Asynchronous Hybrid Kogge-Stone Structure Carry Select Adder Based IEEE-754 Double-Precision Floating-Point Adder

Abhijith Kini G.

Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, NITK-Surathkal. Surathkal, Karnataka 575025, India.

Abstract

In this paper, the design and implementation of a generic fast asynchronous Hybrid Kogge-Stone Structure Carry Select based Adder (HKSS-CSA) is described in detail and its application in the design of asynchronous Double Precision Floating-Point Adder (DPFPA) is presented and the improved latency performance it provides is discussed. A detailed analysis in terms of maximum combinational delay, number of logic levels and logic resources used by both these adders is provided. The proposed HKSS-CSA adder's performance is compared with a generic reference Carry Look-Ahead Adder (CLA) in terms of the above parameters. For the same set of inputs, the HKSS-CSA resulted in approximately 40% (32-bit) – 65% (128-bit) reduction in the number of logic levels, thereby improving the overall latency by a factor of 2 (32-bit) – 6 (128-bit) times compared to a CLA. A 64-bit instance of this HKSS-CSA was made use of in the design of an asynchronous DPFPA and its performance compared with a reference DPFPA which makes use of a CLA in the intermediate stage. The reference DPFPA had a maximum combinational delay of 36.25ns while the newly suggested DPFPA had a delay of 18.60ns for the same set of inputs, giving about 50% improvement in overall latency performance, which can be mainly attributed to the latency improvement provided by the HKSS-CSA.

Keywords: *Double Precision Floating-Point, Hybrid, Kogge-Stone, Carry Select, Carry Look-Ahead, Adder.*

1. Introduction

Floating-point adders (FPAs) are one of the most frequently used components in modern microprocessors, digital signal processors (DSPs) and graphic processing units (GPUs). These adders must not only be fast enough to accommodate the ever increasing speed requirements of designs but also small enough for designs which make use of a number of these adders in parallel; while maintaining the accuracy at the output. The main bottlenecks in the design of FPAs are latency, area and power. Both floating-point addition and subtraction make use of FPA, hence the latency and throughput of FPA is critical in improving the overall performance of a Floating-Point Unit (FPU). Therefore, a lot of effort was spent on reducing the FPA latency [1-4].

In [1], a DPFPA design which makes uses of flagged prefix addition is proposed and the improvement got in latency is described in detail. A two-path FPA design is proposed to improve the overall latency in [2]. In [3], a design which reduces the overall latency by reducing latency at each sub-module by improvement in structural level by using synthesis method is presented. A variable latency algorithm is proposed and made use of in the design of FPA and the improvements got are detailed in [4]. The design tradeoff analysis of FPAs in FPGAs is presented in [5]. The IEEE standard for binary floating-point arithmetic [6] provides a detailed description of the floating-point representation and the specifications for the various floating-point operations. It also specifies the method to handle special cases and exceptions. Nowadays, most floating point units are IEEE compliant and are capable of handling both single precision and double precision floating-point operands.

The main objective of this paper is to present the design of an asynchronous DPFPA which makes use of newly designed HKSS-CSA and is conformable with the latest draft of IEEE-754 standard, its implementation using Very high-speed integrated-circuit Hardware Description Language (VHDL) and its synthesis for a Xilinx Virtex-V FPGA using Xilinx's Integrated Software Environment (ISE) 9.1i. Asynchronous adders offer many advantages, most important is that they do not use a clock signal, hence not constrained to a global timing constraint. The designed DPFPA accepts normalized double precision floating-point numbers as input and the output is also in the same format. It can also handle special cases and exceptions described in the IEEE standard. This design is compared against a reference DPFPA and analyzed in terms of latency, number of logic levels and logic resources used. This design can be easily extended to support operations on single precision floating-point numbers though better delay performance is got for double precision numbers.

The organization of this paper is as follows: In section 2 the standard algorithm for double precision floating-point addition is presented and design improvements in sub-modules which can lead to increase in overall performance of DPFPA are suggested. The design and implementation of generic asynchronous HKSS-CSA used in the proposed

DPFPA design is discussed in section 3 and its performance compared with a standard CLA. In section 4, the design of the suggested DPFPA is described. Also the performance analysis of the proposed DPFPA adder is done and compared with the reference DPFPA in this section. The main conclusions are elucidated in section 5.

2. Standard DPFPA Algorithm

The double precision floating-point addition is one of the most complex operations in a FPU and needs more logical resources compared to 64-bit adder mainly due to normalization, rounding logic and exception handling. The main steps involved in performing an IEEE double precision floating-point addition are summarized below:

- The first step is to extract the sign, exponent and mantissa part of the operands as per the IEEE representation and check whether the inputs are normalized or any of the special types like NaN, infinity or zero.
- Next step is pre-normalization where the two exponents are compared to identify the larger exponent and the smaller operand is aligned by right shifting it by the absolute difference of the exponents.
- Addition of the two aligned mantissa is performed in the next step.
- The post-normalization step involves detection of carry out and right shifting mantissa by 1 and incrementing the exponent in case of carry-out.
- Rounding is done based on certain internal signals, the exponent and mantissa are updated appropriately.
- The final step involves exception handling where checks are done on the output for any special types.

