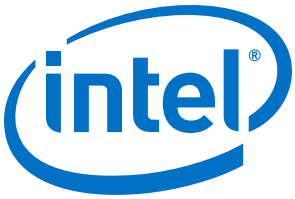

Intel® High Level Synthesis Compiler

Getting Started Guide

Updated for Intel® Quartus® Prime Design Suite: **19.3**

UG-20036 | 2019.09.30
Latest document on the web: PDF | HTML

Contents

UG-20036 | 2019.09.30

Send Feedback

1. Intel® High Level Synthesis (HLS) Compiler Getting Started Guide

The Intel® High Level Synthesis (HLS) Compiler is separately-installable component of Intel Quartus® Prime design software. The Intel HLS Compiler synthesizes a C++ function into an RTL implementation that is optimized for Intel FPGA products. The compiler is sometimes referred to as the i++ compiler, reflecting the name of the compiler command.

The *Intel High Level Synthesis Compiler Getting Started Guide* describes the procedures to set up the Intel HLS Compiler and to run an HLS design example.

The features and devices supported by the Intel HLS Compiler depend on what edition of Intel Quartus Prime you have. The following icons indicate content in this publication that applies only to the Intel HLS Compiler provided with a certain edition of Intel Quartus Prime:

PRO Indicates that a feature or content applies only to Intel HLS Compiler Pro Edition.

STD Indicates that a feature or content applies only to Intel HLS Compiler Standard Edition.

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

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

PRO

Windows	`C:\intelFPGA_pro\19.3`
Linux	`/home/<username>/intelFPGA_pro/19.3`

STD

Windows	`C:\intelFPGA_standard\19.3`
Linux	`/home/<username>/intelFPGA_standard/19.3`

1.1. Intel High Level Synthesis Compiler Prerequisites

The Intel HLS Compiler is part of the Intel Quartus Prime Design Suite. You can install it 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 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

For Linux, install one of the following versions of the GCC compiler and C++ libraries, depending on your edition of Intel Quartus Prime software:

- PRO GCC compiler and C++ Libraries version 5.4.0

  You must install these libraries manually. See Installing the Intel HLS Compiler on Linux Systems for instructions.

- STD GCC compiler and C++ Libraries version 4.4.7

  These libraries are included in the version of Linux supported by the Intel HLS Compiler.

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

For Windows, install one of the following versions of the Microsoft Visual Studio Professional, depending on your edition of Intel Quartus Prime software:

- PRO Microsoft Visual Studio 2017 Professional
- PRO Microsoft Visual Studio 2017 Community
- STD Microsoft Visual Studio 2010 Professional

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

Related Information

- Supported Operating Systems
- Software Requirements
  in *Intel FPGA Software Installation and Licensing*
- EDA Interface Information (Intel Quartus Prime Standard Edition)
- EDA Interface Information (Intel Quartus Prime Pro Edition)
- Mentor Graphics ModelSim Website

1.1.1. PRO Intel HLS Compiler Backwards Compatibility

The Intel HLS Compiler Version 19.3 is compatible with Intel Quartus Prime Pro Edition Version 19.3, Version 19.2, Version 19.1, and Version 17.1.1. This backwards compatibility lets you to take advantage of improvements in the RTL generated by the Intel HLS Compiler without changing the other parts of your current FPGA development environment.

To install Intel HLS Compiler Version 19.3 into an older version of Intel Quartus Prime:

1. Backup the existing <*quartus_installdir*>[1]/hls directory. For example, rename <*quartus_installdir*>/hls to <*quartus_installdir*>/hls_old.
2. Download the Intel HLS Compiler standalone installation package from the **Individual Files** tab of the Intel Quartus Prime Download Center for your edition of Intel Quartus Prime.
3. Run the Intel HLS Compiler standalone installer, and specify your <*quartus_installdir*> directory when prompted.

You can revert to your previous version of the Intel HLS Compiler by renaming your backup directory. Whatever version of the the Intel HLS Compiler that in a directory called <*quartus_installdir*>/hls is the active version of the Intel HLS Compiler.

1.2. Downloading the Intel HLS Compiler

Download the Intel HLS Compiler from the Intel Quartus Prime Download Center.

STD The Intel HLS Compiler is installed as part of your Intel Quartus Prime Standard Edition installation package.

PRO The Intel HLS Compiler is installed as part of your Intel Quartus Prime Pro Edition installation package or from a separate standalone installer.

(1) Where <*quartus_installdir*> is the directory where you installed the Intel Quartus Prime Design Suite. For example, C:\intelFPGA_pro\19.3.

PRO Download the Intel HLS Compiler standalone installer when you want to add the Intel HLS Compiler to an existing Intel Quartus Prime Pro Edition installation. The Intel HLS Compiler Version 19.3 is compatible with Intel Quartus Prime Pro Edition Version 19.3, Version 19.2, Version 19.1, and Version 17.1.1.

- Download the Intel HLS Compiler from the Intel Quartus Prime Download Center for your edition of Intel Quartus Prime:
  - PRO http://fpgasoftware.intel.com/?edition=pro
  - STD http://fpgasoftware.intel.com/?edition=standard

  PRO The Intel HLS Compiler standalone installation package is available on the **Additional Software** tab.

