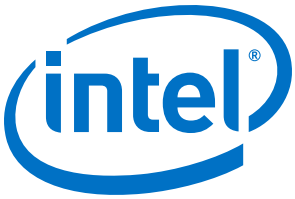


# AN 834: Developing for the Intel® HLS Compiler with an IDE

Updated for Intel® Quartus® Prime Design Suite: **19.2**



**AN-834 | 2019.07.19**
Latest document on the web: **PDF** | **HTML**

# Contents

# 1. Developing for the Intel® HLS Compiler with an Eclipse* IDE

Use your Eclipse* IDE to develop for the Intel® HLS Compiler and take advantage of code highlighting and other code editing features. Also, if you are using Linux, you can configure your Eclipse IDE to run the Intel HLS Compiler from the Eclipse GUI instead of needing to switch to a command line interface whenever you want to compile your component.

## 1.1. Adding Intel HLS Compiler Header Files to the Eclipse CDT Indexer

Add the Intel HLS Compiler header files to the Eclipse C/C++ Development Tooling (CDT) indexer to take advantage of features of the Eclipse code editor like code refactoring, variable navigation, and code completion.

Before you add the Intel HLS Compiler header files to the Eclipse CDT indexer, ensure that you have completed the following prerequisites:

- Install the Intel HLS Compiler

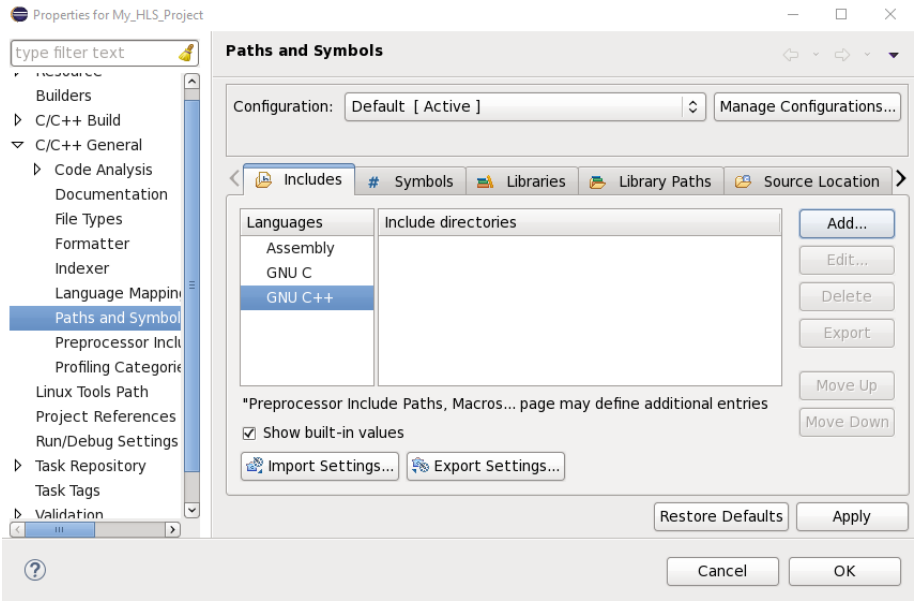
  For information about installing the Intel HLS Compiler, see Intel High Level Synthesis Compiler Getting Started Guide.
- Install an Eclipse IDE and the Eclipse C/C++ Development Tooling (CDT)
- Create a new C++ project in Eclipse (**File ➤ New ➤ C++ Project**) with the following properties:
  - **Project type**: **Makefile project ➤ Hello World C++ Makefile Project**
  - **Toolchains**: **Linux GCC**

  When you create a project, you might receive an additional prompt, choose **C++ managed Build** if prompted.

Adding the Intel HLS Compiler libraries to the Eclipse build path to your project enables the parser to highlight your code correctly in the code editor. Repeat these steps for each Eclipse project that want to use to develop Intel HLS Compiler components.

