FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 38, Nº 3, September 2025, pp. 457 - 468
https://doi.org/10.2298/FUEE2503457B

Original scientific paper

AUTOMATED SYNTHESIS OF A HIGH-SPEED ADDER

Padmanabhan Balasubramanian[1], Nikos E. Mastorakis[2]

[1]College of Computing and Data Science, Nanyang Technological University, Singapore
[2]Technical University of Sofia, English Language Faculty of Engineering (ELFE), Department of Industrial Engineering, Sofia, Bulgaria

ORCID iDs: Padmanabhan Balasubramanian https://orcid.org/0000-0001-9412-4773
Nikos E. Mastorakis N/A

Abstract. *At the gate level, the Kogge-Stone adder (KSA) is known to outperform many high-speed adders including other parallel prefix adders in terms of the speed performance. This paper presents a methodology to synthesize a new high-speed adder automatically, called the AHSA, using a logic synthesis tool. We describe what adder architectures can be input to a logic synthesis tool and what synthesis constraints should be specified so that the AHSA can be automatically synthesized. The AHSA is significant since it has a speed similar to that of the KSA while requiring less area and dissipating less power. In this paper, 32-bit addition serves as an example, and various adders belonging to different architectures were synthesized using a 28 nm Synopsys CMOS standard cell library. The design metrics estimated show that while the KSA has a 5.2% reduced delay than the AHSA, the AHSA occupies 29.1% less area and consumes 9.6% less power than the KSA. In terms of the figures of merit used for a digital circuit design such as power-delay product (PDP), area-delay product (ADP), and power-delay-area product (PDAP), the AHSA achieves a 4.7% reduced PDP, a 25.2% reduced ADP, and a 32.4% reduced PDAP compared to the KSA. This paper demonstrates that when speed is the key factor in an adder design, the AHSA is preferable to the KSA. Moreover, the AHSA is shown to be significantly faster than other high-speed adders at the gate level.*

Key words: *digital circuits, computer arithmetic, adder, logic design, CMOS, high-speed*

1. INTRODUCTION

High-speed adders are critical in digital circuits and computer architecture, primarily used for performing addition operations quickly. Some key uses of high-speed adders are given as follows. High-speed adders are important in arithmetic and logic units that perform arithmetic and logical operations in processors. High-speed adders are used in

Received November 30, 2024; revised March 13, 2025 and April 03, 2025; accepted April 06, 2025
Corresponding author: Padmanabhan Balasubramanian
College of Computing and Data Science, Nanyang Technological University, Singapore
E-mail: balasubramanian@ntu.edu.sg

digital signal processors for the fast computation of algorithms such as filtering and Fourier transforms. High-speed adders help to speed up the operation of embedded systems and general-purpose processors by speeding up arithmetic operations. High-speed adders support fast computations necessary for handling complex graphics in graphics processing units. Fast addition using high-speed adders is critical in cryptographic algorithms where performance can significantly impact security operations. High-speed adders are used in high-performance computing to perform complex mathematical computations where speed is vital. Further, high-speed adders are often implemented in application-specific integrated circuits and field-programmable gate arrays to optimize specific functions. Thus, high-speed adders play a fundamental role in enhancing the speed performance of digital electronic circuits and systems.

Addition and multiplication are fundamental arithmetic operations that consume significant power in computing systems. For instance, in [1], it was noted that in graphics processing units, arithmetic operations such as addition/multiplication were responsible for more than 70% of their power consumption. In fast Fourier transform processors, addition and multiplication account for approximately 80% of the total power [2]. Adders are crucial for performing arithmetic computations and are integral to the data paths of digital signal processors. Studies have shown that addition is often executed in real-time digital signal processing [3]. A study of an ARM processor's arithmetic and logic unit revealed that addition is involved in nearly 80% of its operations [4]. Consequently, developing high-speed adders is critical to improving the efficiency of digital systems.

Literature discusses several adder architectures [5,6], including the ripple carry adder (RCA) comprising a cascade of one-bit full adders or two-bit full adders [7], the conditional-sum adder (CSMA) [8], the carry skip adder (CSKA) [9], the carry-select adder (CSLA) constructed using two RCAs [10] or one RCA and an add-one circuit or a binary to excess-1 code converter [11,12], diverse carry look-ahead adders (CLAs) such as the conventional CLA [13], the delay-optimized CLA [14] and the new CLA [15], as well as a family of parallel prefix adders (PPAs) [16]. All these adder architectures are well documented in the literature, and their performance parameters were compared [15]. The PPA [16] is an advanced adder architecture, and several variants of PPAs have been explored, such as the Brent-Kung adder (BKA) [17], the Sklansky adder [18], and the Kogge-Stone adder (KSA) [19]. While all PPAs aim to accelerate addition through parallelized carry generation, each differs in structure, performance, and area utilization. The KSA, known for its highly parallel binary tree structure, performs prefix operations quickly, making it faster than the other PPAs. However, the KSA requires more area than other PPAs due to its extensive parallelism. Nevertheless, the KSA excels in scalability and handles large bit widths effectively,

This paper presents a method to automatically synthesize a new high-speed adder called AHSA that optimizes the trade-off between area, delay, and power, outperforming the KSA in terms of efficiency. While the KSA is widely acknowledged as one of the fastest gate-level adders, it often underperforms in terms of area and power consumption compared to other high-speed alternatives. In contrast, the newly synthesized adder (AHSA) achieves a speed performance comparable to the KSA but with reduced area and power requirements. The remainder of the paper is structured as follows: Section 2 discusses the methodology used to synthesize the AHSA with a synthesis tool. Section 3 presents the standard design metrics of various 32-bit adders, which were synthesized using a 28 nm Synopsys CMOS standard cell library and characterized using Synopsys EDA tools. Section 4 summarizes the contributions of this work.

