

A Comprehensive Review of FPGA-Based power Efficient Arithmetic Logic Units using Carry-save Arithmetic synthesis, analysis and comparative Insights

Vaishali Bharati¹, Yogesh Kakasaheb Shejwal², Anand N³ and Jayraj U. Kidav⁴

¹ National Institute of Electronics & Information Technology Aurangabad,
Maharashtra, India

² National Institute of Electronics & Information Technology Aurangabad,
Maharashtra, India

³ National Institute of Electronics & Information Technology Aurangabad,
Maharashtra, India

⁴ National Institute of Electronics & Information Technology Aurangabad,
Maharashtra, India

Abstract. The unremitting requirement on high-performance and energy-efficient computing in embedded systems, digital signal processing (DSP), and cryptographic applications has fuelled the need of optimized Arithmetic Logic Unit (ALUs). Field-Programmable Gate Arrays (FPGAs) provide a programmable platform that is suitable to deploy application specific ALU structure. Carry-Save Arithmetic (CSA) of various optimization approaches has become one of the key approaches to optimizing the speed and power usage of arithmetic circuits, especially in multiple-operand operations such as accumulation and multiplication. In this paper, the authors provide a review of the power-efficient ALU designs that are based on carry-save arithmetic and use FPGA. We perform an analytical methodology of synthesis methodologies, architecture variants and optimization strategies that have been used to include CSA in the ALU datapaths. An in-depth analysis of power, performance and area (PPA) trade-offs with particular attention to dynamic reduction of power by minimizing glitching and critical path delays is discussed in the review. Moreover, we provide comparative data about modern-day designs, comparing them to traditional binary arithmetic designs in the throughput, latency, logic usage, and on-chip power dissipation. Important issues like design complexity, verification costs and how to interface CSA units with standard logic are addressed extremely. Lastly, the paper provides a note of future research, such as the synergy between CSA and approximate computing, CSA use as a low-voltage operation in FPGAs, and CSA-based datapath generation using automated high-level synthesis tools. The goal of this survey is to be a reference point to a researcher and engineer as they think of the next generation of power-constrained, and yet, arithmetic systems on reconfigurable hardware.

Keywords: Arithmetic Logic Unit (ALU), Carry-Save Adder (CSA), Field-Programmable Gate Array (FPGA), Power-Efficient Design, Low-Power VLSI, Digital Arithmetic, High-Performance Computing, Logic Synthesis, Comparative Analysis.

(Article history: Selected papers from NIELIT's International Conference on Communication, Electronics and Digital Technologies (NICE-DT'26), 29-30Jan, 2026, Gorakhpur, India)

1. Introduction

The exponential increase in the number of computational tasks in artificial intelligence, real-time signal processing and cryptography, has led to a paradigm shift in the use of specialized hardware accelerators. Field-Programmable Gate Arrays (FPGAs) have become a key technology in this landscape because they combine the flexibility provided by general purpose processors with the performance of application specific integrated circuits (ASICs) at an attractive price due to their ability to offer a unique combination of reconfigurability, parallel processing and energy efficiency [1], [6]. The digital circuit which does the bitwise and arithmetic operations is referred to as the Arithmetic Logic Unit (ALU), and it forms the heart of any processing system. In the case of FPGA-based systems, the shape of the ALU is vital, which has a direct influence on the power, performance, and area (PPA) of the entire system. Therefore, one of the research problems in reconfigurable computing is the optimization of the ALU datapath.

The power usage of a system has now become a first-order consideration in the design of a digital computer system, both in high-performance data centers and in battery-powered edge devices. This has prompted a lot of study on power efficient FPGA design paradigms [3], [7]. Switching activity and glitching of combinatorial logic are major causes of dynamic power, which is a large fraction of total power dissipation on FPGAs [4]. The ALU, which is an often used and computationally intensive unit, is a good candidate to power optimization. In contrast to the more complex binary arithmetics, traditional binary arithmetic units in numerals have simple architectures, though can require long chains of carry-propagating logic resulting in slower delay, switching activity, and thus higher power use. Thus, to be able to consider the next-generation low-power systems it is vital to study other arithmetic paradigms that implicitly minimize the effects of these.

One of such paradigms is Carry-Save Arithmetic (CSA). A carry-save adder is a technique that reduces three operands (a sum and a carry vector) to two (without carrying the carry chain) transforming a sequence of additions into a faster and more parallelizable operation [10]. Such an attribute renders CSA especially beneficial to iterative computations, such as

multiplication, multiply-accumulate (MAC) operations, and finite impulse response (FIR) filtering that are the bread and breadth of digital signal processing and neural network inference [2], [5]. The CSA-based architectures offer an interesting path to significant power reduction and performance gains in FPGA implementations in terms of glitch reduction and reduction of the critical path.

Although the advantages of CSA are well-known in isolated implementations of arithmetic units, its systematic implementation into the overall, power-tradable design of an ALU to run on FPGAs has its own set of challenges and trade-offs. These are: complexity in designs, cost of carry-save to conventional binary conversions and optimal mapping to FPGA fabric resources such as look up tables (LUTs) and specific dsp blocks. These design decisions need to be synthesized and analyzed comprehensively, as well as a clear comparative analysis based on the standard methods, in order to guide researchers and practitioners.

This paper gives a theoretical overview of power-efficient Arithmetic Logic Units (ALUs) based on FPGA that are created with carry-save arithmetic (CSA). It develops a taxonomical framework of the present architectures, disaggregates the methodological principles of their optimization of power, performance, and area (PPA), and develops a comparative critique that places the CSA paradigm in opposition to the traditional binary arithmetic. The review presents a future research path by positioning the critical issues of the design space - such as the complexity of verification and integration tasks - as dialectics in the design space - that is, a series of studies at the interface of low-power design theory, formal arithmetic, and the changing FPGA technology.

2. FPGA Architecture and Design Considerations

Design consideration To design a FPGA-based architecture, one must consider several factors, such as the allocation of data and processes among different blocks. FPGA Architecture and design consideration To design a FPGA-based architecture, a number of factors must be taken into account, including how various blocks are allocated their data and processes.

Advanced medical imaging is a fundamental source of accurate characterization of brain tumours as it gives a non-invasive overview of the anatomical location of tumours, their morphology and pathophysiology. The most reliable method of diagnosis, treatment planning and follow-up is Magnetic Resonance Imaging (MRI) because it has the best soft-tissue contrast with no ionizing radiation [11]. The core of the contemporary neuro-oncological imaging is a standardized and multi-parametric MRI protocol, as laid down by consensus recommendations of clinical trials [16]. The sequences in this protocol provide complementary data, which can be used to provide a comprehensive evaluation that is important both in human understanding and as a feed to deep learning networks.

