ARISTOTLE UNIVERSITY OF THESSALONIKI

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Lab 2
Introduction to the Vitis Tool

A. Athanasiadis – D. Karanassos

Instructor: Ioannis Papaefstathiou

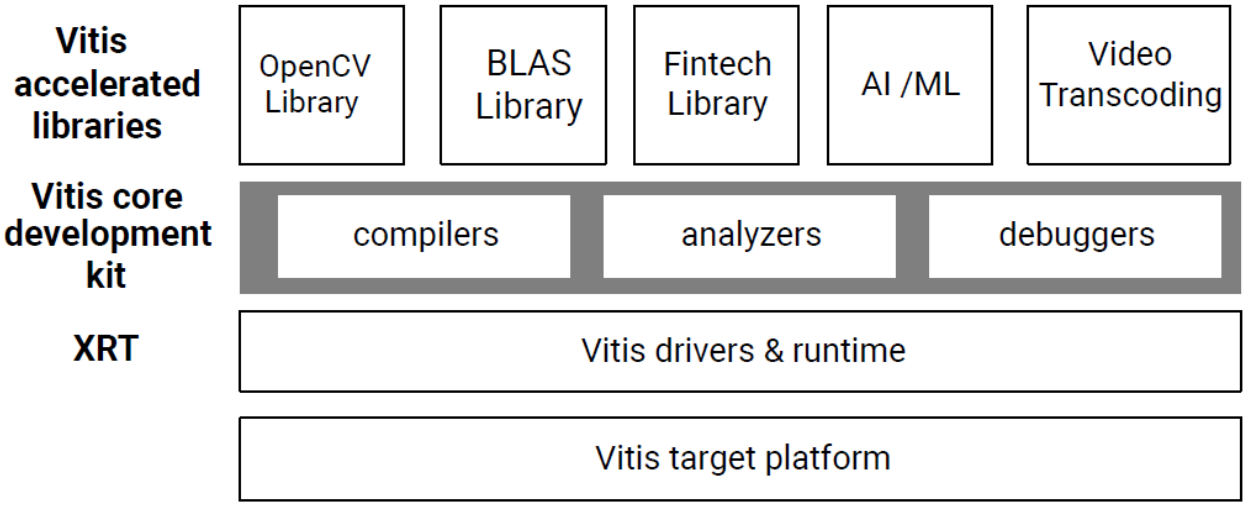
Version 0.5

November 2025

1. Introduction

In Laboratory 1, we focused on designing our accelerator using the Vitis HLS tool. In the present laboratory, you will become familiar with the Xilinx Vitis tool, in which you will execute the accelerator created in Laboratory 1 entirely on the Alveo U200 board.

The unified Vitis software platform is a modern tool that combines all aspects of Xilinx software into a single environment. Vitis can be used to accelerate applications even by software developers who do not have extensive hardware knowledge, through the use of the Xilinx Runtime (XRT) environment.



As shown in the above figure, the unified Vitis software platform consists of the following components:

- Vitis technology targets hardware acceleration platforms such as Alveo™ accelerator cards (used in data centers) and embedded processing platforms (e.g., Zynq® UltraScale+ MPSoC and Zynq-7000 SoC). In this laboratory, we will focus on the Alveo U200 accelerator card.

- XRT provides an Application Programming Interface (API) and drivers that enable seamless communication and data transfer between the program running on the CPU (host system) and the FPGA.

- Vitis provides a complete software development toolchain, including compilers and cross-compilers for building CPU (host) and FPGA programs, analyzers that allow