2. AUTOMATED HIGH-SPEED ADDER SYNTHESIS

The output of a logic synthesis tool depends on the input provided, which may be described in a hardware description language, and the specific synthesis scripts that dictate whether the synthesis should be optimized for area, speed, or power. Through a series of experiments, we noted that certain high-speed adder architectures, when described in Verilog and synthesized under specific constraints using a logic synthesis tool (Synopsys DesignCompiler) can automatically generate a new high-speed adder, which we label as the Automatically generated High-Speed Adder (AHSA). The AHSA is observed to be closer in speed to the KSA but requires less area and dissipates less power. However, we also noted that not all high-speed adder architectures described in Verilog result in an AHSA after synthesis when using the same synthesis scripts. Therefore, this paper provides specific information on which adder architectures to use as input and outlines the necessary design constraints to specify to synthesize an AHSA.

Our experimentation has revealed that when certain adders are described structurally, i.e., using gate primitives in Verilog and synthesized using DesignCompiler by specifying the constraints 'set_dp_smartgen_options -optimize_for speed' and 'compile_ultra,' yield the AHSA. These adders include (i) an RCA composed of cascaded two-bit full adders, (ii) CLA architectures employing standard or delay-optimized CLA modules of uniform size [20], and (iii) an NCLA [15] employing delay-optimized CLA modules of varying sizes.

For reference, we have made available the structural Verilog code of a 32-bit CLA, constructed using eight 4-bit CLAs, and the synthesized gate-level netlist of the AHSA on GitHub [21], which is open for access. The synthesis was carried out using Synopsys DesignCompiler, utilizing the gates of a 28 nm Synopsys CMOS standard cell library [22].

For example, Figs. 1 and 2 illustrate standard 4-bit CLAs used to construct a conventional 32-bit CLA. In Fig. 1, the 4-bit CLA has no carry input, and in Fig. 2, the 4-bit CLA has a carry input. Here, A_{X+4} to A_X and B_{X+4} to B_X represent the augend inputs, C_X denotes the carry input, S_{X+4} to S_X represents the sum outputs, and C_{X+4} denotes the carry output. C_{X+3} to C_{X+1} denote internal carry signals. In Figs. 1 and 2, P_{X+3} to P_X denote carry-propagate logic, and G_{X+3} to G_X denote carry-generate logic. Assuming A_Y and B_Y to be the adder inputs, the carry-propagate and carry-generate logic are generally expressed as the logical exclusivity and logical conjunction of inputs as follows: $P_Y = A_Y \oplus B_Y$, and $G_Y = A_Y B_Y$. The generalized expressions for lookahead carry outputs with no carry input (i.e., $C_X = 0$), and the sum output of a 4-bit CLA is given by (1) to (5) below:

