Intel® High Level Synthesis Compiler Pro Edition

Version 20.1 Release Notes

Updated for Intel® Quartus® Prime Design Suite: **20.1**

Subscribe
Send Feedback

RN-1146 | 2020.05.13
Latest document on the web: PDF | HTML

Contents

1. Intel® High Level Synthesis Compiler Pro Edition Version 20.1 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 20.1.

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 20.1 includes the following new features:

- Added the `ihc::launch_always_run` function that enables you to launch a task function at component power-on or reset, and continuously executes the task function.
- Added *capacity* parameter to the `ihc::launch` function that enables you to add a FIFO buffer with a *capacity* size between the function launching the task and the task function. Adding a FIFO buffer can help improve capacity balancing in your system of tasks.
- Added the `hls_force_pow2_depth` memory attribute to control padding of on-chip memories.
- Added support for concurrent memory accesses to slave memories from outside and within the component when the `volatile` keyword is applied to the slave memory argument.
- Added the `hls_readwrite_mode` attribute that can be applied to slave memory arguments to indicate how the slave memory interface is accessed by external Avalon MM masters.
- Added the `hls_private_copies` attribute. This attribute replaces the `hls_max_concurrency` memory attribute.
- Enhanced component throughput with automated loop fusion, which reduces the number of loop control structures needed in your component.
- Added the `loop_fuse` and `nofusion` pragmas that you can use to influence and control loop fusion when compiling your component.
- Added the `hls_use_stall_enable_clusters` component attribute to force components to use stall-enabled clusters where possible. Stall-enabled clusters cans help improve component latency and area usage while maybe sacrificing throughput when compared to the default stall-free clustering implementation.
- Added the HLS_EXTERNAL macro. You must apply this macro to functions in HLS source code that you want to be callable from outside of the library.

  Review your existing library source code to ensure that you meet this requirement.
- Added pipe support to HLS components and tasks. Pipes allow communication between tasks, and allow you to write HLS function with streaming behavior that can be used in Intel oneAPI designs.
- Moved Schedule Viewer into beta.

Send Feedback

1.2. Changes in Software Behavior

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

- Added requirement that you must apply the HLS_EXTERNAL macro to functions in your library source code if you want the functions to callable from outside of the library.

  Review your existing library source code to ensure that you meet this requirement.
- Deprecated the `hls_max_concurrency` memory attribute. This attribute might be removed from a future version. Use `hls_private_copies` instead.
- Removed the following memory attributes. They are no longer supported.
  - `hls_numreadports`
  - `hls_numwriteports`
  - `hls_numports_readonly_writeonly`

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 9.1.0 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 9.1.0 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 the Microsoft Visual Studio Professional:

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

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 software.

On RedHat 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.

On SUSE Linux systems, you must use your own licensed version of Mentor Graphics ModelSim software.

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 20.1.

Description	Workaround
(Windows only) Compiling a design in a directory with a long path name can result in compile failures.	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.	Move the tutorials and examples to a short path name before trying to run them.
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.	Ensure that you use the same `-march` option value for both the compilation with the `-c` command option stage and the linking stage.
Applying the `hls_merge` memory attribute to an array declared within an unrolled or partially unrolled loop creates an unexpectedly wide memory.	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.

continued...

Send Feedback

Description	Workaround
Slave memories cannot be implemented as MLABs. They can be implemented only as M20K blocks.	N/A
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.

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;`
`}`
	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;`
	`}`

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*
Libraries that target OpenCL* and are written in HLS might cause OpenCL kernels that include the library to have a more conservative incremental compilation.	N/A
Libraries that target OpenCL and are written in HLS cannot use streams or pipes.	N/A
(Windows only) Library functions that use the following HLS features cannot be emulated: • `hls_float` data type • `ac_fixed` data type • System of tasks	N/A

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>`

continued...

Description	Workaround

RTL libraries with RTL source files of certain names fail during compilation when such a library is used. Known bad names:
• `delay.sv`
• `pe.sv`
Example error in `debug.log`:
`Error: efi_function: add_fileset_file:`
`No such file a.prj/lib/my_efi/my_efi/delay.sv`
`while executing "add_fileset_file "delay.sv"`
`SYSTEM_VERILOG PATH "a.prj/lib/my_efi/my_efi/..."`
	Avoid using RTL source file names that match terms for established RTL concepts.

(Windows only) Pipes are not supported on Windows platforms.	N/A

Loop fusion may cause the compiler to crash when one (or more) loops are contained within conditional statements, and the value of the condition is computed in the previous loop.
For example:
`bool repeated;`
`for (int i = 0; i < N; ++i) {`
`    if (i+1 < N) {`
`        repeated = data[i] == data[i+1];`
`    }`
`}`
`if (repeated)`
`{`
`    for (int i = 0; i < N; ++i)`
`    {`
`        int val = data[i];`
`        data2[val][i] += value[i];`
`    }`
`}`
	Mark one of the loops with `#pragma nofusion`.
	For example:
	`bool repeated;`
	`#pragma nofusion`
	`for (int i = 0; i < N; ++i) {`
	`    if (i+1 < N) {`
	`        repeated = data[i] == data[i+1];`
	`    }`
	`}`
	`if (repeated)`
	`{`
	`    for (int i = 0; i < N; ++i)`
	`    {`
	`        int val = data[i];`
	`        data2[val][i] += value[i];`
	`    }`
	`}`

When you apply the `hls_avalon_slave_memory_argument` attribute on an array function argument, the compiler crashes.
For example, the following code causes the compiler to crash:
`component void foo(hls_avalon_slave_memory_argument`
`                    (128*sizeof(int)) int A[128])`
	Use a pointer argument instead. The two styles are equivalent.
	For example:
	`component void foo(hls_avalon_slave_memory_argument`
	`                    (128*sizeof(int)) int *A)`

1.5. Software Issues Resolved

The following issues were corrected or otherwise resolved in the Intel HLS Compiler Pro Edition Version 20.1.

Table 2. Issues Resolved in the Intel HLS Compiler Pro Edition Version 20.1

Customer Service Request Numbers						
00412506	00477655					

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

Intel HLS Compiler Version	Title
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

1.7. Document Revision History for Intel HLS Compiler Pro Edition Version 20.1 Release Notes

Document Version	Intel Quartus Prime Version	Changes
2020.05.13	20.1	• Updated Known Issues and Workarounds on page 6 with an issue that affects the use of the `hls_avalon_slave_memory_argument` attribute.
2020.04.17	20.1	• Updated Known Issues and Workarounds on page 6 with an issue that affects loop fusion.
2020.04.13	20.1	• Initial release.