IOWA STATE UNIVERSITY
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

1-1-2002

The use of a reconfigurable functional cache in a digital signal processor: power and performance

Kathryn Fountain Gossett
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
Gossett, Kathryn Fountain, "The use of a reconfigurable functional cache in a digital signal processor: power and performance" (2002). *Retrospective Theses and Dissertations.* 19860.
https://lib.dr.iastate.edu/rtd/19860

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

The use of a reconfigurable functional cache in a digital signal processor: power and performance

by

Kathryn Fountain Gossett

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Akhilesh Tyagi (Major Professor)
Arun Somani
Julie Dickerson
Dan Ashlock

Iowa State University

Ames, Iowa

2002

Copyright © Kathryn Fountain Gossett, 2002. All rights reserved.

المنارة للاستشارات

www.manaraa.com

Graduate College
Iowa State University

This is to certify that the master's thesis of

Kathryn Fountain Gossett

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

للاستشارات المنارة
www.manaraa.com

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES ix

ABSTRACT x

CHAPTER 1. INTRODUCTION 1

1.1 Performance Improvements in General Purpose Processors and Digital Signal Processors 1

1.2 Problem Explored 3

1.3 Thesis Organization 5

CHAPTER 2. LITERATURE REVIEW 6

2.1 Architectures that Utilize Existing Cache as a Reconfigurable Component 6

2.1.1 Balanced Architectures 6

2.1.1.1 Adaptive Balanced Computing 6

2.1.1.2 Reconfigurable Modules 7

2.1.2 Reconfigurable Cache for Instruction Reuse 8

2.2 Reconfigurable Hardware Extensions to Improve Performance and the I/O Bottleneck 8

2.2.1 Garp 9

2.2.2 OneChip 9

2.2.3 Chimaera 10

2.2.4 MorphoSys 10

2.3 Other Reconfigurable Designs 11

2.3.1 The MIT Raw Microprocessor 11

2.3.2 A System on a Chip 11

المنارة للاستشارات
www.manaraa.com

2.3.3 Another System on a Chip 12

2.3.4 CRISP 13

2.4 Reconfigurable Digital Signal Processors 13

2.4.1 DSP Reconfigurable Logic Hybrid 14

2.4.2 Pleiades 15

CHAPTER 3. MATERIALS AND METHODS 21

3.1 Current C64x Architecture 21

3.1.1 Architecture Overview 21

3.1.2 Instruction Set Architecture 23

3.1.3 Pipeline 25

3.1.4 On-Chip Cache 27

3.1.4.1 L1 Program Cache 27

3.1.4.2 L1 Data Cache 28

3.1.4.3 L2 Unified Cache 29

3.1.5 Power Consumption 30

3.2 Reconfigurable Architecture 32

3.2.1 Architecture Overview 32

3.2.2 Simulator 32

3.2.2.1 Cache Simulation 34

3.2.2.2 Power Estimation 35

3.2.3 Reconfigurable Functional Cache 40

3.2.4 Benchmarks 42

3.2.5 Kernel Implementations in RFC 43

3.2.5.1 DCT 44

المنارة للاستشارات

www.manaraa.com

3.2.5.1.1 1-D DCT with 16 Constant Coefficient Multipliers 47

3.2.5.1.2 1-D DCT with 32 Coefficients and Dedicated Hardware 48

3.2.5.1.3 1-D DCT with a 16KB Module................................ 49

3.2.5.2 Convolution.. 50

3.2.5.3 FFT.. 52

3.2.5.4 32-Tap and 256-Tap FIR.. 56

CHAPTER 4. RESULTS AND DISCUSSION.. 57

4.1 DCT.. 58

4.1.1 Performance 58

4.1.2 Power 60

4.1.3 Energy Requirements.. 61

4.1.4 L1 Data Cache Miss Rates.. 63

4.2 Convolution.. 64

4.2.1 Performance 64

4.2.2 Power 66

4.2.3 Energy Requirements.. 67

4.2.4 L1 Data Cache Miss Rates.. 68

4.3 FFT... 69

4.3.1 Performance 69

4.3.2 Power 71

4.3.3 Energy Requirements.. 72

4.3.4 L1 Data Cache Miss Rates.. 73

4.4 FIR, 32-Tap and 256-Tap... 74

المنارة للاستشارات

www.manaraa.com

4.4.1 Performance 74

4.4.2 Power 76

4.4.3 Energy Requirements 77

4.4.4 L1 Data Cache Miss Rates 78

CHAPTER 5. CONCLUSIONS 81

REFERENECES 84

www.manaraa.com

LIST OF FIGURES

Figure 1. The SHARC DSP with reconfigurable logic 15
Figure 2. Overview of the Design and Software Generation Process 17
Figure 3. Overview of the Pleiades Architecture 18
Figure 4. The Maia Reconfigurable DSP 19
Figure 5. Block Diagram of TMS320C64x 23
Figure 6. Example of the ADD(U) instruction layout 25
Figure 7. Illustration of Fetch Packets Progressing Through the Pipeline 26
Figure 8. Level 2 SRAM/Cache Organizations 30
Figure 9. Diagram of a Reconfigurable Cache 39
Figure 10. 1-D DCT for the rows of an 8x8 Block 46
Figure 11. Design of a signed 4x8 Constant Coefficient Multiplier using 4-LUTs 47
Figure 12. Layout of 4-LUTs for Two DCT coefficients 48
Figure 13. Layout of LUTs for Convolution 52
Figure 14. Layout of LUTs for FFT 55
Figure 15. Layout of LUTs for FIR 56
Figure 16. RFC DCT Kernel Speedups over Non-RFC DCT Kernels 59
Figure 17. Overall Compress Benchmark Speedups 60
Figure 18. Percentage of Non-RFC power consumed by RFC Benchmark 61
Figure 19. Percentage of Non-RFC Energy Required by RFC Benchmark 62
Figure 20. Energy Savings for RFC Compress Benchmark 63
Figure 21. Increases in Miss Rates for RFC Compress Benchmarks 64
Figure 22. Speedup of the RFC Convolution Kernel for Edge Detect 65

www.manaraa.com

Figure 23. Overall Benchmark Speedups for Edge Detect 66

Figure 24. Percentage of Non-RFC power consumed by RFC Benchmarks 67

Figure 25. Percentage of Non-RFC Energy Required by the Edge Detect Benchmark 68

Figure 26. Energy Savings for Edge Detect Benchmark 68

Figure 27. Miss Rate Increase (Decrease) for RFC over Non-RFC 69

Figure 28. RFC Kernel Speedup over Non-RFC for FFT 70

Figure 29. Overall Benchmark Speedups for the Spectral Benchmark 71

Figure 30. Percentage of Non-RFC Power Consumed by RFC 72

Figure 31. Percentage of Non-RFC Energy Required by the Spectral Benchmark 73

Figure 32. Energy Savings for Spectral Benchmark 73

Figure 33. Increases in Miss Rates for the RFC Spectral Benchmarks 74

Figure 34. FIR, 32-tap, Kernel Speedup for RFC Implementation 75

Figure 35. FIR, 256-tap, Kernel Speedup (Decrease) for RFC 76

Figure 36. Percentage of Non-RFC Energy Required by the 32-Tap FIR Kernel 78

Figure 37. Energy Savings for the 32-Tap FIR Filter Kernel 78

Figure 38. Miss Rate Increases for 32-tap Reconfigurable FIR 80

Figure 39. Miss Rate Increases for 256-tap Reconfigurable FIR 80

المنارة للاستشارات

www.manaraa.com

LIST OF TABLES

Table 1. Measured Power Consumption Values for the C64x at 600 MHz, 1.4 V 31

Table 2. Look-Ups Required for LUT structures 48

Table 3. Look-Ups/Cycles Required for LUT and Dedicated Hardware Structures 49

Table 4. Look-Ups Required for LUT structures in Convolution 51

Table 5. Look-Ups Required for LUT structures in FFT 54

Table 6. Look-Ups Required for LUT structures in FIR 56

Table 7. Overall Benchmark Speedups for Compress 59

Table 8. Overall Benchmark Speedups for Edge Detect 66

Table 9. Overall Benchmark Speedups Observed for the Spectral Benchmark 70

المنارة للاستشارات

www.manaraa.com

ABSTRACT

Due to the computationally intensive nature of the tasks that digital signal processors (DSP) are required to perform, it is desirable to decrease the time required to execute these tasks. Minimizing the execution time required for the various algorithms that are commonly and frequently executed (ex: FIR filters) will improve the overall performance. It is known that hardware is able to execute algorithms faster than software, however, due to the size limitations of embedded DSP, not all of the necessary algorithms can be implemented in hardware. A reconfigurable cache architecture in combination with a DSP is proposed as an alternative to increase algorithm performance by using reconfigurable hardware rather than dedicated hardware. Another important issue to consider for embedded processors is the power consumption of the DSP. Due to the fact that most embedded processors operate by battery power, energy efficiency is a necessity. This study looks at the power requirements of a DSP with reconfigurable cache to determine the viability of such an architecture in an embedded system. Others have shown that reconfigurable cache in conjunction with a general purpose processor improves performance for some DSP benchmarks. This study shows that a DSP/reconfigurable cache combination can achieve kernel performance gains ranging from 10-350 times that of a DSP architecture operating alone and can achieve overall benchmark speedups ranging from 1.02 to 1.91 times that of the existing DSP architecture. Further, relative power consumption results show that the power consumption of the reconfigurable architecture is approximately 85 to 95% of the current architecture (5-15% power savings) and attains energy savings ranging from approximately 14 to 50%.

www.manaraa.com

CHAPTER 1. INTRODUCTION

Forward Concepts projects that DSP sales will surpass the $12 billion mark by the year 2005 [1]. The projected increase in sales is expected to be due primarily to cellular phone sales but the increasing popularity of multimedia devices, which use DSPs, like MP3 players and digital recorders also contribute to the market. As this market continues to grow manufacturers of DSPs will continue to search for ways to enhance their products. A DSP that is faster, smaller and more energy efficient than its competitors will enjoy a larger slice of the market.

1.1 Performance Improvements in General Purpose Processors and Digital Signal Processors

In today's fast-paced society people do not like to wait, thus technology users expect prompt and precise computing. If the user is working on a personal computer (PC) it is unlikely that the PC has a specialized DSP chip, thus it is up to the general purpose processor (GPP) to meet these expectations. In an effort to increase performance most modern GPPs use multi-level caches to speed up the retrieval of data from main memory. However, multimedia processing and other common DSP computations fall into a category of computing referred to as single instruction, multiple data (SIMD). This refers to the streaming nature of the data processing where many data elements will be processed in the same manner (thus single instruction). This type of processing does not take advantage of temporal locality. Computer designers rely on temporal and spatial locality of data accesses to justify the large fraction of chip area dedicated to cache. DSP processing under-utilizes large caches [2]. In addition to multi-level caches, GPPs have increased clock speed, implemented Harvard memory architectures (separate data and instruction memories for concurrent accesses) and utilized

المنارة للاستشارات
www.manaraa.com

out-of-order processing (which can take advantage of instruction-level parallelism) to further improve performance. While some DSPs have implemented out-of-order processing and Harvard memory architectures to increase performance, few have increased clock speeds to the level of GPPs due to the fact that as clock speed increases, so does the amount of power consumed. Most DSPs are used in embedded, mobile systems where energy-efficiency is just as crucial as fast computing. Thus, DSPs have explored other avenues to improve performance such as implementing very long instruction word (VLIW) architectures. VLIW architectures are another method of increasing the number of instructions that can be executed in parallel. In a VLIW architecture several instructions are fetched at once and then separated into execute packets depending upon which instructions do not have data, name or control dependencies and thus can be executed in parallel. VLIW architectures rely upon sophisticated compilers that can statically schedule the code in advance to determine which instructions can execute in parallel [3]. New methods of analog-to-digital conversion have also been implemented to increase I/O. With increased I/O speeds and real-time constraints, the possibility exists to increase computation speed as well. To enhance computing performance some of the most frequently used algorithms are implemented in hardware. Unfortunately, space limitations prevent all necessary algorithms from being implemented in hardware.

Recently, some DSP manufactures such as Texas Instruments (TI) have started to include on-chip cache to help increase performance [4]. Since most DSP data computations are of the streaming nature, these cache sizes are so far relatively small (ex: 16KB). However, [5] presents an argument that cache size will continue to increase on DSPs, which can be

المنارة للاستشارات
www.manaraa.com

supported by the fact that TI has increased the cache size in their TMS320C64x line of processors over what was included on their TMS320C62x line.

In the past few years several types of reconfigurable hardware architectures have been proposed to help GPPs better utilize the hardware at their disposal and to increase performance to meet the needs of computationally intensive applications. Another approach that has been taken in some DSP and GPP systems is to use a field-programmable gate array (FPGA) to perform some of the highly repetitive computations such as discrete cosine transform (DCT) or finite impulse response filter (FIR). However, these designs suffer from an input/output bottleneck due to the fact that all the data necessary for the computations cannot be stored in the FPGA.

1.2 Problem Explored

Most of the reconfigurable GPP designs have not explored the effect of reconfigurability on power consumption. While it is the author's belief that society as a whole must consider ways to improve energy efficiency, in DSP applications this is a necessity and thus changes that affect power consumption must be taken seriously. Therefore, this research will explore not only the performance improvements that a reconfigurable architecture can lend a DSP, but also its effects on power consumption. It was shown in [2, 6, 7, 8, 9] that utilizing part of the level-one cache as a functional unit or other type of reconfigurable hardware can enhance the performance of a GPP. A fine-grained reconfigurable coprocessor for an Analog Device's SHARC DSP was proposed in [10] but it did not measure the power consumption effects of the reconfigurable coprocessor. The research team at the University of California, Berkley, that is working on the Pleiades project have written a plethora of papers [11, 12, 13,

المنارة للاستشارات
www.manaraa.com

14, 15, 16, 17, 18, 19, 20, 21] on their proposed reconfigurable DSP. The Pleiades project focuses on designing and implementing a low-energy reconfigurable DSP using an ARM microprocessor and a variety of "satellites" such as a low power field programmable gate array (FPGA), a multiply and accumulator, memory cells, etc. While they have extensively studied the energy effects of their reconfigurable architecture, this was the only literature the author could find on reconfigurable DSPs that also explored power consumption. Both of these reconfigurable DSP designs require additional hardware to implement. Possibly due to the fact that cache on a DSP chip is a relatively new occurrence, this author could not find any literature that explored the use of converting part of the cache to a reconfigurable functional unit for performance improvements. This study proposes a reconfigurable DSP that utilizes existing chip structures for implementation of the reconfigurable hardware. The proposed design uses a reconfigurable cache similar to [6] and [7] to increase performance in a DSP. The difference between this study and the research done in [6] and [7] is that this study uses the reconfigurable cache with a VLIW DSP whereas [6] and [7] used the reconfigurable cache with a superscalar GPP. Further, this research examines power and energy issues; [6] and [7] did not. The performance, energy and power consumption effects will be measured using a simulator that was created by this author. The simulator in [22] that simulates the TI TMS320C62x VLIW processor was modified to simulate the TI TMS320C64x VLIW processor. The simulator in [22] did not include cache simulation, so the cache portion of [23] was modified to work with this code and added to the simulator. The power measurement abilities were achieved by merging the power files from [24] with the new simulator. Due to the fact that the power files in [24] were created to estimate the power consumption of an out-of-order processor these files had to be modified to remove the logic and structure components for out-of-order prediction and miss-prediction correction.

