Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCSE-2006-10

2006-01-01

Automatic Application-Specific Customization of Softcore Processor Microarchitecture, Masters Thesis, May 2006

Shobana Padmanabhan

Applications for constrained embedded systems are subject to strict runtime and resource utilization bounds. With soft core processors, application developers can customize the processor for their application, constrained by available hardware resources but aimed at high application performance. The more reconfigurable the processor is, the more options the application developers will have for customization and hence increased potential for improving application performance. However, such customization entails developing in-depth familiarity with all the parameters, in order to configure them effectively. This is typically infeasible, given the tight time-to-market pressure on the developers. Alternatively, developers could explore all possible configurations, but... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Padmanabhan, Shobana, " Automatic Application-Specific Customization of Softcore Processor Microarchitecture, Masters Thesis, May 2006" Report Number: WUCSE-2006-10 (2006). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/159

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/159

Automatic Application-Specific Customization of Softcore Processor Microarchitecture, Masters Thesis, May 2006

Shobana Padmanabhan

Complete Abstract:

Applications for constrained embedded systems are subject to strict runtime and resource utilization bounds. With soft core processors, application developers can customize the processor for their application, constrained by available hardware resources but aimed at high application performance. The more reconfigurable the processor is, the more options the application developers will have for customization and hence increased potential for improving application performance. However, such customization entails developing in-depth familiarity with all the parameters, in order to configure them effectively. This is typically infeasible, given the tight time-to-market pressure on the developers. Alternatively, developers could explore all possible configurations, but being exponential, this is infeasible even given only tens of parameters. This thesis presents an approach based on an assumption of parameter independence, for automatic microarchitecture customization. This approach is linear with the number of parameter values and hence, feasible and scalable. For the dimensions that we customize, namely application runtime and hardware resources, we formulate their costs as a constrained binary integer nonlinear optimization program. Though the results are not guaranteed to be optimal, we find they are near-optimal in practice. Our technique itself is general and can be applied to other design-space exploration problems.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2006-10

Automatic Application-Specific Customization of Softcore Processor Microarchitecture, Masters Thesis, May 2006

Authors: Shobana Padmanabhan

Corresponding Author: shobanaPadmanabhan@yahoo.com

Web Page: http://www.arl.wustl.edu/~sp3/

Abstract: Applications for constrained embedded systems are subject to strict runtime and resource utilization bounds. With soft core processors, application developers can customize the processor for their application, constrained by available hardware resources but aimed at high application performance.

The more reconfigurable the processor is, the more options the application developers will have for customization and hence increased potential for improving application performance. However, such customization entails developing in-depth familiarity with all the parameters, in order to configure them effectively. This is typically infeasible, given the tight time-to-market pressure on the developers. Alternatively, developers could explore all possible configurations, but being exponential, this is infeasible even given only tens of parameters.

This thesis presents an approach based on an assumption of parameter independence, for automatic microarchitecture customization. This approach is linear with the number of parameter values and hence, feasible and scalable. For the dimensions that we customize, namely application runtime and hardware

Type of Report: Other

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AUTOMATIC APPLICATION-SPECIFIC CUSTOMIZATION OF SOFTCORE PROCESSOR MICROARCHITECTURE

by

Shobana Padmanabhan

Prepared under the direction of Professors Ron K. Cytron and John W. Lockwood

---

A thesis presented to the Henry Edwin Sever Graduate School of
Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

May 2006

Saint Louis, Missouri

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

AUTOMATIC APPLICATION-SPECIFIC CUSTOMIZATION OF SOFTCORE PROCESSOR MICROARCHITECTURE

by

Shobana Padmanabhan

ADVISORS: Ron K. Cytron and John W. Lockwood

May 2006

Saint Louis, Missouri

Applications for constrained embedded systems are subject to strict runtime and resource utilization bounds. With soft core processors, application developers can customize the processor for their application, constrained by available hardware resources but aimed at high application performance.

The more reconfigurable the processor is, the more options the application developers will have for customization and hence increased potential for improving application performance. However, such customization entails developing in-depth familiarity with all the parameters, in order to configure them effectively. This is typically infeasible, given the tight time-to-market pressure on the developers. Alternatively, developers could explore all possible configurations, but being exponential, this is infeasible even given only tens of parameters.

This thesis presents an approach based on an assumption of parameter independence, for automatic microarchitecture customization. This approach is linear with the number of parameter values and hence, feasible and scalable. For the dimensions that we customize, namely application runtime and hardware resources, we formulate their costs as a constrained binary integer nonlinear optimization program. Though the results are not guaranteed to be optimal, we find they are near-optimal in practice. Our technique itself is general and can be applied to other design-space exploration problems.

Dedicated to my mother Saraswathi and late father Padmanabhan.

Contents

List of Figures

Acknowledgments

My foremost acknowledgements go to my advisors Dr. John W. Lockwood and Dr. Ron K. Cytron.

I thank John for giving me the opportunity to work under him as well on this project, which I have thoroughly enjoyed working on. I also thank him for giving me guidance, support, and freedom, whenever I needed them and to the extent I needed them. He is one of the most flexible people I have known.

Ron is an ideal advisor, very knowledged and very caring. I am very grateful for being advised and mentored by him in my research as well as in publishing the results. From him, I have also learned an amazing amount about making effective presentations, seeing the "intrinsic value" of one's work, and chasing own's dreams.

I am equally indebted to Dr. Roger Chamberlain, who have also been my mentor, and thesis committee member. Roger's affability makes him one of a kind. I have personally benefited a great deal from him being available always with suggestions, or a kind word. I sincerely appreciate his patience and attention to detail during the many discussions we have had on evaluation techniques.

I acknowledge Dr. Jason Fritts, Dr. Weixiong Zhang, Dr. Sally Goldman, and Sharlee Climer for their discussions on optimization algorithms.

My special thanks go to Phillip Jones, my "go-to" person when I have a question. I also gratefully acknowledge the help of Liquid Architecture team with the Liquid platform. In particular, I thank Praveen Krishnamurthy, Richard Hough, Paul Spiegel, Scott Friedman, Justin Thiel and Luke Rinard.

I deeply appreciate the illuminating discussions with Sarang Dharmapurikar and Jing Lu of Reconfigurable Network Group (formerly, FPX). For their support with FPX platform and other enjoyable experiences, I thank all other members, Todd Sproull, Chris Zuver, Dave Schuehler, Dave Lim, Qian Wan, Abdel Rigumye, James Moscola, Mike Attig, Bharath Madhusudan, Chris Neely, Adam Covington, Chip Kastner, Andrew Levine, Haoyu Song, Jeff Mitchell, Jack Meier and Young Cho. I also thank all members of Applied Research

Lab, in particular, Manfred Georg, Christoph Jechlitschek, Sailesh Kumar and Michela Becchi, and my other friends in CSE department, in particular, Priyanka Thakan, Roopa Pundaleeka, Vinayak Joshi and Seema Datar, for the intellectual and convivial atmosphere.

I would also like to acknowledge Jean Grothe, Myrna Harbison, Peggy Fuller, Sharon Matlock and Stella Sung for their administrative support.

My sincere gratitude goes to my former manager Matt Stevens at Sun Microsystems, for inspiring and helping me to return to academic life. Of course, I would not be here but for the support of my parents, my sisters Jamuna Dhinakaran, Kalpana Kashyap and their families, and my friends, Gayathri Ramanan, Lakshmi Prasanna, Shanthi Nangunuri, Lalitha Krishnamurthy, Bhuvana Suresh, Smitha and Aby Mathew and Gina Wright. Everyone else who is slipping my mind at the moment, my apologies and acknowledgements!

This work is supported by NSF under grant CCR-0313203.

Shobana Padmanabhan

Washington University in Saint Louis
May 2006

Chapter 1

Introduction

Application performance can be improved in many ways — by having a processor that matches the application's requirements more closely, a compiler that generates more efficient code or a better implementation of the algorithms in the application. Not only can we improve these components individually but we can also improve their interfaces to improve application performance, as depicted in Figure 1.1. This thesis focusses on improving the processor to improve application runtime.

In addition to the demanding performance requirements, many embedded applications have more constraints than general (desktop) applications along a number of other dimensions. These dimensions include constraints on energy, power, memory and other hardware resources. Conventionally, the hardware systems that host embedded applications are often dedicated to that particular application, and need not work well on *all* possible applications. As a result, there has been significant interest in the ability to build Application-Specific Integrated Circuits (ASICs). In these custom-hardware platforms, the hardware design matches the needs of the application closely.

ASIC (ASIC) development is often infeasible due to fabrication cost and time-to-market considerations, prompting developers of embedded systems to consider alternatives. First, there are many off-the-shelf processors that have been optimized for embedded applications. The challenge here is making an appropriate choice among a large number of alternatives. Second, an off-the-shelf processor (processor core) can be paired with a custom logic co-processor, constructed using either ASIC or **Field Programmable Gate Array** (FPGA) technology. The co-processor then executes some fraction of the embedded application [6, 12, 40, 44]. Integrating reconfigurable logic into host processors is also an

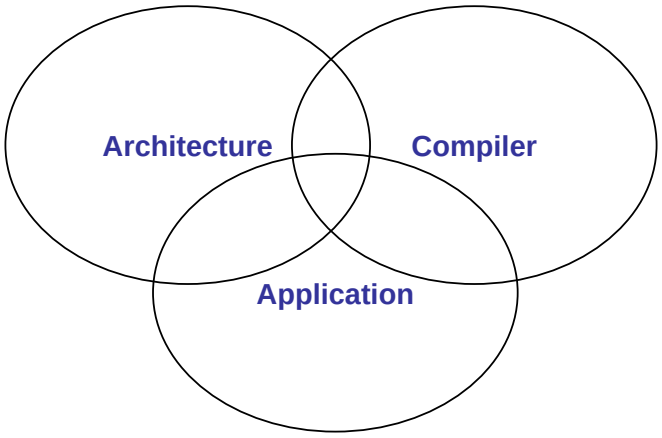

Figure 1.1: Components of application performance

active area of academic research [17, 23, 38]. Third, there are a number of processors available now that can be at least partially configured at the **Instruction Set Architecture** (ISA) level, such as Tensilica [52] and ARC [4]. While the systems from ARC are configurable at fabrication time, Stretch [46] makes the Tensilica processor reconfigurable at execution time through the use of FPGA technology on the chip.

Besides the ISA which is the external interface to which applications are coded, a processor's *microarchitecture* can also be configured, in an attempt to improve application performance. Microarchitecture configuration can be done while (still) designing an ASIC or for a soft core processor. Soft core processors are discussed in Section 1.1. Microarchitecture is more internal to the processor. Examples of microarchitecture subsystems include cache and integer unit. Examples of some parameters in these subsystems include cache size and number of register windows. For changes in these parameters, compiler changes are not required. In this thesis, we explore reconfiguring (customizing) the microarchitecture parameters to meet the requirements of a given application more closely, in terms of application runtime and resource constraints.

1.1 Soft Core Processors

Processor customization, both in terms of ISA and microarchitecture, is possible with soft core processors, also called configurable cores, instantiated on reconfigurable hardware.

These cores are basically general purpose processors with extant toolchain support (compilers). Further, the soft core processors are typically highly parameterized; therefore, they can be customized to a large extent for the specific requirements of a given application in terms of runtime, hardware resource usage, energy consumption, and power dissipation. Examples of soft-core processors include Xtensa from Tensilica [52], Microblaze from Xilinx [56], ARC [4] and LEON used by European Space Agency [20].

Tensilica's Xtensa RISC processor can be configured using Stretch Instruction Set Extension Fabric (ISEF) [46] whereby users can define extensions to the instruction set architecture in their C/C++ applications. Options include 16 and 24 bit instructions; aligned load and store of 8, 16, 32, 64, and 128 bits, unaligned load and store of up to 128 bits; variable byte streaming I/O; and up to 32 bits variable bit streaming I/O.

MicroBlaze hardware options and configurable blocks include the following [56]: hardware barrel shifter, hardware divider, machine status set and clear instructions, hardware exception support, pattern compare instructions, floating-point unit, hardware multiplier enable, hardware debug logic cache and cache interface, data cache, instruction cache, instruction-side Xilinx cache link, data-side Xilinx cache link bus infrastructure, data-side on-chip peripheral bus, instruction-side on-chip peripheral bus, data-side local memory bus, instruction-side local memory bus, and fast simplex link (FSL). For hardware acceleration, Microblaze uses a co-processor rather than user-defined instructions. The low latency FSL can connect up to eight co-processors. For development, Xilinx offers the Embedded Development Kit, which includes the MicroBlaze core, a library of peripheral cores and common software platforms such as device drivers and protocol stacks.

ARC provides a graphical interface called ARChitect [4], for configuring its processor. The tool allows application developers to configure the instruction set, interrupts, instruction and data caches (associativity, cache locking), memory subsystem, DSP features, number of registers, and custom condition codes. Users can also add peripherals, such as an Ethernet media-access controller and 32-bit timers for real-time processing. Another option is clock gating, a power-saving feature that shuts down parts of the processor when they are not needed. The ARCtangent-A4 processor's base-case instruction set includes all the fundamental arithmetic, logical, load/store, and branch/jump operations required for a typical embedded application. By using the ARChitect tool, designers can select from a library

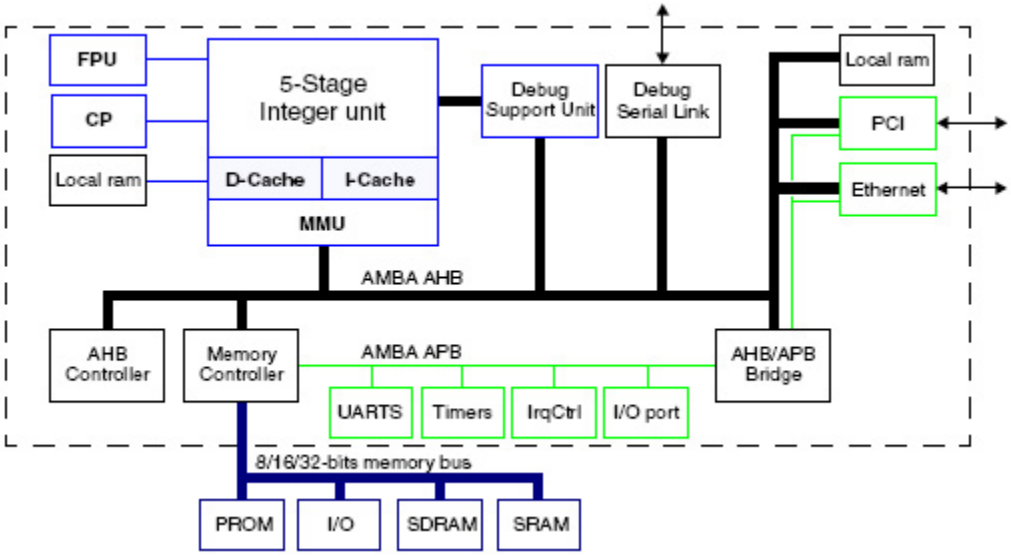

Figure 1.2: High-level architecture of LEON soft core processor (courtesy LEON manual)

of additional instructions and features. Examples include a hardware barrel shifter and associated instructions, and two different hardware multipliers.

LEON [31] **Very High Speed Integrated Circuit (VHSIC) Hardware Description Language** (VHDL) model implements a 32-bit processor conforming to the IEEE-1754 (**Scalable Processor ARChitecture** (SPARC) V8) architecture. Despite being spelled with all upper case, LEON is not an acronym. LEON is designed for embedded applications with the following features on-chip: separate instruction and data caches, hardware multiplier and divider, interrupt controller, debug support unit with trace buffer, two 24-bit timers, two UARTs, power-down function, watchdog, 16-bit I/O port, flexible memory controller, Ethernet MAC and **Peripheral Component Interconnect** (PCI) interface. These features are shown in Figure 1.2. New modules can easily be added using the on-chip **Advanced Microcontroller Bus Architecture** (AMBA) **AMBA Hi-speed Bus** (AHB)/**AMBA Peripheral Bus** (APB) buses. The VHDL model is fully synthesizable with most synthesis tools and can be implemented on both FPGA and ASIC. Simulation can be done with all VHDL-87 compliant simulators. Through the model's configuration record, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation.

LEON is an open source soft core and therefore we use it in our experiments on application-specific microarchitecture customization. For customizing the parameters, LEON provides a graphical interface as well as a VHDL interface. LEON is highly configurable and is highly parameterized. Its parameters are described in detail in Section 2.4.

1.2 Motivation

The parameters of a soft core processor can be thought of as knobs that can be turned to select appropriate values. The idea is to turn those knobs to improve application runtime, subject to the constraints on resources, power, energy and all other dimensions that are being optimized. As mentioned in Section 1, in this thesis, we restrict ourselves to customizing microarchitecture parameters. Henceforth, by parameters we refer to the microarchitecture parameters. The values of parameters that give the best tradeoff constitute the optimal configuration.

Figure 1.3 shows the names of some knobs, cache size, a particular associativity and number of register windows. However, not all parameters affect application's runtime equally. Similarly, different parameters affect hardware resource utilization in varying measures. Therefore, to obtain the optimal configuration, all parameters must be considered simultaneously for their effects on all the dimensions that are being optimized or constrained. Hence, having more customizable parameters in a processor results in more options for customization on one hand, but on the other hand, makes the simultaneous search through them very complex.

LEON, the prototype core used in this thesis, is highly parameterized. Figure D.1 shows that there are 8 systems in LEON and Figure 2.18 shows that there are 95 parameters across these systems, with 246 values that are customizable. Section 2.4 describes them in detail. The graphical interface for the 8 systems and their subsystems are shown in Appendix D.

To tune the processor parameters manually, application developers need to know the impact of the different parameters on application runtime and other dimensions being constrained

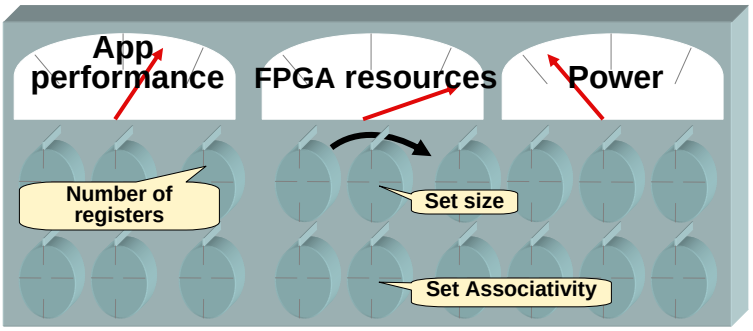

Figure 1.3: Tuning configurable processors

or optimized, such as hardware resource utilization, energy consumption and power dissipation. However, application developers may not be deeply familiar with the entire application. Further, they are neither architecture experts nor do they have the time to develop such expertise, because of the typical time-to-market pressure on them.

Rather than searching for the optimal configuration manually, application developers could search through all possible processor configurations exhaustively and simply select the best one. There are two factors that determine the feasibility of such an approach: the number of processor configurations to search and the time it takes to measure the dimensions that are being optimized or constrained.

The number of processor configurations is exponential with the number of parameter values. For LEON, as we will see in Section 2.4, there are a maximum of 246 parameter values that can be customized. This results in 2^{246} configurations. Clearly, the number of configurations is too large to be searched exhaustively in a feasible timeframe.

For measuring application runtime, we prefer to execute the application directly on the processor and measure the actual runtime nonintrusively. Section 1.4 explains why this approach is preferable. To execute application directly on the processor, we first need to *build* the different processor configurations from the source VHDL. Build process comprises the tasks of compiling, synthesizing, building (which in turn involves checking timing specifications, and expanded design), mapping, place-and-routing (checking physical constraints), meeting timing constraints, tracing, and finally generating *bitfile*. For synthesis, we use Synplify Pro from Synplicity [50] and for all other tasks, we use Xilinx tools [55]. With all these tasks, building a LEON processor configuration takes at least 30

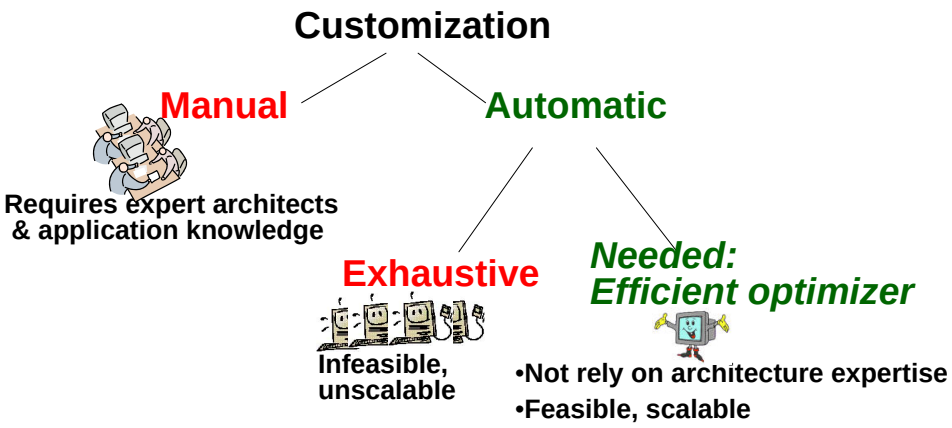

Figure 1.4: Need for efficient optimizer

minutes even on modern processors (computers). The long build time restricts the number of configurations that can be built in a feasible amount of time.

Figure 1.4 summarizes our discussions so far and shows the need for an optimization technique that is feasible and scalable. We prefer to automate the customization process so that application developers' time on this is minimized.

1.3 Thesis Objectives

Soft core processors are general purpose processors instantiated on reconfigurable hardware and hence reconfigurable. We seek to customize the microarchitecture of a soft core processor to meet a given application's requirements and constraints closely. As we saw in Sections 1.2 and 1.4, there exists no efficient technique to do this automatically over the space of *all* relevant microarchitecture parameters. This thesis aims to fill the gap by developing an automatic optimization technique that:

- Considers all feasible microarchitecture parameters for the application-specific customization.
- Develops an optimization technique that is feasible and scalable.

- Automates this technique so as to involve application developers minimally in the customization process.
- Executes applications directly on the processor configurations to obtain actual runtimes. This involves actually building the processor configurations being considered.
- Evaluates this technique by actually customizing a well known soft core processor for some substantive benchmarks.

1.4 Related Work

There has been significant work centered around the idea of customizing a processor for a particular application or application set. Arnold and Corporaal [5] describe techniques for compilation given the availability of special function units. Atasu *et al.* [29] describe the design of instruction set extensions for flexible ISA systems. Choi *et al.* [15] examine a constrained application space in their instruction set extensions for DSP systems. Gschwind [22] uses both scientific computations as well as Prolog programs as targets for his instruction set extensions. Keller and Brebner [28] analyze tradeoffs between chip area for hardware accelerators and sequential execution of software on a processor.

Gupta *et al*. [51] feature a compiler that supports performance-model guided inclusion or exclusion of four functional units of **multiply-accumulate** (MAC), floating point, multi-ported memory and pipelined vs. non-pipelined memory unit. Systems that use exhaustive search for the exploration of the architecture parameter space are described in [36, 30, 42].

Heuristic design-space exploration for application-specific processors is another extant approach [18]. Pruning techniques are used to diminish the size of the necessary search space to find a Pareto-optimal design solution. Pareto optimality, is a central theory in economics with applications in game theory and engineering. An allocation of resources is Pareto optimal when no further Pareto improvements are possible. A Pareto improvement is an alternative allocation of resources that can make at least one of the individuals involved better off, without making any other individual worse.

Yet another approach is to use a combination of analytic performance models and simulation-based performance models to guide the exploration of the design search space [8]. Here,

the specific application is in the area of sensor networks. Analytic models are used early, when a large design space is narrowed down to a "manageable set of good designs" and simulation-based models provide greater detail on the performance of specific candidate designs.

The AutoTIE system [21] is a development tool from Tensilica that assists in the instruction set selection for Tensilica processors. This tool exploits profile data collected from executions of an application on the base instruction set to guide the inclusion or exclusion of candidate new instructions.

Some people perform analytical (hierarchical) searching of parameters in their own dimensions, with some full parameter exploration to avoid local minimum, for tuning multi-level cache for low-energy embedded systems [3].

A different approach explores design options of instruction and data caches, branch predictor, and multiplier, by dividing the search space into piece-wise linear models that are evaluated using integer linear programming [43] .

There are two main problems with most of the approaches mentioned so far. The first problem is that many approaches consider only a few parameters for customization or consider only a specific subsystem (such as cache) for a specific purpose (such as energy conservation). Such approaches do not scale well for the large number of parameters in a typical soft core processor. The second problem is the way application runtime is estimated using analytical models or measured using simulators, as described below.

Performance Measurement

Analytic models can provide the quickest estimations of application performance, and such models are often derived directly from source code. Examples of the use of analytic models include: an approach for the analytical modeling of runtime, idealized to the extent that cache behavior is not included [9]; and a classic paper on estimating software performance in a codesign environment, which reports accuracy of about $\pm$20% [49]. However, for the purpose of application performance improvement, $\pm$20% is a wide deviation. These inaccurate predictions are due to the simplifying assumptions that are necessary to make analysis tractable and are notoriously common when analytic models are used. Moreover,

application models often require sophisticated knowledge of the application itself. By contrast, simulation and the direct execution we use are both “black box” approaches that do not require knowledge of application implementation.

The method normally used to improve accuracy beyond modeling is *simulation*. Simulation toolsets commonly used include: SimpleScalar [7], IMPACT [14], and SimOS [39]. Given the long runtimes associated with simulation modeling, it is common practice to limit the simulation execution to only a single application run, not including the OS and its associated performance impact. SimOS does support modeling of the OS, but requires the simulation user manage the time/accuracy tradeoffs inherent in simulating such a large complex system. In addition, simulation often suffers from uncertainty about conformance to the underlying architecture.

Performance monitoring in a relatively non-intrusive manner using hardware mechanisms built into the processor is an idea that is supported on a number of modern systems. Sprunt [45] describes the specific support built into the Pentium 4 for exactly this purpose. In an attempt to generalize the availability of these resources in a processor-independent manner, the Performance Application Programmer Interface (PAPI) [11] has been designed to provide a processor-independent access path to counters built into many modern processors. There are a number of practical difficulties with this approach, however, as described in [16]. First, the specific semantics of each mechanism are often documented insufficiently by the manufacturer, even to the point where similarly named items on different systems have subtly different meanings. Second, there are a number of items of interest, such as cache behavior, that can not be profiled via these mechanisms.

We avoid all the above-mentioned issues by exploiting the reconfigurable nature of FPGA to profile an application executing directly on a soft core processor. We call this profiler the “statistics module” [26] and this is part of our Liquid architecture platform [37]. The statistics module uses a hardware-based, non-intrusive profiler to count the number of clock cycles taken by the application. Because it gives accurate runtime measures, we use this for our work here.

1.5 Contributions

The three main contributions of work in this thesis are:

- Development of an automatic optimization technique to customize soft core processor microarchitecture per application. This involves formulating the problem as a Binary Integer Nonlinear Program and solving for an optimal solution. To keep this approach feasible and scalable, we assume that the microarchitecture parameters are independent of each other.
- Evaluation of the technique, including the assumption of parameter independence, by customizing LEON processor for some substantive applications. Specifically, the thesis answers the following research questions:
  - What is the effect of the parameter-independence assumption?
  - How much improvement can we get from application-specific microarchitecture customization?
  - Is customization indeed application-specific?
- Development of a software controller for the Liquid architecture platform. The controller provides a web-based as well as command-line interface to a hardware controller that controls LEON and the execution of applications on LEON.

1.6 Thesis Outline

Chapter 2 introduces the Liquid architecture platform and its building blocks: the soft core processor LEON, instantiated on FPGA and a software controller to control it. It then introduces **Binary Integer Nonlinear Program** (BINP) and compares it with the alternatives that were considered. Finally, the chapter describes the different benchmarks used in our experiments.

Chapter 3 provides an overview of the optimization technique.

Chapter 4 formulates the microarchitecture customization problem as Nonlinear Binary Integer Programming, using LEON as an example.

Chapter 5 evaluates the optimization technique and Chapter 6 presents customization results for the different benchmarks. In addition, Chapter 6 also answers the research questions posed in Section 1.5.

Chapter 7 summarizes our results and provides the final analysis. It also contains suggestions for future work in this area.

Chapter 2

Background

2.1 Liquid Architecture Platform

Liquid architecture platform was developed with a goal of measuring and improving application performance, by providing an easily and efficiently reconfigurable architecture, along with software support to expedite its use. This section presents an overview of the platform design [37]. The main components of the platform are the Liquid module, the soft core LEON processor, memory interfaces, cross compiler, control software and statistics module and they are briefly described below.

The Liquid architecture system was implemented as an extensible hardware module on the **Field-programmable Port Extender** (FPX) platform [32, 33]. An overview of the FPX platform is presented in Section 2.2.

Liquid Module

Figure 2.1 shows the high level architecture of the Liquid module. The module is fit within Layered Protocol Wrappers [10], for Internet connectivity. The wrappers format incoming and outgoing data as **User Datagram Protocol** (UDP) / **Internet Protocol** (IP) network packets. A Control Packet Processor (CPP) routes Internet traffic that contains LEON specific packets (command codes) to the LEON controller (leon_ctrl). The different control packet formats used by the Liquid system are presented in Appendix A. The leon_ctrl entity uses these command codes to direct the LEON processor to restart and execute application and to read and write the contents of the external memory that the LEON processor uses

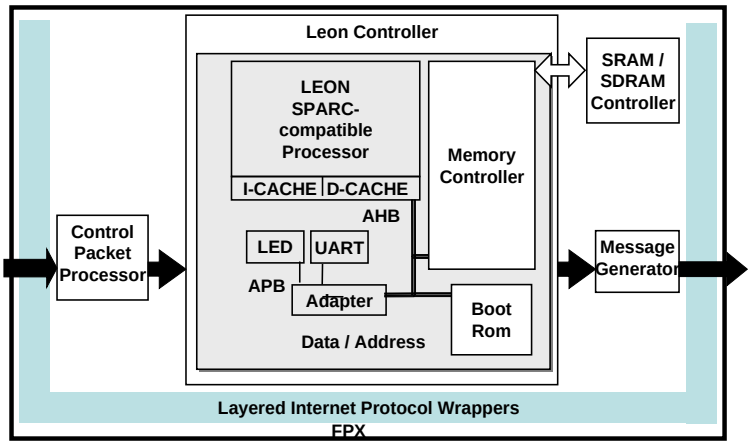

Figure 2.1: High-level architecture of Liquid system

for instruction and data storage. Finally, the Message Generator is used to send IP packets in response to receiving a subset of the command codes such as read memory and check LEON status.

LEON Processor

Figure 2.1 also illustrates some of the main components of the base LEON processor [20]. As can be seen in Figure 2.1, the processor has fairly sophisticated features such as instruction and data caches, support for the full SPARC V8 instruction set, 5-stage pipeline and separate buses for high-speed memory access (AHB) and low-speed peripheral control (APB).

For inclusion in the Liquid system, it was necessary to modify portions of LEON to interface with the FPX platform. One such modification was to change the Boot ROM, such that the processor begins execution of user code using the FPX platform's **Static Random-Access Memory** (SRAM). All other components necessary to implement the interface between the processor, memory, and user were implemented outside of the base processor system. More details on the parameterization of LEON is provided in Section 2.4.

Memory Interface

LEON comes with memory interfaces [31] for programmable ROM, SRAM, **Synchronous Dynamic Random-Access Memory** (SDRAM), and memory-mapped IO devices. LEON uses the standard AMBA bus to connect the processor core to its peripheral devices [2]. The memory controller which comes as part of the LEON package, acts as a slave on the AHB and accesses data from a 2 GB address space. We are currently using the SRAM for our experiments although support for SDRAM has been partially developed. Accesses to the 2 MByte SRAM memory are always performed on a 32-bit word. Memory reads take 4 clock cycles and memory writes can happen on on every clock cycle.

Cross Compiler

Compilers and kernels for SPARC V8 can be used with LEON since LEON is SPARC V8 compliant [20]. For initial software development, Gaisler Research distributes LECCS, a free C/C++ cross-compiler system based on *gcc* and the RTEMS real-time kernel. LECCS allows cross-compilation of single or multi-threaded C and C++ applications for both LEON. Using the *gdb* debugger, it is possible to perform source-level symbolic debugging, either on a simulator or using real target hardware.

Liquid Control Software

The web-based control software provides an interface to load compiled instructions over the Internet into LEON's memory. The different components of the control software system are shown in Figure 2.2. When users submit a request from the web interface, the request is received by a Java servlet [47] running on an Apache Tomcat server [48]. The servlet creates UDP (IP) control packets and sends them to the Liquid module, at a specified destination IP and port. It then waits for a response and handles the display of the response, or errors, if any. The commands currently supported by control software are:

- LEON status - to check if LEON has started up
- Load program - to load a program into memory, at a specific address

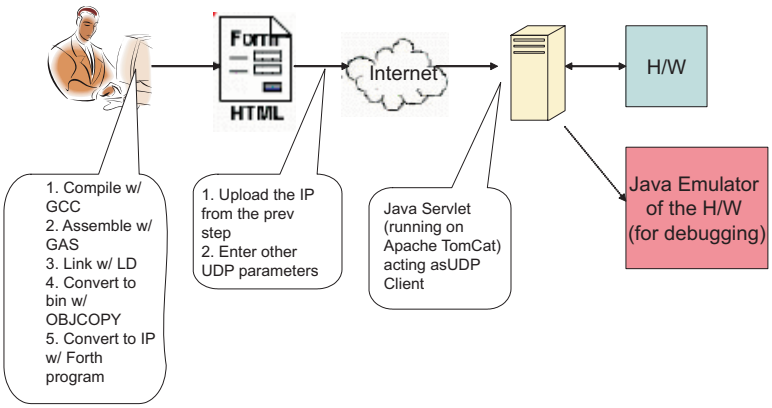

Figure 2.2: Control software toolchain

- Start LEON - to instruct LEON to execute the program that has been loaded into memory at a given address
- Read memory - to read a specified number of bytes at a given address. This can be used to verify the program that was loaded, or to read the results of a program which writes results to memory
- Get statistics - to get program execution time (number of hardware clock cycles) and other microarchitecture statistics such as cache statistics (number of cache reads, writes, hits, misses), etc.
- Reset LEON - to reset LEON

Statistics Module

The statistics module implemented by the Liquid architecture platform is a non-intrusive, hardware-based profiler, to profile application performance at or above the processor microarchitecture layer. The module is parameterized so that software application developers can configure it, over the Internet, for the microarchitecture parameters and application functions that they are interested in profiling. Address ranges of the software functions are obtained from the software application's map file generated during application compilation.

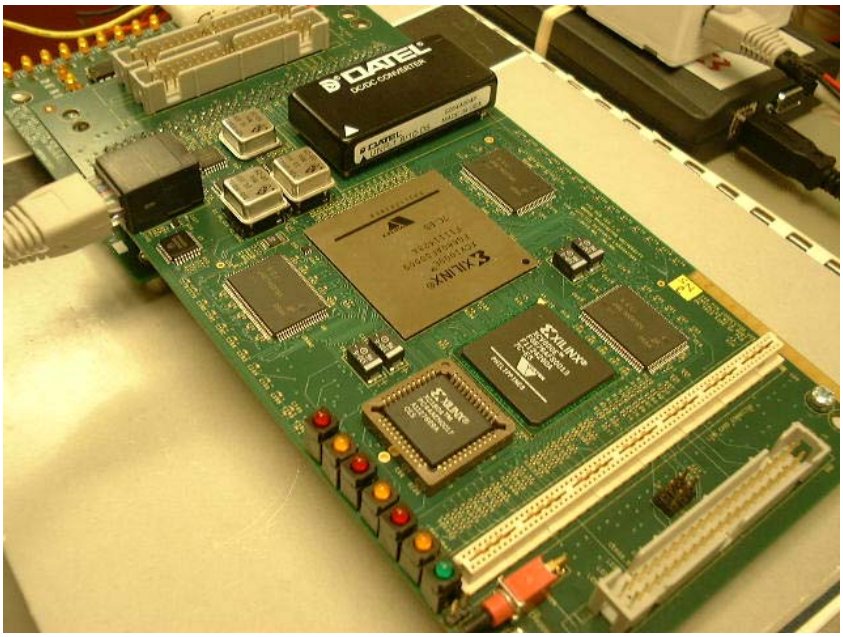

Figure 2.3: FPX Platform

2.2 FPX Platform

The FPX platform was developed by Washington University's Reconfigurable Network Group, formerly called FPX Group [34]. The platform provides an environment where a circuit implemented in FPGA hardware can be interfaced with SRAM, SDRAM, and high speed network interfaces. Hardware modules on the FPX can use some or all of a large Xilinx Virtex XCV2000E FPGA to implement a logic function [25]. By using the FPX platform, resulting hardware modules can be rapidly implemented, deployed and tested with live data [32].

The Liquid architecture system leverages FPX platform to evaluate customizations of the soft core processor and to control experiments remotely over the Internet.

2.3 FPGA

A field-programmable gate array (FPGA) is a large-scale integrated circuit that can be "programmed" (in the "field") even after it is manufactured. The term "gate array" refers to the elements of gates and flip flops that make the reprogramming possible.

FPGAs, irrespective of their sizes and features, are composed of small blocks of memory structured as programmable logic. The blocks are arranged in a grid and interconnected using wires. The blocks consist of lookup tables (LUTs) of n binary inputs (typically, $n = 4$), one or two 1-bit registers and additional logic elements such as multiplexers. These LUTs can implement any combinational function of their inputs. The exact structure of the LUTs is vendor specific. In addition, specialized logic blocks may be found at the periphery of the devices to provide programmable input and output capabilities. FPGAs also typically incorporate on-chip memory resources implemented with blocks of static **Random Access Memory** (RAM) called BlockRAM (BRAM). BRAMs are combined to implement large on-chip memories such as cache and buffers [41, 19].

