RESEARCH ARTICLE OPEN ACCESS

Comparative Design of 16-Bit Sparse-Tree Rsfq Adder

S. Saddam Hussain[1],
PG Student Scholar M. Tech (VLSI),
SVEC, Tirupati,
Chittoor, A.P, India,

S. Mahaboob Basha[2]
Assistant Professor, ECE Dept.,
SVEC, Tirupati,
Chittoor, A.P, India,

Abstract
In this paper, we propse 16-bit sparse tree RSFQ adder (Rapid single flux quantam), kogge-stone adder, carry lookahead adder. In general N-bit adders like Ripple carry adder s(slow adders compare to other adders), and carry lookahead adders(area consuming adders) are used in earlier days. But now the most of industries are using parallel prefix adders because of their advantages compare to kogge-stone adder, carry lookahead adder, Our prefix sparse tree adders are faster and area efficient. Parallel prefix adder is a technique for increasing the speed in DSP processor while performing addition. We simulate and synthesis different types of 16-bit sparse tree RSFQ adders using Xilinx ISE10.1i tool, By using these synthesis results, We noted the performance parameters like number of LUT's and delay. We compare these three adders interms of LUT's represents area) and delay values.

Keywords—digital arithmetic, RSFQ adder, kogge-stone adder ,carry operator, prefix adder.

I. INTRODUCTION

Arithmetic circuits are the ones which perform arithmetic operations like addition, subtraction, multiplication, division, parity calculation. Most of the time, designing these circuits is the same as designing muxers, encoders and decoders. In electronics, an adder or summer is a digital circuits [7] that performs addition of numbers. In many computers and other kind of processors, adders are other parts of the processor, many computers and other kinds of processors, where they are used to calculate addresses, table and similar. The binary adder [7, 10] is the one type of element in most digital circuit designs including digital signal processors(DSP) and microprocessor data path units. Therefore fast and accurate operation of digital system depends on the performance of adders. Hence improving the performance of adder is the main area of research in VLSI system design[10] .

In RSFQ logic, most adder designs demonstrated to date are bit-serial or digit-serial architectures which operate on a single bit or a small group of bits sequentially at a very high processing rate . Such designs allow for simple clocking and compact structures. However, the latency of serial adders Scales O(n), where n is the number of bits per operand, which leads to long latencies for 32-/64-bit operations in general purpose processors. In the past, parallel architectures in RSFQ have been limited to small data widths or relatively long latency ripple carry adders . One study evaluated 32-/64-bit parallel Kogge-Stone RSFQ adders using co-flow clocking.

This paper is organized as follows; Section II explains the 16-bit Sparse-tree RSFQ adder and detail structure of CSA and RSFQ adder respectively. A section III deals with proposed architecture of Sparse-tree RSFQ with CLA and kogge-stone.A section IV explain about Comparisons of area and delay.

II. 16-BIT SPARSE TREE RSFQ ADDER

A. Sparse-tree RSFQ Adder

High-performance parallel adders typically use prefix trees which generate carries in log2(n) time, where n is the number of bits of the datapath. The Kogge-Stone adder (KSA) [1] is considered to be the fastest among parallel-prefix adders. Further enhancements to the KSA prefix structure such as the sparse-tree configuration have been proposed and used in high-performance Intel processors [2].

In our 16-bit RSFQ adder design, we chose the sparse-tree structure to reduce the number of wiring junctions needed for its implementation without any significant effect on its processing rate. As a side effect, this will also lead to a more energy-efficient design by reducing the total bias current and power consumption. Fig. 1 illustrates the structural diagram of our sparse-tree adder. It consists of the following three stages: Initialization, Prefix-Tree and Summation.

