VLSI Implmentation of a Fast Kogge-Stone Parallel-Prefix Adder

International PostGraduate Conference on Applied Science & Physics 2017 IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **1049** (2018) 012077 doi:10.1088/1742-6596/1049/1/012077

VLSI Implmentation of a Fast Kogge-Stone Parallel-Prefix Adder

Lee Mei Xiang, Muhammad Mun'im Ahmad Zabidi, Ainy Haziyah Awab, Ab Al-Hadi Ab Rahman

Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Malaysia

E-mail: `l.mei.xiang@hotmail`, `raden@fke.utm.my`, `ainy.haziyah@gmail.com`, `hadi@fke.utm.my`

Abstract. This paper presents a VLSI implementation of a high speed Kogge-Stone adder (KSA) using 0.18μm process technology. The adder is known to be one of the fastest adder architectures, and this is validated through a comparison with other adder architectures including the standard ripple carry adder and the carry look ahead adder. Furthermore, our KSA adder is also compared with a default optmized adder from the Artisan standard cell library. The adders are compared for bit widths of 8, 16, and 32. The adders are designed using Verilog and synthesized using both front-end and back-end tools, with complete validation and verification stages, including analysis for performance, power, and area. Results show that in terms of performance, KSA results in the lowest propagation delay with almost constant delay for all bit widths, with up to 70% less delay as compared to all other architectures. Area and power penalty is found to also increase by roughly 59%. In terms of energy usage, the KSA adder results in up to 64% less. In the case when speed and energy are critical, this fast and energy efficient KSA adder can be readily integrated into custom VLSI designs.

1. Introduction

Adder is one of the fundamental building blocks in a digital system. It is used in microprocessors, as well as many signal processing operations such as filtering and convolution. These basic blocks are also used in larger signal processing systems such as in image and video compression, object recognition, modulation, etc. Accelerating adders used in these systems therefore enables many applications to speed up. In terms of hardware implementation for VLSI, many different types of adder architectures have been reported, each with their own strenghs and weaknesses.

The main aim of this paper is to implement one of the fastest adder architectures using VLSI technology, which is the Kogge-Stone adder, and validate its performance, power, and area as compared to other adder architectures, including the ripple carry adder, carry look ahead adder, and the default adder from the standard cell library. The objective is to have a fast VLSI adder that can be integrated into larger systems and improve its performance.

Some of the works that implement the KSA include [1] where an adaptive power gating is applied on a 32-bit KSA design. However, the technique is only applicable to partially idle circuits and not efficient for general addition operation. The work in [2] implements the KSA on FPGAs, but may not be as efficient in VLSI since the implementation is mainly on gate-level. The work in [3] is most similar to the work in the present paper, where the KSA is implemented using 0.18μm process technology. However, the focus was mainly on Monte-Carlo simulation and temperature analysis than benchmarking with other adder architectures. The work in this paper proposes to implement in VLSI the standard KSA adder and benchmark against some of the popular and commonly used adder architectures.

2. Background

The adder is a fundamental building block and known to be an important element in many applications especially in digital signal processing and microprocessors. The most basic adder architecture is known as the Ripple Carry adder (RCA), where the longest path is the carry delay. The path becomes longer as the number of bits are increased.

Extensive research was carried out to improve this long carry delay path. In 1958, Weinberger and Smith proposed using propagate and generate blocks to handle the carry issue. This design is today known as the Carry Look Ahead adder (CLA) [4]. It consists of the propagator and generator, and the carry generator block as shown in Figure 1 and 2. The top-level 8-bit CLA is shown in Figure 3. It shows that the PG runs parallel for each bit. However, the process in PPC depends on the C_{out} from previous bit to complete. Hence, the time delay is not solved completely by CLA [5].

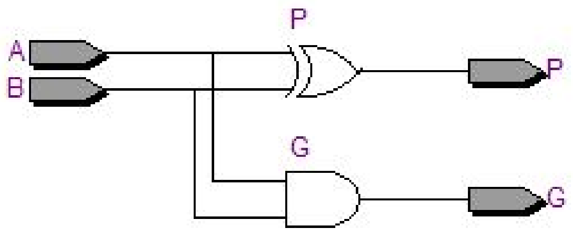


Figure 1. Architecture of Propagator and Generator Block.