  To include the Intel HLS Compiler in your Intel Quartus Prime installation, download one of the following combinations of installation packages:
  - Download a complete installation package from the **Combined Files** tab.
  - PRO Download a Quartus Prime installation package from the **Individual Files** tab and download the Intel HLS Compiler installation package from the **Additional Software** tab. Place both installation packages in the same directory.

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

  PRO The Intel HLS Compiler standalone installation package requires that you have an existing Intel Quartus Prime Pro Edition installation as a target for the Intel HLS Compiler installation.

1.3. Installing the Intel HLS Compiler on Linux Systems

You must have administrator privileges to install the Intel HLS Compiler prerequisites. However, the Intel HLS Compiler and Intel Quartus Prime do not require administrator privileges to install.

To install the Intel HLS Compiler on Linux Systems:

1. Confirm that your operating system version is supported by the Intel HLS Compiler:

Option	Description
PRO	• Red Hat Enterprise Linux 6.x, or a community equivalent • Red Hat Enterprise Linux 7.x, or a community equivalent • SUSE Linux Enterprise Server 12
STD	• Red Hat Enterprise Linux 6.x, or a community equivalent

2. (SUSE Linux only) Ensure that you have the Toolchain Module available.
3. Depending on your current system and what installation package or packages you have downloaded, you might have to complete some additional steps to prepare your installation:

- If you are installing the complete Intel Quartus Prime installation package (from the **Combined Files** tab of the Quartus Prime download page at the Download Center for FPGAs):

  No additional steps are needed before running the installation package.

- PRO If you are installing separately downloaded Intel Quartus Prime (from **Individual Files** tab of the Quartus Prime download page at the Download Center for FPGAs) and Intel HLS Compiler (from **Additional Software** tab of the Quartus Prime download page at the Download Center for FPGAs) installation packages:

  Ensure that you have both installation packages in the same directory. The Intel Quartus Prime installer detects the Intel HLS Compiler installation package and installs both software packages for you.

- PRO If you are updating the Intel HLS Compiler in an existing Intel Quartus Prime Pro Edition installation:

  a. Ensure that you have Intel Quartus Prime Pro Edition already installed.

  b. Take note of the path to your Intel Quartus Prime installation.

     You need this path information to complete the Intel HLS Compiler installation wizard.

  c. Rename the HLS directory in your current Intel Quartus Prime version to keep that version as a backup.

     For example, if you are installing Intel HLS Compiler Version 19.3 into an Intel Quartus Prime Pro Edition Version 19.2 installation, rename `/home/<username>/intelFPGA_pro/19.2/hls` to `/home/<username>/intelFPGA_pro/19.2/hls_old`.

4. Install the package that you downloaded in Downloading the Intel HLS Compiler on page 5.

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

5. Update your Linux repositories with the following command:

   ```
   sudo yum update
   ```

6. Install one of the following versions depending on your edition of the Intel Quartus Prime software:

   - PRO GCC compiler and C++ Libraries version 5.4.0

     To download the required tools and source files, and compile GCC compiler and C++ Libraries version 5.4.0, run the following commands:

     ```
     $ sudo yum groupinstall "Development Tools" "Additional Development"
     $ cd <quartus_installdir>/hls/
     $ ./install_gcc
     ```

     The `install_gcc` command runs a script that downloads and compiles the GCC compiler and C++ Libraries version 5.4.0 in the `/home/<username>/build/gcc` folder. The script installs this version of the GCC compiler alongside the Intel HLS Compiler so that the Intel HLS Compiler can access the required libraries without any further action on your part.

If you want to maintain your own GCC installation, run the script with the `-i` flag to specify a different GCC installation directory. However, if you specify a different GCC installation directory, you must specify the `--gcc-toolchain` option every time you run the Intel HLS Compiler `i++` command. For details about the `--gcc-toolchain` option, see Intel HLS Compiler Command Options in *Intel HLS Compiler Reference Manual*.

For more options of the `install_gcc` command, use the command help option: `./install_gcc -h`.

— STD GCC compiler and C++ Libraries version 4.4.7

You can install GCC compiler and C++ Libraries version 4.4.7 with the following command:

```
$ sudo yum groupinstall "Development Tools"
```

7. (RedHat Linux Only) If you want the Intel HLS Compiler to simulate your components with the 32-bit Mentor Graphics ModelSim software provided with Intel Quartus Prime (ModelSim - Intel FPGA Edition), install the required additional 32-bit libraries with the following command:
   - Red Hat Enterprise Linux 6.x, or a community equivalent

     ```
     $ sudo yum install -y glibc.i686 glibc-devel.i686 libX11.i686 \
                    libXext.i686 libXft.i686 libgcc.i686 libgcc.x86_64 \
                    libstdc++.i686 libstdc++-devel.i686 ncurses-devel.i686 \
                    qt.i686 qt-x11.i686
     ```

   - Red Hat Enterprise Linux 7.x, or a community equivalent

     ```
     $ sudo yum install -y glibc.i686 glibc-devel.i686 libX11.i686 \
                    libXext.i686 libXft.i686 libgcc.i686 libgcc.x86_64 \
                    libstdc++.i686 libstdc++-devel.i686 ncurses-devel.i686 \
                    qt.i686
     ```

   - SUSE Linux Enterprise Server 12

     ModelSim - Intel FPGA Edition is not supported on SUSE Linux. Use you own licensed version of Mentor Graphics ModelSim software.
8. (RedHat Linux Only) If you use the Mentor Graphics ModelSim software provided with Intel Quartus Prime, add the path to ModelSim to your PATH environment variable

   For example:

   ```
   $ export PATH=$PATH:<quartus_installdir>/modelsim_ase/bin
   ```

9. Optional: If you plan to use Platform Designer to integrate your component with a system, add the path to Platform Designer to your PATH environment variable.

