Intel® High Level Synthesis Compiler Pro Edition

User Guide

Updated for Quartus® Prime Design Suite: **24.1**

Online Version
Send Feedback

UG-20037

683456
2024.04.01

Contents

1. Intel® High Level Synthesis Compiler Pro Edition User Guide

The *Intel® HLS Compiler Pro Edition User Guide* provides instructions on synthesizing, verifying, and simulating IP that you design for Intel FPGA products. The Intel High Level Synthesis (HLS) Compiler is sometimes referred to as the i++ compiler, reflecting the name of the compiler command.

Compared to traditional RTL development, the Intel HLS Compiler offers the following advantages:

- Fast and easy verification
- Algorithmic development in C++
- Automatic integration of RTL verification with a C++ testbench
- Powerful microarchitecture optimizations

In this publication, `<quartus_installdir>` refers to the location where you installed Quartus® Prime Design Suite.

The default Quartus Prime Design Suite installation location depends on your operating system:

Windows	`C:\intelFPGA_pro\24.1`
Linux	`/home/<username>/intelFPGA_pro/24.1`

About the Intel HLS Compiler Pro Edition Documentation Library

Documentation for the Intel HLS Compiler Pro Edition is split across a few publications. Use the following table to find the publication that contains the Intel HLS Compiler Pro Edition information that you are looking for:

Table 1. Intel High Level Synthesis Compiler Pro Edition Documentation Library

Title and Description	PRO
Release Notes Provides late-breaking information about the Intel HLS Compiler.	Link
Getting Started Guide Get up and running with the Intel HLS Compiler by learning how to initialize your compiler environment and reviewing the various design examples and tutorials provided with the Intel HLS Compiler.	Link

continued...

Title and Description	PRO
User Guide Provides instructions on synthesizing, verifying, and simulating intellectual property (IP) that you design for Intel FPGA products. Go through the entire development flow of your component from creating your component and testbench up to integrating your component IP into a larger system with the Intel Quartus Prime software.	Link
Best Practices Guide Provides techniques and practices that you can apply to improve the FPGA area utilization and performance of your HLS component. Typically, you apply these best practices after you verify the functional correctness of your component.	Link
Reference Manual Provides reference information about the features supported by the Intel HLS Compiler. Find details on Intel HLS Compiler command options, header files, pragmas, attributes, macros, declarations, arguments, and template libraries.	Link

Discontinuation of the Intel HLS Compiler on page 4

1.1. Discontinuation of the Intel HLS Compiler

Intel is discontinuing the Intel High Level Synthesis (HLS) Compiler software product. For details, refer to *Product Discontinuance Notice PDN2404*.

To keep access to the latest FPGA high-level design features, optimizations, and development utilities, migrate your existing designs to use the Intel oneAPI Base Toolkit. For more information about using the Intel oneAPI Base Toolkit with FPGA devices, refer to *Intel oneAPI DPC++/C++ Compiler Handbook for Intel FPGAs*.

2. Overview of the Intel High Level Synthesis (HLS) Compiler Pro Edition

The Intel High Level Synthesis (HLS) Compiler parses your design and compiles it to an x86-64 object or RTL code optimized for Intel FPGA device families. It also creates an executable testbench. Use the x86-64 object to quickly test and debug the function of your design.

You can use the same testbench to verify your design both when it is compiled to x86-64 instructions or to RTL.

The Intel HLS Compiler Pro Edition is command-line compatible with g++, and supports most of the g++ compiler flags. See the *Intel High Level Synthesis Compiler Reference Manual* for a full list of compiler flags.

The Intel HLS Compiler Pro Edition recognizes the same file name extensions as g++, namely `.c`, `.C`, `.cc`, `.cpp`, `.CPP`, `.c++`, `.cp`, and `.cxx`. The compiler treats all of these file types as C++. The compiler does not explicitly support C, other than as a subset of C++.

Important: The Intel HLS Compiler Pro Edition treats all input files as C++17. The compiler does not support files conforming to newer C++ standards.

