Comparing High-Level Synthesis and RTL for Simple FPGA Kernels – Revisited

Vinicius R. Fertrin
Christopher D. Gill
Roger D. Chamberlain

Vinicius R. Fertrin, Christopher D. Gill, and Roger D. Chamberlain, "Comparing High-Level Synthesis and RTL for Simple FPGA Kernels – Revisited," in *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshops* (IPDPSW), May 2026, p. 391.
DOI: 10.1109/IPDPSW71298.2026.00057

Presented at 33rd Reconfigurable Architectures Workshop, New Orleans, LA, USA.

Dept. of Computer Science and Engineering
Washington University in St. Louis

Comparing High-Level Synthesis and RTL for Simple FPGA Kernels – Revisited

Vinicius R. Fertrin
Washington Univ. in St. Louis
r.vinicius@wustl.edu

Christopher D. Gill
Washington Univ. in St. Louis
cdgill@wustl.edu

Roger D. Chamberlain
Washington Univ. in St. Louis
roger@wustl.edu

***Abstract*—As HLS tools improve, questions remain regarding how closely their output can match the performance and resource utilization of carefully optimized RTL designs. We present a recent comparison between optimized SystemVerilog RTL implementations and C++ HLS implementations of simple kernels generated using AMD Vivado and Vitis HLS, targeting an Artix-7 FPGA. Three compute kernels are evaluated, and the results are compared using latency, initiation interval, maximum achievable clock frequency, and resource utilization. The results show that currently, straightforward HLS implementations consistently favor area efficiency at the expense of latency and clock frequency. Overall, this study highlights that while HLS can achieve performance comparable to RTL for certain simple kernels, explicit RTL design still provides better results due to its superior control over the architecture.**

I. Introduction

High-Level Synthesis (HLS) tools have gained increasing attention since becoming commercially available. HLS tools enable designers to describe hardware functionality at a higher level of abstraction, typically using languages such as C++, from which RTL implementations are automatically generated. In principle, HLS aims to improve design productivity, reduce development time, and simplify hardware development. However, there are still questions regarding how closely HLS-generated hardware can match the performance and resource efficiency of carefully optimized RTL designs [1].

The capabilities of HLS tool flows are in constant flux, with new releases of the tools happening quite frequently. Most comparison studies are now several years old, with limited recent comparisons published. We believe it is worthwhile to revisit this question using a recently released tool flow. We present a direct comparison between optimized SystemVerilog RTL and C++ based HLS implementations. The selected kernels include population count (popcount), leading-zero count (LZC), and a finite impulse response (FIR) filter.

Each kernel is implemented twice: first using SystemVerilog and second using C++ on Vitis HLS. The resulting designs are synthesized targeting the same FPGA, an Artix 7-series xc7a35tcsg324. The comparison focuses on post-synthesis metrics: latency in cycles, initiation interval (II), estimated maximum clock frequency (F_{max}), and resource utilization in terms of lookup tables (LUTs), flip-flops (FFs), and digital signal processing (DSP) blocks used.

Support provided by a gift from BECS Technology, Inc.

II. Results

Table I shows the results for each of the three kernels. The HLS kernels include an initial design, a fully unrolled design, and partially unrolled designs.

TABLE I
Performance and Resource Utilization

Popcount

Implementation	Latency (cycles)	II (cycles)	LUT	FF	DSP	F_{max} (MHz)
RTL	8	1	248	519	0	440
Initial HLS	130	128	126	19	0	190
Unrolled HLS	1	2	1,416	42	0	151
Factor 2 HLS	66	64	157	22	0	187
Factor 4 HLS	34	32	246	24	0	145
Factor 8 HLS	18	16	426	28	0	143
Factor 16 HLS	11	9	788	47	0	182
Factor 32 HLS	7	5	1,509	61	0	145
Factor 64 HLS	5	3	2,951	99	0	145

LZC

Implementation	Latency (cycles)	II (cycles)	LUT	FF	DSP	F_{max} (MHz)
RTL	8	1	457	766	0	594
Initial HLS	133	134	283	160	0	143
Unrolled HLS	3	4	1,090	147	0	143
Factor 2 HLS	69	70	412	179	0	105
Factor 4 HLS	37	38	570	181	0	104
Factor 8 HLS	21	22	886	183	0	103
Factor 16 HLS	13	14	1,566	185	0	100
Factor 32 HLS	10	10	2,858	187	0	91
Factor 64 HLS	8	8	5,390	189	0	91

FIR Filter

Implementation	Latency (cycles)	II (cycles)	LUT	FF	DSP	F_{max} (MHz)
RTL	1	1	283	134	0	81
Initial HLS	25	26	257	150	1	172
Unrolled HLS	1	2	178	150	0	156
Factor 2 HLS	20	21	311	283	2	172
Factor 4 HLS	5	6	369	372	0	144

Across all kernels, the first HLS implementation consistently seems to favor area efficiency at the expense of latency and II. Loop unrolling effectively reduces latency, but often at the cost of a significant increase in resource usage, and in some cases worsens the II and F_{max}.

References

[1] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, 2019.

2995-066X/26/$31.00 ©2026 IEEE
DOI 10.1109/IPDPSW71298.2026.00057