Intel® High Level Synthesis Compiler Pro Edition

Version 21.1 Release Notes

Updated for Intel® Quartus® Prime Design Suite: **21.1**

Subscribe
Send Feedback

RN-1146 | 2021.03.29

Latest document on the web: PDF | HTML

Contents

1. Intel® High Level Synthesis Compiler Pro Edition Version 21.1 Release Notes

The *Intel® High Level Synthesis Compiler Pro Edition Release Notes* provide late-breaking information about the Intel High Level Synthesis Compiler Pro Edition Version 21.1.

For the most recent Standard Edition release notes, see the Intel High Level Synthesis Compiler Standard Edition Release Notes.

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 Provide 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
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
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
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
Quick Reference Provides a brief summary of Intel HLS Compiler declarations and attributes on a single two-sided page.	Link

1.1. New Features and Enhancements

The Intel High Level Synthesis Compiler Pro Edition Version 21.1 includes the following new features:

- Added the `--dsp-mode` option of the `i++` command to control how some supported data types and math functions are implemented in hardware at a global scope.
- Added the `ihc::math_dsp_control` function to control how some supported data types and math functions are implemented in hardware at a local scope.
- Added the following new tutorials:
  - `<quartus_installdir>/hls/examples/tutorials/system_of_tasks/launch_and_collect_capacity`
  - `<quartus_installdir>/hls/examples/tutorials/best_practices/control_of_dsp_usage`
- For Intel Agilex™ devices, the compiler now takes advantage of a new floating point accumulator block to achieve II of 1 for single-precision floating point calculations when you specify `#pragma ii`". Review the Bottlenecks pane of the Loop Analysis report (part of the High-Level Design Reports `[report.html]`)Throughput Analysis report to see when the compiler suggests adding this pragma to your code.

  For other device families, the compiler infers a different floating-point accumulator block to achieve II of 1 for single-precision floating point calculations automatically.
- Enhanced the Schedule Viewer with dependency lines that show the order of execution for the instructions in your design.

1.2. Changes in Software Behavior

This section documents instances where Intel HLS Compiler Pro Edition Version 21.1 features have changed from earlier releases of the compiler.

- In `math.h`, the return type of `signbit(double)` is change to `int`. The previous return type was `long`.
- The `<`*`quartus_installdir`*`>/hls/examples/tutorials/usability/compiler_interoperability` tutorial has been revised.
- The `-ffp-reassoc` command option is deprecated. Use the `-ffp-reassociate` command option instead.
- The `fp reassoc` scoped pragma is deprecated. Use the `fp reassociate` scoped pragma instead.
- In the High-Level Design Reports, the **Graph Viewer** was renamed to **System Viewer**.
- In the High-Level Design Reports, the **System Viewers** menu was renamed to **Views**. The order of the top-level menus has also changed.

1.3. Intel High Level Synthesis Compiler Pro Edition Prerequisites

The Intel HLS Compiler Pro Edition is part of the Intel Quartus® Prime Pro Edition Design Suite. You can install the Intel HLS Compiler as part of your Intel Quartus Prime software installation or install it separately. It requires Intel Quartus Prime and additional software to use.

For detailed instructions about installing Intel Quartus Prime Pro Edition software, including system requirements, prerequisites, and licensing requirements, see Intel FPGA Software Installation and Licensing.

The Intel HLS Compiler requires the following software in addition to Intel Quartus Prime:

C++ Compiler

On Linux, Intel HLS Compiler requires GCC 9.3.0 including the GNU C++ library and binary utilities (binutils).

This version of GCC is provided as part of your Intel HLS Compiler installation. After installing the Intel HLS Compiler, GCC 9.3.0 is available in `<`*`quartus_installdir`*`>/gcc`.

Important: The Intel HLS Compiler uses the `<`*`quartus_installdir`*`>/gcc` directory as its toolchain directory. Use this installation of GCC for all your HLS-related design work.

For Windows, install one of the following versions of Microsoft* Visual Studio* Professional:

- Microsoft Visual Studio 2017 Professional
- Microsoft Visual Studio 2017 Community

For the most up-to-date C++17 support, ensure that you are using the latest version of Visual Studio 2017.