المنارة للاستشارات
www.manaraa.com

These modifications, along with other minor modifications make the power estimates more closely reflect the structure of the TMS320C64x VLIW processor.

1.3 Thesis Organization

In order to give the reader a better understanding of what has been researched in the area of reconfigurable processors, Chapter 2 will focus on an overview of the various proposed reconfigurable GPP and DSP architectures that attempt to either better utilize cache or improve upon the I/O bottleneck. These architectures can be divided into two broad categories: architectures that utilize existing resources in new, reconfigurable ways and architectures that add additional reconfigurable hardware to the chip. A review of previous research into the energy efficiency of reconfigurable DSPs will also be discussed. Chapter 3 will first present background information on the Texas Instruments TMS320C64x digital signal processor, which is used as the base processor in this study and then present the reconfigurable architecture proposed by this author. The experimental setup and multimedia benchmarks that are used will be described in Chapter 3 as well. Chapter 4 will present the results of analysis of these benchmarks on the reconfigurable DSP and Chapter 5 will summarize the conclusions.

المنارة للاستشارات

www.manaraa.com

CHAPTER 2. LITERATURE REVIEW

2.1 Architectures that Utilize Existing Cache as a Reconfigurable Component

As stated above, the different reconfigurable architectures can be divided into two categories: architectures that utilize existing cache as some form of reconfigurable hardware to increase cache utilization and architectures that add reconfigurable hardware to increase performance and eliminate the I/O bottleneck. The architectures that utilize cache will be examined first.

2.1.1 Balanced Architectures

2.1.1.1 Adaptive Balanced Computing

A processor chip can be viewed as consisting of components that do one of two basic functions, memory or computation [6], [7]. As stated in [7], most caches in existence consume over half of the area of a modern microprocessor chip. However, some processes cannot efficiently utilize a large cache. Thus, the authors proposed converting part of the cache (a memory component) into a reconfigurable functional unit (RFU). The goal was to design an architecture that was more balanced in terms of bandwidth needs, thereby improving the performance of the architecture. This architecture used an out-of-order issue, superscalar processor (simulated by SimpleScalar [23]) in conjunction with the reconfigurable module. Balanced reconfigurable architectures were also studied in [2] and [8]. A set-associative cache was used with one of the ways (modules) within the cache converted to a reconfigurable module. The functional modules were given computation capabilities by making them multi-bit lookup tables. Additional hardware as added to each reconfigurable module to act as input and output buffers. The purpose of the buffers was to keep the data flow into and out of the functional modules in order. The out-of-order GPP had

المنارة للاستشارات
www.manaraa.com

four arithmetic functional units each for integer and floating point as well as one multiplier each for integer and floating point. The focus of [6] and [7] was to first examine whether or not this architecture would improve performance. When performance gains were observed, [6] also focused on fine-tuning the improvements by examining various cache sizes and configurations. Further, this study takes a conservative approach for the filter kernels by assuming that an RFU cache access would take three cycles rather than just one. The kernel functions that were mapped to the reconfigurable cache (RC) were FIR (16 tap and 256 tap) and DCT/IDCT as well as an infinite impulse response (IIR) filter. Cache sizes and associations that were compared were 32KB 2-way, 64KB 2-way, 64KB 4-way, 128KB 4-way. Additionally, direct-mapped, 2-way and 4-way caches were compared for 16KB, 32KB, 64KB and 128KB sizes.

2.1.1.2 Reconfigurable Modules

This design is similar to the architecture used in [6] and [7] that converts part of a traditional level-one cache into a look-up table with minor implementation differences in terms of the reconfigurable cache. The primary difference between this design and the previous one is that existing cache modules are used for input and output space rather than adding hardware to implement the input and output buffers. Specifics of the processor were not given, but it was stated that the default simulator values were used. The level one 128KB data cache was divided into 16 modules. Four of the modules carried out normal cache operations while the other 12 modules could be used as either a functional unit or a register mapped module. The kernel functions that were implemented in the reconfigurable modules were a FIR filter and discrete cosine transform/inverse discrete cosine transform (DCT/IDCT).

المنارة للاستشارات
www.manaraa.com

2.1.2 Reconfigurable Cache for Instruction Reuse

Several different possible uses of RC for multimedia applications were given in [9]. Suggestions included using portions of the cache as lookup tables or buffers for applications such as value prediction, memoization and instruction reuse. Another suggestion was to use a portion of the cache as a software or hardware data prefetch area. Due to the streaming nature of multimedia data, storing data on-chip, in advance, would improve performance. The third possibility was to use a portion of the cache as memory that was directly under the control of the compiler or an application. Instruction reuse was the option implemented in [9]. A 1 GHz, eight-way issue, out-of-order processor was simulated for this study. However, the RSIM simulator [25], rather than the SimpleScalar simulator, was used. The level one cache was 128KB 4-way associative. This cache was divided into two 64KB 2-way associative caches and one way of each division was used as a buffer area to store instruction reuse entries. Each entry consisted of the instruction's operand values for arithmetic and logical instructions and addresses for memory instructions. A buffer latency of 2 cycles was assumed.

2.2 Reconfigurable Hardware Extensions to Improve Performance and the I/O Bottleneck

Most of the reconfigurable architectures that have been proposed to date make use of additional hardware component(s) that are similar to, or actually are field-programmable gate arrays (FPGAs). FPGAs are fine-grained reconfigurable lookup tables that can significantly improve performance. However, as stated in [26], the drawbacks to systems that utilize only FPGAs are large reconfiguration overhead times and their inability to hold all the data necessary to complete most computationally intensive tasks (I/O bottleneck). The following architectures examine ways of combining FPGAs or FPGA-like structures with core

المنارة للاستشارات
www.manaraa.com

processors and on-chip memory in an attempt to improve performance while reducing the I/O bottleneck.

2.2.1 Garp

A type of reconfigurable hardware that attempts to eliminate the I/O bottleneck is Garp [26]. Garp places a FPGA on the processor chip and gives it access to both the data cache and off-chip memory thereby giving the FPGA access to all the data necessary to perform data-intensive computations. The FPGA operates in slave mode to the Reduced Instruction Set Computer (RISC) core processor and is referred to as a reconfigurable array (RA). The RA is divided into blocks with 24 columns of blocks. The number of rows of blocks can vary depending on the needs of the application being processed. The granularity of this design is 2 bits, with each block reading as many as four 2-bit pairs and giving up to two 2-bit outputs. Unlike the architectures reviewed above, the RISC processor used in this design is a single-issue processor.

2.2.2 OneChip

Another reconfigurable architecture that incorporates FPGA-like extensions onto the processor chip is OneChip [27]. OneChip uses a 32-bit RISC core processor. The processor is a single-issue, in-order processor with one existing functional unit (FU). OneChip extends the functioning capabilities of the processor by placing several programmable functional units (PFUs) in parallel with the existing FU. The OneChip architecture was improved upon in [28], which introduced OneChip-98. The key difference in OneChip-98 is placement of the reconfigurable hardware in the instruction decode stage of the pipeline. Here the FPGAs function as a flexible interface with memory that has high bandwidth and is capable of

المنارة للاستشارات
www.manaraa.com

buffering instructions, providing local storage for data and conducting logic computations. This extension allowed the processor and reconfigurable logic to operate in parallel at a higher throughput rate. Further, this model upgraded the core processor to an out-of-order RISC processor.

2.2.3 Chimaera

Chimaera is another reconfigurable design that uses embedded FPGA-like devices on-chip with the GPP [29]. This system uses the reconfigurable array of FPGA-like devices as a cache for storing recently used reconfigurable functional unit (RFU) instructions. When a new RFU instruction is loaded, it replaces the least recently used instruction, thereby achieving dynamic partial reconfiguration during runtime.

2.2.4 MorphoSys

MorphoSys [30], [31] proposes yet a different reconfigurable architecture for improving multimedia processing performance. The MorphoSys design consists of a simplified MIPS-type 100 MHz GPP that was named "TinyRISC" combined on a chip with an 8 by 8 coarse-grained reconfigurable cell array. TinyRISC operates on 32-bit words, however, the smallest data size the reconfigurable cell array can work with is 16 bits (thus, it is coarse-grained). The reconfigurable cell array, although similar in basic layout to a FPGA is very different at the cellular level. Each cell is composed of an ALU-multiplier, two multiplexers and a shift unit. The row/column interconnection of these cells allows MorphoSys to reconfigure the interconnections and thus change the overall functionality of the array. A simulator, MorphoSim, has been created to measure MorphoSys performance.

المنارة للاستشارات
www.manaraa.com

2.3 Other GPP Reconfigurable Designs

2.3.1 The MIT Raw Microprocessor

The MIT Raw microprocessor design [32], [33] takes a very different approach from the others in this category in that rather than combining reconfigurable hardware components with a single processor, the entire chip is composed of numerous reconfigurable tiles each with its own RISC-like processor. The architecture is called Raw because the compiler is made aware of (exposed to) the layout of the internal hardware. This awareness enables the compiler to map functions to hardware more optimally than if the compiler was unaware of the underlying hardware. The Raw microprocessor combines various tiles structures on an interconnect fabric to make a structure that is similar in nature to a coarse-grained FPGA. In addition to each tile having its own RISC-like processor, each tile also has its own SRAM memory. This distributed memory helps to eliminate the I/O bottleneck. The Raw microprocessor is usually combined with off-chip DRAM for additional storage capacity. This architecture was analyzed for several different applications to determine the optimal layout of the tiles and amount of SRAM per tile. The Raw architecture in [32], [33] operates at 25 MHz and utilizes one billion transistors. Further research into the Raw microprocessor was reported in [34]. The Raw architecture was implemented at about 225 MHz with 32 tiles and 0.122 billion transistors in [34].

2.3.2 A System on a Chip

An adaptive system on a chip (aSOC) architecture has been proposed in [35]. Increased flexibility and performance are achieved by aSOC by connecting different tiles for different activities. For example, a field-programmable gate array type tile, a GPP RISC type tile, and a digital signal processor type tile may all be interconnected. Other tiles may be RAM or

المنارة للاستشارات
www.manaraa.com

multiple tiles can even be combined to implement a VLIW processor. The interconnection of these tiles for various tasks is scheduled statically prior to execution. However, the implementation of the tiles themselves can be reconfigured dynamically depending on the needs of the application. Two different configurations of the aSOC were explored in [35]. The first consisted of 2 RISC tiles, 1 FPGA tile and 6 multiplier-accumulators. The second consisted of 4 RISC tiles, 2 FPGA tiles and 10 multiplier-accumulators. The RISC tiles were MIPS R4000 processors simulated using SimpleScalar. The FPGA tiles were Altera brand, simulated using an Altera FPGA simulator. In [36] several algorithms were implemented with each algorithm mapped to its own tile within the aSOC. Energy efficiency was considered while mapping these algorithms to the tiles.

2.3.3 Another System on a Chip

The authors of [37] propose another system on a chip design for reconfigurable computing. This design also target multimedia applications in portable devices with a goal of increasing computing speed. The core reconfigurable component of this design is called a Dnode (Data node) and is comprised of an ALU and a few storage registers. Thus the Dnode design is similar to the MorphoSys cell. The Dnodes are arranged around a core controller in a layered, interconnected ring rather than a square array. The ring structure allows for easier data feedback according to [37]. This design operates at a clock frequency of 200 MHz and is reported to be capable of 1600 MIPS (millions instructions per second). For details on other system on a chip designs please see [38] and [39].

المنارة للاستشارات
www.manaraa.com

2.3.4 CRISP

The reconfigurable architecture proposed in [40] is different from all the others discussed thus far in that it is a VLIW processor with a reconfigurable functional unit added to the chip. The name, CRISP, is an acronym for Configurable and Reconfigurable Instruction Set Processor. The CRISP architecture has five integer functional units, 2 load/store units and one branch unit. The reconfigurable functional unit (RFU) is an array of coarse-grained processing elements that can be configured to handle 8, 16 or 32-bit data. These data sizes were chosen because CRISP is specifically targeted at multimedia applications and most multimedia applications use these data sizes. The CRISP processor has 16KB of level one instruction cache and 16KB of level one data cache. The level two cache is unified and is 2MB. Additionally, the CRISP processor's reconfigurable functional unit contains 32 processing elements and the CRISP processor has a 4KB level one configuration cache and a 256KB level two configuration cache. The configuration cache provides nearby storage space for the configuration data so that the reconfiguration time of the RFU is kept to a minimum.

2.4 Reconfigurable Digital Signal Processors

The literature available on reconfigurable DSPs is much more limited than the literature available on reconfigurable GPPs. Two different reconfigurable DSP designs were reviewed and details of them follow. Of these two designs, one designs their reconfigurable DSP to be energy efficient.

المنارة للاستشارات
www.manaraa.com

2.4.1 DSP Reconfigurable Logic Hybrid

An Analog Devices' SHARC floating point DSP was chosen in [10] as the base DSP. The SHARC, which is implemented in 0.6 micron technology has 512KB on-chip memory that is organized as two banks each with an I/O port so that concurrent memory accesses to each bank do not conflict. The SHARC also has a multiplier functional unit, an arithmetic logic unit and a shifter, all of which can operate in parallel. SHARC increases its I/O speed by utilizing several on-chip I/O peripherals such as a direct memory access (DMA) controller. The reconfigurable logic is positioned on the chip so that it has access to all of the I/O ports and to the register files. This way the reconfigurable logic can act partially as a coprocessor and partially as a functional unit. In [10] the reconfigurable logic itself was described as resembling the Xilinx 4000 series of FPGAs but the authors stated that a coarse-grained device similar to the reconfigurable logic used in Garp [26] would probably work better. Approximately 1000 configurable logic blocks (CLBs) were used in the reconfigurable logic. The clock speed of SHARC is not given. Nor is the clock speed of the reconfigurable logic listed. However, [10] does state that they operate at the same frequency. Figure 1 gives an overview of their proposed model.

