Proposing an Architecture-Aware Framework for High-Throughput Arithmetic Mapping on FPGAs

Mahmoud A. Elshenawy and Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
{mahmoud.elshenawy, suhaib.fahmy}@kaust.edu.sa

***Abstract*—Arithmetic-intensive applications such as cryptography, digital signal processing (DSP), and machine learning demand high computational throughput, for which field-programmable gate arrays (FPGAs) are well-suited: their reconfigurable datapaths combine soft logic and hard blocks such as DSPs to offer high throughput and computational density. However, efficiently mapping arithmetic algorithms onto FPGA architectures is difficult: existing high-level synthesis tools do not exploit DSP block features such as cascade chaining, and they do not achieve high throughput. This paper describes ongoing research towards a compilation framework that automatically maps arithmetic algorithmic descriptions into architecture-aware, high-throughput FPGA implementations.**

***Index Terms*—Arithmetic circuits, field programmable gate arrays, high-level synthesis**

I. Introduction

Arithmetic-intensive applications, including cryptographic primitives, digital signal processing (DSP) kernels, and neural network inference, demand high-throughput hardware accelerators [1], [2]. FPGAs are a natural fit: their reconfigurable fabric enables custom datapath design, and their dedicated DSP blocks deliver high-throughput multiply-accumulate operations efficiently. Modern FPGA DSP blocks integrate multipliers, pre-adders, and accumulation logic in a single hard primitive, with cascade interconnect that allows adjacent blocks to be chained without consuming general routing resources [3].

Despite this potential, realizing efficient arithmetic circuits on FPGAs remains difficult: high-level synthesis (HLS) tools bridge the gap between software and hardware [4] but treat arithmetic as a largely opaque operation: multiplications are inferred from operators without consideration of detailed arithmetic primitive features, cascade opportunities are rarely exploited, and adder trees are generated without regard to the compression structures that minimize critical-path depth. DSP block-optimized mapping techniques [5] can improve on generic HLS by considering DSP block configurations, but they rely on template matching against fixed operator patterns and are not designed for complex arithmetic, where efficient mapping requires exploring algebraic transformations rather than matching predetermined structures. Hand-written register-transfer level (RTL) code can achieve better results but demands deep hardware expertise and is time-consuming and error-prone.

The fundamental problem is a *semantic gap*: arithmetic algorithms are expressed as mathematical operations, while efficient FPGA implementation requires explicit reasoning about resource types, wordlengths, cascade topology, and pipeline scheduling. Closing this gap requires a framework that understands both the algorithmic structure of arithmetic and the architectural characteristics of the target FPGA.

This paper presents our ongoing research towards this goal: an automatic compilation framework that takes arithmetic algorithmic descriptions and produces architecture-aware, pipelined Verilog that exploits FPGA primitives.

II. Research Problem

Mapping arithmetic algorithms onto FPGAs efficiently involves three interrelated challenges.

Algebraic complexity. Arithmetic algorithms expose large spaces of equivalent expressions—varying in factorization, operation ordering, and accumulation structure—each with a different hardware cost. Exploring this space manually is infeasible, motivating an automated algebraic optimizer that finds the most efficient implementation for a given target.

Architecture awareness. DSP blocks have specific port-width constraints, fixed cascade shift amounts, and pipeline characteristics that must be exploited explicitly. A multiplication that slightly exceeds a DSP port width may require tiling into multiple blocks; an accumulation chain may benefit from cascade chaining rather than general routing. These decisions interact with each other and with the adder tree that combines partial products.

Verification under transformation. Optimizing arithmetic hardware involves many algebraic and structural transformations. Each must preserve functional correctness, actively verified rather than assumed, especially when transformations interact across pipeline stages.

III. Proposed Framework

Figure 1 illustrates the proposed compilation framework. The entry point is a hardware-agnostic intermediate language (IL) for arithmetic operations with explicit wordlengths, written directly or auto-generated from C.

The IL feeds into an algebraic optimization stage built on the `egg` equality-saturation library [6], which explores equivalent expressions in an e-graph and extracts the variant minimizing area and latency under a hardware cost model. The

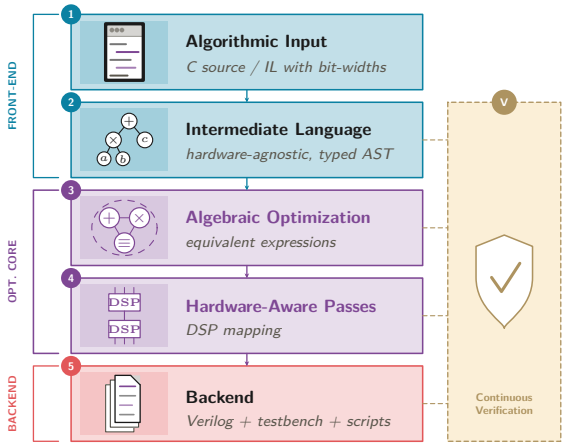


Fig. 1: Overview of the proposed framework.