Important: The Intel HLS Compiler software does not support versions of Microsoft Visual Studio other than those specified for the edition of the software.

Mentor Graphics* ModelSim* Software

On Windows and RedHat Linux systems, you can install the ModelSim* software from the Intel Quartus Prime software installer. The available options are:

- ModelSim - Intel FPGA Edition
- ModelSim - Intel FPGA Starter Edition

Alternatively, you can use your own licensed version of Mentor Graphics* ModelSim or Mentor Graphics Questa* Advanced Simulator software.

On Linux systems, ModelSim software requires the Red Hat* development tools packages. Additionally, any 32-bit versions of ModelSim software (including those provided with Intel Quartus Prime) require additional 32-bit libraries. The commands to install these requirements are provided in Installing the Intel HLS Compiler on Linux Systems.

For information about all the ModelSim software versions that the Intel software supports, refer to the *EDA Interface Information* section in the Software and Device Support Release Notes for your edition of Intel Quartus Prime Pro Edition

Related Information

- Intel High Level Synthesis Compiler Getting Started Guide
- Supported Operating Systems
- Software Requirements
  in *Intel FPGA Software Installation and Licensing*
- EDA Interface Information (Intel Quartus Prime Pro Edition)
- Mentor Graphics ModelSim Website

1.4. Known Issues and Workarounds

This section provides information about known issues that affect the Intel HLS Compiler Pro Edition Version 21.1.

Description	Workaround
(Windows only) Compiling a design in a directory with a long path name can result in compile failures. Check the `debug.log` file for "could not find file" errors. These errors can indicate that your path is too long.	Compile the design in a directory with a short path name.
(Windows only) A long path for your Intel Quartus Prime installation directory can prevent you from successfully compiling and running the Intel HLS Compiler tutorials and example designs. Check the `debug.log` file for "could not find file" errors. These errors can indicate that your path is too long.	Move the tutorials and examples to a short path name before trying to run them.
Libraries that target OpenCL* and are written in HLS cannot use streams or pipes as an interface between OpenCL code and the library written in HLS. However, the library in HLS can use streams or pipes if both endpoints are within the library (for example, a stream that connects two task functions).	N/A

continued...

<table>
<tr><th>Description</th><th>Workaround</th></tr>
<tr><td>Applying the ihc::maxburst parameter to Avalon® Memory-Mapped master interfaces can cause your design to hang in simulation.</td><td>N/A</td></tr>
<tr><td>

In some uncommon cases, if you have two classes whose constructors each require instances of the other class as input, the compiler might crash.

For example, compiling the following code snippet causes the compiler to crash:

```
struct foo;

struct bar {
  int a, b, c;
  bar() : a(0), b(0), c(0) {};
  bar(const foo x);
};

struct foo {
  int a, b, c;
  foo() : a(0), b(0), c(0) {};
  foo(const bar x) {};
};

bar::bar(const foo x) {};
```

</td><td>

Avoid creating a circular definition. Instead, use a pointer or reference in your copy constructor.

For example, transform the earlier code snippet into the following code and pass in the struct as a reference to the constructor:

```
struct bar {
  int a, b, c;
  bar() : a(0), b(0), c(0) {};
  bar(const foo &x);
};

struct foo {
  int a, b, c;
  foo() : a(0), b(0), c(0) {};
  foo(const bar &x) {};
};

bar::bar(const foo &x) {};
```

</td></tr>
<tr><td>Libraries that target OpenCL and are written in HLS might cause OpenCL kernels that include the library to have a more conservative incremental compilation.</td><td>N/A</td></tr>
<tr><td>

When developing a library, if you have a #define defining a value that you use later in a #pragma, the fpga_crossgen command fails.

For example, the following code cannot be compiled by the fpga_crossgen command:

```
#define unroll_factor 5

int foo(int array_size) {
  int tmp[100];
  int sum =0;
//pragma unroll unroll_factor
#pragma ivdep array(tmp) safelen(unroll_factor)
  for (int i=0;i<array_size;i++) {
    sum+=tmp[i];
  }
  return sum;
}
```

</td><td>

Use __pragma instead of #pragma.

