Measuring the Effectiveness of FPGA Programming Languages

Stu Mesnier, csm1@seas.wustl.edu (A project report written under the guidance of Prof. Raj Jain)

Download 

Abstract

Field-Programmable Gate Arrays (FPGA) offer dramatic processing speed ups versus general purpose microprocessors for certain applications. Unfortunately, there are too few programmers to meet industry demand. This paper proposes that commonly taught sequential languages, such as C and Java, can be adapted to replace the ones, such as VHDL and Verilog, commonly used in this realm. The traditional FPGA programming languages require dramatic shifts in programmer thinking toward specifying massively parallel operation logic elements. Little empirical research has been reported comparing the two styles of programming languages, so this paper describes a distributed experiment that can provide investigators with data needed for analysis.

Keywords: FPGA, VHDL, Verilog, C, C++, Java, Handel-C, Impulse-C, programming, programming language, programming efficiency, operational efficiency

Table of Contents

1. Introduction

The history of computing industry has been characterized more than any other by “faster, smaller, cheaper”.

Intel co-founder Gordon Moore's famous "Moore's Law" not only uncannily describes transistor fabrication advances, but also challenges and motivates us to continue along the trajectory it predicts.

1.1. Context

Numerous analogies put advances in computing into perspective with other technologies, for instance, Bill Gates once remarked, "If GM had kept up with technology like the computer industry has, we would all be driving $25.00 cars that got 1,000 miles to the gallon" [Gates07]. The same cannot be said about advances in software, in fact, often just the opposite. Wirth's Law states: "Software is getting slower faster that hardware is getting faster" [Wirth95].

Nonetheless, there are significant computing realms where software and hardware intersect, and while Moore's Law continues to propel hardware development, competing software tools to exploit the hardware violate Wirth's Law. One such area is development of Field Programmable Gate Arrays (FPGA). These remarkable devices operate at clock speeds on the order of 100MHz (or about 10 to 30 times) slower than general purpose microprocessors, but can achieve throughputs hundreds of times faster because of their massively parallel processing abilities [Storaasli08].

FPGAs provide digital circuit designers the ability to rapidly produce intricate digital logic circuits in a fraction of the time and cost required for similar implementations using Application Specific Integrated Circuits (ASIC). While ASIC implementations are almost always smaller and faster than equivalent FPGA implementations, the latter are often tens of thousands of times less expensive to develop while usually only a few percents slower in operation and higher in energy consumption [AMI06].

As attractive as FPGAs are for circuit development, they suffer from a simple problem: there aren't enough well-trained FPGA programmers [Pellerin08]. FPGAs are programmed by loading them with a special bit map that sets the function of each "slice" or Configurable Logic Block (CLB) and another bit map that specifies the interconnection between CLBs and the routing of signals and to the chip's Input Output Blocks (IOB). These bit maps are analogous to the machine code executed by General Purpose Microprocessors, and are similarly difficult to produce manually. Naturally, this is not done in the real world. Instead, programmers use High Level Languages (HLL) which are processed by appropriate compilers to generate the necessary object code.

1.2. The Problem

The realm of FPGA programming, was originally dominated by a group of HLLs referred to as Hardware Description Languages (HDL). VHDL and Verilog are examples. These allow programmers to describe the logical elements of the circuit, particularly the logic gates, and how they interconnect. Standardized languages naturally lead to competition among compiler developers and, especially in the case of FPGA chip makers, to backend software that can be readily calibrated to generate the essential bitmaps for their specific devices. Standardized languages also provide for relative efficiencies while educating programmers in their use.

Most institutions of higher learning teach the majority of new programmers in commonly used commercial and scientific HLLs, such as C, C++, and Java. The unfortunate consequence for industry is an unsatisfied demand for FPGA programmers because few understand the kinds of problems that are attacked with FPGA solutions. Computer Science programs do not emphasize hardware programming leading to a shortage of trained programmers.

There is a completely different programming paradigm in play among these two classes of HLLs: the general

purpose HLLs assume sequential processing with special constructions for loops and branches, while HDLs assume parallel processing and offer limited sequential or conditional logic structure. This requires a radical new way of thinking for programmers originally trained to use sequential logic.

