

Efficient Arithmetic on FPGA

Danila Gorodckyy, Leonel Sousa

INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa
Lisbon, Portugal

danila.gorodckyy@gmail.com, leonel.sousa@tecnico.ulisboa.pt

Abstract—This paper presents an efficient methodology for FPGA arithmetic design based on Boolean function optimization. Focusing on constant multiplication, modular multiplication and reduction, and division by constants, the proposed approach achieves up to 20× LUT reduction and up to 50% delay improvement compared to Vivado-generated designs. Experimental results also demonstrate competitive area–delay trade-offs relative to FloPoCo, highlighting the effectiveness of the method for high-performance FPGA arithmetic implementations.

Index Terms—FPGA, computer arithmetic, Boolean functions.

I. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) combine hardware efficiency and software flexibility, with Look-Up Tables (LUTs) being fundamental for implementing combinational logic. No universal optimal approach exists for efficiently mapping arbitrary arithmetic operations on FPGAs due to the inherent diversity of operation characteristics [1], [2]. Modular reduction is fundamental to Residue Number System (RNS) implementations [1], [2], while division by integer constants occurs frequently, for example, in cryptography [2], [3], networks [4]. FloPoCo [5], [6] is a state-of-the-art arithmetic core generator utilizing various algorithms for designing arithmetic functions. Alternative approaches employ Boolean minimization tools like ABC [7], though these do not account for subtle FPGA architectural features such as dual-output LUT modes. The proposed approach formulates arithmetic operations as optimized Boolean functions specifically tailored to LUT-based FPGA architectures. This results in improved resource utilization and lower latency compared to AMD Vivado-generated designs, while maintaining competitive performance against FloPoCo.

II. SKETCH OF THE APPROACH

The proposed methodology decomposes arithmetic operations (AOs) into superpositions of additions and small bit-width multiplications:

$$\text{AO} = \sum_{i=1}^w A_i \cdot B_i \cdot C_i, \quad (1)$$

where A_i, B_i are m -bit sub-vectors from input operands and C_i represents pre-calculated constants. Each term is

Work partially supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 (DOI: <https://doi.org/10.54499/UID/50021/2025>), UID/PRR/50021/2025 (DOI: <https://doi.org/10.54499/UID/PRR/50021/2025>), and by the European Defence Fund (EDF) grant agreement No. 101168112 (SECURED).

represented as a system of Boolean functions mapped onto LUTs, exploiting the dual-output capability of 6-input LUTs by targeting 5-variable subfunctions where possible. The methodology is carried out in the following three stages.

- 1) *Decomposition and LUT Mapping*: AOs are decomposed per (1), representing each multiplication term as a network of k -input Boolean function systems ($k \leq 2m$ for m -bit sub-vectors). Each output bit is expressed as a Boolean function decomposed into interconnected subfunctions. To fully leverage the dual-output capability of modern 6-input LUTs, subfunctions target five variables where possible—allowing two functions sharing inputs to occupy a single LUT. While tools like ABC perform technology mapping, they do not utilize this FPGA-specific feature, requiring architecture-aware heuristics.
- 2) *Parallel Addition Network*: LUT outputs feed a balanced parallel adder structure that computes higher and lower-order bits simultaneously, significantly reducing critical path compared to sequential tree-structured approaches [8].
- 3) *Result Integration*: intermediate results are combined through concatenation. For modular operations, comparison/subtraction with the modulus P is applied to produce the final corrected result.

III. EXPERIMENTAL RESULTS

Experimental results were obtained on Xilinx Kintex-7 (xc7k70tffb484-3) with Vivado 2022.2¹, using only combinational logic (LUTs) (excluding DSPs and BRAMs) and include the complete place and route process.

Comparisons include Vivado synthesis and FloPoCo arithmetic generator. For FloPoCo, we used: default mode for constant multiplication; *FPC_0* (default architecture, *arch=0*) and *FPC_3* (arithmetic minimization, *arch=3*) for reduction and division, both configured for 6-input LUTs (*alpha=6*). Our approach targets both 5-input (*lut_5*) and 6-input (*lut_6*) mappings.

Constant Multiplication ($A \cdot \text{constant}$): evaluated with constants of 29, 46, 101, 157, and 183 bits multiplied by 7–10 bit variables. For special-form constants ($2^{29} - 3$, $2^{157} - 7$, $2^{183} - 1$), the approach achieves up to 11× LUT reduction versus Vivado by exploiting repetitive truth table patterns. Standard constants show 30–35% LUT reduction with comparable delay. Compared to FloPoCo (default mode), 2–3×

¹Verilog-files prepared for these experiments can be found in https://github.com/ZeboZebo702/DATE_2026.

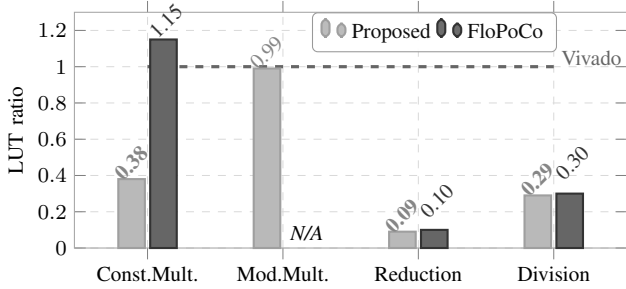


Fig. 1: LUT utilization comparison (normalized to Vivado = 1.0, geometric mean). Lower is better.

better LUT efficiency with similar or less delay across all test cases.