المنارة للاستشارات
www.manaraa.com

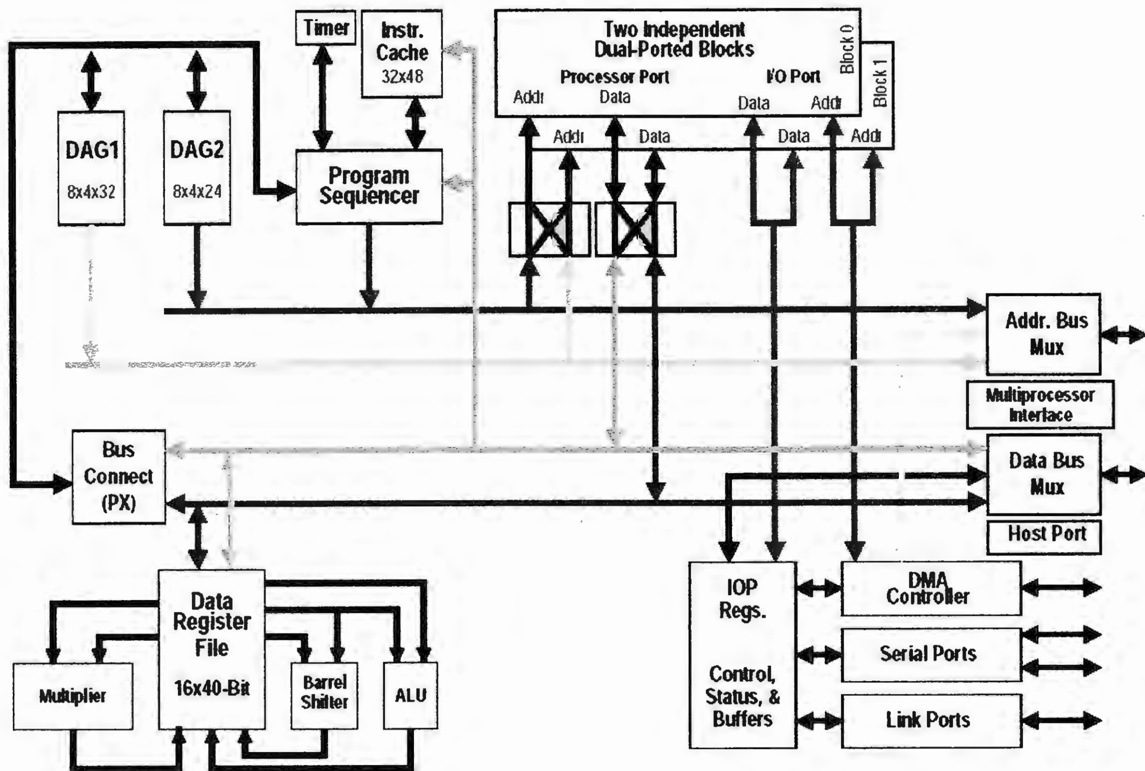

Figure 1. The SHARC DSP with reconfigurable logic [10].

2.4.2 Pleiades

The Pleiades research group at the University of California, Berkeley, have invested a lot of time into the exploration of their reconfigurable DSP architecture. This has allowed them to explore all aspects of the Pleiades architecture from the method to use to map computationally intensive algorithms onto their reconfigurable architecture [13], [14] to what type of reconfigurable interconnect to use [15] to a compiler for generating the code to execute their design [16]. Figure 2 details the design methodology and the code generation process.

During these investigations three main factors were behind all of their decisions: power, delay and area. As stated in [13] each of these factors play heavily in determining the success of a DSP architecture. Unfortunately, optimizations of one of these factors may have detrimental effects on the other factors. Therefore, it is necessary to consider all three factors simultaneously when considering design changes. According to [11] the key to designing an energy-efficient reconfigurable architecture is to closely match the granularity of the algorithm to the granularity of the unit that processes the architecture. The name Pleiades refers to a general reconfigurable architecture that can have many different specific instantiations. The various components that make up the Pleiades architecture area an ARM microprocessor, a hierarchical generalized mesh reconfigurable interconnect, memory units, address generators, multiply and accumulators (MACs), arithmetic logic units (ALUs), I/O ports and low power FPGAs that they designed [41]. Each of these units is referred to as a satellite. Figure 3 shows an overview of the generic Pleiades architecture. The ARM microprocessor is used primarily for sending configuration data to the various satellites to ensure a smooth flow of computations at the satellites. Additionally, the microprocessor is used for simple computations and control flows such as if-then-else statements.

المنارة للاستشارات
www.manaraa.com

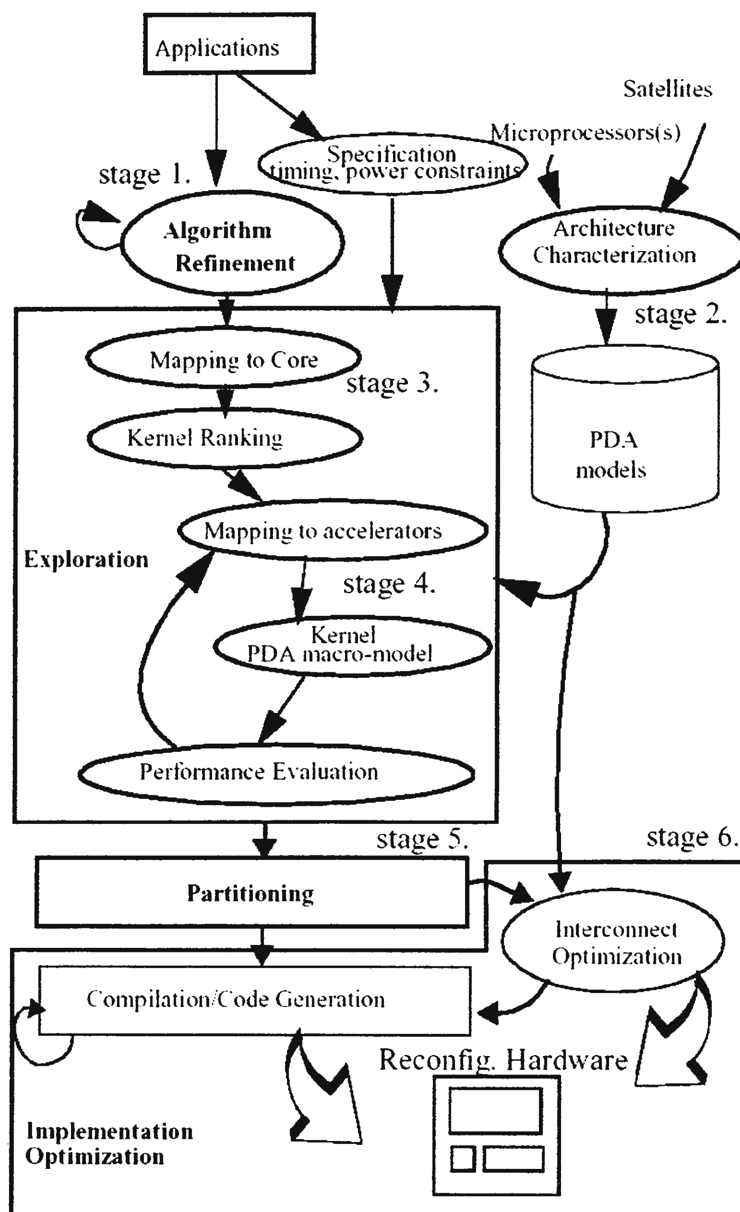

Figure 2. Overview of the Design and Software Generation Process [18].

www.manaraa.com

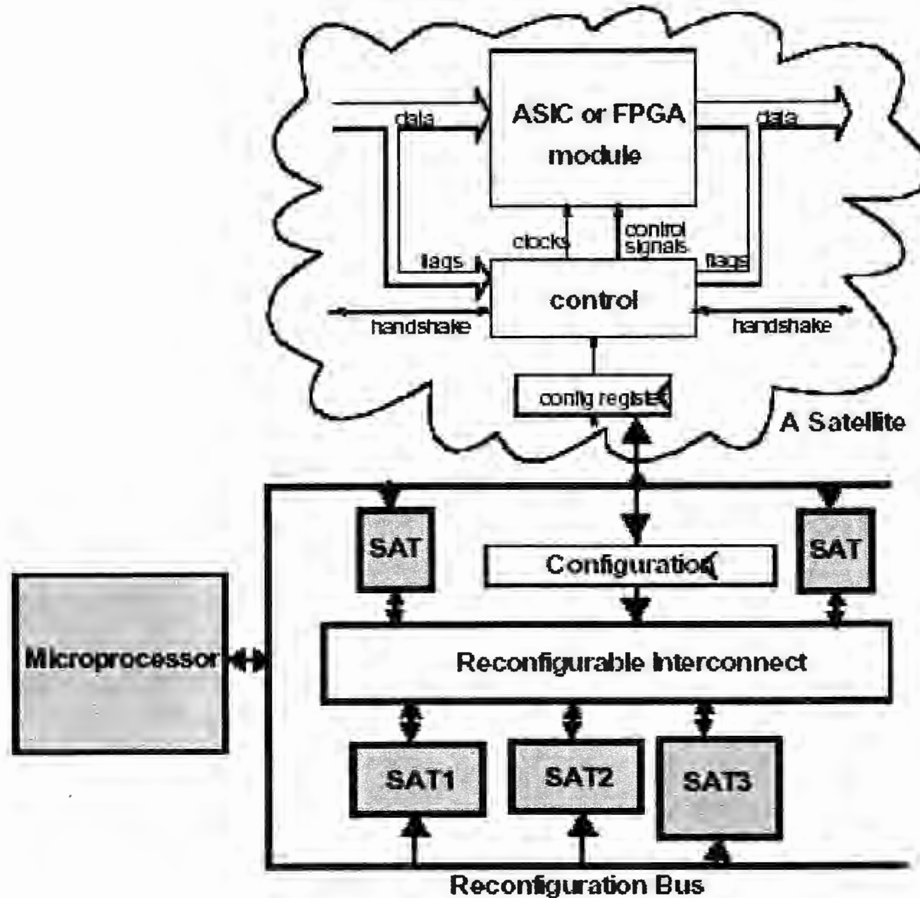

Figure 3. Overview of the Pleiades Architecture [18].

One specific instantiation of the Pleiades architecture that they discuss is the Maia architecture. The Maia architecture was designed to perform the different voice processing algorithms such as VSELP and VCELP [21]. Some of the computationally intensive kernels of these algorithms are dot product, FIR filter, IIR filter and covariance matrix computation. A diagram of the Maia architecture can be studied in figure 4.

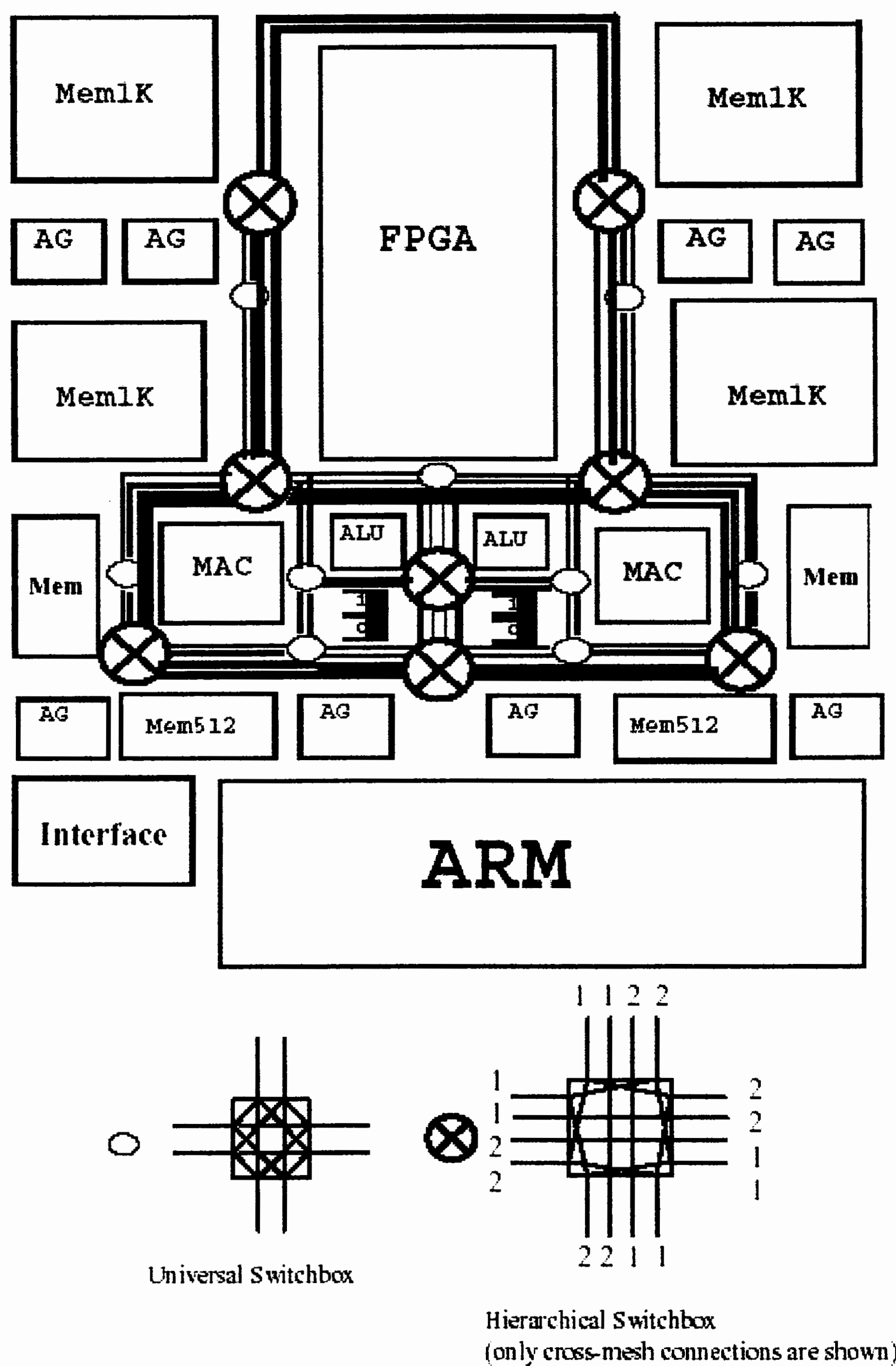

Figure 4. The Maia Reconfigurable DSP [19].

The Maia architecture is implemented in 0.25 micron technology and has been designed to operate at an average speed of 40 MHz. The chip is 5.2 mm x 6.7 mm and contains 1.2 million transistors. Maia runs on a main supply voltage of 1 V and consumes an average of 1.5 – 2.0 mW for VCELP processing [21].

المنارة للاستشارات
www.manaraa.com

CHAPTER 3. MATERIALS AND METHODS

As can be seen from the literature reviewed above, most reconfigurable architectures that have been proposed augment a GPP for performance improvement. Perhaps this is due to the fact that most DSPs already have many hardware features customized to their common computationally intensive needs. However, with the ever-increasing computational demands of multimedia and wireless devices, DSPs can benefit from the incorporation of reconfigurable hardware to improve performance and to provide more flexibility than dedicated hardware can provide. The use of cache with DSPs is a relatively new occurrence [4], however, new DSPs are reducing the I/O bottleneck by adding on-chip cache. The Texas Instruments TMS320C64x (C64x) is one family of DSPs that utilizes on-chip cache for improved performance. Like in [6], [7], portions of this cache could be used as a reconfigurable functional unit.

