AT4-511 Create and show demonstrations showing results of VHDL and OpenCL/OneAPI

Contributors: P. Thiagaraj, C. Vinutha

August 2021

1 Introduction

PSS FDAS module uses VHDL and OpenCL implementations. OneAPI FPGA tools allow integrating pre-verified and optimised VHDL codes as functions for the SYCL kernels. The VHDL integration in to OneAPI and OpenCL is through a static library creation procedure prescribed by intel [1][2][3]. We have set up the OneAPI environment in a server and followed the static library creation. A related PSS document is available in [4].

This report outlines the static library creation method for OpenCL and OneAPI codes. The technique used to create the static library is similar between openCL and OnePI. For brevity, we show the details for the OneAPI and highlight differences where necessary. We also find discrepancies in the library creation procedure recommended by intel and following up with intel to sort them.

2 Packaging VHDL codes for oneAPI

The steps to package a VHDL code from the OneAPI kernel are similar to the C static libraries used in a GCC environment [5] [6].

Three steps to include a VHDL code in an OneAPI DPC++ SYCL kernel are [1][2][7]:

- RTL code preparation
- Packaging to a custom library, and
- Compiling the design.

2.1 RTL code preparation

The RTL stands for register-transfer logic, a hardware design description suited for pipelined architecture accelerators such as an FPGA. An RTL level code can come from VHDL or Verilog. In our studies below, we refer to a VHDL based RTL design modules.

1. VHDL static libraries creation needs previously verified and optimised VHDL codes.
2. The design needs to work on a single clock domain.
3. The VHDL codes may need to be modified to have streaming type inputs and outputs.
4. The number of pipelined stages in the VHDL design, i.e., the latency of the design, is identified apriori.
5. If the latency of the VHDL design is variable, an appropriate forward and back pressure mechanism is incorporated into the design.
6. The VHDL design executes in CPU in an emulation mode, especially during the code development process. For this purpose, a functional model of the VHDL design is made available. This model is developed in C.
7. Prepare an XML format descriptor for the VHDL design. This descriptor, known as an object-manifest file, is used to describe the functionalities of the VHDL code, capturing the function name, input-output interface details, processing capacity (number of multiple inputs that the module can simultaneously process), VHDL source name, C-model name and optionally providing an estimate for the FPGA resources needed.

In addition, the VHDL design that communicates with an external memory must use a memory-mapped interface such as an Avalon streaming interface and follow bursting restrictions that are typically common for the memory interface controllers [9]. The RTL component data inputs are passed as values, and it must receive all its inputs simultaneously. The RTL modules can only be helper functions and get integrated into an SYCL kernel during kernel compilation. An example of the XML format object-manifest file is given in Section 11.1.2.7 of Intel FPGA SDK for OpenCL Pro Edition: Programming Guide [8].

A header file (with extension as *.hpp for the OneAPI and *.hcl for OpenCL flows) declaring the VHDL functions need to be prepared for inclusion during compilation. [1] [10, section 11.1.6]

2.2 Packaging to a custom library

A static library is a single file containing multiple objects created from SYCL, OpenCL, High-Level Synthesis (HLS) sources, or register transfer level (RTL) VHDL codes. The objects are essentially an intermediate representation of the codes with both CPU and FPGA representations. First, the individual object file is created for every input source VHDL file using the fpga_crossgen command (the "aoc" command for the older versions of the openCL flow). Then, these Object files are combined into a library file using the fpga_libtool command. A typical library creation flow is shown in Fig. [1].

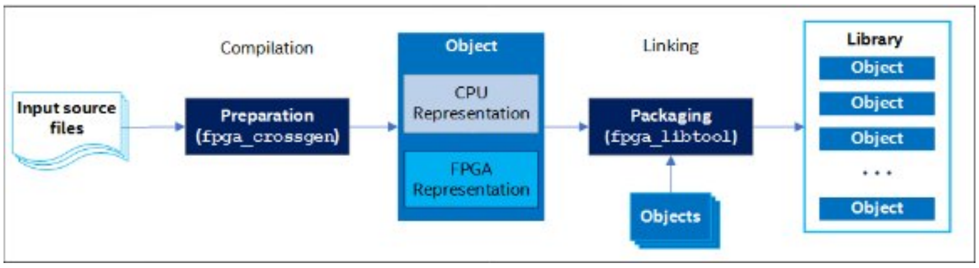

Figure 1: Library creation flow chart

A single library file can get objects from different input source files, provided all objects target the same high-level design. Example command sequences to create the library objects from different input files are provided on Intel pages [1], and [10] for OneAPI and OpenCL flows, respectively.

2.3 Compiling the design

The static library thus created provides functions for the SYCL kernels. Compilation flow to include the static libraries to SYCL kernel (OpenCL flow) is depicted in Fig. [2].