2.4 LEON Parameterization

This section explores the reconfigurable subsystems and parameters of LEON. As shown below, LEON is highly parameterized. LEON distribution provides a graphical user interface to view and configure the parameters. The interface for configuring the different subsystems are shown in Appendix D. The values for the parameters get set in device.vhd, in LEON distribution. The default parameter settings in LEON distribution are presented in Appendix B. Their descriptions below are based on LEON manual [31].

The systems available for reconfiguration in LEON are listed below. Each system is explored in detail in the following sections.

1. Processor
2. Memory controller
3. AMBA bus
4. Peripherals
5. Synthesis options
6. Clock generation options
7. Boot options
8. VHDL debugging

2.4.1 Processor System

LEON processor consists of the following subsystems:

1. Cache
2. Integer Unit
3. Floating-point Unit

4. Memory Management Unit
5. Co-processor
6. Debug Support Unit

Cache subsystem

Figure 2.4 lists the parameters of cache subsystem. LEON uses Harvard cache architecture in that it has separate instruction and data caches and they can be configured independently.

Cache size is parameterized in terms of number of sets and size of each set. Valid settings for the cache set size are 1 - 64 KByte, and must be a power of 2. However, the cache size of 64KB requires a total number of 213 BRAM which is 33% more than what is available on our FPGA. The line size may be 4 - 8 words per line. Valid settings for the number of sets are 1 - 4 (2 if LRR algorithm is selected). Replacement algorithm may be random, LRR (Least Recently Replaced) or LRU (Least Recently Used).

The *dlock* and *ilock* fields enable cache locking for the data and instruction caches respectively. However, application code needs to make use of these but changing application code is beyond the scope of our customization. The *drfast* field enables parallel logic to improve data cache read timing, while the dwfast field improves data cache write timing. If *dlram* is set to true, a local (on-chip) data ram will be enabled. The size of the ram will be *dlramsize* Kilo Bytes (KBytes). The 8 MSB bits of the ram start address are set in *dlramaddr*. However, currently, we do not use *dlram* on the Liquid architecture platform.

In all, there are 15 cache parameters (5 in icache and 10 in dcache) that can be customized and the total number of their values is 53. Of these, for application-specific customization, we consider 10 parameters (4 in icache and 6 in dcache), with 34 different values, as shown in Figure 2.4.

Integer Unit

Figure 2.6 shows the parameters of **Integer Unit** (IU). *Fastjump* uses a separate branch address adder. Setting **Integer Condition Code** (ICC) *hold* will improve timing by adding

Parameter	Range of values	Default	Values for Customization
Processor			
Cache			
Instruction cache (icache)			
nsets	1-4	1	All
Set size	1,2,4,8,16,32,64KB 64KB requires 213 BRAM (i.e.) 33% more than available	4	All but 64
Line size	4,8 words	8	4,8
Replacement	Random,LRR,LRU if LRR, nsets=2; LRU only if nsets=2,3,or 4	Random	All
Icache locking	Enable/disable	Disable	No
Data cache (dcache)			
nsets	1-4	1	All
Set size	1,2,4,8,16,32,64KB 64KB requires 213 BRAM (i.e.) 33% more than available	4	All but 64
Line size	4,8 words	8	All
Replacement	Random, LRR, LRU if LRR, nsets=2; LRU only if nsets=2,3,or 4	Random	All
Dcache locking	Enable/disable	Disable	No
Dcache snooping	Enable/disable	Disable	No
Fast read-data generation	Enable/disable	Disable	Yes
Fast write-data generation	Enable/disable	Disable	Yes
Local data RAM (LRAM)	Enable/disable	Disable	No
If LRAM, size	1,2,4,8,16,32,64 KB	2	No
If LRAM, starting address	8bits	16#8F#	N/A

Figure 2.4: Processor cache parameters

a pipeline hold cycle if a branch instruction is preceded by an icc-modifying instruction. Similarly, *fastdecode* will improve timing by adding parallel logic for register file address generation. The pipeline can be configured to have either one or two load delay cycles using the *lddelay* option. One cycle gives higher performance (lower CPI) but may result in slower timing in ASIC implementations.

To use hardware implementation *multiplier/ divider*, '-mv8' switch needs to be passed to the sparc-elf-gcc, the cross compiler used by Liquid architecture platform. The hardware implementations available are enumerated in Figure 2.5, along with their logic-latency tradeoffs. The options are essentially about different data widths (number of bits) of the

Configuration	latency (clocks)	Kgates
iterative	35	1,000
m16x16 + pipeline reg	5	6,500
m16x16	4	6,000
m32x8	4	5,000
m32x16	2	9,000
m32x32	1	15,000

Figure 2.5: LEON multiplier configuration tradeoffs

operands and hence different data width of the product. For instance, m32x32 multiplies two 32 bit numbers and produces a 64 bit result. If multiplier/ divider are set to none, software implementation will be used.

There is one parameter from the synthesis system (Section 2.4.2) that goes hand in hand with MUL/ DIV in IU. It is the *infer_mult* parameter. If it is false, the multipliers are implemented using the module generators in multlib.vhd in LEON distribution. If *infer_mult* is true, the synthesis tool will infer a multiplier. For FPGA implementations, best performance is achieved when infer_mult is true and m16x16 is selected. ASIC implementations (using synopsys DC) should set *infer_mult* to false since the provided multiplier macros in MULTLIB are faster than the Synopsys generated equivalents. The *mulpipe* option can be used to infer pipeline registers in the m16x16 multiplier when infer_mult is false. This will improve the timing of the multiplier but increase the latency from 4 to 5 clocks.

nwindows set the number of register windows; the SPARC standard allows 2 - 32 windows, but to be compatible with the window overflow/underflow handlers in the LECCS compiler, 8 windows need to be used. Besides, windows less than 16, other than 8, do not synthesize with the LEON distribution we use. Therefore, only 17 options are valid in practice for us.

In addition, MAC option enables the SMAC/UMAC instructions but ISA reconfiguration is beyond the scope of microarchitecture customization. The parameter *rflowpow* enables read-enable signals to the register file write ports, thereby saving power when the register file is not accessed. However, this option might introduce a critical path to the read-enable ports on some register files and hence not included for customization. Setting *watchpoint* to a value between 1 - 4 will enable corresponding number of watchpoints. Setting it to 0,

Parameter	Range of values	Default	Values for Customization
Processor			
Integer Unit (IU)			
Fast jump	Enable/disable	Enable	Yes
ICC hold	Enable/disable	Enable	Yes
Fast decode	Enable/disable	Enable	Yes
Load delay	1,2clock cycles	1	Both
MAC	Enable/disable	Disable	No
Multiplier	32x32,32x16,32x8,16x16, 16x16+pipelineReg,iterative, none	16x16	32x32,32x16, 32x8,16x16 16x16+pipe,iter, none
Divider	radix2,none	radix2	radix2,none
Reg. windows	2-32 3,5,6,7,9-15 windows do not build Applications do not run on 2,4 windows	8	8,16-32
FPU	Enable/disable	Disable	No
Co-processor	Enable/disable	Disable	No
Disable RegFile when not accessed	Enable/disable	Disable	No
Hardware watchpoints	0-4	4	No
Implementation ID	0-15	0	N/A
Version ID	0-15	0	N/A

Figure 2.6: Processor Integer Unit parameters

will disable all watch-point logic. We set it to 4 and do not consider other values during customization.

In summary, there are 12 customizable parameters in IU, with 61 total values. Out of these, for application-specific customization, we consider 7 parameters, totaling 35 values, as shown in Figure 2.4.

Floating-point Unit

Figure 2.7 shows the 3 customizable parameters of (optional) FPU, with 8 different values. The *interface* element defines whether to use a serial, parallel (with IU instruction), or none (no FPU) interface.

If FPU is enabled, three implementations can be interfaced - GRFPU core from Gaisler Research, Meiko core from Sun Microsystems, or an incomplete, open-source core LTH

Parameter	Range of values	Default value
Processor		
Floating-point Unit (FPU)		
Interface	serial,parallel,none	None
Core	Gaisler,Meiko,LTH	Meiko
Reg. windows	32 for serial,0 for parallel	0
Version ID	0-7	0

Figure 2.7: Processor FPU parameters (not included in our customization)

core. According to the LEON manual, LTH does not implement all SPARC V8 instructions and is not IEEE-754 compliant. More specifically, it currently implements single- and double-precision addition, subtraction and compare and does not implement multiplication, division and square root. Hence, LTH is not recommended for general purpose programs. Therefore, we do not include FPU is our customization experiments.

Memory Management Unit

Figure 2.8 shows the 6 customizable parameters of the optional **Memory Management Unit** (MMU), along with their 18 values. The MMU implements a SPARC V8 reference MMU to support operating systems such as Linux and Solaris. The MMU can have separate (Instruction + Data) or a common **Translation Look-aside Buffer** (TLB). The TLB is configurable for 2 - 32 fully associative entries.

In the version of Liquid architecture platform that we use for customization, MMU is not enabled. Further, in our experiments, we run the benchmarks directly on LEON (i.e.) without an operating system (although Liquid architecture platform supports Linux) and hence we do not include MMU for our customization.

Co-processor

A generic co-processor interface is provided to allow interfacing of custom (special-purpose) co-processors. The interface allows an execution unit to operate in parallel to increase performance. One coprocessor instruction can be started each cycle as long as there are no

Parameter	Range of values	Default value
Processor		
Memory Management Unit (MMU)		
MMU	Enable/disable	Disable
TLB (instruction,data)	Combined/split	Combined
TLB replacement	LRU/increment	LRU
TLB instructions (or combined) entries	2,4,8,16,32	8
TLB data entries	2,4,8,16,32	8
Diagnostic Enable/disable	Disable	Disable

Figure 2.8: Processor MMU parameters (not included in our customization)

Parameter	Range of values	Default value
Processor		
Co-processor		
Configuration	Name	cp_none

Figure 2.9: Co-processor parameters (not included in our customization)

data dependencies. When finished, the result is written back to the co-processor register file.

Figure 2.9 shows the one customizable parameter for naming the co-processor, when a core is available. To use the co-processor core, application needs to use instructions for the core and since application changes are beyond the scope of our customization, we do not include co-processor in our customization.

Debug Support Unit

The (optional) debug support unit (DSU) allows non-intrusive debugging on target hardware. The DSU allows insertion of breakpoints and watchpoints, and access to all on-chip registers from a remote debugger. A trace buffer is provided to trace the executed instruction flow and/ or AHB bus traffic. The DSU has no impact on performance and has low area complexity. Communication to an outside debugger (e.g. gdb) is done using a dedicated UART (RS232) or through any AHB master device (e.g. PCI).

Parameter	Range of values	Default value
Processor		
Debug Support Unit (DSU)		
DSU	Enable/disable	Disable
Trace buffer	Enable/disable	Disable
Mixed instruction/ AHB tracing	Enable/disable	Disable
Trace buffer lines	64,128,256,512,1024	128

Figure 2.10: Processor DSU parameters (not included in our customization)

Figure 2.10 shows the 4 customizable parameters (and a total of 11 values) for enabling and using DSU. We leave them disabled because Liquid architecture platform did not support DSU at the time of our customization exercise.

2.4.2 Synthesis Options

The synthesis configuration is used to adapt the model to various synthesis tools and target libraries. Figure 2.11 shows the customizable parameters to be 7 in number and their values to be 14 in total.

Depending on synthesis tool and target technology, the technology dependant mega-cells (ram, rom, pads) can either be automatically inferred or directly instantiated. When using tools with inference capability targeting Xilinx Virtex, a choice can be made to either infer the mega-cells automatically or to use direct instantiation. The choice is done by setting the parameters *infer_ram*, *infer_regf* and *infer_rom* accordingly. The *rftype* option has impact on target technologies which are capable of providing more than one type of register file. *Infer_mult* selects how the multiplier is generated, for details see section 14.2 below. On Virtex targets, *clk_mul* and *clk_div* are used to configure the frequency synthesizer (DCM or CLKDLL).

We already discussed *infer_mult* in Section 2.4.1; we will discuss clock generation in Section 2.4.3. We do not consider other parameters for customization.

Parameter	Range of values	Default	Values for Customization
Synthesis options			
Target technology	Xilinx Virtex,Virtex2, Generic, Atmel ATC35,25,18, UMC-FS90,0.18, TSM0.25, Actel Prosaic,Axcel	Virtex	No
Infer Pads	Enable/disable	Disable	No
Infer PCI	Enable/disable	Disable	No
Infer RAM	Enable/disable	Enable	No
Infer Register file	Enable/disable	Enable	No
Infer ROM	Enable/disable	Disable	No
Infer Mult/Div	Enable/disable	Enable	Yes
Improve RegFile write timing	Enable/disable	Enable	Yes
Target clock	Gen,Virtex CLKDLL, Virtex2 DCM, PCI DLL, PCT SYSCLK	Gen	No
Clock multiplier	$1/2$,1,2	1	No
Clock divider	$1/2$,1,2	1	No
PCI DLL	Enable/disable	Disable	No
PCI system clock	Enable/disable	Disable	No

Figure 2.11: Synthesis and clock generation options

2.4.3 Clock Generation

Synthesis system (Figure 2.11) provides options for technology specific clock generation as well as clock multiply and divide factors but none of our benchmarks need customization on this and hence we do include these options in our customization.

2.4.4 Memory Controller

The flexible memory interface provides a direct interface for PROM, memory mapped I/O devices, static RAM (SRAM) and synchronous dynamic RAM (SDRAM). Memory areas can be programmed to be 8-, 16- or (the default) 32-bit data width. The *ramsle5* field enables the fifth (RAMSN[4]) chip select signal in the memory controller. The *sdramen* field enables SDRAM controller, while *sdinvclk* controls the polarity of the SDRAM clock. For the benchmarks that we consider in this study, we use SRAM. We default the other parameters to false to reduce logic and exclude them from customization.

Parameter	Range of values	Default value
Memory controller		
8-bit bus	Enable/disable	Disable
16-bit bus	Enable/disable	Disable
Feedback to data bus drivers	Enable/disable	Disable
5th RAM select	Enable/disable	Disable
SDRAM controller	Enable/disable	Disable
Invert SDRAM clock	Enable/disable	Disable

Figure 2.12: Memory Controller parameters (not included in our customization)

We leave SDRAM enabled but we do not compare SRAM to SDRAM in our customization experiments because the Liquid architecture platform does not support using both SRAM and SDRAM concurrently. Since SDRAM is slower to access, any runtime gain we obtain from cache customization will only be better with using SDRAM. Figure 2.12 shows these 4 customizable parameters; they are all of type enable/disable and therefore the total number of their values is 8.

2.4.5 AMBA Configuration

The processor has a full implementation of AMBA Hi-speed (AHB) and AMBA Peripheral (APB) on-chip buses. AMBA buses are the main way of adding new functional units in LEON. The LEON model provides a flexible configuration method to add and map new AHB/APB compliant modules.

The default AHB master is the memory controller. The number of AHB slaves and their address range is defined through the AHB slave table in device.vhd. The AHB slaves should be connected to the *ahbsi/ahbso* buses. The *index* field in the table indicates which bus index the slave should connect to. If *split* field is set to true, the AHB arbiter will include split support and each slave must then driver the SPLIT signal.

The number of APB slaves and their address range is defined in APB bridge (apbmst.vhd in LEON distribution). APB slaves can be added by editing the corresponding case statement in apbmst.vhd and adding the modules in MCORE. The APB slaves are connected to the *apbi/apbo* buses.

Parameter	Range of values	Default value
AMBA bus		
Default AMBA master	integer	0
AHB split-transaction support	Enable/disable	Disable

Figure 2.13: AMBA parameters (not included in our customization)

Figure 2.13 shows 2 customizable parameters for AHB, with a total of 3 values. Our benchmarks do not use any peripheral device off APB and neither do they require changes to the parameters mentioned above and therefore we do not include these parameters in our customization experiments.

2.4.6 Peripherals

If not enabled, the corresponding peripheral function will be suppressed, resulting in a smaller design.

The *irq2en* parameter enables secondary interrupt controller. The *ahbram* parameter enables on-chip AHB RAM. *ahbrambits* denote the number of address bits used by the RAM. Since a 32-bit RAM is used, 8 address bits will results in a 1-KByte RAM block.

A 24-bit *watchdog* is provided on-chip. The watchdog is clocked by the timer *prescaler*. When the watchdog reaches zero, an output signal (WDOG) is asserted. This signal can be used to generate system reset.

Figure 2.15 shows 9 customizable parameters and a total of 23 values. None of the benchmarks used in our experiments use any of these parameters and therefore we leave these parameters turned off (to save logic) and not include them for customization.

2.4.7 PCI

A PCI interface supporting target-only or both target and master operation can be enabled. The target-only interface has low complexity and can be used for debugging over the PCI

Parameter	Range of values	Default value
Peripheral		
Config register	Enable/disable	Enable
AHB status register	Enable/disable	Disable
Wprot	Enable/disable	Disable
Watchdog	Enable/disable	Disable
irq2en	Enable/disable	Disable
AHB RAM	Enable/disable	Disable
AHB RAM size	1,2,4,8,16,32,64 KB	4
AHB RAM bits	11	11
Ethernet	Enable/disable	Disable
PCI	Enable/disable	Disable

Figure 2.14: Peripherals parameters (not included in our customization)

bus. The full master/target interface is based on the PCI bridge from OpenCores, with an additional AHB interface.

The *pcicore* field indicates which PCI core to use. Currently, only the OpenCores PCI core is provided with model. If this field is set to none, PCI interface will be disabled. The ahbmasters and ahbslaves fields indicate how many AHB master and slave interfaces the selected core types have. For *opencores*, both these fields should be set to 1. To enable the PCI arbiter, *arbiter* field should be set to true. This should only be done in case the processor acts as a system controller. The PCI vendorid, deviceid and subsystemid can be configured through the corresponding fields.

Figure 2.15 shows the customizable parameters to be 10 in number and their values to be 15 in total. However, since our benchmarks do not use any of these parameters, we leave them all at their default values and not include them in our customization experiments.

2.4.8 Boot Options

Apart from the standard boot procedure of booting from address 0 in the external memory, LEON can be configured to boot from an internal prom or from the debug support unit.

Parameter	Range of values	Default value
PCI		
Core	None,	None
AHB masters	a number	0
AHB slaves	a number	0
Arbiter	Enable/disable	Disable
Fixed priority	Enable/disable	Disable
Priority level	4	4
PCI masters	a number	4
Vendor ID	a number	16#0000#
Device ID	a number	16#0000#
Subsystem ID	a number	16#0000#
Revision ID	a number	16#00#
Class code	a number	16#000000#
PME pads	Enable/disable	Disable
P66 pad	Enable/disable	Disable
PCI Read stall	Enable/disable	Disable

Figure 2.15: PCI parameters (not included in our customization)

Figure 2.16 shows the 7 customizable parameters. There was no reason to change the default values of these parameters or customize them per benchmark that we ran and therefore we do not include boot options for customization.

2.4.9 VHDL Debugging

Two 8-bit UARTs are provided on-chip for communication over serial ports. The baud-rate is individually programmable and data is sent in 8-bits frames with one stop bit. Optionally, one parity bit can be generated and checked.

We leave debug and UART parameters enabled and disable all other debug parameters and exclude the entire system from customization, since we do not expect reconfiguration from this subsystem to improve application performance. Figure 2.17 shows the 11 customizable parameters and a total of 20 values.

Parameter	Range of values	Default value
Boot options		
Boot from	Memory, internal PROM,	PROM
RAM read waitstates	a number	3
RAM write waitstates	a number	0
System clock (MHz)	a number	25
Baud rate	a number	9600
Ext baud	Enable/disable	Disable
PROM Addr bits	a number	8

Figure 2.16: Boot options (not included in our customization)

Parameter	Range of values	Default value
Debug		
Debug	Enable/disable	Enable
UART	Enable/disable	Enable
IU Registers	Enable/disable	Disable
FPU Registers	Enable/disable	Disable
No halt	Enable/disable	Disable
PC low		2
DSU	Enable/disable	Enable
DSU trace	Enable/disable	Enable
DSU mixed	Enable/disable	Disable
DSU DPRAM	Enable/disable	Disable
Trace lines	128	128

Figure 2.17: Debug options (not included in our customization)

2.5 Parameters for Application-Specific Customization

The number of customizable LEON parameters that we discussed in the above section (Section 2.4), is recapped in Figure 2.18. In all, the total number of customizable parameters is 95 and the total number of their values is 246. Of these, we consider 16 parameters, with a total of 62 values, for our application-specific customization experiments. This is shown in Figure 2.18.

			Included in our customization	Included in our customization
LEON system	#params	#param values	#params	#param values
Processor				
Cache	15	53	10	34
Integer Unit	12	61	7	35
Floating-point Unit	3	8	0	0
Memory Management Unit	6	18	0	0
Co-processor	1	2	0	0
Debug Support Unit	4	11	0	0
Synthesis	7	14	1	2
Clock options	4	10	0	0
Memory controller	4	8	0	0
AMBA	2	3	0	0
Peripherals	9	23	0	0
PCI	10	15	0	0
Boot	7	0	0	0
VHDL debugging	11	20	0	0

Figure 2.18: LEON parameters for application-specific customization

2.6 Constrained Binary Integer Nonlinear Programming

Application-specific customization of soft core processor microarchitecture is achieved by formulating the problem as a constrained BINP.

Linear Programming (LP) in *general form* is the problem of minimizing a linear function subject to a finite number of equality and inequality constraints [27]. The following is a general LP problem (or simply, linear program) with n variables, k equality constraints, and l inequality constraints. Minimize $Z = \mathbf{c}^T\mathbf{x}$
subject to
$\mathbf{Ax} = \mathbf{b}$
$\mathbf{A'x} \geq \mathbf{b'}$
$x_1 \geq 0, x_2 \geq 0, ..., x_r \geq 0$; $r \leq n$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables to be solved for, $Z|\mathbb{R}^n- > \mathbb{R}$ is the objective function, $\mathbf{c} \in \mathbb{R}^n$ is the constants vector of cost coefficients, $\mathbf{A} \in \mathbb{R}^{k\times n}$ and $\mathbf{A'} \in \mathbb{R}^{l\times n}$ are matrices of coefficients of functional constraints, $\mathbf{b} \in \mathbb{R}^k$ and $\mathbf{b'} \in \mathbb{R}^l$ are constants vectors of right-hand sides of functional constraints. x_1 through x_r are restricted to be non-negative; x_{r+1} through x_n are not sign constrained. k or l can be 0. *Other forms* of LP include maximization and inequalities that are $\leq$.

If the decision variables are restricted to be integers, the model becomes Integer (Linear) Program (ILP or simply IP). In addition, if the variables are further restricted to be binary-valued (zero or one), the model would be Binary Integer (Linear) Program (BILP or simply BIP).

If the objective function, a constraint or both are nonlinear, then the model is a **Nonlinear Programming** (NLP) and if the variables are binary-valued, then it is Binary Integer Nonlinear Program (BINLP).

LP problems are *continuous* over real numbers, or over non-negative reals, if all variables are restricted to be non-negative [24]. LP problems are guaranteed to have a **Corner-Point Feasible** (CPF) solution (and hence a corresponding basic feasible solution) that is optimal for the overall problem. It is because of this guarantee that LP problems are solved extremely efficiently.

In contrast, IP problems are *discrete* because noninteger feasible solutions are no longer valid and hence removed from the search space. Because of this, IP problems are no longer guaranteed to have a CPF solution that is optimal for the overall problem. One approach would be to solve IP problems through exhaustive enumeration—check each solution for feasibility and if feasible, calculate the value of objective function. However, the number of feasible solutions, though finite, could be very large, making it infeasible to do exhaustive enumeration. For instance, in the simple case of a BIP, if there are n variables, there are 2^n solutions to consider and if n is increased by 1, the number of solutions to consider doubles. That is, the difficulty of the problem grows *exponentially* with the number of variables. To avoid exhaustive enumeration, a popular approach with IP algorithms is to use *branch-and-bound* technique, which enumerates only a small fraction of the feasible solutions. The basic concept with this technique is to *divide and conquer*. The dividing (branching) is done by partitioning the set of feasible solutions into smaller and smaller subsets. The conquering (fathoming) is done partially by bounding how good the best solution in the subset can be and then discarding the subset if its bound indicates that it cannot possibly contain an optimal solution for the original problem. While fathoming, to quickly evaluate the best solution, a technique called *LP relaxation* is used often. LP relaxation involves simply dropping the integrality constraints and solving the problem as an LP problem. (If, on the other hand, LP relaxation is used to directly solve the overall integer programming problem, the optimal solution, if not an integer, will have to be rounded. However, rounding the optimal solution may make it non-optimal or worse, infeasible.)

NLP algorithms are generally unable to distinguish between a local minimum and a global minimum *except* under the following conditions. If the objective function is convex and the feasible region (FR) formed by the constraints is a convex set, then, a local minimum is guaranteed to be a global minimum. A linear objective function is convex (as well as concave) and the FR of an LP problem is a convex set. The FR of a NLP problem is a convex set if all the constraint equations are convex functions. A function of n variables $f(x_1, x_2, ..., x_n)$ is a convex function if, for each pair of points on the graph of $f(x_1, x_2, ..., x_n)$, the line segment joining these two points lies entirely above or on the graph of $f(x_1, x_2, ..., x_n)$.

For the problem of application-specific microarchitecture customization, we use a commercial solver called Tomlab **Mixed Integer Nonlinear Programming Solver** (MINLP) [53].

Tomlab is a plugin for Matlab [35] and it solves our formulation in seconds on modern computers.

2.7 Alternative Search Techniques

Integer Programming problems grow exponentially with the number of variables and hence is NP-hard [24]. Therefore, heuristic algorithms have been developed that are extremely efficient for large problems but they are not guaranteed to find optimal solution.

Three prominent heuristics are tabu search, simulated annealing and genetic algorithms [24]. They all use innovative concepts to move towards an optimal solution. Tabu search explores promising areas holding good solutions by rapidly eliminating unpromising areas classified as tabu. Simulated annealing searches by using the analog of a physical annealing process. The basic concept with genetic algorithms is survival of the fittest through natural evolution. These algorithms can also be applied to integer nonlinear programs that have local optima far removed from global optima.

With heuristics, we would start with a processor configuration, which is a tuple of parameter values, perturb the configuration along a parameter and evaluate the cost. If the cost decreases the objective function (since we are 'minimizing' application runtime and hardware resource usage), then we accept this change [13] and repeat the process in this direction.

We do not use these heuristics for two reasons. First, processor configurations are combinations of a large number of parameters. As shown in Figure 2.18, LEON processor has 95 parameters with 246 values. Given such a large of dimensions, selecting a parameter to perturb becomes a challenge by itself. Secondly, given the large number of possible configurations, measuring costs of all or even many processor configurations is infeasible, because it takes 30 minutes to build a single processor configuration, even on modern computers. This is discussed in detail in Section 3.1.2.

2.8 Benchmarks

The following applications are used in our experiments. They are executed directly on LEON (i.e.) without an operating system. Hence, they have been modified to not use system calls. Examples of modifications include not using *stdio* but instead using memory for input and output and generating random numbers in the application itself. Source code for all the applications are listed in Appendix C.

2.8.1 Benchmark I - BLASTN

Basic Local Alignment Search Tool (BLAST) [1] programs are the most widely employed set of software tools for comparing genetic material. BLASTN ("N" for *nucleotide*) is a variant of BLAST used to compare DNA sequences (lower-level than proteins).

A DNA sequence is a string of characters (*bases*), with each base drawn from the 4-symbol alphabet $\{$A,C,T,G$\}$.

BLASTN is classically implemented as a three-stage pipeline, where each stage performs progressively intensive work over a decreasing volume of data.

We analyze the performance of an open-address, double-hashing scheme to determine word matches as in stage 1 of BLASTN [37]. We use a synthetically generated database and query containing only bases from $\{$A, C, T, G$\}$, for our experiments. The bases were generated within the program, using random-number generators. For the purposes of these experiments, we used a word size (w) of 11, which is also the default value of w used by the flavor of BLASTN that is distributed by the National Center for Biological Information (NCBI).

BLASTN is computation and memory-access intensive. It has approximately 163 lines of code and its runtime on the default LEON configuration is 10.6 seconds.

2.8.2 Benchmark II - Commbench DRR

DRR is a Deficit Round Robin fair scheduling algorithm used for bandwidth scheduling on network links, as implemented in switches. The program kernel focusses on queue maintenance and packet scheduling for fair resource utilization and is computationally intensive [54]. DRR has approximately 117 lines of code and its runtime on the default LEON configuration is 5 minutes.

2.8.3 Benchmark III - Commbench FRAG

FRAG is an IP packet fragmentation application. IP packets are split into multiple fragments for which some header fields have to be adjusted and a header checksum computed. The checksum computation that dominates this application is performed as part of all IP packet application programs besides just forwarding [54]. FRAG has approximately 150 lines of code and its runtime on the default LEON configuration is 2.5 minutes.

2.8.4 Benchmark IV - BYTE Arith

Arith does simple arithmetics of addition, multiplication and division in a loop. This benchmarks tests the processor speed for arithmetic and is not memory intensive. approximately 77 lines of code and its runtime on the default LEON configuration is 32 seconds.

Chapter 3

Approach

The goal of this thesis is to enable application developers of constrained embedded systems to improve performance of their application by customizing the soft core processor's microarchitecture. The processor is customized to match the given application's requirements and constraints closely. To make it easier for the developers, the customization process is automated. The developers are needed only to specify the customizable parameters of the processor and our solution will explore the parameters, without any further involvement of the developers.

The approach we take is to explore *all* microarchitecture parameters that have a bearing on application runtime or hardware resources that are being optimized or constrained. Secondly, for more accurate customization results, we would like to use *actual* measurements (costs) rather than estimates. Section 1.4 points out the shortcomings of using estimates. Accordingly, for application runtime cost, applications are executed directly on the processor. For hardware resource utilization cost, processor configurations are actually built from the source VHDL. For all other dimensions that are being optimized or constrained such as energy consumption, and power dissipation, we would use similar actual cost measurements, although we leave this for future. Finally, despite customizing *all* parameters and measuring their *actual* costs, we would like the optimization technique to be *feasible and scalable*.

Arising from these goals are two challenges. First, considering all microarchitecture parameters means considering hundreds of parameter values, 246 for LEON, as shown in Figure 2.18. Such a large number of parameter values makes the search space huge, 2^{246} configurations for LEON. The second challenge is the long time it takes to measure the

costs of application runtime, hardware resource utilization, energy consumption, and power dissipation.

3.1 Cost functions

3.1.1 Application Runtime Cost

Application runtime is measured by executing the application directly on the soft core processor (LEON) and counting the number of clock cycles the execution takes. We use the non-intrusive and cycle-accurate hardware-based profiler available through Liquid architecture platform described in Section 2.1.

The runtimes for the different benchmarks used in our experiments range from 16 seconds to 9 minutes. However, there could be applications with much longer execution times. We leave it for future work to address such very long execution times; future work is discussed in Section 7.4.

3.1.2 FPGA Resource Cost

Instantiating a soft core processor configuration on an FPGA utilizes hardware resources. We focus on the utilization of **Lookup Tables** (LUTs) and BRAM, described in Section 2.3. Other resources can be included in a similar way.

Hardware resource utilizations are measured by actually building processor configurations from the source VHDL. Processor configurations are independent of applications executed on them, which means, they are generated only once. Even so, each build is very time-consuming, on the order of 30 minutes, even on modern computers.

The total LUTs and BRAM available on the Xilinx Virtex XCV2000E FPGA used in the Liquid architecture platform are 38,400 and 160 respectively and the default LEON configuration utilizes 14,992 (39%) and 82 (51%). Given the difference in their magnitudes, they are normalized as percentages and added together for a unified chip resource cost metric.

3.1.3 Total Cost

To be compatible with chip resource cost, application runtime cost is also normalized as a percentage and they are added together.

3.2 Our Approach

The number of possible soft core processor configurations are exponential with the number of microarchitecture parameters as discussed in Section 1.2, resulting in a huge search space. In addition, the time to measure the "data points" are relatively excessive. The data points comprise hardware resource utilization cost obtained from building the processor configuration and application runtime cost obtained from executing the application on that configuration. Building a processor configuration is relatively expensive – on the order of half an hour for each configuration, even on modern computers (processors). The application execution time can be of any length as discussed in Section 3.1.1. These two characteristics make the problem of automatic microarchitecture customization more challenging than a traditional optimization problem. They make it infeasible to do exhaustive enumeration of all configurations (i.e.) it is infeasible to build an exact model to search for the best solution.

The next best approach is to build an approximate model and solve for an exact solution. We build the model by assuming *parameter independence* and restricting each parameter to its own dimension. Though our results are no longer guaranteed to be optimal in all cases, Section 6 demonstrates that they are near-optimal in practice. With the assumption of parameter independence, the number of configurations is *linear* in the number of parameter values, 62 for the parameters in Figure 2.18. Even if the remaining parameters benefit other applications, the total number of parameter values would still be only a few hundred in number, which can be handled in a feasible and scalable manner by our approach.

We solve for optimal solution by formulating the model as a constrained Binary Integer Nonlinear Problem. Although the search space is built by considering parameters in their own dimensions, the optimization algorithm evaluates points in between. These points represent configurations that have more than one parameter changed simultaneously. The

solver assigns costs for these points through an *approximation* of actual costs provided by us in the model.

As discussed in Section 2.6, Integer Linear Programming is exponential with the number of variables. Therefore, it is not guaranteed to give an optimal solution in the presence of a large number of integer variables and in the absence of special structures that some algorithms exploit to solve for optimal solution. With nonlinear objective function or constraints, if the objective function is not convex or if feasible region formed by the constraints is not *convex*, then, the optimization algorithm is no longer guaranteed to find the optimal solution. Because of these two characteristics, the solution from our customization is not guaranteed to be optimal. However, the resulting configuration is guaranteed to be valid and near-optimal in practice as demonstrated in Section evaluation.

The approach to building the model is summarized as follows. We begin with the default LEON configuration that comes out-of-the-box. We call this the *base* configuration. We then perturb one parameter at a time and build the processor configuration, measuring its chip cost. Thirdly, we execute the application on each configuration, measuring the runtime. Finally, we formulate these costs into a BINP problem and solve for optimal solution using the commercial solver of Tomlab MINLP. The solution obtained is the recommended microarchitecture configuration for the given application. The characteristics of the solution are discussed in Section 2.6. Our approach itself is illustrated in Figure 3.1.

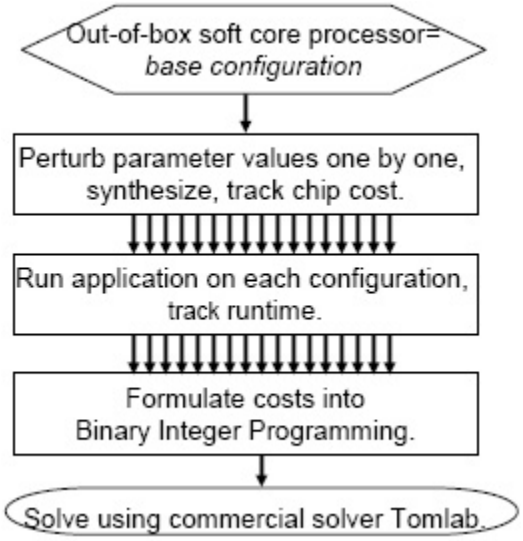

Figure 3.1: Our heuristic for automatic application-specific microarchitecture customization

Chapter 4

Problem Formulation

We formulate the problem of automatic customization of soft core processor microarchitecture as a Binary Integer Program. The objective of the customization is to meet the application's runtime requirements and hardware (FPGA) resource restrictions. The constraint is to select a *valid* microarchitecture configuration that *fits* in the available chip resources.

The FPGA resources considered for the customization are lookup tables (LUTs) and Block RAM (BRAM), described in Section 3.1.2. The total number of LUTs and BRAM available on the Xilinx Virtex XCV2000E FPGA [57] used in our Liquid architecture platform are 38,400 and 160 respectively. For the soft core processor that we use namely LEON, the relevant microarchitecture subsystems for customization are instruction cache (icache), data cache (dcache) and integer unit (IU), described in Section 2.4. We start the customization process with default out-of-the-box LEON distribution. We call this the *base* configuration. The base configuration utilizes 39% LUTs (14,992 out of 38,400) and 51% BRAM (82 out of 160).

The %LUTs and %BRAM remaining unutilized after the base configuration are 61% and 49% and are denoted by L and B respectively, as shown in Figure 4.1. From the base configuration, for the parameters included for customization, one parameter value is changed at a time and the new processor configuration is built. With this approach, the number of new LEON configurations reduce to 52. Each new processor configuration is denoted by x_i. For each x_i, the difference (in percentage) in LUTs and BRAM over the base configuration is denoted by λ_i and β_i respectively; the difference (in percentage) in application execution time over the time on the base configuration is denoted by ρ_i. These symbols are listed in Figure 4.1.

Variable	Symbol
% ΔLUTs remaining after base configuration	L
% ΔBRAM remaining after base configuration	B
Each new processor configuration	x_i
% ΔLUTs of x_i	λ_i
% ΔBRAM of x_i	β_i
% ΔRuntime of x_i	ρ_i

Figure 4.1: LEON reconfigurable parameters

4.1 Parameter Validity Constraints

x_i represents a new processor configuration resulting from a change in *a single parameter value* from the *base* configuration. x_i is binary (i.e.) it represents two integer values, which could be simple *on* or *off*. That implies that for parameters with more than two values, more than one x_i will be used. Therefore, for such parameters, we need to ascertain that at most one of them is selected (turned *on*) at a time. None of them may be selected because the base configuration already includes a value for this parameter. All such constraints are developed below.

4.1.1 Instruction Cache Parameter Validity Constraints

The four parameters in LEON's icache that have an impact on application runtime or FPGA resources and that can be customized without changing the application are number of cache sets, size of each set, cache line size, and cache replacement policy.