3.1 Current C64x Architecture

3.1.1 Architecture Overview

The current C64x is a fixed-point VLIW architecture that operates at 600 MHz. A VLIW architecture is designed to execute multiple instructions in parallel. However, unlike a superscalar processor, instructions are executed in-order and the determination of what instructions can execute at the same time is determined statically by the compiler prior to run-time. Branch prediction does not occur in a VLIW architecture either. Therefore, a VLIW architecture does not need additional complex logic control hardware, such as reorder buffers and branch prediction tables, that a superscalar processor requires. Rather, a VLIW architecture relies on a sophisticated compiler to enforce dependencies during compilation.

المنارة للاستشارات
www.manaraa.com

The C64x has two register files (*A* and *B*) that each contain 32, 32-bit registers. A modified Harvard architecture is used for the cache design, thus utilizing two separate level one (L1) caches. The L1 program cache is a 16KB direct-mapped cache. The L1 data cache is a 16KB 2-way set associative cache. A unified level two (L2) cache is also included on-chip. The L2 cache is 1024KB and can be configured to consist of either all SRAM or partial SRAM and partial 4-way set associative cache in various combinations. There are four 32-bit wide data buses connecting the register files to the L1 cache (two per register file). This enables two 64-bit wide data loads to occur simultaneously. There are some restraints on which load and store instructions can execute in parallel due to port limitations. For example, non-aligned loads and stores of words and double words are permitted, but cannot occur in parallel with any other non-aligned load/store instructions. The bus connecting the CPU to the instruction cache is 256 bits wide to enable eight 32-bit instructions to be fetched at once. The C64x has eight different functional units (FUs) that each connect to the register files. Two of the FUs (`.L1` and `.L2`) handle basic arithmetic operations. The S-units (`.S1` and `.S2`) handle branches in addition to basic arithmetic operations. The M-units (`.M1` and `.M2`) are dedicated to multiply and MAC operations. The final two units (`.D1` and `.D2`) are dedicated to address computations and load/store instructions. A cross-path allows any functional unit that directly connects to Register file *A* to receive and store data in Register file *B* and vice versa. Only two instructions executing in parallel can access data via the cross paths (one to *A* and one to *B*), but up to two instructions can share the operand received on the cross path. If the operand being read via the cross path was updated in the previous cycle a one-cycle delay will occur before the data is available. Figure 5 gives an overview of the C64x layout [42].

المنارة للاستشارات
www.manaraa.com

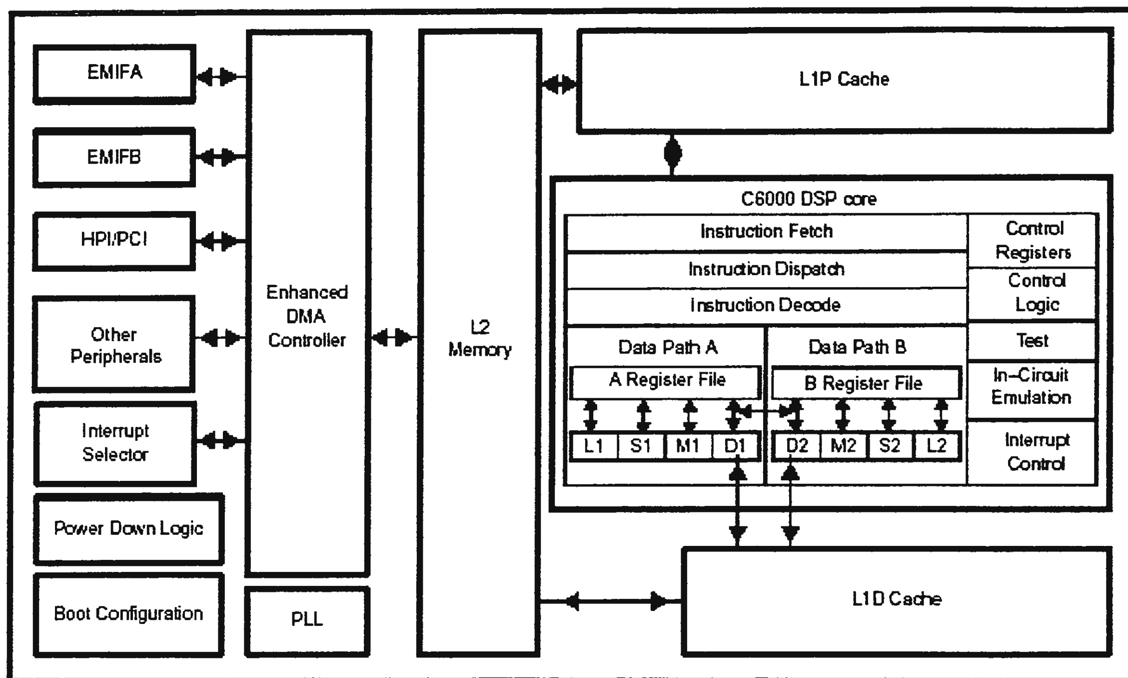

Figure 5. Block Diagram of TMS320C64x [42].

3.1.2 Instruction Set Architecture

The method by which each instruction moves through the pipeline is dictated by various fields in its binary instruction. Figure 6 gives an example of a C64x binary instruction (also referred to as the opcode by TI). All C64x instructions are 32-bits long, but the type and placement of fields within the instructions varies depending upon the instruction type and what functional unit it uses. For all instructions, the least significant bit (using Little Endian notation), denoted by "*p*", indicates that the instruction executes in parallel with the instruction preceding it if the bit is set to one. In assembly code the "||" symbol is used to indicate that the instruction executes in parallel. The next bit, denoted by "*s*", determines whether the *A* register file (*s*=0) or the *B* register file (*s*=1) is used to send operands to the functional unit. This bit is always in the same location for all instructions. The next group of

www.manaraa.com

bits specifies what functional unit to use for execution. The number of bits used to specify the functional unit varies, and while generally consistent for each unit type, it does even vary occasionally for the same unit. In this example, the next three bits are used and are set to 0x6 to denote the `.L` unit. The group of bits labeled "*op*", also known as the opfield, specify the operation performed, which for this example is signed and unsigned addition. The "*x*" bit indicates that one of the operands is obtained from the cross path if it is set to one. In assembly an "X" is listed after the function unit specification (ex: .D2X) to indicate that the cross path is used. The next five bits in this example, labeled "*src1/cst*", indicate the register number where the first operand is stored or, for some instructions the three bit constant value to be added to the other operand. Likewise, the next five bits, labeled "*src2*", indicate the number of the register that contains the second operand and the five bits labeled "*dst*" indicate what register the results will be stored in. Instructions that use a larger constant displacement use more bits. The remaining four most-significant bits are always used to indicate whether or not the instruction is conditional, and if so, what register to use to check the condition. If the "*z*" bit is set to one the instruction is conditional. Only registers *A0* – *A2* and *B0* – *B2* can be used as conditional registers (in TMS320C62x devices the *A0* is not a conditional register). Load and store instructions have slightly different fields. For specifics on individual instructions, please refer to [43].

The C64x is completely backwards compatible with the TMS320C62x. Thus, all instructions in the ISA of the C62x are also included in the C64x's ISA. In addition to the ISA of the C62x the C64x has well over 100 new instructions specific to its architecture. Additionally, the C64x adds a new control register for specifying some of the parameters necessary for

المنارة للاستشارات
www.manaraa.com

Galois Field multiplication. One of the new instructions, GMPY4, implements Galois Field multiplication, which is a common algorithm in Reed Solomon decoding [44].

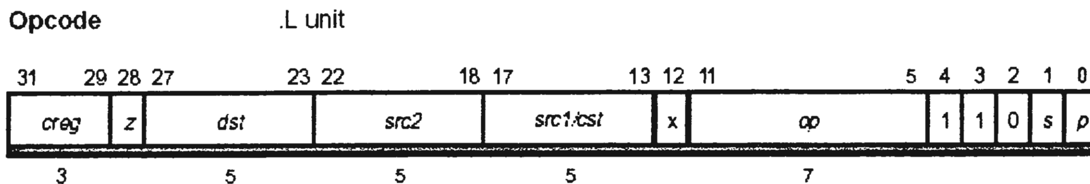

Figure 6. Example of the ADD(U) instruction layout [43].

3.1.3 Pipeline

The C64x takes advantage of an 11-stage pipeline in conjunction with the eight FUs to achieve speeds up to 4800 MIPS [43], [45]. The 11 stages are divided into three groups: Fetch, Decode and Execute. The Fetch group consists of the first four stages: Program Address Generate (PG), Program Address Send (PS), Program Access Ready Wait (PW) and Program Fetch Packet Receive (PR). The names of these stages are fairly self-explanatory. The program counter address is determined by the CPU in the PG stage and then sent to the instruction cache in the PS stage. The role of the PW stage is not as evident from its name. In this stage the instruction cache is read. If a miss occurs, then the appropriate block of instructions is fetched from the unified L2 cache, or from main memory if a miss also occurs in L2. In the PR stage the fetch packet arrives at the CPU. A total of eight instructions are fetched at a time in a fetch packet. The Decode group consists of stages Instruction Dispatch (DP) and Instruction Decode (DC). In the DP stage the fetch packet is broken into one or more execute packets. Execute packets consist of instructions that can be executed in parallel. In the C64x architecture an execute packet can be scheduled by the compiler to be

المنارة للاستشارات
www.manaraa.com

loaded in separate fetch packets. However, due to the fact that there are only eight functional units, an execute packet cannot be larger than eight instructions.

Instruction parallelism is determined statically by the compiler during compilation based on what instructions do not have data or resource conflicts. Due to the limited ability of any compiler to completely disambiguate memory references further parallelism can often be obtained by optimizing the code by hand. If the fetch packet contains more than one execute packet, then further fetches of instructions from the instruction cache are stalled until all of the execute packets in the DP stage have advanced through the pipeline in order. Figure 7 illustrates how fetch packets containing various numbers of execute packets progress through the pipeline. The CPU decodes the instructions within the execute packet in stage DC. All of the stages in the fetch group and the decode group are used by every instruction that moves through the pipeline. The first stage of the execute group, Execute 1 (E1), is also utilized by all instructions. The remaining four stages of the execute group, Execute 2 – 5 (E2, E3, E4 and E5), are only used by instructions that cannot complete execution in one cycle (ex: loads, stores and multiplication).

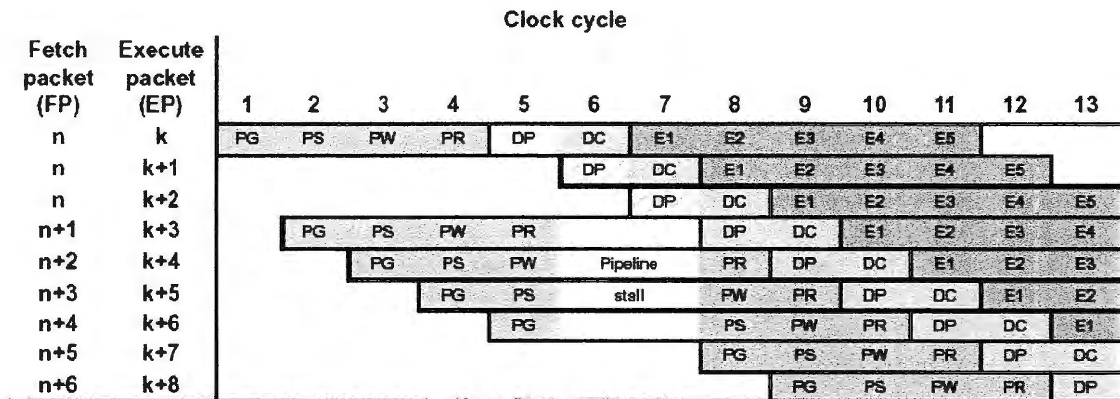

Figure 7. Illustration of Fetch Packets Progressing Through the Pipeline [43].

In stage E1 the instructions in the execute packets begin execution at the functional units indicated by the unit field in the binary instruction. The functional unit assignment, like the execute packet assignment, is also done by the compiler statically during compilation. Most instructions can be assigned to either of the functional units of a particular type. However, there are a few instructions that are specific to an individual FU and cannot be assigned to the same FU on the other side (ex: "Branch Using a Register" can only be assigned to .S2).

All branch instructions in the C64x Instruction Set Architecture (ISA) are unconditional, unless the instruction is predicated. All instructions in the ISA can be predicated for inherent "if-then-else" instruction handling. Branch instructions, although requiring only one cycle to determine the branch PC address, have a five-cycle delay slot before the branch is taken. Load instructions require five cycles to complete, with a four-cycle delay slot (not including potential cache delays) after the address generation in E1. Store instructions require three cycles to complete. A load instruction does not actually access memory until stage E3, thus a load instruction that follows a store to the same memory location does not have to wait any cycles for the data to be available. For this reason, TI states that store instructions do not have any delay slots. However, if a load and a store to the same memory address occur in parallel, the old data will be loaded and then the new data will be stored [43].

3.1.4 On-Chip Cache

3.1.4.1 L1 Program Cache

As stated above, the C64x has a level-one 16KB program cache that is direct-mapped. This cache is a read-only cache to prevent corruption of the program that is running. The cache consists of 32-byte wide blocks. Since a 32-byte block can hold 8 32-bit wide instructions,

المنارة للاستشارات
www.manaraa.com

this block size ensures that each fetch packet can cause at most one cache miss. There are 512 sets in the cache. Only one cycle is required to read a fetch packet from the L1 program cache if a hit occurs. Due to the fact that misses are pipelined, and that the amount of parallelism within the fetch packet can affect how soon some instructions in the packet will be needed, L1 misses can take anywhere from zero to seven cycles if the data is in the L2 cache. If the data is not in the L2 cache, then the data will have to be fetched from external memory. The amount of delay incurred due to an external memory access will vary depending upon what type of external memory the DSP is connected to.

3.1.4.2 L1 Data Cache

The 16KB 2-way set associative data cache is a read-allocate cache. In other words, data currently in the cache will only be evicted to make room for new data on a read miss. Write misses do not cause space allocation to occur. Rather, a write miss will store the data in a write buffer that exists between the L1 and L2 cache to reduce CPU delays. The L2 cache will then empty the write buffer as time permits. The write buffer can only hold up to four double words. If the L2 cache is mapped as SRAM then two single word adjacent stores can be merged to allow the buffer to not fill up as quickly. The L1 data cache is also a write-back cache so when a write hit occurs, the data will just be written in L1. The block that contains the newly written data will be marked as dirty and it will not be written back to L2 until that block is evicted. The L1 cache uses a least-recently-used (LRU) policy to determine what blocks are replaced when space needs to be allocated for read misses.