   For example:

   ```
   $ export PATH=$PATH:<quartus_installdir>/qsys/bin
   ```

After completing these steps, the Intel HLS Compiler is installed on your system. Before you can compile your component with the Intel HLS Compiler `i++` command, you must initialize your Intel HLS Compiler environment for the `i++` command to run successfully. For details, see Initializing the Intel HLS Compiler Environment on page 10.

1.4. Installing the Intel HLS Compiler on Microsoft* Windows* Systems

To install the Intel HLS Compiler on Microsoft* Windows* Systems:

1. Confirm that your operating system version is supported by the Intel HLS Compiler (Microsoft* Windows* 7 SP1, 8.1 or 10).
2. Install one of the following software products, depending on your edition of the Intel Quartus Prime software:
   - PRO Microsoft Visual Studio 2017 Professional
   - PRO Microsoft Visual Studio 2017 Community
   - STD Microsoft Visual Studio 2010 Professional

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

   If you have multiple versions of Visual Studio, Microsoft recommends installing Visual Studio versions in the order in which the versions were released. For example, install Visual Studio 2010 before installing Visual Studio 2015. For details, see Install Visual Studio Versions Side-by-Side in the MSDN Library.
3. Depending on your current system and what installation package or packages you have downloaded, you might have to complete some additional steps to prepare your installation:
   - If you are installing complete Intel Quartus Prime installation package (from the **Combined Files** tab of the Quartus Prime download page at the Download Center for FPGAs):

     No additional steps are needed before running the installation package.
   - PRO Installing separately downloaded Intel Quartus Prime (from **Individual Files** tab of the Quartus Prime download page at the Download Center for FPGA) and Intel HLS Compiler (from the **Additional Software** tab of the Quartus Prime download page at the Download Center for FPGA) installation packages:

     Ensure that you have both installation packages in the same directory. The Intel Quartus Prime installer detects the Intel HLS Compiler installation package and installs both software packages for you.
   - PRO Updating the Intel HLS Compiler in your Intel Quartus Prime Pro Edition installation:
     a. Ensure that you have Intel Quartus Prime Pro Edition already installed.
     b. Take note of the path to your Intel Quartus Prime installation.

        You need this path information to complete the Intel HLS Compiler installation wizard.
     c. Rename the HLS directory in your current Intel Quartus Prime version to keep that version as a backup.

For example, if you are installing Intel HLS Compiler Version 19.3 into an Intel Quartus Prime Pro Edition Version 19.2 installation, rename `C:\intelFPGA_pro\19.2\hls` to `C:\intelFPGA_pro\19.2\hls_old`.

4. Install the package that you downloaded in Downloading the Intel HLS Compiler on page 5.

   For detailed instructions about installing Intel Quartus Prime software, including system requirements, prerequisites, and licensing requirements, refer to the Intel FPGA Software Installation and Licensing.

5. If you use the Mentor Graphics ModelSim software provided with Intel Quartus Prime, add the path to ModelSim to your PATH environment variable.

   For example:

   ```
   set PATH=%PATH%:<quartus_installdir>\modelsim_ase\win32aloem
   ```

6. Optional: If you plan to use Platform Designer to integrate your component with a system, add the path to Platform Designer to your PATH environment variable.

   For example:

   ```
   set PATH=%PATH%:<quartus_installdir>\qsys\bin
   ```

After completing these steps, the Intel HLS Compiler is installed on your system. Before you can compile your component with the Intel HLS Compiler `i++` command, you must initialize your Intel HLS Compiler environment for the `i++` command to run successfully. For details, see Initializing the Intel HLS Compiler Environment on page 10.

1.5. Initializing the Intel HLS Compiler Environment

Before you can compile your component with the Intel HLS Compiler `i++` command, a number of environment variables must be set for the `i++` command to run successfully.

The Intel HLS Compiler environment initialization script applies only to the environment variable settings in your *current* terminal or command prompt session. You must initialize the Intel HLS Compiler environment each time that you start a terminal or command prompt session to develop your design.

To initialize your current terminal or command prompt session so that you can run the Intel HLS Compiler:

- On Linux systems, initialize your environment as follows:
  a. In your terminal session, change directories to the `hls` directory in your Intel Quartus Prime installation directory.

     For example, `/home/<username>/intelFPGA_pro/19.3/hls`

  b. Run the following command from the `hls` directory to set the environment variables for the `i++` command in the current terminal session:

     ```
     source init_hls.sh
     ```

     The command prints out the modified environment variable settings.

  The environment initialization script shows the environment variables that it set, and you can now run the `i++` command from this terminal session.
- On Windows systems, initialize your environment as follows:
  a. Start a Visual Studio x64 native tools command prompt session.

     For example, `C:\Program Files (x86)\Microsoft Visual Studio \2017\Professional\VC\Auxiliary\Build\vcvars64.bat` or `C:\Program Files (x86)\Microsoft Visual Studio \2017\Community\VC\Auxiliary\Build\vcvars64.bat`

  b. In your x64 Native Tools command prompt session, change directories to the `hls` directory in your Intel Quartus Prime installation directory.