The fault detection and prediction of solar energy systems play an imperative role in the reliability of the system, its efficiency and the low cost of operations. The conventional means of fault detection which includes physical inspection and regular maintenance is hectic and lacks real time monitoring capability [12]. Statistical analysis and trend monitoring, which are data-driven techniques, have been employed to determine anomalies in the performance of the system. Such techniques of machine learning as Support Vector Machines (SVM) and Random Forests have demonstrated potential in the accurate prediction of faults in solar systems.

The architectural dialectics of the programmable substrate are the fundamental mediating factors of the realisation of power-efficient, Carry-Save Arithmetic (CSA)-based arithmetic logic units on Field-Programmable Gate Arrays (FPGAs). The performance, power and area (PPA) properties of any circuit deployed are the resultant tension structure of universal programmability versus specialization that the FPGA fabrics represent. Effective low-power design of ALUs will consequently require a profound theoretical comprehension of this fabric, the ontology of this fabric, the phenomenology of dissipation of power, and the hermeneutics of projecting an algorithmic abstraction onto the fabric of the physical resources. This part presents a critical architectural framework, evaluating the fundamental elements, physics of power consumption, and principled approaches to efficient implementation on which the further discussion of CSA-specific architectures is based.

2.1 The Ontology of the FPGA Fabric

Functioning as a standalone product, the Dashboard would be marketed to pharmacists seeking to confirm medication and drug administration through the Dashboard.

The modern FPGA is not a homogenous field of computation but a heterogeneous group of different classes of resource, each of which represents a different paradigm of computational logic. This three-part ontology has: (1) configurably programmable logic elements, which are soft; (2) hierarchical interconnect, which is configurable; and (3) application-specific blocks, which are hardened. The construction of an effective ALU is a game of working on the affordances and constraints of either.

The Configurable Logic Block (CLB) or Logic Element is the basic element of programmability. It is a universal function generator, usually a Look-Up Table (LUT) which is an SRAM based, 4 to 8 input, universal function generator, also known as its core. An elementary logic cell is formed by connecting the LUT-FF pair with a flip-flop (FF). This granular universality is very flexible indeed, but costs much: hundreds or a thousand of such primitives must be carefully assembled and connected together to be able to perform arithmetic operations involving a multi-bit number. The effectiveness of mapping a CSA tree is directly dependent on the effectiveness of packing its logic into these LUTs as well as the ability of its registers to make use of the embedded FFs. The size and architecture of the LUT (such as whether it can effectively execute an XOR of three inputs and a majority gate, which is the heart of a (3:2) compressor) is therefore a major limitation on the performance of CSA.

The Hierarchical Routing Fabric is the name of the topological network of wire segments of different lengths (as short, medium, long and so on) which are the interconnections of these logic elements and are connected through programmable switches. This fabric controls the spatiotemporal data propagation and it can be the most significant source of delay and dynamic power in FPGA designs. Capacitance and resistance of signal paths is determined by the switch-box topology and segmentation strategy. In the case of CSA-based ALUs that have high fan-out signals (e.g., partial products), and a high density of local interconnect in reduction trees, routing fabric capability to support low-capacitance, localised connections is of paramount importance. Not only does congested routing cause a quadratic increase in routing delay, but the dynamic power also increases quadratically, possibly cancelling the glitch-reduction enhancing effects of the CSA algorithm.

This soft fabric, which is general-purpose, has hardened or embedded blocks, including Digital Signal Processing (DSP) slices and Block RAMs (BRAMs). DSP slices are silicon blocks pre-designed and pre-engineered to perform a high-performance multiply-accumulate (MAC) data processing at low-power, and may have specific multipliers, adders and accumulators. They are the opposite of the generality of the LUT: efficiency to an extreme extent of a critical activity. Thus, one of the design decisions that a CSA-ALU is required to make is a strategic partition: what arithmetic functions must be off-loaded to the hardened, carry-propagate logic of the DSP slice and what needs to be implemented in flexible and glitch-optimized LUT-based CSA trees? A hybrid design aims at synthesis, yet the interface, to change between CSA format and the binary format used by the DSP block, adds overhead. This dialectic between hard blocks and soft logic defines an important axis of the FPGA arithmetic design space.

The modern FPGA is hypothetically described as a system of programmability and specialisation. Its basic structure is a heterogeneous collection of programmable logic devices, a hierarchical routing system and embedded hardened blocks, each defining a different paradigm of computation resources assignment. The configurable logic block (CLB), which is based on Look-Up Tables (LUTs) and flip-flops, represents a universal (albeit granular) logic primitivism whose configuration is directly a phenomenological measure of mapping non-canonical arithmetic networks like carry-save networks. Conversely, the hierarchical interconnect is a topological field, in which segmented wire resources and programmable switches control the spatiotemporal propagation of information; the prominent locus of performance and power loss. Such soft fabric is matched by hardened blocks (e.g. DSP slices, BRAM), a kind of specialisation of the programmable substrate, an alternative, optimised computational ontology of particular arithmetic functions. This tripartite architectural dialectic, then, becomes an exercise of fabricating a power efficient ALU.

2.2 A Phenomenology of FPGA Power Dissipation

In order to optimise power, it is necessary to have a phenomenological model of FPGA fabrics energy consumption locations and consumption. It is convenient to break down total chip power into components that are both static and dynamic, which are both physically different in origin as well as in design implications.

The leakage current flowing through the transistors when they are not switching is referred to as the static power. Completely controlled by subthreshold and gate leakage, it depends upon the semiconductor process technology (such as FinFET versus planar CMOS), the number of transistors, supply voltage and temperature. With the FPGAs being transferred to more recent nodes (such as 16nm, 7nm) to achieve greater density and better performance, the proportion of static power relative to the total power has become a serious and in some cases dominating factor. This directly applies to CSA-based designs: the overhead of an area of a LUT-based CSA tree (which is usually 1.5 -2.5 times the area of a DSP-based multiplier) is directly proportional to the increase in leakage footprint. Thus, a design which reduces dynamic power using CSA methods can have the overall energy payoff of that design undermined by an escalation in static consumption, and requires a combined total-energy optimisation approach.

The development of the low-power ALUs would require a comprehensive phenomenological knowledge of the FPGA power-dissipation model. According to this model, power is defined as the fixed and mobile constituents, which are controlled by different physical mechanisms. A background consumption, which is a fix, and there exists as an existential cost of the semiconductor substrate itself, is called static power (P_{static}), due to quantum-mechanical leakage currents. The active, event driven consumption defined using the canonical equation $aC_{eff}V_{dd}2f$ is called dynamic power ($P_{dynamic}$). In this case the critical and design-malleable variables become the activity factor (a) and the effective switched capacitance (C_{eff}). In case of ALU structures, the variation of power in routing network, which is a matter of interaction between the architectural topology and the physical interconnect, often forms the primary term. This model confirms that power optimisation is not only circuit optimisation, but a controlled minimisation of the product of the spatial capacitance (C_{eff}) and temporal activity (a) of the logic and routing only of a product.