The Initialization stage receives two 16-bit data operands A and B to create bitwise Generate (G) and

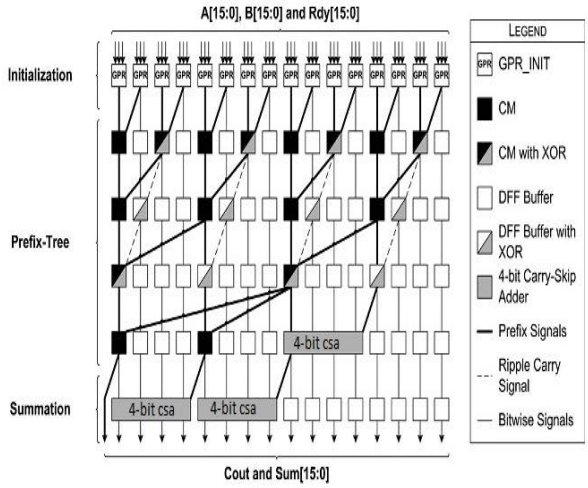

Fig1: 16-bit Sparse-tree RSFQ Adder

Propagate (P) signals which will be merged in a logarithmic manner in the Prefix-Tree stage. The Initialization stage consists of GPR_INIT logic blocks, one for each bit. The GPR_INIT creates the bitwise prefix functions described as Gi = Ai • Bi and Pi = Ai ⊕ Bi where i is the bit index column ranging from 15 down to 0 in the 16-bit adder. These functions are easily realized through clocked AND and XOR gates in a co-flow clocking arrangement. The clock is the R_{dy} signal provided to all bits Additionally, it is necessary to create the trailing reset signal R which will be used to reset the asynchronous elements in the Prefix-Tree. Signal R is a copy of the R_{dy} signal for each bit with wJ-based delay lines to ensure data signals are processed before reset follows in the asynchronously Prefix-Tree.

The Prefix-Tree stage consists of Carry-Merge (CM) blocks to merge the prefix signals and provide a group carry to each 4-bit summation block. In contrast, the Kogge-Stone prefix tree provides a carry to every individual bit of the adder. DFF (D flip-flop) buffers appropriately delay prefix and bitwise P signals until they are ready to be merged or processed at the Summation stage, respectively. The first three levels of the Prefix-Tree also perform the ripple-carry addition within each 4-bit group before data arrive at the Summation stage. The Prefix-Tree stage is built with CM blocks to merge the prefix signals as shown in Fig. 1. Merging of the prefix signals is described in [1]. It is implemented with CFFs (resettable Muller C-flip-flop gates based on the Muller C-element [4], [5]) and confluence buffers used as asynchronous OR gates. The CFFs provide the following functions. First, they behave as asynchronous AND gates. Second, they are used as key re-synchronization elements for wave-pipelining allowing data waves to wait until all their appropriate signals arrive. Due to the encoding of the prefix signals, confluence buffers can be safely used as asynchronous OR gates without any danger of violating the time separation requirement of their input pulses.

The Summation stage computes the final sum with 4-bit carry-skip adders [3]. The lower-half of the adder (bits 7:0) can start the Summation stage early because all appropriate signals are ready. The upper-half of the adder (bits 15:8) must wait until carries for this upper half are calculated by the very last level of the Prefix-Tree stage. The Summation stage has a 4-bit carry-skip adder block [3] for each 4-bit group. In our carry-skip adders, the generation of a carry-in to the two most significant bits of the group is done in parallel with the calculation of the two least significant s

III. PROPOSED SPARSE-TREE RSFQ ADDER

A. Parallel prefix adders

The parallel prefix adders [7] are more flexible and are used to speed up the binary additions. Parallel prefix adders are obtained from Carry Look Ahead (CLA) structure. We use tree structure form to increase the speed [8] of arithmetic operation. Parallel prefix adders are fastest adders and these are used for high performance arithmetic circuits in industries. The construction of parallel prefix adder [9] involves three stages

1. Pre- processing stage
2. Carry generation network
3. Post processing

Pre-possessing stage

In this stage we compute, generate and propagate signals to each pair of inputs A and B. These signals are given by the logic equations 1&2:

Pi=Ai xor Bi ..... (1)
Gi=Ai and Bi ..... (2)