     For example, `C:\intelFPGA_pro\19.3\hls`

  c. Run the following command from the `hls` directory to set the environment variables for the `i++` command in the current terminal session:

     ```
     init_hls.bat
     ```

     The command prints out the modified environment variable settings.

  The environment initialization script shows the environment variables that it set, and you can now run the `i++` command from this command prompt session.

Tip: To set the environment variables permanently, follow your operating system's standard procedure for making persistent changes to environment variable settings. Review the output of the environment initialization script to determine the environment variables to set permanently.

2. High Level Synthesis (HLS) Design Examples and Tutorials

The Intel High Level Synthesis (HLS) Compiler includes design examples and tutorials to provide you with example components and demonstrate ways to model or code your components to get the best results from the Intel HLS Compiler for your design.

High Level Synthesis Design Examples

The high level synthesis (HLS) design examples give you a quick way to see how various algorithms can be effectively implemented to get the best results from the Intel HLS Compiler.

You can find the HLS design examples in the following location:

```
<quartus_installdir>/hls/examples/<design_example_name>
```

Where *<quartus_installdir>* is the directory where you installed the Intel Quartus Prime Design Suite. For example, `/home/<username>/intelFPGA_pro/19.3` or `C:\intelFPGA_pro\19.3`.

For instructions on running the examples, see the following sections:

- Running a High Level Synthesis (HLS) Design Example (Linux) on page 17
- Running a High Level Synthesis (HLS) Design Example (Windows) on page 18

Table 1. HLS design examples

Focus area	Name	Description
Linear algebra	`QRD`	Uses the Modified Gram-Schmidt algorithm for QR factorization of a matrix.
Signal processing	`interp_decim_filter`	Implements a simple and efficient interpolation/decimation filter.
Simple design	`counter`	Implements a simple and efficient 32-bit counter component.
Video processing	`YUV2RGB`	Implements a basic YUV422 to RGB888 color space conversion.
Video processing	`image_downsample`	Implements an image downsampling algorithm to scale an image to a smaller size using bilinear interpolation.

HLS Design Tutorials

The HLS design tutorials show you important HLS-specific programming concepts as well demonstrating good coding practices.

Each tutorial has a README file that gives you details about what the tutorial covers and instructions on how to run the tutorial.

Table 2. Arbitrary precision datatypes design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/ac_datatypes`	
`ac_fixed_constructor`	Demonstrates the use of the `ac_fixed` constructor where you can get a better QoR by using minor variations in coding style.
`ac_fixed_math_library`	Demonstrates the use of the Intel HLS Compiler `ac_fixed_math` fixed point math library functions.
`ac_int_basic_ops`	Demonstrates the operators available for the `ac_int` class.
`ac_int_overflow`	Demonstrates the usage of the `DEBUG_AC_INT_WARNING` and `DEBUG_AC_INT_ERROR` keywords to help detect overflow during emulation runtime.
PRO You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/hls_float`	
PRO `1_reduced_double`	Demonstrates how your application can benefit from `hls_float` by changing the underlining type from `double` to `hls_float<11, 44>` (reduced double).
PRO `2_explicit_arithmetic`	Demonstrates how to use the explicit versions of `hls_float` binary operators to perform floating-point arithmetic operations based on your needs.
PRO `3_conversions`	Demonstrates when conversions appear in designs with `hls_float` types and how to use different conversion modes to generate compile-type constants using various `hls_float` types.

Table 3. Component memories design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/component_memories`	
PRO `memory_bank_configuration`	Demonstrates how to control the number of load/store ports of each memory bank and optimize your component area usage, throughput, or both by using one or more of the following memory attributes: • `hls_max_replicates` • `hls_singlepump` • `hls_doublepump` • `hls_simple_dual_port_memory`
PRO `memory_geometry`	Demonstrates how to control the number of load/store ports of each memory bank and optimize your component area usage, throughput, or both by using one or more of the following memory attributes: • `hls_bankwidth` • `hls_numbanks` • `hls_bankbits`
PRO `memory_implementation`	Demonstrates how to implement variables or arrays in registers, MLABs, or RAMs by using the following memory attributes: • `hls_register` • `hls_memory` • `hls_memory_impl`

continued...

Name	Description
PRO memory_merging	Demonstrates how to improve resource utilization by implementing two logical memories as a single physical memory by merging them depth-wise or width-wise with the `hls_merge` memory attribute.
PRO static_var_init	Demonstrates how to control the initialization behavior of statics in a component using the `hls_init_on_reset` or `hls_init_on_powerup` memory attribute.
PRO attributes_on_mm_slave_arg	Demonstrates how to apply memory attributes to Avalon® Memory Mapped (MM) slave arguments.
PRO exceptions	Demonstrates how to use memory attributes on constants and struct members.
STD bank_bits	Demonstrates how to control component internal memory architecture for parallel memory access by enforcing which address bits are used for banking.
STD depth_wise_merge	Demonstrates how to improve resource utilization by implementing two logical memories as a single physical memory with a depth equal to the sum of the depths of the two original memories.
STD static_var_init	Demonstrates the `hls_init_on_power` and `hls_init_on_reset` flags for static variables and their impact on area and latency.
STD width_wise_merge	Demonstrates how to improve resource utilization by implementing two logical memories as a single physical memory with a width equal to the sum of the widths of the two original memories.