For example, the following compiles successfully with the fpga_crossgen command:

```
#define unroll_factor 5

int foo(int array_size) {
  int tmp[100];
  int sum =0;
//pragma unroll unroll_factor
__pragma ivdep array(tmp) safelen(unroll_factor)
  for (int i=0;i<array_size;i++) {
    sum+=tmp[i];
  }
  return sum;
}
```

</td></tr>
<tr><td>When you use the -c command option to have separate compilation and linking stages in your workflow, and if you do not specify the -march option in the linking stage (or specify a different -march option value), your linking stage might fail with or without error messages.</td><td>Ensure that you use the same -march option value for both the compilation with the -c command option stage and the linking stage.</td></tr>
<tr><td>

Applying the hls_merge memory attribute to an array declared within an unrolled or partially unrolled loop causes copies of the array to be merged across the unrolled loop iterations.

```
#pragma unroll 2
for (int I = 0; I < 8; i++) {
   hls_merge(“WidthMerged”, “width”) int MyMem1[128];
   hls_merge(“WidthMerged”, “width”) int MyMem2[128];
   ...
   hls_merge(“DepthMerged”, “depth”) int MyMem3[128];
   hls_merge(“DepthMerged”, “depth”) int MyMem4[128];
   ...
}
```

</td><td>

Avoid using the hls_merge memory attribute in unrolled loops.

If you need to merge memories in an unrolled loop, explicitly declare an array of struct type for width merging, or declare a deeper array for depth merging.

```
struct Type {int A; int B;};
#pragma unroll 2
for (int I = 0; I < 8; i++) {
   Type WidthMerged[128];  // Manual width merging
   ...
   int DepthMerged[256];   // Manual depth merging
   ...
}
```

</td></tr>
</table>

continued...

Description	Workaround
In the Function Memory Viewer high-level design report, some function-scoped memories might appear as "optimized away".	None. When a file contains functions that are components and functions that are not components, all function-scoped variables are listed in the Function Memory List pane, but only variables from components have information about them to show in the Function Memory View pane.
Some high-level design reports fail in Microsoft Internet Explorer*.	Use one of the following browsers to view the reports: • Google Chrome* • Microsoft Edge* • Mozilla* Firefox*
The Loop Viewer in the High-Level Design Reports has the following restrictions: • The behavior of stall-free clusters is not modeled in the Loop Viewer. The final latency shown in the Loop Viewer for a stall-free cluster is typically more pessimistic (that is, higher) than the actual latency of your design. For a description of clustering and stall-free clusters, refer to *Clustering the Datapath* in the *Intel High Level Synthesis Compiler Pro Edition Best Practices Guide*. • Stalls from reads and writes from memory or print statements are not modeled. • High-iteration counts (>1000) cause slow performance of the Loop Viewer. • You cannot specify an iteration count of zero (0) in the Loop Viewer.	None.
Links in some reports in the High-Level Design Reports generated on Windows systems do not work.	Generate the High-Level Design Reports (that is, compile your code) on a Linux system.

Using a `struct` of a single `ac_int` data type in steaming interface that uses packets (`ihc::usesPackets<true>`) does not work.
For example, the following code snippet does not work:
`// class definition`
`class DataType {`
`      ac_int<155, false> data;`
`...`
`}`
`// stream definition`
`typedef ihc::stream_in<DataType,`
`                       ihc::usesPackets<true>,`
`                       ihc::usesEmpty<true>`
`                      > DataStreamIn;`
	To use this combination in your design, obey the following restrictions:
	• The internal `ac_int` data size must be multiple of 8
	• The stream interface type declaration must specify `ihc::bitsPerSymbol<8>`
	For example, the following code snippet works:
	`// class definition`
	`class DataType {`
	`      ac_int<160, false> data;`
	`// data width must be multiple of 8`
	`...`
	`}`
	`// stream definition`
	`typedef ihc::stream_in<DataType,`
	`                       ihc::usesPackets<true>,`
	`                       ihc::usesEmpty<true>,`
	`                       ihc::bitsPerSymbol<8>`
	`                      > DataStreamIn;`
	`// added ihc::bitsPerSymbol<8>`