When you target the compilation to an FPGA, the Intel HLS Compiler outputs an executable and a project directory. The default executable is `a.out` on Linux and `a.exe` on Windows. The default project directory is `a.prj`, and it contains the following outputs:

- Generated IP
- High-Level Design Reports (`report.html`)
- Verification testbench files
- Quartus project that you can use to accurately estimate area requirements and f_{MAX} for your design.

To specify the name of the compiler output, include the `-o <`*`result`*`>` option in your `i++` command, where `<`*`result`*`>` is the name of the executable. This command creates a project directory called `<`*`result`*`>.prj`.

Running the executable file runs your testbench. When you target the compilation to an x86-64 architecture, the output executable runs your design on the CPU like a regular C++ program. The output executable runs very quickly compared to running a simulation of your component RTL. When you target the compilation to an FPGA architecture, the output executable simulates your component RTL. This simulation can take a long time to run.

2.1. High Level Synthesis Design Flow

The Intel High Level Synthesis (HLS) Compiler helps speed your IP development by letting you compile your IP component C++ code to different targets, depending on where you are in your IP development cycle.

The typical design flow when you use the Intel HLS Compiler Pro Edition consists of the following stages:

1. Creating your component and testbench.

   You can write a complete C++ application that contains both your component code and your testbench code.

   For details, see Creating a High-Level Synthesis Component and Testbench on page 9.

2. Verify the functionality of your component algorithm and testbench.

   Verify the functionality by compiling your design to an x86-64 executable and running the executable. For details, see Verifying the Functionality of Your Design on page 11.

3. Optimize and refine the FPGA performance of your component.

   Optimize the FPGA performance of your component by compiling your design to an FPGA target and reviewing the high-level design report to see where you can optimize your component. This step generates RTL code for your component. For details, see Optimizing and Refining Your Component on page 12.

   After initial optimizations, you can see where to further refine your component by simulating it. For details, see Verifying Your IP with Simulation on page 16.

4. Synthesize your component with Quartus Prime.

   For details, see Synthesize your Component IP with Quartus Prime Pro Edition on page 21.

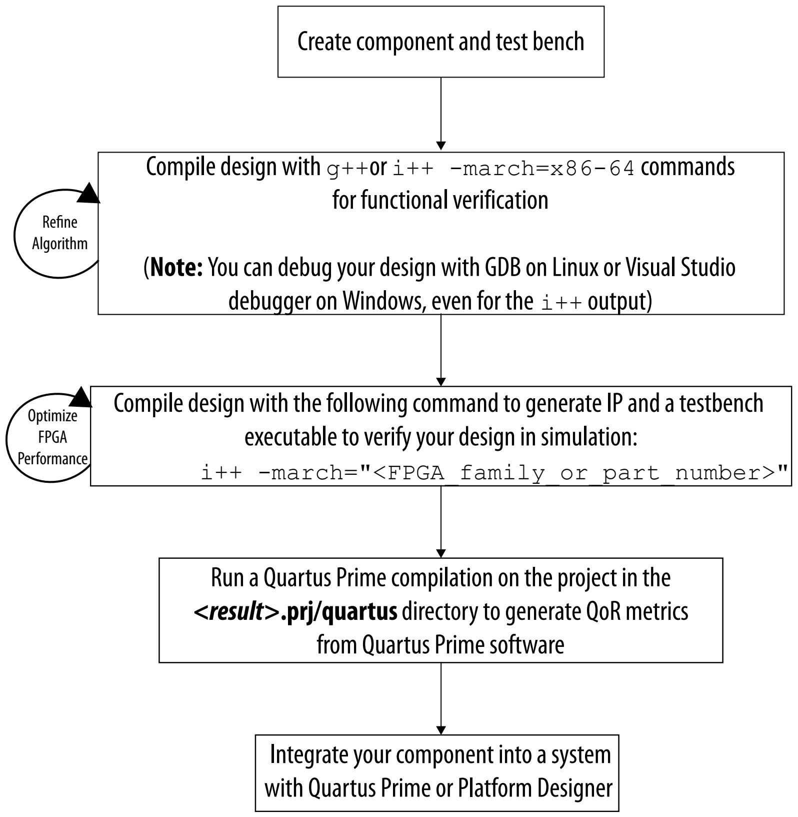
   Synthesizing your component generates accurate quality-of-results (QoR) metrics like FPGA area utilization and f_{MAX}.