Modular Multiplication ($(A \cdot B) \pmod{P}$): Evaluated for moduli $P \in \{241, 491, 997, 2011, 4051\}$ (8–12 bit range). Achieves 15–25% delay improvement versus Vivado; for smaller moduli ($P = 241, 491$), also 20–25% LUT reduction is achieved. FloPoCo does not support modular multiplication (N/A on the plots), making Vivado the only baseline.

Modular Reduction ($A \pmod{P}$): Tested with 168-bit and 270-bit inputs across five moduli. Versus Vivado: up to 20× LUT reduction (e.g., 509 vs 10024 LUTs for 270-bit A and $P = 241$) and 50% delay improvement. FloPoCo comparison reveals clear trade-offs: FPC_0 requires 2–4× more LUTs and 10–20× worse delay than our approach; FPC_3 achieves comparable LUT count (within 10–15%). Our method provides the best area-delay product.

Division by Constant ($A/d = \{Q, R\}$, where Q is the quotient and R is the residue): Evaluated for 16, 32, 48, and 64-bit dividends with divisors $d \in \{5, 11, 13, 23, 47, 113, 241\}$. For 32-, 48-, and 64-bit operands: 3–6× LUT reduction and 25–40% delay improvement versus Vivado. FloPoCo comparison shows opposite trade-offs: FPC_0 is substantially worse in both metrics (2–10× more LUTs, 20–50% worse delay); FPC_3 achieves similar or slightly better LUT count in some cases, but our approach consistently outperforms it in delay by 5–15% across all configurations. For 16-bit operands, Vivado maintains slight LUT advantage due to decomposition overhead, though our approach still achieves 20% less delay.

Aggregated Results (Figs. 1–2): Results are geometric mean of ratios normalized to Vivado (=1.0). Note that normalized values (e.g., 0.09 for Reduction LUT) represent average improvement; individual cases reach up to 20×. The methodology demonstrates consistent advantages: versus Vivado, superior in both metrics for most operations; versus FloPoCo, our approach consistently outperforms FPC_0 in all cases, while FPC_3 shows operation-dependent trade-offs, i.e. FPC_3 achieves better delay for the modular reduction but worse delay for division, with comparable area throughout.

IV. GLOBAL ANALYSIS AND CONCLUSION

The proposed three stage methodology for efficiently implementing AOs on FPGAs demonstrates consistent per-

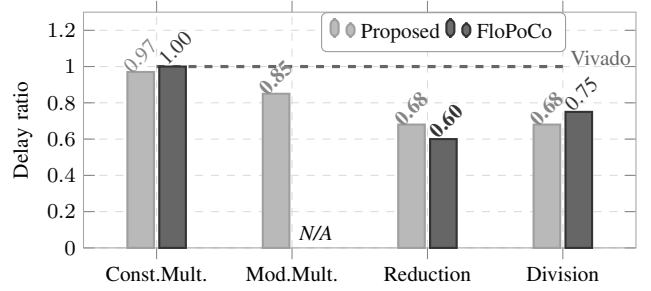


Fig. 2: Critical path delay comparison (normalized to Vivado = 1.0, geometric mean). Lower is better.

formance improvements across diverse AOs. Experimental results show that our methodology achieves competitive or superior performance compared to both Vivado synthesis and FloPoCo framework across constant multiplication, modular multiplication, modular reduction, and division by constant. The approach consistently provides either significant resource savings (up to 20× LUT reduction) or substantial timing improvements (up to 50% critical path reduction), often achieving advantages in both metrics simultaneously.

Comparison with FloPoCo reveals operation-dependent trade-offs. For modular reduction, FloPoCo’s optimized FPC_3 variant achieves slightly better delay but comparable or worse LUT utilization. For division by constant, the situation reverses: FPC_3 shows comparable LUT count in some cases but consistently worse timing across all configurations. The default FPC_0 variant performs substantially worse in both metrics for all operations. These results position our approach as providing the most balanced area-delay trade-off.

Particularly noteworthy is the method’s ability to exploit structural properties of special-form constants like Mersenne numbers, achieving significant efficiency gains that conventional tools cannot match. The strength of the methodology lies in its adaptability to the specific characteristics of different AOs, rather than relying on one-size-fits-all optimization strategies. This flexibility makes it particularly valuable for applications involving arbitrary modulus operations, such as RNS, for which traditional approaches often fail to deliver efficient solutions.

REFERENCES

- [1] P.V.A. Mohan, "Residue Number System. Theory and applications", Springer International Publishing, 2016.
- [2] A.R. Omondi, "Cryptography Arithmetic. Algorithms and Hardware Architectures", Springer Nature Switzerland, 2020.
- [3] L. Sousa, S. Antao, P. Martins, "Combining Residue Arithmetic to Design Efficient Cryptographic Circuits and Systems," in IEEE Circuits and Systems Magazine, vol. 16, no. 4, 2016, pp. 6-32.
- [4] J.L. Hennessy, D.A. Patterson, "Computer architecture. A quantitative approach", 5th ed., Morgan Kaufmann, San Francisco, California, 2012.
- [5] <https://flopoco.org/>
- [6] F. de Dinechin, B. Pasca, "Designing custom arithmetic data paths with FloPoCo", IEEE Design & Test of Computers, 28(4), 18–27, July 2011.
- [7] <https://people.eecs.berkeley.edu/~alanmi/abc/>
- [8] D. Gorodecky, L. Sousa, "Scalable architecture of constant division on FPGA", Proceedings of the 30th IEEE Symposium on Computer Arithmetic, Sep. 4-6, 2023, Portland, Oregon, USA.