# FPGA Development with Visual Studio Code*

Last updated: **2025-05-05**

You can integrate the Intel® oneAPI Base toolkit with Visual Studio Code* (VS Code) to support a seamless software development environment.

## Prerequisites

Download and install the following software:

- Intel® oneAPI Base Toolkit
- Code on Linux*
- Microsoft C/C++ for Visual Studio Code

## Set the Environment Variables and Launch Code

Ensure that your session and any of its terminal sessions or child processes inherit the oneAPI development environment by setting the required environment variables before launching. Perform the following steps to set the required environment variables:

1. Open a terminal or command prompt session.
2. Use the setvars.sh script to initialize your oneAPI environment:

```
source /opt/intel/oneapi/2025.0/oneapi-vars.sh
```

   For more information about the location of the setvars or oneapi-vars script, refer to the following topic:
   Use the setvars and oneapi-vars Scripts with Linux*

3. In the same command line session, launch VS Code* by running the following command:

```
code
```

## Create a VS Code* Project and Enable Code Completion and Debugging

If you do not have a VS Code* project for FPGA development, you must create one. The FPGA Template oneAPI sample project is the recommended starting point for your new FPGA development project. The FPGA Template sample project includes a CMake build system to automate selecting the various command-line flags for the and a simple single-source design to serve as an example.

For more information about the FPGA Template sample project, review the project README file on GitHub: FPGA Template Sample

To create a new project based on the FPGA Template sample project:

1. Initialize a oneAPI development environment using the setvars.sh script.
2. In the same terminal, clone the oneAPI FPGA Code samples, and navigate to the fpga_template code sample. Then, launch VS Code:

```
git clone https://github.com/altera-fpga/hls-samples.git
cd hls-samples/Tutorials/GettingStarted/fpga_template
code
```

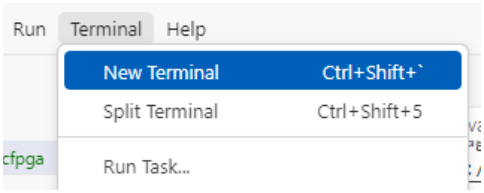
To enable code completion from the oneAPI header files in VS Code, you must enable code completion individually for each new oneAPI project as follows:

3. Open the command palette (**View** > **Command Palette**) and search for C/C++: Edit Configurations (JSON)
4. In the c_cpp_properties.json file, make the following configuration the **only** configuration in the file:

```
{
  "name": "oneaAPI  with FPGA Support Package",
  "includePath": [
   "${default}",
   "${workspaceFolder}/**",
   "${env:INTELFPGAOCLSDKROOT}/../../include/**",
   "${env:INTELFPGAOCLSDKROOT}/../../opt/oclfpga/include/**"
  ],
  "defines": [
   "_DEBUG",
   "UNICODE",
   "_UNICODE"
  ],
  "cStandard": "c17",
  "intelliSenseMode": "linux-clang-x64",
  "compilerPath": "${env:INTELFPGAOCLSDKROOT}/../../bin/icpx",
  "cppStandard": "c++17"
}
```

You can use your native debugger to debug your oneAPI kernel in VS Code* if you disable code optimizations when you compile your code. Before you can debug your application in VS Code*, you must configure running and debugging in your project. You must complete the following instructions for each oneAPI project that you want to enable running and debugging in:

5. If you have not yet compiled your project for debugging, compile your source code for emulation. If the terminal view is not already open in VS Code*, you can open it from the Terminal menu:




   Compile your design in the VS Code* terminal.

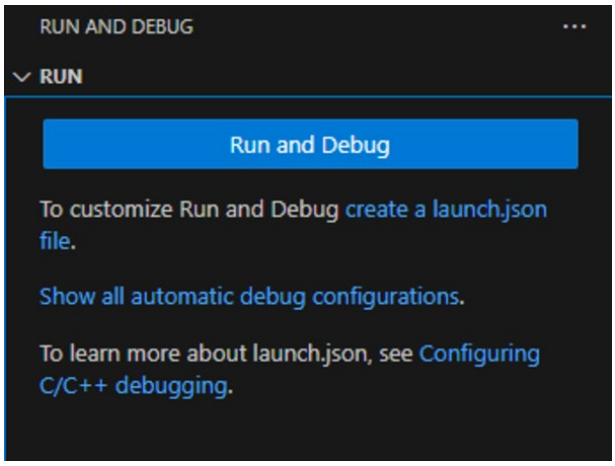
**IMPORTANT:** Ensure that you include the -g and -O0 compiler command options. The -g option enables debugging and the -O0 option disables code optimizations.

```
icpx -fintelfpga -g -O0 <kernel code.cpp> -o fpga_emu
```