Table 4. Interfaces design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/interfaces`	
overview	Demonstrates the effects on quality-of-results (QoR) of choosing different component interfaces even when the component algorithm remains the same.
explicit_streams_buffer	Demonstrates how to use explicit `stream_in` and `stream_out` interfaces in the component and testbench.
PRO explicit_streams_packets_empty	Demonstrates how to use the `usesPackets`, `usesEmpty`, and `firstSymbolInHighOrderBits` stream template parameters.
explicit_streams_packets_ready_valid	Demonstrates how to use the `usesPackets`, `usesValid`, and `usesReady` stream template parameters.
explicit_streams_ready_latency	Demonstrates how to achieve a better loop initiation interval (II) with stream write using the `readyLatency` stream template parameter.
mm_master_testbench_operators	Demonstrates how to invoke a component at different indicies of an Avalon Memory Mapped (MM) Master (`mm_master` class) interface.
mm_slaves	Demonstrates how to create Avalon-MM Slave interfaces (slave registers and slave memories).
multiple_stream_call_sites	Demonstrates the benefits of using multiple stream call sites.
pointer_mm_master	Demonstrates how to create Avalon-MM Master interfaces and control their parameters.
stable_arguments	Demonstrates how to use the `stable` attribute for unchanging arguments to improve resource utilization.

Table 5. Best practices design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/best_practices`	
PRO `ac_datatypes`	Demonstrates the effect of using `ac_int` datatype instead of `int` datatype.
`const_global`	Demonstrates the performance and resource utilization improvements of using `const` qualified global variables.
PRO `divergent_loops`	Demonstrates a source-level optimization for designs with divergent loops
`floating_point_ops`	Demonstrates the impact of `--fpc` and `--fp-relaxed` flags in i++ on floating point operations using a 32-tap finite impulse response (FIR) filter design that is optimized for throughput.
PRO `hyper_optimized_handshaking`	Demonstrates how to use the `--hyper-optimized-handshaking` option of the Intel HLS Compiler i++ command.
STD `integer_promotion`	Demonstrates how integer promotion rules can influence the behavior of a C or C ++ program.
PRO `loop_coalesce`	Demonstrates the performance and resource utilization improvements of using `loop_coalesce` pragma on nested loops. While the `#pragma loop_coalesce` is provided with both Standard and Pro edition, the design tutorial is provided only with Pro edition.
`loop_memory_dependency`	Demonstrates breaking loop carried dependencies using the `ivdep` pragma.
PRO `lsu_control`	Demonstrates the effects of controlling the types of LSUs instantiated for variable-latency Avalon MM Master interfaces.
`parameter_aliasing`	PRO Demonstrates the use of the `__restrict` keyword on component arguments. STD Demonstrates the use of the `restrict` keyword on component arguments.
PRO `random_number_generator`	Demonstrates how to use the random number generator library.
PRO `relax_reduction_dependency`	Demonstrates a method to reduce the II of a loop that includes a floating point accumulator, or other reduction operation that cannot be computed at high speed in a single clock cycle.
`resource_sharing_filter`	Demonstrates an optimized-for-area variant of a 32-tap finite impulse response (FIR) filter design
PRO `set_component_target_fmax`	Demonstrate how to use the `hls_scheduler_target_fmax_mhz` component attribute and how it interacts with the `ii` loop pragma..
`shift_register`	Demonstrates the recommended coding style for implementing shift registers.
PRO `sincos_func`	Demonstrates the effects of using `sinpi` or `cospi` functions in your component instead of `sin` or `cos` functions.
`single_vs_double_precision_math`	Demonstrates the effect of using single precision literals and functions instead of double precision literals and functions.

continued...

Name	Description
struct_interface	Demonstrates how to use ac_int to implement interfaces with no padding bits.
swap_vs_copy	Demonstrates the impact of using deep copying with registers on the performance and resource utilization of a component design.
PRO triangular_loop	Demonstrates a method for describing triangular loop patterns with dependencies.

Table 6. Usability design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: <quartus_installdir>/hls/examples/tutorials/usability	
compiler_interoperability	(Linux only) Demonstrates how to use testbench code compiled with GCC along with code compiled by the i++ command.
enqueue_call	Demonstrates how to run components asynchronously and exercise their pipeline performance in the test bench using enqueue functionality.
PRO platform_designer_2xclock STD qsys_2xclock	Demonstrates the recommended clock and reset generation for a component with a clock2x input.
PRO platform_designer_stitching STD qsys_stitching	Demonstrates how to combine multiple components to function as a single cohesive design.

Table 7. PRO System of tasks design tutorials

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: <quartus_installdir>/hls/examples/tutorials/system_of_tasks	
parallel_loop	Demonstrates how you can run sequential loops in a pipelined manner by using a system of HLS tasks in your component.
resource_sharing	Demonstrates how you can share expensive compute blocks in your component to save area usage.
balancing_pipeline_latency	Demonstrates how to improve the throughput of a component that uses a system of tasks by buffering streams.
balancing_loop_delay	Demonstrates how to improve the throughput of a component that uses a system of tasks by buffering streams.
internal_stream	Demonstrates how to use "internal streams" in HLS tasks with the ihc::stream object.
task_reuse	Demonstrates how to invoke multiple copies of the same task function.

Table 8. PRO **HLS Libraries design tutorials**

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/libraries`	
`basic_rtl_library_flow`	Demonstrates the process of developing an RTL library and using it in an HLS component.
`rtl_struct_mapping`	Demonstrates how to obtain a mapping from C++ struct fields to bit-slices of RTL module interface signals.

