a prefix adder involve only simple operations on signals local to each bit position. Therefore, adder performance mainly depends on prefix-processing stage. Figure 9 shows Han-Carlson and Kogge-Stone prefix adders topologies. Here black dots represent the prefix operator (10), while white dots are simple placeholders. Kogge-Stone adder is composed by levels and present a fanout of two at each level using a large number of black cells and many wire tracks. A good trade-off between fanout, number of logic levels and number of black cells is given by Han-Carlson. The outer rows of the Han-Carlson topology are Brent-Kung [3] graphs, while the inner rows are Kogge-Stone graphs. The Han-Carlson adder in Fig. 9 uses a single Brent-Kung level at the beginning and at the end of the graph, and the number of levels is .

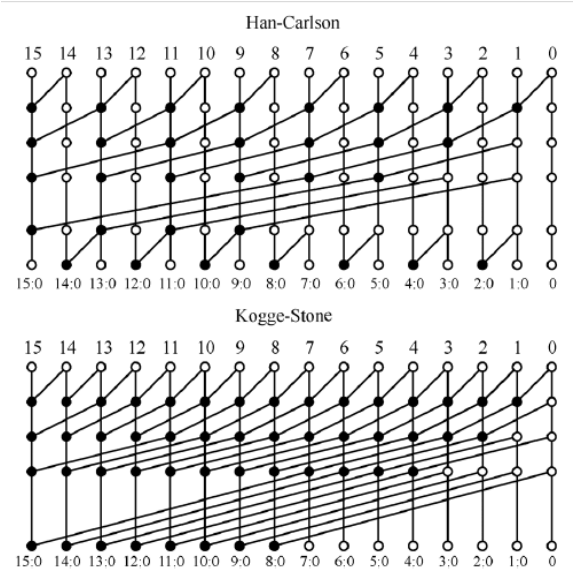

Figure 9: Han-Carlson and Kogge-Stone parallel-prefix topologies n=16.

B. Han-Carlson Topology:

Han-Carlson adder constitutes a good trade-off between fanout, number of logic levels and number of black cells. Because of this, Han-Carlson adder can achieve equal speed performance respect to Kogge-Stone adder, at lower power consumption and area [16]. Therefore it is interesting to implement a speculative Han-Carlson adder. Moved by these reasons, we have generated a Han-Carlson speculative prefix-processing stage by deleting the last rows of the Kogge-Stone part of the adder. As an example, the Fig. 10 shows the Han-Carlson adder of Fig. 9 in which the two BrentKung rows at the beginning and at the end of the graph are unchanged, while the last Kogge-Stone row is pruned. This yields a speculative stage with . In general, one has f levels of the speculative Han-Carlson stage reduces from to (assuming that is a power of two). As it can be observed in Fig. 10, the length of the propagate chains is only for , while for the propagate chain length is . In general, the computed propagate and generate signals for the speculative Han-Carlson architecture are:

$$\begin{aligned} &(g,p)_{[i:0]} && \text{for}: i \leq K \\ &(g,p)_{[i:i-K+1]} && \text{for}: i > K,\ i \text{ odd} \\ &(g,p)_{[i:i-K]} && \text{for}: i > K,\ i \text{ even} \end{aligned} \quad (13)$$

As it will be apparent in the following, having the propagate lengths equal to for half of the outputs greatly simplifies the error detection.

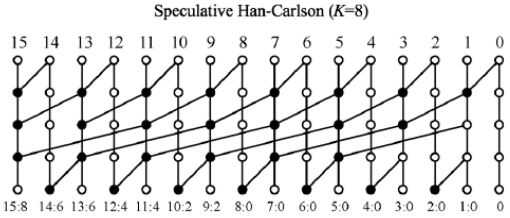

Figure 10: Han-Carlson speculative prefix-processing stage. The last Kogge-Stone row of the n=16 bit graph is pruned, resulting in a speculative prefix processing stage with k=8.