Carry generation network

In this stage we compute carries corresponding to each bit. Execution of these operations is carried out in parallel [9]. After the computation of carries in parallel they are segmented into smaller pieces. It uses carry propagate and generate as intermediate signals which are given by the logic equations 3&4:

CPi:j=Pi:k+1 and Pk:j .....(3)
CGi:j=Gi:k+1 or (Pi:k+1 and Gk:j) .....(4)

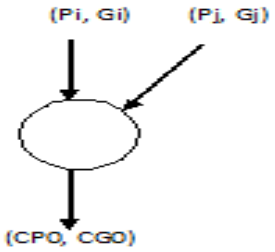

Figure2: Carry operator

Post processing

This is the final step to compute the summation of input bits. It is common for all adders and the sum bits are computed by logic equation 4&5:

Ci-1=(Pi and Cin) or Gi ..... (4)

Si=Pi xor Ci-1 ..... (5)

B. Carry Look Ahead Adder

A Carry Look Ahead adder(CLA) is a type of adder used in digital circuits. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple carry adder[11] for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits. To reduce the computation time, engineers devised faster ways to add two binary numbers by using carry-look ahead adders. They work by creating two signals (*P* and *G*) for each bit position, based on if a carry is propagated through from a less significant bit position (at least one input is a '1'), a carry is generated in that bit position (both inputs are '1'), or if a carry is killed in that bit position (both inputs are '0'). In most cases, *P* is simply the sum output of a half-adder and *G* is the carry output of the same adder. After *P* and *G* are generated the carries for every bit position are created. Some advanced carry-look ahead architectures the Kogge-Stone adder. The modified 16-bit sparse-tree RSFQ adder by using CLA figure shown below.

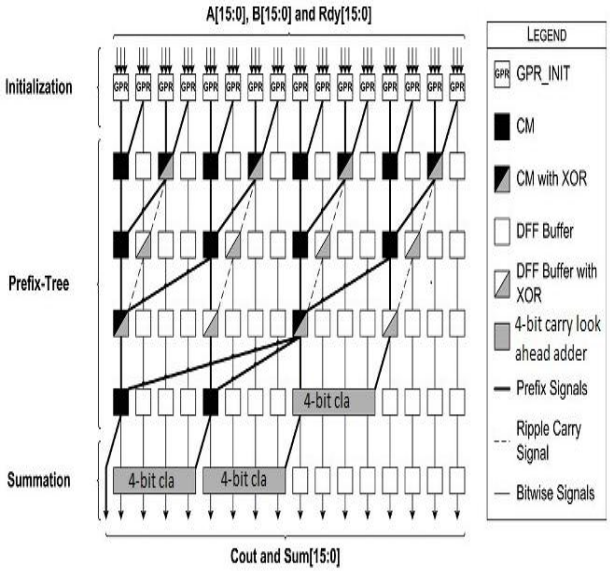

Figure3: Modified 16-bit RSFQ adder by using Carry Look Ahead Adder

C. Kogge-Stone (KS) adder

Kogge-Stone adder is a parallel prefix form carry look ahead adder. The Kogge-Stone adder [6] was developed by peter M. Kogge and Harold S. Stone which they published in 1973. Kogge-Stone prefix adder is a fast adder design. KS adder has best performance in VLSI implementations. Kogge-Stone adder has large area with minimum fan-out. The Kogge- Stone adder is widely known as a parallel prefix adder that performs fast logical addition. Kogge-Stone adder[9] is used for wide adders because of it shows the less delay among other architectures. In fig4 each vertical stage produce Propagate and Generate bits. Generate bits are produced in the last stage and these bits are XORed with the initial propagate after the input to produce the sum bits. The 4-bit Kogge- Stone adder figure shown below.