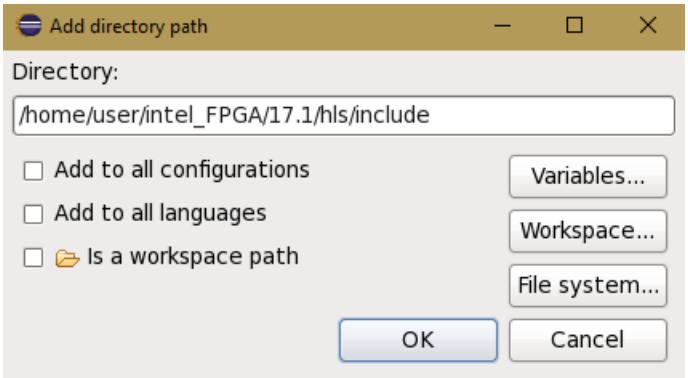
To add the Intel HLS Compiler libraries to your Eclipse project:

1. Launch your Eclipse IDE.
2. In the **Project Explorer** view, right-click your project and select **Properties**.
3. In the **Properties** window for you project, go to **C/C++ General ➤ Paths and Symbols**.
4. In the **Paths and Symbols** pane, go to the **Includes** tab, select **GNU C++**, and click **Add**.



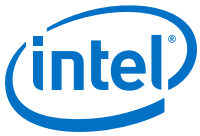

5. In the **Directory** field of the **Add directory path** dialog box, enter the full path to the `include` folder of your Intel HLS Compiler installation and click **OK**, then click **Apply** in the **Properties** window.

   The default location for the Intel HLS Compiler `include` folder is `<quartus_installdir>/hls/include`, where *quartus_installdir* is your Intel Quartus® Prime installation directory (for example, `/home/<username>/intelFPGA_pro/19.2` or `C:\intelFPGA_pro\19.2`).




6. Rebuild the index for the Eclipse Indexer by right-clicking your project and clicking **Index ➤ Rebuild**.

After you complete these steps, you can edit code using your Eclipse IDE, and you can take advantage of features like code refactoring, code navigation, and code completion for Intel HLS Compiler elements.



To compile your code, open a terminal window, initialize the Intel HLS Compiler environment in that window, and run the `i++` command.

## 1.2. Configuring Eclipse to Run the Intel HLS Compiler

If you are using Linux, you can modify the Eclipse-generated makefile for your Eclipse project so that you can launch the Intel HLS Compiler from the Eclipse **Run** and **Debug** buttons.

Before you modify the Eclipse-generated makefile to configure Eclipse to launch the Intel HLS Compiler, ensure that you have completed the following prerequisites:

- Install the Intel HLS Compiler

  For information about installing the Intel HLS Compiler, see Intel High Level Synthesis Compiler Getting Started Guide.
- Install an Eclipse IDE and the Eclipse C/C++ Development Tooling (CDT)

  These instructions were tested on Eclipse Neon (4.6) and Eclipse Helios (3.6).
- Create a new C++ project in Eclipse (**File ➤ New ➤ C++ Project**) with the following properties:
  - **Project type**: **Makefile project ➤ Hello World C++ Makefile Project**
  - **Toolchains**: **Linux GCC**

  When you create a project, you might receive an additional prompt, choose **C++ managed Build** if prompted.
- Add Intel HLS Compiler Header Files to the Eclipse CDT Indexer

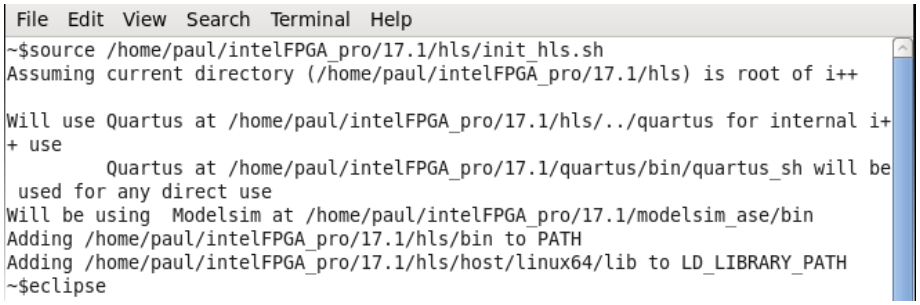
If you are using Linux, modifying the Eclipse-generated makefile for your Eclipse project lets you launch the Intel HLS Compiler from the Eclipse **Run** and **Debug** buttons. You must repeat these steps for each Eclipse project that you want to launch the Intel HLS Compiler from.

1. Open a terminal window and initialize the Intel HLS Compiler environment by running the following command:

   `source <quartus_installdir>/hls/hls_init.sh`

   Where *<quartus_installdir>* is your Intel Quartus Prime installation directory for example, `/home/<username>/intelFPGA_pro/19.2` or `C:\intelFPGA_pro\19.2`).