Memory banks that are 32 bits wide are used to organize the cache in a manner that helps to ensure that two parallel accesses will not conflict. If, however, the accesses are to the same

المنارة للاستشارات
www.manaraa.com

memory bank, are not to the same block within that bank and are greater than 16-bit wide accesses, a stall will occur. Due to miss pipelining in the L1 cache, a miss in L1 that hits in L2 will cause a delay in the range of two to eight cycles. The average delay is five cycles. As with the L1 program cache, if the L1 data cache request also misses in L2 the delay will depend upon the type of external memory implemented.

3.1.4.3 L2 Unified Cache

In the C64x the L2 unified cache consists of 1024KB, however, not all of that space can act as cache. The L2 can be configured in a variety of combinations to consist of either 1024KB of SRAM, or with part of the space acting as a 4-way set associative cache and the remaining space as SRAM. If part of the space is implemented as cache, its size can range from 32KB up to 128KB. Figure 8 depicts the different L2 configurations. The memory banks in the L2 are increased to 64 bits wide rather than the 32-bit wide banks in the L1. The L2 cache is a write-allocate cache, rather than read-allocate. This means that a write miss, not a read miss, will cause space to be allocated in the L2 cache for the new data. Rather, the L2 is a load-through cache, so read misses in L1 that also miss in L2 will cause the data to be loaded directly through the L2 to the L1 without being written in the L2.

The L2 also uses a LRU scheme for replacing data upon misses. If the level two memory is configured entirely as SRAM and a read miss occurs, the miss can cause two different responses. In addition to specifying configuration, the user can also specify whether or not the SRAM region is cacheable. If it is set up as cacheable, a miss will load an entire block of data and forward it on the appropriate L1 cache. If it is set up as non-cacheable, then just the specific piece of data that is needed is retrieved and forwarded to the appropriate L1 cache.

المنارة للاستشارات
www.manaraa.com

Neither way is the data stored in the SRAM space. For more information about the C64x caches please refer to [42].

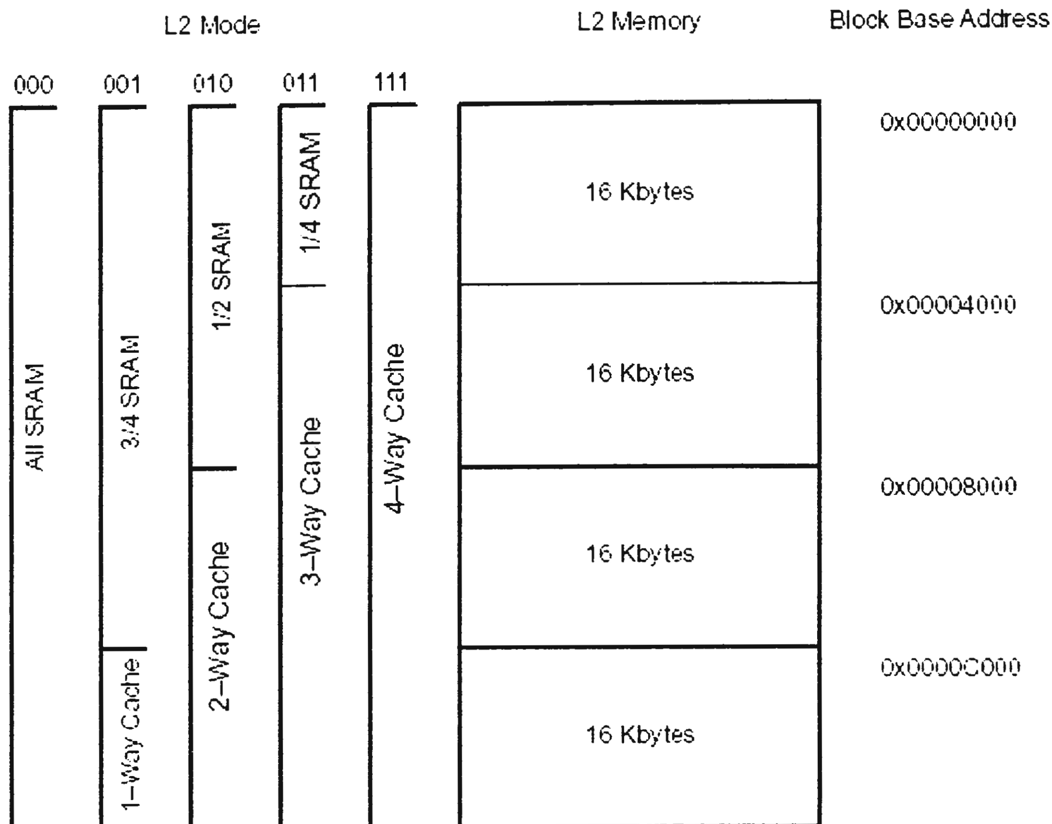

Figure 8. Level 2 SRAM/Cache Organizations [42].

3.1.5 Power Consumption

The power consumption of a processor fluctuates as the bit switching activity fluctuates during program execution. Due to this data-dependent fluctuation, it is impossible to give exact power consumption requirements for the C64x or any processor. However, typical power consumption values based on actual measurements are reported in [46]. The C64x operates at 600 MHz, is implemented in 0.12μm technology and requires 1.4 V to function

المنارة للاستشارات
www.manaraa.com

properly. I/O operates at the common external device voltage of 3.3 V. Table 1 lists the reported power consumption values for C64x. In [46] power is reported "per frequency" and as high/low activity models, so totals of these values are presented in Table 1. High activity is classified as eight instructions executing per cycle, with 32 bytes of data being fetched from L1 program cache and 16 bytes of data being fetched from L1 data cache. Additional specifications are given for the activity of the L2 and other peripherals. Low activity is classified as only two instructions executing per cycle, with 32 bytes of data being fetched from L1 program cache every four cycles and only 2 bytes of data being read from L1 data cache. [46] reports that most applications spend about 50-75% of their time in high activity and the remaining time in low activity. This table includes measured power consumption for the four different low-power modes that the C64x has: Idle, PD1, PD2, PD3. In the idle mode NOPs are continuously executed until an interrupt restarts activity. Modes PD1-PD3 shut down various peripherals and clocking of different components to reduce power consumption. Normal activity can be restored from PD1 with an interrupt, but PD2 and PD3 require a reset to restart activity. The power measurements for PD2 and PD3 in [46] disabled all external memory interface clocking as well to conserve more power. More information about the low-power modes can be obtained in [42].

Table 1. Measured Power Consumption Values for the C64x at 600 MHz, 1.4 V [46].

Power Per Frequency (mW/MHz)	**50/50 High/Low Activity (W)**	**75/25 High/Low Activity (W)**	**Low Power Modes (W)**			
CPU with L1 Caches	Total	Total	Idle	PD1	PD2	PD3
0.7	1.47	1.61	0.94	0.87	0.35	0.31

المنارة للاستشارات
www.manaraa.com

3.2 Reconfigurable Architecture

3.2.1 Architecture Overview

This research examines the use of a reconfigurable functional cache in the C64x to enhance performance while maintaining or reducing current design power consumption. In this research the overall architecture of the C64x is not modified extensively. Existing cache sizes are used for most simulations with the exception of one that uses a 32KB 2-way set-associative cache. This was done to see if the existing DSP could be enhanced with the reconfigurable functional cache (RFC) to improve performance without degrading other factors that affect performance such as the cache miss rate. If performance and power consumption measurements are the same or better with a 16KB 2-way set associative cache, then it is likely that even greater improvements would be seen in the future with larger caches. It was shown in [6] that cache access time for a reconfigurable cache actually decreases when compared to a memory cell array cache and increases only 1% when compared to a base array cache. Therefore, the reconfigurable cache does not significantly affect the cycle time and thus the frequency of the DSP is unchanged. Other aspects of the DSP such as voltage, number and size of registers in the register file, bus widths, etc. are also unchanged. A couple of reconfigurable implementations do require additional hardware such as dedicated adders or a divider, as well as input/output buffers. These will be discussed in more detail in section 3.2.4.

3.2.2 Simulator

The simulator is the embodiment of the instruction set architecture for the C64x in this work. The only comprehensive, freely available simulator for a TI VLIW processor was the one implemented by Vinodh Cuppu from the University of Maryland [22]. His simulator was of

www.manaraa.com

the TMS320C62x DSP and, because of the file names used, will be referred to as the c6000 simulator in this document. The authors of [47] have written a simulator of the C64x, but their version is a very stripped-down model with only about 20 instructions implemented. Further, their version reads in a text version of assembly code to determine what instructions to simulate, whereas the c6000 loads an actual binary executable generated by the TI compiler and is thus capable of simulating entire benchmark programs rather than just brief sections of code.

The c6000 is a functional simulator in that it actually loads data from memory, performs the operations on the appropriate operands from the appropriate registers and then stores the result to the appropriate register and on to memory when a store occurs. The c6000 fully implements the 11-stage pipeline and accurately determines stalls and cycle advancements. Therefore, the c6000 was chosen as the base simulator that was modified to create a new simulator, which will be referred to as the c6400 in this document. It should be noted that the c6000 does not handle interrupts (the actual DSP does), nor does it simulate peripheral devices, but otherwise is fairly comprehensive. The c6400 does not handle interrupts either. The only peripheral devices it simulates are the caches.

The c6400 is a mixture of a functional and a timing simulator that simulates the C64x, including its cache activity, and estimates its power consumption. The c6400 maintains the functionality of the c6000 simulator but only does timing simulations for the cache accesses. Primary modifications to the c6000 simulator include increasing the register file size to 32 registers per file, increasing bus widths to accommodate 64-bit wide data, increasing cache size, and incorporating the instructions into the simulator that the C64x has over and above

www.manaraa.com

the C62x. Since this research looks at the performance of reconfigurable cache and its power consumption, it is necessary for the simulator to simulate and report cache activity as well as estimates of power consumption of the various components of the DSP. A rudimentary, flat memory structure existed in the c6000, but there was no cache and no power estimation capabilities. These capabilities will be discussed next.

3.2.2.1 Cache Simulation

To implement cache in the c6400 the cache files "`cache.c`" and "`cache.h`" were incorporated from the SimpleScalar simulator [23]. In the SimpleScalar simulator the cache component operates as a timing simulation. In other words, it determines if a cache access would be a hit or a miss and thus how much latency is incurred by the access, but it does not actually load data from or store data to the cache. The documentation with the files states that it has the capability to handle data movement by setting one Boolean variable to true. However, difficulty was encountered in trying to incorporate the cache as a functional cache. Therefore, it was decided to continue to use the existing memory simulation for the actual movement of data during loads and stores and to simply use the cache simulation to determine if those accesses would be hits or misses. This way the cache simulation can still track important cache statistics such as miss rate but does not need to be burdened with the actual data movement. Once it was decided to use the cache simulation as a timing model only minor modifications such as variable names and declaration types had to be made to incorporate the cache files into the c6400 simulator.

The cache creation and miss-handling functions were implemented in the main file, "`c6000.c`" of the c6400 to maintain consistency with how cache creation and miss-

المنارة للاستشارات

www.manaraa.com

handling functions are instituted in [23]. Currently, rather than allowing the user to specify the cache types, sizes and associativity, the values are set within the code to reflect the current C64x cache sizes and types. In the future, if a user so desired, command-line specification of these variables could be instituted similar to [23].

3.2.2.2 Power Estimation

As mentioned previously, it is difficult to take accurate power measurements of a processor. It is even more difficult to try to predict the power consumption of a proposed design that has not been manufactured. Since the design and fabrication of new devices is a costly procedure many researchers have focused on ways to accurately estimate the power needs of an architecture that is still in the design stages. The primary means of estimating power consumption are to estimate power at the architecture-level, the behavior-level, the instruction-level or the system-level. At the architectural level analytical and empirical methods have been used. This method attempts to look at the activity switching of various registers and logic to estimate power. Behavior-level power estimation uses static and dynamic activity prediction to estimate power. Instruction-level power estimation was proposed in [48] and its use for embedded systems discussed in [49]. System-Level power estimation attempts to estimate power consumption for all components of the device to give a more comprehensive estimate. For an overview of these methods please see [50].

Various researchers have looked specifically at the power requirements of DSPs such as the Pleiades research referenced earlier. Another power prediction model for a DSP was proposed in [51]. While similar to the work in [48] and [49], [51] is different in that different straight line basic blocks are looped through several times to gain power measurements, and

المنارة للاستشارات
www.manaraa.com

then a linear regression model is created to predict future power consumption. Another research team looked specifically at power estimation for a VLIW DSP [52], but an actual simulator of these power estimations was not found. Therefore, portions of another simulator, Sim-Wattch [24] were included in the c6400 simulator to give power estimation capabilities. Sim-Wattch is built on top of the SimpleScalar simulator, and as such, estimates the power consumption of an out-of-order processor. An out-of-order processor requires complex structures such as a reorder buffer and complex controls for speculation and branch prediction. A VLIW processor does not require any of these complex structures or controls. In order to more closely reflect a VLIW processor the files "`power.c`" and "`power.h`" that were incorporated from [24] were modified to remove the unnecessary hardware structures and account for the appropriate size and number of data buses and registers. Sim-Wattch uses a modified version of the file "`time.c`" from the CACTI cache simulator [53] for estimating timing and capacitance measurements of different components of the simulator. Sim-Wattch can be scaled to better reflect the power needs of different process technologies. However, 0.12μm technology was not one of the technologies implemented. The scaling values for 0. 12μm technology were calculated by interpolating between the values for 0.10μm and 0.18μm technology.

Energy consumption is calculated by the equation:

$$E = P * T = I * Vdd * T \tag{1}$$

المنارة للاستشارات
www.manaraa.com

Where P stands for power, T represents time, I is the current passing through the device, and Vdd is the voltage required by the device. The derivative of this equation can be taken to determine the instantaneous power measurement. This is given by Equation 2.

$$P(t) = V(t) * I(t) \tag{2}$$

The modified version of Sim-Wattch used in the c6400 initially estimates the power requirements of the register files, the instruction cache, the data cache, the L2 cache, the functional units, the buses that connect the functional units to the registers and the clock. It then scales these values each cycle based on the activity measured in the data that is transferred between the functional units and the registers and on how many times each component was accessed during that cycle. Running totals are kept for each component. The summation of all of these component totals except the clock power represents the energy consumed during the simulation of the program being executed by the simulator. This is shown in Equation 3. An average of the overall total is taken to estimate the average power consumed each cycle and is shown in Equation 4.

$$Total\ Energy = Register + Icache + Dcache + L2cache + FU_power + Resultwires \tag{3}$$

$$Average\ Total\ Power\ per\ cycle = Total\ Energy \div total\ cycles \tag{4}$$

Sim-Wattch also has three different conditional clocking levels that energy and power are estimated for. The first, referred to as "cc1" in the code is a basic, non-aggressive conditional clocking. The second (cc2), is an aggressive conditional clocking that assumes

المنارة للاستشارات
www.manaraa.com

zero power is consumed by components that are disabled that cycle. This is an idealistic clocking model as generally some residual power loss will occur even when a component is disabled. The third clocking model (cc3) accounts for this residual loss and is perhaps a more realistic aggressive conditional clocking model.