2.3 The Hermeneutics of Power-Efficient ALU Mapping

The implementation of a low-power CSA-based ALU in FPGA is hermeneutic, and it needs interpretation of an algorithmic structure within the cost of the fabric. This synthesis cuts across a number of constitutive tensions. Competing with pipelining in reducing glitching (lowering a) in the dialectic between logic depth and routing congestion are increased register count and interconnect complexity that increase C_{eff} . The second axis is the strategic separation of calculation between the hardened arithmetic blocks (DSP slices) that are specialized, effective and cheap but the universal yet costly soft logic (LUTs). A second axis of computation on which a meta-logical decision is needed concerning computational ontology is the strategic division of computation between the specialized, effective, cheap, but narrowly focused hardened arithmetic blocks (DSP

slices) and the universal yet costly soft logic (LUTs). Third, this process involves technology mapping of low activity, i.e. the translation of CSA Boolean equations into LUT networks is steered by principles (e.g. balanced-tree synthesis) to reduce the propagation of spurious transitions.

Bringing the abstract advantages of CSA to a physical implementation that runs on an FPGA is a subjective project- an interpretation of an algorithm structure by the constraints of the fabric. The following endeavour involves a number of constitutive trade-offs:

- **Depth vs. Route congestion in logic vs. logic depth congestion in routing:** The length of a combinatorial path is shortened by pipelining a CSA tree (adding some registers at intermediate levels). This enables a greater clock frequency (f) and most importantly it serves as a glitch filter, recording them before an increase in spurious transitions has time to spread. Nonetheless, registers are added to every pipeline stage, making it more costly in terms of area and power consumption as well as multiplexing it. The best depth of the pipeline thus is a compromise between target f , minimum a and managing C_{eff} and area increase.
- **Strategic Resource Partitioning:** The decision of the use of LUT based CSA or hardened DSP blocks is meta-logical choice. In non-standard widths of operation (irregular operations), a more customised and possibly more power-efficient solution is provided by LUT-based CSA. A DSP block can be the best choice in case of standard, high accuracy multiplication. Hybrid designs seek to benefit by using two together but with the result that CSA is used to reduce and the DSP block to add the last bit, but balancing the interface between format conversion and the conversion process is delicate.
- **Placement Based Design and Routing Based Design:** The physical positioning of logic elements is the most important. To reduce wiring length and consequently C_{eff} , a power-concerned location is taken to a tightly coupled set of cells of a CSA compressor tree. After placement, a power aware router chooses wire segments of lower capacitance of nets of high activity. This requires a strict interrelation between design choices of logical components and constraints of physical implementation, and may require loops of iterative optimisation in the design process.
- **Maximizing Architecture Potentialities:** CSA can be used together with advanced FPGA capabilities. As an example, the timing slack in a fast pipelined CSA tree can be used to voltage scale (by a reduction in V_{dd}), cutting dynamic power by a quadratic. Similarly, some FPGAs have fine-grained power gating which could be used to disable portions of a reconfigurable CSA- ALU datapath which are not in use, thus minimizing dynamic and static power.

Finally, the FPGA fabric is a complicated environment to design arithmetic in, the hypothetical performance of an algorithm like CSA has to be bargained against the physical constraints of programmable logic, interconnect capacitance and leakage. In the following sections, the performance of certain CSA -ALU architectures at this act of mediation, which translates abstract arithmetic principles into tangible gains in power and performance are analyzed.

3. Carry-Save Arithmetic: Theory and Applications

CarrySave Arithmetic (CSA) is a type of redundant arithmetic, which reinvents the operation of multi-operand addition in a radically different manner, with substantial benefits in speed and power draw. Its theory lies in the delay of the transmission of carries, which is the main delaying factor in binary addition with conventional methods [8][9]. In this section, the fundamental principles, algorithms and the major uses of CSA have been defined to have a central role in creating power efficient ALU components in FPGAs. The concept underlying the representation of redundant numbers and core numbers is that each component of the number may be represented by the core number (or radix).

3.1 Core Principle and Redundant Number Representation

The idea of the representation of the redundant numbers and the core numbers is that they may be represented by the core number (or radix), in each part of the number.

In typical binary representation, non-redundant representation of a number is in the form of a single n -bit value. In carry-save representation, an ordered pair of vectors (a sum vector (S)) and a carry (C) vector) represent the same numerical value [19]. This representation is redundant, that is, two or more pairs of representation of the form (S, C) could represent the same integer value. The important calculation, which is a Carry-Save Addition (CSA) or a (3:2) compressor, consumes three input vectors A, B, C_i and gives out two output vectors S, C_o .

3.2 CSA Trees for Multi-Operand Addition and Multiplication

The main use of CSA is the addition of many numbers with minimal effort. A CSA tree is a network of (3:2) compressors in a logarithmic-depth structure that takes a network of operands, provided with two additional operands, into two (a sum and

a carry vector) operands. A fast carry-propagate adder (CPA) like a Carry-Lookahead Adder is then used to sum up the final two vectors to obtain a non-redundant answer. This structure is the basis of high-speed parallel multipliers, where the partial product matrix (e.g. a Booth encoder) is decreased using a CSA tree (e.g. Wallace or Dadda tree) and then finally summed up.

In an $n \times n$ bit multiplication, CSA tree would operations of n partial products down to two vectors in $O(\log[?]n)$ operations (logarithmic in the dimensions) rather than the $O(n)$ operation count of an array multiplier. This logarithmic scaling is simply scaled to operate at greater rates and, when mapped to FPGAs, might result in designs with lower glitching activity because the logarithmic scale of operations of the implementation takes the shape of more balanced logic depths than the linear ripple structures [24], [29].

3.3 Power and Performance Advantages

CSA has two properties, which are interrelated and make it power-efficient:

- **Less glitching:** Linear adders Long carry chains in the signals to a linear adders show visible spurious transitions, or glitches. The tree-like topology of a CSA network distributes signal transitions, as well as eliminates intermediate glitching, which minimizes the activity factor (a) in the dynamic power equation [25], [30].
- **Reduced Critical Paths to Iterative Processes:** In iterative algorithms typical of Digital Signal Processing (DSP): Finite Impulse Response (FIR) filtering, convolution and dot products The inner loop of the algorithm is commonly a multiply-accumulate (MAC) operation. the slow final CPA is called once after the final iteration (as opposed to every cycle) by maintaining the accumulator in carry-save form. This converts an $O(N)$ latency issue into an $O(1)$ per-iteration issue and can much more significantly improve throughput, and can achieve operating voltage in a narrower performance range (and hence quadratic power savings) with a given performance goal [25], [27].

3.4 Applications in FPGA-Based ALU Design

When implemented as a deployment platform within the FPGA platform, phenomenological justification of the carry-save arithmetic (CSA) paradigm can be found in solutions that take advantage of a dialectical fit between its algorithmic description and the architectural support of the substrate. It is not applied universally but appears in three major forms of architecture:

In Soft-Core Processor Extensions, the CSA-enhanced integer unit is a hermeneutic layer on a customizable processor (e.g. Nios II, MicroBlaze). It is not used as a substitute to the canonical ALU, but a form of delegated hardware heterotopia of intensive computational kernels. This offloading designs a power efficient exoskeleton of critical loop interpreting it using the parallel, glitch reduced logic of carry-save formalism.