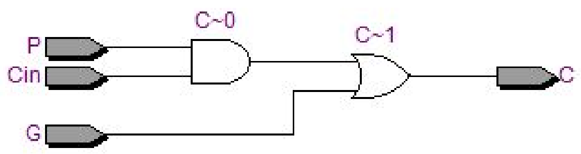


Figure 2. Architecture of Various Carry Generator Block.

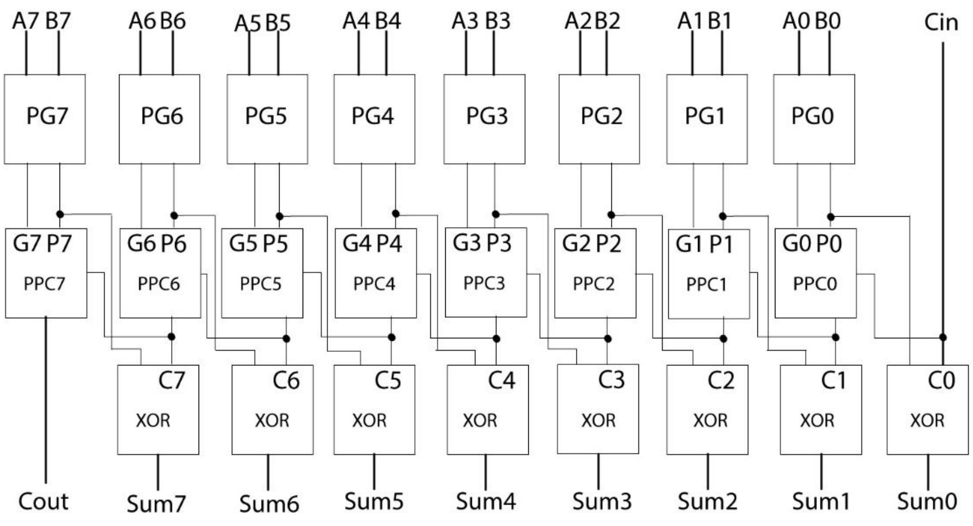


Figure 3. Architecture of 8-bit Carry Look Ahead Adder.

Parallel Prefix Adder was proposed, and expected to have better performance than previous adder. In Parallel Prefix adder, binary addition usually presents in terms of Generation Signal (G$_i$), Propagation Signal (P$_i$), Carry Signal (C$_i$) and Sum Signal (Sum$_i$). Each bit position follows the range $n \leq i \leq 1$. All of these signals can be obtained from the equation below [6]:

$$G_i = A_i + B_i \tag{1}$$

$$P_i = A_i \oplus B_i \tag{2}$$

$$C_i = \begin{cases} G_i \\ G_i + P_i C_{i-1} \end{cases} \tag{3}$$

$$Sum_i = P_i \oplus C_{i-1} \tag{4}$$

The sum for Parallel Prefix Adder can be obtained by three stages as shown in the block diagram in Figure 4. [7].

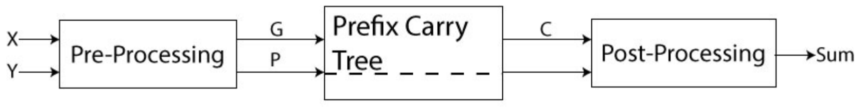


Figure 4. Block Diagram in getting the sum for Parallel Prefix Adder.

P$_i$ and G$_i$ generated from pre-processing block based on equation (1) and (2). The signals then proceeds to the next stage, Prefix Carry Tree to generate C$_i$. This stage is the part that differentiates the performance of an adder used. On the other hand, the final stage is aimed to get the final adder result Sum$_i$ by following equation (4).

There are a lot of Parallel Prefix adders that have been developed starting from 1960s. For example J.Sklansky conditional adder from 1960, Kogge-Stone adder from 1973, Ladner-Fisher adder from 1980 and etc [8].

The concept of KSA was developed by Peter M. Kogge and Harold S. Stone, published in a seminal paper in 1973 [9]. It was introduced to solve a large class of recurrence problems in parallel computer such as the Illiac IV. Nowadays, KSA is known as the special and fast adder in VLSI. The architecture consists of three blocks, which are pre-processing, carry generator and post processing blocks. Figure 5 shows the architecture of KSA.

