SimBSP: Enabling RTL Simulation for Intel FPGA OpenCL Kernels*

Ahmed Sanaullah Chen Yang Daniel Crawley Martin C. Herbordt
CAAD Lab, ECE Department, Boston University

***Abstract*—RTL simulation is an integral step in FPGA development since it provides cycle accurate information regarding the behavior and performance of custom architectures, without having to compile the design to actual hardware. Despite its advantages, however, RTL simulation is not currently supported by a number of commercial FPGA OpenCL toolflows, including Intel OpenCL SDK for FPGAs (IOCLF). Obtaining reliable performance values for OpenCL kernels requires a full compilation to hardware, while emulation can only provide functional verification of the C code. Thus, development and optimization time-frames for IOCLF designs can be on the order of days, even for simple applications. In this work, we present our custom Board Support Package for IOCLF, called SimBSP, which enables OpenCL kernels to be compiled for RTL simulation. Use of SimBSP reduces the time taken per OpenCL code optimization iteration from hours to minutes. We provide details regarding the standard kernel ports created by the IOCLF compiler, which can be used by testbenches to interface the generated design. We also list the addresses and descriptions of configuration registers that are used to set kernel parameters and provide a start trigger. Finally, we present details of SimBSP toolflow, which is integrated into the standard IOCLF and automates the process of generating kernel HDL and testbenches, and setting up the simulation environment. Our work on SimBSP will be made available Open Source to drive a community effort towards further improving the toolflow.**

I. Introduction

Compiling FPGA-based designs to hardware is a time-consuming process that can take many hours to complete, depending on architectural complexity. As shown in Figure 1a, placing this compilation on the critical path–especially for the multiple iterations of optimization inherent in the development process–can significantly increase development time-frames; users must wait for hardware generation before validating their implementation and measuring performance values. RTL simulation, using tools such as ModelSim [1], helps alleviate this compilation overhead by computing cycle-accurate behavior of the HDL design without generating actual hardware. This allows users to easily debug their designs, identify performance bottlenecks, analyze implementation efficiency, and iterate over the optimization space for their code without requiring the design to be programmed onto the FPGA (Figure 1b).

The Intel OpenCL SDK for FPGAs (IOCLF) toolflow [2] already provides a number of methods for estimating kernel performance without requiring a full compilation to hardware. These include Emulation and Reports. Emulation is used to simulate kernel code for functional verification. Compiling for emulation allows the compiler to generate CPU equivalent code for FPGA-specific constructs, such as channels, and then execute the entire computation in software. This is not only useful for ensuring that computation and memory accesses have been correctly defined, but can also identify run-time faults, such as occurrences of deadlocks. It does not, however, provide any information regarding kernel code mapping to hardware or estimated performance.

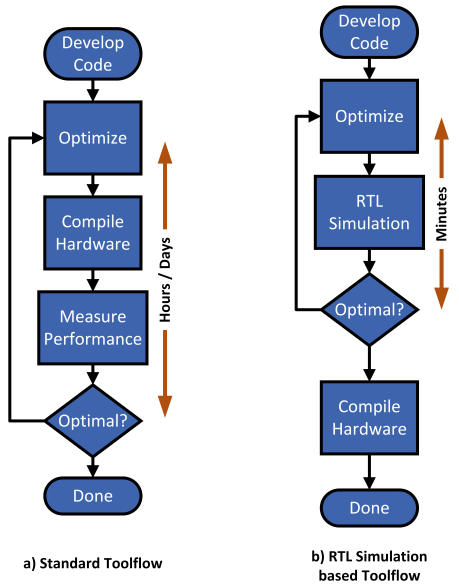

Fig. 1. Toolflows for developing FPGA applications. The standard toolflow (a) requires compilation to hardware in order to validate designs and measure performance. Performing design optimizations using this approach is slow due to large compilation time (hours/days). Having the capability of RTL simulations (b) enables debugging and reliable performance estimates without compiling to hardware.

Reports are generated by the compiler to provide an overview of kernel translation to hardware. Here, we briefly list the main categories of these reports and their contribution towards code optimization. A comprehensive list of reports and their detailed description are provided in the IOCLF best practices guide [3].

- **Loop analysis** is used to determine initiation intervals (II) for loops in the kernel and the dependencies causing high IIs. Resolving these dependencies allows loops to

*This work was supported in part by the NSF through Awards #CNS-1405695 and #CCF-1618303/7960; by the NIH through Award #1R41GM128533; by grants from Microsoft and Red Hat; and by Altera through donated FPGAs, tools, and IP.

operate stall free.

- **Area analysis** provides estimates of resource usage and implementation details for data structures. This is particularly useful for determining whether the compiler has correctly inferred the optimal hardware based on access patterns, or is resorting to sub-optimal, high-resource "safe" options such as memory replication and barrel shifters.
- **System viewer** gives a graphical overview of the kernel computation and memory accesses. Kernel execution is represented as sequential blocks, with each block carrying out a varying number of operations such as memory transactions, channel calls and loop iterations. Details provided include latencies, stalls, types and sizes of Load-Store units created for each memory transaction, and the dependencies between blocks.
- **Kernel memory viewer** gives a graphical overview of the connectivity of Load-Store units with external memory banks. This can be used to verify that the compiler has correctly inferred off-chip access patterns.