The algorithm used is shown in the form of a flow chart in Fig. 1. Operand A is unpacked based on IEEE double precision floating-point representation into sign (SA), exponent (EA) and mantissa (MA). Similarly SB, EB and MB are got from operand B. The output sign (SO), exponent (EO) and mantissa (MO) are packed in the last stage of the DPFPA and given out in IEEE double precision floating point format.

A detailed delay analysis of the different sub-modules used in the design of DPFPA revealed that the 53-bit mantissa adder lies in the critical path and is also the largest functional unit block in the data path. Improvements in the design of this adder to have better delay performance should lead to an improvement in the overall latency performance of the DPFPA. Parallel prefix addition is a general technique for speeding up binary addition and has a flexible area-time tradeoff. Parallel prefix adders are derived from the family of CLA. A CLA adder improves the speed by reducing the amount of time required to determine the carry bits.

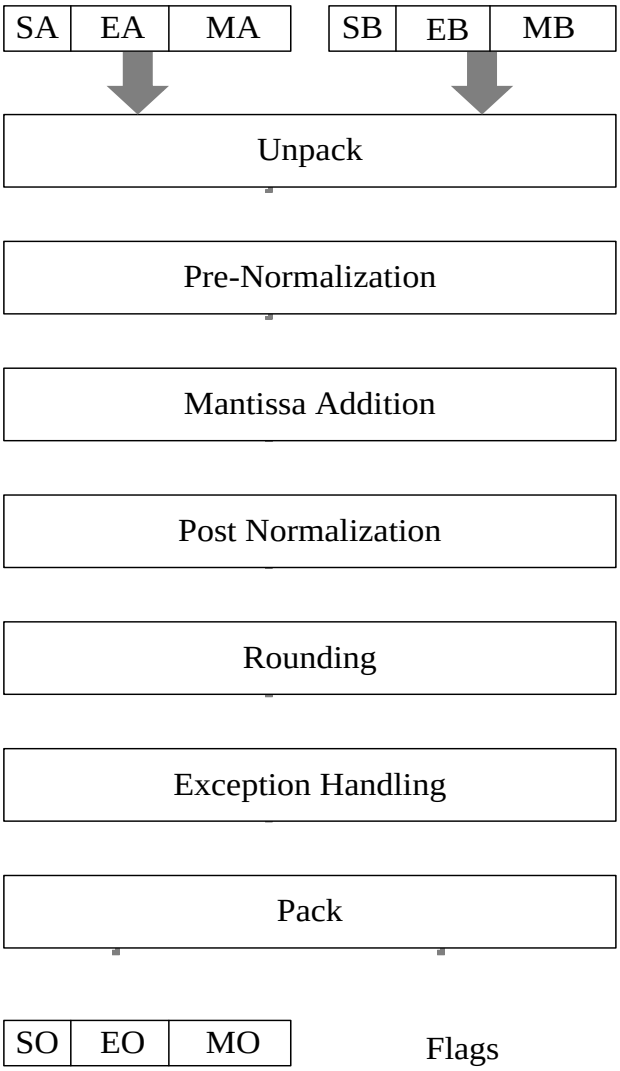


Fig. 1 Asynchronous DPFPA Algorithm.

3. Asynchronous Hybrid Kogge-Stone Structure-Carry Select Based Adder

A general overview of different addition techniques has been presented in [7]. A number of fast adders like carry-skip adder, carry-select adder and carry look-ahead adders have been proposed in the past [8]. The CLA uses the concepts of generating and propagating the carries. The addition of two 1-bit inputs A and B is said to *generate* if the addition will always carry, regardless of whether there is an input carry. The addition of two 1-bit inputs is said to *propagate* if the addition will carry whenever there is an input carry. The equations (1) and (2) are used to find generate and propagate terms respectively.

$$G(A,B) = A.B \quad (1)$$

$$P(A,B) = A \oplus B \quad (2)$$

The parallel prefix addition can be viewed as a 3-stage process shown in Fig. 2. The improvements can be made primarily in the carry generation stage which is the most intensive one. These adders make use of a tree structure for calculating the carry and hence reduce the latency of the output. Among all trees, the Kogge-Stone tree structure is the most commonly used parallel prefix topology in high performance data paths. The main features are minimum logic depth, regular structure and uniform fan-out. The main disadvantages are large number of wires and high power dissipation. The Kogge-Stone Structure (KSS) based adder is a parallel prefix form of CLA and generates the carry signals in $O(\log_2(N))$ time (where N represents number of input bits) and is considered the fastest adder design widely used in industry.