   Keep this terminal window open for the next step.
2. Launch your Eclipse IDE from the command-line prompt in the open terminal window.



3. Open the Eclipse-generated makefile for your project. The makefile is in the folder you entered when you created your C++ project.
4. Replace the content of the makefile with the following code:

```
## (c) 2017 Intel Corp.
## THIS MAKEFILE IS PROVIDED AS-IS AND NO GUARANTEES ARE PROVIDED
WHATSOEVER.

SRC     :=  <YOUR SOURCEFILES HERE>

FPGA    :=  "<YOUR TARGET INTEL FPGA DEVICE FAMILY>"

TARGETS :=  target-x86 \
            target-fpga\
            target-fpga-nosim\
            target-fpga-qii

RM      :=  rm -rf

.DEFAULT_GOAL := target-x86

.PHONY: run
run: $(TARGETS)
    @$(foreach t,$(TARGETS),echo time ./$(t); time ./$(t); echo "";)

.PHONY: all
all: $(TARGETS)

.PHONY: clean
clean:
    -$(RM) $(TARGETS) $(foreach t,$(TARGETS),./$(t).prj ./$(t)) ./
time_fpga.out transcript *.tmp *.o

$(TARGETS) : % : $(SRC)
    time $(CXX) $^ $(CXXFLAGS) -o $@

target-x86:        CXX = i++
target-x86:        CXXFLAGS = -march=x86-64 -O0 -v

target-fpga:       CXX = i++
target-fpga:       CXXFLAGS = -march=$(FPGA) -v --time time_fpga.out -ghdl

target-fpga-nosim: CXX = i++
target-fpga-nosim: CXXFLAGS = -march=$(FPGA) -v --time time_fpga.out -ghdl
--simulator none

target-fpga-qii:   CXX = i++
target-fpga-qii:   CXXFLAGS = -march=$(FPGA) -v --time time_fpga.out -ghdl
--quartus-compile
```

Where the variable parts of this makefile are defined as follows:

| | |
|---|---|
| *<YOUR SOURCEFILES HERE>* | The names of the `.cpp` files that contain your components and testbench. |
| *<YOUR TARGET INTEL FPGA DEVICE FAMILY>* | The device target FPGA family supported by the Intel HLS Compiler. For example, `Arria10` or `Cyclone10GX`. For a list of supported FPGA device families, see " Intel HLS Compiler Command Options" in *Intel High Level Synthesis Compiler Reference Manual*. |

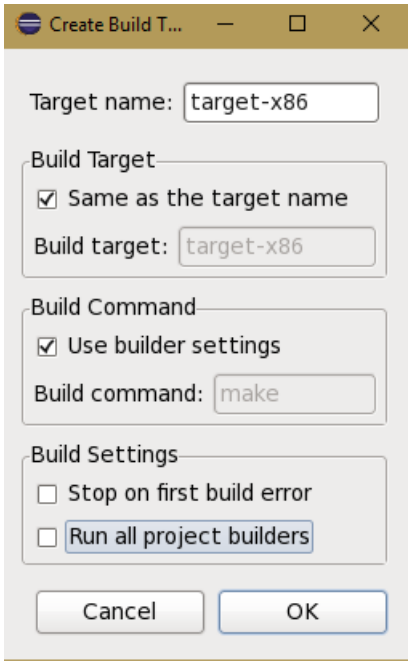
You can configure your `i++` build flags for a target by changing the CXXFLAGS arguments for that target.

5. Add code to your project.

   You can use the following simple addition component code to test your configuration.

```
#include "HLS/stdio.h"
#include "HLS/hls.h"

component int add(int a, int b)
{
    return a + b;
}

int main(void) {
    printf("Hello.");

    int a = 3;
    int b = 4;
    int c = add(a, b);

    printf("Add: %d + %d = %d (should be %d)", a, b, c, (a + b));

    return 0;
}
```