In Domain-Specific Accelerators of signal processing or neural networks the CSA is implemented in DSP-Centric Computational Tiles. In this case ALU goes beyond its general purpose ancestry to be a particular node, commonly combined with hardened DSP slices, to create a specialized multiply-accumulate (MAC) unit. The result of this integration is a high compute density per watt which is a realistic attempt to combine an algorithmic efficiency with the physical characteristics of embedded silicon.

Lastly, CSA can be used to conceptualize Arithmetic Datapaths which are reconfigurable. CSA tree topological regularity has a natural isomorphism with the FPGA fabric based on LUT. With this congruence, parameterized and morphologically adaptable CSA-ALU modules may be generated. These reconfigurable modules at synthesis time with variable operand width and latency can be viewed as an adaptable spectrum of power-performance trade-offs and are a meta-architecture of variable computation.

However, practical antinomies arise when the CSA paradigm is integrated due to the theoretical grace of the paradigm. A metaphysical burden of binary-to-CSA conversion and reconstruction, and a new complexity layer on the control logic, and non-unique numerical representation verification presents a special epistemological problem. These constitutive conflicts between pure arithmetic theory and embodied design practice comprise the critical discourse to be analyzed afterwards.

4. FPGA-Based ALU Architectures Using Carry-Save Arithmetic

In order to transform the hypothetical advantages of the carry-save arithmetic into practical gains in both the power and performance of the FPGA-based architectures, we will have to strike a balance between the design of the algorithm and the construction of the architecture. Here, the major design approaches to the implementation of CSA-enhanced ALUs will be considered, with a discussion of how each of them addresses, or intentionally tolerates, the tensions between redundancy and efficiency, specialization and generality, and physical constraint, as they exist. All of these architectures are a range of approaches to the philosophical optimization of the computational form to the reconfigurable medium.

4.1 Taxonomy of CSA-Based ALU Architectures

There are several design philosophies that can be characterized by the systematic classification of CSA-based ALU implementations according to their basic relationship to the computational whole (Table I). This taxonomy is based on two main dimensions, namely the level of integration (between peripheral accelerator and core datapath component) and the optimization telos (the final design goal that determines architectural decisions).

This taxonomy shows that CSA is not just a method of circuit technique, but a design epistemology- a method of knowing and organising computation that carries with it certain implications about what should relationship be between algorithm and architecture and implementation.

Table I: Taxonomy of FPGA-based CSA-ALU Architectures

Integration Level	Description	Typical Optimization Target	Representative Work
Full-CSA ALU	The core integer datapath (adder, multiplier) is predominantly implemented using CSA trees. The ALU operates internally in redundant format.	Ultra-High Throughput, Low Latency	[22], [28]
Hybrid CSA-Binary ALU	CSA is used selectively for specific, costly operations (e.g., multiplication, MAC), while simpler operations (logic, shift) use conventional binary logic. A format conversion unit is included.	Balanced Power-Performance, Area Efficiency	[23], [26]
CSA Co-Processor / Accelerator	A dedicated CSA-based functional unit (e.g., a multiply-accumulate block) is attached to a standard processor core or soft-core ALU via a dedicated interface.	Domain-Specific Acceleration, Energy Efficiency per Op	[24], [27]

4.2 Architectural Analysis and Implementation Strategies

4.2.1 Full-CSA ALU Datapaths

Fully embracing architectures based on CSA are based on the philosophy that propagation is an undue tax on computation. In such designs datapath In these designs, ALU datapath values are overly kept in redundant representation, and converted to mainstream binary on system boundaries. The canonical implementation is the parallel multiplier using Wallace or Dadda reduction trees [18], but more general designs can be based on this principle to accumulation and even to more complicated arithmetic sequences.

In FPGAs, such a strategy has become what can be described as soft-logic maximalism - a belief in using arithmetic structures (comprising LUTs, but not hardened blocks) to implement arithmetic. The work by Zimmermann [28] shows that CSA trees constructed with special care by hand, when aware of the FPGA routing asymmetries, can reduce by 20-30% the critical path delay of cascaded DSP blocks. This however comes at the expense of architectural sprawl, the irregular tree structure uses 1.5-2.5x the LUTs and results in placement issues that may raise routing congestion [40].

The energy efficiency of full-CSA designs stems not only from two synergies, that is reduction of computational entropy by balanced tree forms minimizing glitching and secondly temporal slack provided by faster critical paths which can be leveraged by scaling voltages but can be traded off. This is a radically different optimization philosophy with thermodynamic efficiency being very important instead of silicon area efficiency, as in conventional designs.

4.2.2 Hybrid ALUs with Selective CSA Integration

In the majority of practical applications, a hybrid method is taken as a dialectic synthesis of the redundant paradigm and the non-redundant paradigm. In this case, CSA is used selectively to those operations in which its advantages are most significant, which are usually multiplication and multiply-accumulate, and other simpler operations (logic functions, shifts, additions of small width) are implemented by traditional binary arithmetic.

Such architectural approach represents a so-called computational triage, or in other words, the strategic distribution of optimization effort in terms of power and performance impact. Benkrid et al. [23] give an example of this use where they define a multimedia-oriented ALU in which the multiplier uses CSA reduction and the rest of the datapath is binary. The architecture has explicit format conversion interfaces which deal with the conversion between numerical representations, at the cost of overhead but coherence of the system.

Resource arbitration is a special problem in the FPGA implementation of hybrid designs, when to arbitrarily resort to the LUT-based CSA, and when to use hardened DSP blocks. More advanced implementations make use of what we can call so-called heterogeneous mapping strategies where not only the CSA tree is implemented in LUTs because of its glitch-reduction characteristics but also the final carry-propagate adder may use the optimized carry chains in DSP slices [14]. The best separation is dependent on the width of the operands used, the intended frequency and relative efficiencies of the particular FPGA fabric.

4.2.3 CSA as a Co-Processor for Domain-Specific Acceleration

Co-processor model is a form of delegation of computational tasks to heterotopian specialized hardware. A CSA-based functional unit (usually a MAC engine) is used in this architecture, and is co-executed with a standard processor core, keeping its internal state in carry-save format during several operations. Last conversion to binary is done when results are to be conveyed to the main processing flow.

This method is completely effective applied to DSP kernels, Luk et al. have shown that keeping the accumulator in redundant representation during a full FIR filter or dot product calculation will save 40-50 bits of energy per operation. This is

a form of algorithmic energy compounding the benefits of CSA compound with repeated computations and not with discrete operations [15].