These designs are better suited for adders with wider word lengths and there is a minimal fan-out of 2 at each node, hence faster performance. The algorithm developed by Kogge-Stone [9] has both optimal depth and low fan-out but produces massively complex circuit realizations and also account for large number of interconnects. Brent-Kung adder [10] makes use of minimal number of computation nodes, hence results in reduced area, but the structure has maximum depth which results in increased latency compared with other structures. The Han-Carlson adder [11] combines Brent-Kung and Kogge-Stone structures to achieve a balance between logic depths and interconnect count. An algorithm for generating parallel prefix adders with variable parameters has been discussed in [12]. A matrix representation for the gate level design of parallel prefix adders has been presented in [13].

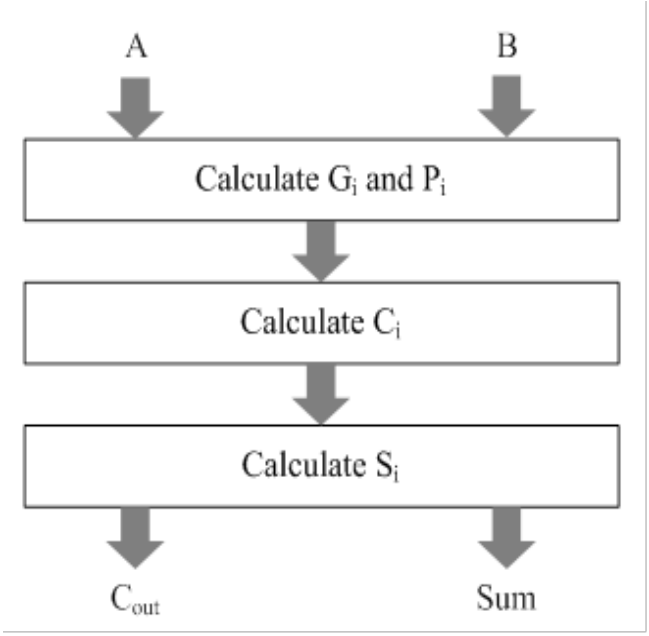


Fig. 2 Parallel-Prefix 3-stage Addition Process.

3.1 Design and Implementation

The parallel prefix adders and CLA adders differ in the way their carry generation block is implemented. In Stage I, generate and propagate components are found for each bit combination of the inputs A and B. The block level representations for each of the cell modules with its corresponding inputs and outputs are shown in Fig. 3. Each of these cell modules were written in VHDL and verified individually. The top module consists of instances of these sub-modules and is verified finally as a single block.

White cell module is used to form the initial generate and propagate components G_i and P_i with A_i and B_i as its inputs. The expression to calculate G_i and P_i are given by (3) and (4) respectively.

$$G_i = A_i . B_i \quad (3)$$

$$P_i = A_i \oplus B_i \quad (4)$$

In Stage II, the calculation of the carries C_i is done by making of use of the Kogge-Stone structure. This stage makes use of black cell, grey cell and buffer cell modules, each having their unique functionalities. The Black cell module takes (G_{in1}, P_{in1}) and (G_{in2}, P_{in2}) as its inputs and forms the (G_{out}, P_{out}) using (5) and (6).

$$G_{out} = G_{in1} + G_{in2} . P_{in1} \quad (5)$$

$$P_{out} = P_{in1} . P_{in2} \quad (6)$$

The Grey cell module outputs only the generate component G_{out} using (5) taking (G_{in1}, P_{in1}) and G_{in2} as its input. The buffer cell acts as a buffer passing its inputs (G_{in1}, P_{in1}) as its outputs (G_{out}, P_{out}).

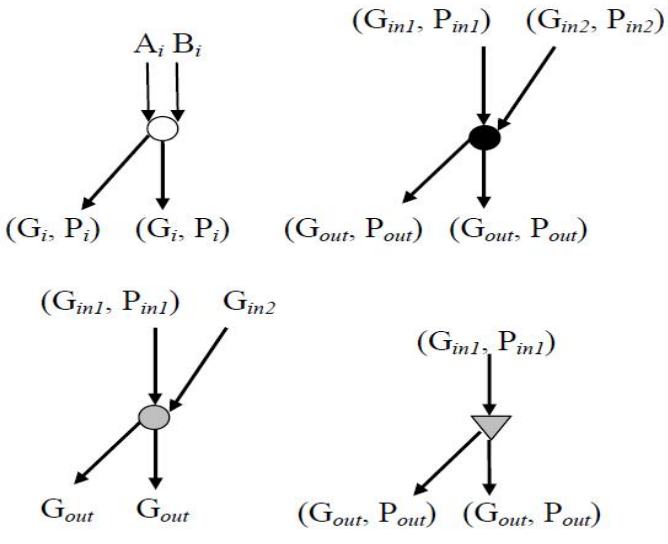


Fig. 3 (Clockwise)
White cell, Black cell, Grey cell, Buffer cell.