A new class of HLLs are emerging to address these problems. Instead of retraining sequential-processing-thinking programming about the intricacies of parallel programming for digital circuits, the approach is to adapt common HLLs for the HDL task. Variations of C and Java, among others, are provisioned with special functions to support chip IO, and also pragmas and functions programmers use to specify parallelism and circuit choices, along with hidden optimizations that identify and exploit inherent parallelism. These language variations do not require radical changes to thought processes. Instead, they simply require some additional concepts and how to use them effectively.

The question is: Are some languages better for specifying FPGA circuits than others? And if the answer appears to be "yes", then can meaningful differences be measured with sufficient accuracy so that decisions can be made regarding their selection and, when the time is right, abandon earlier choices for new ones? This paper outlines an experiment to gather information to answer these questions.

1.3. Programming Languages

Two programming languages have traditionally accounted for most FPGA programming in the usual HDL style: VHDL and Verilog. Verilog was designed in the mid 1980's, and is somewhat more C-like than VHDL. It is conceded by some to be easier to use than VHDL because of simpler syntax and fewer constructs [Verilog08]. VHDL was originally created for the Department of Defense (DoD) to document the behavior of ASICs that were created by supplier companies. Its syntax is similar to that of the Ada programming language, also developed for the DoD. The first version designed to meet IEEE standards was introduced in 1987 [VHDL08].

Both languages translate source code elements into a common transportable notation called Register Transfer Level (RTL). Many software vendors produce specialized tools for producing RTL and also for using RTL to drive simulators and circuit synthesizers. Another common feature is the ability to express logic through either structural or behavioral constructs. Structural specification allows the programmer to describe the circuit in terms of elements and connections, while behavioral specification ignores component descriptions and simply defines output function in terms of the inputs and stored states.

Numerous adaptations have been made to C and Java programming languages to produce new HDLs; among these are Impulse C, Handel C, SystemC, JHDL (based on Java), and MyHDL (based on Python). Mostly, they provide only behavioral specification, and leave structural-like specification to disciplined programmers. Their primary advantage over the original HDLs is they allow program development to begin using familiar programming languages, including comfortable verification techniques, then apply a series of transformations in order to adapt the program for parallel processing as a circuit simulation. Transformations include recoding with special functions, specifying pragmas, and utilization of specialized classes or variables.

Verification techniques are substantially different between the original and adapted HDLs. Adapted HDLs allow programmers to use ordinary print statements and data files for supplying input and capturing output. Verilog offers limited provisions for I/O, but is not as robust as its C counterparts. Integrated Development Environments generally provide for breakpoints and watchpoints, but these are sometimes awkward to use in a parallel programming environment under test. Waveform analysis is another method of verification specific to HDLs. As the circuit operation is simulated, all or selected subsets of signals are displayed as synchronized logic waveforms in a separate window. This can be either indispensible or useless clutter, depending on the experience of the programmer.

Throughout the remainder of this paper, VHDL and Verilog are referred collectively as the Traditional HDLs (THDLs). The newer class of HDLs, such as Handel-C, Impulse-C, JHDL, et. al., are referred collectively as the Adapted Sequential HDL Programming Languages (ASPLs).

2. Performance Measurements

The broad question to answer is "Are any HDLs better than the others?" in particular, "Does the class of ASPLs offer significant net advantages for hardware programming than the THDL class?" If so, can the relative advantages and disadvantages be measured and quantified so that cost-related trade-offs can drive decisions about their purchase and use? In order to begin to answer, suitable factors and metrics must be identified. The various HDLs can be compared individually or collectively. Application types and programmers' level of experience may influence the observations. This paper focuses on the information needed to make decisions and the design of an experimental framework to gather it, rather than attempting a comprehensive comparison.

Change naturally imposes a learning requirement. Ideally, skilled programmers will learn a new language and become proficient in a "short" or "reasonable" time period. Ease of learning seems important since, from a business perspective, it indicates the time until "maximum", or "sufficient", productivity can be achieved. Productivity can be measured, but may not be the best factor for comparing HLLs since many factors affect it. Yet, presumably, productivity rises with competence. Competence can also be measured, but can also be useful if self-reported.