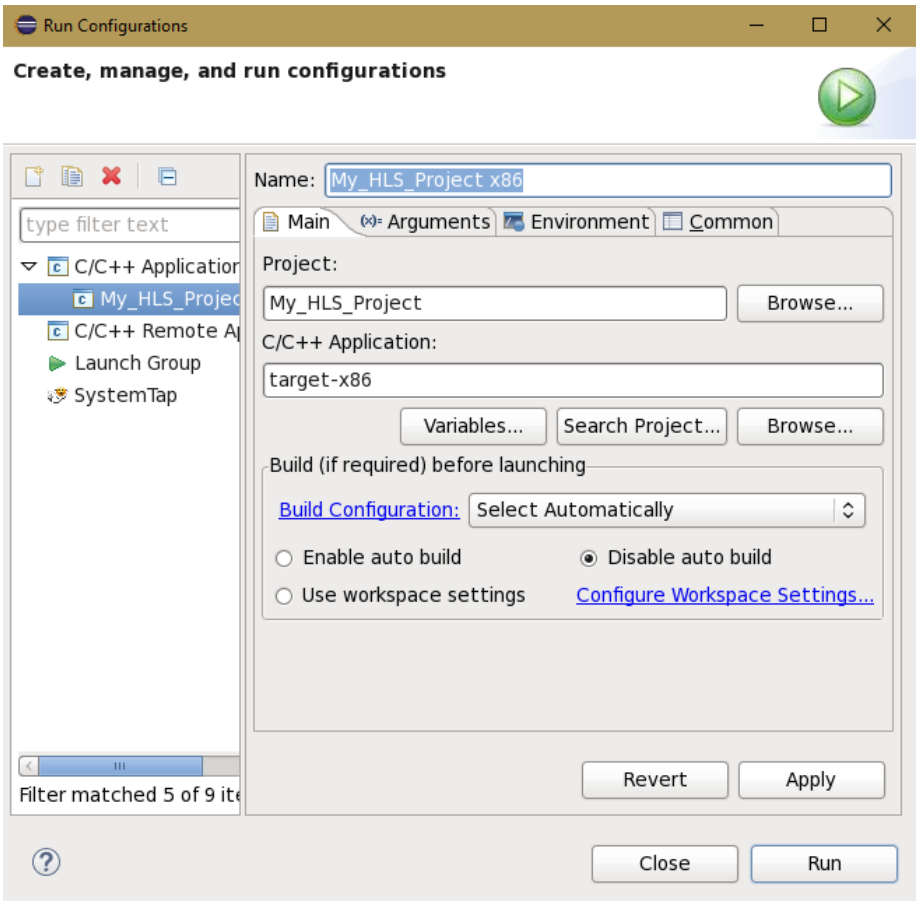
6. Enable click-to-run by linking the build targets in the makefile:
   a. In the **Build Targets** view (the view might be called **Make Targets** in some versions of Eclipse), right-click your project and select **New**.
   b. In the **Target name** field of **Create Build Target** (or **Create Make Target**) window, enter one of the build target names from the makefile (`target-x86`, `target-fpga`, `target-fpga-nosim`, `target-fpga-qii`), and click **OK**.



c. Repeat these steps to add each run target.

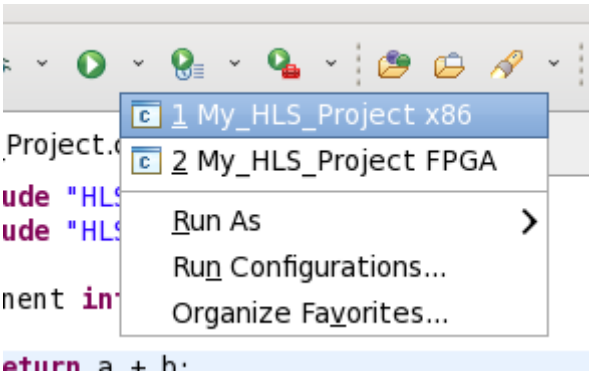
When this step is complete, you can double-click a build target in the **Build Targets** view to launch a build.