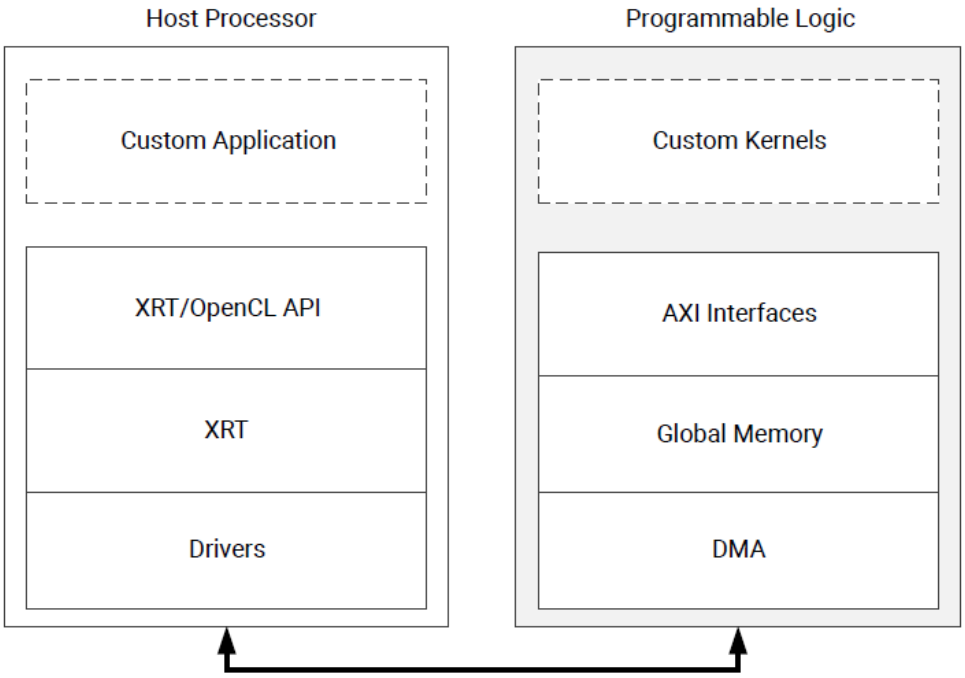
users to profile and analyze application performance, and debuggers that assist in identifying and resolving application issues.

- Vitis also offers acceleration libraries that provide performance-optimized acceleration with minimal code changes and without requiring users to redesign their algorithms. These libraries are available for common mathematical, statistical, and linear algebra operations, as well as for domain-specific applications such as signal and image processing, databases, and data compression. In this laboratory, however, no pre-built libraries will be used, as the goal is to understand the fundamental concepts of the Vitis tool.

As described in Laboratory 1, a Vitis application is divided between a CPU (host) application and one or more hardware-accelerated kernels, connected via a communication channel. The CPU application is written in C/C++ and, using APIs such as OpenCL, executes on x86 or ARM architectures (for embedded platforms), while the hardware-accelerated kernels execute within the programmable logic (PL) region of the FPGA.

API calls managed by XRT are used to process transactions between the CPU program and the hardware accelerators. Communication between the CPU and the accelerator—including control and data transfer—is performed via the PCIe bus or via an AXI bus for embedded platforms. While control information is transferred between specific memory locations in hardware, global memory is used for data transfer between the CPU program and the kernels. Global memory is accessible by both the CPU and the hardware accelerators, whereas the CPU system RAM is accessible only by the CPU application.

For example, a typical CPU application first transfers data from the CPU system RAM to the FPGA global memory. The FPGA kernel then processes the data and stores the results back into global memory. After kernel execution completes, the results are transferred back from global memory to the CPU system RAM. Data transfers between the CPU system RAM and global memory introduce latency, which can be costly for overall application performance. To achieve effective acceleration in a real system, the performance gains achieved by the hardware-accelerated kernels must outweigh the overhead of data transfers.



An FPGA contains the accelerated kernels, global memory, and direct access to global memory via Direct Memory Access (DMA). Kernels may have one or more global memory interfaces and are fully programmable. The Vitis execution model can be summarized in the following steps:

1. The program running on the CPU (host) writes the data required by a kernel into the global memory of the connected device via the PCIe interface (for Alveo accelerator cards in data centers) or via the AXI bus (for embedded platforms).
2. The host-side program configures the kernel with its input parameters and launches kernel execution on the FPGA.
3. The kernel running on the FPGA performs the required computation while reading data from global memory as needed. It then writes the results back to global memory and notifies the CPU upon completion.
4. The CPU program transfers the results from global memory back to the CPU system RAM and continues processing as required.

The Vitis compiler provides three execution modes: two emulation modes used for debugging and validation, and one mode used to generate the actual FPGA kernel:

- **Software Emulation** (sw_emu): Both the CPU application code and the FPGA kernel code are compiled to run on the CPU. This mode is useful for detecting syntax errors,

debugging kernel code alongside the host application, and verifying system-level behavior.

- **Hardware Emulation** (hw_emu): The kernel code is converted into a hardware (RTL) model, which is executed in a dedicated simulator. This process takes longer but provides a detailed and accurate view of kernel activity. This mode is useful for validating the logic intended for the FPGA and obtaining initial performance estimates.
- **Hardware** (hw): The kernel code is compiled into an RTL hardware design and implemented on the FPGA, producing a binary file that runs on the physical FPGA device.