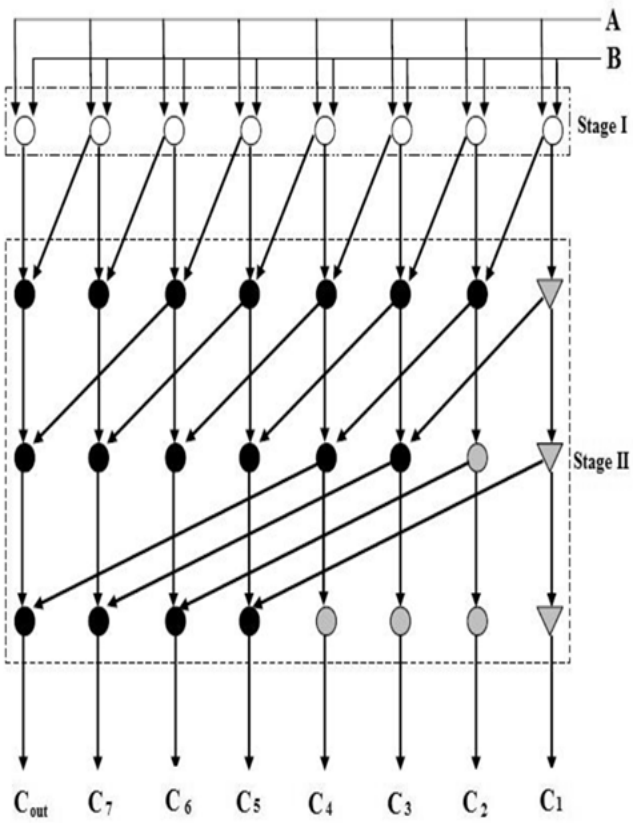


Fig. 4 Kogge-Stone Structure for 8-bit Adder.

The Kogge-Stone tree structure for an 8-bit adder is shown in Fig. 4. The height of a Kogge-Stone tree for an N-bit adder is given by $\log_2(N)$ logic stages. Hence for an 8-bit adder, we have a height of 3 logic stages. The *radix* of the adder refers to how many results from the previous level of computation are used to generate the next one. In this design Radix-2 implementation is done. Higher radix levels can be made use of to reduce the number of stages, but this increases the power and delay. The *sparsity* of the adder refers to how many carry bits are generated by the carry-tree. In this design every carry bit is generated, hence making it Sparsity-1 design.

Kogge-Stone structure has minimal depth but high node count, hence occupies more area in comparison to other tree structures like Brent-Kung, Han-Carlson, and Ladner-Fischer, but has a minimal fan-out of 2 at each given stage, hence giving a better delay performance.

The carries C_i generated in stage II is made use of to calculate the sum S_i in stage III. These carries are used as the carry-in inputs for much shorter RCAs or some other adder design, which generates the final sum bits. But by making use of a Carry Select Adder (CSA) in the last stage, the delay can be further reduced since the sum bits will already be calculated. In a CSA, we compute two results in parallel, each for different carry input assumption. A CSA consists of two Ripple Carry Adders (RCAs) and a multiplexer. Adding two N-bit numbers with a CSA is done with two RCAs in order to perform the calculation twice, one time with the assumption of the carry being zero (C_i = '0') and the other assuming one (C_i = '1'). These two calculations can be done in parallel. After the two results are calculated, the correct sum is selected with the multiplexer once the carry is known. The carries C_i from stage II are fed as inputs to the CSA, so as to choose the appropriate S_i which has already been calculated, thereby reducing the overall delay of the final output. Fig. 5 shows the block diagram of CSA used in the design. The block diagram of the RCA implemented is shown is Fig. 6.

In order to compare the performance of the proposed adder with the CLA, a generic CLA was designed in a structural manner. The block diagram of a generic N-bit CLA is shown in Fig 11. Each of the GPB blocks contains logic which calculates the G_i, P_i and C_j for i = N-2, N-1 … 2, 1, 0 and j = N-1, N-2…, 2, 1 using equations (3), (4) and (7).

$$C_j = G_i + P_i \cdot C_i \qquad (7)$$

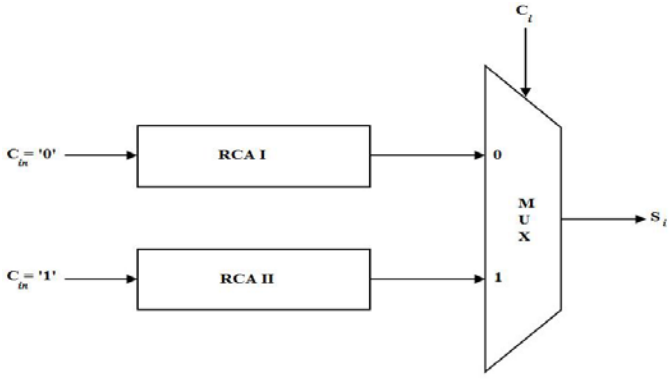


Fig. 5 Block diagram of Carry Select Adder (CSA).