The two approaches for estimating kernel performance described above provide high level (and select) details regarding the C to HDL translation, which can be used to perform initial code improvements. These approaches, however, do not guarantee good performance. Kernel codes with no loop dependencies, initialization intervals equal to 1, efficient memories and low latencies can still be sub-optimal. This is because little information is provided regarding the composition, organization, and connectivity of compute pipelines. To truly identify bottlenecks in the design and optimize them, low-level details are required regarding implementation and behavior of the entire system. Therefore, RTL simulation continues to be a key development stage that does not have a reliable alternative in the commercial IOCLF toolflow.

In this work, we have developed a custom Board Support Package (BSP) [4]–[6], called SimBSP, that enables compilation of OpenCL kernels for RTL simulation. SimBSP is based on the Quartus [7] API, and can thus be used as part of the standard IOCLF compilation by setting appropriate environment variables. It is composed primarily of two components, (i) a testbench template that can interface IOCLF generated kernels, and (ii) compilation scripts for generating simulation models and setting up the simulation environment.

SimBSP can reduce the time needed for an iteration of design optimization from hours to minutes, depending on the complexity of the design. We have used this approach ourselves when optimizing OpenCL kernels [8], [9]. Moreover, the simplicity of SimBSP enables it to be used in academic teaching environments, including in-class practical training and workshops. At the PAPAA short course [10], students were able to quickly apply multiple levels of OpenCL optimizations after only a small amount of instruction and with virtually no previous OpenCL expertise.

In the rest of this paper, we provide details regarding how these two components are implemented, and also discuss further features that can be easily supported in future versions of SimBSP. Our work is based on the Intel Arria 10 reference BSP (a10gx) and IOCLF 17.1 toolflow.

II. Testbench

In this section, we discuss details regarding the SimBSP testbench template. We first describe the interfaces exposed by the kernel module (instantiated within the testbench). We then list the configuration registers that are used to set kernel parameters; the testbench must assign appropriate values to these registers before kernel execution can be started.

A. Kernel Interfaces

Table I provides details regarding instantiated kernel ports. Clock frequency is determined at compile time based on post-routing timing models and so an accurate value cannot be known for simulation. Therefore, performance estimates made using SimBSP are a measure of compute latencies, instead of actual execution time. *kernel_cra* is an Avalon Memory Mapping (AVMM) slave interface to configuration registers within the kernel while *kernel_mem0* is an AVMM master interface for reads/writes to external memory. *kernel_irq* is a 1-bit flag raised on kernel completion. Finally, the *crc_snoop* streaming slave interface is not used in SimBSP since it is not directly involved in kernel execution.

TABLE I
KERNEL INTERFACE AND DESCRIPTIONS

Name	Type	Interface Description
clock_clk	Clock	Kernel clock
clock_reset_n	Reset	Active low kernel reset
cc_snoop	Streaming	Not used
kernel_cra	Memory Mapped	Interface to configuration registers
kernel_irq	Interrupt	Interrupt to host machine
kernel_mem0	Memory Mapped	Interface to global memory

B. Configuration Registers

Table II lists the addresses and kernel parameters that are stored in configuration registers. These registers are 64 bits wide, and the testbench can set the value of an entire register (2 32-bit parameters) every cycle using the *kernel_cra* port. Once all configuration registers have been assigned required values, setting Bit 0 of the register at address 0x0 triggers the start of kernel execution.

TABLE II
OPENCL CONFIGURATION REGISTERS

Address	Bits [63:32]	Bits [31:0]
0x0	-	Start (Bit 0)
0x28	Workgroup_Size	Workgroup_Dimensions
0x30	Global_Size[1]	Global_Size[0]
0x38	Number_of_Workgroups[0]	Global_Size[2]
0x40	Number_of_Workgroups[2]	Number_of_Workgroups[1]
0x48	Local_Size[1]	Local_Size[0]
0x50	Global_Offset[0]	Local_Size[2]
0x58	Global_Offset[2]	Global_Offset[1]
0x60 - end	Argument_Pointer[63:32]	Argument_Pointer[31:0]

Apart from specifying the shape and size of work-items/work-groups, configuration registers are also used to

store 64-bit pointers to off-chip memory for kernel arguments. Since the number of kernel arguments can vary for individual applications, addresses from 0x60 onwards can all be used for this purpose.

III. Compilation Scripts

In this section, we present the compilation scripts that are used as part of the IOCLF toolflow to enable RTL simulation. There are two such scripts used by SimBSP as shown in Figure 2, i.e. simulate.tcl and msim_setup.tcl, while the entire process is divided into three stages. These stages are discussed in detail below. It is important to note that only simulate.tcl is a new contribution, while msim_setup.tcl is automatically generated when compiling for simulation.