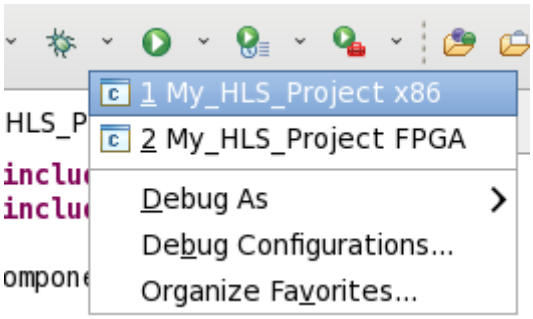
7. If you have built a binary target, you add it to a run configuration in Eclipse so that you can run the configuration from the **Run As** and **Debug** buttons. To attach a binary to run configuration:
   a. Right-click your project, and select **Run As ➤ Run Configurations**, depending on your version of Eclipse.
   b. In the **Run Configurations** windows, right-click **C/C++ Applications** and click **New**.
   c. Click **Search Project** to find and select the project binary files that you want to assign to the run configuration.
   d. In the **Name** field, give this run configuration a meaningful name so that you know which of your project binaries the run configuration corresponds to.
   e. Select **Disable auto build**.
   f. Click **Apply**, then click **Close**.



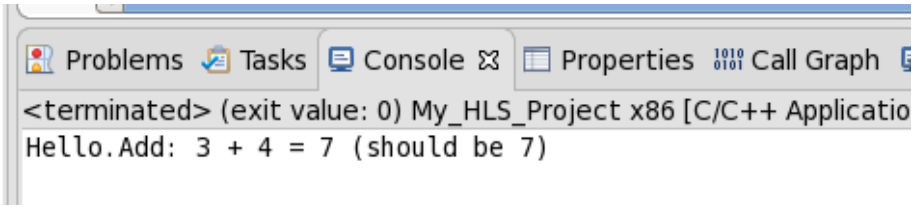
After you complete this step, you can run the configuration from the **Run** button and debug the executable from the **Debug** button.

*Restriction:* You an debug only code compiled for the `target-x86` build target. For details, see Intel High Level Synthesis Compiler Reference Manual.




You can view output in the **Console** view:




*Remember:* To launch the Intel HLS Compiler from within Eclipse, you must always initialize the Intel HLS Compiler from a command line and launch your Eclipse IDE from that same command line session. That is, repeat step 1 on page 6 and step 2 on page 6 every time that you want to start your Eclipse IDE.

# 2. Developing for the Intel HLS Compiler with Microsoft* Visual Studio

Use Microsoft* Visual Studio to develop for the Intel HLS Compiler and take advantage of code highlighting and other code editing features.

## 2.1. Adding Intel HLS Compiler Header Files to the Microsoft* Visual Studio* Indexer

Add the Intel HLS Compiler header files to the Microsoft* Visual Studio* indexer to take advantage of features of Visual Studio code editor features like code refactoring, code navigation, and code completion.

Before you add the Intel HLS Compiler header files to the Microsoft Visual Studio Indexer, ensure that you have completed the following prerequisites:

- Install the Intel HLS Compiler

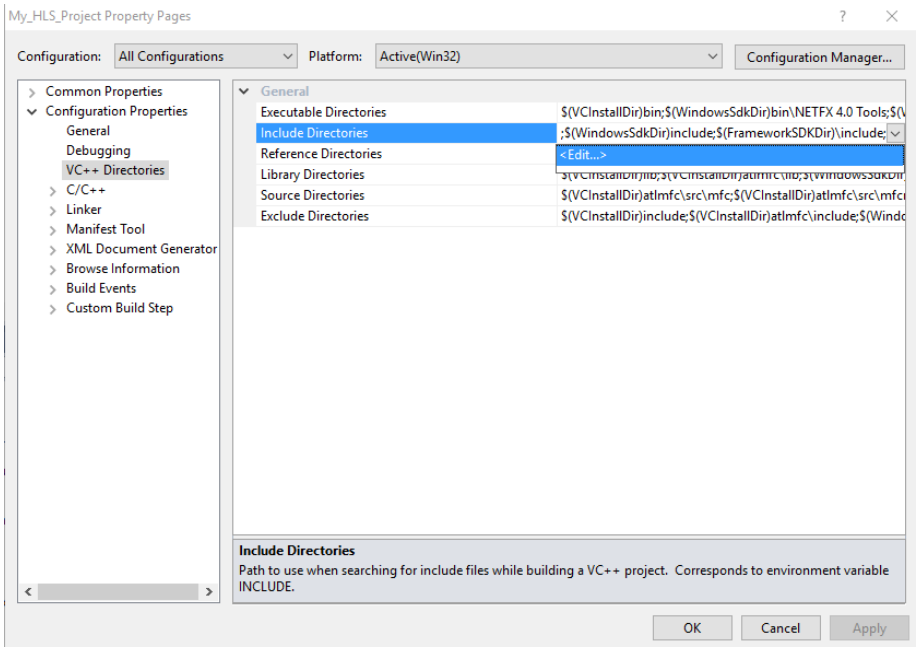
  For information about installing the Intel HLS Compiler, see Intel High Level Synthesis Compiler Getting Started Guide.
- Install a version of Microsoft Visual Studio that is supported by the Intel HLS Compiler