ICache Number of sets

LEON icache is *direct-mapped* by default in the base LEON configuration. However, it can be changed to be 2, 3, or 4-way associative. We let variables x_1 through x_3 represent these changes (i.e.) x_1 represents whether 2-way associativity is used (i.e) turned *on* or *off*. Similarly, x_2 and x_3 represent whether 3-way and 4-way associativity are turned on or off

Parameter	Value	Variable	Runtime	ChipResource
nsets	1	base	0	0
	2	x_1	ρ_1	$\lambda_1 + \beta_1$
	3	x_2	ρ_2	$\lambda_2 + \beta_2$
	4	x_3	ρ_3	$\lambda_3 + \beta_3$
	$x_1 + x_2 + x_3 \leq 1$			

Figure 4.2: ICache formulation - variables and constraint

Parameter	Value	Variable	Runtime	ChipResource
nsets	1	base	0	0
	2	x_1	ρ_1	$\lambda_1 + \beta_1$
	3	x_2	ρ_2	$\lambda_2 + \beta_2$
	4	x_3	ρ_3	$\lambda_3 + \beta_3$
	$x_1 + x_2 + x_3 \leq 1$			
setsize (KB)	1	x_4	ρ_4	$\lambda_4 + \beta_4$
	2	x_5	ρ_5	$\lambda_5 + \beta_5$
	4	base	0	0
	8	x_6	ρ_6	$\lambda_6 + \beta_6$
	16	x_7	ρ_7	$\lambda_7 + \beta_7$
	32	x_8	ρ_8	$\lambda_8 + \beta_8$
	$x_4 + ... + x_8 \leq 1$			

Figure 4.3: ICache formulation - variables and constraints

respectively. The variables and the validity constraint for the parameter of number of sets is presented in Figure 4.2.

ICache Set size

ICache *set* size is 4 Kilo Bytes (KB) by default in the base LEON configuration and it can be changed to be 1, 2, 8, 16, 32, or 64 KB. Of these, 64 KB requires a total number of 213 BRAM but this exceeds what is available on our FPGA—160. The variables for these values and the constraint to select at most one of them is shown in Figure 4.3.

Parameter	Value	Variable	Runtime	ChipResource
nsets	1	base	0	0
	2	x_1	ρ_1	$\lambda_1 + \beta_1$
	3	x_2	ρ_2	$\lambda_2 + \beta_2$
	4	x_3	ρ_3	$\lambda_3 + \beta_3$
	$x_1 + x_2 + x_3 \leq 1$			
set size (KB)	1	x_4	ρ_4	$\lambda_4 + \beta_4$
	2	x_5	ρ_5	$\lambda_5 + \beta_5$
	4	base	0	0
	8	x_6	ρ_6	$\lambda_6 + \beta_6$
	16	x_7	ρ_7	$\lambda_7 + \beta_7$
	32	x_8	ρ_8	$\lambda_8 + \beta_8$
	$x_4 + ... + x_8 \leq 1$			
line size (words)	4	x_9	ρ_9	$\lambda_9 + \beta_9$
	8	base	0	0
	No constraint needed to ensure parameter validity			

Figure 4.4: ICache formulation - variables and constraints

ICache Line size

ICache line size is 8 words (8 $\times$32 bits) by default in the base LEON configuration and it can be changed to be 4 words. x_9 represents the value of 4 words. Since x_9 is binary, having it *on* indicates cache line size of 4 words and having it *off* indicates the default value of 8 words. Therefore, unlike the two parameters discussed so far, for this parameter, a constraint is *not* needed here to ensure validity. This is true for all cases where there is only one value besides the default value in the base configuration. Variables for the parameter values are shown in Figure 4.4.

ICache Replacement policy

With multi-way set associativity, the default cache replacement policy in the base LEON configuration is *random*. Other replacement policies supported are LRR (Least Recently Replaced) with 2-way associativity and LRU (Least Recently Used) with all multi-way

Parameter	Value	Variable	Runtime	ChipResource
nsets	1	base	0	0
	2	x_1	ρ_1	$\lambda_1 + \beta_1$
	3	x_2	ρ_2	$\lambda_2 + \beta_2$
	4	x_3	ρ_3	$\lambda_3 + \beta_3$
	$x_1 + x_2 + x_3 \leq 1$			
set size (KB)	1	x_4	ρ_4	$\lambda_4 + \beta_4$
	2	x_5	ρ_5	$\lambda_5 + \beta_5$
	4	base	0	0
	8	x_6	ρ_6	$\lambda_6 + \beta_6$
	16	x_7	ρ_7	$\lambda_7 + \beta_7$
	32	x_8	ρ_8	$\lambda_8 + \beta_8$
	$x_4 + ... + x_8 \leq 1$			
line size (words)	4	x_9	ρ_9	$\lambda_9 + \beta_9$
	8	base	0	0
	No constraint needed to ensure parameter validity			
replacement policy	random	base	0	0
	LRR	x_{10}	ρ_{10}	$\lambda_{10} + \beta_{10}$
	LRR	x_{11}	ρ_{11}	$\lambda_{11} + \beta_{11}$
	$x_{10} + x_{11} \leq 1$			
	$x_{10} - x_1 \leq 0$			
	$(x_1 + x_2 + x_3) - x_{11} \geq 0$			

Figure 4.5: ICache formulation - variables and constraints

associativity. Because LRR and LRU are contingent on the number of cache sets, we need an additional constraint for each. They are shown in Figure 4.5.

4.1.2 Data Cache Parameter Validity Constraints

LEON's dcache has all the parameters of icache and two additional parameters of fast data read and write. Figure 4.6 presents all the parameters and their constraints.

4.1.3 Integer Unit Parameter Validity Constraints

LEON's Integer Unit (IU) consists of the parameters of fast jump generator, ICC (Integer Condition Code) hold, fast decode generator, number of pipeline load delay cycles, different hardware multipliers and divider, and number of register windows. Relevant to the hardware multiplier options, is a *synthesis* parameter called infer multiplier. The variables for these parameter values and the constraint for their validity are similar to the cache parameters and are presented in Figure 4.7.

4.2 FPGA Resource Constraints

Microarchitecture customization is constrained by the available FPGA (chip) resources. Of these, we focus on the resources of LUTs and BRAM. Figure 4.1 shows the %LUTs and %BRAM available after the base configuration as L and B. For each x_i the figure also shows the %LUTs and %BRAM over the base configuration as λ_i and β_i. We need to ensure that the LUTs and BRAM used by all the configurations selected during customization fit in the available LUTs (L) and BRAM (B).

$$\sum_{i=1}^{n=52} \lambda_i x_i \leq L \text{ and}$$
$$\sum_{i=1}^{n=52} \beta_i x_i \leq B$$

Cache size is expressed in terms of two parameters in LEON *viz.* number of cache sets and size of each set. We saw that variables x_1 through x_3 represent icache sets and x_4 through x_8 represent icache set size. We also saw that variables x_{12} through x_{14} represent dcache sets and x_{15} through x_{19} represent dcache set size. The constraints for icache size and dcache size would then be:

$$(1 + x_1 + 2x_2 + 3x_3) \times (\sum_{i=4}^{8} \beta_i x_i) \text{ and}$$
$$(1 + x_{12} + 2x_{13} + 3x_{14}) \times (\sum_{i=15}^{19} \beta_i x_i)$$

The FPGA resource constraints then become:

$$(1 + x_1 + 2x_2 + 3x_3) \times (\sum_{i=4}^{8} \lambda_i x_i) +$$
$$(1 + x_{12} + 2x_{13} + 3x_{14}) \times (\sum_{i=15}^{19} \lambda_i x_i) +$$
$$\sum_{i=1}^{3} \lambda_i x_i + \sum_{i=9}^{11} \lambda_i x_i + \sum_{i=12}^{14} \lambda_i x_i + \sum_{i=20}^{52} \lambda_i x_i \leq L$$

$$(1 + x_1 + 2x_2 + 3x_3) \times (\sum_{i=4}^{8} \beta_i x_i) +$$
$$(1 + x_1 2 + 2x_1 3 + 3x_1 4) \times (\sum_{i=15}^{19} \beta_i x_i) +$$
$$\sum_{i=1}^{3} \beta_i x_i + \sum_{i=9}^{11} \beta_i x_i + \sum_{i=12}^{14} \beta_i x_i + \sum_{i=20}^{52} \beta_i x_i \leq B$$

In the nonlinear constraints presented above, *convexity* of the equations happens to be contingent on the values of x_i. Therefore, the optimization algorithm is no longer guaranteed to find global optimum in *all* cases. Hence, to optimize the problem formulation, we leave the constraint on LUTs as a linear function, since variation in LUTs utilization is very minimal. We analyze the effect of this in Section 6.

4.3 Objective Function

4.3.1 Application Runtime Optimization

The primary objective of our microarchitecture customization is to minimize application runtime. This is expressed as:

Minimize $\sum_{i=1}^{n=52} [\rho_i x_i]$

where n is the number of processor configurations generated, ρ_i is the application runtime on x_i over that on the base configuration.

4.3.2 FPGA Resources Optimization

In addition to minimizing application runtime, optionally, we can also minimize FPGA resource usage. This is expressed as:

Minimize $\sum_{i=1}^{n=52}[\rho_i x_i + (\lambda_i + \beta_i)x_i]$

where λ_i, β_i are the %LUTs and %BRAM utilization on x_i over that on the base configuration.

The above equation optimizes application runtime and FPGA resources equally. To change this, we can use *weights* to combine them. This gives us the flexibility to optimize one over the other.

Minimize $\sum_{i=1}^{n=52}[w_1(\rho_i x_i) + w_2((\lambda_i + \beta_i)x_i)]$

where w_1 and w_2 are independent.

4.3.3 Power Dissipation Optimization

Optimizing for a third dimension such as power dissipation, would be a simple extension as shown below:

Minimize $\sum_{i=1}^{n=52}[w_1(\rho_i x_i) + w_2((\lambda_i + \beta_i)x_i) + w_3(p_i x_i)]$

where p_i is the %power dissipation on x_i over that on the base configuration.

4.4 Overall Problem Formulation

The overall problem formulation for customizing LEON's microarchitecture is recapped below.

Minimize $\sum_{i=1}^{n=52}[w_1(\rho_i x_i) + w_2((\lambda_i + \beta_i)x_i)]$

subject to

$$x_1 + x_2 + x_3 \leq 1$$

$$x_4 + ... + x_8 \leq 1$$

$$x_{10} + x_{11} \leq 1$$

$$x_{10} - x_1 \leq 0$$

$$x_1 + x_2 + x_3) - x_{11} \geq 0$$

$$x_{12} + x_{13} + x_{14} \leq 1$$

$$x_{15} + ... + x_{19} \leq 1$$

$$x_{21} + x_{22} \leq 1$$

$$x_{21} - x_{12} \leq 0$$

$$x_{12} + x_{13} + x_{14}) - x_{22} \geq 0$$

$$x_{30} + ... + x_{46} \leq 1$$

$$x_{47} + ... + x_{51} \leq 1$$

$$\sum_{i=1}^{n=52} \lambda_i x_i \leq L$$

$$(1 + x_1 + 2x_2 + 3x_3) \times (\sum_{i=4}^{8} \beta_i x_i) + (1 + x_{12} + 2x_{13} + 3x_{14}) \times (\sum_{i=15}^{19} \beta_i x_i) +$$

$$\sum_{i=1}^{3} \beta_i x_i + \sum_{i=9}^{11} \beta_i x_i + \sum_{i=12}^{14} \beta_i x_i + \sum_{i=20}^{52} \beta_i x_i \leq B$$

Parameter	Value	Variable	Runtime	ChipResource
nsets	1	base	0	0
	2	x_{12}	ρ_{12}	$\lambda_{12} + \beta_{12}$
	3	x_{13}	ρ_{13}	$\lambda_{13} + \beta_{13}$
	4	x_{14}	ρ_{14}	$\lambda_{14} + \beta_{14}$
	$x_{12} + x_{13} + x_{14} \leq 1$			
set size (KB)	1	x_{15}	ρ_{15}	$\lambda_{15} + \beta_{15}$
	2	x_{16}	ρ_{16}	$\lambda_{16} + \beta_{16}$
	4	base	0	0
	8	x_{17}	ρ_{17}	$\lambda_{17} + \beta_{17}$
	16	x_{18}	ρ_{18}	$\lambda_{18} + \beta_{18}$
	32	x_{19}	ρ_{19}	$\lambda_{19} + \beta_{19}$
	$x_{15} + ... + x_{19} \leq 1$			
line size (words)	4	x_{20}	ρ_{20}	$\lambda_{20} + \beta_{20}$
	8	base	0	0
	No constraint needed to ensure parameter validity			
replacement policy	random	base	0	0
	LRR	x_{21}	ρ_{21}	$\lambda_{21} + \beta_{21}$
	LRR	x_{22}	ρ_{22}	$\lambda_{22} + \beta_{22}$
	$x_{21} + x_{22} \leq 1$			
	$x_{21} - x_{12} \leq 0$			
	$(x_{12} + x_{13} + x_{14}) - x_{22} \geq 0$			
fast read	1	x_{27}	ρ_{27}	$\lambda_{27} + \beta_{27}$
	0	base	0	0
	No constraint needed to ensure parameter validity			
fast write	1	x_{52}	ρ_{52}	$\lambda_{52} + \beta_{52}$
	0	base	0	0
	No constraint needed to ensure parameter validity			

Figure 4.6: DCache formulation - variables and constraints

Parameter	Value	Variable	Runtime	ChipResource
fast jump	1	base	0	0
	0	x_{23}	ρ_{23}	$\lambda_{23} + \beta_{23}$
	No constraint needed to ensure parameter validity			
ICC hold	1	base	0	0
	0	x_{24}	ρ_{24}	$\lambda_{24} + \beta_{24}$
	No constraint needed to ensure parameter validity			
fast decode	1	base	0	0
	0	x_{25}	ρ_{25}	$\lambda_{25} + \beta_{25}$
	No constraint needed to ensure parameter validity			
load delay	1	base	0	0
	2	x_{26}	ρ_{26}	$\lambda_{26} + \beta_{26}$
	No constraint needed to ensure parameter validity			
hardware divider	none	base	0	0
	radix2	x_{28}	ρ_{28}	$\lambda_{28} + \beta_{28}$
	No constraint needed to ensure parameter validity			
no infer multiplier (for synthesis)	1	base	0	0
	0	x_{29}	ρ_{29}	$\lambda_{29} + \beta_{29}$
	No constraint needed to ensure parameter validity			
multiplier 16x16	0	base	0	0
	16x16+pipelineReg	x_{47}	ρ_{47}	$\lambda_{47} + \beta_{47}$
	iterative	x_{48}	ρ_{48}	$\lambda_{48} + \beta_{48}$
	32x8	x_{49}	ρ_{49}	$\lambda_{49} + \beta_{49}$
	32x16	x_{50}	ρ_{50}	$\lambda_{50} + \beta_{50}$
	32x32	x_{51}	ρ_{51}	$\lambda_{51} + \beta_{51}$
	$x_{47} + ... + x_{51} \leq 1$			
register windows	8	base	0	0
	16	x_{30}	ρ_{30}	$\lambda_{30} + \beta_{30}$
	17	x_{31}	ρ_{31}	$\lambda_{31} + \beta_{31}$
	18	x_{32}	ρ_{32}	$\lambda_{32} + \beta_{32}$
	19	x_{33}	ρ_{33}	$\lambda_{33} + \beta_{33}$
	20	x_{34}	ρ_{34}	$\lambda_{34} + \beta_{34}$
	21	x_{35}	ρ_{35}	$\lambda_{35} + \beta_{35}$
	22	x_{36}	ρ_{36}	$\lambda_{36} + \beta_{36}$
	23	x_{37}	ρ_{37}	$\lambda_{37} + \beta_{37}$
	24	x_{38}	ρ_{38}	$\lambda_{38} + \beta_{38}$
	25	x_{39}	ρ_{39}	$\lambda_{39} + \beta_{39}$
	26	x_{40}	ρ_{40}	$\lambda_{40} + \beta_{40}$
	27	x_{41}	ρ_{41}	$\lambda_{41} + \beta_{41}$
	28	x_{42}	ρ_{42}	$\lambda_{42} + \beta_{42}$
	29	x_{43}	ρ_{43}	$\lambda_{43} + \beta_{43}$
	30	x_{44}	ρ_{44}	$\lambda_{44} + \beta_{44}$
	31	x_{45}	ρ_{45}	$\lambda_{45} + \beta_{45}$
	32	x_{46}	ρ_{46}	$\lambda_{46} + \beta_{46}$
	$x_{30} + ... + x_{46} \leq 1$			

Figure 4.7: Integer Unit formulation - variables and constraints

Chapter 5

Evaluation of the Technique

In this section we analyze the impact of our assumption of parameter independence. The naive approach of comparing our solution to the one obtained by generating all LEON configurations exhaustively is infeasible, as discussed in Section 3. Therefore, we take the approach of evaluating our technique on a small *subsystem* of LEON. With this subsystem, it now becomes feasible to generate all configurations exhaustively. Then, we run our optimization algorithm on this subsystem and compare the solution from this to the one from exhaustive configurations. The fact that the two solutions are close verify that our assumption of parameter independence is acceptable.

The subsystem we chose for evaluation is *dcache*. We chose this subsystem because we had manually optimized it for BLASTN application in [37]. The cache subsystem has pronounced variations in application performance and chip resource utilization, for changes in parameter values. As enumerated in Section 4, dcache has 7 reconfigurable parameters—number of sets, size of each set, associativity, line size, replacement policy, fast read and write. The number of values these parameters take are 4, 7, 4, 2, 3, 2 and 2 respectively. The exhaustive combinations of the parameter values are 2,688 and it would take at least 56 days to generate these configurations. The excessive time required is not scalable and therefore we consider only two parameters—number of sets and set size, which result in 28 combinations. We chose these two parameters because perturbing them affects both LUTs and BRAM utilization, at varying degrees. The base configuration has 1 set of 4KB size.

The optional parameters in Tomlab MINLP when left empty are computed by the tool. Such parameters include gradient vector, Hessian matrix, Hessian pattern matrix, constraint gradient, upper bound of the expected solution (used for cutting branches), constraint Jacobian, and second part of Lagrangian function.

5.1 Benchmark I - BLASTN

5.1.1 Analysis of Parameter Independence Assumption

Figure 5.1 shows BLASTN's runtime and chip resource costs for the *exhaustive* combinations of dcache parameters of sets and set size. Optimizing for runtime, a simple sort yields the optimal configuration of 2 sets of 16KB each (i.e.) a total of 32KB. The performance gain is 3.63% over the base configuration, utilizing no additional LUTs but 39% more BRAM than the base configuration.

We then compare this solution to the one from our optimization approach shown in Figure 5.2. Optimizing only for application runtime, the configuration we select is 1x32 = 32KB, which is the same cache size as selected by the exhaustive search although organized slightly differently. The performance gain with this configuration is 3.61%, which is 0.02% less than the optimal configuration from the exhaustive approach; LUTs utilization is 1% less here and BRAM utilization is the same. For these evaluations, we set $w_1 = 100$ and $w_2 = 0$ in the objective function.

The fact that our optimization was able to achieve performance gain within 0.02% difference from the exhaustive solution and with 1% reduction in FPGA usage despite the assumption of parameter independence, is very encouraging.

5.1.2 Analysis of Cost Approximations

BLASTN's runtime on the base LEON configuration is 10.60 seconds. The runtime on the exhaustive configurations (of the dcache parameters of sets and set size) range from 10.22 seconds (-3.63%) on 2x16 configuration to 10.71 seconds (+1.03%) on 1x1 configuration. The linear runtime approximations performed by the optimization algorithm for the configurations not input directly in our model are presented in Figure 5.3. The maximum deviation of the approximation is 1.08%, in the cases of 2x16 and 1x1 configurations.

LUTs utilization on the base LEON configuration is 39%. The LUTs utilization on the exhaustive configurations (of the dcache parameters of sets and set size) is also 39% except

Base configuration				
1	4	10.60	39	51
BLASTN: exhaustive: dcache sets,setsize				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
1	1	10.710	38	47
1	2	10.639	38	48
1	4	10.601	39	51
1	8	10.537	39	56
1	16	10.504	38	68
1	32	10.218	38	90
2	1	10.577	39	49
2	2	10.549	39	51
2	4	10.529	39	56
2	8	10.499	39	68
2	16	10.217	39	90
3	1	10.560	39	51
3	2	10.538	39	55
3	4	10.515	39	62
3	8	10.446	39	79
4	1	10.547	39	53
4	2	10.527	39	58
4	4	10.499	39	68
4	8	10.219	39	90
Dcache exhaustive - optimal BLASTN runtime				
2	16	10.22	39	90

Figure 5.1: BLASTN on exhaustive configurations of dcache parameters of sets and setsize

in the case of 1 set, where it is only 38%. The linear approximations performed by the optimization algorithm for the configurations not supplied directly in the model are presented in Figure 5.3. The approximations of LUTs matches in the cases of 2x8, 3x8 and 4x8 and is less by 1% in all other cases. The difference comes from the measurements obtained by varying set size parameter while holding number of sets at 1. These measurements are 38% each except in the case of 8KB set size when it is 39%.

BRAM utilization on the base LEON configuration is 51%. The utilization on dcache exhaustive configurations (of parameters sets and set size) range from 47% (-4%) for 1x1

Base configuration				
1	4	10.60	39	51
BLASTN: optimizer: dcache sets,setsize ($w_1 = 100, w_2 = 0$)				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
2	4	10.529	39	56
3	4	10.515	39	62
4	4	10.499	39	68
1	1	10.710	38	47
1	2	10.639	38	48
1	4	10.601	39	51
1	8	10.537	39	56
1	16	10.504	38	68
1	32	10.218	38	90
Dcache optimization for BLASTN runtime; the solution configuration is a point we provided in the model				
1	32	10.218	38	90

Figure 5.2: Dcache optimization for BLASTN runtime

to 90% (+39%) for 4x8 configurations. The nonlinear approximations performed by the optimization algorithm for the configurations not input directly in the model are presented in Figure 5.3. The maximum deviation of the approximation is -2%, in the cases of 2x8, 3x2, 3x8, 4x2 and 4x8 configurations.

Future work can further investigate the deviations and explore more sophisticated approximations.

5.2 Benchmark II - CommBench DRR

5.2.1 Analysis of Parameter Independence Assumption

Figure 5.4 shows DRR's runtime and chip resource costs for the *exhaustive* combinations of dcache parameters of sets and set size. Optimizing for runtime, a simple sort yields two optimal configurations of 1x32 and 2x16 (i.e.) a total of 32KB. The performance gain is

12.21% over the base configuration, utilizing no additional LUTs but 39% more BRAM than the base configuration.

We then compare this solution to the one from our optimization approach shown in Figure 5.5. Optimizing only for application runtime, the configuration we select is 2x16 = 32KB, which is one of the two configurations selected by the exhaustive approach. For these evaluations, we set $w_1 = 100$ and $w_2 = 0$ in the objective function.

The solution of 2x16 configuration is a data point that we do *not* input directly in the model. Therefore, it serves as a proof for our claim in Section 3.2 that the optimization algorithm considers points not directly provided by us. The cost approximation for this configuration as well as other data points not provided directly in our model are discussed in Section 5.2.2. Finally, the fact that we are able to *build* this configuration proves that we generate *valid* configurations. This is in fact true for all our results.

The fact that our optimization selects the same configuration as the exhaustive approach shows that our assumption of parameter independence is well acceptable.

5.2.2 Analysis of Cost Approximations

DRR's runtime on the base LEON configuration is 297.98 seconds. The runtime on the exhaustive configurations (of the dcache parameters of sets and set size) range from 261.61 seconds (-12.21%) on 2x16 to 356.50 seconds (+19.64%) on 1x1 configuration. The linear runtime approximations performed by the optimization algorithm for the configurations not input directly in our model are presented in Figure 5.6. The maximum deviation of the approximation is 17.82%, in the case of 4x1 configuration.

The approximations of LUTs and BRAM utilization are application-independent and are discussed in Section 5.1.2.

5.3 Benchmark III - CommBench FRAG

5.3.1 Analysis of Parameter Independence Assumption

Figure 5.7 shows FRAG's runtime and chip resource costs for the *exhaustive* combinations of dcache parameters of sets and set size. Optimizing for runtime, a simple sort yields the optimal configuration of 1x32 = 32KB. The performance gain is 1.91% over the base configuration, utilizing 1% less LUTs and 39% more BRAM than the base configuration.

We then compare this solution to the one from our optimization approach shown in Figure 5.8. Optimizing only for application runtime, the configuration we select is 2x16 = 32KB, which is of the same cache size selected by the exhaustive approach, but organized slightly differently, similar to what we saw for BLASTN. The performance gain with this configuration is 1.91% or more precisely 1.91240%, which is 0.00006% less than the gain of 1.91246% with the exhaustive approach; the LUTs utilization is 1% more but BRAM utilization is the same. For these evaluations, we set $w_1 = 100$ and $w_2 = 0$ in the objective function.

As we discussed in Section 5.2.1, 2x16 here is again a data point that we do *not* input directly in the model. The cost approximation for this configuration as well as other data points not provided directly in our model are presented in Section 5.3.2.

The fact that our optimization was able to achieve the same performance gain, utilizing 1% more LUTs but same BRAM as the solution from the exhaustive approach once again proves that the assumption of parameter independence is acceptable.

5.3.2 Analysis of Cost Approximations

FRAG's runtime on the base LEON configuration is 150.75 seconds. The runtime on the exhaustive configurations (of the dcache parameters of sets and set size) range from 147.87 seconds (-1.91%) on 1x32 to 152.04 seconds (0.85%) on 1x1 configuration. The linear runtime approximations performed by the optimization algorithm for the configurations

not input directly in our model are presented in Figure 5.9. The maximum deviation of the approximation is 3.65%, in the case of 2x1 configuration.

The approximations of LUTs and BRAM utilization are application-independent and are discussed in Section 5.1.2.

5.4 Benchmark IV - BYTE Arith

Arith is different from the applications discussed so far in that it is not memory access intensive and hence variations in dcache configuration do not have any impact on the application's runtime. Figure 5.10 shows this.

5.5 Summary of Evaluation

Evaluation results of all our benchmarks are summarized in Figure 5.11. Arith is not memory access intensive and hence does not exhibit variations in performance on different dcache configurations. For all other applications, the total cache size selected from optimization match the respective cache sizes selected from exhaustive approaches. In the cases of DRR and FRAG, the cache configurations also match.

There are other key observations. The cache configurations chosen for both DRR and FRAG are 2x16. These are *not* configurations that we input directly in the model. This demonstrates that while we construct the search space considering parameters in their own dimensions, the optimization algorithm considers points in between and picks a solution that is simultaneously reconfigured in many dimensions. We then build these configurations to measure the actual chip resource utilization and application runtime. The fact that we are able to build the solutions proves that we generate *valid* configurations. Finally, the configurations selected for the different applications are indeed application-specific.

Base configuration								
1	4	Dir	10.601	N/A	39	N/A	51	N/A
Cost approximations for BLASTN, on dcache exhaustive(sets,setsize)								
nsets	Setsz	Direct/	Runtime(sec)		LUTs(%)		BRAM(%)	
	(KB)	Apprxm	Actual	Apprxm	Actual	Approx	Actual	Approx
1	1	Dir	10.710	N/A	38	N/A	47	N/A
1	2	Dir	10.639	N/A	38	N/A	48	N/A
1	4	Dir	10.601	N/A	39	N/A	51	N/A
1	8	Dir	10.537	N/A	39	N/A	56	N/A
1	16	Dir	10.504	N/A	38	N/A	68	N/A
1	32	Dir	10.218	N/A	38	N/A	90	N/A
2	1	App	10.577	10.638	39	38	49	48
2	2	App	10.549	10.567	39	38	51	50
2	4	Dir	10.529	N/A	39	N/A	56	N/A
2	8	App	10.499	10.465	39	39	68	66
2	16	App	10.217	10.432	39	38	90	90
3	1	App	10.560	10.624	39	38	51	50
3	2	App	10.538	10.553	39	38	55	53
3	4	Dir	10.515	N/A	39	N/A	62	N/A
3	8	App	10.446	10.450	39	39	79	77
4	1	App	10.547	10.608	39	38	53	52
4	2	App	10.527	10.537	39	38	58	56
4	4	Dir	10.499	N/A	39	N/A	68	N/A
4	8	App	10.219	10.434	39	39	90	88
dcache exhaustive - optimal BLASTN runtime								
2	16	App	10.217	10.432	39	38	90	90

Figure 5.3: Cost approximations for BLASTN on exhaustive configurations of dcache parameters of sets and setsize

Base configuration				
1	4	297.98	39	51
DRR: exhaustive: dcache sets,setsize				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
1	1	356.50	38	47
1	2	317.07	38	48
1	4	297.98	39	51
1	8	283.66	39	56
1	16	271.60	38	68
1	32	261.61	38	90
2	1	305.39	39	49
2	2	274.52	39	51
2	4	268.33	39	56
2	8	261.91	39	68
2	16	261.61	39	90
3	1	279.43	39	51
3	2	268.89	39	55
3	4	263.35	39	62
3	8	261.61	39	79
4	1	271.77	39	53
4	2	267.28	39	58
4	4	261.66	39	68
4	8	261.61	39	90
dcache exhaustive - optimal DRR runtime				
1	32	261.61	38	90
2	16	261.61	39	90

Figure 5.4: DRR on exhaustive configurations of dcache parameters of sets and setsize

Base configuration				
1	4	297.98	39	51
DRR: optimizer: dcache sets,setsize ($w_1 = 100, w_2 = 0$)				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
2	4	268.33	39	56
3	4	263.35	39	62
4	4	261.66	39	68
1	1	356.50	38	47
1	2	317.07	38	48
1	4	297.98	39	51
1	8	283.66	39	56
1	16	271.60	38	68
1	32	261.61	38	90
Dcache optimization for DRR runtime; costs based on the actual build of the solution configuration				
2	16	261.61	39	90

Figure 5.5: Dcache optimization for DRR runtime

Base configuration								
1	4	Dir	297.98	N/A	39	N/A	51	N/A
Cost approximations for DRR, on dcache exhaustive(sets,setsize)								
nsets	Setsz	Direct/	Runtime(sec)		LUTs(%)		BRAM(%)	
	(KB)	Apprxm	Actual	Apprxm	Actual	Approx	Actual	Approx
1	1	Dir	356.50	N/A	38	N/A	47	N/A
1	2	Dir	317.07	N/A	38	N/A	48	N/A
1	4	Dir	297.98	N/A	39	N/A	51	N/A
1	8	Dir	283.66	N/A	39	N/A	56	N/A
1	16	Dir	271.60	N/A	38	N/A	68	N/A
1	32	Dir	261.61	N/A	38	N/A	90	N/A
2	1	App	305.39	326.85	39	38	49	48
2	2	App	274.52	287.42	39	38	51	50
2	4	Dir	268.33	N/A	39	N/A	56	N/A
2	8	App	261.91	254.00	39	39	68	66
2	16	App	261.61	241.94	39	38	90	90
3	1	App	279.43	321.87	39	38	51	50
3	2	App	268.89	282.44	39	38	55	53
3	4	Dir	263.35	N/A	39	N/A	62	N/A
3	8	App	261.61	249.03	39	39	79	77
4	1	App	271.77	320.18	39	38	53	52
4	2	App	267.28	280.75	39	38	58	56
4	4	Dir	261.66	N/A	39	N/A	68	N/A
4	8	App	261.61	247.33	39	39	90	88
dcache exhaustive - optimal DRR runtime								
2	16	App	261.61	241.94	39	38	90	90

Figure 5.6: Cost approximations for DRR on exhaustive configurations of dcache parameters of sets and setsize

Base configuration				
1	4	150.75	39	51
FRAG: exhaustive: dcache sets,setsize				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
1	1	154.60	38	47
1	2	152.04	38	48
1	4	150.75	39	51
1	8	150.22	39	56
1	16	148.79	38	68
1	32	147.87	38	90
2	1	149.62	39	49
2	2	149.56	39	51
2	4	149.43	39	56
2	8	148.56	39	68
2	16	147.87	39	90
3	1	149.51	39	51
3	2	149.45	39	55
3	4	149.05	39	62
3	8	147.90	39	79
4	1	149.51	39	53
4	2	149.36	39	58
4	4	148.55	39	68
4	8	147.87	39	90
dcache exhaustive - optimal FRAG runtime				
1	32	147.87	38	90

Figure 5.7: FRAG on exhaustive configurations of dcache parameters of sets and setsize

Base configuration				
1	4	150.75	39	51
FRAG: optimizer: dcache sets,setsize ($w_1 = 100, w_2 = 0$)				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
2	4	149.43	39	56
3	4	149.05	39	62
4	4	148.55	39	68
1	1	154.60	38	47
1	2	152.04	38	48
1	4	150.75	39	51
1	8	150.22	39	56
1	16	148.79	38	68
1	32	147.87	38	90
Dcache optimization for FRAG runtime; costs based on the actual build of the solution configuration				
2	16	147.87	39	90

Figure 5.8: Dcache optimization for FRAG runtime

Base configuration								
1	4	Dir	150.75	N/A	39	N/A	51	N/A
Cost approximations for FRAG, on dcache exhaustive(sets,setsize)								
nsets	Setsz (KB)	Direct/ Apprxm	Runtime(sec) Actual	Apprxm	LUTs(%) Actual	Approx	BRAM(%) Actual	Approx
1	1	Dir	154.60	N/A	38	N/A	47	N/A
1	2	Dir	152.04	N/A	38	N/A	48	N/A
1	4	Dir	150.75	N/A	39	N/A	51	N/A
1	8	Dir	150.22	N/A	39	N/A	56	N/A
1	16	Dir	148.79	N/A	38	N/A	68	N/A
1	32	Dir	147.87	N/A	38	N/A	90	N/A
2	1	App	149.62	153.28	39	38	49	48
2	2	App	149.56	150.71	39	38	51	50
2	4	Dir	149.43	N/A	39	N/A	56	N/A
2	8	App	148.56	148.89	39	39	68	66
2	16	App	147.87	147.47	39	38	90	90
3	1	App	149.51	152.91	39	38	51	50
3	2	App	149.45	150.34	39	38	55	53
3	4	Dir	149.05	N/A	39	N/A	62	N/A
3	8	App	147.90	148.52	39	39	79	77
4	1	App	149.51	152.40	39	38	53	52
4	2	App	149.36	149.84	39	38	58	56
4	4	Dir	148.55	N/A	39	N/A	68	N/A
4	8	App	147.87	148.02	39	39	90	88
dcache exhaustive - optimal FRAG runtime								
2	16	App	147.87	147.47	39	38	90	90

Figure 5.9: Cost approximations for FRAG on exhaustive configurations of dcache parameters of sets and setsize

Base configuration				
1	4	32.33	39	51
Arith: exhaustive: dcache sets,setsize				
nsets	Setsz(KB)	Runtime(sec)	LUTs(%)	BRAM(%)
1	1	32.33	38	47
1	2	32.33	38	48
1	4	32.33	39	51
1	8	32.33	39	56
1	16	32.33	38	68
1	32	32.33	38	90
2	1	32.33	39	49
2	2	32.33	39	51
2	4	32.33	39	56
2	8	32.33	39	68
2	16	32.33	39	90
3	1	32.33	39	51
3	2	32.33	39	55
3	4	32.33	39	62
3	8	32.33	39	79
4	1	32.33	39	53
4	2	32.33	39	58
4	4	32.33	39	68
4	8	32.33	39	90
Dcache optimization for Arith runtime				
Optimizing for runtime, any of the above configurations can be selected.				

Figure 5.10: Arith on exhaustive configurations of dcache parameters of sets and setsize

Optimizer: dcache sets,setsize ($w_1 = 100, w_2 = 0$)					
	Sets	Setsz(KB)	Time(sec)	LUT%	BRAM%
BLASTN					
Exhaust	2	16	10.220	39	90
Optimiz	1	32	10.218	38	90
CommBench DRR					
Exhaust	1	32	261.609	38	90
	2	16	261.609	39	90
Optimiz	2	16	261.609	39	90
CommBench FRAG					
Exhaust	1	32	147.869	38	90
Optimiz	2	16	147.869	39	90
BYTE Arith					
Exhaust	No effect, as application is not data intensive				
Optimiz	No effect, as application is not data intensive				

Figure 5.11: Dcache optimization for BLASTN, DRR, FRAG, Arith runtimes

Chapter 6

Results

The research objectives of our experiments were threefold: find out how much improvement we gain from the application-specific microarchitecture customization, demonstrate that the customization is indeed application-specific, and analyze the cost approximations performed by the optimization algorithm. The results in Section 5 discussed all three for a subset of dcahe parameters. This section presents results for *all* LEON parameters discussed in Section 4.

We begin by presenting the runtime and chip resource costs for all our benchmarks We then present application-specific optimization results, first optimizing for application performance over chip resources and then *vice versa*. Finally, we build the configurations selected by the optimization and compare the actual costs against the approximations performed by the optimization algorithm.

Hardware resource utilizations of the different LEON configurations generated by assuming parameter independence, along with runtimes for BLASTN, DRR, FRAG and Arith applications (on the different processor configurations) are presented in Figures Figure 6.1, Figure 6.3, Figure 6.5, and Figure 6.7. The figure also shows ρ_i, λ_i and β_i, which are the percentage differences of runtime, LUT and BRAM utilization costs and listed in Figure 4.1. Hardware resource utilizations are application-independent.

BLASTN runtimes range from 10.12 seconds on the configuration using a 32x32 hardware multiplier to 14.21 seconds on the configuration using an *iterative* hardware multiplier. The distribution of the costs are better visualized in Figure 6.2.

DRR runtimes range from 261.61 seconds on the configuration using a 1x32 dcache (1 set of 32 KB size) to 359.18 seconds on the configuration using an *iterative* hardware *multiplier*. The distribution of the costs are plotted in Figure 6.4.

FRAG runtimes range from 145.02 seconds on the configuration not using ICC hold to 179.58 seconds on the configuration using a 1x1 icache (1 set of 1 KB size). The distribution of the costs are plotted in Figure 6.6.