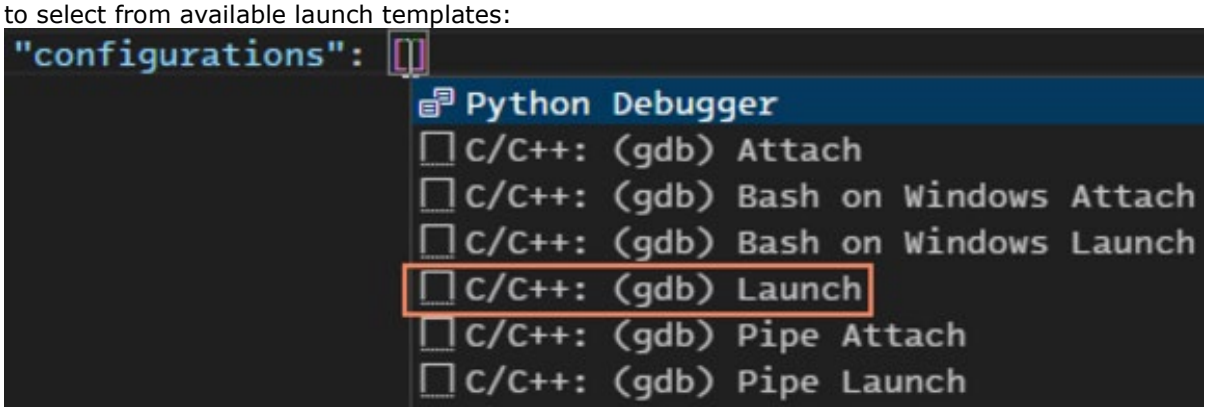
6. If you are compiling a code sample, CMake generates the debug flags for you:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug
make fpga_emu
```

7. Click the Run and Debug icon in the Activity Bar (or press Ctrl+Shift+D).

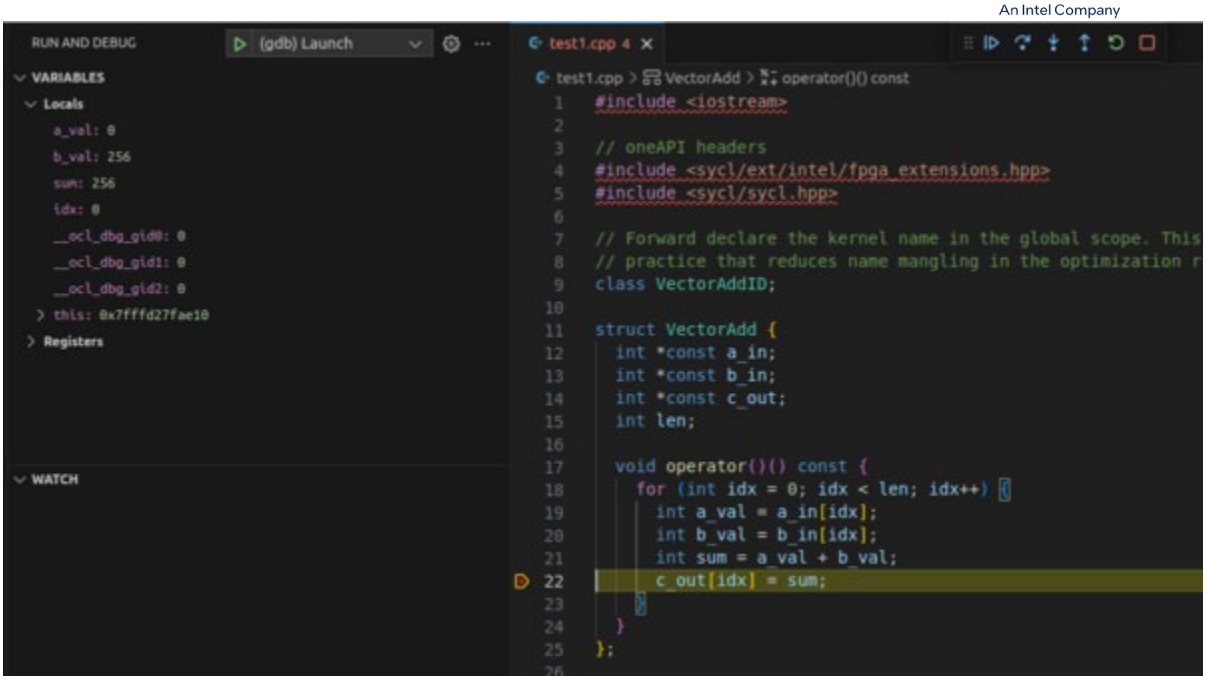



8. Click **create a launch.json file**.
9. Select the **C++ (GDB/LLDB)** debugger.
10. Add the configuration to your launch.json file as follows:
    a. Place your cursor in between the `[]` of the `"configurations":[]` line and press Ctrl+Space to select from available launch templates:




    b. Select the **C/C++: (gdb) Launch** template.
    c. Update the `"program": "enter program name, for example ${workspaceFolder}/a.exe"`, pair to point at the executable file (generated when you compiled your kernel for emulation) that you want to debug.
11. Save the launch.json file and close it.
12. Click the Run and Debug icon in the Activity Bar (or press Ctrl+Shift+D). At the top of the Run and Debug Side Bar, ensure that the correct debugger is selected:


13. Press F5 (or click the arrow on the left of the selected debugger) to start debugging.
    The debugger automatically stops at any breakpoints that you have set in your code. You can inspect your variables and step through the code as you would with any GUI-based debugger.




# Emulate and Debug your Kernel

Before emulating and debugging your kernel, ensure that you have initialized the oneAPI environment as described in Set the Environment Variables and Launch Code. Ensure that you have configured the VS Code* project at least once as described in Create a VS Code* Project and Enable Code Completion and Debugging. Make sure you are using a project based off the FPGA Template Sample.

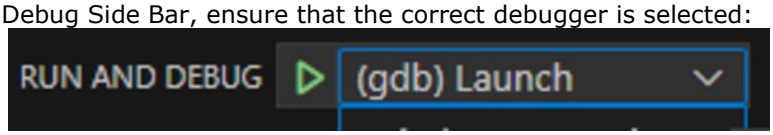
1. In the VS Code* terminal, create a build directory if you don't already have one and navigate to it.

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug
make fpga_emu
```