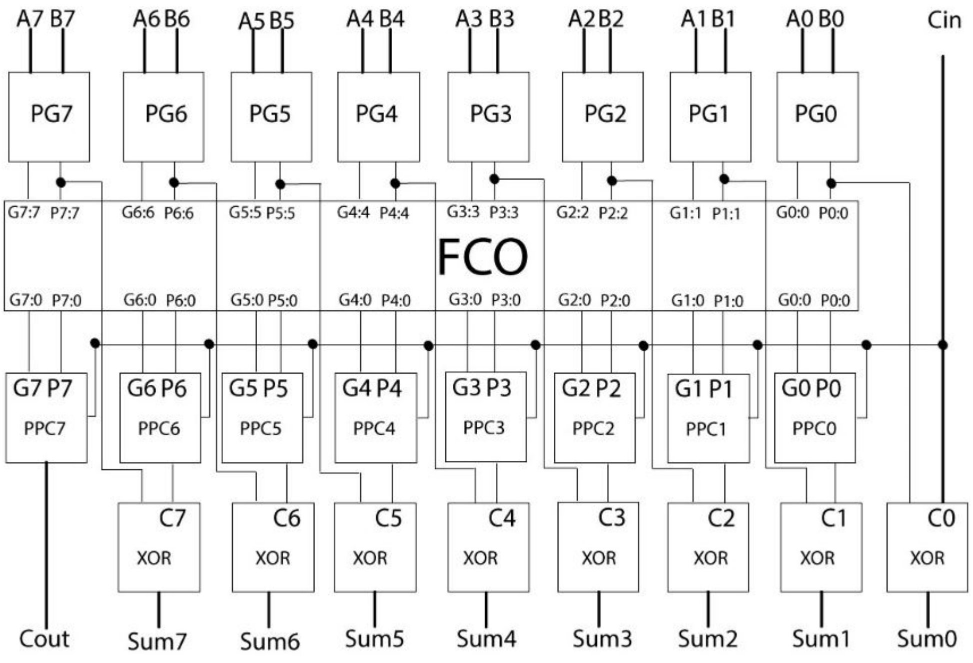


Figure 5. Architecture of 8-bit Kogge Stone Adder.

The architecture of KSA is quite similar to CLA. However, the fundamental carry operator (FCO) block lies between the PG blocks and PPC blocks. This block processes the propagator and generator bits in pre-processing structure. Figure 6 shows the architecture of FCO used in KSA.

In one bit KSA circuit, it requires 2 gate delays. However, it can be used to combine any P_i and G_i signals that are generated from PG block, or combine some P_i and G_i signals that are already combined. The combination does not affect the previous bits but only the level.

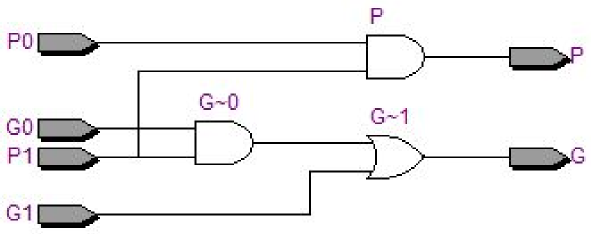


Figure 6. Architecture of FCO Block.

In addition, The PPC block in KSA is slightly different compared to CLA. The input to each bit of PPC block in KSA is the C_{in} which is decided by end-user or external factors. Unlike CLA, the input in PPC block requires the Cout from previous bit. With this architecture, the time delay in KSA will be in the optimum stage.

3. Methodology
The basic ripple carry adder, CLA, and KSA as explained in the previous section are implemented in Verilog and sent for both front-end and back-end digital IC design flow, including full validation and analysis for performance, power, and area. The adder from the standard cell library is instantiated directly using the Verilog code below:

```
module Default_adder (A, B, Cin, Sum, Cout);
parameter N=7;
input [N:0] A, B;
input Cin;
output [N:0] Sum;
output Cout;
assign (Cout, Sum) = A + B + Cin;
endmodule
```

3.1. EDA Tools Flow
The Synopsis and Mentor Graphics EDA tools are used for VLSI implementation and verifications of the adder architectures. Synopsys Design Vision and PrimeTime are used for front-end synthesis and timing analysis, while the Synopsys ICC is used for backend layout generation. Simulation is performed using Synopsys VCS, while DRC, LVS, and PEX are performed using Mentor Graphics Calibre. Figure 7 shows our tool flow.

