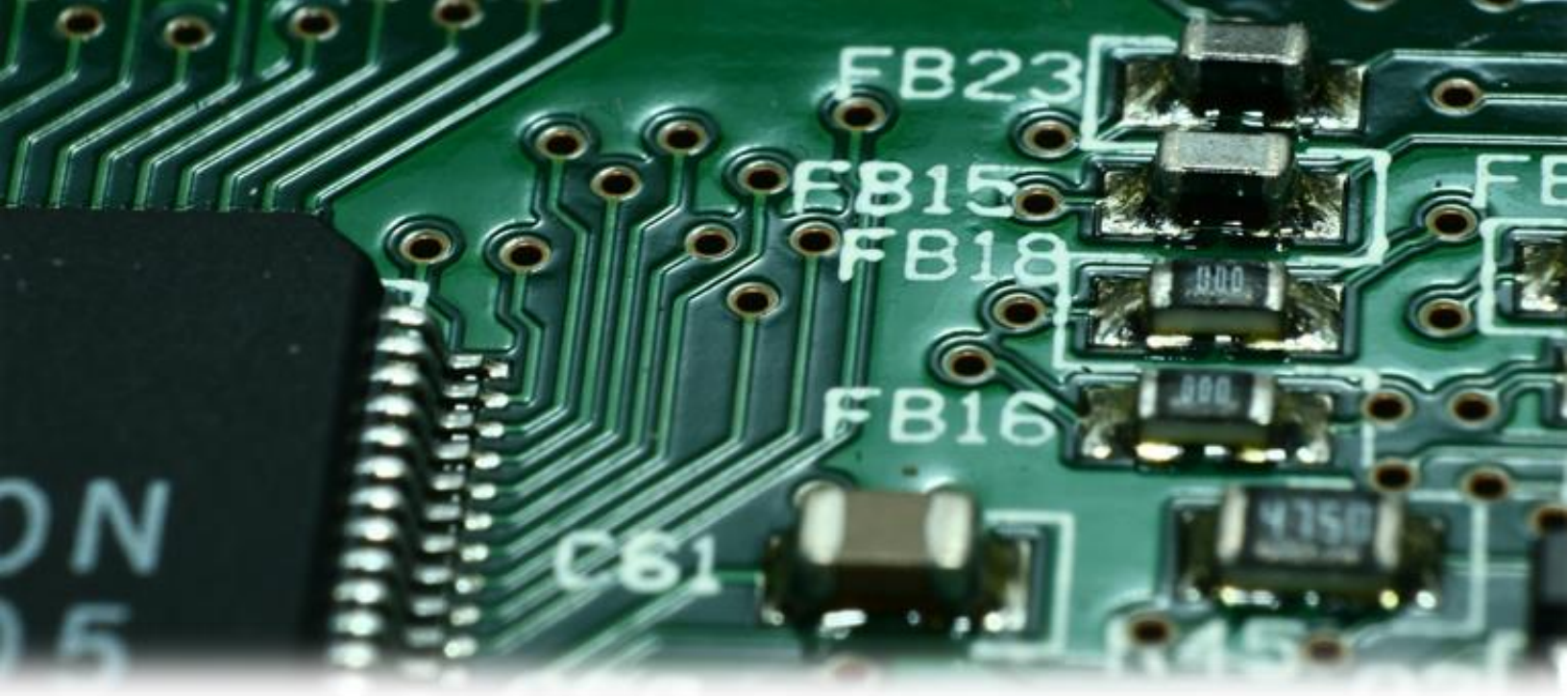

Benefits of High-Level Synthesis for FPGA Design

White Paper

July 2020

INTRODUCTION

As FPGA designs continue to get larger and more complex, engineers need to improve their productivity to meet tight design schedules. This white paper will explain how engineers can speed up their FPGA development by using the LegUp high-level synthesis tool to generate their hardware blocks from C++ software.

In the high-level synthesis design flow, the engineer implements their design in C++ software and verifies the functionality with software tests. Next, they specify a top-level C++ function, which LegUp will compile into an equivalent Verilog hardware module. LegUp can run co-simulation to verify the hardware module behaviour matches the software. LegUp uses Libero SoC to generate the post-layout timing and resource reports for the Verilog module. Finally, LegUp generates a SmartDesign IP component that the engineer can instantiate into their SmartDesign system in Libero SoC. Figure 1 shows the high-level synthesis FPGA design flow for targeting a Microchip PolarFire FPGA.