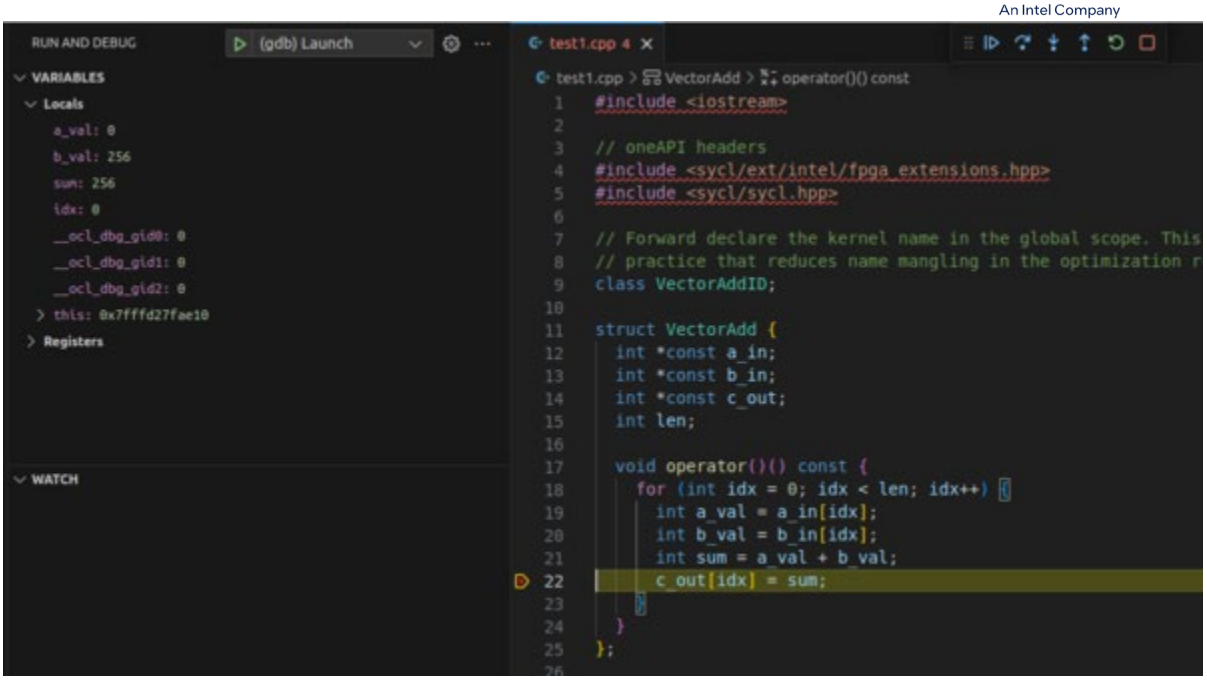
2. Run the sample on the FPGA emulator

```
./<project name>.fpga_emu
```