Arith runtimes range from 30.65 seconds on the configuration using a 32x32 hardware multiplier to 44.50 seconds on the configuration using an *iterative* hardware multiplier. The distribution of the costs are plotted in Figure 6.8.

6.1 Application Performance Optimization

We optimized for application performance over hardware resource utilization by setting $w_1 = 100$ and $w_2 = 1$ in the equation in Section 4.2. For the four applications that we ran, namely BLASTN, DRR, FRAG and Arith, Figure 6.9 presents the parameters reconfigured from the base configuration, along with results from the actual build of the resulting configurations. Based on the latter, runtime decrease for the four applications are 11.59%, 19.39%, 6.15% and 6.49%, over the runtimes on their respective base configurations. The linear approximations performed by our optimization algorithm estimate the performance improvements to be 11.77%, 39.14%, 7.67% and 6.49% for the four applications. The range of overestimation is 0–19.75%.

The performance gains came at the expense of additional chip resources. The increase in chip resource utilization, expressed as a tuple of LUTs and BRAM, is $(0\%, 39\%)$, $(0\%, 39\%)$, $(8\%, 42\%)$ and $(1\%, -3\%)$ respectively. The approximations performed by the optimization algorithm estimate LUTs and BRAM utilizations to be (-4%,36%), (-4%,41%), (-4%,44%) and (-2%,-4%). We consistently underestimate LUTs utilization; our estimates for BRAM are mixed, from -2% to 3%.

6.1.1 Cost Approximations

As we saw in Section 4, we simplified the cost function for LUTs to be linear while leaving it nonlinear for BRAM. To evaluate the simplification, we also present what the nonlinear approximations would be for LUTs in Figure 6.9. As seen there, the nonlinear approximations are slightly worse. In addition, to demonstrate how *better* the nonlinear cost function *is* over the linear for BRAM, we present the linear approximations also.

We present here cost approximations only for the solutions, as against presenting them for all possible configurations as we did in Section 5. This is because the number of possible configurations is exponential with the number of parameter values as discussed in Section 3 and since we are considering all reconfigurable subsystems and parameters of LEON here, it is infeasible to enumerate all possible configurations.

Parameter(x_i)	Base	Runtime(sec)	ρ_i	LUTs(%)	λ_i	BRAM(%)	β_i
base	N/A	10.60	0.00	39	0	51	0
icachesets2	1	10.60	0.00	39	0	56	5
icachesets3	1	10.60	0.00	39	0	62	11
icachesets4	1	10.60	0.00	39	0	68	17
icachesetsz1	4	10.62	0.19	39	0	47	-4
icachesetsz2	4	10.60	0.00	39	0	48	-3
icachesetsz8	4	10.60	0.00	38	-1	56	5
icachesetsz16	4	10.60	0.00	38	-1	68	17
icachesetsz32	4	10.60	0.00	39	0	90	39
icachelinesz4	8	10.60	0.00	38	-1	51	0
icachereplacelrr	rand	10.60	0.00	39	0	56	5
icachereplacelru	rand	10.60	0.00	40	1	56	5
dcachesets2	1	10.53	-0.68	39	0	56	5
dcachesets3	1	10.51	-0.81	39	0	62	11
dcachesets4	1	10.50	-0.97	39	0	68	17
dcachesetsz1	4	10.71	1.03	38	-1	47	-4
dcachesetsz2	4	10.64	0.36	38	-1	48	-3
dcachesetsz8	4	10.54	-0.61	39	0	56	5
dcachesetsz16	4	10.50	-0.91	38	-1	68	17
dcachesetsz32	4	10.22	-3.61	38	-1	90	39
dcachelinesz4	8	10.58	-0.20	39	0	51	0
dcachereplacelrr	rand	10.53	-0.67	39	0	56	5
dcachereplacelru	rand	10.52	-0.76	39	0	56	5
nofastjump	yes	10.60	0.00	38	-1	51	0
noicchold	yes	10.24	-3.42	39	0	51	0
nofastdecode	yes	10.60	0.00	39	0	51	0
lddelay2	1	11.23	5.95	39	0	51	0
cachedrfast	false	10.60	0.00	39	0	51	0
nodivider	radix2	10.60	0.00	37	-2	51	0
noinfer	infer	10.72	1.13	39	0	51	0
nwindows16	8	10.60	0.00	39	0	53	2
nwindows17	8	10.60	0.00	39	0	53	2
nwindows18	8	10.60	0.00	39	0	53	2
nwindows19	8	10.60	0.00	39	0	53	2
nwindows20	8	10.60	0.00	39	0	53	2
nwindows21	8	10.60	0.00	39	0	53	2
nwindows22	8	10.60	0.00	39	0	53	2
nwindows23	8	10.60	0.00	39	0	53	2
nwindows24	8	10.60	0.00	39	0	53	2
nwindows25	8	10.60	0.00	39	0	53	2
nwindows26	8	10.60	0.00	39	0	53	2
nwindows27	8	10.60	0.00	39	0	53	2
nwindows28	8	10.60	0.00	39	0	53	2
nwindows29	8	10.60	0.00	39	0	53	2
nwindows30	8	10.60	0.00	39	0	53	2
nwindows31	8	10.60	0.00	39	0	53	2
nwindows32	8	10.60	0.00	39	0	58	7
mulpipefalse	true	10.60	0.00	39	0	51	0
multiplieriterative	16x16	14.21	34.04	37	-2	51	0
multiplierm32x8	16x17	10.60	0.00	39	0	51	0
multiplierm32x16	16x18	10.36	-2.27	39	0	51	0
multiplierm32x32	16x19	10.12	-4.54	40	1	51	0
cachedwfasttrue	false	10.60	0.00	39	0	51	0

Figure 6.1: BLASTN runtime, chip resource costs

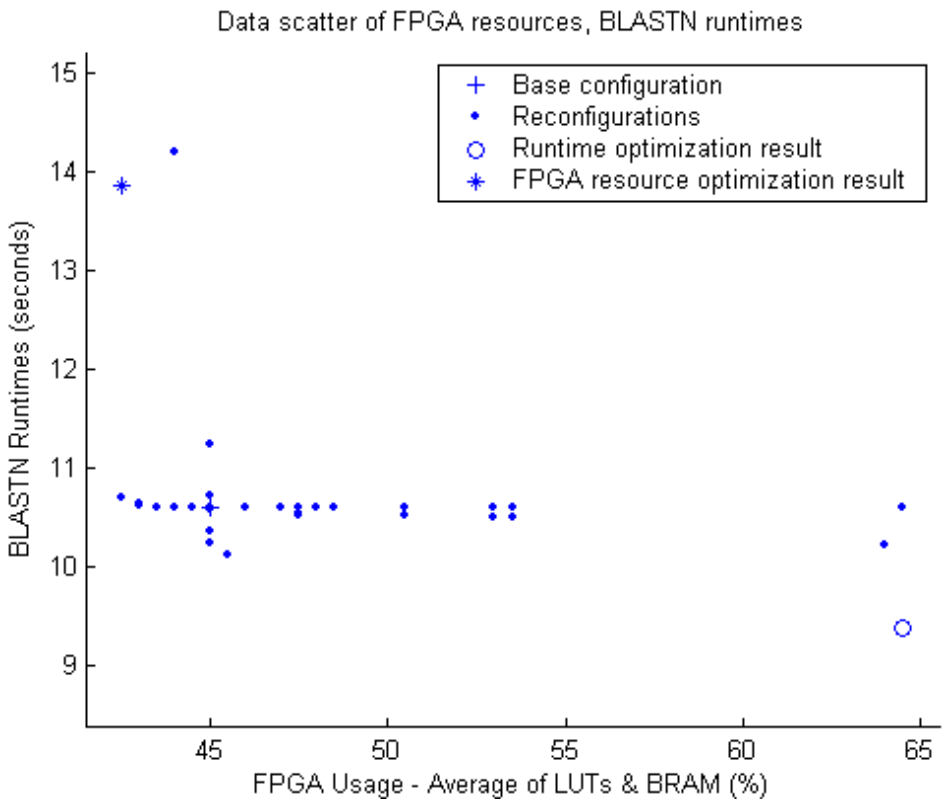

Figure 6.2: Data scatter plot of FPGA resources and BLASTN runtimes

6.1.2 Comparison with Dcache Optimization

Given our assumption of parameter independence, an interesting observation is to compare the customization in dcache to the one from optimizing only dcache in Section evaluation. However, the weights in the objective function are slightly different–for the former, $w_1 = 100$ and $w_2 = 1$ and for the latter $w_1 = 100$ and $w_2 = 0$. Despite the minor difference in the weights, the resulting dcache configurations are identical for all applications except Arith. For Arith, it was 1x4 in Section evaluation but here it is 1x1. This is because of the chip resource consideration resulting from $w_2 = 1$.

Parameter(x_i)	Base	Runtime(sec)	ρ_i	LUTs(%)	λ_i	BRAM(%)	β_i
base	N/A	297.9786842	0.00	39	0	51	0
icachesets2	1	297.9786842	0.00	39	0	56	5
icachesets3	1	297.9786842	0.00	39	0	62	11
icachesets4	1	297.9786842	0.00	39	0	68	17
icachesetsz1	4	298.1214379	0.05	39	0	47	-4
icachesetsz2	4	297.9786847	0.00	39	0	48	-3
icachesetsz8	4	297.9786842	0.00	38	-1	56	5
icachesetsz16	4	297.9786842	0.00	38	-1	68	17
icachesetsz32	4	297.9786842	0.00	39	0	90	39
icachelinesz4	8	297.9786845	0.00	38	-1	51	0
icachereplacelrr	rand	297.9786842	0.00	39	0	56	5
icachereplacelru	rand	297.9786842	0.00	40	1	56	5
dcachesets2	1	268.3250568	-9.95	39	0	56	5
dcachesets3	1	263.3489335	-11.62	39	0	62	11
dcachesets4	1	261.6559325	-12.19	39	0	68	17
dcachesetsz1	4	356.5043431	19.64	38	-1	47	-4
dcachesetsz2	4	317.0707616	6.41	38	-1	48	-3
dcachesetsz8	4	283.6556616	-4.81	39	0	56	5
dcachesetsz16	4	271.5967389	-8.85	38	-1	68	17
dcachesetsz32	4	261.6085536	-12.21	38	-1	90	39
dcachelinesz4	8	288.7742601	-3.09	39	0	51	0
dcachereplacelrr	rand	267.993657	-10.06	39	0	56	5
dcachereplacelru	rand	268.2213798	-9.99	39	0	56	5
nofastjump	yes	297.9786842	0.00	38	-1	51	0
noicchold	yes	284.730941	-4.45	39	0	51	0
nofastdecode	yes	297.9786842	0.00	39	0	51	0
lddelay2	1	345.0590222	15.80	39	0	51	0
cachedrfast	false	297.9786842	0.00	39	0	51	0
nodivider	radix2	297.9786842	0.00	37	-2	51	0
noinfer	infer	300.0188886	0.68	39	0	51	0
nwindows16	8	297.9786842	0.00	39	0	53	2
nwindows17	8	297.9786842	0.00	39	0	53	2
nwindows18	8	297.9786842	0.00	39	0	53	2
nwindows19	8	297.9786842	0.00	39	0	53	2
nwindows20	8	297.9786842	0.00	39	0	53	2
nwindows21	8	297.9786842	0.00	39	0	53	2
nwindows22	8	297.9786842	0.00	39	0	53	2
nwindows23	8	297.9786842	0.00	39	0	53	2
nwindows24	8	297.9786842	0.00	39	0	53	2
nwindows25	8	297.9786842	0.00	39	0	53	2
nwindows26	8	297.9786842	0.00	39	0	53	2
nwindows27	8	297.9786842	0.00	39	0	53	2
nwindows28	8	297.9786842	0.00	39	0	53	2
nwindows29	8	297.9786842	0.00	39	0	53	2
nwindows30	8	297.9786842	0.00	39	0	53	2
nwindows31	8	297.9786842	0.00	39	0	53	2
nwindows32	8	297.9786842	0.00	39	0	58	7
mulpipefalse	true	297.9786842	0.00	39	0	51	0
multiplieriterative	16x16	359.1848046	20.54	37	-2	51	0
multiplierm32x8	16x17	297.9786842	0.00	39	0	51	0
multiplierm32x16	16x18	293.8982764	-1.37	39	0	51	0
multiplierm32x32	16x19	289.8178687	-2.74	40	1	51	0
cachedwfasttrue	false	297.9786842	0.00	39	0	51	0

Figure 6.3: DRR runtime, chip resource costs

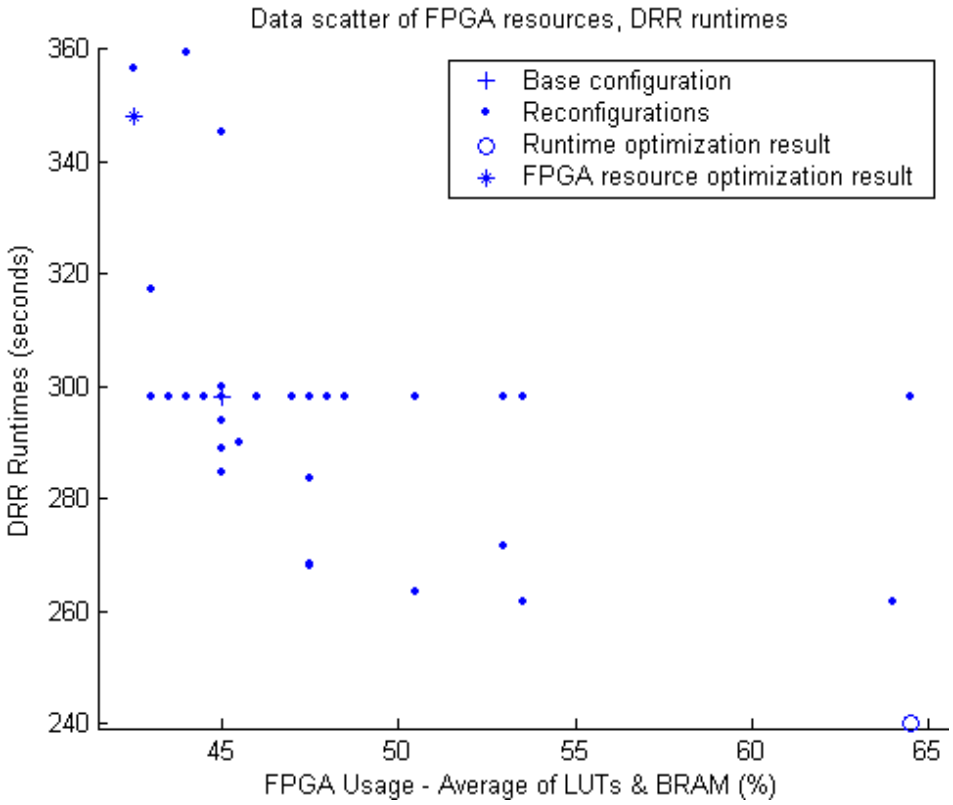

Figure 6.4: Data scatter plot of FPGA resources and DRR runtimes

6.2 FPGA Resource Optimization

We optimized for hardware resource utilization over application performance by setting $w_1 = 1$ and $w_2 = 100$ in the equation in Section 4.2. Figure 6.10 shows the parameters reconfigured from the base configuration, along with results from the actual build of the resulting configurations. Based on the latter, decrease in chip resource utilization are $(2\%, 3\%)$, $(2\%, 3\%)$, $(3\%, 3\%)$ and $(1\%, 3\%)$. The approximations performed by our optimization algorithm estimate the chip resource savings to be $(5\%, 4\%)$, $(7\%, 4\%)$, $(7\%, 4\%)$ and $(5\%, 4\%)$. We consistently overestimate the resource savings; for LUTs, the range is 3—5% and for BRAM, it is always 1%.

Similar to the cost approximations presented in the section on application runtime optimization, here also, we present the nonlinear approximations for LUTs and linear approximations for BRAM in Figure 6.10. However, the parameter for number of cache sets is not

Parameter(x_i)	Base	Runtime(sec)	ρ_i	LUTs(%)	λ_i	BRAM(%)	β_i
base	N/A	150.7520498	0.00	39	0	51	0
icachesets2	1	150.7520498	0.00	39	0	56	5
icachesets3	1	150.7520498	0.00	39	0	62	11
icachesets4	1	150.7520498	0.00	39	0	68	17
icachesetsz1	4	179.5841508	19.13	39	0	47	-4
icachesetsz2	4	167.425314	11.06	39	0	48	-3
icachesetsz8	4	150.7520498	0.00	38	-1	56	5
icachesetsz16	4	150.7520498	0.00	38	-1	68	17
icachesetsz32	4	150.7520498	0.00	39	0	90	39
icachelinesz4	8	150.7520522	0.00	38	-1	51	0
icachereplacelrr	rand	150.7520498	0.00	39	0	56	5
icachereplacelru	rand	150.7520498	0.00	40	1	56	5
dcachesets2	1	149.4255093	-0.88	39	0	56	5
dcachesets3	1	149.0546878	-1.13	39	0	62	11
dcachesets4	1	148.5512996	-1.46	39	0	68	17
dcachesetsz1	4	154.6042898	2.56	38	-1	47	-4
dcachesetsz2	4	152.0361298	0.85	38	-1	48	-3
dcachesetsz8	4	150.2175721	-0.35	39	0	56	5
dcachesetsz16	4	148.7932656	-1.30	38	-1	68	17
dcachesetsz32	4	147.8689698	-1.91	38	-1	90	39
dcachelinesz4	8	150.1549326	-0.40	39	0	51	0
dcachereplacelrr	rand	149.4908616	-0.84	39	0	56	5
dcachereplacelru	rand	149.4673321	-0.85	39	0	56	5
nofastjump	yes	150.7520498	0.00	38	-1	51	0
noicchold	yes	145.017526	-3.80	39	0	51	0
nofastdecode	yes	150.7520498	0.00	39	0	51	0
lddelay2	1	168.9770567	12.09	39	0	51	0
cachedrfast	false	150.7520498	0.00	39	0	51	0
nodivider	radix2	150.7520498	0.00	37	-2	51	0
noinfer	infer	150.9158898	0.11	39	0	51	0
nwindows16	8	150.7520498	0.00	39	0	53	2
nwindows17	8	150.7520498	0.00	39	0	53	2
nwindows18	8	150.7520498	0.00	39	0	53	2
nwindows19	8	150.7520498	0.00	39	0	53	2
nwindows20	8	150.7520498	0.00	39	0	53	2
nwindows21	8	150.7520498	0.00	39	0	53	2
nwindows22	8	150.7520498	0.00	39	0	53	2
nwindows23	8	150.7520498	0.00	39	0	53	2
nwindows24	8	150.7520498	0.00	39	0	53	2
nwindows25	8	150.7520498	0.00	39	0	53	2
nwindows26	8	150.7520498	0.00	39	0	53	2
nwindows27	8	150.7520498	0.00	39	0	53	2
nwindows28	8	150.7520498	0.00	39	0	53	2
nwindows29	8	150.7520498	0.00	39	0	53	2
nwindows30	8	150.7520498	0.00	39	0	53	2
nwindows31	8	150.7520498	0.00	39	0	53	2
nwindows32	8	150.7520498	0.00	39	0	58	7
mulpipefalse	true	150.7520498	0.00	39	0	51	0
multiplieriterative	16x16	155.6672498	3.26	37	-2	51	0
multiplierm32x8	16x17	150.7520498	0.00	39	0	51	0
multiplierm32x16	16x18	150.4243698	-0.22	39	0	51	0
multiplierm32x32	16x19	150.0966898	-0.43	40	1	51	0
cachedwfasttrue	false	150.7520498	0.00	39	0	51	0

Figure 6.5: FRAG runtime, chip resource costs

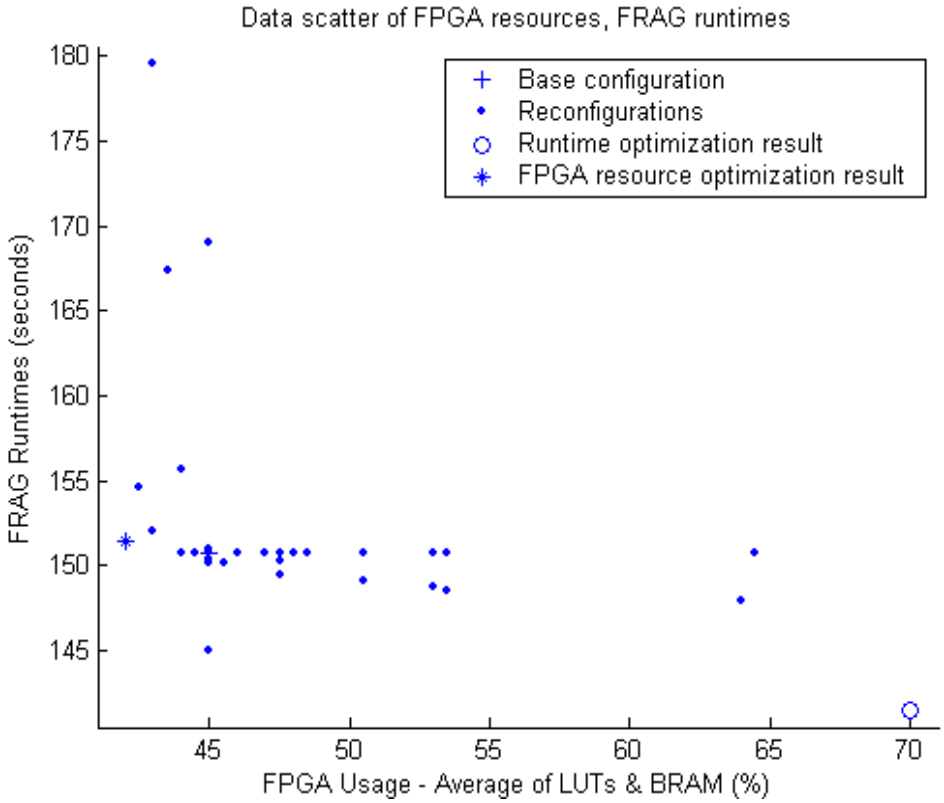

Figure 6.6: Data scatter plot of FPGA resources and FRAG runtimes

reconfigured from the base configuration for any application here and therefore, there are no differences between the linear and nonlinear cost approximations.

The savings in FPGA resources are at the expense of application performance, often a significant loss – 30.66% for BLASTN, 16.76% for DRR, 0.43% for FRAG and 36.34% for Arith.

Parameter(x_i)	Base	Runtime(sec)	ρ_i	LUTs(%)	λ_i	BRAM(%)	β_i
base	N/A	32.33	0.00	39	0	51	0
icachesets2	1	32.33	0.00	39	0	56	5
icachesets3	1	32.33	0.00	39	0	62	11
icachesets4	1	32.33	0.00	39	0	68	17
icachesetsz1	4	32.33	0.00	39	0	47	-4
icachesetsz2	4	32.33	0.00	39	0	48	-3
icachesetsz8	4	32.33	0.00	38	-1	56	5
icachesetsz16	4	32.33	0.00	38	-1	68	17
icachesetsz32	4	32.33	0.00	39	0	90	39
icachelinesz4	8	32.33	0.00	38	-1	51	0
icachereplacelrr	rand	32.33	0.00	39	0	56	5
icachereplacelru	rand	32.33	0.00	40	1	56	5
dcachesets2	1	32.33	0.00	39	0	56	5
dcachesets3	1	32.33	0.00	39	0	62	11
dcachesets4	1	32.33	0.00	39	0	68	17
dcachesetsz1	4	32.33	0.00	38	-1	47	-4
dcachesetsz2	4	32.33	0.00	38	-1	48	-3
dcachesetsz8	4	32.33	0.00	39	0	56	5
dcachesetsz16	4	32.33	0.00	38	-1	68	17
dcachesetsz32	4	32.33	0.00	38	-1	90	39
dcachelinesz4	8	32.33	0.00	39	0	51	0
dcachereplacelrr	rand	32.33	0.00	39	0	56	5
dcachereplacelru	rand	32.33	0.00	39	0	56	5
nofastjump	yes	32.33	0.00	38	-1	51	0
noicchold	yes	31.91	-1.30	39	0	51	0
nofastdecode	yes	32.33	0.00	39	0	51	0
lddelay2	1	35.27	9.09	39	0	51	0
cachedrfast	false	32.33	0.00	39	0	51	0
nodivider	radix2	Error	N/A	37	-2	51	0
noinfer	infer	32.75	1.30	39	0	51	0
nwindows16	8	32.33	0.00	39	0	53	2
nwindows17	8	32.33	0.00	39	0	53	2
nwindows18	8	32.33	0.00	39	0	53	2
nwindows19	8	32.33	0.00	39	0	53	2
nwindows20	8	32.33	0.00	39	0	53	2
nwindows21	8	32.33	0.00	39	0	53	2
nwindows22	8	32.33	0.00	39	0	53	2
nwindows23	8	32.33	0.00	39	0	53	2
nwindows24	8	32.33	0.00	39	0	53	2
nwindows25	8	32.33	0.00	39	0	53	2
nwindows26	8	32.33	0.00	39	0	53	2
nwindows27	8	32.33	0.00	39	0	53	2
nwindows28	8	32.33	0.00	39	0	53	2
nwindows29	8	32.33	0.00	39	0	53	2
nwindows30	8	32.33	0.00	39	0	53	2
nwindows31	8	32.33	0.00	39	0	53	2
nwindows32	8	32.33	0.00	39	0	58	7
mulpipefalse	true	32.33	0.00	39	0	51	0
multiplieriterative	16x16	44.50	37.64	37	-2	51	0
multiplierm32x8	16x17	32.33	0.00	39	0	51	0
multiplierm32x16	16x18	31.49	-2.60	39	0	51	0
multiplierm32x32	16x19	30.65	-5.19	40	1	51	0
cachedwfasttrue	false	32.33	0.00	39	0	51	0

Figure 6.7: Arith runtime, chip resource costs

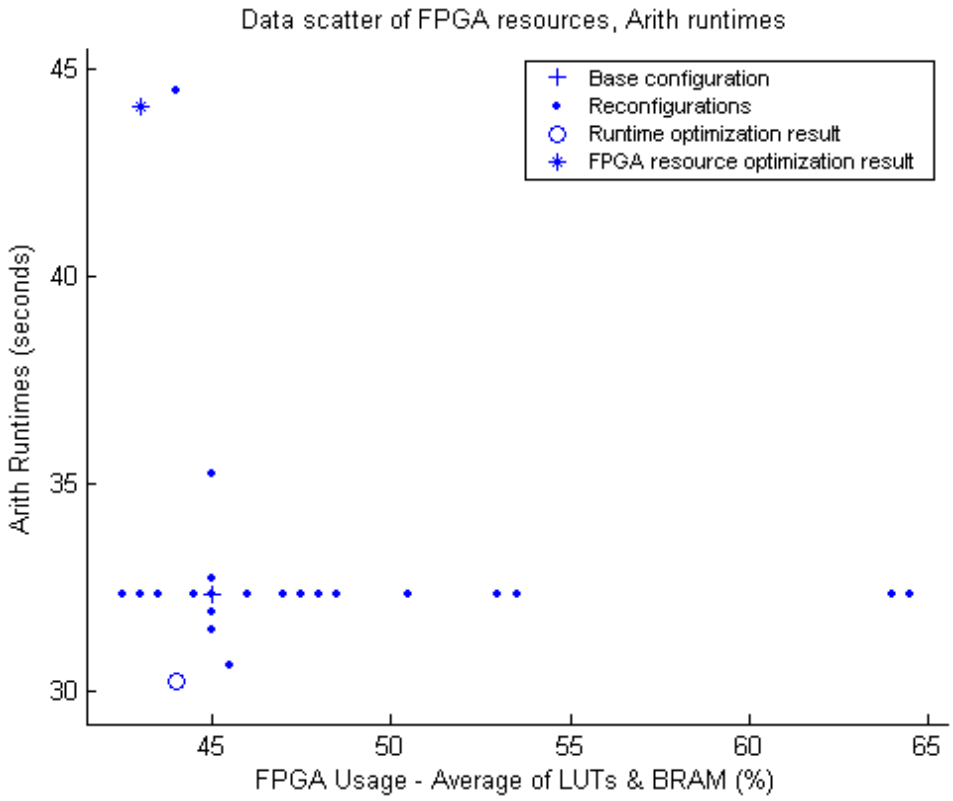

Figure 6.8: Data scatter plot of FPGA resources and Arith runtimes

Application runtime optimization ($w_1 = 100, w_2 = 1$)					
Param	Base	BLAST	DRR	FRAG	Arith
icachsetsz	4	2	2	4	4
icachlinesz	8	4	4	4	4
dcachsets	1	1	2	2	1
dcachsetsz	4	32	16	16	1
dcachlinesz	8	4	4	4	8
dcachreplace	rand	LRU	LRR	LRU	rand
fastjump	on	off	off	off	off
icchold	on	off	off	off	off
divider	radix2	none	none	none	radix2
multiplier	16x16	32x32	32x32	32x32	32x32
Base configuration					
runtime(sec)	N/A	10.60	297.98	150.75	32.33
Cost approximations by the optimizer					
runtime(sec)	N/A	9.35	181.35	139.20	30.23
LUTs%	39	35	35	35	37
LUTs%-nonlin	*39*	*35*	*34*	*34*	*37*
BRAM%	51	87	92	95	47
BRAM%-lin	*51*	*87*	*75*	*78*	*47*
Actual synthesis					
runtime(sec)	N/A	9.37	240.20	141.48	30.23
LUTs%	39	39	39	47	40
BRAM%	51	90	90	93	48

Figure 6.9: Application runtime optimization

FPGA resource optimization ($w_1 = 1, w_2 = 100$) * indicates sub-optimal solution					
Param	Base	BLAST*	DRR*	FRAG	Arith*
icachsetsz	4	2	2	4	2
icachlinesz	8	4	4	4	4
dcachsets	1	1	1	1	1
dcachsetsz	4	2	2	1	2
dcachlinesz	8	4	4	4	8
dcachreplace	rand	rand	rand	rand	rand
fastjump	on	off	off	off	off
icchold	on	off	off	off	off
divider	radix2	none	none	none	radix2
registers	8	28*	31*	8	30*
multiplier	16x16	iter	iter	iter	iter
Base configuration					
runtime(sec)	N/A	10.60	297.98	150.75	32.33
Cost approximations by the optimizer					
runtime(sec)	N/A	13.86	355.82	153.19	44.08
LUTs%	39	34	32	32	34
LUTs%-nonlin	*39*	*34*	*32*	*32*	*34*
BRAM%	51	47	47	47	47
BRAM%-lin	*51*	*47*	*47*	*47*	*47*
Actual synthesis					
runtime(sec)	N/A	13.85	347.91	151.40	44.08
LUTs%	39	37	37	36	38
BRAM%	51	48	48	48	48

Figure 6.10: FPGA resource optimization

Chapter 7

Conclusion

Applications for constrained embedded systems are subject to strict runtime and resource utilization bounds. With soft core processors, application developers can customize the processor for their application, constrained by available hardware resources but aimed at high application performance.

The more reconfigurable the processor is, the more options the application developers have for customization and hence, increased potential for improving application performance. However, such customization entails developing in-depth familiarity with the parameters, in order to configure them effectively. This is typically infeasible, given the tight time-to-market pressure on the developers. Alternatively, developers could explore all possible configurations, but being exponential, this is infeasible even given only tens of parameters, as we saw in Section 3.

7.1 Summary of Approach

This thesis presented a heuristic for automatic application-specific reconfiguration of a soft core processor microarchitecture. This approach runs in time that is *linear* with the number of reconfigurable parameters, with an assumption of parameter independence, to make the approach feasible and scalable.

The approach to building the search space is summarized as follows. We begin with the default LEON configuration. We call this the *base* configuration. We then perturb one parameter at a time and build the processor configuration, measuring its chip cost. Thirdly,

we execute the application on each configuration, measuring the runtime. Finally, we formulate these costs into a Binary Integer Nonlinear Program and solve for optimal solution using the commercial solver of Tomlab. The solution obtained is the recommended microarchitecture configuration for the given application.

7.2 Summary of Results

We optimized for application performance over chip resources by setting $w_1 = 100$ and $w_2 = 1$. Figure 7.1 shows the parameters reconfigured from the base configuration, along with results from the actual build of the solution. Based on the latter, runtime decrease for the four applications of BLASTN, DRR, FRAG and Arith are 11.59%, 19.39%, 6.15% and 6.49%, over the runtimes on their respective base configurations. The performance gains come at the expense of additional chip resources.

We optimized for chip resource utilization over application performance by setting $w_1 = 1$ and $w_2 = 100$. Figure 7.2 shows the parameters reconfigured from the base configuration, along with results from the actual build of the solution. Based on the latter, decrease in chip resource utilization are $(2\%, 3\%)$, $(2\%, 3\%)$, $(3\%, 3\%)$ and $(1\%, 3\%)$. The savings in chip resources come at a loss of application performance.

For our experiments, we use SRAM memory. However, if we used DRAM instead, the performance gains we achieve from customization will be more significant because access to DRAM is slower.

7.3 Contributions

In this thesis we developed an automatic optimization technique for application-specific reconfiguration of a soft core processor microarchitecture. We then evaluated the technique by customizing the open source soft core processor of LEON, for some substantive applications of BLASTN, CommBench DRR, CommBench FRAG and BYTE Arith.

Application runtime optimization ($w_1 = 100, w_2 = 1$)					
Param	Base	BLAST	DRR	FRAG	Arith
icachsetsz	4	2	2	4	4
icachlinesz	8	4	4	4	4
dcachsets	1	1	2	2	1
dcachsetsz	4	32	16	16	1
dcachlinesz	8	4	4	4	8
dcachreplace	rand	LRU	LRR	LRU	rand
fastjump	on	off	off	off	off
icchold	on	off	off	off	off
divider	radix2	none	none	none	radix2
multiplier	16x16	32x32	32x32	32x32	32x32
Base configuration					
runtime(sec)	N/A	10.60	297.98	150.75	32.33
Cost approximations by the optimizer					
-runtime%	N/A	11.77	39.14	7.67	6.49
+LUTs%	39	-4	-4	-4	-2
+BRAM%	51	36	41	44	-4
Actual synthesis					
-runtime%	N/A	11.59	19.39	6.15	6.49
+LUTs%	39	0	0	8	1
+BRAM%	51	39	39	42	-3

Figure 7.1: Application runtime optimization

7.3.1 Conclusions Drawn

To evaluate the impact of our simplifying assumption of parameter independence, we generated exhaustive configurations of the parameters of dcache sets and set size and compared its solution to our optimization solution. While the results matched for DRR and FRAG, they differed by 0.02% for BLASTN. This evaluation did not apply to Arith because it is not at all memory access intensive. The close match between our optimization results and exhaustive results demonstrates that the impact of parameter independence is negligible.

These experiments simultaneously demonstrated other characteristics of our solution. The cache configurations selected for BLASTN, DRR and FRAG were 1x32, 2x16 and 2x16

Chip resource optimization ($w_1 = 1, w_2 = 100$) * indicates sub-optimal solution					
Param	Base	BLAST*	DRR*	FRAG	Arith*
icachsetsz	4	2	2	4	2
icachlinesz	8	4	4	4	4
dcachsets	1	1	1	1	1
dcachsetsz	4	2	2	1	2
dcachlinesz	8	4	4	4	8
dcachreplace	rand	rand	rand	rand	rand
fastjump	on	off	off	off	off
icchold	on	off	off	off	off
divider	radix2	none	none	none	radix2
registers	8	28*	31*	8	30*
multiplier	16x16	iter	iter	iter	iter
Base configuration					
runtime(sec)	N/A	10.60	297.98	150.75	32.33
Cost approximations by the optimizer					
+runtime%	N/A	30.66	19	1.62	36.34
-LUTs%	39	5	7	7	5
-BRAM%	51	4	4	4	4
Actual synthesis					
+runtime%	N/A	30.66	16.76	0.43	36.34
-LUTs%	39	2	2	3	1
-BRAM%	90	3	3	3	3

Figure 7.2: Chip resource optimization

respectively. This demonstrates *application-specific* customization. Further, the configuration of 2x16 is *not* a configuration that we provide directly in the model. This demonstrates that while we construct the search space by reconfiguring parameters in their own dimensions, the optimization algorithm considers points in between which are points reconfigured *simultaneously* in many dimensions. Finally, the fact that we are able to successfully *build* our solutions shows that we generate *valid* configurations. The same characteristics were observed when we extended our experiments to customize *all* parameters of LEON's microarchitecture.

While presenting the results of customizing *all* parameters of LEON's microarchitecture, we demonstrated that the application developers can not only optimize for application performance over FPGA resource utilization but also vice versa. The time for generating the processor configurations is in the order of hours but this is performed only once, as the processor configurations are independent of applications being run. The time for optimization itself is very low–on the order of seconds. Given such reasonable time requirements, we demonstrate that our approach is indeed very feasible and scalable, even with a large number of parameters. Further, during the customization process, application developers were not actively involved, even though they control the performance-resource tradeoff. Best of all, application developers were spared from having to develop familiarity with the processor parameters or modifying their application to use our optimization technique.

7.4 Future Work

Future work can recast our nonlinear constraints so that they are convex functions for all values of x_i. This will guarantee that the optimization algorithm finds the global optimum. We can also analyze the cost approximations performed by the optimization algorithm and explore more sophisticated approximations.

In this thesis, we rely on empirical performance measurements to substantiate our simplifying assumption of parameter independence. This can be improved by including measurements of two-parameter interactions in the form of covariance matrices, where the two parameters will be selected from different microarchitecture subsystems. Because of our assumption and the cost approximations resulting from it, there is potential to improve the solution that we obtain from the first search. This is achieved by conducting a local search near the solution obtained from the first search. An even better approach would be not to rely on empirical performance measurements but instead rely on microarchitecture parameter statistics such as cache hits and misses to reason parameter interactions.