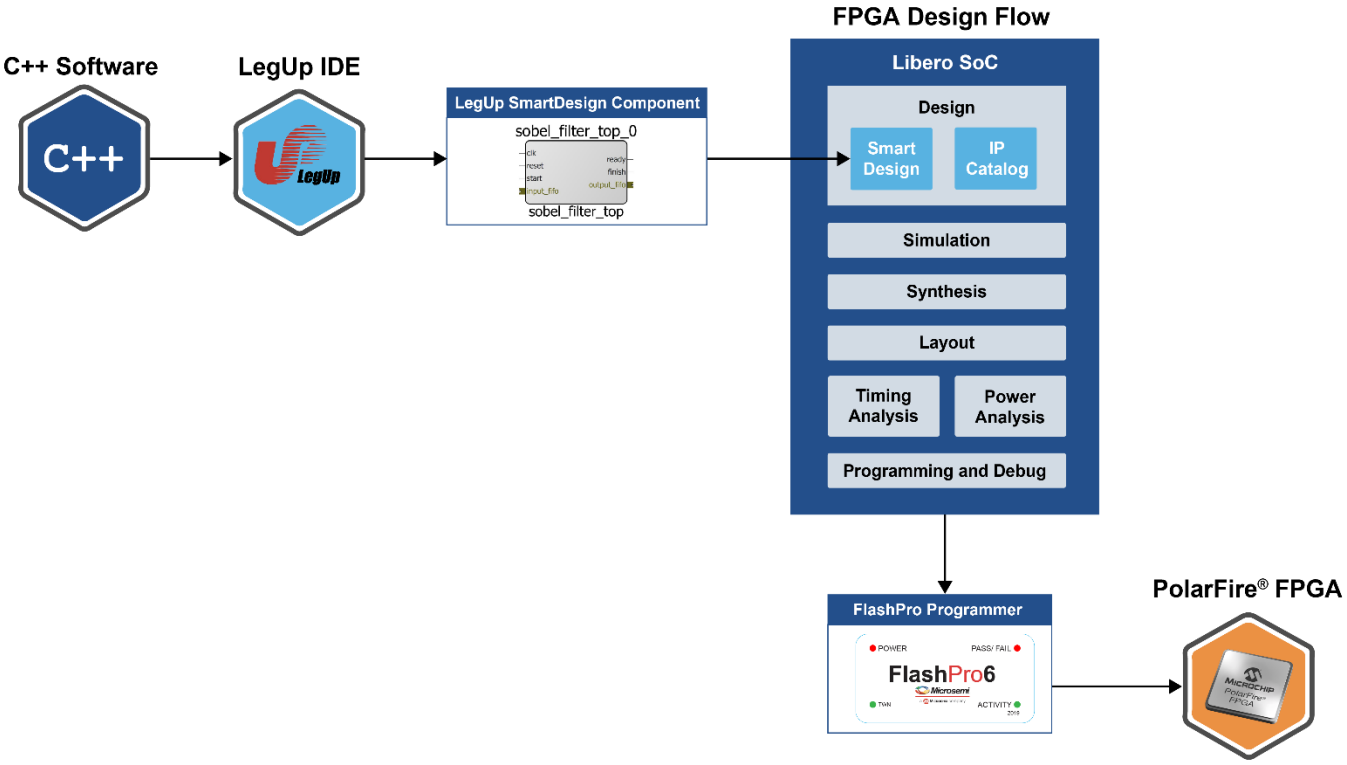

Figure 1: High-level Synthesis FPGA Design Flow Targeting a PolarFire FPGA

FPGA DESIGN: RTL VS HLS

For readers unfamiliar to high-level synthesis, we compare the traditional FPGA design flow using a register transfer level (RTL) language like Verilog/VHDL to the newer high-level synthesis design approach using C++ software.