4. Results and Discussion
This section presents results for the VLSI implementation of the adder architectures. Each adder architecture has been designed using Verilog and synthesized using full ASIC flow to obtain an IC layout. Figure 8 shows an example of a VLSI layout for the 32-bit KSA. All the results have also been analyzed post-layout for performance, area, and power. Registers are also inserted at input and output of each adder in order to obtain valid timing analysis data.

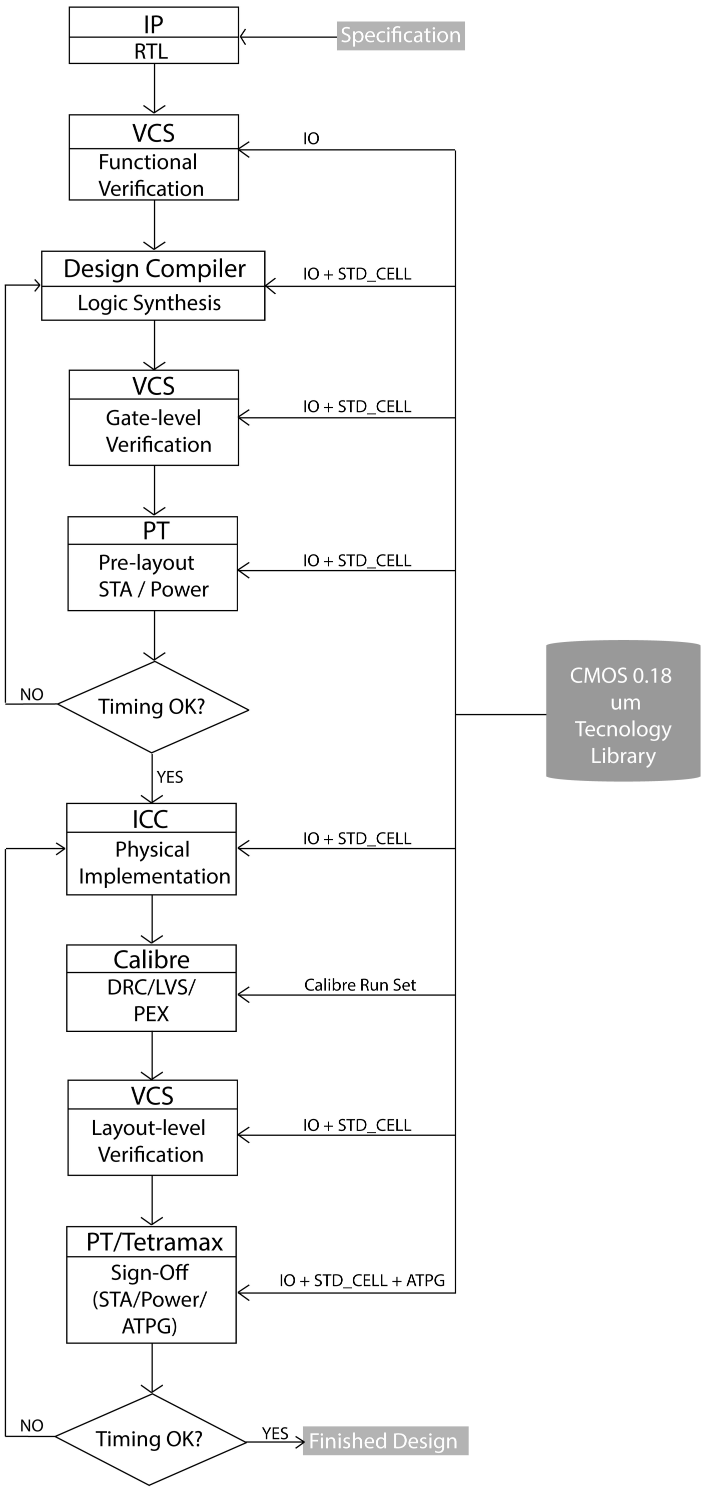


Figure 7. Synopsis design flow with Mentor Graphics tools for verifications

4.1. Propagation Delay

Propagation delay information is obtained from post-layout timing analysis. The required frequency is set to 100MHz. The slack value obtained is then used to estimate the actual