- Stage 1: We use the standard command for kernel compilation, i.e. *aoc kernel.cl*, to invoke a C to HDL translation stage. The result of this translation is a QSYS [11] system file which contains the kernel implementation.
- Stage 2: After generating the QSYS file, the compiler automatically runs our custom script called *simulate.tcl*. This script performs three important functions. First, it removes logic that cannot be simulated from the QSYS system; this logic can be safely eliminated since it corresponds to modules that do not impact kernel execution, e.g., System Description ROM. Next, the QSYS file is compiled for simulation in order to generate the required HDL files. Finally, the default testbench is replaced with a SimBSP testbench. At this point, the simulation directories have been set up, and so the command *aoc kernel.cl* terminates.
- Stage 3: In the last stage, we use Modelsim to manually source the final script, i.e. *msim_setup.tcl*. This will compile the generated HDL (from stage 2) and launch the RTL simulation.

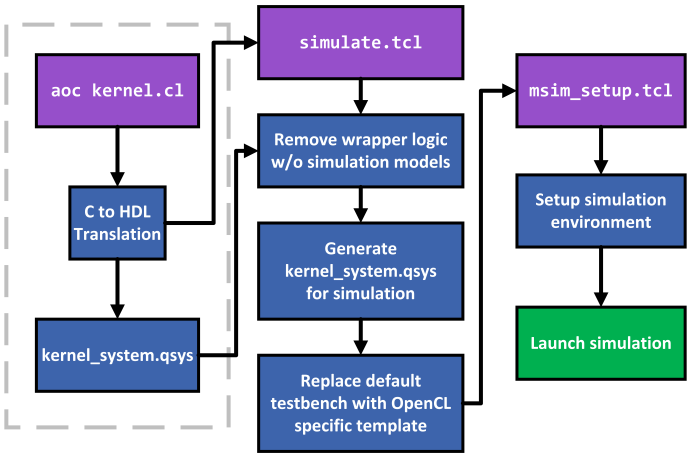

Fig. 2. The compilation process used by SimBSP to generate simulation models. Blocks in the dashed rectangle represent the standard IOCLF toolflow, while the remaining blocks are specific to SimBSP.

IV. Discussion and Future Work

In this abstract, we outline the importance of RTL simulations for reducing development timeframes of FPGA based designs. We present a custom Board Support Package, called SimBSP, which adds this important functionality to the Intel OpenCL SDK for FPGAs. We provide details regarding how the SimBSP testbench interfaces kernel logic, and how the entire process of C to RTL simulation can be achieved using simple compilation scripts within SimBSP.

SimBSP provides an initial exploration into the addition of RTL simulation to the IOCLF toolflow. In future versions of SimBSP, we are aiming to achieve the following targets.

- Make SimBSP available as Open Source in order to drive community efforts towards adding more features/capabilities to SimBSP.
- Implement a cycle-accurate simulation model for off-chip memory in order to get even more reliable performance estimates. Currently, all DRAM access latencies are simulated as a constant.
- Provide support for other interfaces, apart from the ones listed in Table I, e.g., a streaming network interface.

This work is part of a larger project to develop a systematically and empirically guided toolflow for OpenCL on FPGAs [8], [9]. In directly related work [12], we describe in detail methods for removing OpenCL wrappers, and applying test cases directly at the inputs of generated compute pipelines.

References

[1] "ModelSim," https://www.mentor.com/products/fv/modelsim/ , accessed: 2018-01-16.

[2] "Intel FPGA SDK for OpenCL," https://www.intel.com/content/www/us/en/programmable/products/design-software/embedded-software-developers/opencl/developer-zone.html, accessed: 2018-08-30.

[3] "Intel FPGA SDK for OpenCL Pro Edition: Best Practices Guide," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf, accessed: 2018-10-12.

[4] "Nallatech FPGA Accelerators support the Altera SDK for Open Computing Language (OpenCL)," http://www.nallatech.com/solutions/fpga-accelerated-computing/opencl-software-bsps/, accessed: 2018-01-16.

[5] "BittWare OpenCL Board Support Packages," http://www.bittware.com/wp-content/uploads/datasheets/ds-OpenCL_BSP.pdf, accessed: 2018-01-16.

[6] "Development Tools and IP," http://gidel.com/development-tools-and-ip/, accessed: 2018-08-30.

[7] "Intel Quartus Prime Software Suit," https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html, accessed: 2018-08-30.

[8] A. Sanaullah and M. Herbordt, "FPGA HPC using OpenCL: Case Study in 3D FFT," in *Proc. International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2018.

[9] ——, "An Empirically Guided Optimization Framework for FPGA OpenCL," in *Proc. IEEE Conf. on Field Programmable Technology*, 2018.

[10] "Croucher Summer Course: Performance-Aware Programming with Application Accelerators," http://cscpapaa.eee.hku.hk/, accessed: 2018-09-1.

[11] "Platform Designer (formerly Qsys)," https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/features/qts-platform-designer.html, accessed: 2018-08-30.

[12] A. Sanaullah and M. Herbordt, "Unlocking Performance-Programmability by Penetrating the Intel FPGA OpenCL Toolflow," in *IEEE High Perf. Extreme Computing Conf.*, 2018.