Traditional RTL design flow for FPGAs

1. RTL design: Hardware engineer decides the hardware microarchitecture and manually writes the hardware block in RTL language like Verilog/VHDL.
2. Optimization & Timing closure: Hardware engineer iterates on design to meet the clock period and area constraints by adding pipeline registers and restructuring the RTL.
3. Verification: Hardware engineer writes a testbench to verify the RTL block in simulation. Acceptance testing compares functionality to the original software model.

As shown in Figure 2, using high-level synthesis to design FPGA hardware blocks with C++ software code can save engineers 2-5X design time compared to traditional RTL design.

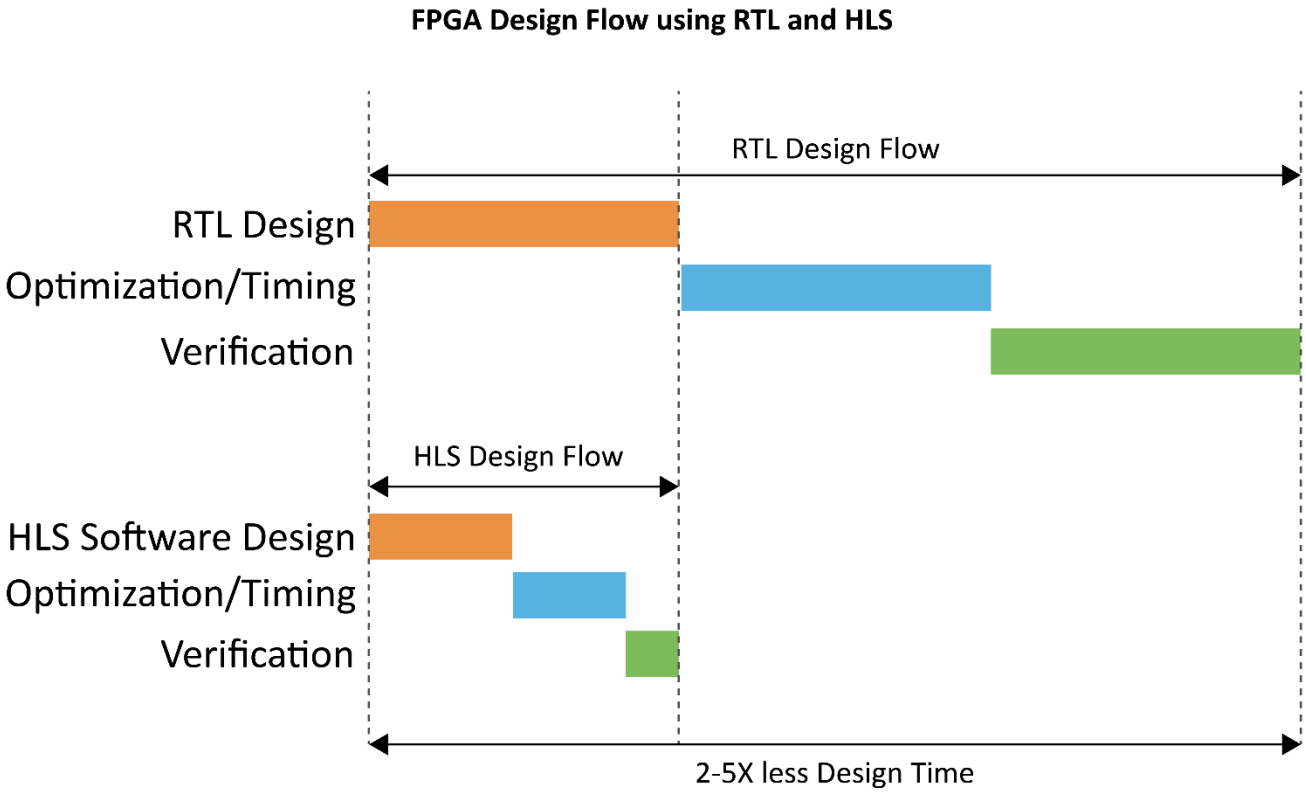

Figure 2: FPGA design flow using RTL vs. HLS

High-Level Synthesis design flow for FPGAs

1. HLS Software Design: engineer writes a software model of the design in C++ software. Software is tested to verify desired functionality. The engineer generates a hardware block from the C++ software using high-level synthesis. Engineer rewrites the C++ in HLS style as necessary, for example using FIFO types for dataflow computation, to get the desired throughput.
2. Optimization & timing closure: engineer adds HLS constraints for the desired clock period. Engineer tunes the HLS settings, for example to reduce area by sharing hardware operators. Engineer runs the Libero SoC design suite to verify area/timing.
3. Verification: The generated circuit will be correct by construction. Engineer can use co-simulation to verify the generated RTL block in Modelsim simulation. Co-simulation will automatically create a testbench for the tests from the original software model.