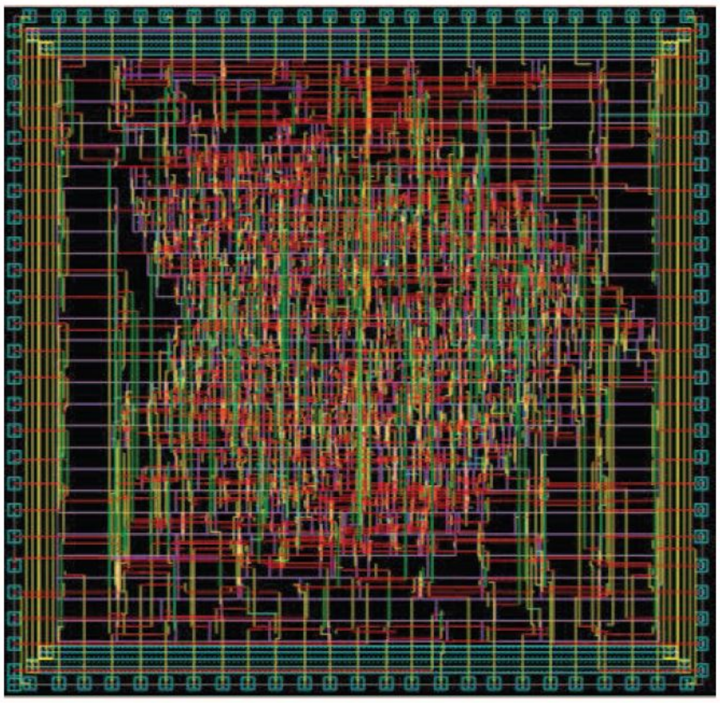
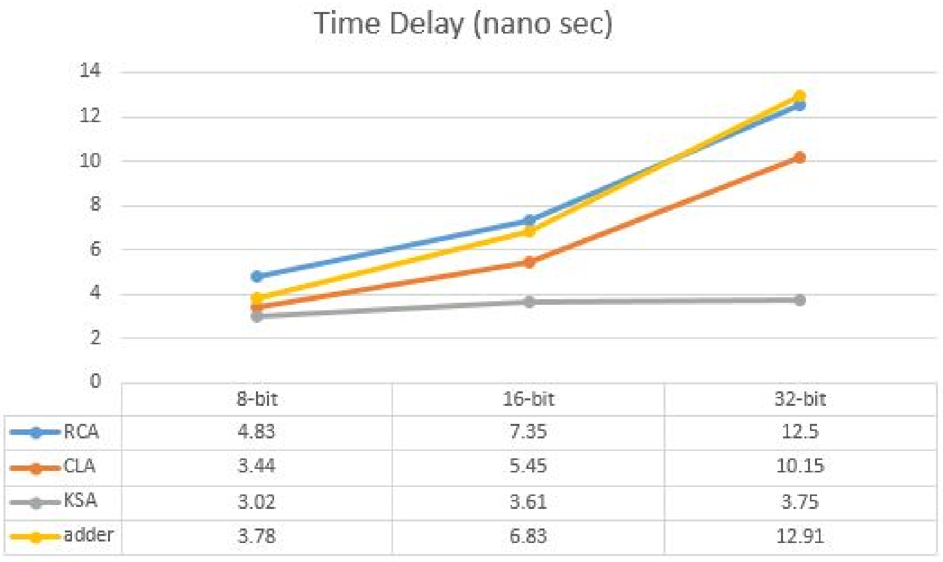


Figure 8. VLSI layout of a 32-bit KSA.