Table 9. PRO **HLS Loop Control tutorials**

Name	Description
You can find these tutorials in the following location on your Intel Quartus Prime system: `<quartus_installdir>/hls/examples/tutorials/loop_controls`	
`max_interleaving`	Demonstrates a method to reduce the area utilization of a loop that meets the following conditions: • The loop has an II > 1 • The loop is contained in a pipelined loop • The loop execution is serialized across the invocations of the pipelined loop

2.1. Running a High Level Synthesis (HLS) Design Example (Linux)

To run an HLS design example on Linux systems:

1. Start a terminal session and initialize the Intel HLS Compiler environment.

   For instructions how to initialize the environment, see Initializing the Intel HLS Compiler Environment on page 10.

2. Navigate to the `<quartus_installdir>/hls/examples/<design_example_name>` directory, where `<quartus_installdir>` is the directory where you installed Intel Quartus Prime software.

   For example, `/home/<username>/intelFPGA_pro/19.3`.

3. Run the `make test-x86-64` command. This command compiles the C++ source code to an x86-64 binary executable. Then, run the generated executable on your CPU.
   Expected outcome after you run the `make test-x86-64` command:

   - The console displays the command used to generate the binary. For example, `i++ -march=x86-64 -o test-x86-64 <source_files>`.
   - The HLS compiler creates an executable file (for example, `test-x86-64`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

```
$ make test-x86-64
i++ MGS.cpp QRD_Testbench.cpp TestbenchHelpers.cpp  --fpc --fp-relaxed -
march=x86-64 -o test-x86-64
```

```
+--------------------------------------+
| Run ./test-x86-64 to execute the test. |
+--------------------------------------+
```

4. Run the `make test-fpga` command. The command compiles the C++ source code to a hardware executable and then runs a simulation of the generated HDL. Expected outcome after you run the `make test-fpga` command:
   - The console displays the command it uses to generate the testbench binary and the contents of the project directory. For example, `i++ -march="<FPGA_family_or_part_number>" <source_files> -o test-fpga`.
   - The HLS compiler creates a `.prj` directory (for example, `test-fpga.prj`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

```
$ make test-fpga
i++ MGS.cpp QRD_Testbench.cpp TestbenchHelpers.cpp  -v --fpc --fp-relaxed -
march=Arria10 -o test-fpga
Target FPGA part name:   10AX115U1F45I1SG
Target FPGA family name: Arria 10
Target FPGA speed grade: -2
Analyzing MGS.cpp for testbench generation
Creating x86-64 testbench
Analyzing MGS.cpp for hardware generation
Analyzing QRD_Testbench.cpp for testbench generation
Creating x86-64 testbench
Analyzing QRD_Testbench.cpp for hardware generation
Analyzing TestbenchHelpers.cpp for testbench generation
Creating x86-64 testbench
Analyzing TestbenchHelpers.cpp for hardware generation
Optimizing component(s) and generating Verilog files
Generating cosimulation support
Generating simulation files for components: qrd
HLS simulation directory: /data/username/HLS_Trainings/examples/QRD/test-
fpga.prj/verification.
Linking x86 objects
+--------------------------------------+
| Run ./test-fpga to execute the test. |
+--------------------------------------+
```

2.2. Running a High Level Synthesis (HLS) Design Example (Windows)

To run an HLS design example on Windows systems:

1. Start a terminal session and initialize the Intel HLS Compiler environment.

   For instructions how to initialize the environment, see Initializing the Intel HLS Compiler Environment on page 10.

2. Navigate to the `<quartus_installdir>\hls\examples\<design_example_name>` directory, where <quartus_installdir> is the directory where you installed Intel Quartus Prime software.

For example, `C:\intelFPGA_pro\19.3`.

3. Run the `build.bat test-x86-64`. This command compiles the C++ source code to an x86-64 binary executable. Then, run the generated executable on your CPU.
   Expected outcome after you run the `build.bat test-x86-64` command:
   - The console displays the command it uses to generate the binary. For example, `i++ -march=x86-64 -o test-x86-64 <source_files>`.
   - The HLS compiler creates an executable file (for example, `test-x86-64.exe`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

```
C:\intelFPGA_pro\19.3\hls\examples\QRD>build.bat test-x86-64
i++ --fpc --fp-relaxed -march=x86-64 MGS.cpp QRD_Testbench.cpp
TestbenchHelpers.cpp -o test-x86-64.exe
Run test-x86-64.exe to execute the test.
```

4. Run the `build.bat test-fpga` command. The command compiles the C++ source code to a hardware executable and then runs a simulation of the generated HDL.
   Expected outcome after you run the `build.bat test-fpga` command:
   - The console displays the command it uses to generate the testbench binary and the contents of the project directory. For example, `i++ -march="<FPGA_family_or_part_number>" <source_files> -o test-fpga`.
   - The HLS compiler creates a `.prj` directory (for example, `test-fpga.prj`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

```
C:\intelFPGA_pro\19.3\hls\examples\QRD>build.bat test-fpga
i++ --fpc --fp-relaxed -march=Arria10 MGS.cpp QRD_Testbench.cpp
TestbenchHelpers.cpp -o test-fpga.exe
Run test-fpga.exe to execute the test.
```

3. Troubleshooting the Setup of the Intel HLS Compiler

This section provides information that can help you troubleshoot problems you might encounter when setting up the HLS compiler.