As extensions to our model, we can include power and energy optimizations, runtime sampling to facilitate analysis of long-running applications, running applications on an operating system (running on the processor) and supporting ISA level customization. As extensions to our benchmarking, we can include MiBench applications. For long running

applications, we can also use “phase detection” to identify different phases and customize architecture per phase.

By integrating our solution with LEON and other such open source soft core processors, we can contribute back to the community. Finally, and more interestingly, we can evaluate our technique on other configuration and feature management problems.

Appendix A

Liquid Control Packet Formats

This appendix lists control packet formats used for the different commands supported by the Liquid architecture platform, as described in Section 2.1.

The UDP control packet formats for starting and halting LEON (command codes x50, x54 respectively), for reading from memory (command code x60) and for writing to memory (command code x64) are shown in Figures A.1, A.2 and A.3 respectively.

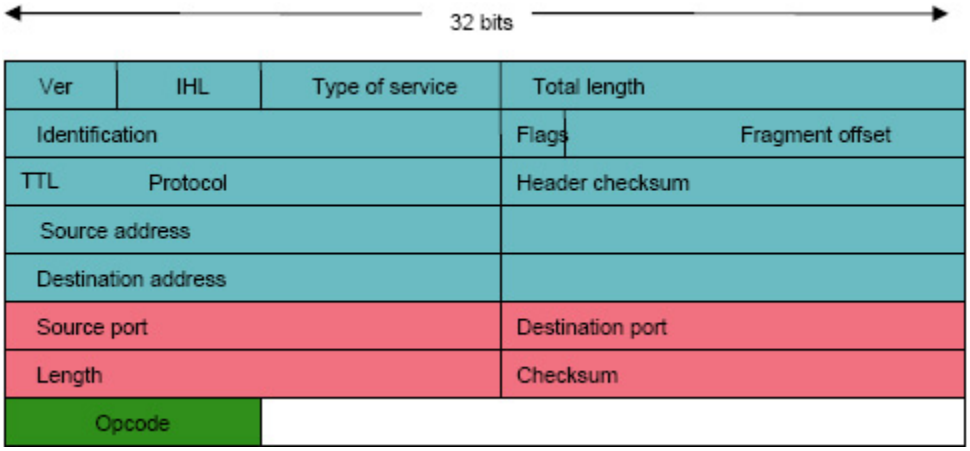

Figure A.1: Liquid architecture control packet format for starting, halting LEON

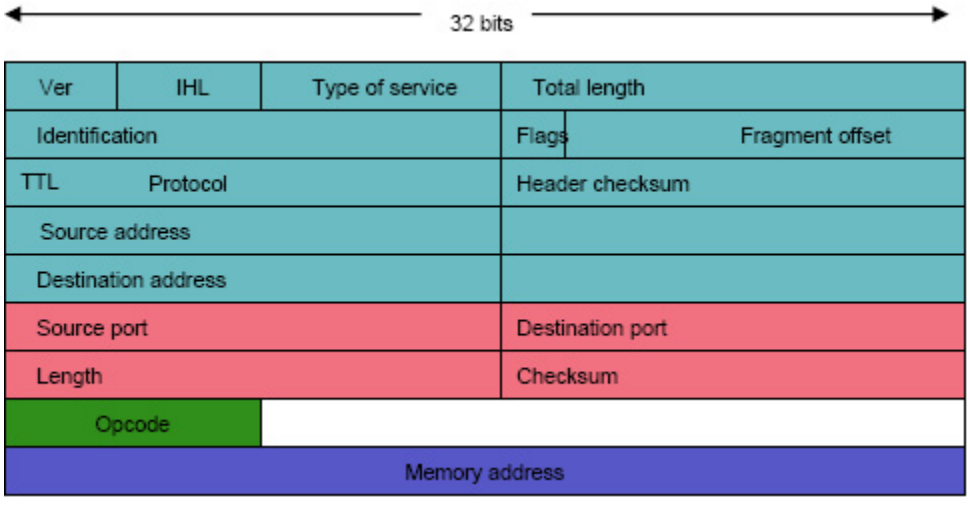

Figure A.2: Liquid architecture control packet format for reading from memory

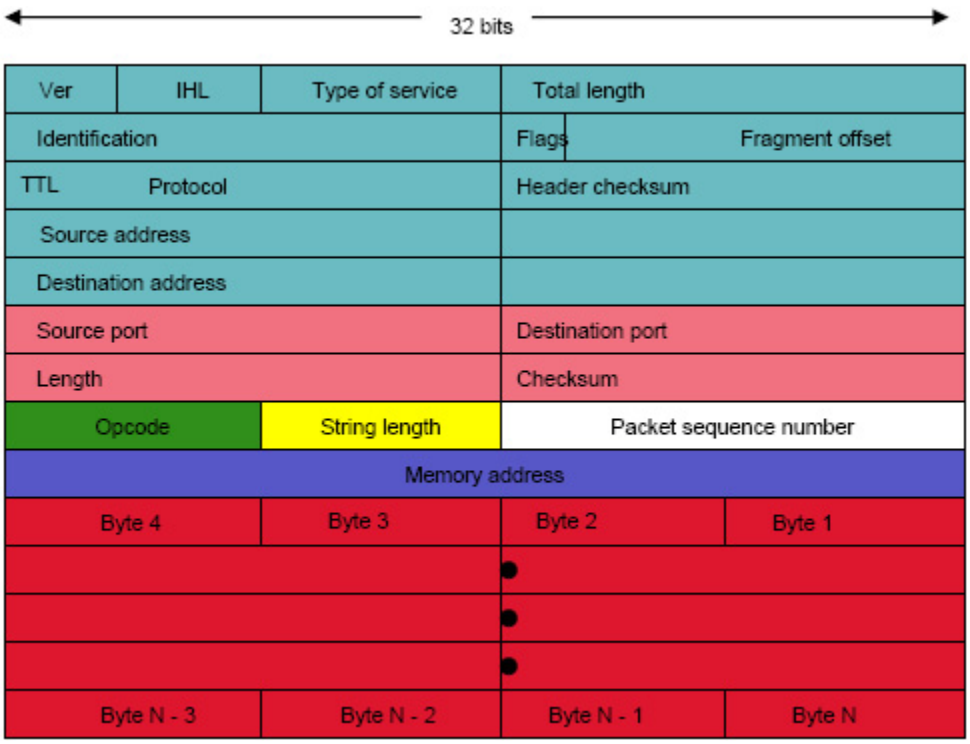

Figure A.3: Liquid architecture control packet format for writing to memory

Appendix B

Default LEON Configuration

This appendix shows the default values for all the reconfigurable parameters in LEON. These values are in device.vhd in the LEON distribution. This is also our “base configuration”.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.target.all;

package device is

  constant syn_config : syn_config_type := (
    targettech => virtex , infer_pads => false, infer_pci => false,
    infer_ram => true, infer_regf => true, infer_rom => false,
    infer_mult => true, rftype => 1, targetclk => gen,
    clk_mul => 1, clk_div => 1, pci_dll => false, pci_sysclk => false );

  constant iu_config : iu_config_type := (
    nwindows => 8, multiplier => m16x16, mulpipe => true,
    divider => radix2, mac => false, fpuen => 0, cpen => false,
    fastjump => true, icchold => true, lddelay => 1, fastdecode => true,
    rflowpow => false, watchpoints => 4, impl => 0, version => 0);

  constant fpu_config : fpu_config_type :=
    (core => meiko, interface => none, fregs => 0, version => 0);

  constant cache_config : cache_config_type := (
    isets => 1, isetsize => 4, ilinesize => 8, ireplace => rnd, ilock => 0,
    dsets => 1, dsetsize => 4, dlinesize => 8, dreplace => rnd, dlock => 0,
```

```
    dsnoop => none, drfast => false, dwfast => false, dlram => false,
    dlramsize => 1, dlramaddr => 16#8F#);

  constant mmu_config : mmu_config_type := (
    enable => 0, itlbnum => 8, dtlbnum => 8, tlb_type => combinedtlb,
    tlb_rep => replruarray, tlb_diag => false );

  constant ahbrange_config  : ahbslv_addr_type :=
        (0,0,0,0,0,0,0,0,1,2,7,7,7,7,7,7);

  constant ahb_config : ahb_config_type := ( masters => 2, defmst => 0,
    split => false, testmod => false);

  constant mctrl_config : mctrl_config_type := (
    bus8en => false, bus16en => false, wendfb => false, ramsel5 => false,
    sdramen => true, sdinvclk => false);

  constant peri_config : peri_config_type := (
    cfgreg => true, ahbstat => false, wprot => false, wdog => false,
    irq2en => false, ahbram => false, ahbrambits => 11, ethen => false );

  constant debug_config : debug_config_type := (
    enable => true, uart => true,
    iureg => false, fpureg => false, nohalt => false, pclow => 2,
    dsuenable => false, dsutrace => false, dsumixed => false,
    dsudpram => false, tracelines => 128);

  constant boot_config : boot_config_type := (boot => prom, ramrws => 3,
    ramwws => 0, sysclk => 25, baud => 9600, extbaud => false,
    pabits => 8);

  constant pci_config : pci_config_type := (
    pcicore => none , ahbmasters => 0, ahbslaves => 0,
    arbiter => false, fixpri => false, prilevels => 4, pcimasters => 4,
    vendorid => 16#0000#, deviceid => 16#0000#, subsysid => 16#0000#,
    revisionid => 16#00#, classcode =>16#000000#, pmepads => false,
    p66pad => false, pcirstall => false);

  constant irq2cfg : irq2type := irq2none;
end;
```

Appendix C

Source Code for Benchmarks

C.1 BLASTN

```
/*
   This program is similar to a hashing scheme used by BLAST
   It has been hardcoded to support a query of 500 strings.
   The hash table is of size 5 times the query  = 2048 locations.
   We are modeling a query of size 11 bases, which implies a string of
   22 bits.
   The database is modeled as a circular buffer of length 17 bases
   (17 is an arbitrarily chosen which can be changed with ease)
*/

/*
   Hashing fucntion, it is a open addressing scheme with double hashing.
   The hash table stores the key (query string).
   The hash table still does not store the position in query,
   though it can added easily later.
*/ /*
   TODO
*/ # define MINT 0x7fffffff # define SIZE 8*1024 //size of the
hash table # define NUM_QUERY 2500 # define NUM_DATABASE 1000000
//*1024*1024 // 100 Mbases database # define mask 4194300 # define
mask1 (SIZE - 1) /* Function Declaration */ unsigned int
addQuery(unsigned int base1, unsigned int *currentString);
unsigned int findMatch(unsigned int base1, unsigned int
*currentString); inline unsigned int computeKey(unsigned int base,
unsigned int currentString); inline unsigned int
```

```
computeBase(unsigned int key); inline unsigned int
computeStep(unsigned int key); unsigned int Rnd(unsigned int *u);
void fillQuery(int qNum); int coreLoop(unsigned int base, unsigned
int step, unsigned int last,
             unsigned int *currentString);

unsigned int hashTable[SIZE]; // It is twice the size of the query

main () {
   int index = 0, counter = 0, found = 0, matches = 0, *ans;
   unsigned int currentString = 348432612, base = 0, random = 0;
   //currentString above is used as a seed also
   ans = (int*)0x40000004; //mem where the #matches is stored
   for (index = 0; index < SIZE; index++) {
      hashTable[index] = 4194304;
   }

   fillQuery(NUM_QUERY);  //populates the hashtable
   // the loop below generates random bases for the database
   for (counter = 0; counter < NUM_DATABASE; counter++) {
      random = Rnd(&random);
      if (random <= MINT / 4) {
       base = 0;
      } else if (random <= MINT / 2) {
         base = 1;
      } else if (random <= ((MINT / 2) + (MINT / 4))){
         base = 2;
      } else {
         base = 3;
      }
      found = findMatch(base, &currentString);
      if (found == 1) {
         matches++;
      }
   }
  //printf ("Total number of matches found = %d\n", matches);
  ans[0] = matches;
}

unsigned int addQuery(unsigned int base1, unsigned int
*currentString) {
```

```
   unsigned int base = 0, step = 0, last = 0, current = 0;
   *currentString = computeKey(base1, *currentString);
   base = computeBase(*currentString);
   step = computeStep(*currentString);
   last = (base + (SIZE - 1) * step) % SIZE;
   current = base;
   while (current != last) { // should be able to check all positions
      if (hashTable[current] == 4194304) {
         hashTable[current] = *currentString;
         return 1;
      } else if (hashTable[current] = *currentString) {
         return 1;
      } else {
         current = (current + step) % SIZE;
      }
   }
   return 0;
}

//uses open address, double hashing
unsigned int findMatch(unsigned int base1, unsigned int
*currentString) {
   unsigned int base, step, last, current;
   *currentString = computeKey(base1, *currentString);
   base = computeBase(*currentString);
   step = computeStep(*currentString);
   last = (base + (SIZE - 1) * step) % SIZE;
   if (coreLoop(base, step, last, currentString)) {
      return 1;
   } else {
      return 0;
   }
}

inline unsigned int computeKey(unsigned int base, unsigned int
currentString) {
   currentString <<= 2;
   currentString &= mask;
   currentString |= base;
   return currentString;
}
```

```
inline unsigned int computeBase(unsigned int key) {
   return (2634706073U * key) & mask1;
}

inline unsigned int computeStep(unsigned int key) {
   return ((1013257199U * key) | 1) & mask1;
}

void fillQuery(int qNum) {
   int success, index;
   unsigned int currentString = 473246;
   unsigned int random = 782333;
   unsigned int base = 0;

   for (index = 0; index < qNum; index++) {
      random = Rnd(&random);
      if (random <= MINT/ 4) {
         base = 0;
      } else if (random <= MINT / 2) {
         base = 1;
      } else if (random <= ((MINT / 2) + (MINT / 4))){
         base = 2;
      } else {
         base = 3;
      }
      success = addQuery(base, &currentString);
      if (success) {
         success = 0;
      } else {
      }
   }
}

int coreLoop(unsigned int base, unsigned int step, unsigned int
last,
             unsigned int *currentString) {
   while (base != last) { // should be able to check all positions
      if (hashTable[base] == *currentString) {
         return 1;
      } else if (hashTable[base] == 4194304) {
```

```
         break;
      } else {
         base = (base + step) % SIZE;
      }
   }
   return 0;
} unsigned int Rnd(unsigned int *u) {
   return ((314159265 * (*u) + 271828182) & MINT);
}
}
```

C.2 Commbench DRR

```
#define LEON #ifndef LEON #include <stdio.h> #include <stdlib.h>
#endif

#define NO_OF_QUEUES 100 //0 to 99 #define NO_OF_ELEMENTS 1000
#define QUANTUM 10 #define MAX_SIZE 125 #define MIN_SIZE 25
#define SEED 38734278 #define MINT 0x7fffffff #define
PKTS_TO_PROCESS 10000000

typedef unsigned int UINT;

struct q_head {
   struct queue *queue;
   struct queue **tail;
   struct q_head *next_q;
   int deficit;
};
struct queue {
   struct queue *next;
   struct q_head *head;
   int size;
};

//Global variables
UINT gRandom; UINT done = 0; UINT elemProcessed = 0; struct q_head
q[NO_OF_QUEUES]; struct queue el[NO_OF_ELEMENTS];
```

```
//Functions used
UINT Random() {
   gRandom = ((314159265 * gRandom + 271828182) & MINT);
   return gRandom;
}
UINT GenQNum(void) {
   return (Random() %
   NO_OF_QUEUES);
}
UINT GenPktSize(void) {
   return (Random() %
   (MAX_SIZE - MIN_SIZE + 1) + MIN_SIZE);
}

void FillNextRequest(void) {
   UINT i, qNum;

   #ifndef LEON
   printf ("Elements processed = %
   u\n", elemProcessed);
   #endif
   if (elemProcessed == PKTS_TO_PROCESS) { done = 1; }

   //Reset the pointers for each batch of request
   for (i = 0; i < NO_OF_QUEUES; i++) {
      q[i].tail = &(q[i].queue);
      q[i].next_q = &(q[(i + 1) %
      NO_OF_QUEUES]);
   }


   for (i = 0; i < NO_OF_ELEMENTS; i++) {
      el[i].size = GenPktSize();
      qNum = GenQNum();
      *(q[qNum].tail) = &(el[i]);
      q[qNum].tail = &(el[i].next);
   }

   elemProcessed += NO_OF_ELEMENTS;
```

```
}

void Schedule(void) {
   UINT i, toProcess = NO_OF_ELEMENTS;
   struct q_head *p;
   p = q;

   while (toProcess > 0) {
      while(p->next_q->queue == 0){ // remove inactive queues
         p->next_q = p->next_q->next_q;
      }

      p = p->next_q;
      p->deficit += QUANTUM;

      while(p->queue && (p->deficit >= p->queue->size)) {
         // transmit all the packets for this queue
         p->deficit -= p->queue->size;
         p->queue = p->queue->next;
         toProcess--;
      }
   }
} #ifdef LEON int *ans = (int *) 0x40000800; #endif

int main(void) {
   //ReportStart(ans++);
   gRandom = SEED;
   while (!done) {
      FillNextRequest();
      Schedule();
   }
   //ReportSuccess(ans++);
   #ifdef LEON
   ans[0] = 1729;
   #endif
}
```

C.3 Commbench Frag

```
//#include <stdio.h>
//#include <stdlib.h>

#define MAX_SIZE 1536 #define MIN_SIZE 1536 #define SEED 38734278
#define MINT 0xffffffff #define PKTS_TO_PROCESS 1024000 //10240000
#define FRAGSIZE 576

typedef unsigned int UINT;

struct ip {
   unsigned char ip_v_hl;      /* version and header length */
   unsigned char ip_tos;       /* type of service */
   unsigned short ip_len;      /* total length */
   unsigned short ip_id;       /* identification */
   unsigned short ip_off;      /* fragment offset field */
   #define  IP_RF 0x8000       /* reserved fragment flag */
   #define  IP_DF 0x4000       /* dont fragment flag */
   #define  IP_MF 0x2000       /* more fragments flag */
   #define  IP_OFFMASK 0x1fff  /* mask for fragmenting bits */
   unsigned char ip_ttl;       /* time to live */
   unsigned char ip_p;         /* protocol */
   unsigned short ip_sum;      /* checksum */
   unsigned int ip_src, ip_dst;/* source and dest address */
}; #define IPBUF 1024 #define FRAGBUF 32

//Global variables
UINT gRandom; struct ip ip[IPBUF]; struct ip frag[FRAGBUF];

#define ADDCARRY(x)  (x > 65535 ? x -= 65535 : x) #define REDUCE
{l_util.l = sum; sum = l_util.s[0] + l_util.s[1]; ADDCARRY(sum);}

//Functions used
UINT Random() {
   gRandom = ((314159265 * gRandom + 271828182) & MINT);
   return gRandom;
}

struct ip GenPkt() {
```

```
   struct ip myip;
   UINT id = 0;

   // remains the same for every ip packet
   myip.ip_v_hl = 0x45;
   myip.ip_tos = 0x10;
   myip.ip_off = IP_OFFMASK & 0;
   myip.ip_ttl = 255;
   myip.ip_sum = 0;

   //randomly generated values
   myip.ip_p = Random() & 0xff;
   myip.ip_src = Random() & 0xffffffff;
   myip.ip_dst = Random() & 0xffffffff;
   myip.ip_len = MIN_SIZE + (Random() %
   (MAX_SIZE - MIN_SIZE + 1));
   myip.ip_id = ++id & 0xffff;

   return myip;
}

void MyMemcopy(void *dest, void *src, int len) {
   UINT i;

   for (i = 0; i < len; i++) {
      *(char *)dest = *(char *)src;
      dest++;
      src++;
   }
}

long InChkSum(register char *buf, register int len) {
   long sum = 0;

   while (len > 1) {
      sum += *((unsigned short *) buf)++;
      if (sum & 0x80000000) {
         sum = (sum & 0xFFFF) + (sum >> 16);
      }
      len -= 2;
   }
```

```
   if (len) // if len is odd
      sum += (unsigned short) *buf;
   while (sum >> 16) {
      sum = (sum & 0xFFFF) + (sum >> 16);
   }
   return ~sum;
}

int Fragment(struct ip *ip) {
   int l;
   int f=0;

   l = ip->ip_len;
   while (l > FRAGSIZE) {
      memcpy(&(frag[f]), ip, sizeof(struct ip));
      frag[f].ip_len = FRAGSIZE;
      frag[f].ip_off = (f*FRAGSIZE) >> 3;
      frag[f].ip_sum = 0;
      frag[f].ip_sum = InChkSum((char *)&(frag[f]), 20);
      f++;
      l-=FRAGSIZE;
   }
   memcpy(&(frag[f]), ip, sizeof(struct ip));
   frag[f].ip_len = l;
   frag[f].ip_off = (f*FRAGSIZE) >> 3;
   frag[f].ip_sum = 0;
   frag[f].ip_sum = InChkSum((char *)&(frag[f]), 20);
   f++;
   return f;
}

void FillIpBuf(void) {
   UINT i;
   for (i = 0; i < IPBUF; i++){
      ip[i] = GenPkt();
   }
} int *ans = (int *) 0x40008000; int main(void) {
   UINT toProcess = PKTS_TO_PROCESS;
   UINT i, f;
   gRandom = SEED;
```

```
   //ReportStart(ans++);
   if (FRAGSIZE != (FRAGSIZE & 0xfffffff8)) {
      //printf("fragsize must be multiple of 8!\n");
      //To code in error notation
      //exit(-1);
   }

   while (toProcess > 0) //was != 0
   {
      FillIpBuf();  // replaces the read method in original commbench

      for (i = 0; i < IPBUF; i++) {
         f = Fragment(&(ip[i]));
      }
      toProcess -= IPBUF;
   }
   //ReportSuccess(ans);
   ans[0] = 1729;
}
```

C.4 BYTE Arith

```
/*************************************************************
 *  The BYTE UNIX Benchmarks - Release 2
 *          Module: arith.c   SID: 2.4 4/17/90 16:45:31
 *
 *************************************************************
 * Bug reports, patches, comments, suggestions should be sent to:
 *
 *  Ben Smith or Rick Grehan at BYTE Magazine
 *  bensmith@bixpb.UUCP    rick_g@bixpb.UUCP
 *
 *************************************************************
 *  Modification Log:
 *  May 12, 1989 - modified empty loops to avoid nullifying
 *                 by optimizing compilers
 *
```

```
 ************************************************************/

#define arithoh #define LEON

char SCCSid[] = "@(#) @(#)arith.c:2.4 -- 4/17/90 16:45:31"; /*
 *  arithmetic test
 *
 */

int *ans = (int *)0x40008000;

main(argc, argv) int argc; char    *argv[]; {
    int iter;
    int i;
    int result;
    //ReportStart(ans++);
    #ifdef LEON
        ans[0] = 666;
        #endif

    iter = 10000;
    while (iter-- > 0)
    {
    /* the loop calls a function to insure that something is done */
    /* the results of the function are thrown away. But a loop with */
    /* unused assignments may get optimized out of existence */
    result = dumb_stuff(i);
        //printf("iter:%
        d result=%
        d\n", iter, result);
    }
    //ReportSuccess(ans);
    #ifdef LEON
       //ans[0] = result;
       ans[0] = 1729;
        #endif
}


/************************ dumb_stuff ******************/
dumb_stuff(i) int i; {
```