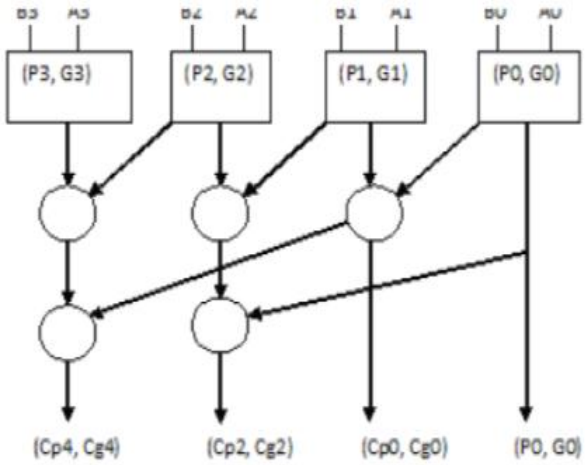

Figure4: 4-bit Kogge-Stone Adder

In this proposed method modification is done by replacing the parameter 4-bit carry skip adder with 4-bit carry look ahead adder, 4-bit Kogge-Stone adders. By using this logic we can reduce delay and area. The figure3&5 shows structure of modified sparse-tree RSFQ adder using CLA and Kogge-stone adder logic.

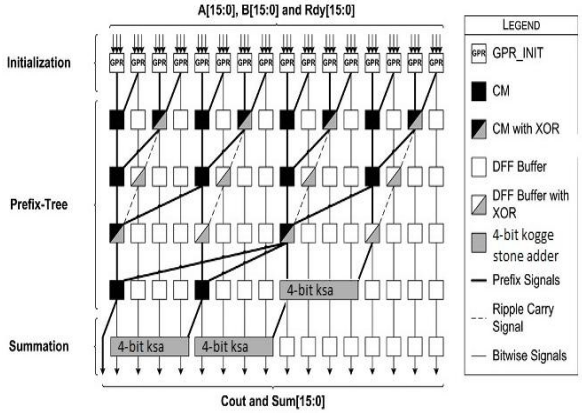

Figure5: Modified 16-bit RSFQ adder by using KSA

IV. SIMULATION RESULTS AND COMPARISIONS

Various adders were designed using Verilog language in Xilinx ISE Navigator 10.1 and all the simulations are performed using Modelsim 6.5e simulator. The performance of proposed adders are analyzed and compared. In this proposed architecture , the implementation code for modified 16-bit sparse-tree RSFQ adder by using Kogge-Stone, carry look Ahead adders were developed and corresponding values of delay and area were observed. Table1 shows the comparision of adders. The simulated outputs of 16-bit proposed adders are shown in Figure 6,7&8.

Table1: Comparisons of Adders

Topology	Delay	No of LUT's
RSFQ WITH CSA	11.352ns	72
RSFQ WITH CLA	5.161ns	17
RSFQ WITH KSA	4.04ns	20