$$C_{X+1} = G_X \tag{1}$$

$$C_{X+2} = G_{X+1} + P_{X+1}G_X \tag{2}$$

$$C_{X+3} = G_{X+2} + P_{X+2}G_{X+1} + P_{X+2}P_{X+1}G_X \tag{3}$$

$$C_{X+4} = G_{X+3} + P_{X+3}G_{X+2} + P_{X+3}P_{X+2}G_{X+1} + P_{X+3}P_{X+2}P_{X+1}G_X \tag{4}$$

$$S_K = P_K \oplus C_K \tag{5}$$

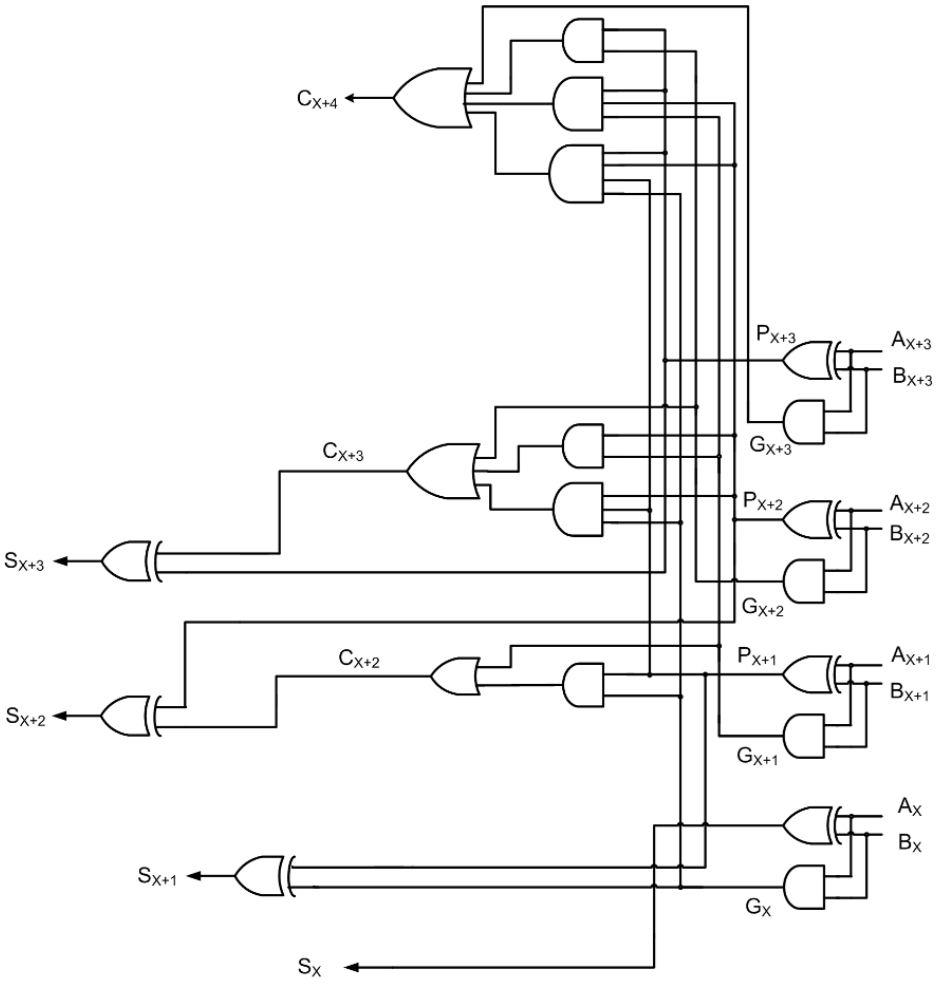

Fig. 1 Structural diagram of a 4-bit CLA with no carry input

The generalized expressions for lookahead carry outputs of a 4-bit CLA with a carry input (C_X) are given by (6) to (9) below:

$$C_{X+1} = G_X + P_X C_X \tag{6}$$

$$C_{X+2} = G_{X+1} + P_{X+1} G_X + P_{X+1} P_X C_X \tag{7}$$

$$C_{X+3} = G_{X+2} + P_{X+2} G_{X+1} + P_{X+2} P_{X+1} G_X + P_{X+2} P_{X+1} P_X C_X \tag{8}$$

$$C_{X+4} = G_{X+3} + P_{X+3} G_{X+2} + P_{X+3} P_{X+2} G_{X+1} + P_{X+3} P_{X+2} P_{X+1} G_X + P_{X+3} P_{X+2} P_{X+1} P_X C_X \tag{9}$$