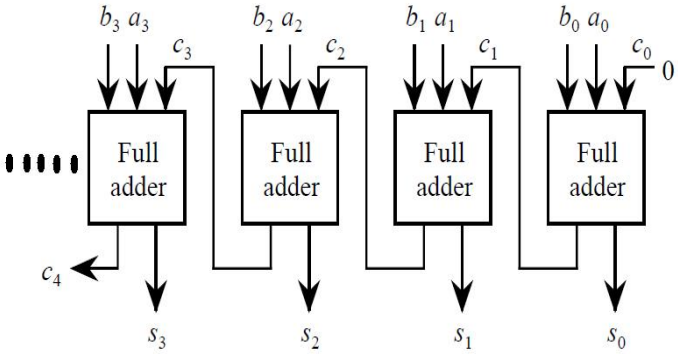
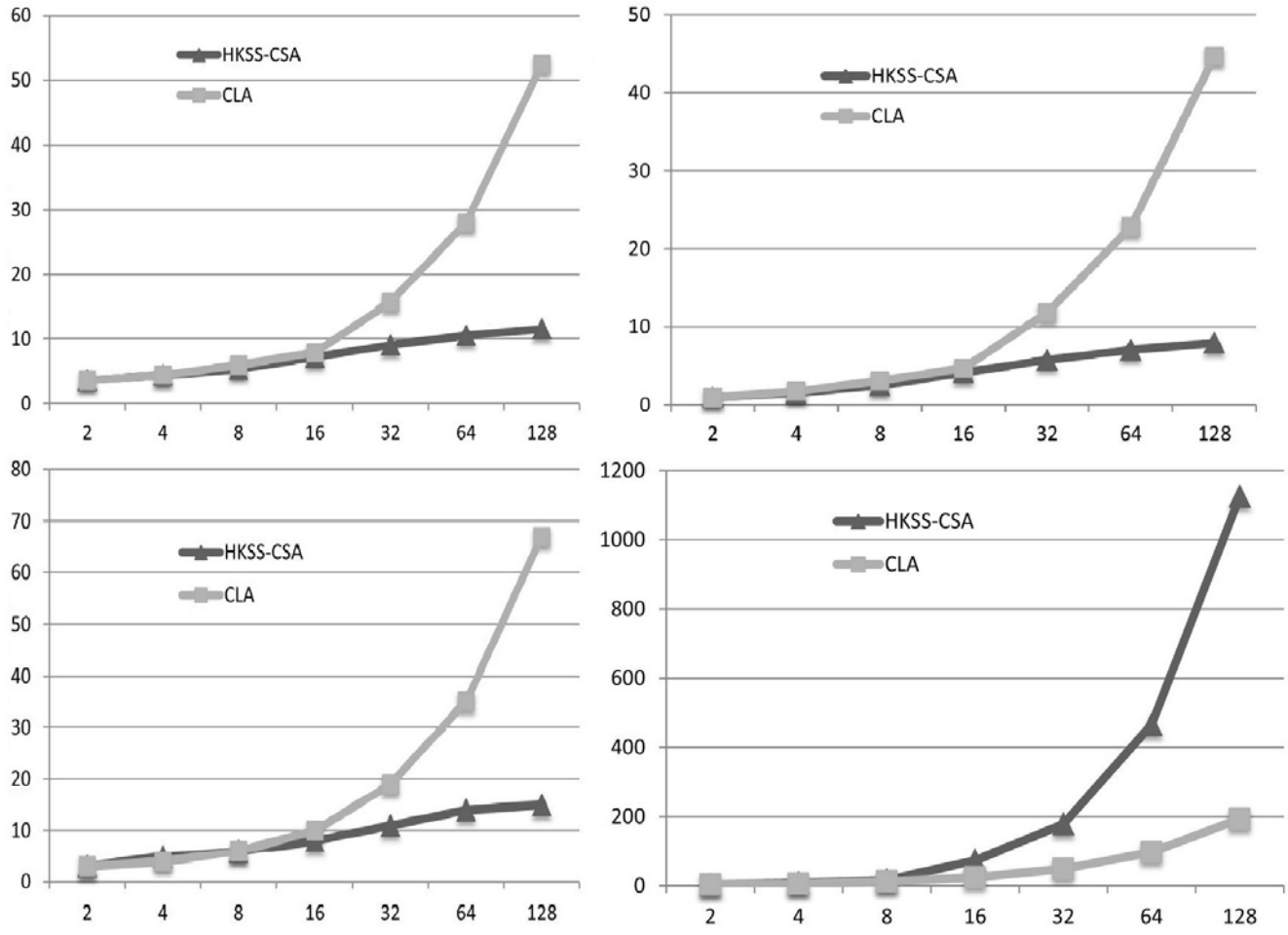


Fig. 6 Block diagram of Ripple Carry Adder (RCA). [7]

Table 1: Delay and device utilization summary.