```
    int x = 0;
    int y = 0;
    int z = 0;
    //ReportProgress(ans++);
        for (i=2; i<=1050; i++) //was <=101
        {
            x = i;
            y = (x*x)+1;
            z += y/(y-1);
            //(i*2)+100;
            //(i/2)+200;
        }
        //printf("x+y+z=%
        //d\n", x+y+z);
        return(x+y+z);
}
```

Appendix D

LEON Parameterization

LEON, the prototype core we are using, is highly parameterized. Figure D.1 shows these parameters organized into 8 systems viz. synthesis options, clock generation, processor, AMBA, memory controller, peripherals, boot options and VHDL debugging. Each system is shown in subsequent sections.

D.1 LEON Synthesis options

Figure D.2 shows the user interface for synthesis options. The parameters are described in Section 2.4.2.

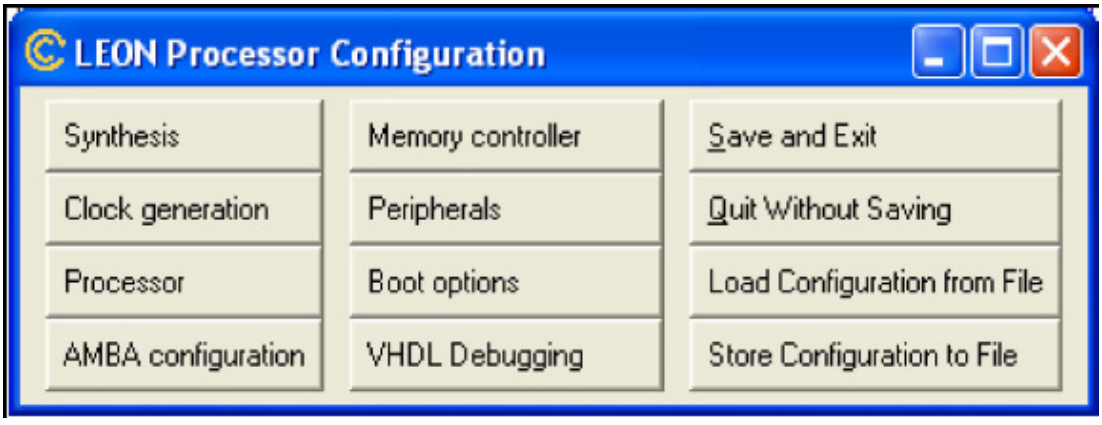

Figure D.1: LEON configuration

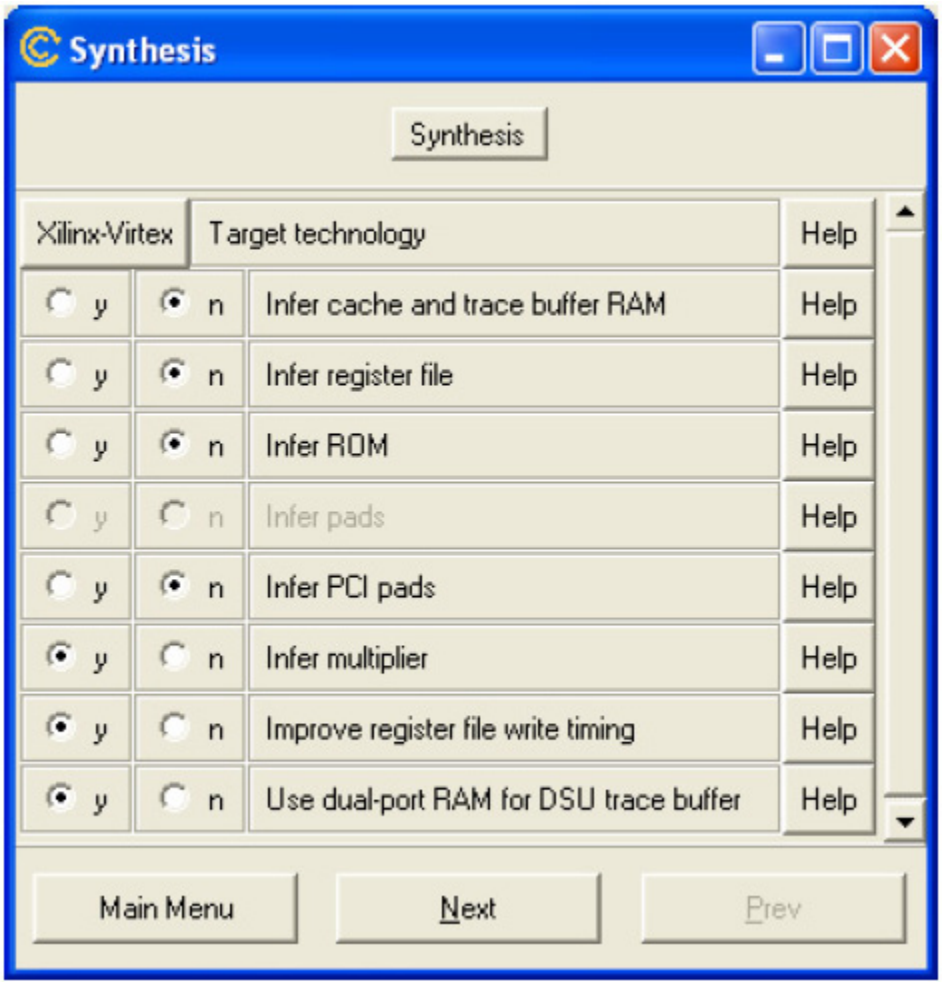

Figure D.2: LEON configuration - Synthesis options

D.2 LEON Clock Generation options

Figure D.3 shows the user interface synthesis options. The parameters are described in Section 2.4.3.

D.3 LEON Processor system

Figure D.4 shows the user interface for processor configuration. The parameters are described in Section 2.4.1.

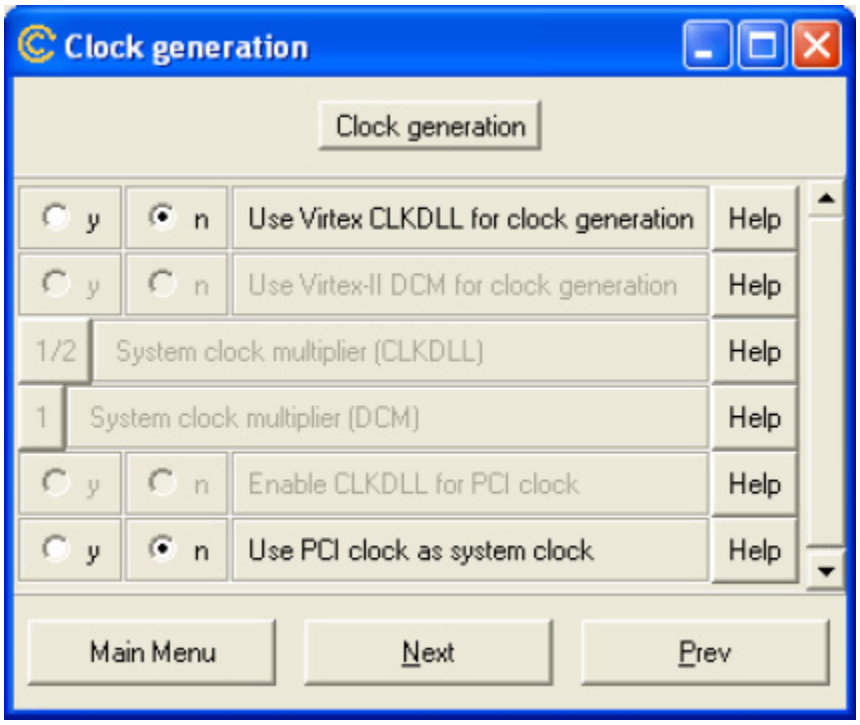

Figure D.3: LEON configuration - Clock generation

D.3.1 Processor Integer Unit

Figure D.5 shows the user interface for processor IU configuration. The parameters are described in Section 2.4.1.

D.3.2 Processor Floating-point Unit

Figure D.6 shows the user interface for processor FPU configuration. The parameters are described in Section 2.4.1.

D.3.3 Co-processor

Figure D.7 shows the user interface for co-processor configuration. The parameters are described in Section 2.4.1.

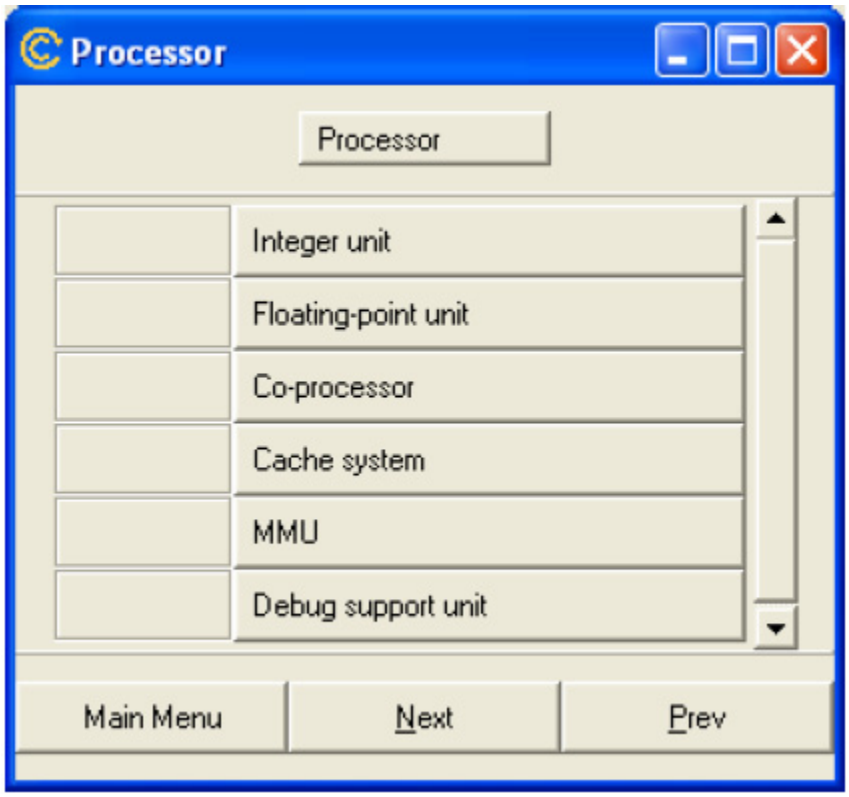

Figure D.4: LEON configuration - Processor system

D.3.4 Processor Cache

Figure 2.4 shows the user interface for processor cache configuration. The parameters are described in Section 2.4.1.

D.3.5 Processor Memory Management Unit

Figure 2.4 shows the user interface for processor MMU configuration. The parameters are described in Section 2.4.1.

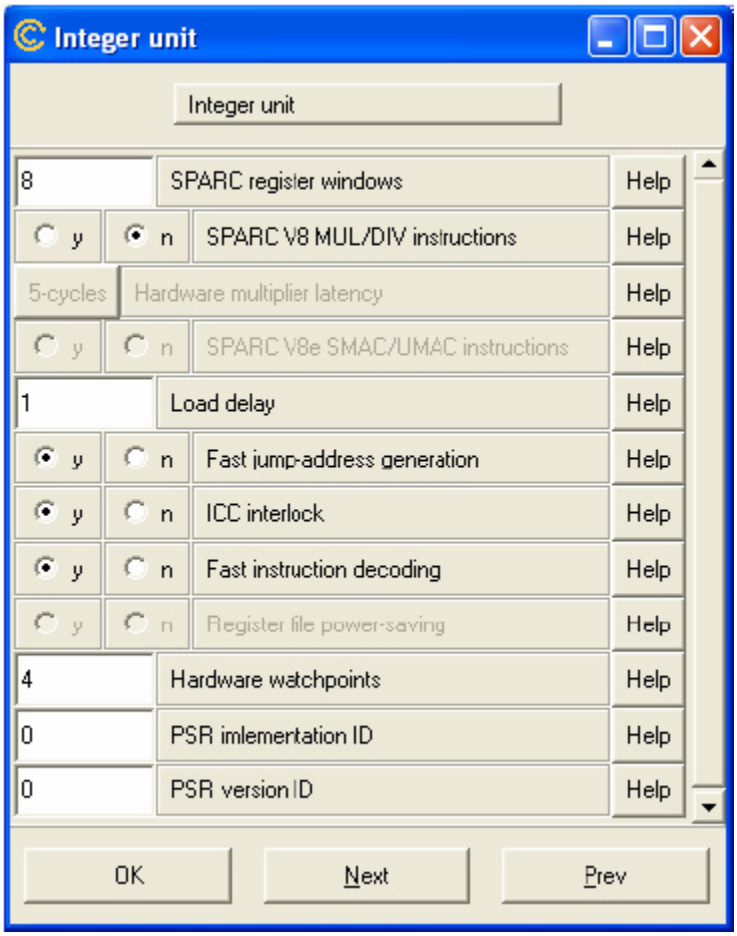

Figure D.5: LEON configuration - Processor Integer Unit

D.3.6 Processor Debug Support Unit

Figure 2.4 shows the user interface for processor DSU configuration. The parameters are described in Section 2.4.1.

D.4 LEON AMBA bus

Figure D.11 shows the user interface for AMBA bus configuration. The parameters are described in Section 2.4.5.

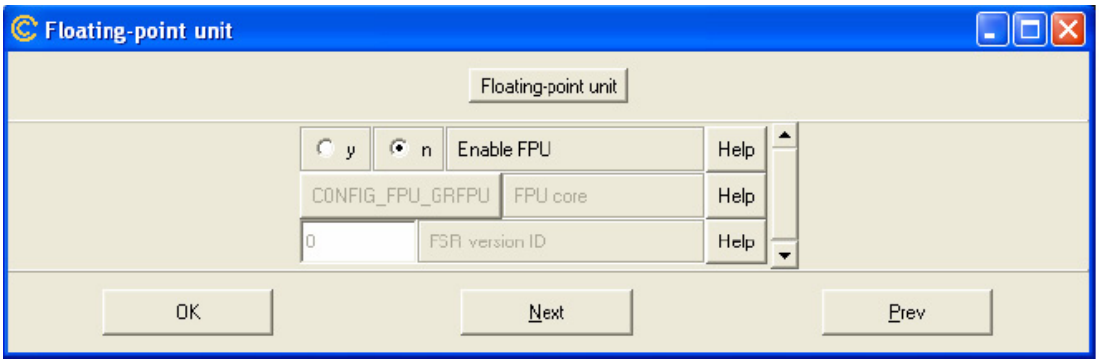

Figure D.6: LEON configuration - Processor Floating-point Unit

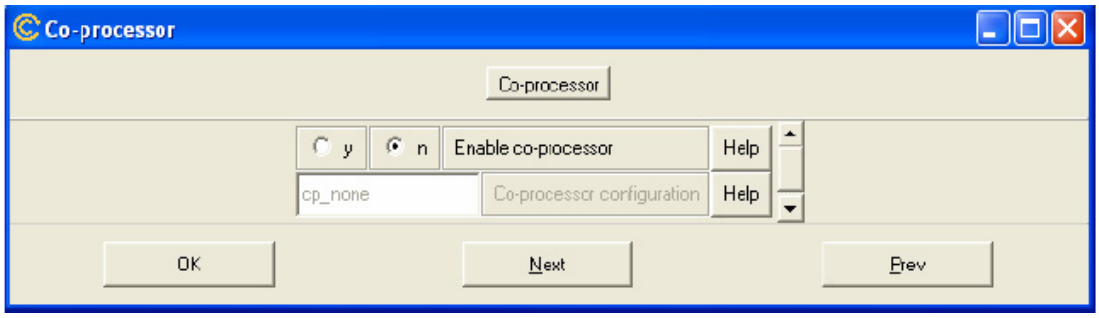

Figure D.7: LEON configuration - Co-processor

D.5 LEON Memory Controller

Figure D.4 shows the user interface for configuring LEON memory controller. The parameters are described in Section 2.4.4.

D.6 LEON Peripherals

Figure D.4 shows the user interface for configuring LEON peripherals. The parameters are described in Section 2.4.6.

D.7 LEON Ethernet Interface

Figure D.4 shows the user interface for configuring LEON ethernet interface. The parameters are described in Section 2.4.6.

D.8 LEON PCI

Figure D.4 shows the user interface for configuring LEON PCI. The parameters are described in Section 2.4.7.

D.9 LEON Boot options

Figure D.4 shows the user interface for configuring LEON boot options. The parameters are described in Section 2.4.8.

D.10 LEON VHDL Debugging

Figure D.4 shows the user interface for configuring LEON VHDL debug options. The parameters are described in Section 2.4.9.

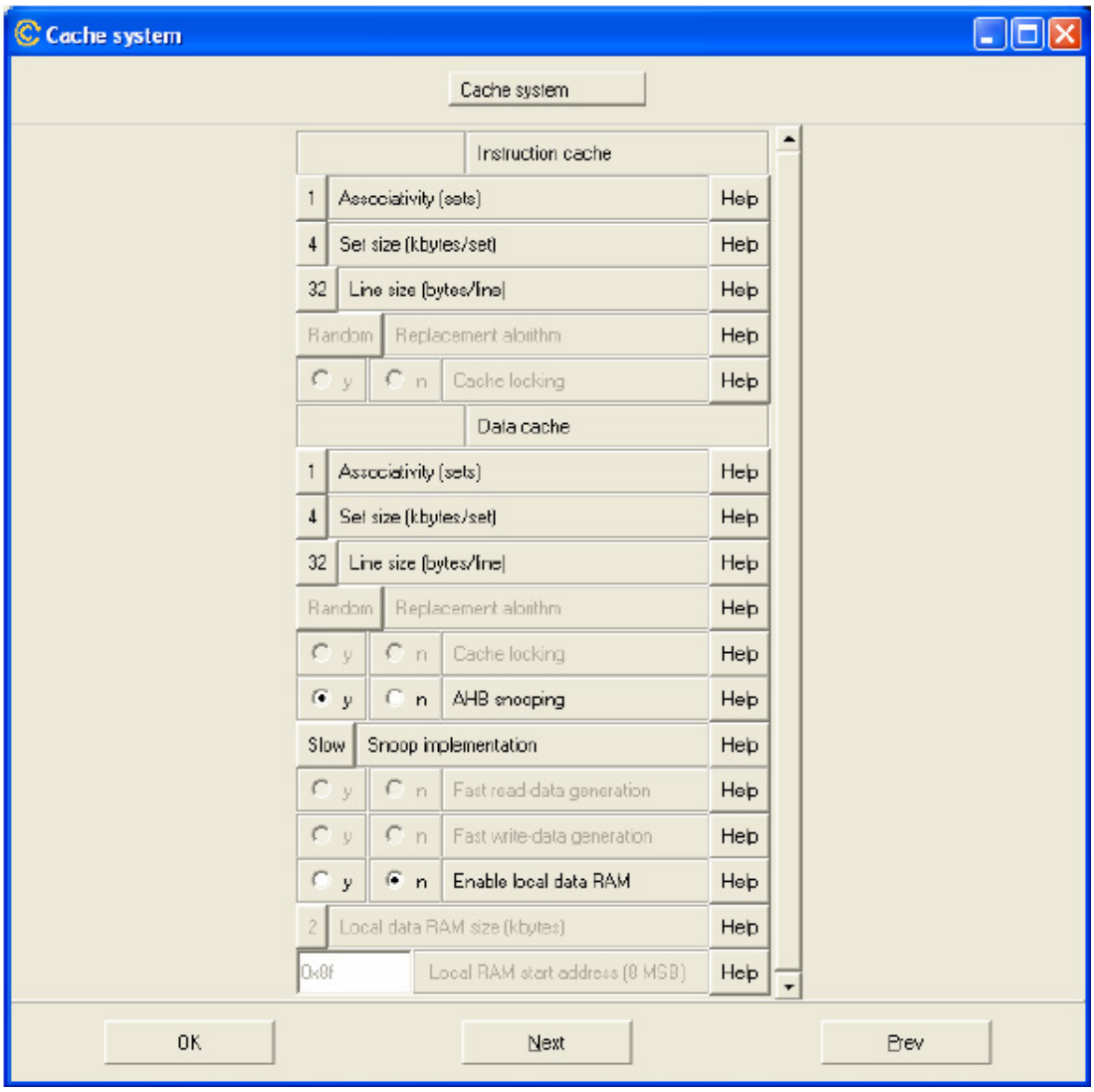

Figure D.8: LEON configuration - Processor Cache

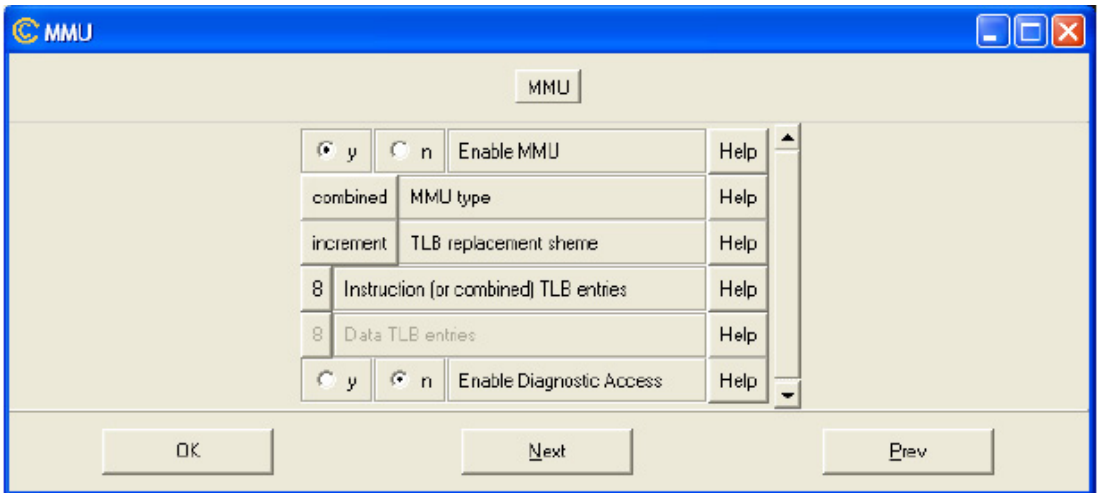

Figure D.9: LEON configuration - Processor Memory Management Unit

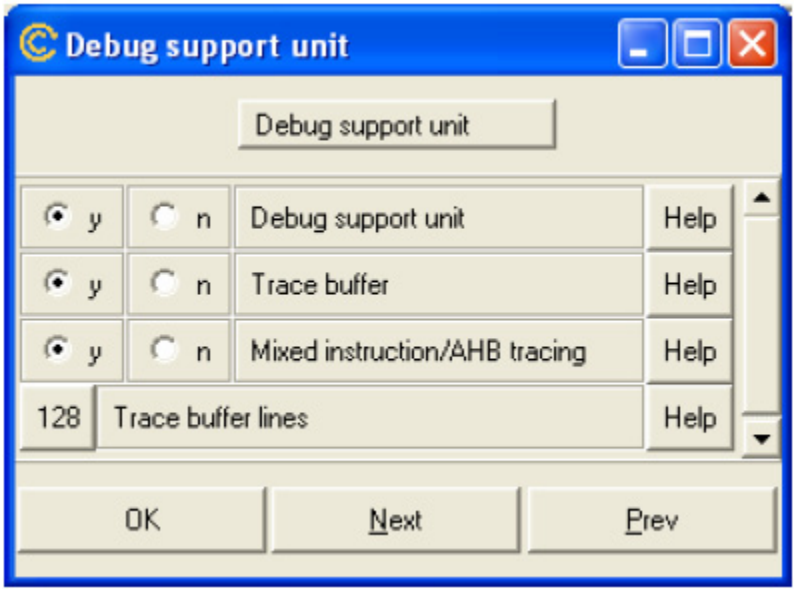

Figure D.10: LEON configuration - Processor Debug Support Unit

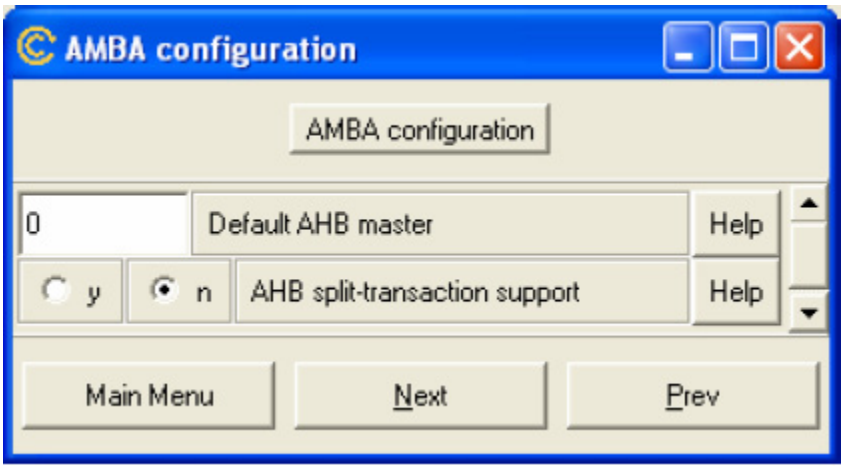

Figure D.11: LEON configuration - AMBA bus

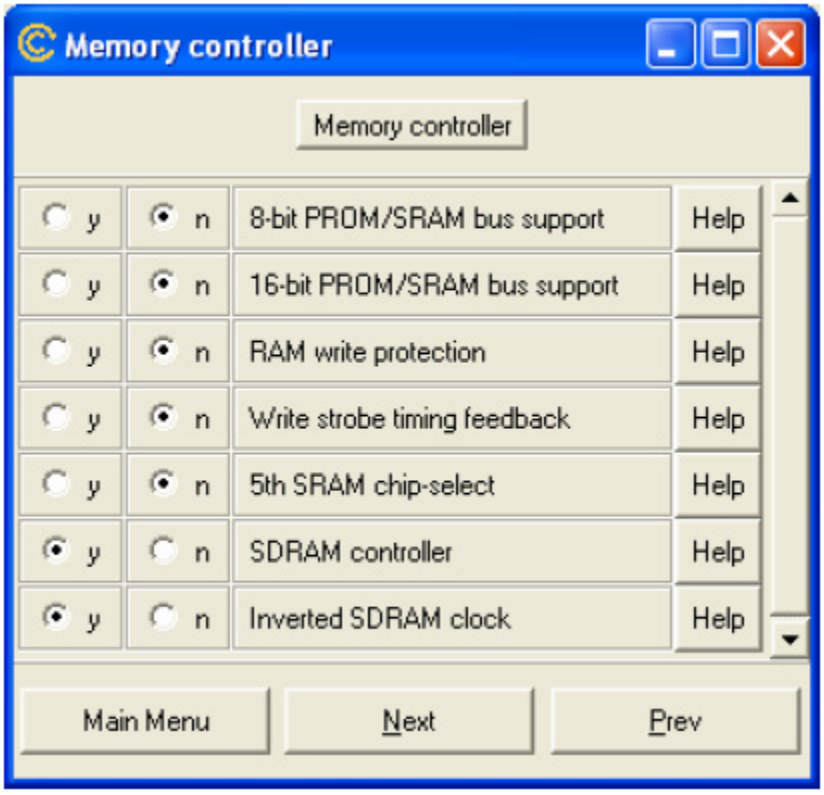

Figure D.12: LEON configuration - Memory Controller

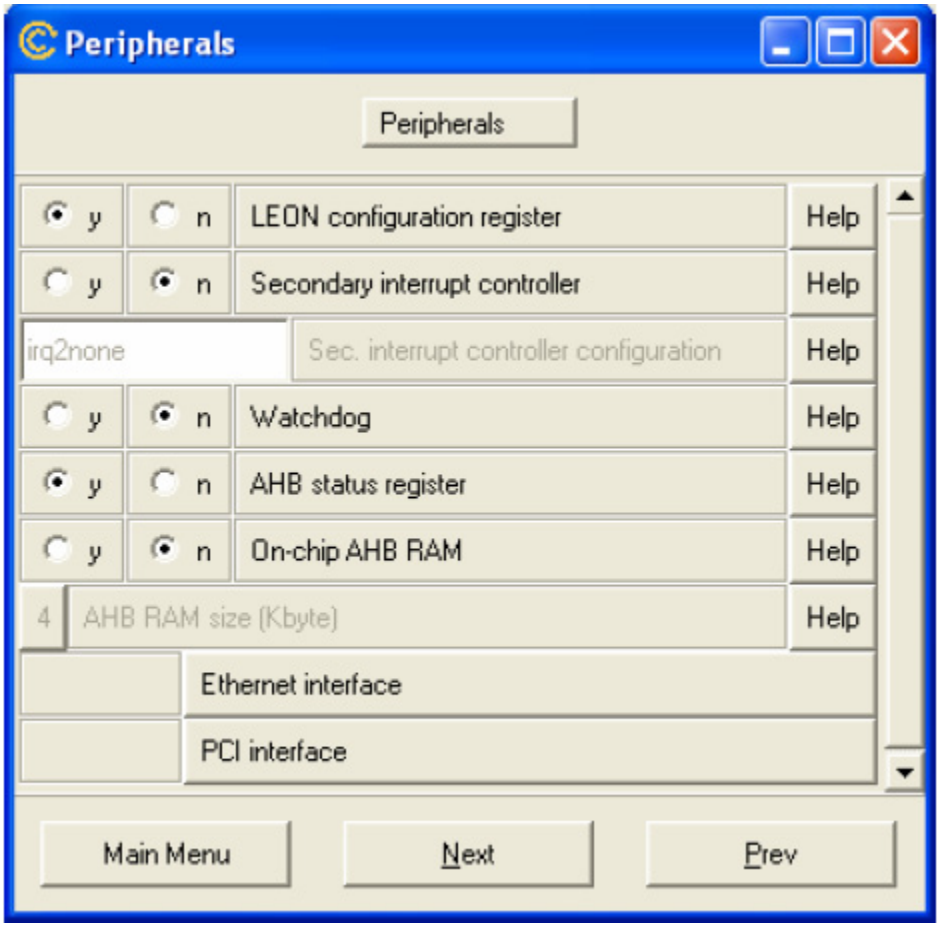

Figure D.13: LEON configuration - Peripherals

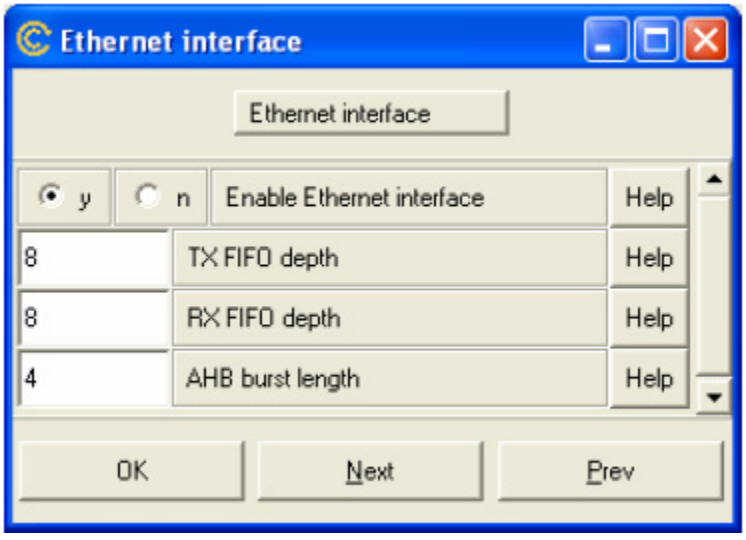

Figure D.14: LEON configuration - Ethernet Interface

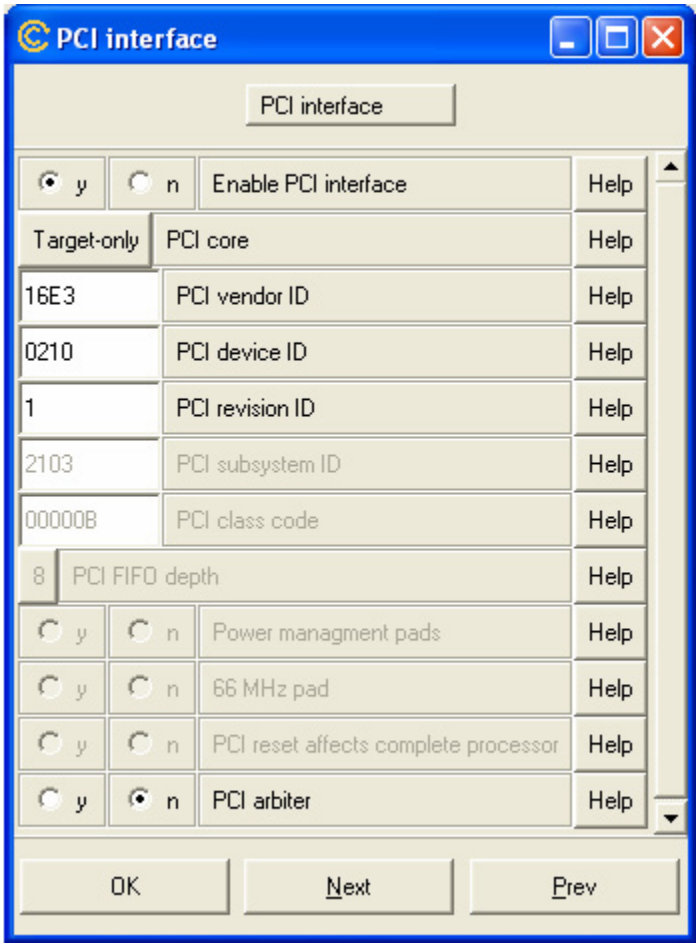

Figure D.15: LEON configuration - PCI

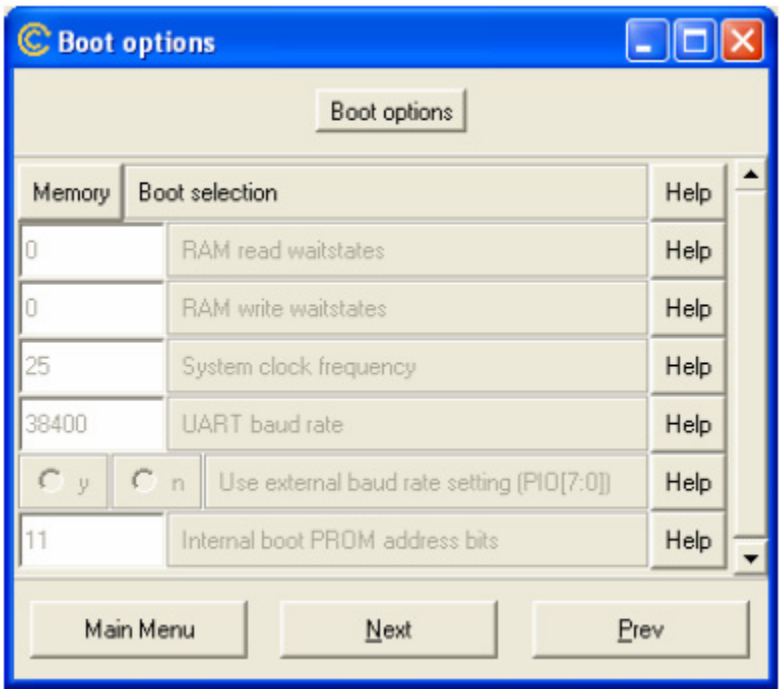

Figure D.16: LEON configuration - Boot options

Figure D.17: LEON configuration - VHDL Debugging

Appendix E

Script to Generate Processor Configurations

E.1 genbitIU.pl

“genbitIU.pl” is a custom perl script that builds LEON Integer Unit (IU) configurations. The script is highly parameterized such that, we can build any one configuration or all of them, in one invocation. gebitCache.pl is similar and builds cache configurations. The script is invoked as follows:

```
perl genbitIU.pl 2>&1|tee -a genbitIU_out_oct14_05.txt
```

The source code for the script itself is listed below.

```
#! /usr/bin/perl

require "genbit.pl";

$subsys = "iu"; $leon_vhdl_path = "../aqua/vhdl/leon/";
$syn_rad_path = "../aqua/syn/rad-xcve2000-64MB/"; $syn_path =
"../aqua/syn/"; $sim_path = "../aqua/sim/"; $apache_path =
"/usr/local/apache/htdocs/"; $bitfile_destination =
$apache_path."documents/"; $outfile = "config2.out";
$makefile_path = "../aqua/";
```

```
@iu_bin_params=("fastjump","icchold","fastdecode","multiplier",
    "multiplier","multiplier","multiplier","multiplier",
    "mulpipe","divider","lddelay");
@iu_bin_base=("true","true","true","m16x16","m16x16","m16x16",
    "m16x16","m16x16","true","radix2","1");
@iu_bin_values=("false","false","false","none","iterative","m32x8",
    "m32x16","m32x32","false","none","2");
@do_bin_params=(0,0,0,0,0,0,0,0,1,0,0);

#since there are many values for nwindows, its easier to handle it
#explicitly $nwindows_lo=2; $nwindows_hi=32;

$do_windows = 0; $do_nomult_nomac = 0; $do_nomult_mac = 0;
$do_noinfermult = 0;

if ( (@ARGV[0] eq "") ) {
    p("\ndefault args: $binFileName(BLASTN) & map.\n".
        " To change, rerun as: perl config.pl options");
    print "   7 = endingSetsize (<=64)\n";
} else {
    $binFileName = @ARGV[0];
    p("args: $binFileName $mapFileName $genBit $whichCache".
        " $setsize $asso $endSetsize $endAsso..\n");
}

$starttime = time(); print "starttime = ".$starttime."\n";

$retValue = shellcall("cp $leon_vhdl_path"."device_min.vhd
$leon_vhdl_path"."device_$subsys".".vhd", 0); #IUbase $retValue =
shellcall("rm $leon_vhdl_path"."device.vhd", 0); #to not end up
reusing erroneously from previous run

print "generating bit files..\n";

$indx = 1; $nparams = 0; #just initializing

    ########### handle all binary's here ################
    $loopindx = 0;
    $nparams = @iu_bin_params; #array length
    print "nparms=$nparams..\n";
    while ($loopindx < $nparams) #since index starts from 0
```

```
{
  print "loopindx=$loopindx..\n";
  if ($do_bin_params[$loopindx] == 1) #=if element at index is true
  {
    $param = $iu_bin_params[$loopindx];
    $base = $iu_bin_base[$loopindx];
    $value = $iu_bin_values[$loopindx];
    print "$indx. changing $param from $base to $value \n";

    $retValue = shellcall("sed 's/".$param." => ".$base."/".
        $param." => ".$value."/g' $leon_vhdl_path"."device_".
        $subsys.".vhd > $leon_vhdl_path"."device.vhd", 0);

    buildbit("".$subsys.$param.$value);
  }

  $loopindx = $loopindx + 1;
  $indx += 1;
}


########### nwindows=2-32 ################
if ($do_windows == 1)
{
  $loopindx = $nwindows_lo;
  $nparams=$nwindows_hi;
  print "nparms=$nparams..\n";
  while ($loopindx <= $nparams) #since index starts from 0
  {
    $param = "nwindows";
    print "$indx. configuring $param..\n";

    $retValue = shellcall("sed 's/".$param." => [0-9]*/".$param.
        " => $loopindx/g' $leon_vhdl_path"."device_".$subsys.
        ".vhd > $leon_vhdl_path"."device.vhd", 0);

    buildbit("".$bitnamePrefix.$param.$loopindx);
    $loopindx = $loopindx + 1;
    $indx += 1;
  }
}
```

```
    #called from below, for exhaustive; not used.

sub config
{
    $bitnamePrefix = shift; #my
    print "bitnamePrefix=$bitnamePrefix..\n";

    ########### mult,div off, mac=f ################
    if ($do_nomult_nomac == 1)
    {
    $param = "multiplier";
    print "$indx. configuring $param"."=false, mac=f..\n";

    $retValue = shellcall("sed 's/multiplier => m16x16/".
        "multiplier => none/g' $leon_vhdl_path"."device_".
        $subsys.".vhd > $leon_vhdl_path"."device_tmp.vhd", 0);
    $retValue = shellcall("sed 's/divider => radix2/".
        "divider => none/g' $leon_vhdl_path".
        "device_tmp.vhd > $leon_vhdl_path"."device.vhd", 0);
    buildbit("".$bitnamePrefix."nomultdiv");
    }

    $indx += 1;
}

config("");

if ($do_noinfermult == 1)
#if ($do_noinfer) #tmp
{
  $retValue = shellcall("sed 's/infer_mult => true/".
    "infer_mult => false/g'".
    " $leon_vhdl_path"."device_".$subsys.".vhd > $leon_vhdl_path".
    "device_tmp.vhd", 0);
  $retValue = shellcall("cp $leon_vhdl_path"."device_tmp.vhd".
    " $leon_vhdl_path"."device_".$subsys.".vhd", 0);
  $retValue = shellcall("cp $leon_vhdl_path"."device_tmp.vhd".
    " $leon_vhdl_path"."device.vhd", 0);
```

```
  buildbit("noinfermult"); #renamed noinfer to noinfermult

  config("noinfermult_");


  $retValue = shellcall("sed 's/infer_mult => false/".
    "infer_mult => true/g'".
    " $leon_vhdl_path"."device_".$subsys.".vhd > $leon_vhdl_path".
    "device_tmp.vhd", 0);
  $retValue = shellcall("cp $leon_vhdl_path".
    "device_tmp.vhd $leon_vhdl_path"."device_".$subsys.".vhd", 0);
}


###### common to all ###### $endtime = time(); print "endtime =
".$endtime."\n";

print "total time = ".($endtime - $starttime)."\n";
```

E.2 genbit.pl

"genbit.pl" is a custom perl script that contains function definitions used by other perl scripts implemented to build LEON configurations.

```
 #!/usr/bin/perl

use warnings;

$genBit = 1;

$runApp = "1"; #i = run the app

$leon_vhdl_path = "../aqua/vhdl/leon/";

$syn_rad_path = "../aqua/syn/rad-xcve2000-64MB/";

$syn_path = "../aqua/syn/";
```

```
$sim_path = "../aqua/sim/";

$apache_path = "/usr/local/apache/htdocs/";

$bitfile_destination = $apache_path."documents/";

$makefile_path = "../aqua/";

$outfile = "config2.out";


#!/usr/bin/perl

use warnings;

$genBit = 1; $runApp = "1"; #i = run the app $leon_vhdl_path =
"../aqua/vhdl/leon/"; $syn_rad_path =
"../aqua/syn/rad-xcve2000-64MB/"; $syn_path = "../aqua/syn/";
$sim_path = "../aqua/sim/"; $apache_path =
"/usr/local/apache/htdocs/"; $bitfile_destination =
$apache_path."documents/"; $makefile_path = "../aqua/"; $outfile =
"config2.out";


######## if a function is not predeclared, use () while calling it

$sysstring = "";

sub shellcall {
    my $command = shift;
    my $expectedRetVal = shift;
    $sysstring = $sysstring.$command."<br>";
    p (".$command);

    my $sysresult = system($command);
    if ($expectedRetVal != null && $expectedRetVal ne "" &&
        $sysresult != $expectedRetVal) {
        print "**Failed with result=".$sysresult." when expected was".
            " $expectedRetVal. exit..\n";
        exit (1); #die();
```

```
    }

    return $sysresult;
}

sub p {
    print("$_[0]\n");
}

sub d {
    #print("$_[0]\n");
}

$get_chipAbsolutes = "false";

sub buildbit
{
    $bitnameIn = shift;
    d ("buildbit.bitnameIn=$bitnameIn..\n");

    $newFileName = "".$bitnameIn;
    $newBitFileNameNoXtn = "lq2_".$newFileName;
    $newBitFileName = $newBitFileNameNoXtn.".bit";
    p ("newBitFileName=$newBitFileName ..\n");

    $retValue = shellcall("cp $leon_vhdl_path"."device.vhd device2_".
        $newFileName.".vhd", 0);

    if ($genBit == -1) {
         shellcall("cp $syn_rad_path"."liquid_sp.bit $syn_rad_path".
            "liquid.bit", 0);
    } elsif ($genBit == 0) {
         shellcall("cp $syn_rad_path".$newBitFileName." ".
            $syn_rad_path."liquid.bit", 0);
    } else {
        shellcall("make");

        #check for errors as xilinx tools don't quit on error and
        #neither is there a cmd line arg to make them quit
        $ret = system("grep error $sim_path"."compileoutput.txt");
        if ($ret == 0) {
```

```
        p ("Modelsim compiler produced errors and so returning..\n");
        return -1;
    }

    $ret = system("grep \"exited with errors\" $syn_path".
        "synoutput.txt");
    if ($ret == 0) {
        p ("Synplicity compiler exited with errors;".
         "and so returning..\n");
        return -1;
    }
}


#check for errors as xilinx tools don't quit on error and neither
#is there a cmd line arg to make them quit
$ret = system("grep OVERMAPPED $syn_rad_path"."mapoutput.txt");
#chk specific errors. 1=no matches
if ($ret == 0) {
    return -1;
}

##### from synplicity mapoutput.txt, extract fpga being used
$luts = mygrep("Total Number 4 input LUTs:", "".
    $syn_rad_path."mapoutput.txt", "grepluts.txt", 11);
p ("luts is $luts");

$bram = mygrep("Number of Block RAMs:", "".$syn_rad_path.
    "mapoutput.txt", "grepbram.txt", 18);
if ($bram == 0 || $bram == "") {
    $bram = mygrep("Number of Block RAMs:", "".$syn_rad_path.
        "mapoutput.txt", "grepbram.txt", 19);
    if ($bram == 0 || $bram == "") {
        $bram = mygrep("Number of Block RAMs:", "".
            $syn_rad_path."mapoutput.txt", "grepbram.txt", 17);
    }
}
p ("bram is $bram");

if ($get_chipAbsolutes) {
  $gates = mygrep_absolutes("Total equivalent gate count for ".
```

```
        "design:, ".$syn_rad_path."mapoutput.txt", "grepgates.txt", 7);
      p ("gates is $gates");
    }

    $retValue = shellcall("cp $syn_rad_path"."liquid.bit $syn_rad_path".
        $newBitFileName, 0);

    #don't use the prev one if the curr build fails..
    $retValue = shellcall("rm $syn_rad_path"."liquid.bit", 0);

    $retValue = shellcall("cp $syn_rad_path".$newBitFileName.
        " $bitfile_destination".".", 0);

    $retValue = shellcall("cp $syn_rad_path".$newBitFileName." .", 0);

    $retValue = shellcall("cp $syn_rad_path"."mapoutput.txt mapout2_".
        $newBitFileNameNoXtn.".txt", 0);

    ########## get current time
    open(DATEPROC, "date|"); #open (the stdout of date) for reading
    @datearray = <DATEPROC>;
    $date = $datearray[0];

    #write bitfilename, $luts, $brams to file conf.data
    $totalClk = 0;
    print "opening file $outfile for writing \n$newBitFileNameNoXtn ".
        "\t$gates \t$luts \t$bram \t$totalClk \t$date.\n";

    open(OUTFILE, ">>$outfile") || die "could not open file".
        " $outfile for writing";

    print OUTFILE "$newBitFileNameNoXtn \t$luts \t$bram \t$date";
    close(OUTFILE);
}

# for chip usage as percent wordToSearchFor, fileToSearchIn,
#fileToWriteTheResultTo, indexToRead

sub mygrep
{
    d("$_[0] $_[1]|tee $_[2]\n");
```

```
    $retValue = shellcall ("grep \"$_[0]\" $_[1]|tee $_[2]");
    #shellcall ("grep \"$_[0]\" $_[1]>$_[2]");
    $searchfile = $_[2];

    d("opening file $searchfile for reading..\n");
    open(INFILE, "$searchfile") || die "could not open file".
    " $searchfile for reading";

    #d("reading file into an array..\n");
    @lines = <INFILE>;
    close(INFILE);
    $maxlines = scalar @lines;
    d("maxlines is $maxlines..\n");

    $line = $lines[0]; #chomp
    d("line is $line\n");

    @words = split(/ /, $line);
    $maxwords = scalar @words;
    d("maxwords is $maxwords..\n");

    $percentWord = @words[$maxwords-1]; #the last element of the array
    d("percentWord is $percentWord..\n");

    @x = split(/%/, $percentWord); #a second way of tokenizing
    $maxx = scalar @x;
    d("maxx is $maxx..\n");

    #for($linenum = 0; $linenum <$maxx ; $linenum++) {
    #    d("x[$linenum]=@x[linenum]\n");
    #}

    $percent = @x[0]; #the last element of the array
    p("$_[2] is $percent percent..");

    return $percent;
}
```

```
# wordToSearchFor, fileToSearchIn, fileToWriteTheResultTo,
#indexToRead

sub mygrep_absolutes
{
    d("$_[0] $_[1]|tee $_[2] $_[3]\n");

    $infile = $_[2];
    $index = $_[3];
    $retValue = shellcall ("grep \"$_[0]\" $_[1]|tee $_[2]");

    #remove the comma's in numbers
    $retValue = shellcall("sed 's/,//g' $infile > $infile.x", 0);
    $retValue = shellcall("mv $infile.x $infile");

    #$retValue = shellcall("sed 's/   //g' $infile > $infile.x", 0);
    #$retValue = shellcall("mv $infile.x $infile");

    d("opening file $infile for reading..\n");
    open(INFILE, "$infile") || die "could not open file".
        " $infile for reading";

    #d("reading file into an array..\n");
    @lines = <INFILE>; #shud be just one line??
    close(INFILE);
    $maxlines = scalar @lines;
    d("maxlines is $maxlines..\n");
    $line = $lines[0]; #chomp
    d("line is $line\n");

    @words = split(/ /, $line);
    $maxwords = scalar @words;
    d("maxwords is $maxwords..\n");
    for($i = 0; $i <$maxwords ; $i++) {
        p("words[$i]=$words[$i]");
    }

    $numb = $words[$index]; #sorry for hardcoding
    d("numb is $numb..\n");

    #$percentWord = @words[$maxwords-1]; #the last element of the array
```

```
    #d("percentWord is $percentWord..\n");
    #@x = split(/%/, $percentWord); #a second way of tokenizing
    #$maxx = scalar @x;
    #d("maxx is $maxx..\n");
    ##for($linenum = 0; $linenum <$maxx ; $linenum++) {
    ##    d("x[$linenum]=@x[linenum]\n");
    ##}
    #$percent = @x[0]; #last element of the array is % #was [5] oct30
    #p("$_[2] is $percent percent..");

    return $numb;
}

#simply pass in the mapoutfile and this sub will parse the lut,
#bram usage as percents

sub grep1
{
    $infile = shift;
    p ("infile = $infile");

    ##### from synplicity mapoutput.txt, extract fpga being used
    $luts = mygrep("Total Number 4 input LUTs:", "$infile",
        "grepluts.txt"); #, 11
    p ("luts is $luts");

    $bram = mygrep("Number of Block RAMs:", "$infile",
        "grepbram.txt", 18);
    if ($bram == 0 || $bram == "") {
        $bram = mygrep("Number of Block RAMs:", "$infile",
            "grepbram.txt", 19);
        if ($bram == 0 || $bram == "") {
            $bram = mygrep("Number of Block RAMs:", "$infile",
                "grepbram.txt", 17);
        }
    }
    p ("bram is $bram");

        #write bitfilename, $slices, $brams percents to file
        print "opening file $outfile for writing \n".
            " $infile \t $luts \t $bram..\n";
```

```
        open(OUTFILE, ">>$outfile") || die "could not open file".
            " $outfile for writing";
        print OUTFILE "$infile \t $luts \t $bram\n";
        close(OUTFILE);
}

sub print_header
{
  print "-----------------------------------------------\n";
  print "Purpose: Automatically optimize LEON micro-arch\n";
  print "          for a given application.             \n";
  print "-----------------------------------------------\n";
  print "--     Shobana Padmanabhan                   --\n";
  print "--     Liquid Architecture Group             --\n";
  print "--     Washington University, St. Louis      --\n";
  print "-----------------------------------------------\n";
}

# not used currently

sub do_logic_counts
{
    ##### from synplicity mapoutput.txt, extract fpga being used
    $luts = mygrep("Total Number 4 input LUTs:", "".
        $syn_rad_path."mapoutput.txt", "grepluts.txt", 11);
    p ("luts is $luts");

    $bram = mygrep("Number of Block RAMs:", "".
        $syn_rad_path."mapoutput.txt", "grepbram.txt", 18);
    if ($bram == 0 || $bram == "") {
        $bram = mygrep("Number of Block RAMs:", "".
            $syn_rad_path."mapoutput.txt", "grepbram.txt", 19);
        if ($bram == 0 || $bram == "") {
            $bram = mygrep("Number of Block RAMs:", "".
                $syn_rad_path."mapoutput.txt", "grepbram.txt", 17);
        }
    }
    p ("bram is $bram");

    $gates = mygrep("Total equivalent gate count for design:", "".
```

```
        $syn_rad_path."mapoutput.txt", "grepgates.txt", 7);
    p ("gates is $gates");
}


$perlExpectsLastVarToBeTrue = "true"; #could also be 1
```

Appendix F

Script to Execute Applications on Processor Configurations

F.1 runbit.pl

“runbit.pl” is a custom perl script to execute applications on LEON configurations.