Of greater interest, however, are the programming and operational efficiencies that can be achieved after programmers reach stable levels of competence. While the experiment proposed here captures and evaluates data in all three areas, learning, programming, and operating, the most useful business decision-making information regarding productivity and efficiency come from the latter two. The important business questions are "How productive are programmers using ASPLs vs. THDLs" and "How efficient are operating FPGA implementations produced by ASPLs vs. THDLs?"

Table 1 organizes the metrics used to address these questions in the three aforementioned categories. A detailed discussion of each follows.

Table 1. Categorization of Study Metrics.

Category	Metrics
Language Acquisition	Time required until competence is achieved
	Number of programs written until competence is achieved
	Number of Lines of Code written until competence is achieved
Programming Efficiency	Time required to produce verified operational code
	Time required to code

	Time required to test and debug
	Time required to optimize
Operational Efficiency	CLB usage, pre-optimization
	CLB usage, post-optimization
	Operating Frequency, pre-optimization
	Operating Frequency, post-optimization
	Throughput, pre-optimization
	Throughput, post-optimization

2.1 Language Acquisition

Language acquisition is measured as either time or experience until competence is achieved. Ascertaining when "competence is achieved" is mostly subjective. Until a standard tool for demonstrating competence can be developed and proven, only the programmer himself or peers can evaluate a level of competence. Assessing programmer competence fairly, accurately, and economically is notoriously difficult. [McNamara02]

Presumably, programmer competence improves with each program written, though many factors can influence mastery and the false appearance of mastery. For instance a programmer may code 50 programs over the course of a year, yet 45 of them might be variations of the other five. Has competence been achieved? Perhaps, but possibly his experience encompasses a mere 25% of the HLL constructs, and thus is generally incompetent with the other 75%.

Often, programmers become specialized in one application area and thus remain naïve regarding data structures and techniques useful in other applications. This should not be regarded as an obstacle to competence, since such a specialist is perfectly competent and achieving high productivity in his specialty. Thus, subjective classification of one's own level of competence into Low, Medium, or High competence is satisfactory.

2.2. Programming Efficiency

Programming efficiency refers to the time (or times) required for a programmer to reach production milestones. Overall, the important quantization is the time required to produce verified operational code. The measurement should be in terms of hours, though a useful hourly estimate can usually be derived from weeks or months consumed. This experiment proposes allocating the overall hours into three distinct categories to illuminate the extra value an HDL IDE offers to programmers. Thus programmers are asked to allocate hours

to the subtasks of coding, verification (test and debug), and optimization.

2.3. Operational Efficiency

Operating efficiency empirically characterizes the results of a programming effort. Three metrics of greatest use are the number of Configurable Logic Blocks (CLB) used, the theoretical operating frequency (i.e. the inverse of the time of the critical combinational logic path), and achievable throughput. These are both subdivided into pre- and post-optimizaition metrics because a common approach to FPGA programming is to first produce a design that works, and then to refine it to take advantage of parallelism and special device provision, such as Block Random Access Memory (RAM), multipliers, and Digital Signal Processing (DSP) circuits, etc.

The first pair measures chip resources employed to satisfy a design and affects the selection of FPGA chip for an application. Chips with larger CLB counts cost more. Programs that generate smaller CLB requirements allow for either smaller and less expensive (and potentially cooler) FPGAs or for increased functionality on the same chip. The critical time is the maximum amount of time required for all combinational logic to stabilize, and thus establishes the minimum clock period and maximum clock rate. All metrics can be improved substantially through optimization features offered by the HDL and by the skill and inventiveness of the programmer.

The first two metrics, CLB usage and operating frequency, are easy to capture from the place and route output produced during circuit synthesis. Throughput is determined by simple analysis or by testing after implementation.

3. Performance Comparison Experiments

Experimentation is chosen because suitable programming languages exist at reasonable cost and are currently used to produce commercial quality applications by a significant pool of programmers at various experience levels. In order to capture metrics from a large pool of programmers and application areas, a novel scheme is designed. FPGA programmers from all industries are invited to participate. To encourage standardization, programmers are registered and expected to read a description of experimental goals and guidelines, and particularly the self-assessed ranking of programming competency.