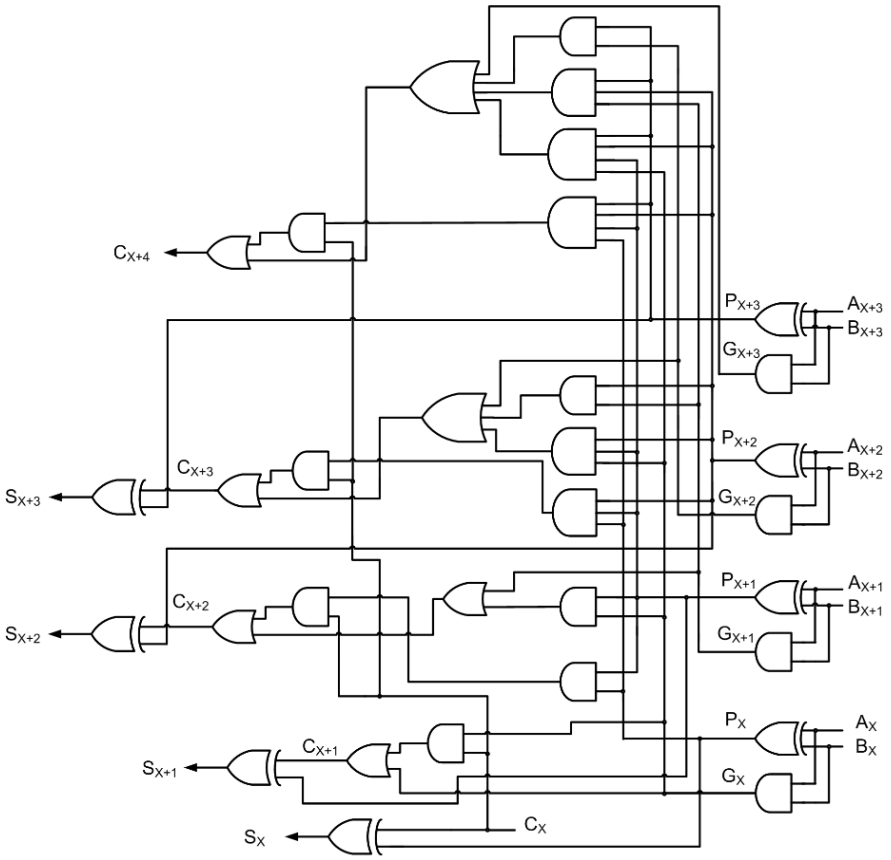

Fig. 2 Structural diagram of a 4-bit CLA having a carry input

After synthesizing a 32-bit AHSA using a standard digital cell library, its gate-level netlist was simulated for functionality using Synopsys VCS by supplying approximately a thousand random inputs at a nominal latency of 4 ns. An image showing a segment of simulation waveforms of the 32-bit AHSA is shown in Fig. 3 where a[31:0] and b[31:0] represent the adder's inputs, and sum[32:0] represents the adder's output. The adder inputs and output are shown in hexadecimal in Fig. 3. Four markers, namely M1, M2, M3, and M4, are highlighted in Fig. 3, which capture specific instances of the input-output waveforms. M1 highlights the scenario where inputs (EEEE EEEE) and (EFFF EEEF) are added, resulting in a sum of (1 DEEE DDDD). M2 highlights the scenario where (FFFF FFFF) and (FEEE FFFE) are added, resulting in a sum of (1 FEEE FFFD). M3 highlights the scenario where (0000 0000) and (0000 0000) are added, resulting in a sum of (0 0000 0000). Lastly, M4 highlights the scenario where (1111 1111) and (1111 1111) are added, resulting in a sum of (2222 2222).

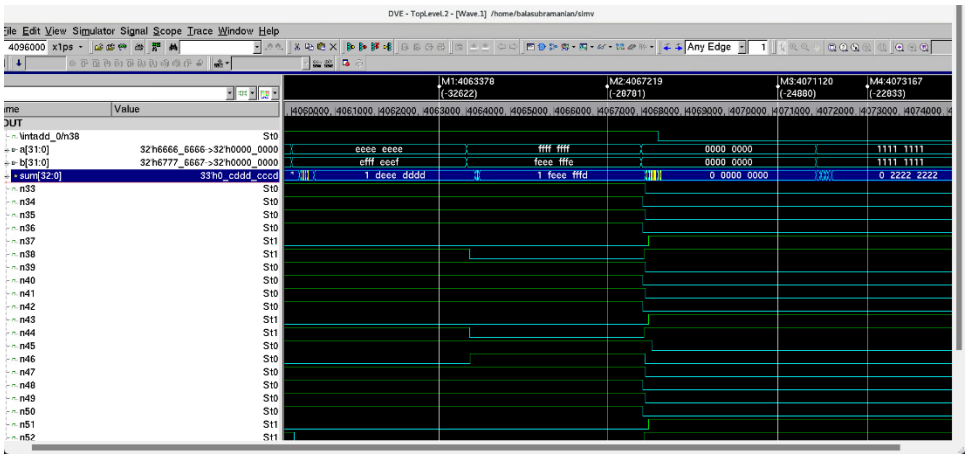

Fig. 3 An image showing a segment of simulation waveforms of the 32-bit AHSA, obtained using Synopsys VCS

3. Design Metrics of Adders

Here, we consider 32-bit addition as an example, although the addition of any size could be considered. Contrary to some existing works such as say [12] which considered different adder designs corresponding to one specific architecture viz. the CSLA for implementation and comparison, here, various 32-bit adders belonging to diverse architectures such as the RCA, CSKA, CSMA, CSLA, CLAs (CCLA, DCLA, and NCLA), and well-known PPAs were considered for implementation and comparison. To synthesize a 32-bit RCA, a 32-bit adder was described in data-flow style in Verilog using the arithmetic operator (+), and the 'compile_ultra' command of DesignCompiler was used for synthesis, resulting in an RCA comprising 31 full adders and a half adder logic.

All the high-speed adders were described at the gate level in Verilog for synthesis using a 28-nm CMOS standard digital cell library [22]. The synthesis targeted a typical process, voltage, and temperature condition of the cell library (supply voltage = 1.05 V; operating junction temperature = 25 °C). The operating condition chosen is 'ttlp05v25c', and the library specification used is 'saed32hvt_ttlp05v25c'.

Default wire load models were automatically chosen by DesignCompiler to estimate the delay and power consumption of interconnects (wires) during the synthesis process. Two common wire load models are ForQA and 8000. ForQA is primarily used for quality assurance (QA) testing. This model provides a conservative estimate of wire delay and capacitance for synthesizing designs, focusing on ensuring that the design will meet timing and power requirements across a wide range of conditions. It is typically used in environments where design quality needs to be validated against typical wire characteristics. The 8000 wire load model is more specific and widely used, typically for large designs. This model estimates the wire delay based on a set of empirical parameters and is named '8000' because it is often applied in designs with around 8000 gates. It provides more accurate estimations for wire delay and capacitance, especially for larger designs, making it more suited for general production use. Both models serve to help designers understand the impact of interconnects on overall performance, but they differ in their application contexts and accuracy, with ForQA

focusing on QA and 8000 offering more practical use in larger designs. Here, ForQA was used as the default wire load model for synthesis using the compile_ultra command; 8000 was used as the default wire load model for synthesis using the compile_ultra command with speed specified as the optimization goal; and the wire load model 8000 was used for the top module while the wire load model ForQA was used for the bottom (i.e., instantiated) modules for synthesis using the compile command with speed specified as the optimization goal.