Power measurements are fundamentally the same for the reconfigurable architecture as for the base architecture. For both the normal and reconfigurable cache the data caches are assumed to be segmented, base-array caches. Documentation was not found to support this assumption of the current implementation of cache in the C64x, but, results can be modified to reflect a memory-cell only array if desired. There are two major differences for the reconfigurable DPS. First, cache computing functionality is gained by implementing 4-bit input look-up tables (4-LUTs) within the data array of an 8KB cache module. Each 4-LUT requires a four-bit decoder. Please see Figure 9 for a diagram of a reconfigurable cache. When the reconfigurable cache is in computing mode only the four-bit decoders are accessed, not the main decoder for the data array. To account for this power the power used by the data array decoder is scaled to estimate the power for a four-bit decoder. This value is then scaled by the number of LUTs in the reconfigurable module. Most functional units and main registers are not accessed while the DSP is in reconfigurable mode, so their totals do not increase during RFC cycles. The only existing functional units that are routinely accessed while in reconfigurable mode are the `.D1` and `.D2` units. These are used during reconfigurable cache configuration and data input load/stores for address calculations. Second, when dedicated adders and/or dividers are used by a reconfigurable layout power estimates for these components are included in the total. The estimates for these components use the values that are used for the `.L` units and the `.M` units respectively and assume a

المنارة للاستشارات
www.manaraa.com

switching activity of 0.5 for a worst-case estimate. While the addition of these dedicated devices dictates that bitlines within the cache must be stretched, [6] determined that the increased capacitance on these stretched bitlines was negligible and thus is ignored here in power estimates. For a more detailed description of the cache at a transistor level, please refer to [53] or [6].

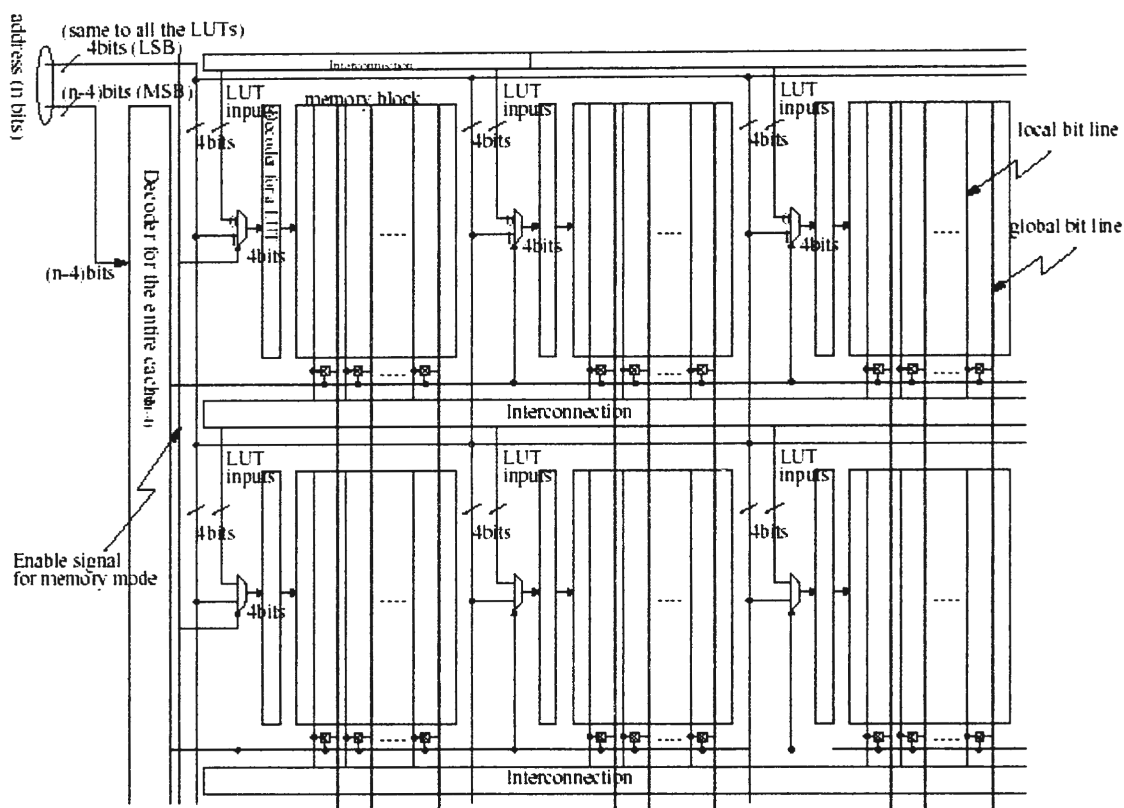

Figure 9. Diagram of a Reconfigurable Cache [7].

The modifications to the Sim-Wattch power simulation have not been verified against actual measured power consumption of a VLIW processor. As stated in [50], when a designer is considering optimizations of a potential new architecture, relative power estimations are

المنارة للاستشارات
www.manaraa.com

more important than accurate, actual measurements. The power estimates in the c6400 allow relative power comparison to see how the reconfigurable modifications affect the power consumption of the C64x DSP. This is sufficient for this research.

3.2.3 Reconfigurable Functional Cache

To add reconfigurable cache capabilities to the c6400 the cache file "`cache.c`" was further modified to handle hit/miss estimations for the RFC. When the simulator is in RFC mode way 0 of the 2-way cache is tagged as the reconfigurable module. This means that only way 1 is available for normal cache operations. This effectively reduces the data cache size to 8KB from 16KB. When way 0 is specified as the RFC module the only allowed accesses to way 0 are the loading of configuration data and table look-ups. Input and output buffers are added to way 0 to allow for buffering of input data and output data during RFC computing.

The "`c6000.c`" file was modified to detect the entry into RFC mode and the exit from RFC mode. These modifications are based upon the methods used by [6] to implement RFC but are not identical. For instance, since the binaries that the c6400 uses are generated by a proprietary compiler, new instructions to handle the configuration of the LUTs in the RFC could not be added to the ISA. To compensate for this limitation the user must specify the PC value of the first instruction to the kernel that is implemented in RFC and the PC value of the exit point from this kernel. These values are easily obtained by viewing the disassembly code in the TI Code Composer Studio Integrated Development Environment. The simulator then watches for these values. When the entry PC value is encountered the simulator sets a global Boolean variable "RFC_mode" to true.

المنارة للاستشارات
www.manaraa.com

When RFC_mode is true the simulator will continue to execute the instructions that pass through the pipeline in order obtain the results of these instructions for later use. However, cache accesses are not made for these instructions, the cycle counter does not advance and power updates are not made for these cycles. This is done to limit the impact of these instructions as much as possible on the device. If the RFC cache were truly implemented in the device, these instructions would be replaced by RFC instructions, and results of data calculations would be obtained from the RFC. For simplicity, the RFC simulation handles only the cycle advancement aspects of the RFC to determine the number of cycles required for the RFC to configure, load data, compute and store the results back to memory.

Upon initially entering RFC mode the appropriate kernel function in "`RFC_funcs.c`" is called. The functions within this file step through the stages needed to use a cache module as RFC to determine the cycles required as well as incrementing access counters when appropriate for instruction and data cache and then call the appropriate power function to calculate power updates each cycle. Each RFC function begins by simulating the loading of configuration data. To simulate the configuration loads cache accesses are made to both the instruction cache and data cache to determine cache hit/miss statistics for these loads. Way 0 is not initially flushed, but rather dirty lines that are replaced during configuration loading are simply handled as they normally would be. If the user wishes to completely flush way 0 before configuration, this can be done without incurring additional CPU stalls due to C64x's ability to hide these writebacks within the pipeline [42].

Once the configuration is loaded, then input data is loaded to begin computing. Depending upon the kernel, this usually involves loading a block of inputs with look-ups not

المنارة للاستشارات
www.manaraa.com

commencing until a complete block of data is present in the input buffer. Output from the look-ups is stored in the output buffer to await further processing or storage back to memory. Once all input has been loaded and processed results are stored back to memory. If the implementation requires reconfiguration of the LUTs before processing can complete, then computing is delayed until reconfiguring completes. While the current block is processing the next block of data can be loaded into the input buffer. This allows subsequent data loads to be hidden so additional delays are not incurred. To ensure timely reconfiguration and data loading, these values can be locked into a portion of L2 that is acting as SRAM.

3.2.4 Benchmarks

The benchmarks and kernels used to measure the performance of the reconfigurable DSP and existing C64x DSP were from the University of Toronto's DSP benchmark suite (UTDSP). However, these benchmarks and kernels originally used floating point variables. With look-up table computations it is easier to use integer values. Fractional values can be handled if distributed arithmetic is used [55], but for simplicity these variables and their inputs were modified to whole number types. The benchmarks chosen for simulation were `Compress`, `Edge Detect`, and `Spectral`. `Compress` is an image compression program that uses DCT. The DCT processing occurs 256 times in Compress and accounts for approximately 45% of the cycles needed to complete the benchmark. Amdahl's Law shows that optimization efforts should be focused on the portions of an architecture or code that account for a large fraction of the overall computing time for the effects of these optimizations to be greatest. Therefore, DCT in `Compress` is an excellent candidate for RFC. `Edge Detect` is an image processing program that uses 2-D convolution as part of the edge detection process. Convolution is repeated three times in `Edge Detect` and accounts for

المنارة للاستشارات
www.manaraa.com

approximately 13% of the cycles needed to complete `Edge Detect`. This percentage does not make convolution in `Edge Detect` as good a candidate for RFC as DCT, but it is large enough that speedups can still be gained with RFC implementation. The third benchmark, `Spectral`, does a spectral estimation on an input speech signal using a Fast Fourier Transform (FFT). FFT is called 16 times during `Spectral` and consumes about 20% of the total benchmark cycles, making it another good candidate for RFC.

UTDSP also contains several kernel programs that execute common digital signal processing filters like FIR, IIR, and normalized lattice filters. The only additional code within these kernel programs handles I/O of the data. Kernels do not give a true picture of the performance of an architecture in the real world because seldom will one algorithm run by itself. However, in digital signal processing these filters are important and thus DSP architectures are generally tailored to them. Therefore, a 32-tap FIR filter and a 256-tap FIR filter with inputs ranging from 256 – 4096 were implemented in RFC.

3.2.5 Kernel Implementations in RFC

The 4-LUTs used in these configurations are 16-bits wide in order to better fit within existing cache designs. In most of the configurations though, not all of these bits are actually necessary for the computation that is occurring. The different structures implemented with the 4-LUTs are 8x8 and 8x16 constant coefficient multipliers, various sized adders, adder/subtracters, accumulators, multiplexers and 16-bit ROM. For each kernel implementation, a variety of layouts for the LUTs may be possible. With the exception of DCT, only one layout is presented per benchmark/kernel.

المنارة للاستشارات
www.manaraa.com

3.2.5.1 DCT

An 8x8 2-D DCT with 8-bit inputs and 16-bit coefficients was implemented as 2 1-D DCTs in RFC. This implementation was based on the DCT design in [56]. The basic equation for an 8x8 2-D DCT is:

$$XC_{pq} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} XN_{mn} \cdot \frac{c(p)c(q)}{4} \cdot \cos\frac{\pi(2m+1)p}{2M} \cdot \cos\frac{\pi(2n+1)q}{2N} \quad (5)$$

Where M is the total number of rows and N is the total number of columns. To break this into 2 1-D transforms 1-D for the rows is calculated and then 1-D for the columns. The coefficient equation for 1-D of the rows is:

$$C = K \bullet \cos\frac{(2 \cdot column\ number + 1) \bullet row\ number \bullet \pi}{2 \bullet M} \quad (6)$$

For Equation 6 $K = \sqrt{1}/N$ for row 0 and $\sqrt{2}/N$ for all other rows. The coefficient equation for the 1-D of the columns is:

$$C' = K \bullet \cos\frac{(2 \cdot row\ number + 1) \bullet column\ number \bullet \pi}{2 \bullet N} \quad (7)$$

For the column DCT $K = \sqrt{1}/M$ for column 0 and $\sqrt{2}/M$ for all other columns. Due to the symmetry of the coefficients a 1-D transform can be completed with only 32 coefficients although a total of 64 coefficients are generated. Figure 10 shows the basic organization of

المنارة للاستشارات
www.manaraa.com

the 1-D row transform. In the diagram *K* ranges from zero to seven. The diagram for the column transform is similar, so it is not shown.

To implement a 1-D transform in 4-LUTs 8-bit adder/subtracters, 8x16 constant coefficient multipliers, and 24-bit adders are needed. How to create a 4x8 constant coefficient unsigned and signed multiplier with two 4-LUTs that have 12 bit outputs is explained in [57]. The signed version is shown in Figure 11. This idea can be expanded to create an 8x16 constant coefficient multiplier out of four 4-LUTs that have 16-bit outputs. In this case, the four most significant bits of each LUT are not used. The primary drawback of this design is that not all of the 32 constant coefficient multipliers that are needed can be implemented in an 8KB cache module. Each eight inputs of a row are multiplied with the same eight coefficients and then the results are added together to generate a total. Since the inputs from two rows are combined before multiplication this can occur in parallel for all eight rows if there are 32 constant coefficient multipliers. If not, then reconfiguration will need to occur during each 1-D DCT. Four different implementations of DCT were designed to explore the tradeoff between using a larger cache module to avoid reconfiguration, using dedicated hardware to avoid reconfiguration and using a layout that requires implementation. Three of the implementations are discussed next. The fourth implementation, which uses distributed arithmetic, is based on the implementation of DCT in [6] and [7]. For details of the fourth implementation please see [6] or [7].