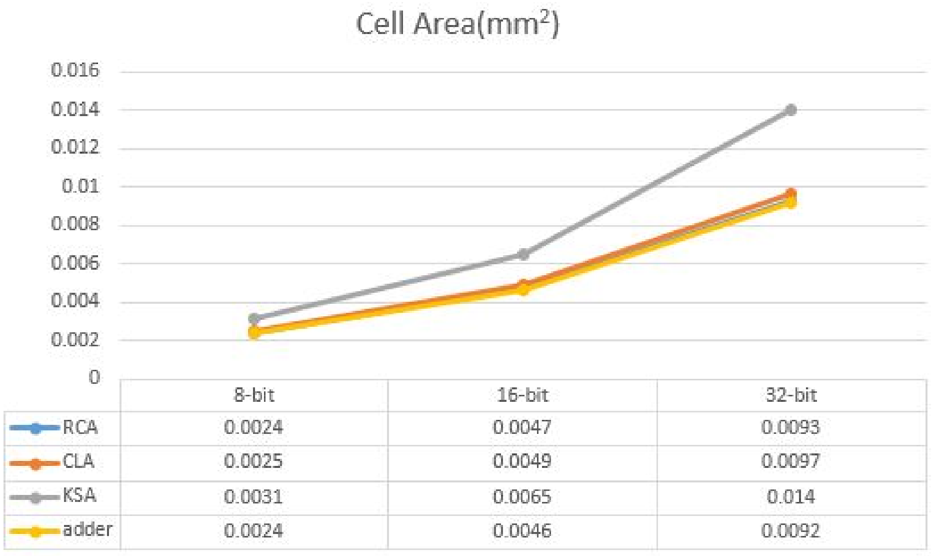
arrival time, which is then normalized to propagation delay. As shown in Figure 9, the KSA results in the lowest with almost constant delay from 8-bit to 32-bit. The higher the number of bits required, the further the distance between KSA and others adders. KSA is rougly 37% less delay in 8-bit and 70% less delay in 32-bit as compared to the worst case. In terms of maximum frequency, the KSA can run at an estimated maximum speed of $1/3.02ns = 331MHz$ for 8-bit KSA. As explained in section 2, the logic depth of KSA is constant regardless of the number of bits, which conforms to our experimental data.



	8-bit	16-bit	32-bit
RCA	4.83	7.35	12.5
CLA	3.44	5.45	10.15
KSA	3.02	3.61	3.75
adder	3.78	6.83	12.91

Figure 9. Time delays for each adder.

4.2. Cell Area
According to the results in cell area as shown in Figure 10, KSA requires the highest cell area. This is expected since extra FCO blocks are required in order to generate the sum out and C_{out} results in parallel. The area increase is up to 59% for 32-bit adders.



	8-bit	16-bit	32-bit
RCA	0.0024	0.0047	0.0093
CLA	0.0025	0.0049	0.0097
KSA	0.0031	0.0065	0.014
adder	0.0024	0.0046	0.0092

Figure 10. Cell Area coverage of each adder.

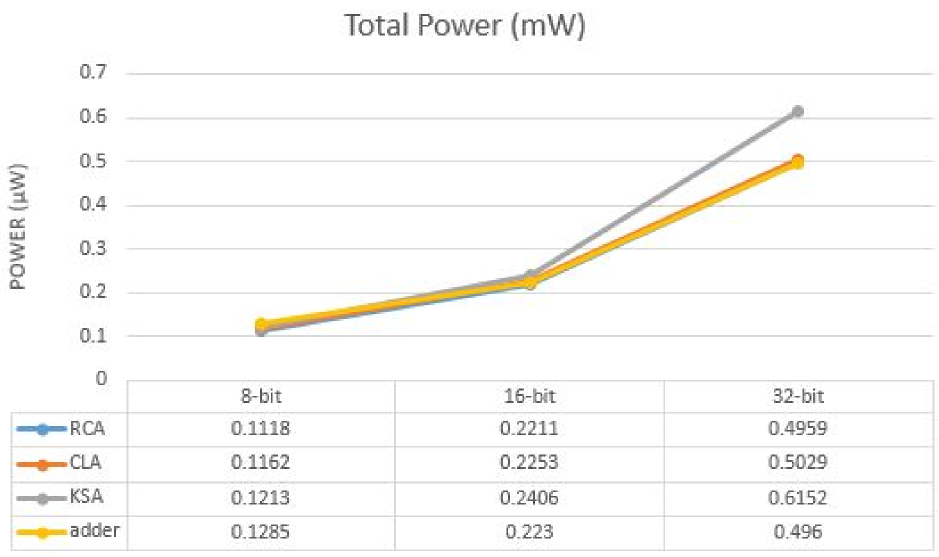
Cell count information for various adder architectures is given in Table 1.

4.3. Total Power
Power consumption information is obtained from Quality of Result (QoR) report for post-layout. It is taken as the sum of static and dynamic power. Results show that maximum increment is

Table 1. Cell count for various adder architectures.

Type of Adder	Cell Count (8-bit)	Cell Count (16-bit)	Cell Count (32-bit)
RCA	34	66	134
CLA	51	99	199
KSA	68	148	330
Default	28	52	104

up to 24% for 32-bit adder as compared between RCA and KSA. The total power is found to be almost the same for all adder architectures in 8-bit and 16-bit. However, the power consumption for KSA increases sharply when going to 32-bit. This is shown in figure 11.



