High Level Synthesis II
Executive Summary

Joshua Glover
Toby Harradine
William Miles

April 8, 2019
Introduction

FPGAs are becoming increasingly attractive for numerous applications since they provide a significant improvement in speed and energy efficiency over a software implementation. However, the register transfer level (RTL) development for producing digital hardware is notoriously cumbersome and inaccessible due to the requirement of lower level knowledge. High level synthesis aims to overcome this by providing a means of generating hardware implementations from higher level languages such as C or C++. Many HLS tools have been developed over the last decade, each with varying methodologies and approaches towards the level of abstraction and control over optimisation provided to the user. This seminar compares two HLS tools, LegUp and VivadoHLS, analysing their differences in terms of their level of abstraction, performance, accessibility and the availability of their source code. From this analysis, we ask about the future of these tools and how their HLS implementations shape their usability and design methodology.

LegUp

LegUp is an open source HLS tool aimed at increasing the accessibility of hardware design to developers with a software background. The tool achieves this by prioritising a high level of abstraction, removing much of the knowledge (e.g. clocking, memory sharing, communication) required to design hardware. Since not all C program code is suitable for hardware implementation, LegUp provides the ability to synthesis to a hybrid system of a MIPS soft processor connected to hardware accelerators. Through this, code segments unsuitable for hardware (e.g. sequential operations) can be run on the soft processor while the rest can be synthesised as hardware accelerators. Doing this drastically increases the coverage of supported C language, preventing developers from having to write their high level programs whilst tip toeing around the support of the tool or with the hardware implementation in mind.

LegUp’s open source nature allows anyone to extend the tool to support additional FPGA architectures or add optimisations to synthesis itself. Furthermore, it allows for further HLS research that ensures long term viability, accelerating HLS closer to widespread adoption.

Although LegUp does provide options for optimisations like unrolling loops and parallelising threads, its ease of use comes at the cost of fine grain control. As a result, performance of LegUp’s synthesised designs does not match that of more complex HLS tools.

The question we have to ask is whether this ease of use, highly abstracted approach is the most effective when it comes at the cost of performance and control over the final hardware design.

VivadoHLS

Xilinx’s approach to HLS is to define a subset of the target language (C/C++/SystemC) which can be fully synthesised into hardware. Each function in the hierarchy is separated into its own logical block. This can be used to develop IP blocks and include that in a block design through Vivado’s IP Integrator.
This approach has been shown to drastically reduce the time it takes to implement and verify some hardware components in programmable logic, however it also raises some questions:

- How does this limit the programmer’s ability to accurately describe hardware?
- How does Xilinx provide extra functionality to this subset of C/C++ which allows the programmer to reap the advantages of reconfigurable computing?
- Does the time saved on implementation and verification justify the potential loss of optimization?

We will attempt to answer these questions in a number of ways. Firstly, we can analyze how this approach has already made an impact on commercial development. According to Xilinx, using their HLS workflow to implement hardware designs has yielded a speed up in development time by orders of magnitude, however they have also sometimes lead to poorer quality of results.

Secondly, we will discuss how we feel about the design and limitations of the language features provided when using HLS. In particular, we will examine style of some example code, its ease of verification, and the benefits and ramifications of inferred parallelism in C code.

Lastly, we will brainstorm some ways in which Xilinx could improve its HLS tools, with relation to its explicitness, readability, and optimisation.

Comparison

LegUp, and Xilinx’s use of HLS in their ‘UltraFast High-Level Design Productivity Design Methodology’, propose two paradigms for HLS use. LegUp aims for a complete replacement of current hardware design methodologies via complete system translation and synthesis, while the Xilinx methodology is augmenting the current design approach to achieve greater productivity in system prototyping, verification, and design-space exploration.

These differing approaches impact the performance, accessibility, and use cases for these systems. LegUp is completely open source and FPGA vendor agnostic; aimed squarely at the research community. Xilinx’s HLS toolset integrates into their existing proprietary toolset and is much more focused on use for commercial system development. This is reflected in Xilinx’s HLS system significantly outperforming LegUp when used to its fullest extent.

This does however raise some important questions that we will be further exploring: do the benefits of the differing paradigms outweigh the losses incurred? Do these tools have viability within commercial and research areas? Do either of them actually provide a system that is worth using over traditional HDL methodologies? And, are there better alternatives to HLS?