A fanout-of-4 driving strength was uniformly associated with the sum bits of all the adders. During the synthesis, a virtual clock was used to constrain the adders' inputs and outputs. The clock does not form a part of the physical design, and it has no impact on the design metrics.

A 32-bit RCA implemented with a cascade of two-bit full adders, as outlined in [7], is referred to as RCA-2 in this work. We considered three types of CLAs: the conventional CLA (CCLA) consisting of standard and uniform-size CLA modules, the delay-optimized CLA (DCLA) comprising delay-optimized and uniform-size CLA modules, and the new CLA (NCLA) featuring delay-optimized and non-uniform-size CLA modules. The details of CCLAs and DCLAs are given in [20], and the details of NCLAs are given in [15]; hence, an interested reader is suggested to refer to the same for necessary information. Given that we consider the 32-bit addition as a representative example here, three 32-bit CCLAs were examined, each comprising a different number of CLAs: one with sixteen 2-bit CLAs (CCLA-2×16), another with eight 4-bit CLAs (CCLA-4×8), and another with four 8-bit CLAs (CCLA-8×4). Likewise, three 32-bit DCLAs were also considered, referred to as DCLA-2×16, DCLA-4×8, and DCLA-8×4 in the paper. Among several NCLAs, NCLA-8844422, which utilizes 2-bit, 4-bit, and 8-bit CLAs, was the best optimized [15]. Therefore, we considered this NCLA alone for this work. For the 32-bit KSA, we referenced the structural description provided in [23]. Regarding the CSLA, we followed the guidance given in [24], which recommended an 8-8-8-8 input partition to realize a delay-optimized CSLA. Additionally, we referred to the Synopsys DesignWare library that contains synthesis-ready models of some high-speed adders such as the Ling adder, CSMA, BKA, and the Sklansky adder. All the high-speed adders were synthesized using the 'compile' command, with speed set as the optimization goal in the first round. Next, they were all synthesized using the 'compile_ultra' command with speed set as the optimization goal in the second round.

For synthesis using DesignCompiler, the 'compile' command with speed optimization uses less aggressive optimization techniques, and its focus is on achieving a balance between timing and other design goals, such as area or power. The 'compile' command with speed optimization generally includes gate resizing, buffer insertion, basic logical restructuring, etc. On the other hand, the 'compile_ultra' command generally leverages advanced and more aggressive timing optimization techniques, including advanced retiming, multilevel optimization, critical path emphasis, etc. The 'compile_ultra' command with speed optimization primarily targets speed but also considers other metrics such as area and power. This partly explains why RCA-2, CCLAs, DCLAs, and the NCLA yield the AHSA when the synthesis was performed with speed specified as the optimization goal and the 'compile_ultra' command was used in DesignCompiler. The 32-bit CSKA yields a high-speed adder that has the same speed and dissipates the same power as the AHSA, but there is a slight variation in the area occupancy. Hence, the CSKA is said to yield the AHSA partly. However, the difference in area between the AHSA and the high-speed adder derived from the CSKA is negligible. Moreover, in the case of the Ling adder, CSLA, BKA, Sklansky adder, and KSA, some trade-offs in design metrics are noticed when synthesized using 'compile' and 'compile_ultra' commands. This is due to the native logic optimization algorithms embedded

in a (commercial) synthesis tool that are not accessible to an end user, so a trial-and-error approach may have to be adopted to achieve a preferred design outcome.

After synthesis using DesignCompiler, the total area of each adder, which includes their cell area and interconnect area, was estimated. The gate-level netlists of the adders obtained by synthesis were validated for functional correctness using Synopsys VCS, with a test bench comprising approximately one thousand random inputs. The inputs to the adders were provided at a nominal latency of 4 ns to accommodate the critical path delay of the slower RCA. The power dissipation of adders was estimated by considering their switching activity recorded via functional simulations performed using VCS. Synopsys PrimePower was used to estimate the total (average) power dissipation. The critical path delay of adders was determined using PrimeTime, with a virtual clock constraining the inputs and outputs of the adder. Since the virtual clock is not part of the actual design, it did not affect the design metrics. A fanout-of-4 drive strength was applied to the sum outputs of all adders. A default wire-load model was used during synthesis to account for interconnect and parasitic effects. The following versions of Synopsys EDA tools were utilized in this research: (i) DesignCompiler: Q-2019-12-SP5, (ii) PrimeTime (PrimePower): O-2018.06-SP5-2, and (iii) VCS: 2020_12_SP2_6.