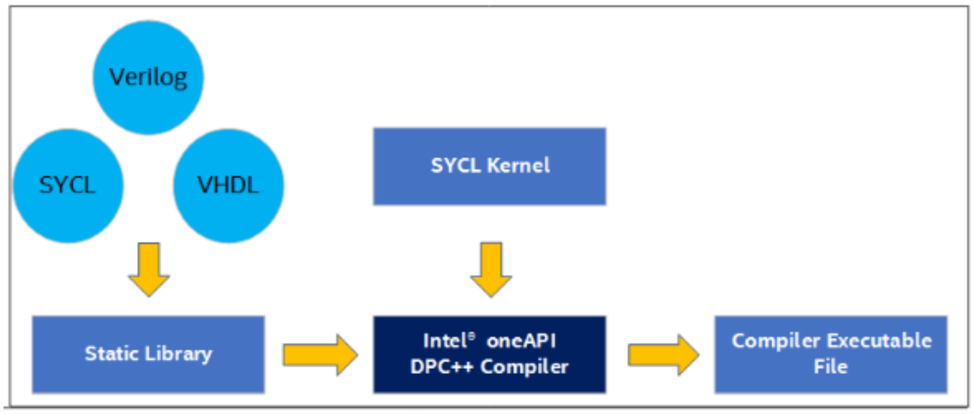

Figure 2: Use of static Libraries for FPGA - OneAPI compilation flow

For this compilation flow, the intel's data parallel c++ compiler dpcpp compiler is used for OneAPI and the aoc compiler is used for the OpenCL flow [1][13]. In both these cases (onAPI and OpenCL), the present compilers depends on the intel Quartus synthesis tools and OpenCL BSP layers to produce the executable files/binaries.

3 Status from the Work

We have studied the custom library creation procedure and installed the intel OneAPI and OpenCL tools on a local server. The library object creation procedure documented in the intel references needs updates regarding the XML file and its use with the library-object creator tool flow. Currently, the documents section on XML object-manifest file format lead to a shared page for both OneAPI and OpenCL [8], which appear incompatible with the OneAPI fpga_crossgen tool flow. Although the OpenCL objects got created they couldn't be combined to form a library due to a (tool version?) compatibility issue. We are following the issue up with intel.

4 Conclusion

Intel's OneAPI is a new code-development environment meant for CPU, GPU and FPGA based accelerators. OneAPI and OpenCL are high-level languages, and their code compilation flow is different from what is typically followed for the VHDL designs. PSS FDAS module uses VHDL and OpenCL implementations. Specific complex signal processing tasks are optimally expressed in a VHDL based design. Intel OneAPI and OpenCL have provisions for packaging pre-verified and optimised VHDL codes with OneAPI DPC++ and openCL code kernels and executing them. It involves creating static libraries of VHDL codes for DPC++ or OpenCL during the compilation and producing the final FPGA executable. Our earlier study investigated the OneAPI code compilation, execution and debugging flows [4][11][12[13] This study investigated the VHDL custom library creation flow for the OneAPI and OpenCL environment and updated the findings in this document. We have identified one main issue with the library creation tool flow and will follow up on this work in the future.

—x—

A References

1. Creating static libraries for OneAPI. https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/programming-interface/fpga-flow/use-of-static-library-for-fpga.html

2. Intel FPGA SDK for OpenCL Advanced Features - Section 11: https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html#ewa1455918581924

3. Building an RTL Module for the Intel FPGA SDK for OpenCL https://www.youtube.com/watch?v=1SdBctS4SKw

4. AT4-507 Comparison of Compilation, Execution,Debugging flows between oneAPI and OpenCL. - https://drive.google.com/file/d/18eoq0mSqkYiulqJTcIEwAXi0oEAKN669/view?usp=sharing

5. Making static librraies for C - https://www.youtube.com/watch?v=JU-vwvSH_0g

6. How to handle dynamic and static libraries in Linux - https://opensource.com/article/20/6/linux-libraries

7. Details about SYCL
https://www.khronos.org/sycl/

8. XML Object Manifest File Syntax of an RTL Module
https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html#ewa1454438935654

9. Restrictions and Limitations in RTL Support. https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/programming-interface/fpga-flow/use-of-static-library-for-fpga/restrictions-and-limitations-in-rtl-support.html

10. Specifying an OpenCL Library when Compiling an OpenCL Kernel - section 11.1.6.
https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html#ewa1452613689582

11. Main reference page for OneAPI The Intel® oneAPI Base Toolkit (Base Kit) is a core set of tools and libraries for OneAPI applications across diverse architectures. Main software Download page:
https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html#gs.3xqmxt

12. Build and Run a Sample Project
https://software.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-oneapi-base-linux/top/run-a-sample-project-using-the-command-line.html

13. Intel® FPGA Add-on for oneAPI Base Toolkit
https://software.intel.com/content/www/us/en/develop/articles/oneapi-standalone-components.html#fpga