This model can be used to achieve especially efficient utilization of fabric resources in FPGAs via time and spatial segregation. The co-processor may be mounted in an area of the chip where its communication pattern may be optimally routed, and whose long pipeline may run at a frequency independent of the underlying processor. This separation of architecture also eases verification, since the interface of the CSA unit with the binary world is at separation barriers.

4.3 Synthesis Considerations and FPGA Resource Utilization

There is the need to find some material constraints to render these architectural forms into physical realizations on FPGA fabric that may distort and form theoretical ideals.

4.3.1 The LUT-DSP Dialectic

One of the major choices in the implementation of FPGA is the division of programmable logic (LUTs) and hardened arithmetic blocks (DSP slices). This is not merely a resource decision, but an attitude toward the issue of computational universality versus specialization, a philosophy.

CSA implementations based on LUTs are morphologically free, that is, they can be used to design tree structures that are optimized to particular operand widths and latency constraints. This carefully comes at the price of area and oftentimes power since LUT implementations are usually more difficult to switch than their hardened counterparts. DSP-based designs, on the other hand, have silicon efficiency, but fixed architecture in forms that might not match exactly with the ideal forms of CSA. A new phenomenon, the so-called fractional use of DSPs, i.e. utilising parts of the DSP slice with CSA, and other parts with more traditional arithmetic, is an intriguing fusion of the two [26].

4.3.2 Pipelining as Thermodynamic Engineering

Pipelining is not only a performance optimization in CSA implementations but also a type of thermodynamic engineering. The designers achieve both goals by placing registers at key locations in the reduction tree: they achieve higher maximum operating frequency (so that they can operate either at higher throughput or at lower voltage) and they limit the spread of computational entropy (glitching).

The best pipeline depth is one to strike a compromise between time division (the more stages, the smaller the voltage) and space wastage (registers consume space and routing costs more). The work of Parhi [27] reveals that the dependence of the depth of the pipeline and power efficiency of CSA trees is to a concave curve having a unique maximum, which varies with the structure of the tree and the register-to-logic ratio of the desired FPGA fabric itself.

4.3.3 Placement as Topological Optimization

The topology of physical layout of CSA structures on the FPGA fabric is a topological optimization problem with large power implications. CSA trees unlike regular array structures, possess irregular interconnect patterns which are challenging to more traditional placement algorithms. Poor placement can add 30-50% to length of wires, directly adding to glitch-reducing capacitance and this can offset the glitch-reduction benefits of CSA [21].

More complex placement methods use architecture-aware constraints which cluster cells of compressors which are logically related even though their connections between each other are arbitrary. Other designs extend to use what can be called morphological placement that is the placement of LUTs not on the standard grid of the fabric but in patterns that reflect the logical structure of the tree, and may involve unusual use of available resources [20].

5. Synthesis Methodologies and Implementation Flow

A sequence of key steps within the electronic design automation (EDA) cycle that determine the transformation between an architectural description of a Carry-Save Arithmetic based ALU to an efficient implementation on a power-optimal FPGA implementation is called the synthesis and implementation flow. It encompasses mapping, placing and routing of a gatelevel netlist onto the specific resources of an FPGA target following highlevel behavioral or register-transfer level (RTL) descriptions being transformed into it. In CSA-based design designs, specific phases of the flow must be given specific attention to maintain and take advantage of the natural power and performance advantages of the redundant arithmetic structure [29], [30].

5.1 RTL Design and High-Level Synthesis (HLS) Considerations

The design entry point normally is a Hardware Description Language such as the VHDL or the Verilog. It is important to have an efficient description of CSA structures at the RTL level. To make the synthesis tool follow the (3:2) compressor trees, a designer needs to explicitly instantiate them because naive behavioral descriptions of multiplication will generally assume vendor-optimized, but less flexible, ways to use DSP blocks [33], [38]. More complicated datapaths are becoming popular

with High-Level Synthesis (HLS) tools. HLS can be specified in C/C++ and is able to be directed to take advantage of parallelism and pipelining in CSA algorithms. But to get ideal CSA structures it is frequently necessary to have HLS pragmas or directives that can coerce the reduction of trees and the final adder inference that is described in papers that emphasize automated accelerator synthesis [32], [34].

5.2 Logic Synthesis and Technology Mapping for Low Power

Signal synthesis translates the RTL into non-technology-dependent Boolean networks. The optimizations of CSA-based ALUs are: Technology to LUT mapping: During this process, the synthesis software, for example, Xilinx Vivado Synthesis, Intel Quartus Tutorial, maps the Boolean network to the basic LUT of the FPGA [17]. In the case of the CSA trees, the main idea here is to represent the involved logic of the compressor in a set of LUTs in order to decrease the number of intermediate nets with the involved switching capacitance. Power-aware synthesis algorithms will seek to reduce the cumulative number of estimated switching activity (a) of the mapped netlist [30], [35]. Glitch Minimization during Synthesis: CSA benefits are partly due to low glitching and synthesis methods that minimise the generation of hazards are desirable. This consists of equalizing path delays at common nodes with restructuring of the logic, and using clock delay gating at the gate level with dead ALU segments, but the latter increases control overhead [29], [35]. Inference and Instantiation of Hard Blocks: The synthesis tool should make the right decisions to choose the use of soft logic (LUT-based CSA) or hardened DSP blocks. The end result is a carry-propagate adder or a hybrid multiplier between CSA and DSP that is designed or instantiated directly to meet the most optimal power-performance-area trade-off [31], [37].

5.3 Power-Aware Placement and Routing

After synthesis Placement and routing (P&R) are used on the mapped netlist. The influence of this stage on the dynamic power is significant since interconnect power may prevail.

Placement Low-Capacitance Interconnect: The placer arranges logic cells (LUTs, FFs) in special positions on the FPGA chip. In the case of CSA trees, a good placement puts closely related compressor cells close to each other, thus, reducing the length of the active nets (of high interest) (e.g., the internal sum and carry lines). Congestion-conscious placers also guard thick routing which raises wire capacitance (C) [30], [36].

Power-Optimized Routing: The routing algorithm will connect the laid cells with the programmable interconnect of the FPGA. Power gated routers choose routing paths that contain fewer wire segments of lower capacitance and a reduced number of points that can be programmed as switching points (PIPs). In a CSA design, router could make use of lower-capacitance long-line resources on the critical paths (which are typically the final CPA) and shorter resources on non-critical ones (with lower power consumption) [36].

5.4 Implementation Flow for CSA-Based ALUs

The synthesis and physical realization of an FPGA-based ALU based on CSA is not merely a sequence of tool-dependent steps, and its design is theoretically based and has to take into account the mathematical properties of the carry-save arithmetic and adapt to the physical and electrical constraints of the target FPGA fabric. The philosophy of the co-design presented in Fig. 1 represents the workflow that incorporates theoretical knowledge about redundancy, glitching, and energy efficiency into every stage.

