PRODUCT BRIEF

Intel® HLS Compiler

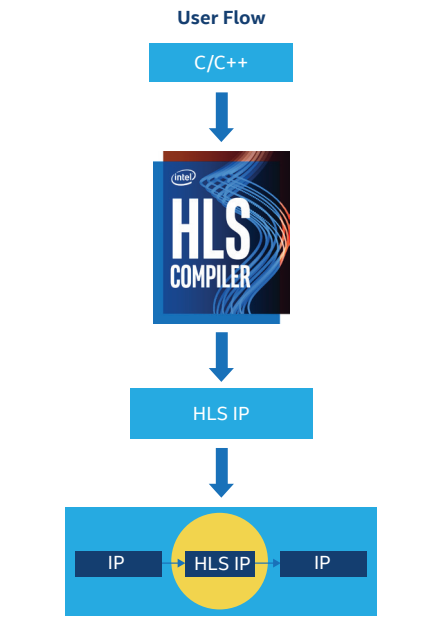

Figure 1. Intel HLS Compiler Tool Flow

Accelerating FPGA Development with C++

Intel® HLS Compiler is a high-level synthesis (HLS) tool that accelerates FPGA development while rivaling hand-coded register transfer level (RTL).

Figure 1 shows the Intel HLS Compiler tool flow: it takes in C++ code as input and generates production-quality RTL that can be instantiated into a larger system as a custom component using the Intel Quartus® Prime Design Software to program the FPGA.

The Intel HLS Compiler is designed with hardware developers, algorithm designers, and intellectual property (IP) library designers in mind. The compiler can also be used by anyone who wants to target Intel FPGA hardware but would rather code in C++ than in a traditional RTL, such as Verilog. It is also ideal for those who have already mastered the backend flow from high-level design to FPGA bitstream.

Faster Verification Time

The functional debug and verification iteration cycle for RTL simulation is long and limits iterations per day, resulting in longer development for FPGAs. Fortunately, verifying the design source in C++ is much faster than simulating in C++, as shown in Figure 2. Since C++ is the input, this tool accelerates verification time over RTL by raising the abstraction level for FPGA hardware designs. It also allows you to debug and optimize on a CPU. The HLS Compiler generates reusable, high-quality code that meets performance and is within 10%-15% of the area of hand-coded RTL.†

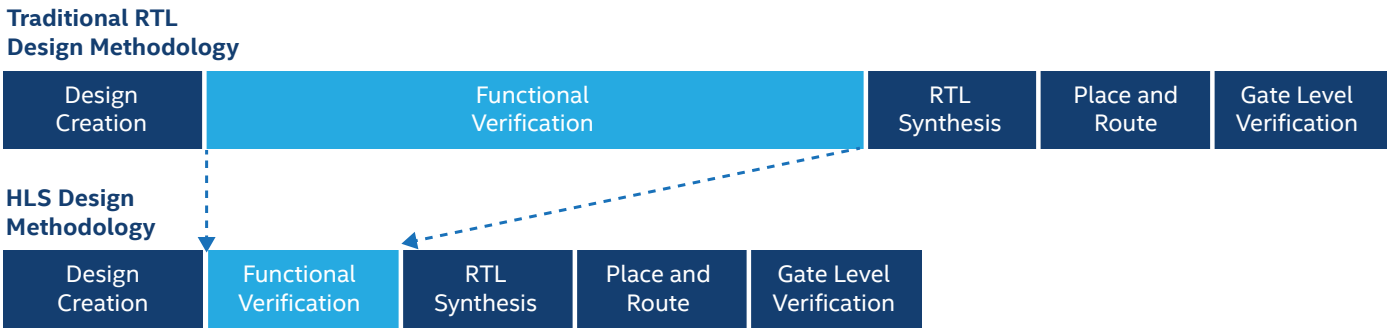

Figure 2. Functional Verification Improvements

Intel HLS Compiler Key Features

The latest release of the Intel HLS Compiler for the Intel Quartus Prime Design Software provides the following capabilities that enable hardware programmers to use C++ for accelerating their FPGA development process:

- **Supports C++ input** – Supports C++ for algorithmic development, which allows you to transfer existing programs that use the GCC compiler to the Intel HLS Compiler for FPGA development. This allows for seamless verification of a user's algorithm in the software world while using a software testbench.
- **Automatic RTL verification to C++** – Supports software testbench verification against the compiler-generated hardware model (RTL) automatically.
- **Design exploration** – Supports attributes and pragmas that enable you to quickly explore hardware architectures including interfaces, parallelism, memories, datapaths, and loops.
- **System of tasks** – Allows for expression of thread-level parallelism within a HLS component. Common use cases include: executing multiple loops in parallel, sharing an expensive compute block, or designing a HLS system hierarchically so starting with the small building blocks, stitching them together, and connecting them with streams.
- **Variable precision types** - Supports a software compiler use model and industry standards, including ac_int and ac_fixed data types
- **Floating-point support** – Provides native support for floating-point and fixed-point variables and operators. Integrated hardened floating-point multiple-add units in Intel FPGAs. Fixed-point support is also useful for its ability to both cover the legacy designs and enable the fastest, smallest designs.
- **Reporting** – Generates interactive analysis reports after converting, which gives you a birds-eye view of the design. Reports include cross-probing of the source code that allows easy micro-architecture optimizations, such as loop-unrolling and variable dependency fixing.

There are several quality of result (QoR) reports that allow for better insight into what hardware the compiler has generated. You can take advantage of new controls, iterate designs, and make progress on QoR.

- **Block viewer report** – Shows clusters and their surrounding logic.
- **Cluster viewer** – Shows fine-grained details within clusters, including instructions and dependences of the instructions for the generated data path in a graphical manner, which enables you to quickly iterate optimizations to your design and see the changes.
- **f_{MAX}/II reports** – Lists key performance metrics or indicators on all basic blocks. It is intended to help expert users identify f_{MAX} bottlenecks in their design, set proper f_{MAX} targets, and estimate execution latency. The f_{MAX}/II report for the Intel HLS Compiler also provides guidance for users who use the loop pragmas.

† Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Gen-1027-1.1