المنارة للاستشارات

www.manaraa.com

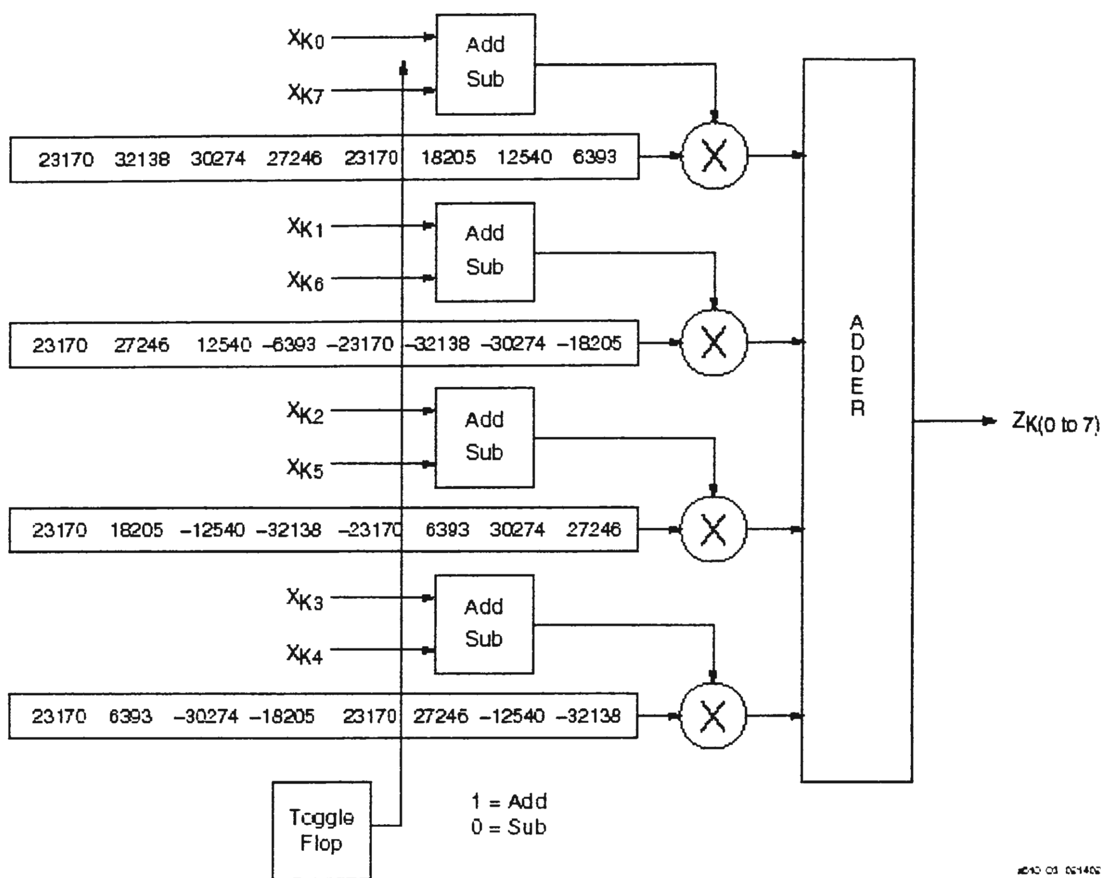

Figure 10. 1-D DCT for the rows of an 8x8 Block [56].

المنارة للاستشارات

www.manaraa.com

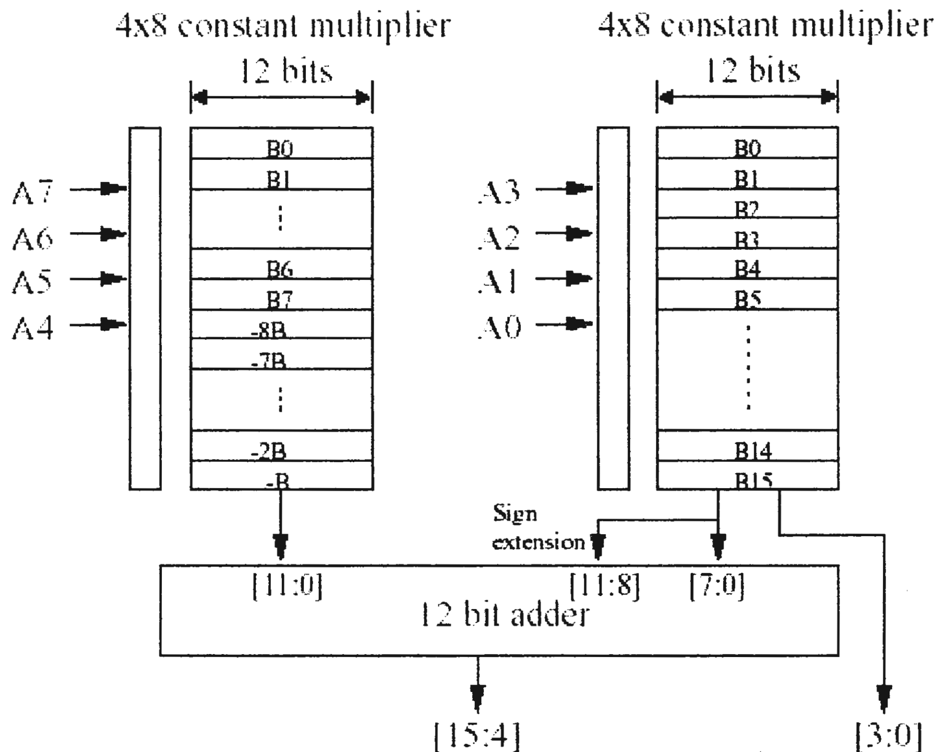

Figure 11. Design of a signed 8x8 Constant Coefficient Multiplier using 4-LUTs [57].

3.2.5.1.1 1-D DCT with 16 Constant Coefficient Multipliers

The first implementation of the DCT used an 8KB cache with no dedicated hardware. In an 8KB cache only 16 coefficients can be configured in LUTs at a time. This requires four LUT rows, leaving four more LUT rows for the adder/subtracters and 24-bit adders. Four 8-bit adder/subtracters and two 24-bit adders can be implemented in each of the remaining four rows. This configuration allows four rows to complete 1-D of the DCT before reconfiguration occurs. After reconfiguration the remaining four rows are able complete. Each 8x16 constant coefficient multiplier requires at least one 24-bit adder to sum its partial products. Due to the constraint on the number of 24-bit adders available, only two coefficients can complete at a time and three look-ups are needed to complete the summation

المنارة للاستشارات
www.manaraa.com

for each. Table 2 details the number of look-up requirements for each LUT structure. One LUT look-up can occur per cycle, so these numbers translate directly to the number of cycles required for that structure. The total cycles required for the DCT depends on how many of the look-ups can occur in parallel. This implementation requires a total of 11 cycles to complete one input each from four rows. Figure 12 gives a layout of 2 rows of LUTs. Three more replicas of these rows complete the entire 8KB configuration.

Table 2. Look-Ups Required for LUT structures.

1-D DCT Implementation Using an 8KB Cache Module	
Structure	**Look-ups Required**
8-bit adder/subtracter	1
8x16 constant coefficient multiplier	4
24-bit Accumulator	2

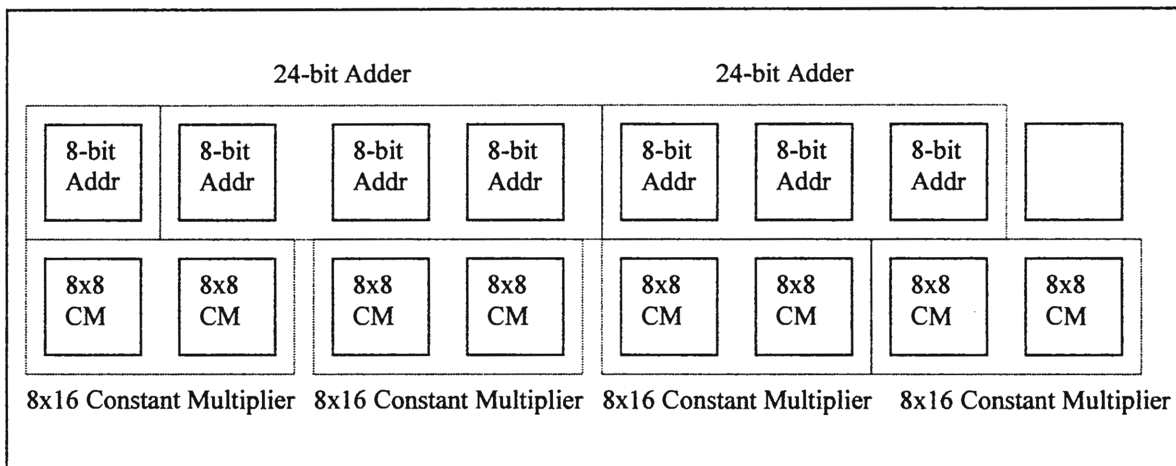

Figure 12. Layout of 4-LUTs for Two DCT coefficients. 1 Box = 4, 4-LUTs.

3.2.5.1.2 1-D DCT with 32 Coefficients and Dedicated Hardware

The second implementation of 1-D DCT in RFC implements the partial product LUTs of all 32 constant coefficient multipliers using 4-LUTs in an 8KB cache module. One dedicated 8-

www.manaraa.com

bit adder/subtracter is added for each row to do the initial addition/subtraction of the two inputs. Additionally, eight dedicated 24-bit adders are placed between each row of LUTs (one adder for each constant coefficient multiplier). This implementation avoids reconfiguration costs at the expense of adding dedicated hardware. The look-ups required for each structure are given in Table 3. The advantages of this implementation are that eight constant coefficient multiplication and accumulations can occur in parallel rather than just two as in the previous implementation. A diagram of the LUTs for this layout is not given as it is basic.

Table 3. Look-Ups/Cycles Required for LUT and Dedicated Hardware Structures.

1-D DCT Implementation Using an 8KB Cache Module, 32 Coefficients, and Dedicated Hardware	
Structure	**Look-ups/Cycles Required**
8-bit adder/subtracter (dedicated hardware)	1 (cycle)
8x16 constant coefficient multiplier	1 (LUT) + 3 (cycles)
24-bit Accumulator (dedicated hardware)	1 (cycle)

3.2.5.1.3 1-D DCT with a 16KB Module

This layout is similar to the first layout, but the total cache size is increased to 32KB to allow a 16KB reconfigurable module to be used. The doubling in size allows an 8-bit adder/subtracter and four 24-bit adders to be placed in one row and eight constant coefficient partial product lookups in the next row. These two rows are repeated three more times to give the entire configuration of the 16KB module. The advantage of this layout is that an entire 1-D DCT can be completed without reconfiguration and without the cost of additional dedicated hardware. The tradeoff is, of course, that more area is required on-chip to implement a 32KB 2-way cache. The number of look-ups required for each structure is the

المنارة للاستشارات
www.manaraa.com

same as the first implementation, but cycles are saved because four coefficients per row can complete their look-ups in parallel and reconfiguration is not required. Again, a diagram is not shown as the layout is the same as the first, only the number of 24-bit adders and coefficients are doubled.

3.2.5.2 Convolution

The convolution kernel implemented in `Edge Detect` multiplies a 128x128 matrix with 16-bit inputs of an image with a 3x3 filter matrix of 8-bit coefficients. Initially, the absolute values of the filter matrix are summed to determine a normalization value. Then, each value in the image matrix is multiplied with each value of the filter matrix and the results are accumulated. The final sum of each of these accumulations is divided by the normalization value to give the output image value. The equation for the matrix multiplication not including normalization is:

$$sum = \sum_{j=0}^{K=1} \sum_{j=0}^{K-1} input_image[r+i][c+j] * filter[i][j] \tag{8}$$

Where r and c range from zero to N-K, N=128 and K=3. This equation can easily be modified to execute more quickly by simple software loop unrolling to allow nine MACs to occur in parallel. The reconfigurable version of convolution takes advantage of this fact by implementing the nine filter coefficients as 16x8 bit constant coefficient multipliers along with one 24-bit adder for each multiplier. Therefore, an input buffer large enough to hold nine two-byte inputs is necessary for this RFC implementation. The normalization is done once at the beginning with a 16-bit adder/subtracter and a multiplexer implemented in a 4-LUT. The next stage depends upon nine input values being available simultaneously for

المنارة للاستشارات
www.manaraa.com

processing. Once the input buffer is full the MAC stage begins. The nine values obtained from the multipliers are added pair-wise using five 24-bit adders to reduce the number of consecutive look-ups required for accumulation. Results are stored in the output buffer until execution is complete. As with DCT, additional data loads can occur while the LUT look-ups are occurring, thereby hiding the latency of these loads. When execution has completed for all inputs the results are stored back to memory. Table 4 lists the number of look-ups needed for the structures in convolution and Figure 13 shows the layout of the LUTs within the 8KB cache module.

Table 4. Look-Ups Required for LUT structures in Convolution.

Convolution Implementation Using an 8KB Cache Module	
Structure	**Look-ups Required**
16-bit adder/subtracter (sums 9 values)	9
Two input multiplexer	1
16x8 constant coefficient multiplier	4
24-bit Accumulator	4

المنارة للاستشارات
www.manaraa.com

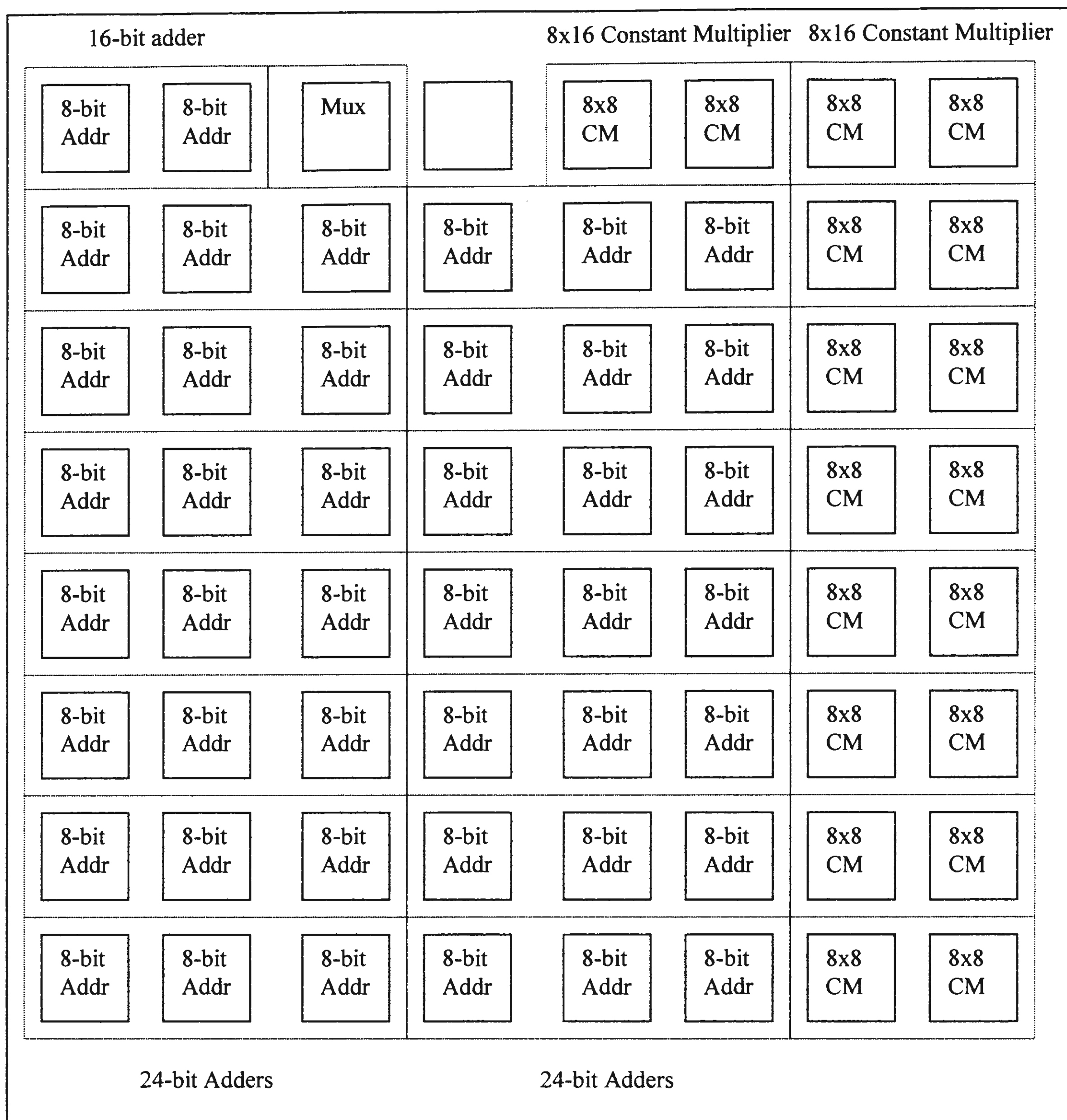

Figure 13. Layout of LUTs for Convolution. 1 Box = 4, 4-LUTs. Mux only uses ½ box.