3.1. Intel HLS Compiler Licensing Issues

The Intel High Level Synthesis (HLS) Compiler is licensed as part of your Intel Quartus Prime license. However, the Intel HLS Compiler depends on ModelSim software. If you use a version of ModelSim software other than ModelSim - Intel FPGA Edition or ModelSim - Intel FPGA Starter Edition, ensure that your version of ModelSim software is licensed correctly.

In some cases, you might encounter problems with the licensing for ModelSim software.

3.1.1. ModelSim Licensing Error Messages

The HLS compiler issues error messages if it cannot locate the license for the installed version of ModelSim software.

If the HLS compiler fails to locate the ModelSim software license, it issues the following error message when you compile your design to the FPGA architecture:

```
$ i++ -march="<FPGA_family_or_part_number>" program.cpp
HLS Elaborate cosim testbench. FAILED.
See ./a.prj/a.log for details.
Error: Missing simulator license. Either:
1) Ensure you have a valid ModelSim license
2) Use the --simulator none flag to skip the verification flow
```

Common causes for these errors include:

- Missing, expired, or invalid licenses
- Incorrect license server name in the `license.dat` file
- Unspecified or incorrectly-specified license location

Note: The running speed of the HLS compiler might decrease if the compiler has to search the network for missing or corrupted licenses. If this problem occurs, correct the license file or license location accordingly.

3.1.2. LM_LICENSE_FILE Environment Variable

Intel and third-party software use the *LM_LICENSE_FILE* environment variable to specify the locations of license files or license servers. For example, both the Intel Quartus Prime software and the ModelSim software use the *LM_LICENSE_FILE* variable to specify the locations of their licenses.

Note: The time it takes for your development machine to communicate with the license server directly affects compilation time. If your *LM_LICENSE_FILE* environment variable setting includes paths to many license servers, or if the license server is hosted in a distant locale, you will notice a significant increase in compilation time.

On Linux or UNIX systems, insert a colon (:) after each license file or license server location that you append to the *LM_LICENSE_FILE* environment variable.

On Windows systems, insert a semicolon (;) after each license file or license server location that you append to the *LM_LICENSE FILE* environment variable.

Note: When modifying the *LM_LICENSE_FILE* setting to include the locations of your software licenses, do not remove any existing license locations appended to the variable.

3.1.2.1. ModelSim Software License-Specific Considerations

When setting up the ModelSim software license, you need to append the license location to the *LM_LICENSE_FILE* environment variable. However, you can also append the location of the ModelSim software license to the *MGLS_LICENSE_FILE* environment variable.

For Mentor Graphics applications, including the ModelSim software, you can specify the paths to license files and license servers in five different locations. If you specify paths to license files or license servers in multiple locations, the following search order is used to find the first valid path:

- *MGLS_LICENSE_FILE* environment variable you set in the user environment
- *MGLS_LICENSE_FILE* environment variable you set in the registry
- *LM_LICENSE_FILE* environment variable you set in the environment
- *LM_LICENSE_FILE* environment variable you set in the registry
- `<path to FLEXlm>\license.dat`, where *<path to FLEXlm>* is the default location of the FLEXlm license file.

When you install a Mentor Graphics product license on a computer, the *MGLS_LICENSE_FILE* environment variable settings take precedence over the *LM_LICENSE_FILE* environment variable settings. If you set both environment variables, set *LM_LICENSE_FILE* to point to the ModelSim license server and set *MGLS_LICENSE_FILE* to only point to the license server for other Mentor Graphics applications. If you only use the *MGLS_LICENSE_FILE* environment variable, ensure that the ModelSim license server and the license servers for other Mentor Graphics applications are on the same machine.

4. Document Revision History for Intel HLS Compiler Getting Started Guide

Document Version	Intel Quartus Prime Version	Changes
2019.09.30	19.3	• PRO Removed the following tutorials: — bank_bits — mm_slave — rom — struct_member_attributes • Added the following tutorials: — memory_bank_configuration — memory_geometry — memory_implementation — memory_merging — static_var_init — attributes_on_mm_slave_arg — exceptions — lsu_control — relax_reducation_dependency • PRO Added Microsoft Visual Studio 2017 Professional and Microsoft Visual Studio 2017 Community as supported C++ compilers. • PRO Removed Microsoft Visual Studio 2015 Professional and Microsoft Visual Studio 2015 Community as supported C++ compilers.
2019.07.01	19.2	• Updated Installing the Intel HLS Compiler on Linux Systems on page 6 to clarify the command to run to install the 32-bit libraries required by ModelSim - Intel FPGA Edition.
2019.04.01	19.1	• The Intel HLS Compiler can now be installed separately from Intel Quartus Prime. The following sections were added or updated: — Downloading the Intel HLS Compiler on page 5 — Installing the Intel HLS Compiler on Linux Systems on page 6 — Installing the Intel HLS Compiler on Microsoft* Windows* Systems on page 9 • The Intel HLS Compiler is now backwards-compatible with earlier versions of Intel Quartus Prime. See Intel HLS Compiler Backwards Compatibility on page 5 for details. • Updated Initializing the Intel HLS Compiler Environment on page 10 to include the requirement that you must initialize your Intel HLS Compiler environment on Windows operating systems from a Visual Studio x64 Native Tools command prompt. • Revised and updated the list of tutorials that are provided with the Intel HLS Compiler in High Level Synthesis (HLS) Design Examples and Tutorials on page 12.

continued...