2. Introduction to Vitis (20%)

Carefully read the document "Introduction to Vitis" available in the "**Εργαστήρια**" section of the e-learning platform and execute all steps of the provided example (vadd), using **only the first two** Vitis execution modes (**Software Emulation and Hardware Emulation**). **IMPORTANT**: Please do **not** execute the steps described on page 13 (Hardware mode), where full place-and-route is performed and a bitstream for the Alveo U200 is generated. This is a computationally intensive process (approximately 3 hours) and will negatively impact server responsiveness. You may, however, read this section for educational purposes.

Note: Each group must create only one project on the server, due to limited disk space.

3. Implementing the Accelerator from Laboratory 1 in Vitis (80%)

After successfully completing all steps of the vadd example described in "Introduction to Vitis," integrate into Vitis the IMAGE_DIFF_POSTERIZE accelerator developed in Laboratory 1, extending it to additionally apply a 3×3 filter to the difference image. As seen in the vadd example, the host side is implemented using OpenCL. Based on experience and Xilinx examples, Vitis typically operates using OpenCL code on the host side rather than plain C/C++. **Hint: It is recommended to use the vadd example code as a reference and modify it accordingly, rather than creating all files from scratch.**

For the purposes of the laboratory, assume fixed threshold values (as defined in Laboratory 1).

For simplicity and consistency with the vadd example, you may assume that image data is represented using 32-bit integer types (e.g., int), even though pixel values conceptually lie in the range [0, 255]. Ensure that intermediate computations (subtractions, multiplications,

additions) use sufficient precision and that the final result is clipped to the range [0, 255] before being written to the output.

To avoid excessive execution time during Hardware Emulation, select relatively small image dimensions, e.g., HEIGHT = WIDTH = 64 or HEIGHT = WIDTH = 128.

Adding a 3×3 Sharpen Filter to the Difference Image

In this laboratory, you will extend your accelerator so that after computing image C, a 3×3 filter is applied to produce a final output array C_filt, with the same dimensions as A, B, and C.

For the purposes of the laboratory, assume a fixed sharpening filter:

$$F = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

For each interior pixel (i, j), with $1 \le i \le \text{HEIGHT} - 2$ and $1 \le j \le \text{WIDTH} - 2$, define the filtered value accordingly:

$$S(i,j) = \sum_{a=-1}^{1} \sum_{b=-1}^{1} F_{a+1,b+1} \cdot C[i+a][j+b]$$

And then compute:

$$C_{\text{filt}}[i][j] = \text{clip}(S(i,j),0,255)$$

The function clip(x, 0, 255):

- returns 0, if x < 0
- returns 255, if x > 255
- or else returns x.

For border pixels (e.g., i = 0, i = HEIGHT − 1, j = 0, j = WIDTH − 1), you may choose one of the following policies for simplicity:

- C_filt[i][j] = C[i][j] (copy the difference image value), or
- C_filt[i][j] = 0.

The chosen policy must:

- be identical in the software reference implementation and the HLS code,
- be explicitly stated in your report.

In practice, this filter:

- enhances the central pixel relative to its neighbors,
- emphasizes sharp transitions (edges) in image C,
- makes regions with significant differences between A and B sharper and more distinguishable in the final image C_filt.

The host code must also compute the results using a purely software implementation (on the CPU) and verify that the output array C_filt returned by the FPGA accelerator matches

the software reference element-by-element (e.g., printing "Test Passed" when no mismatches are found).

ΠΡΟΣΟΧΗ: As before, stop at the **first two Vitis execution modes (Software Emulation and Hardware Emulation)** for the reasons stated above.

More information about Vitis can be found at the official documentation[1]

After completing the above steps, select Hardware Emulation, run your project, and open the Vitis Analyzer (vadd_system > vadd > Emulation-HW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin)).

Record the values obtained from the Profile Summary for the following:

i) Kernels & Compute Units → Kernel Execution
ii) Kernel Data Transfers → Top Kernel Data Transfer
iii) Host Data Transfer → Host Transfer

[1] https://docs.xilinx.com/r/2022.2-English/ug1393-vitis-application-acceleration/Getting-Started-with-Vitis