N (No. of bits)	*Type of Adder*	*Maximum combinational path delay (ns)*	*Route Delay (ns)*	*Logic Delay (ns)*	*Levels of Logic*	*No. of Slices*
2	HKSS-CSA	3.600	0.970	2.630	3	3
	CLA	3.600	0.970	2.630	3	3
4	HKSS-CSA	4.418	1.515	2.903	5	8
	CLA	4.433	1.723	2.710	4	6
8	HKSS-CSA	5.367	2.497	2.870	6	16
	CLA	5.962	3.092	2.870	6	12
16	HKSS-CSA	7.155	4.125	3.030	8	73
	CLA	7.880	4.690	3.190	10	24
32	HKSS-CSA	9.030	5.760	3.270	11	178
	CLA	15.754	11.844	3.910	19	49
64	HKSS-CSA	10.561	7.051	3.510	14	466
	CLA	27.986	22.796	5.190	35	97
128	HKSS-CSA	11.540	7.951	3.590	25	1125
	CLA	52.451	44.701	7.750	67	193



(Clockwise)
Fig. 7 Maximum combinational delay (in ns) as a function of N (bits).
Fig. 8 Routing delay (in ns) as a function of N (bits).
Fig. 9 Levels of logic as a function of N (bits).
Fig. 10 No. of slices as a function of N (bits).

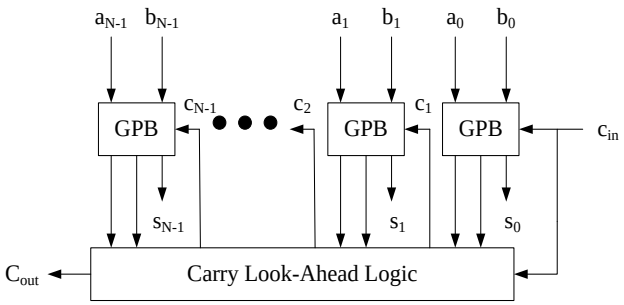


Fig. 11 Block diagram of N-bit CLA.

3.2 Performance Results

The design of the proposed HKSS-CSA and CLA was modeled in VHDL by making use of a module based structural approach. Both the adders where designed and synthesis was done using Xilinx-ISE EDA software on Virtex5 device. Functional testing was done on corner cases, as well as large number of test cases were written to verify the functionality of the adders. The simulation of the waveforms was done using ModelSim-SE.

The performance comparison of the two designs is done in Table 1, which provides the delay, number of logic levels and device utilization summary of HKSS-CSA and CLA. In Fig. 7, a graph of maximum combinational path delay as a function of number of input bits (N) is shown. From this we can observe that there is a drastic increase in the maximum combinational delay for CLA as N increases compared to HKSS-CLA. Hence HKSS-CSA design is more suitable for adders with higher input widths.

As evident from Fig. 8, in CLA the routing delay increases for higher values of N. The advantage of the HKSS-CSA becomes more apparent for higher values of N; delay is reduced by 6 times (N=128) and by 3 times (N=64). The routing delay increases almost in a linear manner for HKSS-CSA while its almost exponential for CLA.

As can be observed from Fig. 9, there is a reduction of about 40% (32-bit) to 65% (N=128) in the number of logic levels when compared with a CLA of same width, thus leading to a better overall delay for the proposed HKSS-CSA when compared to CLA. Also on comparing with Fig. 7, we can see that the number of logic levels has a definite relationship to the maximum combinational path delay.

Also as expected the HKSS-CSA makes use of more number of logic resources when compared to the CLA. CLA uses lesser logic resources in comparison to HKSS-CSA and this difference increases considerably for higher values of N. Hence as the number of required logic resources increases, area also increases. In Fig. 10, a plot of number of slices used as a function of N is shown.

4. Asynchronous Hybrid Kogge-Stone Structure Carry Select Based DPFPA

This section will review the DPFPA algorithm architecture and the hardware sub-modules used in implementing it. The block diagram of DPFPA is shown in Fig. 12.

4.1 Design and Implementation

The standard DPFPA algorithm presented in section 2 is designed and implemented here. The two 64-bit inputs A and B are both in IEEE double precision floating-point format and the output sum will also be in the same format. A 64-bit double precision floating point number defined as per the IEEE-754 standard consists of the 1- bit sign ([63]), 11- bit exponent ([62:52]) and 52- bit mantissa ([51:0]). The design consists of mainly three blocks: pre-adder, adder and post-adder. Each of these blocks contains some Functional Unit Blocks (FUBs) which perform specific functions.

- Pre-Adder Block

The FUBs in the pre-adder block perform operation on the operands to be done before the mantissa addition is done. The *Unpack* FUB basically extracts and signs the sign, exponent and mantissa parts of A (SA, EA, MA) and B (SB, EB, MB) as per IEEE double precision floating point format. The check for special formats is also done in this FUB and flags (EI) sent to the post-adder block. Pre-normalization is done in the *Exp* FUB, in which the two exponents EA and EB are compared to identify the larger exponent. The right shifting of the smaller operand by an amount equal to the absolute difference of the exponents is done in the *Align* FUB. The sign bit of the final sum is got from the *Sign* FUB.

- Adder Block

A 53-bit adder is needed to add the mantissas MAF and MBF coming in from the pre-adder block. The improving the latency performance of this adder we can improve the overall latency performance of the DPFPA. As shown in section III, the newly proposed 64-bit KSS-CSA gives good latency performance in comparison to the CLA. A 64-bit adder is chosen here instead of a 53-bit adder so as to take advantage of the well-ordered tree structure for carry generation, minimal fan-out at each stage, reduced number of logic levels, and the carry select adder in the last stage. Hence the inputs MAF and MBF to the adder block are zero padded before addition and the final result is sent to the post-adder block.