HLS PRODUCTIVITY GAIN

Engineers developing FPGA hardware in C++ using high-level synthesis design tools reduce their design effort significantly compared to engineers writing hardware in RTL. Writing software code is much easier for engineers than writing hardware in RTL because software code is more concise, with 5-10X less lines of C++ required than RTL, as shown in Figure 3. Software code is also much easier to read and understand for future improvements or maintenance compared to RTL. Software conciseness and readability mean less bugs in your FPGA design. As shown in Figure 2, engineers will see a 2-5X reduction in FPGA design time by using HLS. Better designer productivity leads to faster time-to-market.

C++ software code describes an algorithm, which is at a much higher level of abstraction than hardware description languages. For example, in C++ code the programmer does not need to specify any hardware timing constructs like clock cycles. In contrast, a hardware engineer writing RTL must specify cycle-by-cycle hardware behavior using finite state machines and other control logic. If any control logic is off by one cycle, then circuit functionality will break. High-level synthesis also supports C++ data types to

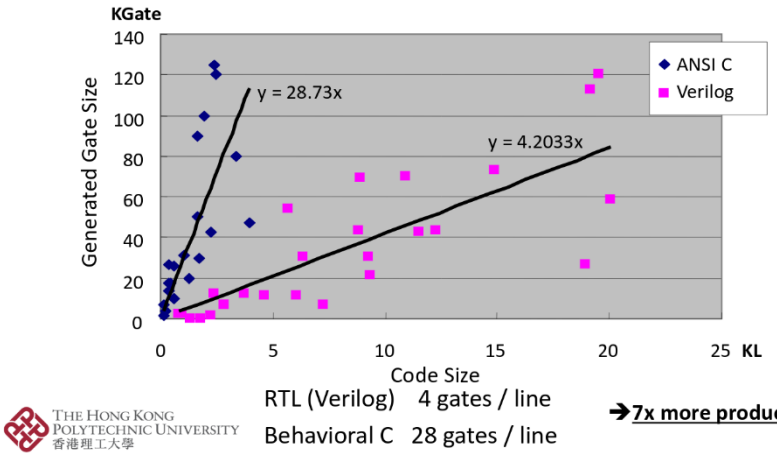

Figure 3: Lines of C code versus lines of RTL code [1]

represent floating or fixed-point computation. These data types are not natively supported in Verilog/VHDL languages therefore the engineer must handle these operations manually.

High-level synthesis also offers much more flexible hardware module parameterization compared to Verilog/VHDL. In a software description of the design, C++ template parameters offer more fine-grained control versus simple RTL parameters. Furthermore, HLS user-provided constraints can fine-tune the generated hardware microarchitecture for different latency, timing, and area design constraints. Meanwhile, for an RTL design the engineer must manually add pipeline registers to the data path to meet different timing constraints. Last-minute pipeline changes are painful and difficult to parameterize in RTL.

PORT EXISTING SOFTWARE TO FPGA

High-level synthesis allows engineers to reuse a pre-existing software implementation of their algorithm and port this software to an FPGA. In many cases, this existing software implementation has been verified and tested extensively. For an engineer to manually re-implement the software design in Verilog/VHDL could introduce subtle errors that can be tedious to catch and time consuming to verify. High-level synthesis greatly simplifies this process and avoids manual reimplementation saving design time and engineering resources.

For example, a real-time control application may include software or firmware running on a microcontroller that can no longer meet the required response time. The solution is to migrate the software to an FPGA which offers deterministic latency. High-level synthesis allows the engineer to automatically generate Verilog/VHDL from the existing C++ software. See our customer case-study white paper: “Migrating Motor Controller C++ Software from a Microcontroller to a PolarFire FPGA with LegUp High-Level Synthesis”.