Participants are asked to attempt from one to three programs from common application areas common to FPGA endeavors, such as cryptography, DSP, genomics, and data compression, by following predefined program requirements. Programmers can either submit multiple programs using a single HDL or the same application accomplished once with a THDL and once again with an ASPL. The programming workload is meant to be moderately challenging, and without causing stress. In addition to providing examples from a selected suite, programmers are invited to submit measurements from any additional HDL endeavors.

Data is collected through a web-based application and reviewed by at least one other registrant (the Peer Reviewer). The observations and statistics are posted.

3.1 Factors and levels

Five factors are likely to affect the efficiencies of applications programmed using an ASPL vs a THDL: programmer ability, program complexity, programming language, application category and target device.

3.1.1 Programmer Ability

Programmer ability or competence is difficult to measure, and will be self assessed with confirmation by the Peer Reviewer. Programmer's rank their competence with each submitted program because it is likely to vary among the language and application choices. This factor may illuminate some valuable information regarding the speed at which programmers can become competent or proficient using an HDL. Low, Medium, and High levels are provided.

3.1.2 Program Complexity

Program complexity is another difficult factor to assess empirically. Participants can categorize their attempts into levels of Easy, Moderate, and Challenging. After a sufficiently large number of responses are received, then a cluster analysis can help identify appropriate breakpoints between categories to confirm if three levels is appropriate and whether independent programmers agree about application complexity.

3.1.3 Programming Language

Programming Language (and its accompanying integrated development environment) comprise the single most important factor under study. Submitted results are easily tagged with the specific category. Observations can be pooled at different levels, for instance, VHDL and Verilog results can be combined to create the THDL group; Impulse-C and Handel-C can be combined into the C subgroup and combined with JHDL and MyHDL in the larger ASPL group.

3.1.4 Application Category

Application category is controlled because different applications are likely to require different techniques and may illuminate particular strengths and weaknesses among HDLs. Carefully coding this predictor is important because it, is likely to be pooled with other applications as analyses prove the metric similarities between certain application categories.

3.1.5 Target Device

Target device is readily coded and lends itself to grouping, especially with similar devices offered by the same manufacturer. Different devices have different sizes, interconnection fabrics, clock and switching speeds, and special circuitry provisions. These can effect the Operating Metrics. Allocating results and forming conclusions on these differences is outside the scope of this proposed study, however, it may be advantageous to control variability among devices by normalizing chip usage and speed measurements as a percentage of device capacities.

3.2 Workload

The workload consists of a small group of specific programs from a standard collection of four programs from different application areas, and are expected to be of medium complexity; challenging, but not extraordinarily demanding. The collection is small and well defined to minimize effects from different functionality. However, metrics for any program can be submitted and the Programmer and Peer Reviewer can collaborate to assign an Application Category.

The period of study is expected initially to be about one year. Registrants can submit metrics for work completed before joining the study, and work, if desired, on additional programs chosen from the standard collection. A web page advertises the study, collects registrations and metrics, and displays results as they are compiled.

3.3 Example Case Study

A brief paper hints at the operational efficiency that can be achieved using Handel-C when compared with a similar programming attempt using VHDL [Mylonas02]. The paper examines several implementations of the Data Encryption Standard (DES) algorithm employing differing sophistication in pipelining and implementation of Look-Up Tables (LUTs).

They present a table of the operating efficiency metrics, reproduced here in Table 2. It indicates that the best Handel-C implementation requires about 20% more chip resources than the VHDL implementation but is about 30% faster.

Table 2. Operating Efficiency of a DES Implementation using VHDL vs. Handel-C.

Circuit	**No. of CLB Slices**	**Maximum Clock**
Pipelined DES VHDL	2,524	68.999MHz
Pipelined DES Handel-C Static Array LUTs	10,327	40.548MHz
Pipelined DES Handel-C Static ROM LUTs	3,813	74.145MHz
Pipelined DES Handel-C Celoxica Implementation	3,025	101MHz

The authors do not offer any guidance about their investment of time to produce their results. They conclude, "Handel-C is not as mature as VHDL. Great care has to be taken when describing circuits. Although working circuits can be produced from different descriptions, circuit size and speed can vary greatly."