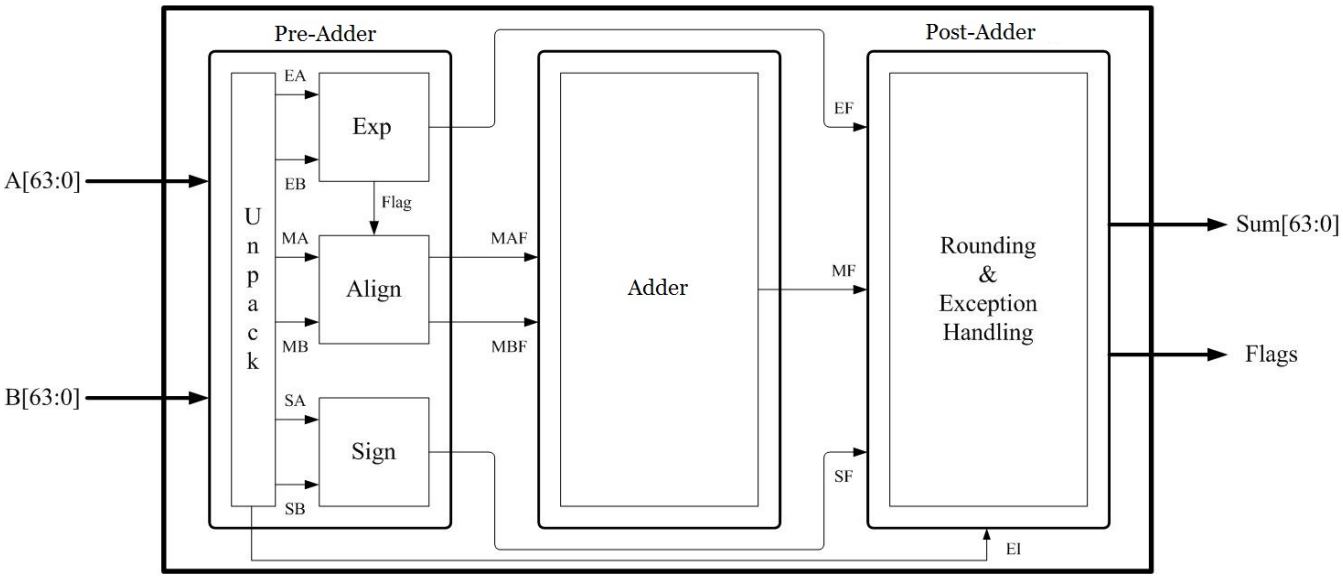


Fig. 12 Block diagram of DPFPA.

- Post-Adder Block

The FUBs in the post-adder block perform operations on the results of mantissa addition. Post-normalization of the output mantissa is done in this FUB. Rounding and exception handling is done in the last stage of the FUB. The result sign, exponent and mantissa are stitched together to form the 64-bit floating point output SUM. Additional flags are also generated at the output to detect special cases.

4.2 Performance Results

In order to compare the performance of the proposed HKSS-CSA based DPFPA against a reference CLA based DPFPA, both the designs were modeled in VHDL and synthesized using Xilinx-ISE EDA software on Virtex5 device. The functionality of the designs was tested by applying a large number of test vectors and Modelsim-SE was used to verify the design. Various corner and special cases were also covered and the performance statistics in terms of maximum combinational delay, route and logic delays, number of levels of logic and slice logic utilization were collected. Table 2 provides the performance comparison data with respect to the proposed and reference DPFPA. Here again we can see a definitive relationship between the number of logic levels and maximum combinational path delay.

The suggested design led to a reduction in the number of logic levels by about 50% compared to the reference DPFPA for the same set of inputs. The route and logic delays which comprise the total delay for the two designs can be seen in Fig. 13. The maximum combinational path delay was reduced from 36.25ns to 18.60ns, so the proposed HKSS-CSA based DPFPA is almost twice as fast as the CLA based DPFPA.

Also as expected the proposed design makes use of more amount of logical resource while giving a better delay performance which can be attributed mainly to the improved latency performance of the intermediate adder. The delay of an adder depends on how fast the carry bit can be generated and used in sum generation. Thus by making use of HKSS-CSA in the mantissa addition stage, the suggested DPFPA gave a better overall delay performance in contrast to the DPFPA which uses the CLA.

Table 2: Delay and device utilization summary of DPFPA.