The standard design metrics, namely total area (i.e., cells area + interconnect area), critical path delay, and total power dissipation of various adders synthesized, are given in Table 1. Table 1 presents three categories of adders synthesized using DesignCompiler by applying different synthesis settings to the Verilog description of adders. The first category comprises the conventional RCA obtained by synthesizing an adder described using the arithmetic operator (+) using the 'compile_ultra' command. The second category comprises the KSA and NCLA-8844422 synthesized using DesignCompiler using the 'compile' command with speed designated as the optimization goal. Reference [15] presented a comparison of design metrics of various adders belonging to diverse architectures, which were synthesized using the 'compile' command with speed set as the optimization goal. It was observed in [15] that the KSA is the fastest, although it occupies more area and dissipates more power than other adders, while NCLA-8844422 offers a good trade-off between delay, area, and energy compared to its counterparts. Hence, these two adders are of primary interest here, and therefore, their design metrics are shown in Table 1 for speed-oriented synthesis using the compile command. Also, we noted that the AHSA did not result based on the synthesis of any high-speed adder using the 'compile' command with the optimization goal set as speed. So, the design metrics of the rest of the high-speed adders, given in [14], are not repeated here. The third category of adders given in Table 1 was synthesized using DesignCompiler using the 'compile_ultra' command with speed set as the optimization goal. Instances of AHSA synthesis corresponding to the third category of adders are highlighted in boldface in Table 1.

Figure 4 shows the split-up of total power dissipation of various adders synthesized based on different synthesis settings using DesignCompiler. The summation of cell internal power and net switching power is referred to as the dynamic power, and the cell leakage power denotes the static power, as reported by Synopsys PrimePower. In Figure 4, three notations are used within brackets, namely 'CU', 'C&S', and 'CU&S' – these refer to the synthesis of adders using compile_ultra command, compile command with speed set as the optimization goal, and compile_ultra command with speed set as the optimization goal, respectively.

Table 1 Design attributes of various 32-bit adders estimated after synthesis using a 28 nm CMOS standard digital cell library by applying different synthesis settings

Adder input to the synthesis tool	Area (μm²)			Delay (ns)	Power (μW)	AHSA yielded
	Cells	Interconnect	Total			
RCA synthesized using the 'compile_ultra' command; no optimization goal specified						
Adder (+)	155.03	10.98	166.01	3.40	42.13	No
Synthesis using the 'compile' command; optimization goal – speed						
KSA	1014.29	174.43	1188.72	0.73	84.99	No
NCLA-8844422	476.52	53.21	529.73	0.99	50.00	No
Synthesis using the 'compile_ultra' command; optimization goal – speed						
Adder (+)	387.06	62.75	449.81	2.74	56.63	No
RCA-2	728.63	114.19	842.82	0.77	76.81	**Yes**
CSKA	729.91	114.18	844.09	0.77	76.81	Partly
CSMA	412.48	77.65	490.13	1.71	69.43	No
CSLA	444.75	72.41	517.16	1.62	66.47	No
Ling adder	391.89	76.10	467.99	2.56	64.19	No
CCLA-2×16	728.63	114.19	842.82	0.77	76.81	**Yes**
CCLA-4×8	728.63	114.19	842.82	0.77	76.81	**Yes**
CCLA-8×4	728.63	114.19	842.82	0.77	76.81	**Yes**
DCLA-2×16	728.63	114.19	842.82	0.77	76.81	**Yes**
DCLA-4×8	728.63	114.19	842.82	0.77	76.81	**Yes**
DCLA-8×4	728.63	114.19	842.82	0.77	76.81	**Yes**
NCLA-8844422	728.63	114.19	842.82	0.77	76.81	**Yes**
BKA	425.69	82.61	508.30	2.35	67.29	No
Sklansky adder	391.89	76.10	467.99	2.56	64.19	No
KSA	515.15	94.39	609.54	1.85	61.92	No

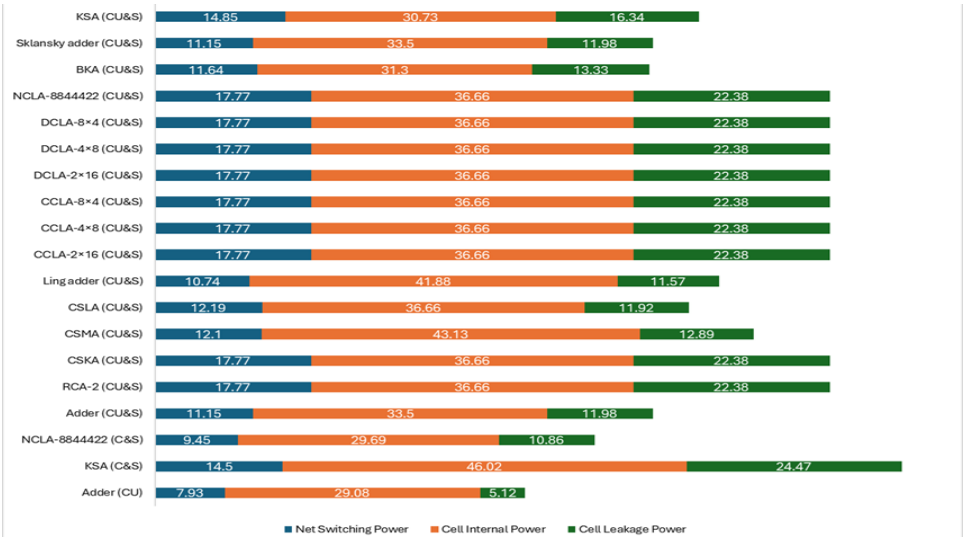

Fig. 4 Split-up of total power dissipation of various adders in terms of their dynamic and static power components (given in μW). The blue, orange, and green bars represent the net switching power, cell internal power, and cell leakage power, respectively. The net switching power and cell internal power together account for the dynamic power, and the cell leakage power accounts for the static power.