5. Integrate your IP into a system with Quartus Prime or Platform Designer (formerly Qsys).

   For details, see Integrating your IP into a System on page 22.

The following flowchart shows a coarse-grained progression through the stages of a typical Intel High Level Synthesis (HLS) Compiler design flow.

Figure 1. Overview of Procedure for Synthesizing IP for Intel FPGA Products


Create component and test bench
Compile design with g++or i++ -march=x86-64 commands for functional verification
(Note: You can debug your design with GDB on Linux or Visual Studio debugger on Windows, even for the i++ output)
Refine Algorithm
Compile design with the following command to generate IP and a testbench executable to verify your design in simulation:
i++ -march="<FPGA family or part number>"
Optimize FPGA Performance
Run a Quartus Prime compilation on the project in the <result>.prj/quartus directory to generate QoR metrics from Quartus Prime software
Integrate your component into a system with Quartus Prime or Platform Designer

For an example of the Intel HLS Compiler design flow, watch the *HLS Walkthrough* series at the Intel FPGA channel on YouTube or complete the `full-design` tutorial found in `<quartus_installdir>/hls/examples/tutorials/usability`.

Related Information

Intel FPGA channel on YouTube

2.2. The Project Directory

The project directory (`<result>.prj`) that the Intel HLS Compiler Pro Edition outputs has four subdirectories.

Table 2. Subdirectories within the .prj Directory

Directory	Description
`components`	Contains a folder for each component, and all HDL and IP files that are needed to use that component in a design.
`verification`	Contains all the files for the verification testbench.
`reports`	Contains the High-Level Design Reports. The High-Level Design Reports are a set of reports and viewers that you open in a web browser. Use the reports to analyze the synthesized hardware implementation of your components and tasks.
`quartus`	Contains an Quartus Prime project that instantiates the components. You can compile this Quartus Prime project to generate more detailed timing and area reports. Do not use the contents of this subdirectory to integrate your component in a design. Use the contents of the `components` directory.

3. Creating a High-Level Synthesis Component and Testbench

The Intel HLS Compiler Pro Edition converts individual functions into RTL code. The components are part of a C++ application that acts as a testbench for your component functions, and you can test your components by calling them from your `main()` function and verifying that the output is correct.

The compiler supports C++17 and can synthesize some C++ constructs, which might be easier for you to use to create cleaner code. For more information about the supported C++ subset and its restrictions, see "Supported C and C++ Subset for Component Synthesis" in *Intel High Level Synthesis Compiler Pro Edition Reference Manual*.

The Intel HLS Compiler Pro Edition synthesizes all the code in the function or functions that you identify as components, and any code that these components call, to an RTL representation.

Identify a function in your C++ application that you want to synthesize into RTL with the `component` function attribute.

Important: Components are synthesized into RTL for all functions with the `component` function attribute and for all components listed in the `--component` *<component_list>* option of the `i++` command. Avoid combining these methods because you might unexpectedly synthesize unwanted components.

If you do not want components synthesized into RTL for a function, ensure that you do not have the `component` function attribute specified in the function and ensure that the function is not specified in the `--component` *<component_list>* option of the `i++` command.

You can see which components were synthesized into RTL in the summary page of the High-Level Design Reports (*<name>*`.prj/reports/report.html`). For more information about the High-Level Design Reports, see The High-Level Design Reports on page 12.

The HLS compiler creates an executable to run on the CPU. The compiler then sends any calls to functions that you declared as components to simulation of the synthesized IP core, and the simulation results are returned.

3.1. Intel HLS Compiler Pro Edition Compiler-Defined Preprocessor Macros

The Intel HLS Compiler Pro Edition has built-in macros that you can use to customize your code to create flow-dependent behaviors.