When running a high-throughput simulation of your component using enqueue function calls, if you do not use the `ihc_hls_component_run_all` function to run the enqueued component calls after all of the `ihc_hls_enqueue` calls for that component, the following behaviors occur: • In emulation, the enqueued component functions are run. • In simulation, the enqueued component functions are not run, with no error or warning messages provided.	Ensure that you use the `ihc_hls_component_run_all` function after all of the `ihc_hls_enqueue` calls for that component to run enqueued component function calls.

continued...

<table>
<tr><th>Description</th><th>Workaround</th></tr>
<tr><td>Launching a task function with <code>ihc::launch_always_run</code> strips away optimization attributes applied to the task function.
In the following code example, the attribute applied to the function is ignored. The High-Level Design Reports show an II of 1 for this task instead of the requested II of 4.<pre>
hls_component_ii(4) void noop()
{
    bool sop, eop;
    int empty;
    auto const data = data_in.read(sop, eop, empty);

    data_out.write(data, sop, eop, empty);
}

component void main_component()
{
    ihc::launch<noop>();
}
</pre></td><td>To avoid stripping away the optimization, add a <code>while(1)</code> loop to the affected function apply the corresponding control pragma to the <code>while(1)</code> loop instead of the function.
The following code example show how you can implement this change for the earlier code example:<pre>
void noop()
{
#pragma ii 4
    while (1)
    {
        bool sop, eop;
        int empty;
        auto const data = data_in.read(sop, eop,
empty);

        data_out.write(data, sop, eop, empty);
    }
}

component void
main_component()
{
    ihc::launch_always_run<noop>();
}
</pre></td></tr>
<tr><td>For Cyclone® V projects that contain multiple HLS components, when you use the <code>i++</code> command to compile your project to hardware (<code>i++ -march=CycloneV</code>), you might receive an error.
While the error text differs depending on your project, the error signature is an Intel Quartus Prime compilation failure due to bad Verilog syntax. A module tries to use a function that the Intel Quartus Prime compiler cannot find.</td><td>If you encounter this issue, put each HLS component in a separate project.</td></tr>
<tr><td>When you specify the <code>-ghdl=1</code> option of the <code>i++</code> command, the simulation of your component can fail.
Specifying this option does not affect the generated RTL. The option affects only the generated simulation script.</td><td>When compiling your design for simulation, specify the <code>-ghdl</code> option instead.
When you use this option, you prevent your simulation from failing, but this option results in a larger simulation waveform.</td></tr>
</table>

1.5. Intel High Level Synthesis Compiler Pro Edition Release Notes Archives

Intel HLS Compiler Version	Title
20.4	Intel High Level Synthesis Compiler Pro Edition Version 20.4 Release Notes
20.3	Intel High Level Synthesis Compiler Pro Edition Version 20.3 Release Notes
20.2	Intel High Level Synthesis Compiler Pro Edition Version 20.2 Release Notes
20.1	Intel High Level Synthesis Compiler Pro Edition Version 20.1 Release Notes
19.4	Intel High Level Synthesis Compiler Pro Edition Version 19.4 Release Notes
19.3	Intel High Level Synthesis Compiler Pro Edition Version 19.3 Release Notes
19.2	Intel High Level Synthesis Compiler Pro Edition Version 19.2 Release Notes
19.1	Intel High Level Synthesis Compiler Pro Edition Version 19.1 Release Notes
18.1	Intel High Level Synthesis Compiler Version 18.1 Release Notes
18.0	Intel High Level Synthesis Compiler Version 18.0 Release Notes
17.1	Intel High Level Synthesis Compiler Version 17.1 Release Notes

1.6. Document Revision History for Intel HLS Compiler Pro Edition Version 21.1 Release Notes

Document Version	Intel Quartus Prime Version	Changes
2021.03.29	21.1	• Initial release.