From Table 1, it is noted that the KSA synthesized using the 'compile' command with speed specified as the optimization goal, and the AHSA synthesized by applying the 'compile_ultra' command on select adders with speed specified as the optimization goal stand out from the rest in terms of speed. From Table 1, it may be understood that while the CSKA, CSMA, CSLA, Ling adder, CCLAs, DCLAs, NCLA (NCLA-8844422), BKA, and Sklansky adder are considered high-speed adders, the existing KSA and the newly presented AHSA may be categorized as 'very high-speed adders' due to their significantly faster performance. The gates present in the critical path of the 32-bit AHSA and the 32-bit KSA (synthesized by the 'compile' command), determined from the timing reports generated by PrimeTime, are expressed by (10) and (11).

$$D_{AHSA\_32b} = D_{NOR2X0} + (3\times D_{OAI21X1}) + (2\times D_{AOI21X1}) + D_{XNOR2X1} \quad (10)$$

$$D_{KSA\_32b} = (2\times D_{XOR2X1}) + (5\times D_{AO21X1}) + D_{AND2X1} \quad (11)$$

In (1) and (2), D_{NOR2X0} denotes the average delay of a 2-input NOR gate, $D_{OAI21X1}$ denotes the average delay of an OR-AND-INVERT complex gate, $D_{AOI21X1}$ denotes the average delay of an AND-OR-INVERT complex gate, $D_{XNOR2X1}$ denotes the average delay of a 2-input XNOR gate, D_{XOR2X1} denotes the average delay of a 2-input XOR gate, D_{AO21X1} denotes the average delay of an AND-OR complex gate, and D_{AND2X1} denotes the average delay of a 2-input AND gate. In the gate delay notations, the suffix 'X0' implies a gate drive strength of 0.5, indicating an associated capacitive load of 2 fF, and the suffix 'X1' implies a gate drive strength of 1, indicating an associated capacitive load of 4 fF.

Substituting the average delays of gates in the standard cell library [22] into (1) and (2), the theoretical critical path delays of 32-bit AHSA and 32-bit KSA are calculated as 0.599 ns and 0.585 ns. Though a theoretical calculation of critical path delays is approximate in the absence of interconnect and parasitic, nevertheless, the theoretical calculation points to a slightly lesser delay for the KSA compared to the AHSA which agrees with the trend noticed in the practical (physical) estimates of 0.73 ns for the former and 0.77 ns for the latter.

The AHSA demonstrates substantial reductions in critical path delay compared to other high-speed adders, as summarized below.

- 73.7% reduction compared to CSKA (synthesized by 'compile_ultra')
- 55% reduction compared to CSMA (synthesized by 'compile_ultra')
- 33.6% reduction compared to CSLA (synthesized by 'compile_ultra')
- 67.8% reduction compared to Ling adder (synthesized by 'compile_ultra')
- 33.6% reduction compared to CCLA-8×4 (synthesized by 'compile_ultra')
- 26.2% reduction compared to NCLA-8844422 (synthesized by 'compile_ultra')
- 68.2% reduction compared to BKA (synthesized by 'compile_ultra')
- 71.9% reduction compared to Sklansky adder (synthesized by 'compile_ultra')

From the perspective of speed, the AHSA and the KSA are comparable. The synthesis results show that the KSA achieves a 5.2% reduction in delay compared to the AHSA. However, the AHSA occupies 29.1% less area and consumes 9.6% less power than the KSA.

Three well-known figures of merit used for a digital logic design are power-delay product (PDP), area-delay product (ADP), and power-delay-area product (PDAP). PDP quantifies the energy consumed per switching event in a digital circuit, making it a key metric for energy-efficient design. A lower PDP indicates that the circuit consumes less power while maintaining fast switching, which is crucial for battery-powered and low-power applications. PDP is particularly useful for evaluating trade-offs in high-speed

circuits where power dissipation and operational speed are both critical factors. ADP measures the trade-off between circuit area and speed, helping designers optimize for both compactness and performance. A lower ADP signifies a design that is both smaller and faster, which is particularly important for system-on-chip and applications with stringent area constraints. This metric ensures that performance gains do not come at an excessive cost in chip real estate. PDAP integrates power, delay, and area into a single metric, offering a comprehensive evaluation of circuit efficiency. It is particularly useful in scenarios where designers should optimize for low power, high speed, and minimum area simultaneously, such as in energy-efficient processors and compact embedded systems. A lower PDAP is ideal for applications that require balanced power, performance, and silicon cost. We noted from Table 1 that the AHSA achieves a 4.7% reduced PDP, a 25.2% reduced ADP, and a 32.4% reduced PDAP compared to the KSA. When speed is the key factor in an adder design, the AHSA is preferable to the KSA. Moreover, the AHSA is shown to be significantly faster than other high-speed adders at the gate level.