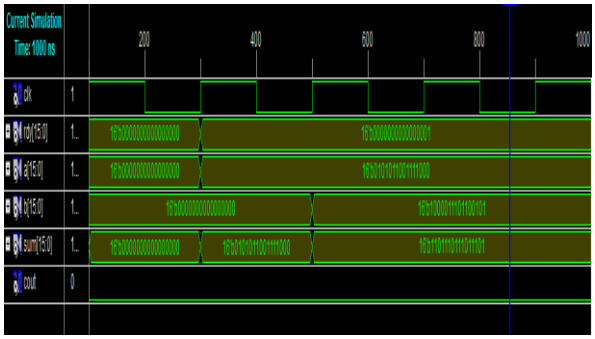

Fig6: Simulated output of 16-bit Sparse-tree RSFQ with CSA

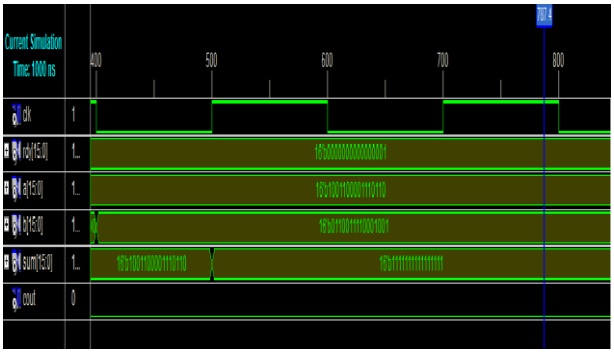

Fig7: Simulated output of 16-bit Sparse-tree RSFQ with CLA

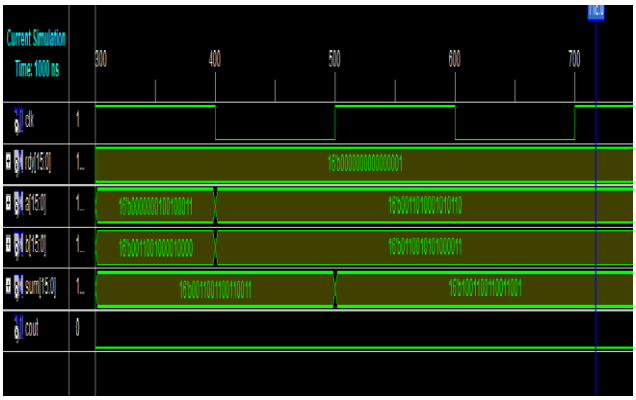

Fig8: Simulated output of 16-bit Sparse-tree RSFQ with KSA

V. CONCLUSION

The proposed adders are faster because of less delay and area efficient compared to other basic adders. Among these three prefix adders Sparse-tree RSFQ with Kogge-stone adder has better performance compared to remaining adders. The performance comparisons between these adders are measured in terms of area and delay. It would be interesting to investigate the design of the 32 and 64 bit adders. These adders are popularly used in VLSI implementations.

VI. ACKNOWLEDGEMENT

S. Saddam Hussain would like to thank Mr. S. Mahaboob basha, Assistant professor ECE Department who had been guiding throughout the project and supporting me in giving technical ideas about the paper and motivating me to complete the work efficiently and successfully.

REFERENCES

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

[2] S. Mathew, M. Anders, R. K. Krishnamurthy, and S. Borkar, "A 4-GHz

130-nm address generation unit with 32-bit sparse-tree adder core," IEEE J. Solid-State Circuits, vol. 38, no. 5, pp. 689–695, May 2003.

[3] A. G. M. Strollo and E. Napoli, "A fast and area efficient complimentary pass-transistor logic carry-skip adder," in Proc. 21st Int. Conf. Microelectron., Sep. 1997, vol. 2, pp. 701–704.

[4] O. A. Mukhanov, S. V. Rylov, V. K. Semonov, and S. V. Vyshenskii, "RSFQ logic arithmetic," IEEE Trans. Magn., vol. 25, no. 2, pp. 857–860, Mar. 1989.

[5] Z. J. Deng, N. Yoshikawa, J. A. Tierno, S. R. Whiteley, and T. van Duzer, "Asynchronous circuits and systems in superconducting RSFQ digital technology," in Proc. 4th Int. Symp. Adv. Res. Asynchronous Circuits Syst., Apr. 1998, pp. 274–285.

[6] Kogge P, Stone H, "A parallel algorithm for the efficient solution of a general class Recurrence relations," IEEE Trans. Computers, Vol.C-22, pp 786-793,Aug. 1973.

[7] Reto Zimmermann. Binary Adder Architectures for Cell-Based VLSI an their Synthesis. Hartung-Gorre, 1998.

[8] Y. Choi, "Parallel Prefix Adder Design," Proc. 17th IEEE Symposium on Computer Arithmetic, pp 90-98, 27th June 2005.

[9] D. Harris, "A taxonomy of parallel prefix networks," in Signals, Systems and Computers, 2003. Conference Record of Thirty Seventh Asilomar Conference on, vol. 2, the Nov. 2003,pp.2217.

[10] N. H. E. Weste and D. Harris, CMOS VLSI Design, 4th edition, Pearson Addison-Wesley, 2011.

[11] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y.Zorian, "Easily Testable Cellular Carry Look ahead Adders," Journal of Electronic Testing: Theory and Applications 19, 285-298, 2003.