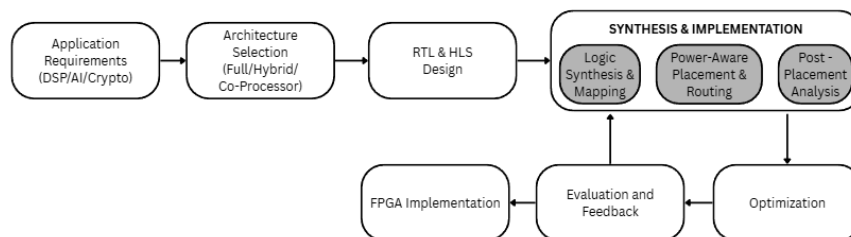


Fig. 1: Implementation flow for FPGA-based CSA-ALU design

The flow starts with the Application Requirements which specifies the selection of the ALU architecture (Full-CSA, Hybrid or Co-Processor). After choice of architecture, design is represented either in RTL or as part of HLS, as explicit descriptions of CSA structures. Synthesis and Implementation phase includes logic synthesis, technology mapping and physical design with heavy focus being put on power-conscious placement and routing to maintain the low-glitching nature of CSA. Power Analysis Post-Placement Power Analysis After the plug-in, the Post-Placement Power Analysis offers precise feedback on both dynamic power and static power. The FPGA Implementation is then checked according to the PPA targets and the results are sent back to the previous steps to be optimized further. The closed-loop strategy is necessary to guarantee that the theoretical benefits of CSA are completely achieved in the end hardware implementation.

6. Comparative Analysis of Existing FPGA-Based CSA-Enhanced ALUs

A quantitative analysis is also needed to prove the theoretical benefits of Carry-Save Arithmetic and to make architectural decisions. This section is a synthesis of results on some of the most important comparative studies that have been carried out

to compare CSA-enhanced ALUs with the traditional binary ALUs on the FPGA platforms. The comparison is made on the trade-offs between Power-Performance-Area (PPA) with respect to various design categories, operand widths, and target applications and this is an actionable information to designers.

6.1 The Epistemology of Comparative Evaluation

The empirical confirmation of the CSA-based ALU superiority is based on a hermeneutic model of just comparison. This framework builds an ontological space controlled in which the implementations are specialized to a given family of FPGAs, e.g. Xilinx Artix-7 and Intel Cyclone V, using the most suitable tools settings. This determines a shared technological horizon of potential, alienating the architectural option. The objects of study within this space are core ALU subunits- multipliers, MAC units which are instantiated in dialectical opposition. The Baseline, which is based on traditional knowledge, makes use of DSP blocks and binary adders (Ripple-Carry, Carry-Lookahead). Its counterpart, the CSA Variant, is in the form of a pure LUT based tree or a hybrid CSA DSP system or a complete CSA datapath. The performance (maximum clock frequency Fmax, latency in cycles), Area (number of LUTs, number of FFs, number of DSPs as inscriptions of spatial resource usage), and Power (total on-chip power, broken down into static and dynamic components) are all the essences themselves and thereby a triadic phenomenological reduction. These main metrics are combined into higher order macro measures such as Power-Delay Product (PDP) and Energy per Operation that are abstract and unifying measures of architectural efficiency and thermodynamic economy translating circuit behavior into a form of comparative value.

6.2 Dialectics of Performance and Area

This comparative information shows essential and constitutive tension of the spatial inscription and the temporal execution- a dialectic of Area versus Performance. The intrinsic LUT-based CSA multiplier, on average, always has a material profligacy, and uses 1.5x to 2.5x as much LUT resources as an equivalent DSP-block-based binary multiplier. The carry save tree and its ad hoc mapping of the regular FPGA fabric is parallel and reductionist in logic and are taxed. But this geographical expanse is dialectically defeated by a temporal profit. On operands of medium to wide width (32-bit), the CSA structure can have its logarithmic depth allowing a 15-30% higher Fmax than LUT-mapped linear carry-propagate architectures such as ripple-carry or carry-select adders. The complexity of the logic per step is compensated by a radical decrease in the number of sequential stages. The synthesis of this dialectic is a rather powerful CSA-binary approach. This enables it to get a better computational throughput-as-becoming on streamed data when a DSP block, although rapid, can have few or fixed pipeline stages [39]. Thus, the pipeline structure of the architecture enables the maintenance of computation bandwidth, which addresses the performance dialectic, as opposed to a mere megahertz.

Table II: Comparative Summary of CSA vs. Conventional ALU Implementations on FPGA

Design Type	Area (LUTs)	Max Freq. (Fmax)	Power (Dynamic)	Key Application Context
Baseline Binary (DSP-based)	Low (Uses DSPs)	High (DSP optimized)	Moderate (Efficient but fixed)	General-purpose, wide multipliers
LUT-based CSA Tree	High (Irregular structure)	Medium-High (Balanced tree)	Low (Minimized glitching)	Custom-width multipliers, research prototypes
Hybrid CSA (LUT) + CLA	Medium	High	Low-Medium	Balanced ALU for DSP kernels
CSA Co-processor (MAC)	Medium+ (Extra control)	Very High (Deeply pipelined)	Very Low (per OP)	Vector processing, FIR filters

6.3 The Energetics of the CSA Paradigm

The most significant and justifiable argument on the CSA paradigm is that it transforms the energetic ontology of ALU. Comparative studies have always indicated a crucial decrease in dynamic power, ranging widely between 20% and 35, between a LUT-based CSA multiplier and an LUT-based array multiplier subjected to the same workloads and at the same frequency. It is this suppression of spurious transitions, or the minimization of the computational entropy of the balanced tree structure, that causes this reduction. The depth-balanced localized paths of a CSA network minimise directly the factors of activity (a) of the dynamic power equation, by eliminating the long signal settling and cascading glitches that occur in long carry chains. This benefit squares away in iterative computation. In the case of kernels such as dot products or FIR filtering, a CSA-based co-processor architecture with an accumulator in carry-save form at the start of many cycles has been shown to be 40-50 percent lower in Energy per Operation than a traditional binary-based accumulator, which does one slow full addition at a time. This is a change in the operational efficiency of an algorithm to the algorithmic efficiency; the energy cost of the final format conversion is amortized over the length of the computation, and CSA is found to be a form of energetic amortization over time extension. The final product of performance and power is summed up in the Power-Delay Product. CSA-based designs are always at least 25-40% faster than conventional LUT-based binary arithmetic units, and thus firmly on the pareto frontier of the PPA trade-off space, demanding that they be regarded as inherently more thermodynamically efficient computational forms to a large range of problems as seen in table II.