	8-bit	16-bit	32-bit
RCA	0.1118	0.2211	0.4959
CLA	0.1162	0.2253	0.5029
KSA	0.1213	0.2406	0.6152
adder	0.1285	0.223	0.496

Figure 11. Power Consumption of each adder.

Another interesting criteria to evaluate is the energy efficiency. Energy can simply be defined as follows:

$$E = Pt \tag{5}$$

Where P is the total power and t is the time delay. Table 2 summarizes this information, where it can be seen that the KSA uses significantly less energy compared to other adders, with up to 64% less in the best case at 32-bit.

Table 2. Energy efficiency of various adder architectures.

Type of Adder	Energy (pJ) (8-bit)	Energy (pJ) (16-bit)	Energy (pJ) (32-bit)
RCA	0.54	1.63	6.19
CLA	0.40	1.23	5.10
KSA	0.37	0.87	2.31
Default	0.49	1.52	6.40

5. Conclusion

In this paper, we have designed a fast Kogge-Stone adder and implemented using 0.18um process technology. The high speed adder has been validated and compared against other adder architectures including the ripple carry adder, carry look-ahead adder, and the default adder from the standard cell library. All of these adder architectures are implemented using Verilog, and synthesized using both front-end and back-end tools from Synopsys to obtain the ASIC layout. The designs have also been validated using tools from Mentor Graphics. Results show a very significant improvement in time delay for the KSA as compared to other architectures, at the cost of relatively larger area and power consumption. This adder can be readily applied to speed critical applications in VLSI. Currently, work is ongoing to further improve the adder by custom VLSI design.

Acknowledgments

The authors would like to acknowledge UTM for providing the facilites for research, and also the Ministry of Education for providing the funds (Vot No 4F659).

References

[1] Bezati E, Yviquel H, Raulet M and Mattavelli M 2011 A unified hardware/software co-synthesis solution for signal processing system *Proceedings of the 2011 Conference on Design and Architectures for Signal and Image Processing (DASIP)* p1
[2] Xilinx 2017 ZedBoard Technical Specifications Internet: http://zedboard.org/content/zynq-fpga-fabric-clock-speed
[3] Lin Z and Chow C 2013 Zcluster: A zynq-based hadoop cluster *Field Programmable Technology (FPT) International Conference on. IEEE.* p450
[4] Powell A and Silage D 2015 Statistical performance of the ARM cortex A9 accelerator coherency port in the xilinx zynq SoC for real-time applications *ReConFigurable Computing and FPGAs (ReConFig) International Conference on* p1
[5] Sugimoto N and Amano H 2014 Hardware/software co-design architecture for Blokus Duo solver *Field-Programmable Technology (FPT) International Conference on* p358
[6] Ahmed K E and Farag M M 2015 Hardware/software co-design of a dynamically configurable SHA-3 System-on-Chip (SoC) *Electronics, Circuits, and Systems (ICECS)* p617
[7] ISO/IEC 2006 Information technology âĂŤ MPEG video technologies âĂŤ Part 2: Fixed-point 8x8 IDCT and DCT transforms ISO/IEC 23002-2
[8] Xilinx 2017 FPGA coprocessing for C/C++ programmers (part II) Internet: http://xillybus.com/tutorials/vivado-hls-c-fpga-howto-2
[9] Miano J 1999 Compressed Image File Formats âĂŞ JPEG, PNG, GIF, XBM, BMP Addison Wesley Longman Inc USA
[10] Chiper D F 2016 A new VLSI algorithm for a high-throughput implementation of type IV DCT *International Conference on Communications (COMM)* p17
[11] Arai Y, Agui T and Nakajima M 1988 A Fast DCT-SQ Scheme for Images *Transactions of IEICE* E71 1095
[12] Kovac M and Ranganathan K 1995 JAGUAR: A Fully Pipeline VLSI Architecture for JPEG Image Compression Standard *Proceedings of the IEEE* 83 247
[13] Bhaskaran V and Konstantinides K 1999 Image and Video Compression Standards Algorithms and Architectures âĂŞ Second Edition Kluwer Academic Publishers USA
[14] Zabidi N N and Ab Rahman A A 2017 VLSI Design of a Fast Pipelined 8x8 Discrete Cosine Transform *International Journal of Electrical and Computer Engineering (IJECE)* 7 1430
[15] Xilinx 2013 Xilinx Zynq-7000 All Programmable SoC Overview Xilinx Inc