Intel® High Level Synthesis Compiler
Standard Edition

Getting Started Guide

Updated for Intel® Quartus® Prime Design Suite: 19.1

The Intel® High Level Synthesis (HLS) Compiler is part of the Intel Quartus® Prime Standard Edition 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 Standard Edition Getting Started Guide describes the procedures to set up the Intel HLS Compiler and to run an HLS design example.

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:

- **Windows**: C:\intelFPGA_standard\19.1
- **Linux**: /home/<username>/intelFPGA_standard/19.1

**About the Intel HLS Compiler Standard Edition Documentation Library**

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

<table>
<thead>
<tr>
<th>Title and Description</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Notes</td>
<td>Link</td>
</tr>
<tr>
<td>Getting Started Guide</td>
<td>Link</td>
</tr>
<tr>
<td>User Guide</td>
<td>Link</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.*
1.1. Intel High Level Synthesis Compiler Standard Edition

Prerequisites

The Intel HLS Compiler Standard Edition is part of the Intel Quartus Prime Standard Edition Design Suite. It is installed as part of your Intel Quartus Prime software installation, but it requires additional software to use.

For detailed instructions about installing Intel Quartus Prime Standard 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**

For Linux, install GCC 4.4.7 including the GNU C++ library.

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

**Important:** The Intel HLS Compiler Standard Edition software does not support versions of the GCC compiler other than GCC 4.4.7.

For Windows, install Microsoft Visual Studio 2010 Professional.

**Important:** The Intel HLS Compiler Standard Edition software does not support versions of Microsoft Visual Studio other than Microsoft Visual Studio 2010 Professional.

**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 Standard Edition on Linux Systems on page 5.

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 Standard Edition.

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

1.2. Installing the Intel HLS Compiler Standard Edition 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 Standard Edition on Linux Systems:

1. Confirm that your operating system version is supported by the Intel HLS Compiler Standard Edition:
   - Red Hat Enterprise Linux 6.x, or a community equivalent
2. Install the complete Intel Quartus Prime Standard Edition installation package. You can download the installation package from the the Combined Files tab of the Quartus Prime download page at the Download Center for FPGAs:
   For detailed instructions about installing Intel Quartus Prime software, including system requirements, prerequisites, and licensing requirements, refer to Intel FPGA Software Installation and Licensing
3. Update your Linux repositories with the following command:
   `sudo yum update`
4. Install GCC 4.4.7 with the following command:
   `sudo yum groupinstall "Development Tools"`
5. 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:

   ```bash
   $ 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
   ```

6. 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:

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

7. 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:

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

After completing these steps, the Intel HLS Compiler Standard Edition 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 Standard Edition Environment on page 7.

### 1.3. Installing the Intel HLS Compiler Standard Edition on Microsoft* Windows* Systems

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

1. Confirm that your operating system version is supported by the Intel HLS Compiler (Microsoft* Windows* 8.1 or 10).


   You can download the installation packaged from the `Combined Files` tab of the Quartus Prime download page at the Download Center for FPGAs:

   ```
   ```

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

4. 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:

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

5. 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:

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

After completing these steps, the Intel HLS Compiler Standard Edition 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 Standard Edition Environment on page 7.

### 1.4. Initializing the Intel HLS Compiler Standard Edition 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_standard/19.1/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.

- On Windows systems, initialize your environment as follows:
  a. Start a Visual Studio x64 Win64 Command Prompt session.
     For example, C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\amd64\vcvars64.bat.
  b. In your x64 Win64 Command Prompt session, change directories to the hls directory in your Intel Quartus Prime installation directory.
     For example, C:\intelFPGA_standard\19.1\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.

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 Standard Edition 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_standard/19.1` or `C:\intelFPGA_standard\19.1`.