DPFPA based on	*Maximum combinational path delay (ns)*	*Route Delay (ns)*	*Logic Delay (ns)*	*Levels of Logic*	*Slice Logic*
HKSS-CSA	18.600	13.756	4.844	28	1850
CLA	36.253	29.889	6.364	51	1556

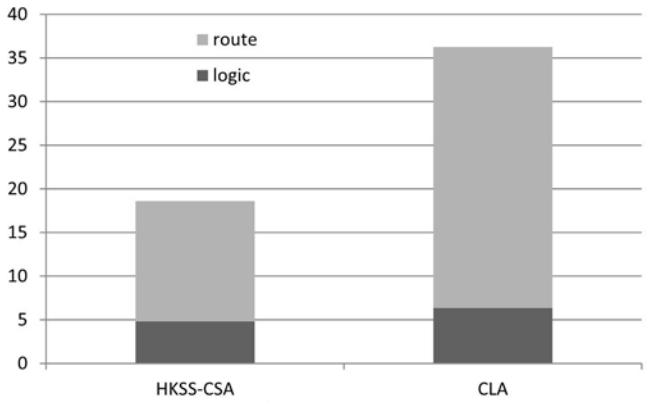


Fig. 13 Delay performance of DPFPA based on HKSS-CSA and CLA.

5. Conclusions

In this paper, the design and implementation of generic HKSS-CSA is proposed and its performance compared with a generic CLA. We have analyzed the delay, number of logic levels and logic resources needed by the HKSS-CSA and CLA for different input widths. The results show that the HKSS-CSA is faster in comparison to CLA because of a number of factors like minimal fan-out at each stage, reduced number of logic levels, well ordered tree structure for carry generation and the carry select adder in the last stage. But with regard to the use of logic resources, the CLA performs better. The new design led to a reduction by 40%-65% in the levels of logic and thus a latency improvement by a factor of 2-6 times when compared to CLA for higher values of N (32, 64, 128 bit). Also this paper proves the tradeoff that exists between maximum combinational delay and amount of logic resources used, and hence the area i.e. area-timing tradeoff. In addition to this, we can see that lower delay advantage of HKSS-CSA can be got for adders with higher input widths.

An application of this HKSS-CSA in the design of fast asynchronous DPFPA is investigated and the performance of this adder compared with a DPFPA which makes use of CLA in the intermediate stage. The suggested design led to a reduction by about 50% in the number of levels of logic compared to one with CLA thereby leading to an overall decrease in the maximum combinational delay. For future work we plan to make use of this HKSS-CSA in the design of single precision and double precision floating point multipliers, to add the mantissa part.

References

[1] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim, “Reduced Latency IEEE Floating-Point Standard Adder Architectures”, Proc.14th IEEE Symp. Computer Arithmetic, pp. 35-43, 1999.

[2] Peter-Michael Seidel and Guy-Even, “On the Design of Fast IEEE Floating-Point Adders”, Proc. International Symposium on Computer Arithmetic, 2001.

[3] Huang, C., X. Wu, J. Lai, C. Sun and G. Li, “A design of high speed double precision floating point adder using macro modules”, Proc. of the Asia and South Pacific Design Automation Conference, pp: D11-D12, 2005.

[4] S. Oberman, H. Al-Twaijry, and M. Flynn, “The SNAP Project: Design of Floating Point Arithmetic Units,” Proc. 13th IEEE Symp. Computer Arithmetic, pp. 156-165, 1997.

[5] Ali Malik, Lee Moon Ho, Dongdoung Chen, Younhee Choi, Seok Bum Ko, “Design Tradeoff Analysis of Floating-point Adders in FPGAs”, Canadian Journal of Electrical and Computer Engineering, vol. 33, pp. 170 - 175, 2008.

[6] The Institute of Electrical and Electronic Engineers, Inc. IEEE Standard for Binary Floating-point Arithmetic. ANSI/IEEE Std 754-1985.

[7] Prof. Vojin G. Oklobdzija, “VLSI Arithmetic”, http://www.ece.ucdavis. edu/acsel

[8] B. Parhami, Computer Arithmetic–Algorithm and Hardware Designs, Oxford University Press, 2000.

[9] P.Kogge and H.Stone, “A Parallel Algorithm for the efficient solution of a general class of recurrence relations,” IEEE Transactions on Computers, vol. C-22, no.8, August 1973, pp.786-793.

[10] R.Brent and H.Kung, “A Regular Layout for Parallel adders,” IEEE Transaction on Computers, vol. C-31, no.3, March 1982, pp. 260-264.

[11] T. Han and D. Carlson, “Fast Area Efficient VLSI adders,” Proceedings of the 8th Symposium on Computer Arithmetic, September 1987, pp. 49-56.

[12] A. Beaumont Smith, C. C. Lim, "Parallel Prefix Adder Design", Proceedings of 15th IEEE Symposium on Computer Arithmetic, June 2001, pp. 218-225.

[13] Youngmoon Choi, Earl E. Swartzlander Jr, "Parallel Prefix Adder Design with Matrix Representation", Proceedings of 17th IEEE Symposium on Computer Arithmetic, June 2005, pp. 90 - 98.

Abhijith Kini G.: holds a Bachelor’s degree (2006-2010) in Electronics and Communication Engineering from National Institute of Technology Karnataka, NITK-Surathkal, India. His areas of interest are VLSI systems, signal processing and communication systems. He has an IEEE publication in the field of wireless sensor networks and a journal in the field of VLSI signal processing. He is presently working as a component design engineer at Intel India Technology Pvt. Ltd since August 2010.