  For supported versions of Microsoft Visual Studio, see "*Intel High Level Synthesis Compiler Prerequisites*" in Intel High Level Synthesis Compiler Getting Started Guide.
- Create a new empty Visual C++ project in Visual Studio (**File ➤ New ➤ Project ➤ Visual C++ ➤ General ➤ Empty Project**).

Adding the Intel HLS Compiler libraries to the Visual Studio build path to your project enables the parser to highlight your code correctly in the code editor. Repeat these steps for each Visual Studio project that you want to use to develop Intel HLS Compiler components.

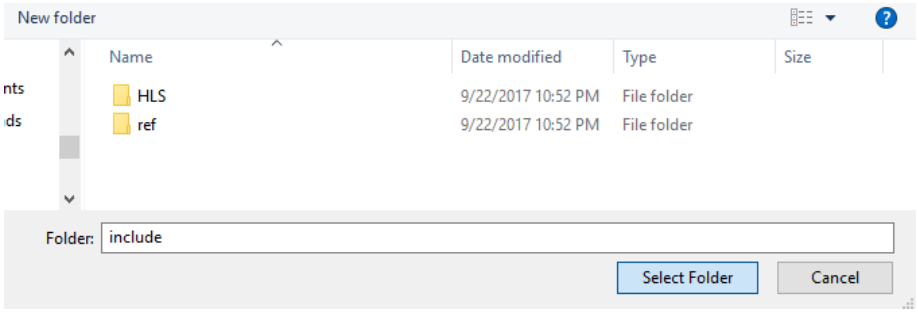
To add the Intel HLS Compiler libraries to your Visual Studio project build path:

1. Launch Microsoft Visual Studio.
2. Right-click your project and select **Properties**.
3. In the **Property Pages** window for your project, go to **Configuration Properties ➤ VC++ Directories**.
4. From the **Configuration** menu, select **All Configurations**.
5. Under **General**, edit the **Include Directories** entry by clicking the entry content and selecting **Edit**.




6. In the **Include Directories** window, click **New Line** ( ) and find the location of the Intel HLS Compiler `include` folder in your Intel Quartus Prime installation folder. Click **Select Folder**, then click **OK**.

   The default location for the Intel HLS Compiler `include` folder is `<quartus_installdir>/hls/include`, where *quartus_installdir* is your Intel Quartus Prime installation directory (for example, `C:\intelFPGA_pro\19.2\`).

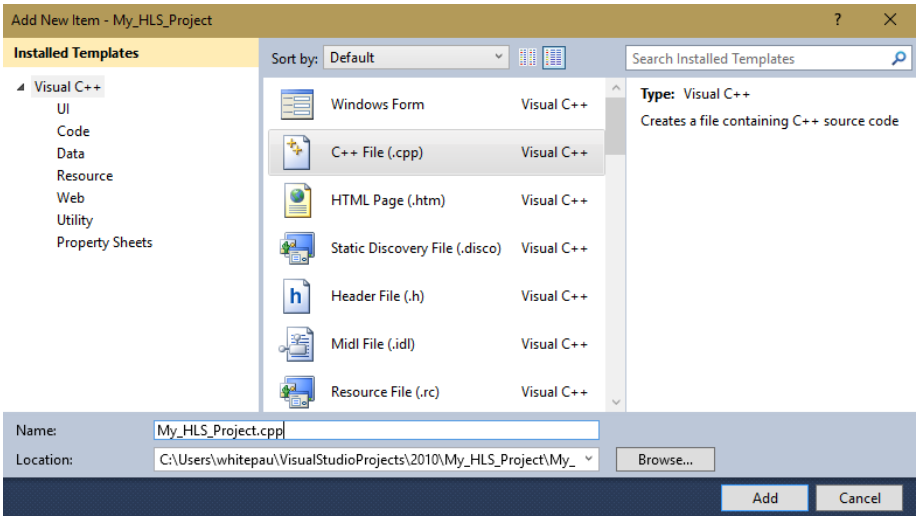



After you complete these steps, you can edit code using your Microsoft Visual Studio IDE, and you can take advantage of features like code refactoring, variable navigation, and code completion for Intel HLS Compiler elements.