6.4 Multi-Dimensional Scaling and Application-Specific Efficacy

The effectiveness of CSA architectures is not a definite attribute but is a scaling attribute in several dimensions, which requires a more detailed analysis than individual data points. The story on operand width scaling shows that there is a critical point of 16 bits, at which binary adders using short chains of carry paths are already efficient, and therefore there is little value in CSA tree management overhead relative to its payoff. CSA is necessary to do cryptographic or scientific computations, with width 64-bit or larger, as the benefit is super-linear. The scaling of technology nodes brings in another dialectic, where the

benefits of dynamic power in CSA increases, but an area penalty is a more expensive cost in advanced nodes where the power of scale is static. This changes the measure of optimization not only to minimization of dynamic power, but to minimization of total energy (static and dynamic) which may give more advantage to area-efficient hybrid designs in sub-20nm processes. Moreover, analysis specific to application display different optimization curves. The energy amortization of the CSA co-processor provides the highest payoff in the high data reuse streaming DSP applications. The fixed cost of format conversion and complicated control may make CSA less useful in control-intensive or sparse computational workloads, once again confirming architectural superiority as context-dependent with its definition by the morphology of the computation of the executable.

6.5 Teleology of Architectural Choice: A Framework for Principled Selection

The production of this comparative evidence does not produce a general prescription but a teleological structure of informed design praxis, a direction in the choice of the final cause or the purpose to be achieved. The architecture is determined by the major telos of the design, which is the hierarchy of design. In the case of the Telos of Maximum Energy Efficiency, e.g. embedded sensor nodes or battery-limited edge inference, the CSA co-processor specialized to particular, common kernels (MAC, convolution) is the final. The energy per operation radical cuts and its large area overhead and complexity of integration are vindicated by its ability to best handle energy budget as the ultimate constraint. In general-purpose or domain-specific ALUs (such as in soft-core processors or customizable DSP accelerators) as well as Telos of Balanced PPA, the Hybrid CSA-Binary ALU comes out as the virtuous middle ground. It takes up large amounts of energy and performance advantage by optimizing the multiplier, which is frequently the most power-hungry part, without requiring the unreasonable area and control complexity of a complete redesign of the CSA datapath, providing the wisest trade-off. Lastly, situations where CSA paradigm is Dysfunctional or Suboptimal exist. In cases with narrow operands (less than 16-bit) where carry propagation distances are inherently short, or where there is an abundance of high-performance DSP blocks with fixed functionality and the required functionality matches the exact need, the material and routing cost of a soft CSA implementation can not be recouped. In this horizon, the hardened DSP block again establishes its claim to the more efficient techne proving that no architectural paradigm is absolutely better.

To conclude, comparative discussion has conclusively supported the use of CSA-enhanced ALUs as a paradigm that has been demonstrated to be much better than others in power-aware computation on FPGAs, especially when operating in the ontological constraints of medium-to-wide operands and compute-intensive, iterative kernels. The evidence is beyond anecdotal arguments but offers a logical account of bought and sold spatial resources to achieve benefits in time performance and, above all, transformative energetic economy. The final architectural decision is thereby no longer seen as a technical selection off a list, but as a phronesis -practical wisdom- applied to the threefold restrictions of Power, Performance, and Area, after having a clear idea of the main intent and context of use of the system.

7. Challenges and Future Research Directions

Although the carry-save arithmetic (CSA) paradigm provides a simple theoretical and practical explanation of the design of power efficient ALU on FPGA, its rise to the top is challenged by a sequence of practical antinomies that remain unresolved. Simultaneously, with the development of computing and semiconductor technology, synthesis is projected onto newly opened possibilities. This part outlines the constitutive issues that constitute the present frontier and hypothesizes future vectors of research that can put an end to these dialectical tensions.

7.1 Constitutive Antinomies of the CSA Paradigm

CSA has been faced with a number of aporias that prevent its praxis. The main antinomy is that regarding Complexity vs. Accessibility: the design and verification of manual optimized CSA structures is a deep intellectual and methodological problem. The redundancy of the representation itself introduces an epistemic gap, and formal verification and simulation is more challenging since it no longer can be directly compared to canonical binary models. This is enhanced by the Antinomy of Toolflow Integration, in which modern Electronic Design Automation (EDA) flows, based ontologically on traditional binary arithmetic do not phenomenologically deduce the CSA abstraction of high-level behavioral descriptions [13]. The absence of structural RTL in this break in the design abstraction causes a back-sliding of the design process in terms of productivity and limits exploration. Additional material limitations are due to the Antinomy of Resource Materiality because the high LUT overhead of soft CSA logic is a great strain on space-constrained devices, posing a competitive threat the FPGA fabric. This is made worse by the Antinomy of Hardened Logic, under which FPGA DSP slices, being pure implementations of binary arithmetic, have no natural ontology of carry-save formats, and must make an unnatural decision between hardened performance and the energetic benefits of CSA. Lastly, the paradigm runs into the Antinomy of Technological Scaling: as process nodes get better and as static power dominance ensues, the overhead of the area in CSA is directly proportional to the increased leakage footprint, and the dynamic power delivery of CSA would be offset in this case, and the optimization telos would switch to minimizing total energy.

7.2 Prospective Syntheses and Future Research Vectors

In order to overcome these antinomies, new syntheses on both multiple layers of the design stack should be sought out in future research. One of the main frontiers is creating Arithmetic-Aware High-Level Synthesis, called a compiler with

arithmetic phronesis can semantically analyse computational kernels to synthesize the optimal, pipelined CSA dataflows automatically and hence democratize the technique. This is in line with a second vector: the Synthesis with Approximate Computing. The formalism of CSA has an innate tendency to approximate; a more radical approximation by truncation or stochastic rounding of CSA trees may become a potent paradigm in exchange of a marginal loss in accuracy with a radical shift in power and size, especially in settings where robustness to error is an important consideration, such as machine learning. Architectural Synthesis of Next-Generation FPGAs is a simple solution. A material synthesis, such as the addition of hardened CSA reduction blocks or natively-configurable arithmetic units to the FPGA fabric, would give a power efficient synthesis, combining the power efficiency of CSA with the area-performance profile of hardened logic. On the circuit level, this research needs to utilize the Temporal Slack to Energetic Synthesis; the low delay of the critical path in CSA provides excess timing slack that can be used to achieve aggressive voltage scaling or near-threshold operation to attain quadratic dynamic power reductions. Finally, to reach the full potential of CSA, Holistic, Cross-Layer Synthesis Frameworks, which are hermeneutic in nature and co-optimize the algorithm, architecture, and physical implementation, would be needed, and possibly with machine learning predicting the best topologies based on the constraints in the high-level description [41]. To make carry-save arithmetic a mainstream concept, rather than a disputed niche technology, syntheses are needed to make it a principle of efficient, automated, ultra-low-power reconfigurable platforms.

8. Conclusion

This is a review journal paper of the FPGA-based power-efficient Arithmetic Logic Unit that is written in Carry-Save Arithmetic. With the basic aspects of ALUs and CSA, we have defined the theoretical basis on why CSA with its carry-free addition and redundant number representation is a powerful method to overcome the power-performance bottleneck of arithmetic Datapath's. The FPGA architecture analysis has explained the platform-related limitations and opportunities, especially the effects of LUT-implemented implementation, routing power, and the use of hardened DSP blocks.