For instructions on running the examples, see the following sections:
- Running a High Level Synthesis (HLS) Design Example (Linux) on page 12
- Running a High Level Synthesis (HLS) Design Example (Windows) on page 13

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

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 3. Arbitrary precision datatypes design tutorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ac_fixed_constructor</code></td>
<td>Demonstrates the use of the <code>ac_fixed</code> constructor where you can get a better QoR by using minor variations in coding style.</td>
</tr>
<tr>
<td><code>ac_fixed_math_library</code></td>
<td>Demonstrates the use of the Intel HLS Compiler <code>ac_fixed_math</code> fixed point math library functions.</td>
</tr>
<tr>
<td><code>ac_int_basic_ops</code></td>
<td>Demonstrates the operators available for the <code>ac_int</code> class.</td>
</tr>
<tr>
<td><code>ac_int_overflow</code></td>
<td>Demonstrates the usage of the <code>DEBUG_AC_INT_WARNING</code> and <code>DEBUG_AC_INT_ERROR</code> keywords to help detect overflow during emulation runtime.</td>
</tr>
</tbody>
</table>

You can find these tutorials in the following location on your Intel Quartus Prime system:

```
<quartus_installdir>/hls/examples/tutorials/ac_datatypes
```

---

### Table 4. Component memories design tutorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bank_bits</code></td>
<td>Demonstrates how to control component internal memory architecture for parallel memory access by enforcing which address bits are used for banking.</td>
</tr>
<tr>
<td><code>depth_wise_merge</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>static_var_init</code></td>
<td>Demonstrates the <code>hls_init_on_power</code> and <code>hls_init_on_reset</code> flags for static variables and their impact on area and latency.</td>
</tr>
<tr>
<td><code>width_wise_merge</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

You can find these tutorials in the following location on your Intel Quartus Prime system:

```
<quartus_installdir>/hls/examples/tutorials/component_memories
```

---

### Table 5. Interfaces design tutorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>overview</code></td>
<td>Demonstrates the effects on quality-of-results (QoR) of choosing different component interfaces even when the component algorithm remains the same.</td>
</tr>
<tr>
<td><code>explicit_streams_buffer</code></td>
<td>Demonstrates how to use explicit <code>stream_in</code> and <code>stream_out</code> interfaces in the component and testbench.</td>
</tr>
<tr>
<td><code>explicit_streams_packets_ready_valid</code></td>
<td>Demonstrates how to use the <code>usesPackets</code>, <code>usesValid</code>, and <code>usesReady</code> stream template parameters.</td>
</tr>
<tr>
<td><code>mm_master_testbench_operators</code></td>
<td>Demonstrates how to invoke a component at different indices of an Avalon Memory Mapped (MM) Master (<code>mm_master</code> class) interface.</td>
</tr>
<tr>
<td><code>mm_slaves</code></td>
<td>Demonstrates how to create Avalon-MM Slave interfaces (slave registers and slave memories).</td>
</tr>
</tbody>
</table>

You can find these tutorials in the following location on your Intel Quartus Prime system:

```
<quartus_installdir>/hls/examples/tutorials/interfaces
```
### Table 6. Best practices design tutorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple_stream_call_sites</td>
<td>Demonstrates the benefits of using multiple stream call sites.</td>
</tr>
<tr>
<td>pointer_mm_master</td>
<td>Demonstrates how to create Avalon-MM Master interfaces and control their parameters.</td>
</tr>
<tr>
<td>stable_arguments</td>
<td>Demonstrates how to use the <code>stable</code> attribute for unchanging arguments to improve resource utilization.</td>
</tr>
</tbody>
</table>