Document Version	Intel Quartus Prime Version	Changes
2018.12.24	18.1	• PRO Added Microsoft Visual Studio 2015 Community to the list of supported C++ compilers on Microsoft Windows systems. • Added step to add path to Mentor Graphics ModelSim software provided with Intel Quartus Prime to operating system PATH environment variable in Installing the Intel HLS Compiler on Linux Systems on page 6 and Installing the Intel HLS Compiler on Microsoft* Windows* Systems on page 9.
2018.09.24	18.1	• PRO The Intel HLS Compiler has a new front end. For a summary of the changes introduced by this new front end, see *Improved Intel HLS Compiler Front End* in the Intel High Level Synthesis Compiler Version 18.1 Release Notes. • PRO The Intel HLS Compiler provided with Intel Quartus Prime Pro Edition has new prerequisites. Review Intel High Level Synthesis Compiler Prerequisites on page 3 to learn more. • PRO The installation instructions for Linux systems have changed. See Installing the Intel HLS Compiler on Linux Systems on page 6 for details. • PRO The `best_practices/parameter_aliasing` tutorial description in High Level Synthesis (HLS) Design Examples and Tutorials on page 12 changed to cover the `__restrict` keyword. The `restrict` keyword is no longer supported in the Intel HLS Compiler Pro Edition. • PRO Removed the `best_practices/integer_promotion` tutorial. Integer promotion is now done by default when use the Intel HLS Compiler Pro Edition.
2018.05.07	18.0	• Starting with Intel Quartus Prime Version 18.0, the features and devices supported by the Intel HLS Compiler depend on what edition of Intel Quartus Prime you have. Intel HLS Compiler publications now use icons to indicate content and features that apply only to a specific edition as follows: PRO Indicates that a feature or content applies only to the Intel HLS Compiler provided with Intel Quartus Prime Pro Edition. STD Indicates that a feature or content applies only to the Intel HLS Compiler provided with Intel Quartus Prime Standard Edition. • PRO Added the following tutorials to the list of tutorials in High Level Synthesis (HLS) Design Examples and Tutorials on page 12: — `interfaces/explicit_streams_packets_empty` — `interfaces/explicit_streams_ready_latency` — `best_practices/ac_datatypes` — `best_practices/loop_coalesce` — `best_practices/random_number_generator` • PRO Renamed the following tutorials to reflect some Intel Quartus Prime component name changes: — `usability/qsys_2xclock` is now `usability/platform_designer_2xclock` — `usability/qsys_stitching` is now `usability/platform_designer_stitching`

continued...

Document Version	Intel Quartus Prime Version	Changes
2017.12.22	17.1.1	• Added the `interfaces/overview` tutorial to the list of tutorials in High Level Synthesis (HLS) Design Examples and Tutorials on page 12.
2017.12.08	17.0	• Updated the Mentor Graphics ModelSim software requirements to include the required Red Hat development tools packages.
2017.11.06	17.0	• The Intel High Level Synthesis (HLS) Compiler is now part of Intel Quartus Prime Design Suite, resulting in the following changes: — Revised to document to reflect that you now get the Intel HLS Compiler by installing Intel Quartus Prime software. — Removed most licensing information. Licensing the Intel HLS Compiler is now covered by your Intel Quartus Prime Design Suite licensing. Some third-party software required by the HLS compiler might continue to require additional licensing. — Removed information about overriding compilers. — Revised prerequisites to reflect only additional prerequisites required by the HLS compiler. — Revised path information to reflect the new file system locations of the Intel HLS Compiler files. • Renamed the following tutorials: — The `explicit_streams` tutorial is now called `explicit_streams_buffer`. — The `explicit_streams_2` tutorial is now called `explicit_streams_packets_ready_valid`.
2017.06.23	—	• Minor changes and corrections.
2017.06.09	—	• Updated High Level Synthesis (HLS) Design Examples and Tutorials on page 12 with information about new examples. • Revised Overriding the Default GCC Compiler for Intel HLS Compiler.
2017.03.14	—	• Removed bit operations (3DES) from list of supplied design examples.
2017.03.01	—	• Added installation of required packages and libraries needed for Linux.
2017.02.03	—	• Changed success message in Quick Start sections to PASSED. • Added *HLS Design Examples and Tutorials* section. • Moved *Running an HLS Design Example on Linux* and *Running an HLS Design Example on Windows* to *HLS Design Examples and Tutorials*.
2016.11.30	—	• In *HLS Compiler Prerequisites*, updated software requirements to note that the HLS compiler supports all ModelSim software editions that the Intel Quartus Prime software supports. • In *HLS Compiler Quick Start*, added a note that you must run the `init_hls` script each time you start a shell or terminal to develop your design. • In HLS Compiler Quick Start, separated the Linux and Windows instructions. • In *Running an HLS Design Example*, separated the Linux and Windows instructions. For Linux, run the `make` command; for Windows, run the `build.bat` command. • Changed the `test_x86-64` command option to `test-x86-64`. • Changed the `test_fpga` command option to `test-fpga`. • Removed the instruction to run the `make test_qii` command for Linux and the `build.bat test_qii` command for Windows because it is no longer necessary. • In *HLS Licensing Error Messages*, updated the error message you will see if the HLS compiler fails to locate the ModelSim software license.
2016.09.12	—	• Initial release.