```
#! /usr/bin/perl

$binpath = "/usr/local/apache/htdocs/fpxControl/test/tmp_bin/";
$mappath = "/usr/local/apache/htdocs/fpxControl/test/tmp_map/";

$binFileName = ""; #no bin extension; set below, in a loop
$mapFileName = "";

$filesToRun = "bitfiles.txt"; $nRuns = 3; $srvr = "aqua2";

@binaries=("blast2_mv8","drr2_mv8","frag2_mv8","arith2_mv8");
@do_binaries=(0,1,0,0);

###################### begin ################################
$starttime = time(); print "starttime = ".$starttime."\n";

#################### loop thru the binaries to be run $loopindx =
0; $nparams = @binaries; #array length print
"nparams=$nparams..\n";

while ($loopindx < $nparams) #since index starts from 0 {
```

```
  print "\nloopindx=$loopindx..\n";
  if ($do_binaries[$loopindx]) #=if element at index is true;
  works if its an int
  {
    $binFileName = $binaries[$loopindx];
    $mapFileName = $binFileName.".map";
    d ("running $binFileName..\n");

    d ("opening file filelist.in for reading..\n");
    #open(TEXTSIZE_FILE, "bitfiles.txt") || die "could not open
    file filelist.in for reading";
    open(TEXTSIZE_FILE, $filesToRun) || die "could not open file
    filelist.in for reading";

    @list = <TEXTSIZE_FILE>;
    close(TEXTSIZE_FILE);
    d ("files to run: @list ..\n");
    d ("read line= @list[0]..\n"); #was line[0]

    $listlength = scalar @list;

    $i=0;
    while ($i < $listlength)
    {
        $infile = $list[$i];
        chomp($infile);
        d ("nextfile after chomp=$infile");
    out, we don't need this anymore
        runit($infile);
        ++$i;
    }
  }

  #be sure to increment loop outside any if conditions
  $loopindx = $loopindx + 1;
}

sub runit {
    $newBitFileName = shift;
    d ("bitfile = $newBitFileName");
    p ("----------------------------\n");
```

```
    $retValue = shellcall("perl auto_run.pl $newBitFileName
    $binFileName $nRuns $srvr");
    d ("retVal after run = $retValue");
}

###### common to all ###### $endtime = time(); print "endtime =
".$endtime."\n"; print "total time = ".($endtime -
$starttime)."\n";
```

F.2 auto_run.pl

“runbit.pl” is a custom perl script to execute applications on LEON configurations.

```
#!/usr/bin/perl

# includes Justin’s rewrites

# Includes for CGI/Time Access use CGI; use     strict; use
Time::localtime; use HTTP::Request::Common qw(POST); require
LWP::UserAgent;

# Never Buffer My Output, Punk $|             = 1;

# Hard Coded Paths, Can Change These You Wish (User Directory,
Etc...) # These Also Affect What the ’Help’ Specifies my
$source_path=
"/usr/local/apache/htdocs/fpxControl/test/benchmarks/"; my
$server_path= "/usr/local/apache/htdocs/fpxControl/test/runs/";

# Check to See if Valid Parameters were Passed, If Not Then Exit
if ($#ARGV < 3) {
  print "Syntax : auto_run.pl [bit_file] [src_file] [runs] [srvr]\n";
  print "*bit_file : Bit File Located on [srvr] in ".
    "/usr/local/apache/htdocs/documents/\n";
  print "*src_file : Src Files (bin/map) on [srvr] in ".
    "/usr/local/apache/htdocs/fpxControl/test/benchmarks/\n";
```

```
  print "*runs     : Total Number of Simulations to Run [Integer]\n";
  print "*srvr     : Server on Which to Run the Simulation ".
    "[aqua, aqua2, etc...]\n\n";
  print "Note  : Results are Saved to [src_file]_results.txt\n";
  print "        Remember to Reserve the Hardware Prior to
    Running to Ensure Safe Operation\n";

  exit;
}

# Necessary Variables and Constants my $query      = new CGI;
# Used to Access Query Information my $user_agent =
LWP::UserAgent->new;     # Used to Generate 'POST' Requests my
$request;                  # Misc. Variable for CGI Requests my
$response;                 # " my $content;                 # "
my $target;               # " my $results;                # "

my $start_time = time();          # Used for Time-Tracking Tasks
my $end_time;                # "

my $start_str  = 'date';          # Used to Time-Tracking Tasks as
Strings my $end_str;                 # "

my $counter    = 0;           # Used for Tracking Runs

my $bit_file   = $ARGV[0];        # Argument 0 is Always Bit/Map
File to Load my $source_file= $ARGV[1];        # Argument 1 is
Always Program to Run my $total_runs = $ARGV[2];        # Argument
2 is Always Total Runs my $srvr       = $ARGV[3];        #
Argument 3 is Always Target Server

my $start_addr;               # Used for Address Tracking of
Target Program my $load_addr  = "40000000";          # " my
$read_addr  = "40000004";          # initialized here but reset
below before actual use my $end_addr;                # "

my $output_file;              # Per-Run Result Files ('wgetted'
From Target Server) my @run_results;              # Clock Counts
for Each Run

my @file_input;               # Used for Reading Input for Files
```

```
on Target Server

my $program    = $source_file . '.bin';   # Input Binary File in
Raw Format my $map_file   = $source_file . '.map';   # Mapfile
Associate with Input Binary File

my $random = int( rand(100)) + 10000;         # a random number
between 100 and 1100

my $output_file2 = "mem".$random.".txt";  # Per-Run Mem Result
Files ('wgetted' From Target Server) my @mem_results;
# Mem Result for Each Run my @file_input2;              # Used for
Reading Input for Files on Target Server

# Print Opening Messages to Console

print "----------------------------\n"; print "Started At:
$start_str"; #print "Started At: $start_time\n"; print "Bit File
: $bit_file\n"; print "Program   : $program\n"; print "Total Runs:
$total_runs\n";

# Print Opening Messages to Output File $output_file =
$source_file . "_results.txt"; open (OUTPUT_FILE,
">>$output_file");
  #print OUTPUT_FILE "Started At: $start_str";
  ##print OUTPUT_FILE "Started At: $start_time\n";
  #print OUTPUT_FILE "Bit File  : $bit_file\n";
  #print OUTPUT_FILE "Program   : $program\n";
  #print OUTPUT_FILE "Total Runs: $total_runs\n\n";
  print OUTPUT_FILE "\n$bit_file\t";
  print OUTPUT_FILE "$program\t";
close(OUTPUT_FILE);


# Setup Statistics Module $target    = "http://" . $srvr .
".arl.wustl.edu/fpxControl/test/statselect_mod.cgi"; $request   =
(POST $target,
          [ "arg0" => $source_path . $program, "arg1" => $source_path
          . $map_file,
        "arg2" => $load_addr, "arg3" => $server_path . "text_size_"
        . $source_file]);
```

```
$results   = $user_agent->request($request); $content   =
$results->content;

# Get the File Generated by the Stats Analysis System and Read it
In system("wget -q http://" . $srvr .
".arl.wustl.edu/fpxControl/test/runs/text_size_$source_file");
open(TEXTSIZE_FILE, "text_size_$source_file") || die "Whoops, No
File!\n";
  @file_input = <TEXTSIZE_FILE>;
close(TEXTSIZE_FILE);

# First Line of File Always Contains Addresses We Need @file_input
= split(/ /,$file_input[0]); $start_addr = @file_input[0];
$end_addr   = @file_input[1];

# Print Out Addresses Harvested From statselect CGI Script print
"Start Addr: $start_addr\n"; print "End Addr  : $end_addr\n";

# Run Through the Loop Until Complete while ($counter <
$total_runs) {
  # Run Indicator
  print "Run #     : " . ($counter + 1) . "\n";

  # ID Each Run According to the Value of 'counter'
  $output_file  = "statoutTotal".$counter."_".$source_file.".txt";
  #print("output_file requested=$output_file\n");
  #$output_file2  = "memResult".$counter."_".$source_file.".txt";

  # Remotely Run the Simulation
  $target    = "http://" . $srvr .
    ".arl.wustl.edu/fpxControl/test/control_mod.cgi";
  $request   = (POST $target,
        [ "arg0" => $bit_file, "arg1" => $source_path . $program,
          "arg2" => $read_addr, "arg3" => $load_addr,
          "arg4" => $start_addr,
          "arg5" => $end_addr, "arg6" => $server_path . $output_file,
                  "arg7" => $server_path . $output_file2]);
  #print "issuing request next = $request\n";
  $results   = $user_agent->request($request);
  #print "reading results next = $results\n";
  $content   = $results->content;
```

```
#print "content = $content\n";

# Grab the Results of the Simulation
system("wget -q http://" . $srvr .
  ".arl.wustl.edu/fpxControl/test/runs/$output_file");
system("wget -q http://" . $srvr .
  ".arl.wustl.edu/fpxControl/test/runs/$output_file2");


# Read in Data to Analyze Simulation Results
#print("output_file recd=$output_file\n");
open(STAT_FILE, $output_file);
@file_input = <STAT_FILE>;
close(STAT_FILE);

# We Want the Total Clocks In the Run
#print("file_input[0]=$file_input[0]\n");
$run_results[$counter] = $file_input[0];
print "Total Clks: $run_results[$counter]\n";


my $ret = -1;
# next, read in the result read written to memory,
#by the application just run
if ($source_file eq "blast2_mv8") {
  $ret = verify_hash_leon_coreLoop_32K_HT();
} elsif ($source_file eq "drr2_mv8") {
  $ret = verify_default();
} elsif ($source_file eq "frag2_mv8") {
  $ret = verify_default();
} elsif ($source_file eq "arith2_mv8") {
  $ret = verify_default();
} else {
  print (" *** can't verify $source_file".".bin's output **\n");
}


# record the result for future references
$mem_results[$counter] = $ret;
```

```
  # Increment the Run Counter
  $counter = $counter + 1;
}

# Concatenate Run Results to a Single Output File print
"$total_runs Runs Completed, Writing Results\n"; $output_file =
$source_file . "_results.txt";

open(OUTFILE, ">>$output_file");

  foreach $counter (@run_results)
  {
    print OUTFILE "$counter\t";
  }
  foreach $counter (@mem_results)
  {
    print OUTFILE "$counter\t";
  }



  # Print Timing Information to the File
  $end_time = time();
  $end_str  = ‘date‘;

    print OUTFILE "$end_str";

  #print OUTFILE "\nEnd Time  : $end_str";
  #print OUTFILE "Total Time: " . ($end_time - $start_time)
    #. " Seconds\n\n";
  print "End Time  : $end_str";
  print "Total Time: " . ($end_time - $start_time) . " Seconds\n\n";

close(OUTFILE);

sub verify_hash_leon_coreLoop_32K_HT {
  my $expected = 508;
  my $mem = "40000800";
  my $retVal = parse_mem($mem);
  print "Result from mem: $retVal\n";
  if ($retVal != $expected)
  {
```

```
    print "!!INVALID result from memory. expected=$expected;".
        " recd=$retVal\n";
  }

  return $retVal;
}

sub verify_default {
  my $expected = 1729;
  my $mem = "40000800";
  my $retVal = parse_mem($mem);
  print "Result from mem: $retVal\n\n";
  if ($retVal != $expected)
  {
    print "!!INVALID result from memory. expected=$expected;".
        " recd=$retVal\n";
  }

  return $retVal;
}

sub parse_mem {
    my $infile = "mem7.txt";
    my $readAdr = shift; #40000004;
    print "readAdr=$readAdr\n";

    #shellcall ("sed 's/tr>//g' mem.txt > mem2.txt", 0);
    shellcall ("sed 's/tr>//g' $output_file2 > mem2.txt", 0);
    shellcall ("sed 's/td>//g' mem2.txt > mem3.txt", 0);
    shellcall ("sed 's/<//g' mem3.txt > mem4.txt", 0);
    shellcall ("sed 's/>//g' mem4.txt > mem5.txt", 0);
    shellcall ("sed 's/br//g' mem5.txt > mem6.txt", 0);
    shellcall ("sed 's/$readAdr//g' mem6.txt > mem7.txt", 0);
    shellcall ("rm -f mem2.txt mem3.txt mem4.txt mem5.txt".
        " mem6.txt", 0);

    #print "opening file $infile for reading..\n";
    open(INFILE, "$infile") || die
        "could not open file $infile for reading";

    #d "reading file into an array..\n";
```

```
    my @lines = <INFILE>; #shud be just one line??
    close(INFILE);
    my $maxlines = scalar @lines;
    #print "maxlines is $maxlines..\n";
    my $line = $lines[0]; #chomp
    #print "line is $line\n";

    my @words = split(/\//, $line);
    my $maxwords = scalar @words;
    #print "maxwords is $maxwords..\n";
    #my $i = 0;
    #for($i = 0; $i <$maxwords ; $i++) {
    #    print "words[$i]=$words[$i]\n";
    #}

    my $result = $words[1];
    #print "result = $result..\n";

    #so that we don’t end up reusing from prev runs..
    #shellcall ("rm -f mem.txt mem7.txt", 0);
    shellcall ("rm -f $output_file2 mem7.txt", 0);
    #, 0 doesn’t work as rm fails when no permission/ over NFS
    shellcall ("rm -f $server_path"."$output_file2");

    return $result;
}

# Remove Temporary Files shellcall ("rm -f statout*", 0);
shellcall ("rm -f text_size*", 0);

# Print out a "Semi-Informative" Footer print "Total Run Time: " .
($end_time - $start_time) . " Seconds\n";

# End Now! exit;
```

F.3 config_mod.pl

“runbit.pl” is a custom perl script to execute applications on LEON configurations.

```
#!/usr/bin/perl

use CGI; require
"/usr/local/apache/htdocs/fpxControl/test/htmleon.pl";

#use Thread; #use Thread qw(async);

sub p {
    print("$_[0]\n");
} sub d {
    #print("$_[0]\n");
}

my $query = new CGI;

# [jthiel] Modified for Calling Via Post Methods my $bitFile
= $query->param(’arg0’); my $cProg     = $query->param(’arg1’); my
$readAddress   = $query->param(’arg2’); my $optionalLoad  =
$query->param(’arg3’); my $startAdr      = $query->param(’arg4’);
my $endAdr    = $query->param(’arg5’); my $statTotalfile =
$query->param(’arg6’); my $memResultfile = $query->param(’arg7’);

# [jthiel] Apache Doth Command It print "Content-type:
text/html\n\n";

#my $bitFile = @ARGV[0]; #my $cProg = @ARGV[1]; #my $readAddress =
@ARGV[2]; #my $optionalLoad = @ARGV[3]; #my $startAdr = @ARGV[4];
#my $endAdr = @ARGV[5]; #$statTotalfile = @ARGV[6];

#my $readSize = 0x01; #0x0A; my $readSize = "01"; #0x0A;

#my $mapFile = $query ->param(’mapFile’); #my $numMethods = $query
->param(’numMethods’); #my $numSignals = $query
->param(’numSignals’);

print ("bit=$bitFile cPrg=$cProg readAdr=$readAddress
```

```
load=$optionalLoad readSz=$readSize strtAdr=$startAdr end=$endAdr
memResfile=$memResultfile");

system "chmod a+wrx $cProg"; #FIX IT

#Now we have to store the k ranges given to us by statselect.cgi

#@range_array_startpt= ();                                    #we
could combine these into one #@range_array_endpt= ();

my $temp; my $arrayCounter = 0;


###################### Begin Simulation

### #Obtain Lock on syncFile ### #print "<font color=red>
Obtaining Lock </font><br>"; #use Fcntl qw(:flock); #my $file =
'syncFile'; #open (S, ">$file"); #flock (S, LOCK_EX) or die "flock
failed";
#print "Lock Obtained <br><hr>";


### #Proceed to Simulation ###

p "/usr/local/apache/cgi-bin/./basic_send 0.0 c $bitFile";
load_bitfile($bitFile);

print "<font color=red>Resetting</font><br>"; reset_leon(1);

print "<font color=red>Checking Status</font><br>";
check_status();

print "<font color=red> Loading Program </font><br>";
load_binary($cProg, $optionalLoad);

#print "<font color=red> Configuring Stats Module </font><br>";
configure_counter(".text", 0, $startAdr, $endAdr);

$srvr="aqua"; #yes, hardcoded for now; was aqua2 $statfile =
"statout.".$srvr;
```

```
w_start_leon($optionalLoad, $statfile); #, 1

### #Start the Listener (The Java Program that Collects and
Processes the UDP Packets) ### p "stat_report\n"; #$statfile =
"statout.txt"; $statfile = "statout.".$srvr;
stat_report($statfile, $statTotalfile); p "stat_report done..\n";


###### read mem to ensure that the program finished ok

#sorry, doesn't work #w_read_mem($readAddress, $readSize,
$memResultfile); #system($memResultfile);

#only explicitly calling java udp works system("java udp 60
40000800 01 | tee $memResultfile");

sub beginsWith {
   $str = shift;
   $sub = shift;
   #print("len=".length($sub)."\n");
   return ( substr($str, 0, length($sub ) ) eq $sub );
}

### #Unlock syncFile (also unlocks on script termination) ###
#flock FILE, LOCK_UN;

######################################## End of File
```

Appendix G

Software Controller

The following is a Java program that sends and receives UDP control packets from hardware.

G.1 UdpServlet.java

```
import java.io.*; import java.util.Enumeration; import
java.util.Hashtable; import javax.servlet.*; import
javax.servlet.http.*;

public class UdpServlet extends HttpServlet {
    public void doGet
        (HttpServletRequest request, HttpServletResponse response)
        throws IOException
    {
        performAction(request, response);
    }

    private void performAction
        (HttpServletRequest request, HttpServletResponse response)
    {
        Debug.emptyLine();
        Debug.verbose("servlet received request");

        Ack ack = null;
        HttpSession session = null;
        long startTime = System.currentTimeMillis();
```

```
//String dirName = "c:\\temp";
String dirName = "/usr/tmp";
Hashtable ht = new Hashtable();
PrintWriter out = null;

try {
    //set response to no caching
    response.setHeader("Pragma", "no-cache");
    response.setHeader("Cache-Control", "no-cache");

    //set DateHeader automatically converts
    //seconds to the right date format
    response.setDateHeader("Expires", 0);

    //Get the HTTP session and the sessionContext
    //object on that session.
    session = request.getSession(true);

    response.setContentType("text/plain");

    out = response.getWriter();
    File file = null;

    try {
      // Use an advanced form of the constructor
      //that specifies a character
      // encoding of the request (not of the file contents)
      //and a file rename policy.
      MultipartRequest multi = new MultipartRequest(
        request, dirName, 10*1024*1024,"ISO-8859-1",
        new DefaultFileRenamePolicy());

      out.println("PARAMS:");
      Enumeration params = multi.getParameterNames();
      while (params.hasMoreElements()) {
        String name = (String)params.nextElement();
        String value = multi.getParameter(name);
        if (value != null) {
            ht.put(name, value);
        }
        out.println(name + "=" + value);
```

```
    }
    out.println();

    out.println("FILES:");

    Enumeration files = multi.getFileNames();

    //this handles only one file currently
    while (files.hasMoreElements()) {
      String name = (String)files.nextElement();

      String filename = multi.getFilesystemName(name);
      if (filename != null) {
          ht.put("file_name", filename);
      }

      String originalFilename = multi.getOriginalFileName(name);
      String type = multi.getContentType(name);

      file = multi.getFile(name);
      if (file != null) {
          ht.put("file", file);
      }
      //Debug.print("file: " + file);

      out.println("name: " + name);
      out.println("filename: " + filename);
      out.println("originalFilename: " + originalFilename);
      out.println("type: " + type);
      if (file != null) {
        out.println("f.toString(): " + file.toString());
        out.println("f.getName(): " + file.getName());
        out.println("f.exists(): " + file.exists());
        out.println("f.length(): " + file.length());
      }
      out.println();
    }
  }
  catch (IOException lEx) {
    lEx.printStackTrace();
      this.getServletContext().log(lEx,
```

```
            "error reading or saving file");
        }

        ack = invokeAction(ht);

        if (ack.specialMsg != null &&
            !ack.specialMsg.equalsIgnoreCase("null") ) {
            out.println(ack.specialMsg);
        }
        out.println(processAck(ack.ackAscii));
        out.println("Response from hardware in ASCII:");
        out.println(ack.ackAscii);
        out.println("Response from hardware in HEX:");
        out.println(ack.ackHex);
    } catch (Exception e) {
        //e.printStackTrace();
        if (out != null) out.println(e.getMessage());
    } finally {
        long stopTime = System.currentTimeMillis();
    }
}

protected Ack invokeAction(Hashtable ht) throws Exception
{
    Ack ack = null;
    String payload = null;
    File file = null;
    String className = this.getClass().getName();
    String methodName = "invokeAction()";
    String prefix = className + "." + methodName + ".";

    String RESET        = "54";
    String CHECK_STATUS = "44";
    String WRITE        = "64"; // = load pgm
    String START_PGM    = "50"; // = start leon
    String READ         = "60";
    String STAT         = "40";

    if (ht == null || ht.size() <= 0) {
        Debug.print("servlet.invokeAction(): empty param values
            - shouldn't have happened");
```

```
    throw new RuntimeException("Invalid parameters.
        Please retry your request");
}

//IP header
String destIP     = (String)ht.get("dest_ip");
int    destPort   = new Integer(
        (String)ht.get("dest_port") ).intValue();
String srcIP      = (String)ht.get("src_ip");
int    srcPort    = new Integer(
        (String)ht.get("src_port") ).intValue();

//payload
String opcode     = (String)ht.get("opcode");
String memAddr    = (String)ht.get("mem_addr");
String readLength = (String)ht.get("read_length");
String program    = (String)ht.get("program");
Object fileObj    = ht.get("file");
if(fileObj != null) file = (File)fileObj;
String endMemAddr   = null;

Debug.verbose(prefix + "params recd: destIP= " + destIP +
    " destPort=" + destPort + " srcIP=" + srcIP + " srcPort=" +
    srcPort + " opcode= " + opcode + " memAdr=" + memAddr +
    " leng=" + readLength + " pgm=" + program + " file=" + file);

payload = opcode;

if ( ! StringHelper.isEmpty(opcode) )
{
    if (opcode.equals(READ) ) {
        if ( StringHelper.isEmpty(readLength) ) {
            throw new RuntimeException(prefix +
                "read length can't be empty for read");
        }
        payload += readLength;

        payload += "0000"; //don't cares; needn't be done here

        if ( StringHelper.isEmpty(memAddr) ) {
            throw new RuntimeException(prefix +
```

```
                "memory addr can’t be empty for read");
            payload += memAddr;
        } else if (opcode.equals(WRITE) ) {
            payload += "000001"; //seq #

            if ( StringHelper.isEmpty(memAddr) ){
                throw new RuntimeException(prefix +
                    "memory addr can’t be empty for write");
            payload += memAddr;

            // for backward compatibility
            if ( !StringHelper.isEmpty(program) ) {
                payload += program;
            }
        } else if (opcode.equals(START_PGM) ) {
            payload += "000000"; //seq number

            if ( StringHelper.isEmpty(memAddr) ) {
                throw new RuntimeException(prefix +
                    "memory addr can’t be empty for start leon");
            payload += memAddr;
        } else if (opcode.equals(STAT) ) {
            payload += "000000";
            endMemAddr   = (String)ht.get("end_mem_addr");

            if ( StringHelper.isEmpty(memAddr) ) {
                throw new RuntimeException(prefix +
                "starting memory addr can’t be empty for statistics");
            if ( StringHelper.isEmpty(endMemAddr) ) {
                throw new RuntimeException(prefix +
                "ending memory addr can’t be empty for statistics");

payload += memAddr + endMemAddr;
}
    Debug.all(prefix + "payload at the end of read switch-case: "
    + payload);
}

try {
    UdpClient sender = new UdpClient();
    ack = sender.sendPacket(
```

```
            srcIP, srcPort, destIP, destPort, payload, file, memAddr);
        if (opcode.equals(READ) ) {
    if (ack != null) {
        ack.specialMsg = "Data read from memory appears under"+
        " the section \"Response from "+
        "hardware\" \n-------------------------------------";
       }
    }
    } catch (Exception e) {
        //e.printStackTrace();
        throw e;
    }

    return ack;
}

private String processAck(String ack)
{
String contactInfo = "contact Liquid Arch FPX group at 935-4658";
String result = "";
if (ack.startsWith("AK54") ) {
    result = "Request to reset Leon was received"+
    " by hardware successfully";
} else if (ack.startsWith("DONE") ) {
    result = "Leon was started successfully";
} else if (ack.startsWith("AK64") ) {
    result = "Program was loaded successfully"; //414b3634
} else if (ack.startsWith("WR04") ) {
    result = "Program was loaded successfully";
} else if (ack.startsWith("AK50") || ack.startsWith("DATA")) {
    result = "Program was started successfully"; //414b3530
} else if (ack.startsWith("RD04") ) {
    result = "Program was started successfully";
} else if (ack.startsWith("RS01") || ack.startsWith("RS04") ) {
    result = "Internal error resetting Leon. Should never happen; "
    + contactInfo + " or try again later";
} else if (ack.startsWith("ST01") || ack.startsWith("ST02")
    || ack.startsWith("ST03") ) {
    result = "Leon Internal error starting the program. " +
    "Check back the status after a few seconds and if " +
    "you still see this message, reset Leon and retry the " +
```

```
    "request and if that doesn’t help either, " + contactInfo;
} else if (ack.startsWith("ST04") ) {
    result = "Leon waiting for program to finish - could be " +
    "because the program is computation intensive and "+
    "so check back the status in a few seconds."+
    " If it looks unreasonably long, " +
    "reset Leon and retry the request and if that doesn’t help either, "
    + contactInfo;
} else if (ack.startsWith("WR01") || ack.startsWith("WR03") ) {
    result = "Leon waiting to write program. Check back the status"+
    " in a few seconds and if you still this message,"+
    " reset Leon and retry the request and if that doesn’t"+
    " help either, " + contactInfo;
} else if (ack.startsWith("WR02") ) {
    result = "Leon still loading program - could be because the " +
    "program is long and so check back the status in "+
    "a few seconds. If it looks unreasonably long, reset "+
    "Leon and retry the request " +
    "and if that doesn’t help either, " + contactInfo;
} else if (ack.startsWith("RD01") ) {
    result = "Leon waiting to read memory. "+
    "Check back the status in a "+
    "few seconds and if you still this message, "+
    "reset Leon and retry "+
    "the request and if that doesn’t help either, " + contactInfo;
} else if (ack.startsWith("RD02") || ack.startsWith("RD03") ) {
    result = "Leon still reading data from memory - "+
    "could be because "+
    "a lot of data was requested and so check back "+
    "the status in a "+
    "few seconds. If it looks unreasonably long, "+
    "reset Leon and retry "+
    "the request and if that doesn’t help either, " + contactInfo;
} else if (ack.startsWith("RD03") ) {
    result = "Leon waiting to send the data read - "+
    "could be because a "+
    "lot of data was requested and so check back the "+
    "status in a "+
    "few seconds. If it looks unreasonably long, "+
    "reset Leon and retry "+
    "the request and if that doesn’t help either, "
```

```
        + contactInfo;
    } else if (ack.startsWith("ERRq") ) {
        result = "Write aborted; " + contactInfo;
    } else if (ack.startsWith("DOWN") ) {
        result = "LEON crashed; retry the request or if the problem"+
        " persists, " + contactInfo;
    }

    if ( !StringHelper.isEmpty(result) ) {
        result += "\n--------------------------------------";
    }

    return result;
    }

    /*
    private void displayPage()
    {
    try {
        // The servlet engine is responsible for showing the page
        HttpServletResponse response =
            (HttpServletResponse) actionRequest.getHttpResponse();
        HttpServletRequest request = actionRequest.getHttpRequest();
        RequestDispatcher rd = actionRequest.
            getActionResources().getServletContext().
            getRequestDispatcher(aJspName);
        rd.forward(request, response);
    }
    catch (Exception e) {
        throw new ActionException("DisplayAction.displayPage()"+
        " - Unable to display java server page, page name: " +
        aJspName, e);
    }
    }
    */
}
```

G.2 UdpClient.java

```
import java.io.*; import java.net.*; import
java.net.DatagramPacket; import java.net.DatagramSocket; import
java.lang.System; import java.lang.Integer;


public class UdpClient {
    public static final int MAX_PAYLOAD_SIZE = 1000;
    public static final int MAX_SOCKET_RECEIVE_TIME = 10000;//millisec
    private String className = this.getClass().getName();
    protected static BufferedReader in = null;
    private DatagramSocket socket = null;


    public UdpClient()
    {

    }

    public Ack sendPacket
        (String srcIP, int srcPort, String destIP, int destPort,
        String hdr, File file, String memAddr) throws Exception
    {
        String methodName = "sendPkt()";
        String prefix = className + "." + methodName + ".";

        Ack ack = null;
        InetAddress address = null;

        Debug.verbose(prefix + "recd hdr="+hdr+" hdr.leng="+
            hdr.length()+" file="+file+" srcIP="+srcIP+" srcPort="+
            srcPort+" destIP="+destIP+" destPort="+
            destPort+" memAdr="+memAddr);

        try {
            if(socket == null) {
                Debug.print(prefix + "getting a new socket");
                socket = new DatagramSocket(srcPort);
            }
```

```
} catch(BindException e) {
    Debug.print(prefix + e.toString());
    e.printStackTrace();
    //Debug.print(prefix  +" retrying");
}

// send request
address = InetAddress.getByName(destIP);
Debug.all(prefix + "addr: " + address);

try {
    int hdrLeng = hdr.length();
    if (hdrLeng == 8) {
        hdr += "00000000";
    }
    Debug.all(prefix + "hdrLeng=" + hdrLeng);

    int tmpFileLeng = 0;
    if(file != null) {
        tmpFileLeng = (int)(file.length());
        if (tmpFileLeng < 0) tmpFileLeng = 0;
    }
    Debug.verbose(prefix + "file.leng: " + tmpFileLeng);

    byte[] bytes = new byte[hdrLeng/2 + tmpFileLeng];
    Debug.all(prefix + "bytesArr.leng: " + bytes.length);

    byte a = 0;
    byte b = 0;
    int j = 0;

    for (int byteIndex=0; byteIndex < hdrLeng &&
        byteIndex < hdrLeng-1 &&
        j < hdrLeng; byteIndex++, j++) {
        a = (byte)(new Integer(
            Integer.parseInt("" + hdr.charAt(byteIndex),16)).
            byteValue() << 4);
        b = new Integer(
            Integer.parseInt("" + hdr.charAt(byteIndex+1),16)).
            byteValue();
        bytes[j]= (byte)(a ^ b);
```

```
    //Debug.all("a: " + a + " b: " + b +
    //" byte["+byteIndex+"]: " + bytes[byteIndex]);
    //Debug.all("bytes["+j+"]: " + bytes[j]);
    byteIndex++;
}

Debug.all(prefix + "j=" + j + " hdrLeng=" + hdrLeng);
byte[] hdrBytes = new byte[hdrLeng/2]; //feb 27
System.arraycopy(bytes, 0, hdrBytes, 0, hdrLeng/2 );
Debug.all(prefix + "hdrBytes.leng=" + hdrBytes.length);
Debug.all(hdrBytes, 16, false);

int fileStartIndex = j;

byte[] fileBytes = new byte[tmpFileLeng]; //feb 26
if (file != null) {
    FileInputStream fis = new FileInputStream(file);
    int i = fis.read();
    //fis.read(fileBytes); //try this - feb 26
    while (i >= 0) {
        bytes[j] = (byte)i;
        Debug.all("bytes["+j+"]: " + bytes[j]);
        j++;
        i = fis.read();
    }
    Debug.all(prefix + "bytes.Leng=" +
        bytes.length + " just created:");
    Debug.all(bytes, 10, false);

    Double aDouble = new Double(
        Math.ceil( (bytes.length-8)/4.0 ) );
    bytes[1] = aDouble.byteValue();

    System.arraycopy(bytes, fileStartIndex,
        fileBytes, 0, tmpFileLeng ); //feb 26
    Debug.all(prefix + "fileBytes.leng="
    + fileBytes.length + " just created:");
    Debug.all(fileBytes, 10, false);
}

Debug.verbose(prefix + "calling sliceNdiceNsend()");
```

```
            ack = sliceNdiceNsend(socket, address, destPort,
                hdrBytes, fileBytes, memAddr);
        } catch(InterruptedIOException e) {
            throw e;
        } catch(Exception e) {
            Debug.print(prefix + e.toString() );
            e.printStackTrace();
        }

        socket.close();
        return ack;
    }

    private Ack sliceNdiceNsend(DatagramSocket socket,
        InetAddress address, int destPort,
        byte hdrBytes[], byte fileBytes[], String memAddr)
        throws Exception
    {
        String methodName = "sliceNdiceNsend()";
        String prefix = className + "." + methodName + ".";

        Ack ack = null;
        StringBuffer ackBuf = new StringBuffer();
        DatagramPacket packet = null;
        byte[] pktBytes;
        int seqNum = 0;

        Debug.all(prefix + "hdrBYTES.leng=" + hdrBytes.length
        + " " + new String(hdrBytes) );
        Debug.all(hdrBytes, 10, false);

        //if there is no file, just send the hdr
        //handle this also in the loop
        if(fileBytes.length <= 0)
        {
            packet = new DatagramPacket(hdrBytes,
                hdrBytes.length, address, destPort);

            Debug.print("\nsending the request (in HEX): ");
            Debug.all(hdrBytes, 10, false);
```

```
    socket.send(packet);

    Debug.verbose(prefix + "calling receiveAck()");
    return receiveAck(socket);
}

int start = 0, end = -1, pktByteCount = 0;
boolean firstTime = true;
Double dLeng = null;
String tmpSeqNumStr = null;
int pktSize = 0;

while(pktByteCount+1 < (fileBytes.length))
{
    //calcualte the range of file bytes (max=1000)
    //to copy in this iteration; remember,
    //we'll add the hdr bytes before the file bytes
    if(fileBytes.length - (end+1) <= MAX_PAYLOAD_SIZE)
    {
        end += (fileBytes.length)-start-1;
        pktSize = (fileBytes.length)-start;
    } else {
        end += MAX_PAYLOAD_SIZE;
        pktSize = MAX_PAYLOAD_SIZE;
    }

    Debug.all(prefix + "start=" + start + " end="
    + end + " file.leng=" + fileBytes.length
    + " pktSize=" + pktSize);

    /////////// 1. prepare the hdr  bytes
    dLeng = new Double(Math.ceil((pktSize)/4.0));

    tmpSeqNumStr = "";
    ++seqNum;
    if(seqNum < 0x100) {
        tmpSeqNumStr = StringHelper.toHex(0);
    }

    int nextMemAddr = 0;
    if (firstTime) {
```

```
    nextMemAddr = Integer.parseInt(memAddr,16);
} else {
    nextMemAddr = Integer.parseInt(memAddr,16)
    +start;
}
Debug.all(prefix + "nextMemAdr=" + nextMemAddr
+ " inHex=" + StringHelper.toHex(nextMemAddr));

String hdr = "64" + StringHelper.toHex(dLeng.intValue())
+ tmpSeqNumStr + StringHelper.toHex(seqNum)
+ StringHelper.toHex(nextMemAddr);
Debug.all(prefix + "hdr b4 bytezing: " + hdr);

byte a = 0;
byte b = 0;
int j = 0;
Debug.all(prefix + "creating hdr bytes for hdr "
+ hdr + " of leng " + hdr.length() );
for (int i=0; i < hdr.length() && i < hdr.length()-1
    && j < hdr.length(); i++, j++) {
    Debug.all(prefix + "hdr.charAt("+i+"): "
    + hdr.charAt(i) + " next: " + hdr.charAt(i+1));
    a = (byte)(new Integer(
        Integer.parseInt(""+
        hdr.charAt(i),16)).byteValue() << 4);
    b = new Integer(
        Integer.parseInt(""+
        hdr.charAt(i+1), 16)).byteValue();
    hdrBytes[j]= (byte)(a^b);
    //if (seqNum < 12) {
        Debug.all("hdrBytes["+j+"]: " +
        StringHelper.toHex(hdrBytes[j]) );
    //}
    i++;
}
Debug.verbose(prefix + "modified hdrBytes:");
Debug.all(hdrBytes, 10, false);

////////////// 2. prepare the file bytes
Debug.all(prefix + "fileBytes:");
Debug.all(fileBytes, 10, false);
```

```
            // 3. finally, prepare the pkt bytes = hdr + file
            pktBytes = new byte[pktSize + hdrBytes.length];

            System.arraycopy(hdrBytes,0,pktBytes,0,hdrBytes.length);
            Debug.all(fileBytes, 10, false);
            Debug.verbose(prefix + "hdrBytes.leng=" + hdrBytes.length
            + " pktBytes.leng=" + pktBytes.length + " fileBytes.leng="
            + fileBytes.length + " start=" + start + " end=" + end);
            Debug.all(hdrBytes, 10, false);

            System.arraycopy(fileBytes, start, pktBytes,
                hdrBytes.length, pktSize );
            packet = new DatagramPacket(pktBytes, pktBytes.length,
                address, destPort);

            Debug.print("\nsending the request (in HEX): ");
            Debug.all(pktBytes, 10, false);
            socket.send(packet);

            Debug.print("\nwaiting for an ACK");
            ack = receiveAck(socket);

            pktByteCount += pktSize; //sp
            Debug.verbose(prefix + "pktByteCount=" + pktByteCount +
            " fileBytes.leng=" + fileBytes.length);

            start = end + 1;
            firstTime = false;
        }

        Debug.verbose("end of sliceAndDlice()");
        return ack;
    }

    protected Ack receiveAck(DatagramSocket socket)
    throws InterruptedIOException
    {
        String methodName = "receiveAck()";
        String prefix = className + "." + methodName + ".";
```

```
        DatagramPacket packet = null;
        StringBuffer ackBuf = new StringBuffer();
        Ack ack = new Ack();

        try {
            byte[] buf = new byte[256];

            // receive request
            packet = new DatagramPacket(buf, buf.length);
            socket.setSoTimeout(MAX_SOCKET_RECEIVE_TIME);
            socket.receive(packet);

            Debug.print("\n=============");

            byte[] bytes2 = packet.getData();
            ack.ackAscii = new String(bytes2);
            Debug.print("response from hardware in ascii: "
            + ack.ackAscii );
            Debug.print("response from hardware in HEX: ");

            for (int i=0; i < bytes2.length; i++) {
                ackBuf.append( StringHelper.getString(bytes2[i], 16) );
            }
            ack.ackHex = ackBuf.toString();
            Debug.print(ack.ackHex);

            Debug.print("--------");
        } catch (InterruptedIOException e) {
            throw new InterruptedIOException(
            "No response from hardware."+
            " Please retry your request");
        } catch (IOException e) {
            Debug.print(prefix + e.toString());
            e.printStackTrace();
        }

        return ack;
    }

}
```

G.3 UdpServer.java

This java class emulates hardware.