You can find these tutorials in the following location on your Intel Quartus Prime system:

```<quartus_installdir>/hls/examples/tutorials/best_practices```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const_global</td>
<td>Demonstrates the performance and resource utilization improvements of using <code>const</code> qualified global variables.</td>
</tr>
<tr>
<td>floating_point_ops</td>
<td>Demonstrates the impact of <code>--fpc</code> and <code>--fp-relaxed</code> flags in i++ on floating point operations using a 32-tap finite impulse response (FIR) filter design that is optimized for throughput.</td>
</tr>
<tr>
<td>integer_promotion</td>
<td>Demonstrates how integer promotion rules can influence the behavior of a C or C++ program.</td>
</tr>
<tr>
<td>loop_memory_dependency</td>
<td>Demonstrates breaking loop carried dependencies using the <code>ivdep</code> pragma.</td>
</tr>
<tr>
<td>parameter_aliasing</td>
<td>Demonstrates the use of the <code>restrict</code> keyword on component arguments.</td>
</tr>
<tr>
<td>resource_sharing_filter</td>
<td>Demonstrates an optimized-for-area variant of a 32-tap finite impulse response (FIR) filter design.</td>
</tr>
<tr>
<td>shift_register</td>
<td>Demonstrates the recommended coding style for implementing shift registers.</td>
</tr>
<tr>
<td>single_vs_double_precision_math</td>
<td>Demonstrates the effect of using single precision literals and functions instead of double precision literals and functions.</td>
</tr>
<tr>
<td>struct_interface</td>
<td>Demonstrates how to use <code>ac_int</code> to implement interfaces with no padding bits.</td>
</tr>
<tr>
<td>swap_vs_copy</td>
<td>Demonstrates the impact of using deep copying with registers on the performance and resource utilization of a component design.</td>
</tr>
</tbody>
</table>

### Table 7. Usability design tutorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiler_interoperability</td>
<td>(Linux only) Demonstrates how to use testbench code compiled with GCC along with code compiled by the i++ command.</td>
</tr>
<tr>
<td>enqueue_call</td>
<td>Demonstrates how to run components asynchronously and exercise their pipeline performance in the test bench using enqueue functionality.</td>
</tr>
<tr>
<td>qsys_2xclock</td>
<td>Demonstrates the recommended clock and reset generation for a component with a <code>clock2x</code> input.</td>
</tr>
<tr>
<td>qsys_stitching</td>
<td>Demonstrates how to combine multiple components to function as a single cohesive design.</td>
</tr>
</tbody>
</table>
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 Standard Edition Environment on page 7.

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_standard/19.1`.

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 QRD_Testbench.cpp for testbench generation
Creating x86-64 testbench
Analyzing QRD_Testbench.cpp for hardware generation
```

*Intel High Level Synthesis Compiler Standard Edition: Getting Started Guide*
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 Standard Edition Environment on page 7.

2. Navigate to the `<quartus_installdir>/hls/examples` directory, where `<quartus_installdir>` is the directory where you installed Intel Quartus Prime software.
   For example, C:\intelFPGA_standard\19.1.