An elaborate taxonomy, and analysis of existing architectures grouped implementations into Full-CSA, Hybrid, and Co-processor models, each having varying trade-offs in terms of complexity of integration, area and efficiency of the implementation. The synthesis methodology article has emphasized the importance of the fact that the benefits of CSA can only be achieved when the implementation flow is carefully designed to be power-conscious, through the RTL design, the placement and routing processes. This was confirmed by the following comparative study which showed that properly implemented CSA-enhanced ALUs always provide 20-40% better power-delay product and energy per operation over their traditional binary equivalents, especially in iterative computations and medium-to-wide operands.

Nevertheless, there are still major issues, such as the complexity of design, support of the EDA tools, and overhead of the area. Automation, innovative architecture, and cross-layer co-design are the solution to the future of CSA in FPGA-based computing. The future directions appear to be in work on approximate computing, developing of CSA-conscious HLS, and incorporation of CSA-optimized hardened blocks into future FPGA fabrics.

Finally, Carry-Save Arithmetic is a developed but little used paradigm of high performance computing. The CSA integration into the ALU data path is an established, highly effective way of achieving significant improvements in energy efficiency, especially to designers of next-generation embedded systems, DSP accelerators, and low-power processing units on FPGAs. The principles and architectures discussed in this paper will be more important as the demand of performance-per-watt is further increasing across the entire field of computing.

References

1. M. Gokhale and P. S. Graham, *FPGA-Based Reconfigurable Computing: Architectures, Tools, and Applications*. Springer, 2005.
2. C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2015, pp. 161–170.
3. J. Cong and B. Xiao, "FPGA vs. custom VLSI: A case study on the cost-effectiveness of reconfigurable computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 292–299, Feb. 2010.
4. W. Luk, S. Guo, N. Shirazi, and N. Zhuang, "A framework for developing power-efficient FPGA-based systems," *IEEE Des. Test Comput.*, vol. 27, no. 3, pp. 68–78, May/Jun. 2010.
5. K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel FPGA-based microprocessor," *J. Syst. Archit.*, vol. 54, no. 10, pp. 913–925, Oct. 2008.
6. S. Hauck and A. DeHon, Eds., *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2008.
7. V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Springer, 2012.
8. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed. Oxford Univ. Press, 2010.
9. M. D. Ercegovic and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2004.
10. S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed. McGraw-Hill, 2013.
11. R. Zimmermann, "ALU design in modern processors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 7, pp. 375–379, Jul. 2005.
12. V. Betz, J. Rose, and A. Marquardt, "Architecture and CAD for deep-submicron FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 2, pp. 103–130, Apr. 1999.
13. I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, 2008.
14. J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "FPGA and parallel computing for high performance," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2006, pp. 814–817.
15. D. Lewis et al., "The Stratix II logic and routing architecture," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2005, pp. 14–20.

16. G. Lemieux and D. Lewis, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, 2001, pp. 59–68.
17. S. J. E. Wilton, "Low-power FPGA design using pre-computed software," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12, no. 4, pp. 433–438, Apr. 2004.
18. A. DeHon, "The density advantage of configurable computing," Computer, vol. 33, no. 4, pp. 41–49, Apr. 2000.
19. F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, 2003, pp. 175–184.
20. S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, Field-Programmable Gate Arrays. Springer, 1992.
21. J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," ACM Trans. Reconfigurable Technol. Syst., vol. 7, no. 2, pp. 1–30, Jun. 2014.
22. R. Zimmermann, "A survey of carry-save addition," IEEE Trans. Comput., vol. 48, no. 11, pp. 1235–1242, Nov. 1999.
23. K. Benkrid, D. Crookes, and A. Benkrid, "FPGA implementation of a low-power ALU using carry-save adders," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 15, no. 2, pp. 197–207, Feb. 2007.
24. W. Luk, N. Shirazi, S. Guo, and P. Y. K. Cheung, "Pipeline morphing and reconfigurable instruction set processing for adaptive embedded systems," IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 5, pp. 831–839, May 2012.
25. R. Brent and H. T. Kung, "A regular layout for parallel adders," IEEE Trans. Comput., vol. C-31, no. 3, pp. 260–264, Mar. 1982.
26. J. Cong, Y. Fan, and G. Han, "Application-specific instruction generation for configurable processor architectures," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 25, no. 9, pp. 1657–1670, Sep. 2006.
27. K. K. Parhi, "Efficient FPGA hardware development: A layered approach," IEEE Signal Process. Mag., vol. 34, no. 3, pp. 118–127, May 2017.
28. R. Zimmermann, "High-speed multipliers using carry-save adders," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 9, no. 2, pp. 249–253, Apr. 2001.
29. G. De Micheli, Synthesis and Optimization of Digital Circuits, 1st ed. McGraw-Hill, 1994.
30. J. Cong, V. Sarkar, G. Reimman, and A. Bui, "Customizable domain-specific computing," IEEE Des. Test Comput., vol. 28, no. 2, pp. 6–15, Mar./Apr. 2011.
31. Xilinx Inc., Vivado Design Suite User Guide: Synthesis (UG901). Xilinx, 2018.
32. D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, High-Level Synthesis: Introduction to Chip and System Design. Kluwer Academic Publishers, 1992.
33. L. Benini and G. De Micheli, "System-level power optimization: Techniques and tools," ACM Trans. Des. Autom. Electron. Syst., vol. 5, no. 2, pp. 115–192, Apr. 2000.
34. D. Koch, C. Beckhoff, and J. Teich, "Advanced synthesis techniques for reconfigurable computing," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 32, no. 7, pp. 1126–1135, Jul. 2013.
35. F. Li, Y. Lin, and L. He, "FPGA power reduction using glitch-free design and its applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 4, pp. 717–728, Apr. 2014.
36. V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in Proc. Int. Conf. Field-Program. Logic Appl., 1997, pp. 213–222.
37. K. Benkrid, A. Benkrid, and S. Belkacemi, "A comparative study of FPGA-based arithmetic units for digital signal processing," J. Signal Process. Syst., vol. 57, no. 2, pp. 213–225, Nov. 2009.
38. R. Zimmermann, "Efficient arithmetic for digital signal processing," in Proc. IEEE Int. Symp. Circuits Syst., 2012, pp. 253–256.
39. W. Luk, S. Guo, N. Shirazi, and N. Zhuang, "Hardware compilation for configurable computing," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 5, pp. 1189–1198, May 2014.
40. D. B. Thomas and W. Luk, "FPGA-optimized DSP algorithms," in FPGAs for Software Programmers, D. Koch, F. Hannig, and D. Ziener, Eds. Springer, 2016, pp. 147–168.
41. K. K. Parhi, "Stochastic computing and deep neural networks," IEEE Trans. Signal Process., vol. 67, no. 18, pp. 4743–4758, Sep. 2019.