```
public static void main(String[] args) throws IOException {
    String mode = "ack"; //others are echo
    if (args.length > 0) {
        if (args[0].equals("ack") || args[0].equals("echo") ) {
            mode = args[0];
        }
    }
    new UdpServerThread(mode).start();
}
```

G.4 UdpServerThread.java

```
import java.io.*; import java.net.*; import java.util.*;

public class UdpServerThread extends Thread {
    protected DatagramSocket socket = null;
    protected BufferedReader in = null;
    protected String _mode = "ack";

    public UdpServerThread(String mode) throws IOException
    {
        this(_mode, "UdpServerThread");
    }

    public UdpServerThread(String mode, String name) throws IOException
    {
        super(name);
        _mode = mode;
        socket = new DatagramSocket(4446);
        print("server ready");
    }

    public void run()
```

```
{
    int nPktRecd = 0;
    byte[] buf = new byte[2560];
    DatagramPacket pktForSending = null;

    DatagramPacket pktRecd = new DatagramPacket(buf, buf.length);

    byte[] payload;
InetAddress address = null;
    int port = 0;

    while(true)
    {
        try {
            // receive request
            socket.receive(pktRecd);
    nPktRecd++;
            print("server recd: " + nPktRecd);

            payload = pktRecd.getData();

            // send the response to the client at "address" and "port"
            //from where we recd the pkt
            if (_mode.equals("ack")) {
                ackBuf = do_ack();
            } else if (_mode.equals("echo")) {
                ackBuf = do_echo(pktRecd);
            }

            pktForSending = new DatagramPacket(ackBuf,ackBuf.length);

            pktForSending.setAddress(pktRecd.getAddress());
            pktForSending.setPort(pktRecd.getPort());

            socket.send(pktForSending);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } //end of while
}
```

```
    DatagramPacket do_echo(DatagramPacket pktRecd)
    {
        byte[] ackBuf = new byte[4];
        ackBuf = "AK64".getBytes();
        DatagramPacket pktForSending = new
            DatagramPacket(ackBuf, ackBuf.length);
        return pktForSending;
    }

    byte[] do_ack()
    {
        byte[] ackBuf = new byte[4];
        ackBuf = "AK64".getBytes();
        DatagramPacket pktForSending = new
            DatagramPacket(ackBuf, ackBuf.length);
        return pktForSending;
    }

    private static void print(String str)
    {
        System.out.println(str);
    }
}
```

G.5 Ack.java

```
public class Ack {
    public String ackAscii = "";
    public String ackHex;
    public String specialMsg;
}
```

G.6 Debug.java

```
public class Debug {
    public static final int INFO = 1;
```

```
public static final int DEBUG = 2;
public static final int VERBOSE = 3;
public static final int ALL = 4;

static int debug_level = 1;

public static void print(String str)
{
    System.out.println(str);
}

public static void info(String str)
{
    if(debug_level >= INFO) {
        System.out.println(str);
    }
}

public static void debug(String str)
{
    if(debug_level >= DEBUG) {
        System.out.println(str);
    }
}

public static void verbose(String str)
{
    if(debug_level >= VERBOSE) {
        System.out.println(str);
    }
}

public static void all(String str)
{
    if(debug_level >= ALL) {
        System.out.println(str);
    }
}

public static void emptyLine()
{
```

```
    System.out.println();
}

public static void print(
    byte bytes[], int radix, boolean displayAs2digits)
{
    for (int x=0; x < bytes.length; x++) {
        System.out.print(
            StringHelper.getString(bytes[x],
            radix, displayAs2digits) );
    }
    //prevent subsequent prints on the same line
    Debug.emptyLine();
}

public static void oneByOne(
    byte bytes[], int radix, boolean displayAs2digits)
{
    for (int x=0; x < bytes.length; x++) {
        System.out.print( x + "=" +
        StringHelper.getString(bytes[x],
        radix, displayAs2digits) + " ");
    }
}

public static void debug(
    byte bytes[], int radix, boolean displayAs2digits)
{
    if(debug_level >= DEBUG) {
        print(bytes, radix, displayAs2digits);
    }
}
public static void verbose(
    byte bytes[], int radix, boolean displayAs2digits)
{
    if(debug_level >= VERBOSE) {
        print(bytes, radix, displayAs2digits);
    }
}
public static void all(
    byte bytes[], int radix, boolean displayAs2digits)
```

```
    {
        if(debug_level >= ALL) {
            print(bytes, radix, displayAs2digits);
        }
    }
}
```

G.7 StringHelper.java

```
public class StringHelper {
    public static String getString(
        byte aByte, int radix)
    {
        String tmp = null;
        tmp = Integer.toHexString(aByte);

        if (tmp != null && tmp.length() == 1)
        {
            tmp = "0" + tmp;
        }

        if (tmp != null && tmp.length() > 2)
        {
            int len = tmp.length();
            tmp = tmp.substring(len-2);
        }

        return tmp;
    }

    public static String toHex(int tmp)
    {
        String className = StringHelper.class.getName();
        String methodName = "toHext(int)";
        String prefix = className + "." + methodName + ".";

        String zero = Integer.toHexString( Integer.parseInt("0", 16) );
        String twoZeros = zero + zero;
```

```
    if (tmp == 0) {
        return twoZeros;
    }

    String str = Integer.toHexString(tmp);
    //if (str.length() == 1) {
    if (str.length() == 1 || str.length() == 3) //mar 10
    {
        str = zero + str;
    }
    Debug.all(prefix + str);
    return str;
}

public static String getString(
    byte aByte, int radix, boolean fixLeng)
{
    if(fixLeng) {
        throw new RuntimeException("if u want the leng to be fixed,"+
        " call the regular getString(byte, int)");
    }
    String tmp = null;
    tmp = Integer.toHexString(aByte);

    if (tmp != null && tmp.length() > 2)
    {
        int len = tmp.length();
        tmp = tmp.substring(len-2);
    }

    return tmp;
}

/**
 * currrently not being used but toHex(int) was copied from this
 */
/*public static String toHex(String tmp)
{
    String className = StringHelper.class.getName();
    String methodName = "toHext(str)";
```

```
    String prefix = className + "." + methodName + ".";

    String zero = Integer.toHexString( Integer.parseInt("0", 16) );
    String twoZeros = zero + zero;

    if (tmp == null) {
        return twoZeros;
    }

    String str = Integer.toHexString( Integer.parseInt(tmp, 16) );
    if (str.length() == 1) {
        str = zero + str;
    }
    Debug.verbose(prefix + str);
    return str;
}*/

public static boolean isEmpty(String s)
{
    return (s == null || s.length() <= 0);
}

public static boolean isEmpty(String s, int lengthToCheckFor)
{
    return (s == null || s.length() <= lengthToCheckFor);
}

/* NOT TESTTED; from internet
private static String byteToHex(byte b)
{
  // Returns hex String representation of byte b
  char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7','8', '9',
  'a', 'b', 'c', 'd', 'e', 'f'};
  char[] array = { hexDigit[(b >> 4) & 0x0f], hexDigit[b & 0x0f] };
  return new String(array);
}   // end of method byteToHex
*/

/* NOT TESTED; from internet
static public String charToHex(char c)
{
```

```
        // Returns hex String representation of char c
        byte hi = (byte) (c >>> 8);
        byte lo = (byte) (c & 0xff);
        return byteToHex(hi) + byteToHex(lo);
    }
    */
}
```

Appendix H

Tomlab scripts

H.1 main.m

```
Name = 'Cost function weights w1 = w2 = 1';

%BinVars just need to be non-zero, for them to be bin's
BinVars_all = [1    2   3   4   5   6   7
    8   9   10  11  12  13  14  15
    16  17  18  19  20  21  22  23
    24  25  26  27  28  29  30  31
    32  33  34  35  36  37  38  39
    40  41  42  43  44  45  46  47  48
    49  50  51  52];
BinVars_dcache = [1 2   3   4   5   6   7   8];

% No priorities given
VarWeight = [ ];

% the linear objective function is defined in objfun.m

% coeffs of linear constraints
%  1. luts 2. only one dcach-sets 3. only one dcach-setsize
% 16, 17: if lrr, nsets=2; for icache & dcache.
% 18, 19: if lru, nsets=2,3,or4; again, for icache and dcache
A_all = [0  0   0   0   0   -1  -1  0
    -1  0   1   0   0   0   -1  -1
    0   -1  -1  0   0   0   -1  0
    0   0   0   -2  0   0   0   0
```

```
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   -2
0   0   1   0;...
1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   1   1   1   1   1   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1
    1   1   1   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   1   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   1   1   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
```

```
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    0   0   0   0   0   0;...
-1  0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
 0  0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0  0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0  0   0   0   0   0;...
 0  0   0   0   0   0   0   0   0   0   0  -1   0   0   0   0
    0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
 1  1   1   0   0   0   0   0   0   0  -1   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0;...
 0  0   0   0   0   0   0   0   0   0   0   1   1   1   0   0
    0   0   0   0   0  -1   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
        0   0   0   0   0   0;...
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        1   1   1   1   1   0;...
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        0   0   0   0   0   1 ...
    ];
A_dcache = [ 0   0   0   -1  -1  0   -1  -1; 1 1 1 0 0 0 0 0 ; 0 0
0 1 1 1 1 1 ];


% RHS of the linear constraints
% lower bounds are zero because none need to be selected.
% this is b/c base confgn isn't included.
b_L_dcache = [-inf   0 0 ]'; %-500 is hand calculated
b_U_dcache = [61  1 1 ]';

b_L_all = [-inf 0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0
    -1  -1  0   0   0   0]';
b_U_all = [61  1     1     1     1     1     1     1     1     1
    1  1   1     1     1     1     1  0   0   1   1    1   1]';

% BinVars = BinVars_all;
% A = A_all;
% b_L = b_L_all;
% b_U = b_U_all;

BinVars = BinVars_dcache; A = A_dcache; b_L = b_L_dcache; b_U =
b_U_dcache;

% bounds of the one nonlinear constraint defined in confun.m for BRAM
c_L = [0]; c_U = [49];

% decn vars are bin's
x_0 = zeros( length(BinVars), 1 );
%x_0 = [-1 -1 -1 -100 -100 -100 -100 -100]';
x_L = zeros( length(BinVars), 1 ); x_U = ones( length(BinVars), 1
```

```
);

g = []; %gradient vector
H = []; %Hessian matrix
%All elements in Hessian are nonzero. if all 0, spalloc(5,5,0)
HessPattern = [];
% constraint gradient. 0 indicates 0s in constraint Jacobian
ConsPattern = [];
fIP = []; % An upper bound on the IP value wanted; to cut branches.
xIP = []; %  x-values giving the fIP value
dc = []; %constraint Jacobian mN x n
d2c = []; %second part of Lagrangian function
f_opt = []; %opt function value, if known
x_opt = []; %x-values corresponding to f_opt

% Generate the problem structure using the TOMLAB Quick format
Prob = minlpAssign(@objfun, g, 'objfun_H', HessPattern, ...
                   x_L, x_U, Name, x_0, ...
                   BinVars, VarWeight, fIP, xIP, ...
                   A, b_L, b_U, 'confun', dc, d2c, ConsPattern, ...
                   c_L, c_U, ...
                   x_L, x_U, ...
                   f_opt, x_opt);
%Prob.DUNDEE.optPar(20) = 1;
Prob.P = 33;   % Needed in minlpQG_xxx files
%Prob.LargeScale = 1;

Solver = {'minlpBB','oqnlp','glcFast'};

for i=1:1
    Result  = tomRun(Solver{i},Prob,2);
end

%checkDerivs(Prob, Result.x_k)

Result2  = tomRun('filterSQP',Prob,2);
```

H.2 objfun.m

```
function f = objfun(x,Prob)

w_100_1 = [100 1]; w_1_100 = [1 100];

L_all = [0  0   0   0   0   -1  -1  0
    -1  0   1   0   0   0   -1  -1
    0   -1  -1  0   0   0   -1  0
    0   0   0   -2  0   0   0   0
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   -2
    0   0   1   0];
b_all = [5  11  17  -4  -3  5   17  39
    0   5   5   5   11  17  -4  -3
    5   17  39  0   5   5   0   0
    0   0   0   0   0   2   2   2
    2   2   2   2   2   2   2   2
    2   2   2   2   2   7   0   0
    0   0   0   0];
L_all_arith = [0    0   0   0   0   -1  -1
    0   -1  0   1   0   0   0   -1
    -1  0   -1  -1  0   0   0   -1
    0   0   0   0   80  0   0   0   0
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   -2
    0   0   1   0];
b_all_arith = [5    11  17  -4  -3  5   17
    9   0   5   5   5   11  17  -4
    -3  5   17  39  0   5   5   0
    0   0   0   0   80  0   2   2
    2   2   2   2   2   2   2   2
    2   2   2   2   2   2   7   0
    0   0   0   0   0];

L_dcache = [0   0   0   -1  -1  0   -1  -1]; b_dcache = [5   11
17  -4  -3  5   17  39];

L_icache = L_dcache; b_icache = b_dcache;
```

```
r_all_blast = [0    0   0   0.186306783 4.15048E-06
    0   0   0   -4.90511E-06    0   0
    -0.679644632    -0.813927779    -0.96554824 1.028720067
    0.358113296 -0.609421892    -0.912860527    -3.611279413
    -0.199577003    -0.672483166    -0.761805664    0
    -3.416724492    0   5.948311029 0   0   1.1347807
    0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0
    34.04337722 0   -2.269557626    -4.539115252    0];
r_dcache_blast = [-0.679644632  -0.813927779    -0.96554824
1.028720067
    0.358113296 -0.609421892    -0.912860527    -3.611279413];

r_all_drr = [0  0   0   0.047907346 1.61085E-07 0   0
    0   1.0739E-07  0   0   -9.951593511    -11.62155299
    -12.18971477    19.64088775 6.4071957   -4.806727246
    -8.853635082    -12.20561489    -3.088953864    -10.06280944
    -9.986386939    0   -4.445869434    0   15.79990129
    0   0   0.684681324 0   0   0   0   0
    0   0   0   0   0   0   0   0   0
    0   0   0   0   20.54043582 0   -1.36936230
    -2.73872460 0];
r_dcache_drr = [-9.951593511    -11.62155299    -12.18971477
19.64088775
    6.4071957   -4.806727246    -8.853635082    -12.20561489];

r_all_frag = [0 0   0   19.12551172 11.06005803 0   0
    0   1.53895E-06 0   0   -0.879948605    -1.125929685
    -1.459847612    2.555348338 0.851782779 -0.354540957
    -1.299341669    -1.912464874    -0.396092312    -0.836597765
    -0.852205832    0   -3.803944163    0   12.08939242 0
    0   0.108681773 0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0
    0   0   0   3.260453178 0   -0.217363545
    -0.43472709 0];
r_dcache_frag = [-0.879948605   -1.125929685    -1.459847612
2.555348338
    0.851782779 -0.354540957    -1.299341669    -1.912464874];

r_all_arith = [0    0   0   1.36086E-06 0   0   0
    0   -2.84544E-06    0   0   0   0   0   0
```

```
    0   0   0   0   0   0   0   0
    -1.300242768    0   9.089327768 0   70  1.297768471
    0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0
    37.63528567 0   -2.595536943    -5.191073885    0];
r_dcache_arith = [0.00  0.00    0.00    0.00    0.00    0.00
0.00
    0.00];


P=Prob.P;

%%%%%%%%%%%%%% blast
if Prob.P == 1 w = w_100_1; r = r_all_blast; L = L_all; b = b_all;
elseif Prob.P == 2 w = w_1_100; r = r_all_blast; L = L_all; b =
b_all; elseif Prob.P == 3 w = w_100_1; r = r_dcache_blast; L =
L_dcache; b = b_dcache; elseif Prob.P == 4 w = w_1_100; r =
r_dcache_blast; L = L_dcache; b = b_dcache; elseif Prob.P == 5 w =
w_100_1; r = r_icache_blast; L = L_icache; b = b_icache; elseif
Prob.P == 6 w = w_1_100; r = r_icache_blast; L = L_icache; b =
b_icache;

%%%%%%%%%%%%%% drr
elseif Prob.P == 11 w = w_100_1; r = r_all_drr; L = L_all; b =
b_all; elseif Prob.P == 12 w = w_1_100; r = r_all_drr; L = L_all;
b = b_all; elseif Prob.P == 13 w = w_100_1; r = r_dcache_drr; L =
L_dcache; b = b_dcache; elseif Prob.P == 14 w = w_1_100; r =
r_dcache_drr; L = L_dcache; b = b_dcache; elseif Prob.P == 15 w =
w_100_1; r = r_icache_drr; L = L_icache; b = b_icache; elseif
Prob.P == 16 w = w_1_100; r = r_icache_drr; L = L_icache; b =
b_icache;

%%%%%%%%%%%%%% frag
elseif Prob.P == 21 w = w_100_1; r = r_all_frag; L = L_all; b =
b_all; elseif Prob.P == 22 w = w_1_100; r = r_all_frag; L = L_all;
b = b_all; elseif Prob.P == 23 w = w_100_1; r = r_dcache_frag; L =
L_dcache; b = b_dcache; elseif Prob.P == 24 w = w_1_100; r =
r_dcache_frag; L = L_dcache; b = b_dcache; elseif Prob.P == 25 w =
w_1_100; r = r_icache_frag; L = L_icache; b = b_icache; elseif
Prob.P == 26 w = w_1_100; r = r_icache_frag; L = L_icache; b =
b_icache;
```

```
%%%%%%%%%%%%%% arith
elseif Prob.P == 31 w = w_100_1; r = r_all_arith; L = L_all_arith;
b = b_all_arith; elseif Prob.P == 32 w = w_1_100; r = r_all_arith;
L = L_all_arith; b = b_all_arith; elseif Prob.P == 33 w = w_100_1;
r = r_dcache_arith; L = L_dcache; b = b_dcache; elseif Prob.P ==
34 w = w_1_100; r = r_dcache_arith; L = L_dcache; b = b_dcache;
elseif Prob.P == 35 w = w_1_100; r = r_icache_arith; L = L_icache;
b = b_icache; elseif Prob.P == 36 w = w_1_100; r = r_icache_arith;
L = L_icache; b = b_icache; end


if Prob.P==3 || Prob.P==5 || Prob.P==13 || Prob.P==15 ||
Prob.P==23 ||
    Prob.P==25 || Prob.P==33 || Prob.P==35
    cost = r;
    f = cost*x;
else
    cost = w(1)*r+w(2)*L+w(2)*b;
    f = cost*x;
end
```

H.3 confun.m

```
function [c, ceq] = confun(x,Prob)

% Nonlinear inequality constraints


% BRAM usage happen to be the same for i and d cache currently
%note matlab expects <= 0 BUT tomlab expects <= 49


if Prob.P==1 || Prob.P==2 || Prob.P==11 || Prob.P==12 ||
Prob.P==21 ||
    Prob.P==22 || Prob.P==31 || Prob.P==32
c =
[(1+1*x(1)+2*x(2)+3*x(3))*(-4*x(4)-3*x(5)+5*x(6)+17*x(7)+39*x(8))+
```

```
    2+(5*x(1)+11*x(2)+17*x(3))+...
    (1+1*x(12)+2*x(13)+3*x(14))*(-4*x(15)-3*x(16)+5*x(17)+17*x(18)+
        39*x(19))+2+(5*x(12)+11*x(13)+17*x(14))+...
    0*x(9)+5*x(10)+5*x(11)+0*x(20)+5*x(21)+5*x(22)+0*x(23)+0*x(24)+
        0*x(25)+...
    0*x(26)+0*x(27)+0*x(28)+0*x(29)+2*x(30)+2*x(31)+...
    2*x(32)+2*x(33)+2*x(34)+2*x(35)+2*x(36)+2*x(37)+2*x(38)+2*x(39)+
        2*x(40)+...
    2*x(41)+2*x(42)+2*x(43)+2*x(44)+2*x(45)+7*x(46)];
else c =
[(1+1*x(1)+2*x(2)+3*x(3))*(-4*x(4)-3*x(5)+5*x(6)+17*x(7)+39*x(8))+
    2+(5*x(1)+11*x(2)+17*x(3))]; %-49
end


ceq = [];
```

H.4 dc.m

Useful for debugging.

```
function dc = minlpQG_dc(x, Prob)

dc = [ ... -8*x(4)-6*x(5)+10*x(6)+34*x(7)+78*x(8)+5
-12*x(4)-9*x(5)+15*x(6)+51*x(7)+117*x(8)+11 ...
-16*x(4)-12*x(5)+20*x(6)+68*x(7)+156*x(8)+17
-4-8*x(1)-12*x(2)-16*x(3) ... -3-6*x(1)-9*x(2)-12*x(3)
5+10*x(1)+15*x(2)+20*x(3) ... 17+34*x(1)+51*x(2)+68*x(3)
39+78*x(1)+117*x(2)+156*x(3)];
```

H.5 d2c.m

Useful for debugging.

```
%           lam' * d2c(x)
```

```
%
% in
%
%   L(x,lam) =   f(x) - lam' * c(x)
% d2L(x,lam) = d2f(x) - lam' * d2c(x) = H(x) - lam' * d2c(x)
%
% function d2c=minlpQG_d2c(x, lam, Prob)

function d2c=nlcon_d2c(x, lam, Prob) d2c =
spalloc(length(x),length(x),1); d2c(1,4) = lam(1)*(-20); d2c(1,5)
= lam(1)*(-15); d2c(1,6) = lam(1)*25; d2c(1,7) = lam(1)*85;
d2c(2,4) = lam(1)*(-44); d2c(2,5) = lam(1)*(-33); d2c(2,6) =
lam(1)*55; d2c(2,7) = lam(1)*187; d2c(4,1) = lam(1)*(-20);
d2c(4,2) = lam(1)*(-44); d2c(5,1) = lam(1)*(-15); d2c(5,2) =
lam(1)*(-33); d2c(6,1) = lam(1)*25; d2c(6,2) = lam(1)*55; d2c(7,1)
= lam(1)*85; d2c(7,2) = lam(1)*187;
```

H.6 objfun_g

Useful for debugging. objfun_g is the same as the cost function in objfun.m.

H.7 objfun_H.m

Useful for debugging.

```
function H = objfun\_H(x, Prob)

H = sparse(Prob.N, Prob.N);
```

H.8 GenerateHessian.java

```
import java.io.*;
```

```
public class TestHessian {
    public static void main(String s[])
    {
         int[] a = {5,11,17,-4,-3,5,17,39};

         Hessian h = new Hessian();
         h.hess(a, "hess.txt");
    }
}

class Hessian {
     public void hess(int[] a, String writetoFileName)
     {
       int len = a.length;
       int[][] hessian = new int[len][len];
       int i = 0;
       int j = 0;

       for (i=0; i<len; i++)
       {
           for (j=0; j<len; j++)
           {
               hessian[i][j] = 0;
           }
       }

       for (i=0; i<2; i++)
       {
           for (j=3; j<7; j++)
           {

               hessian[i][j] = a[i]*a[j];
               hessian[j][i] = hessian[i][j];
           }
       }
/*
       for (i=10; i<12; i++)
       {
           for (j=13; j<17; j++)
           {
```

```
                hessian[i][j] = a[i]*a[j];
                hessian[j][i] = hessian[i][j];
            }
        }
*/
        persist(hessian, writetoFileName, len);

        d2c(hessian, "d2c.txt", len);
    }

    public void persist(
        int[][] hessian, String writetoFileName, int len)
    {
        //FileInputStream in = null;
        //FileOutputStream out = null;
        int i = 0;
        int j = 0;

        try {
          //String inFileName = baseFileName;
          //File inFile = new File(inFileName);

          //String outFileName = inFileName + "Hex";
          //File outFile = new File(writetoFileName);

          //in = new FileInputStream(inFile);
          //out = new FileOutputStream(outFile);
          BufferedWriter writer = new BufferedWriter(
              new FileWriter(writetoFileName));

          for (i=0; i<len; i++)
          {
              for (j=0; j<len; j++)
              {
                  //out.write(hessian[i][j] + '\t');
                  writer.write(hessian[i][j] + "\t");
              }
              //out.write('\n');
              writer.write("\n");
          }
          //out.close();
```

```
            //in.close();
            writer.close();
        } catch(java.io.IOException e) {
            System.out.println("IOEXCEPTION: " + e.toString());
            //Object[] messageArguments = {messages.getString("file")};
        }
    }

    public void d2c(
        int[][] hessian, String writetoFileName, int len)
    {
        int i = 0;
        int j = 0;

        try {
            BufferedWriter writer = new BufferedWriter(
                new FileWriter(writetoFileName));

            writer.write("function d2c=nlcon_d2c(x, lam, Prob)\n");
            writer.write("d2c = spalloc(length(x),length(x),1);\n");

            for (i=0; i<len; i++)
            {
                for (j=0; j<len; j++)
                {
                 if (hessian[i][j] != 0) {
                     writer.write("d2c(" + (i+1) + "," + (j+1) +
                     ") = lam(1)*" + hessian[i][j] + ";\n");
                 }
                }
            }
            writer.close();
        } catch(java.io.IOException e) {
            System.out.println("IOEXCEPTION2: " + e.toString());
        }
    }
}
```

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, et al. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–10, 1990.

[2] AMBA Specification. `http://www.gaisler.com/doc/amba.pdf`, 2003.

[3] Ann Gordon-Ross, Chuanjun Zhang, Frank Vahid and Nikil Dutt. *Tuning caches to applications for low-energy embedded systems*. Kluwer Academic Pub, June 2004.

[4] ARC International. Customizing a Soft Microprocessor Core. `http://www.arccores.com`.

[5] Marnix Arnold and Henk Corporaal. Designing domain-specific processors. In *Proc. of the 9th Int'l Symp. on Hardware/Software Codesign*, pages 61–66, April 2001.

[6] Peter M. Athanas and Harvey F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *IEEE Computer*, 26(3):11–18, April 1993.

[7] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, February 2002.

[8] Amol Bakshi, Jingzhao Ou, and Viktor K. Prasanna. Towards automatic synthesis of a class of application-specific sensor networks. In *Proc. of Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 50–58, 2002.

[9] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. Source-level execution time estimation of C programs. In *Proc. of the 9th Int'l Symp. on Hardware/Software Codesign*, pages 98–103, April 2001.

[10] Florian Braun, John Lockwood, and Marcel Waldvogel. Protocol wrappers for layered network packet processing in reconfigurable hardware. *IEEE Micro*, 22(3):66–74, January 2002.

[11] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *Int'l Journal of High Performance Computing Applications*, 14(3):189–204, 2000.

[12] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *IEEE Computer*, 33:62–69, April 2000.

[13] Roger Chamberlain. *Analysis of parallel algorithms for mixed-mode simulation*. PhD thesis, Washington University in St. Louis, 1989. Printed by University Microfilms International Dissertation Information Service in 1991.

[14] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu. IMPACT: an architectural framework for multiple-instruction-issue processors. In *Proc. of the 18th Int'l Symp. on Computer Architecture*, May 1991.

[15] Hoon Choi, Jong-Sun Kim, Chi-Won Yoon, In-Cheol Park, Seung Ho Hwang, and Chong-Min Kyung. Synthesis of application specific instructions for embedded DSP software. *IEEE Trans. on Computers*, 48(6):603–614, June 1999.

[16] J. Dongarra, K. London, S. Moore, P. Mucci, D. Terpstra, H. You, and M. Zhou. Experiences and lessons learned with a portable interface to hardware performance counters. In *Proc. of Workshop on Parallel and Distributed Systems: Testing and Debugging (at IPDPS)*, April 2003.

[17] J. E. Carrillo Esparza and P. Chow. The effect of reconfigurable units in superscalar processors. In *Proc. ACM Int'l Symp. on Field Programmable Gate Arrays*, pages 141–150, 2001.

[18] Dirk Fischer, Jürgen Teich, Michael Thies, and Ralph Weper. Efficient architecture/compiler co-exploration for asips. In *Proc. of Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 27–34, 2002.

[19] Prentice Hall Professional Technical Reference. `http://www.phptr.com/-articles/article./asp?p=382614`.

[20] Gaisler Research. `http://www.gaisler.com`.

[21] David Goodwin and Darin Petkov. Automatic generation of application specific processors. In *Proc. of Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 137–147, 2003.

[22] Michael Gschwind. Instruction set selection for ASIP design. In *Proc. of the 7th Int'l Symp. on Hardware/Software Codesign*, pages 7–11, May 1999.

[23] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera reconfigurable functional unit. In *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, pages 87–96, 1997.

[24] F. Hillier and G. Lieberman. *Introduction to Operations Research*. Tata McGraw-Hill, New Delhi, seventh reprint edition, 2004.

[25] Edson L. Horta, John W. Lockwood, David E. Taylor, and David Parlour. Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In *Design Automation Conference (DAC)*, New Orleans, LA, June 2002.

[26] Richard Hough, Phillip Jones, Scott Friedman, Roger Chamberlain, Jason Fritts, John Lockwood, and Ron Cytron. Cycle-accurate microarchitecture performance evaluation. In *Proc. of Workshop on Introspective Architecture*, February 2006.

[27] Howard Karloff. *Linear Programming*. Birkhauser Boston, August 1991.

[28] Eric Keller, Gordon J. Brebner, and Philip James-Roxby. Software decelerators. In *FPL*, pages 385–395, 2003.

[29] Paolo Ienne Kubilay Atasu, Laura Pozzi. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *Proc. of Design Automation Conf.*, June 2003.

[30] Mika Kuulusa, Jari Nurmi, Janne Takala, Pasi Ojala, and Henrik Herranen. A flexible DSP core for embedded systems. *IEEE Design and Test of Computers*, 14(4):60–68, 1997.

[31] LEON Specification. `http://www.gaisler.com/doc/leon2-1.0.21--xst.pdf`, 2003.

[32] John W Lockwood. Evolvable Internet hardware platforms. In *The Third NASA/DoD Workshop on Evolvable Hardware (EH'2001)*, pages 271–279, July 2001.

[33] John W. Lockwood. The Field-programmable Port Extender (FPX). `http://www.arl.wustl.edu/arl/projects/fpx/`, December 2003.

[34] John W. Lockwood. Reconfigurable network group. `http://www.arl.wustl.edu/arl/projects/fpx/reconfig.htm`, May 2004.

[35] MathWorks. `http://www.mathworks.com`.

[36] Ivan C. Kraljic Olivier Hebert and Yvon Savaria. A method to derive application-specific embedded processing cores. In *Proc. of the 8th Int'l Symp. on Hardware/Software Codesign*, pages 88–92, May 2000.

[37] Shobana Padmanabhan, Phillip Jones, David V. Schuehler, Scott J. Friedman, Praveen Krishnamurthy, Huakai Zhang, Roger Chamberlain, Ron K. Cytron, Jason Fritts, and John W. Lockwood. Extracting and improving microarchitecture performance on reconfigurable architectures. *International Journal of Parallel Programming*, 33(2–3):115–136, June 2005.

[38] Christian Plessl, Rolf Enzler, Herbert Walder, Jan Beutel, Marco Platzner, Lothar Thiele, and Gerhard Troester. The case for reconfigurable hardware in wearable computing. *Personal and Ubiquitous Computing*, 7(5):299–308, 2003.

[39] Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Stephen A. Herrod. Using the SimOS machine simulator to study complex computer systems. *ACM Trans. on Modeling and Computer Simulation*, 7(1):78–103, January 1997.

[40] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale. The NAPA adaptive processing architecture. In *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, pages 28–37, 1998.

[41] David Pellerin Scott Thibault. *Practical FPGA Programming in C*. Prentice Hall PTR, April 2005.

[42] Barry Shackleford, Mitsuhiro Yasuda, Etsuko Okushi, Hisao Koizumi, Hiroyuki Tomiyama, and Hiroto Yasuura. Memory-CPU size optimization for embedded system designs. In *Proc. of Design Automation Conf.*, pages 246–251, June 1997.

[43] Timothy Sherwood, Mark Oskin, and Brad Calder. Balancing design options with sherpa. In *CASES '04: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 57–68, New York, NY, USA, 2004. ACM Press.

[44] H. Singh, Ming-Hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. on Computers*, 49:465–481, May 2000.

[45] Brinkley Sprunt. Pentium 4 performance-monitoring features. *IEEE Micro*, 22(4):72–82, 2002.

[46] Stretch, Inc. S5500 Software-Configurable Processor. `http://www.stretchinc.com`.

[47] Sun Microsystems, Inc. Java Servlet Specification. `http://java.sun.com/products/servlet/download.html`.

[48] Sun Microsystems, Inc. JSP - Apache Tomcat. `http://java.sun.com/products/jsp/tomcat`.

[49] Kei Suzuki and Alberto Sangiovanni-Vincentelli. Efficient software performance estimation methods for hardware/software codesign. In *Proc. of Design Automation Conf.*, pages 605–610, June 1996.

[50] Synplicity Inc. `http://www.synplicity.com`.

[51] Roberto E. Ko T. Vinod Kumar Gupta and Rajeev Barua. Compiler-directed customization of ASIP cores. In *Proc. of the 10th Int'l Symp. on Hardware/Software Codesign*, pages 97–102, May 2002.

[52] Tensilica, Inc. http://www.tensilica.com.

[53] Tomlab Optimization. Users Guide for TOMLAB /MINLP. http://www.-tomlab/.biz/docs/TOMLAB_MINLP.pdf.

[54] Tilman Wolf and Mark A. Franklin. Commbench - a telecommunications benchmark for network processors. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 154–162, Austin, TX, April 2000.

[55] Xilinx Inc. http://www.xilinx.com.

[56] Xilinx Corp. MicroBlaze Product Brief, FAQ. http://www.xilinx.com/-ipcenter/processor_central/microblaze.htm, May 2001.

[57] Xilinx Inc. Virtex-e 1.8v field programmable gate arrays, February 2000.

Vita

Shobana Padmanabhan

Place of Birth Tiruchirappalli, India

Degrees Master of Management Studies, Birla Institute of Technology and Science, Pilani, India
Master of Science in Computer Science, Washington University, St.Louis, USA, May 2006

Professional Societies IEEE (Institute of Electrical and Electronics Engineers, Inc.)

Publications *A Hardware Implementation of Hierarchical Clustering;* by Shobana Padmanabhan, Moshe Looks, Dan Legorreta, Young Cho and John Lockwood; *Poster summary in proceedings of: Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, Apr 24-26, 2006.

Automatic Application-Specific Microarchitecture Reconfiguration; by Shobana Padmanabhan, Ron K. Cytron, Roger D. Chamberlain, and John W. Lockwood; *In Proceedings of: 13th Reconfigurable Architectures Workshop (RAW) at IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, Apr 25-29, 2006.

Extracting and Improving Microarchitecture Performance on Reconfigurable Architectures; by Shobana Padmanabhan, Phillip Jones, David V. Schuehler, Scott J. Friedman, Praveen Krishnamurthy, Huakai Zhang, Roger Chamberlain, Ron K. Cytron, Jason Fritts, and John W. Lockwood; *International Journal on Parallel Programming (IJPP)*, 33(23):115136, June 2005.

Semi-Automatic Microarchitecture Configuration of Soft-Core Systems; by Shobana Padmanabhan, John Lockwood, Ron Cytron,

Roger Chamberlain, and Jason Fritts; *In Proceedings of: Workshop on Architecture Research using FPGA Platforms (WARFP) at 11th Inter. Symp. on High Performance Computer Architecture (HPCA11)*, San Fransisco, CA, USA, Feb 13, 2005.

Microarchitecture Optimization for Embedded Systems;
by David V. Schuehler, Benjamin C. Brodie, Roger D. Chamberlain, Ron K. Cytron, Scott J. Friedman, Jason Fritts, Phillip Jones, Praveen Krishnamurthy, John W. Lockwood, Shobana Padmanabhan, and Huakai Zhang; *In Proceedings of: 8th Inter. Symp. on High Performance Embedded Computing (HPEC)*, Boston, MA, Sep 28-30, 2004.

Extracting and Improving Microarchitecture Performance on Reconfigurable Architectures;
by Shobana Padmanabhan, Phillip Jones, David V. Schuehler, Scott J. Friedman, Praveen Krishnamurthy, Huakai Zhang, Roger Chamberlain, Ron K. Cytron, Jason Fritts, and John W. Lockwood; *In Proceedings of: Workshop on Compilers and Tools for Constrained Embedded Systems (CTCES) at International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Washington DC, Sep 22, 2004.

Liquid Architecture;
by Phillip Jones, Shobana Padmanabhan, Daniel Rymarz, John Maschmeyer, David V. Schuehler, John W. Lockwood, and Ron K. Cytron; *In Proceedings of: Workshop on Next Generation Software at IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, Apr 26, 2004.

May 2006