IV. RESULTS

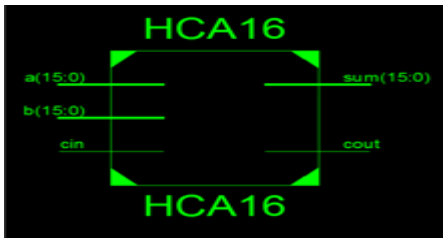

Figure 11: RTL schematic of Top-level 16-Bit Han Carlson Adder

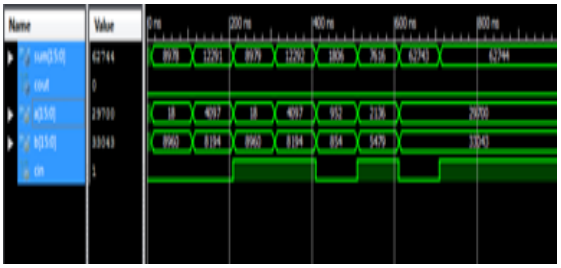

Figure 12: Simulated output for 16-Bit Han Carlson Adder

V. CONCLUSION

In this paper a novel variable latency Han-Carlson parallelprefix speculative adder for high-speed application is proposed. A new, more accurate, error detection network is introduced, which allows reducing the error probability compared to the previous approaches. An extensive set of implementation results for 65 nm CMOS technology shows that proposed Han-Carlson variable latency adders outperforms previously developed variable latency Kogge-Stone architectures. Compared with traditional, non-speculative, adders, our analysis demonstrates that variable latency Han-Carlson adders show sensible improvements when the highest speed is required; otherwise the burden imposed by error detection and error correction stages overwhelms any advantage. Additional work is required to extend the speculative approach to other parallel-prefix architectures, such as Brent-Kung, Ladner-Fisher, and Knowles.

REFERENCES

[1] I. Koren, Computer Arithmetic Algorithms. Natick, MA, USA: A K Peters, 2002.

[2] R. Zimmermann, "Binary adder architectures for cell-based VLSI and their synthesis," Ph.D. thesis, Swiss Federal Institute of Technology, (ETH) Zurich, Zurich, Switzerland, 1998, Hartung-GorreVerlag.

[3] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," IEEE Trans. Comput., vol. C-31, no. 3, pp. 260–264, Mar. 1982.

[4] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Comput., vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[5] J. Sklansky, "Conditional-sum addition logic," IRE Trans. Electron. Comput., vol. EC-9, pp. 226–231, Jun. 1960.

[6] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders," in Proc. IEEE 8th Symp.Comput.Arith. (ARITH), May 18–21, 1987, pp. 49–56.

[7] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," J. ACM, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[8] S. Knowles, "A Family of Adders," in Proc. 14th IEEE Symp. Comput.Arith., Vail, CO, USA, Jun. 2001, pp. 277–281.

[9] S.-L. Lu, "Speeding up processing with approximation circuits," Computer, vol. 37, no. 3, pp. 67–73, Mar. 2004.

[10] T. Liu and S.-L.Lu, "Performance improvement with circuit-level speculation," in Proc. 33rd Annu.IEEE/ACM Int. Symp.Microarchit. (MICRO-33), 2000, pp. 348–355.

[11] N. Zhu, W.-L.Goh, and K.-S. Yeo, "An enhanced low-power highspeed Adder For Error-Tolerant application," in Proc. 2009 12th Int. Symp. Integr.Circuits (ISIC '09), Dec. 14–16, 2009, pp. 69–72.

[12] S. M. Nowick, "Design of a low-latency asynchronous adder using speculative completion," IEE Proc. Comput.Digit. Tech., vol. 143, no. 5, pp. 301–307, Sep. 1996.

[13] A. K. Verma, P. Brisk, and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in Proc. Design, Autom., Test Eur. (DATE '08), Mar. 2008, pp. 1250–1255.

[14] A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," in Proc. Design, Autom., Test Eur. Conf. Exhib. (DATE '09), Apr. 2009, pp. 664–669.