Table 3. Macro Definition for __INTELFPGA_COMPILER__

Tool Invocation	__INTELFPGA_COMPILER__
`g++` or `cl`	Undefined
`i++ -march=x86-64`	`2410`
`i++ -march="`*`<FPGA_family_or_part_number>`*`"`	`2410`

Table 4. Macro Definition for HLS_SYNTHESIS

Tool Invocation	HLS_SYNTHESIS	
	Testbench Code	HLS Component Code
`g++` or `cl`	Undefined	Undefined
`i++ -march=x86-64`	Undefined	Undefined
`i++ -march="`*`<FPGA_family_or_part_number>`*`"`	Undefined	Defined

4. Verifying the Functionality of Your Design

Verify the functionality of your design by compiling your component and testbench to an x86-64 executable that you can debug with a native C++ debugger. This process is sometimes referred to as *debugging through emulation*.

Compiling your design to an x86-64 executable is faster than generating and simulating RTL. This faster compilation time lets you debug and refine your component algorithms quickly before you move on to see how your component is implemented in hardware.

You can compile your component and testbench to an x86-64 executable for functional verification through any of the following methods:

- Use the `i++ -march=x86-64` command.
- On Linux systems, use the g++ command.
- On Windows systems, use Microsoft Visual Studio.

To verify the functionality of your design from the x86-64 emulation of your testbench and component, you can use typical debugging techniques like the following:

- Running the program to see if generates the expected output.
- Using `printf` statements in your code to output variable values at certain points in your code.
- Stepping through your code with a debugger.

If you want step through your code with a debugger, ensure that you set your compiler command to include debug information and to generate unoptimized binary files. The `i++` command generates debug information by default, and the `-march=x86-64` command option generates unoptimized binary files.

On Linux systems, you can use GDB to debug your component and testbench, even if you used the `i++` command to compile your code for functional verification.

On Windows systems, you can use Microsoft Visual Studio to debug your component and testbench, even if you used the `i++` command to compile your code for functional verification.

Using the g++ command or Microsoft Visual Studio might require additional configuration to compile your Intel HLS Compiler Pro Edition code. For details, see Compiler Interoperability in the *Intel High Level Synthesis Compiler Pro Edition Reference Manual*.

You can automate the process by using a makefile or batch script. Use the makefiles and scripts provided in the Intel HLS Compiler Pro Edition example designs and tutorials as guides for creating your own makefiles or batch scripts.

5. Optimizing and Refining Your Component

After you have verified the functionality of your component and testbench, you can compile your component to RTL and review the High-Level Design Reports to further optimize and refine your component design. The High-Level Design Reports show estimates of various aspects of how your component will be implemented in hardware.

By compiling your component to RTL and reviewing the High-Level Design Reports, you can see how your code changes affect your component hardware implementation without needing to run a simulation or a full Quartus compilation.

To compile your component to RTL without running a simulation, issue the following command:

```
i++ -march="<FPGA_family_or_part_number>" --simulator none
```

You can also compile your component with a Questa* simulation flow by omitting the `--simulator none` option. Compiling without a simulation test bench is faster, but you cannot simulate your design to measure its latency and generate waveforms.

To view the High-Level Design Reports, open the following file in a web browser:

```
<result>.prj/reports/report.html
```

For information about techniques that you can apply to optimize and refine your component, see *Intel High Level Synthesis Compiler Pro Edition Best Practices Guide*.

5.1. The High-Level Design Reports

The High-Level Design Reports are a group of reports and viewers that you can use to help optimize your design by reviewing the statistics and visualizations that the reports provide.

Access the High-Level Design Reports by using a web browser to open the `report.html` file found in the `<result>.prj/reports` folder created when you compile your component to RTL.

Use the tutorials provided with the Intel HLS Compiler to view examples of the reports and learn how to use the reports and viewers to help optimize and refine your component design.

For details about using the reports to help optimize your design, review Reviewing the High-Level Design Reports (report.html) on page 24.

In some cases, information in the reports and viewers are estimates and might not match the results from compiling your design with Quartus Prime software. However, compiling your component using the Quartus Prime software might take several hours. In contrast, the Intel HLS Compiler can generate the High Level Design Report in minutes for most designs.