FASTER VERIFICATION & LESS BUGS

For engineers, verifying the functional correctness of a hardware design written in RTL is usually the most time-consuming part of designing for an FPGA. The engineer must manually write testbenches in RTL to simulate their design under various input/output test stimulus. In contrast, an engineer designing hardware in C++ with high-level synthesis can significantly reduce their time spent on verification. When using high-level synthesis, the engineer can write tests in software to verify their design, since the circuit generated from HLS will be correct by construction. Writing tests in C++ software is simpler than writing hardware testbenches in RTL, which usually leads to writing more comprehensive tests, increasing test coverage.

LegUp HLS supports automatic co-simulation of the generated hardware with Modelsim, by re-using the C++ software tests. During co-simulation, the LegUp HLS tool runs the software program with instrumentation added around the top-level C++ function to collect golden inputs/outputs for each function argument. Then LegUp automatically generates a hardware testbench that reads the golden inputs/outputs from test vector files. The testbench verifies that the generated hardware module behaviour matches the software model. Engineers can also write their own custom testbench with a more restricted set of tests to be run on the HLS-generated RTL as a sanity check. Much less RTL simulation is necessary with HLS design.

RTL simulation times can become long and impractical for larger FPGA designs as the number of tests increase. High-level synthesis allows for software-based testing and verification which has 100-1000X faster runtime than RTL simulation. Faster software-based verification runtimes mean that engineers using HLS can still verify hardware after last-minute design changes if requirements change late in the design process.

DESIGN SPACE EXPLORATION

Design space exploration is the process of making hardware microarchitecture design trade-offs within a set of constraints on FPGA clock frequency, throughput, latency, and area. An RTL engineer will usually decide on a hardware microarchitecture early in the design process. After the RTL is written, an engineer will typically avoid making microarchitecture changes, such as adding pipeline stages, because RTL redesign is a time-consuming process.

In contrast, high-level synthesis simplifies design space exploration and allows continuous refinement of the hardware microarchitecture throughout the design process. For example, the HLS tool automatically inserts pipeline registers based on the HLS target clock period constraint. The engineer can easily modify the HLS target clock period to achieve different performance/area targets without manual redesign. HLS typically runs in a few

minutes, so the designer can quickly get feedback on resource and throughput estimates without waiting for a time-consuming FPGA synthesis run.

LegUp HLS offers a rich set of user-constraints and pragmas for the engineer to specify their desired hardware microarchitecture based on the software description. For example, loops can be pipelined to improve performance, or inner loops can be unrolled. Functions with FIFO dataflow streaming inputs/outputs can be pipelined. C++ arrays can be partitioned into registers or RAMs to achieve better memory bandwidth and performance. High-level synthesis also supports sharing larger hardware operations such as floating-point cores. HLS constraints give the designer the ability to easily perform more design space exploration, which can lead to better trade-offs between performance and area for their FPGA designs.

FPGA DEVICE PORTABILITY

A verified hardware IP block can be reused for many years. Therefore, future proofing is important if the hardware block could later target another FPGA device family. RTL designs have a specific microarchitecture targeting one FPGA family. Therefore, RTL redesign will be required if the timing constraints are not met when porting. RTL designs may also use primitive blocks specific to the FPGA family, which will need to be rewritten.

High-level synthesis automatically adds pipelining stages to your hardware depending on the target FPGA device and clock period constraints. HLS settings can easily be changed to re-generate a hardware block targeting a new FPGA family while still meeting the timing constraints. Portability is simplified because there are no FPGA family-specific C++ design constructs.

CONCLUSION

In conclusion, FPGA design with C++ using high-level synthesis can offer 2-5X better design productivity compared to writing in RTL. The resulting C++ code is 5-10X shorter than an equivalent RTL design and C++ is expressed at a higher level of abstraction. This results in less bugs and easier readability. HLS software-based verification and testing saves significant design effort compared to RTL verification. HLS also enables easier design space exploration and FPGA device portability.

Key Takeaways

- LegUp HLS simplifies FPGA design by allowing you to program the FPGA using C/C++ software
- 2-5X better design productivity
- Higher abstraction level means less code and less bugs
- Faster verification and testing
- Design space exploration
- FPGA device portability