4. Summary

A survey of literature reveals little evidence providing empirical justification for the selection of one language for HDL applications over another. In the author's experience, this choice is often made by the emotional case made by the most senior, vocal, or impassioned Lead Programmer or committee. Non-technical business reasons, such as cost and alignment with industry competitors, suppliers, and customers often exert greater influence regarding the selection once the barest case for technical suitability is made. the and , but even more important are qualities such as the ability to express and manage complex designs efficiently, and for the code to operate efficiently as well.

Little research is available comparing the value of ASPLs and THDLs in either commercial or academic settings. This may be the result of marketing decisions informally adopted by language developers to conceal

their deficiencies, or it may be that there is difficulty in capturing quantitative data for meaningful analysis. The small bits of information found are either of single programming endeavors or of anecdotal quality.

This paper describes the impetus and an experiment for studying the relative efficiencies between ASPLs and THDLs so that decision makers can more effectively address their genuine and long term productivity concerns. Tools are evolving to assist the hardware programming industry overcome productivity bottlenecks. It is time to employ performance engineering methods to discover and present quality information.

References

[AMI06] Active Motif, Inc., "The Promise of FPGA Programming", 2006. http://www.timelogic.com/technology_fpga.html]

[Gates07] Gates, W. Presenting at COMDEX, 2007. http://silversunshine285.blogspot.com/2007/02/if-gm-had-advanced-at-same-rate-as.html

[McNamara02] McNamara, R. "Concept mapping for Introductory Programming", November 2002. http://www.csse.monash.edu.au/hons/projects/2002/Robyn.Mcnamara/index.html

[Mylonas02] Mylonas, M; Holding, D.J.; Blow, K.J.. "DES Developed in Handel-C." Aston University. http://www.ee.ucl.ac.uk/lcs/papers2002/LCS057.pdf

[Pellerin08] Pellerin, D. Blog: "Programmable Arrays are more than Reconfigurable HDL Executers." June 12, 2008. http://fpgacomputing.blogspot.com/2008/06/programmable-arrays-are-more-than.html

[Storaasli08] Storaasli, O., Strenski, D. "Accelerating Genome Sequencing 100 – 1000x with FPGAs, Published after May 2007. http://private.ecit.qub.ac.uk/MSRC/Wednesday_Abstracts/Storaasli_OakRidge.pdf

[Verilog08] Verilog. http://en.wikipedia.org/wiki/Verilog

[VHDL08] VHDL. http://en.wikipedia.org/wiki/VHDL

[Wirth95] Wirth, N. "A plea for leaner software", Computer, Volume 28, Issue 2, February, 1995, pages 64-68.]

Definitions

ASPL - Adapted Sequential Programming Language. A variation of a common language such as C or Java enabling it to meet the needs of FPGA programmers.

ASIC - Application Specific Integrated Circuit. A special electronic circuit implemented on a single chip to perform a specific, though arbitrarily complex, operation. Generally, an ASIC cannot be reconfigured to perform a different operation.

CLB - Configurable Logic Block. The smallest programmable element in an FPGA that performs a logical function. Interconnections among CLBs are also programmable, but simply route signals and do not perform any logic operations.

FPGA - Field Programmable Gate Array. A special electronic circuit implemented on a single chip designed so that it can be configured or reconfigured to perform digital logic operations of vastly different kinds by programmers who are unrelated to the manufacturer.

HDL - Hardware Design Language. A group of HLLs used to specify the operation of an ASIC or FPGA, among which are found VHDL and Verilog.

HLL - High Level Language. A computer programming language that is effective for allowing human programmers to describe computer operations using abstract rather than detailed instructions.

IOB - Input Output Block. Specialized circuit elements on FPGA chips that enable interfaces with external devices.

THDL - Traditional HDL. A collective reference to Verilog and VHDL.

VHDL - VLSI HDL. A widely used HDL developed for the DoD, and based on the Ada programming language.

VLSI - Very Large Scale Integrated Circuit. A single integrated circuit with populations of more than 5000 transistors and other components (though this number is inadequately small for describing modern FPGA chips).

Last modified on November 24, 2008
This and other papers on latest advances in performance analysis are available on line at
http://www.cse.wustl.edu/~jain/cse567-08/index.html


Back to Raj Jain's Home Page