TABLE I: Preliminary results on the AMD Alveo U280 (UltraScale+)

Benchmark	Impl.	LUTs	FFs	DSPs	Freq. (MHz)	Lat. (cyc)
Mult (1024b)	HLS [7]	165618	–	729	298	24
	This work	128252	221612	810	458	100
Mult (64b)	HLS	258	446	6	282	4
	AMD IP [8]	219	722	16	1038	18
	This work	456	1058	11	1038	15
4×4 GEMM (32b)	HLS	2994	1026	192	140	1
	This work	2198	7072	192	543	48
4×4 GEMM (16b)	HLS	370	261	64	392	4
	This work	0	708	64	954	8

cost model captures DSP, LUT, and FF usage alongside latency and frequency estimates based on the target architecture's characteristics. The optimized expression then passes through a configurable sequence of hardware-aware transformation passes, each a targeted abstract syntax tree (AST) rewrite that can be toggled for design-space exploration. Key passes include DSP-primitive promotion (`Mul` to `PrimMul` when widths fit port constraints), cascade identification (chaining `PrimMul` outputs via PCIN-PCOUT to keep additions in the DSP, thus saving resources and avoiding general routing), and multiplication tiling (partitioning wide operands into optimal DSP-sized tiles that minimize DSP count). Architecture-specific parameters (port widths, cascade shift amount, and tile geometry) are isolated in a single target abstraction, so retargeting a new device requires only updating that abstraction.

The final stage generates synthesizable Verilog with testbenches, timing constraints, and synthesis scripts; correctness is re-verified automatically after every transformation stage.

IV. Preliminary Results

Table I reports look-up table (LUT), flip-flop (FF), and DSP usage on the AMD Alveo U280 (UltraScale+) across multiplication and general matrix multiplication (GEMM) benchmarks. The framework achieves substantial frequency gains over HLS across all benchmarks (54% on 1024-bit multiplication, 3.7× on 64-bit multiplication, 3.9× on 32-bit GEMM, and 2.4× on 16-bit GEMM) via automatic architecture-aware deep pipeline insertion, at the cost of increased flip-flop count and pipeline depth.

LUT usage is reduced on every benchmark except the 64-bit multiplication: 22.6% on 1024-bit multiplication and 26.6% on 32-bit GEMM. Additionally, *zero* LUTs are consumed in the 16-bit GEMM as the entire arithmetic datapath resides within DSP primitives by routing all eligible accumulation through the DSP cascade path (PCIN-PCOUT), bypassing the LUT fabric entirely. On 64-bit multiplication, the framework matches the AMD IP frequency (1038 MHz) using fewer DSPs.

Since designs are fully pipelined, pipeline depth affects only fill latency (amortized over streaming workloads) and the elevated flip-flop counts reflect pipeline registers, not arithmetic logic; latency can instead be reduced at the cost of frequency if desired.

V. Conclusion and Future Work

This paper presented an ongoing research effort towards a compilation framework that closes the semantic gap between arithmetic algorithmic descriptions and architecture-aware FPGA implementations. The framework is largely complete: the front-end, algebraic optimizer, hardware-aware pass pipeline, and Verilog synthesizer are all functional. Preliminary results across multiplication and GEMM benchmarks show consistent LUT reduction, frequency gains of up to 3.9× over HLS, and competitiveness with commercial IP, indicating the approach extends beyond academic HLS baselines.

The primary remaining work is expanding the benchmark suite to confirm that this approach generalizes across a wider range of arithmetic kernels and operand sizes. Planned workloads include stencil computation. The abstraction will be extended by adding AMD Versal support. A further research direction is extending to support mixed-precision datapaths.

References

[1] S. Memik, A. Katsaggelos, and M. Sarrafzadeh, "Analysis and FPGA implementation of image restoration under resource constraints," *IEEE Transactions on Computers*, pp. 390–399, 2003.

[2] T. Ye, S. R. Kuppannagari, R. Kannan, and V. K. Prasanna, "Performance modeling and FPGA acceleration of homomorphic encrypted convolution," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 115–121.

[3] Xilinx, Inc., "Ultrascale architecture DSP slice user guide (UG579)," 2021.

[4] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 473–491, 2011.

[5] B. Ronak and S. A. Fahmy, "Mapping for maximum performance on FPGA DSP blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 573–585, 2016.

[6] Y. Zhang, Y. R. Wang, O. Flatt, D. Cao, P. Zucker, E. Rosenthal, Z. Tatlock, and M. Willsey, "Better together: Unifying datalog and equality saturation," 2023.

[7] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "IMpress: Large integer multiplication expression rewriting for FPGA HLS," in *Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10.

[8] Xilinx, Inc., "Multiplier v12.0 LogiCORE IP product guide (PG108)," 2015.