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_standard\19.1\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.

```bash
C:\intelFPGA_standard\19.1\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.
# A. Intel HLS Compiler Standard Edition Getting Started Guide Archives

<table>
<thead>
<tr>
<th>Intel HLS Compiler Version</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.1</td>
<td>Intel HLS Compiler Standard Edition Getting Started Guide</td>
</tr>
<tr>
<td>18.1.1</td>
<td>Intel HLS Compiler Getting Started Guide</td>
</tr>
<tr>
<td>18.1</td>
<td>Intel HLS Compiler Getting Started Guide</td>
</tr>
<tr>
<td>18.0</td>
<td>Intel HLS Compiler Getting Started Guide</td>
</tr>
<tr>
<td>17.1.1</td>
<td>Intel HLS Compiler Getting Started Guide</td>
</tr>
<tr>
<td>17.1</td>
<td>Intel HLS Compiler Getting Started Guide</td>
</tr>
</tbody>
</table>
### B. Document Revision History for Intel HLS Compiler Standard Edition Getting Started Guide

|------------------|--------------------------------------------|---------|

### Document Revision History for Intel HLS Compiler Getting Started Guide


<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2019.09.30       | 19.3                        | • 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_reduction_dependency  
  • Added Microsoft Visual Studio 2017 Professional and Microsoft Visual Studio 2017 Community as supported C++ compilers.  
  • Removed Microsoft Visual Studio 2015 Professional and Microsoft Visual Studio 2015 Community as supported C++ compilers. |
<p>| 2019.07.01       | 19.2                        | • Updated Installing the Intel HLS Compiler on Linux Systems to clarify the command to run to install the 32-bit libraries required by ModelSim - Intel FPGA Edition. |</p>
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2019.04.01       | 19.1                        | • **PRO** 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  
  — Installing the Intel HLS Compiler on Linux Systems  
  — Installing the Intel HLS Compiler on Microsoft* Windows* Systems  
• **PRO** The Intel HLS Compiler is now backwards-compatible with earlier versions of Intel Quartus Prime. See Backwards Compatibility for details.  
• Updated Initializing the Intel HLS Compiler Environment 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. |
| 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 and Installing the Intel HLS Compiler on Microsoft* Windows* Systems. |
| 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 to learn more.  
• **PRO** The installation instructions for Linux systems have changed. See Installing the Intel HLS Compiler on Linux Systems for details.  
• **PRO** The `best_practices/parameter_aliasing` tutorial description in High Level Synthesis (HLS) Design Examples and Tutorials 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. |
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.05.07</td>
<td>18.0</td>
<td>• 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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRO Indicates that a feature or content applies only to the Intel HLS Compiler provided with Intel Quartus Prime Pro Edition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STD Indicates that a feature or content applies only to the Intel HLS Compiler provided with Intel Quartus Prime Standard Edition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRO Added the following tutorials to the list of tutorials in High Level Synthesis (HLS) Design Examples and Tutorials:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— interfaces/explicit_streams_packets_empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— interfaces/explicit_streams_ready_latency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— best_practices/ac_datatypes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— best_practices/loop_coalesce</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— best_practices/random_number_generator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PRO Renamed the following tutorials to reflect some Intel Quartus Prime component name changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— usability/qsys_2xclock is now usability/platform_designer_2xclock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— usability/qsys_stitching is now usability/platform_designer_stitching</td>
</tr>
<tr>
<td>2017.12.22</td>
<td>17.1.1</td>
<td>• Added the interfaces/overview tutorial to the list of tutorials in High Level Synthesis (HLS) Design Examples and Tutorials.</td>
</tr>
<tr>
<td>2017.12.08</td>
<td>17.0</td>
<td>• Updated the Mentor Graphics ModelSim software requirements to include the required Red Hat development tools packages.</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>17.0</td>
<td>• The Intel High Level Synthesis (HLS) Compiler is now part of Intel Quartus Prime Design Suite, resulting in the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Revised to document to reflect that you now get the Intel HLS Compiler by installing Intel Quartus Prime software.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Removed information about overriding compilers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Revised prerequisites to reflect only additional prerequisites required by the HLS compiler.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Revised path information to reflect the new file system locations of the Intel HLS Compiler files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Renamed the following tutorials:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— The explicit_streams tutorial is now called explicit_streams_buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— The explicit_streams_2 tutorial is now called explicit_streams_packets_ready_valid.</td>
</tr>
<tr>
<td>2017.06.23</td>
<td>—</td>
<td>• Minor changes and corrections.</td>
</tr>
<tr>
<td>2017.06.09</td>
<td>—</td>
<td>• Updated High Level Synthesis (HLS) Design Examples and Tutorials with information about new examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Revised Overriding the Default GCC Compiler for Intel HLS Compiler.</td>
</tr>
<tr>
<td>2017.03.14</td>
<td>—</td>
<td>• Removed bit operations (3DES) from list of supplied design examples.</td>
</tr>
<tr>
<td>Document Version</td>
<td>Intel Quartus Prime Version</td>
<td>Changes</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>2017.03.01</td>
<td>—</td>
<td>• Added installation of required packages and libraries needed for Linux.</td>
</tr>
</tbody>
</table>
| 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. |