To compile your component with the Intel HLS Compiler, open a command prompt session, initialize the Intel HLS Compiler environment, and run the `i++` command. You cannot use the Visual Studio to call the `i++` command.

You can test code highlighting from the Intel HLS Compiler header files with the following steps:

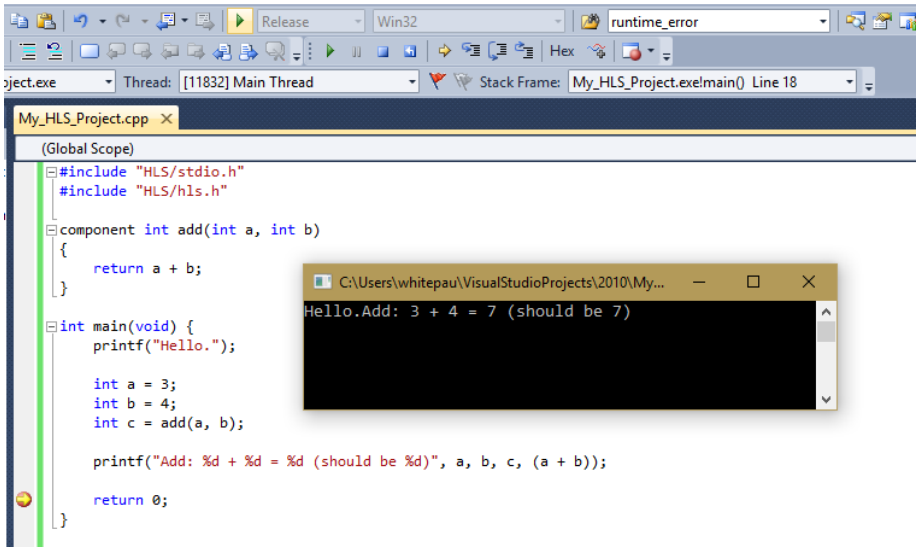
1. Create a new C++ source file by right-clicking your project and selecting **Add ➤ New Item**.
2. In the **Add New Item** window, choose **C++ File (.cpp)**, give the item a meaningful name (for example, `My_HLS_Project.cpp`), and click **Add**.




3. Paste the following code into your new `.cpp` file:

```
#include "HLS/stdio.h"
#include "HLS/hls.h"

component int add(int a, int b)
{
    return a + b;
}

int main(void) {
    printf("Hello.");

    int a = 3;
    int b = 4;
    int c = add(a, b);

    printf("Add: %d + %d = %d (should be %d)", a, b, c, (a + b));

    return 0;
}
```

4. To build and execute your component with Microsoft Visual Studio (not the Intel HLS Compiler), click the **Run** button ( ). The component runs in a console window. To prevent the console window from closing when the component finishes running, place a breakpoint at the end of the `main()` function.


Release
Win32
runtime_error
Hex
Thread: [11832] Main Thread
Stack Frame: My_HLS_Project.exe!main() Line 18
My_HLS_Project.cpp
(Global Scope)
#include "HLS/stdio.h"
#include "HLS/hls.h"
component int add(int a, int b)
{
return a + b;
}
int main(void) {
printf("Hello.");
int a = 3;
int b = 4;
int c = add(a, b);
printf("Add: %d + %d = %d (should be %d)", a, b, c, (a + b));
return 0;
}
C:\Users\whitepau\VisualStudioProjects\2010\My...
Hello.Add: 3 + 4 = 7 (should be 7)

# 3. Document Revision History

**Table 1. Document Revision History for *AN 834: Developing for the Intel HLS Compiler with an IDE***

| Date | Version | Changes |
|---|---|---|
| July 2019 | 2019.09.19 | • Revised references to supported versions of Microsoft Visual Studio<br>• Updated example paths to a more recent version<br>• Fixed broken links to other Intel HLS Compiler documentation |
| December 2017 | 2017.12.01 | • Initial release |