4. CONCLUSION

This paper expounded the automatic synthesis of a very high-speed adder (called the AHSA), using a commercial logic synthesis tool. The AHSA demonstrated its competitive performance alongside the KSA, which is generally considered the fastest gate-level adder. Through experimentation, we observed that an RCA constructed via a cascade of two-bit full adders, or CLAs when given as inputs to the synthesis tool (DesignCompiler), and synthesized using the 'compile_ultra' command with speed specified as the optimization goal, the AHSA was generated. The functionality of the AHSA, along with several other high-speed adders, was verified through simulation, and the design metrics were estimated. Notably, the AHSA is significantly faster than all other high-speed adders, except the KSA. Nevertheless, the critical path delays of AHSA and KSA are comparable, and AHSA occupies less area and has less power dissipation. Therefore, it can be concluded that when speed is the primary criterion for an adder design, which is the norm in a high-performance computing environment, the AHSA emerges as a preferred option to the KSA. We also provided an example adder code that was input to the synthesis tool and the gate-level netlist of the AHSA synthesized in a publicly accessible format through GitHub [21], which could be useful to researchers in academia/industry. Although we discussed the synthesis of AHSA in this paper using Synopsys DesignCompiler, there is no information about whether the AHSA could be synthesized using any other logic synthesis tool (open-source or commercial) based on specific synthesis constraints as reported in this work in the existing literature. Hence, we leave this issue to the research community to ponder further work in this aspect. Also, the scope for further work exists whereby the AHSA may be considered for the realization of arithmetic circuits such as multipliers, multiply-and-accumulate units, etc., to investigate the significance of the AHSA.

REFERENCES

[1] H. Zhang, M. Putic and J. Lach, "Low Power GPGPU Computation with Imprecise Hardware", In Proceedings of the 51st Design Automation Conference, 2014, pp. 1-6.

[2] L. Wanhammar, *DSP Integrated Circuits*, Cambridge, MA, USA: Academic Press, 1999.

[3] D. C. Chen, L. M. Guerra, E. H. Ng, M. Potkonjak, D. P. Schultz and J. M. Rabaey, "An Integrated System for Rapid Prototyping of High Performance Algorithm Specific Data Paths", In Proceedings of the International Conference on Application Specific Array Processors, 1992, pp. 134-148.
[4] J. D. Garside, "A CMOS VLSI Implementation of an Asynchronous ALU", In Proceedings of the IFIP WG10.5 Working Conference on Asynchronous Design Methodologies, 1993, 181-192.
[5] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*, New York, USA: Prentice Hall, 1994.
[6] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Burlington, MA, USA: Morgan Kaufmann Publishers, 2004.
[7] P. Balasubramanian, K. Prasad and N. E. Mastorakis, "A Standard Cell Based Synchronous Dual-Bit Adder with Embedded Carry Look-Ahead", *WSEAS Trans. Circ. Syst.*, vol. 9, pp. 736-745, Dec. 2010.
[8] J. Sklansky, "Conditional-Sum Addition Logic", *IRE Trans. Electron. Comput.*, vol. EC-9, pp. 226-231, June 1960.
[9] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 1st Edition, New York, USA: Oxford University Press, 2000.
[10] O. J. Bedrij, "Carry-Select Adder", *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 340-346, June 1962.
[11] T.-Y. Chang and M.-J. Hsiao, "Carry-Select Adder using Single Ripple-Carry Adder", *Electron. Lett.*, vol. 34, pp. 2101-2103, Oct. 1998.
[12] B. Ramkumar and H. M. Kittur, "Low-Power and Area-Efficient Carry Select Adder", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, pp. 371-375, Feb. 2012.
[13] G. B. Rosenberger, *Simultaneous Carry Adder*, U.S. Patent 2966305, 27 December 1960.
[14] H. Ling, "High-Speed Binary Adder", *IBM J. Res. Dev.*, vol. 25, pp. 156-166, May 1981.
[15] P. Balasubramanian and D. L. Maskell, "A New Carry Look-Ahead Adder Architecture Optimized for Speed and Energy", *Electronics*, vol. 13, p. 3668, Sept. 2024.
[16] S. Knowles, "A Family of Adders", In Proceedings of the 15th IEEE Symposium on Computer Arithmetic, 2001, pp. 1–8.
[17] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Trans. Comput.*, vol. C-31, pp. 260-264, March 1982.
[18] J. Sklansky, "An Evaluation of Several Two-Summand Binary Adders", *IRE Trans. Electron. Comput.*, vol. EC-9, pp. 213-226, June 1960.
[19] 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. 100, pp. 786-793, Aug. 1973.
[20] P. Balasubramanian and N. E. Mastorakis, "High-Speed and Energy-Efficient Carry Look-Ahead Adder", *J. Low Power Electron. Appl.*, vol. 12, p. 46, Aug. 2022.
[21] [Online] Available at: https://github.com/balaccds/32-bit-carry-look-ahead-adder
[22] Synopsys SAED_EDK32/28_CORE Databook, Revision 1.0.0. January 2012.
[23] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh and P. Lofti-Kamran, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing", *IEEE Design & Test*, vol. 34, pp. 60-68, Apr. 2017.
[24] P. Balasubramanian and N. Mastorakis, "Performance Comparison of Carry-Lookahead and Carry-Select Adders Based on Accurate and Approximate Additions", *Electronics*, vol. 7, p. 369, Dec. 2018.