3. To debug, click the Run and Debug icon in the Activity Bar (or press Ctrl+Shift+D). At the top of the Run and Debug Side Bar, ensure that the correct debugger is selected:




4. Press F5 (or click the arrow on the left of the selected debugger) to start debugging.
The debugger automatically stops at any breakpoints that you have set in your code. You can inspect your variables and step through the code as you would with any GUI-based debugger.




For more information on emulation, see the Intel® oneAPI DPC++/C++ Compiler Handbook for FPGAs:

Emulate and Debug your Design

## Generate and View the FPGA Optimization Report

The FPGA optimization report can provide high-level details about your application performance even before you generate an actual FPGA hardware image. The FPGA code samples also include instructions for building and viewing the FPGA Optimization Report.

**IMPORTANT:** The report is generated by the Intel® oneAPI DPC++/C++ Compiler in the form of HTML pages that you can view in a web browser. For more information about using the FPGA optimization report for achieving best performance, refer to the Review the FPGA Optimization Report.

Before generating an FPGA optimization report, make sure that you've followed the instructions in Create a VS Code* Project and Enable Code Completion and Debugging. The following instructions assume that you are using an FPGA code sample, which includes CMake infrastructure for compiling oneAPI code.

1. In the VS Code* terminal, create a `build` directory, if you do not already have one, and navigate to it. Generate an FPGA optimization report:

```
mkdir build
cd build
cmake ..
make report
```

2. The generated report will appear in the <project_name>.prj/reports directory. You can open it with a web browser (such as FireFox*) with the following command:

```
firefox <project_name>.prj/reports/report.html
```

# Simulate your Kernel

You can simulate your kernel using Questa* simulator software to verify that the FPGA logic generated by the oneAPI FPGA compiler behaves consistently with the emulated code. FPGA simulation requires Quartus® Prime software and a Questa* simulator (installed separately).

Before simulating your kernel, ensure that you have initialized the oneAPI environment as described in Set the Environment Variables and Launch Code. Ensure that you have configured the VS Code* project at least once as described in Create a VS Code* Project and Enable Code Completion and Debugging. Make sure you are using a project based off the FPGA Template Sample.

1. In the VS Code* terminal, create a build directory if you don't already have one and navigate to it. Compile the code for simulation. This will take longer than compiling for emulation.

```
mkdir build
cd build
cmake ..
make fpga_sim
```

2. Run the sample on the FPGA simulator. This will take longer than running the sample on the FPGA emulator.

```
CL_CONTEXT_MPSIM_DEVICE_INTELFPGA=1 ./<project name>.fpga_sim
```

For more information on simulation, see the Intel® oneAPI DPC++/C++ Compiler Handbook for FPGAs:

Evaluate Your Kernel Through Simulation

# Generate and Compile an FPGA Hardware Image

You can compile your kernel into an FPGA hardware image, which run Quartus Prime to obtain accurate estimates of resource utilization and $f_{MAX}$. Generating an FPGA hardware image requires Quartus® Prime software.

Before compiling your kernel, ensure that you have initialized the oneAPI environment as described in Set the Environment Variables and Launch Code. Ensure that you have configured the VS Code* project at least once as described in Create a VS Code* Project and Enable Code Completion and Debugging. Make sure you are using a project based off the FPGA Template Sample.

1. In the VS Code* terminal, create a build directory if you don't already have one and navigate to it. Compile the code for simulation. This will take longer than compiling for emulation.

```
mkdir build
cd build
cmake ..
make fpga
```

2. The generated report will appear in the <project_name>.prj/reports directory. You can open it with a web browser (such as FireFox*) with the following command:

```
firefox <project_name>.prj/reports/report.html
```

3. The report will now additionally contain resource estimates from Quartus Prime. These estimates are more accurate than the compiler-generated resource estimates.

# Document Revision History

| Date | Version | Changes |
|---|---|---|
| 2025-05-05 | 1.0 | Initial release. |