3.2.5.3 FFT

A mathematical understanding of FFT can be gained by looking at Equation 9. This equation is from [58], which gives a good explanation of FFT. Most software implementations of FFT

المنارة للاستشارات
www.manaraa.com

do not follow this equation closely though, due to the fact that it has a complexity of $O(n^2)$. Rather, most software implementations strive to achieve a complexity of $O(n*\log_2(n))$. In this equation n is the number of inputs and the outputs. A restriction is placed on n that it must be a power of two. The variables x_k and y_p are the inputs and the outputs. The subscripts, which can range from zero to n-1, denote which complex-valued input/output is being used that iteration. The inputs are in the time domain and outputs are transformed to the frequency domain.

$$y_p = \sum_{k=0}^{n-1} x_k \left(\cos\left(2\pi \frac{kp}{n}\right) + i \sin\left(2\pi \frac{kp}{n}\right)\right) \tag{9}$$

The Fast Fourier Transform that is used by the `Spectral` benchmark was implemented in RFC using 8-bit inputs and 16-bit constant twiddle values. The benchmark code computes the twiddle factors each time FFT is called, but these values are not dependent on the input values, so the RFC assumes the twiddle factors have been pre-computed and stored with the LUT configuration data. Unlike the other implementations, there is no easy division of how many inputs need to be present before look-ups begin. Therefore, an input buffer large enough to hold two arrays of 64 one-byte inputs each is used and all data loads must occur before processing begins. This slows the FFT implementation down due to the fact that some of the load delays cannot be hidden in the LUT cycles. The FFT code has several statements that assign the value of one variable to another variable. This is handled in the LUTs by using an adder with one input fixed to zero and the other input being the variable whose value is being assigned to the other variable. This could certainly be handled other ways, but this method was chosen to give a logical flow to the data movement. Another difference

المنارة للاستشارات
www.manaraa.com

between the FFT RFC implementation and the others is the use of four 16x16 ROMs. These are similar to the 16x16 ROM used in [6] and [7] for the DCT RFC. There are a total of 32 real twiddle factors and 32 imaginary twiddle factors. Sixty-four constant coefficients were too many to implement in multipliers like the others. Therefore, two 16x16 ROMs are used for real and two for imaginary to store pre-computed partial products. Table 5 lists the cycles required for the different structures used in the RFC to implement FFT. Figure 14 details the layout of the LUTs in an 8KB cache. The final stage of FFT performs a bit-reversal on all of the real and imaginary inputs. The C64x has an instruction that handles this already—BITR. The BITR instruction requires two cycles to complete. Therefore, for the last stage the inputs are moved to registers, the bit reversal is performed on each input and then the results are stored back to the output buffer.

Table 5. Look-Ups Required for LUT structures in FFT.

FFT Implementation Using an 8KB Cache Module	
Structure	**Look-ups Required**
8-bit adders and 8-bit subtracters	1
16x16 ROM	1
24-bit adder for partial products	2

www.manaraa.com

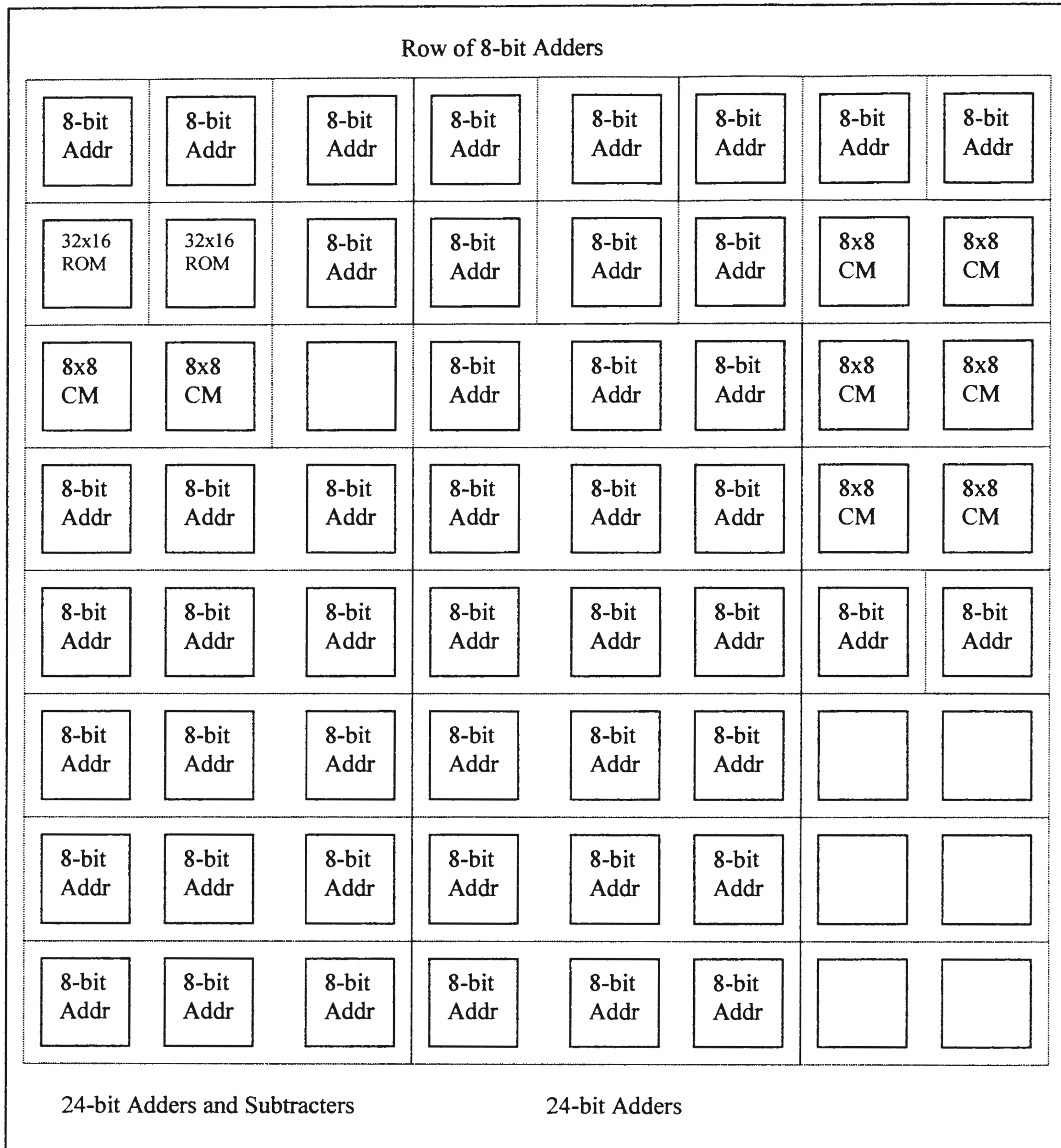

Figure 14. Layout of LUTs for FFT. 1 Box = 4, 4-LUTs.

www.manaraa.com

3.2.5.4 32-Tap and 256-Tap FIR

The implementation of the 32-tap FIR and 256-tap FIR filters differ only in how many times reconfiguration needs to occur. Therefore, they are both discussed together in this section. The equation for FIR is shown in Equation 10.

$$y(n) = \sum_{m-0}^{M} b_m x(n-m) \tag{10}$$

An FIR filter is simply a convolution (MAC) operation. This implementation uses 8-bit inputs and 8-bit coefficients. The types of LUT structures used are identical to [6] and [7] but they are arranged differently within the 8KB cache module. The layout differences are necessary since the C64x cache module is 32 bytes wide rather than 16 bytes wide as in [6] and [7]. Look-up accesses are given in Table 6 and the layout of one row in Figure 15.

Table 6. Look-Ups Required for LUT structures in FIR.

FIR Implementation Using an 8KB Cache Module	
Structure	**Look-ups Required**
8x8 constant coefficient multipliers	2
24-bit accumulator	4

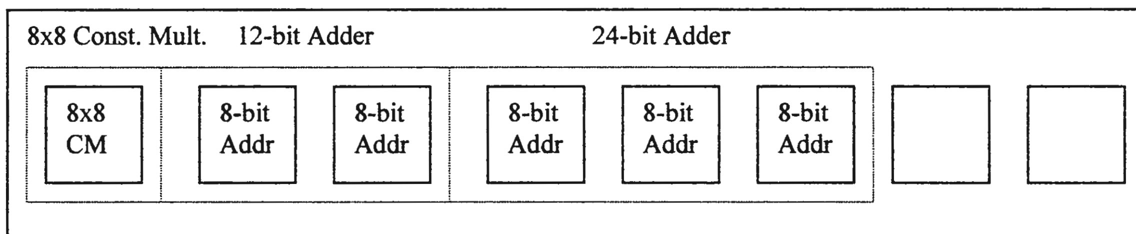

Figure 15. Layout of One Row of LUTs for FIR. 1 Box = 4, 4-LUTs. 12-Bit Adder only uses 6 LUTs total.

www.manaraa.com

CHAPTER 4. RESULTS AND DISCUSSION

The TI compiler can compile code with either no optimizations or one of four different levels of optimization. Each level of optimization builds on the previous level, so optimizations at one level are expanded upon at the next level. For a detailed description of what each compiler optimization level does, please see [59]. Briefly, though, level –o0 optimizes register usage, -o1 optimizes the code locally, -o2 optimizes the code globally and –o3 performs file optimizations. The benchmarks and kernels were compiled with the target processor being the C64x but with the libraries for the C62x included for all except the FIR kernels, which use the C64x libraries. This was done due to compilation difficulties. Often the code that was generated when the C64x libraries were included seemed to have infinite loops in them and would not process correctly on the c6400 simulator. Again, some difficulties were encountered with different optimization levels. Code using the top-most optimization would not work for any of the benchmarks or kernels. All benchmarks and kernels were implemented with no optimizations as a baseline and then with at least one level of optimization. Convolution and FFT used optimizations –o0 through –o2, DCT only used optimization –o1 and FIR used optimization –o2. Results are given for performance, power consumption, energy requirements and L1 data cache miss rates. Performance is measured by cycles required to complete execution rather than execution time as the time will vary depending upon the computer the simulator is running on, but the cycles required will not.

المنارة للاستشارات
www.manaraa.com

4.1 DCT

4.1.1 Performance

The results of the four different DCT implementations and the two different compiler code generations are presented in Figure 16. Results are presented as speedup over the corresponding non-reconfigurable simulations (cycles required by Non-RFC divided by cycles required for RFC). All of the implementations afforded the DCT kernel quite a bit of speedup, with speedups ranging from 128X to 359X (average speedup was 234X). The implementations that required fewer reconfigurations performed the best, as was expected. With most any implementation there is a tradeoff between performance and area. If the minimization of area is more important to the target design environment then the 8KB module that implements 16 coefficients with reconfiguration would be the best choice. If performance is more important then the implementation that uses distributed arithmetic would be the best choice. All of the optimized versions performed better than the unoptimized versions. While the portion of RFC code that performs look-ups is unaffected by the compiler optimizations, the configuration and data loads are affected by cache misses. Therefore, the improved performance for the optimized versions is most likely due to a lower L1 data cache miss rate than the unoptimized RFC versions. It would have been interesting to see how the higher optimizations affected the performance of the RFC kernel, but, unfortunately, the code generated by the compiler at higher optimizations would not work on the simulator.

المنارة للاستشارات
www.manaraa.com

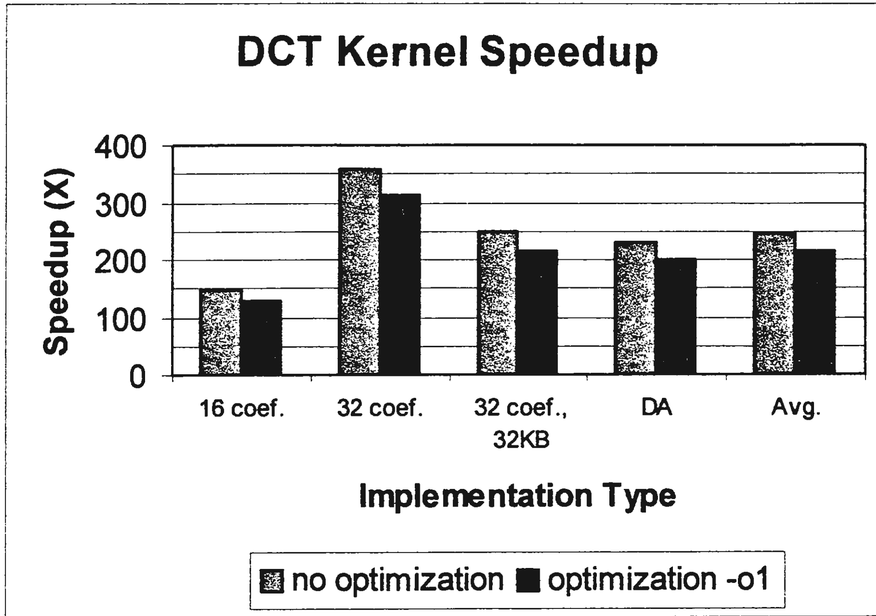

Figure 16. RFC DCT Kernel Speedups over Non-RFC DCT Kernels.

While the kernels achieved a lot of speedup, the overall benchmark speedups were not as impressive, but are still quite good, as expected by Amdahl's Law. These speedups are listed in Table 7 and shown in Figure 17. Due to the fact that the RFC has no control over the I/O functions, which also consume a large fraction of the total cycles, overall the decrease in total cycles observed for the benchmark is not as great.

Table 7. Overall Benchmark Speedups for Compress.

Overall Compress Benchmark Speedups (Total Non-RFC Benchmark cycles/Total RFC Benchmark Cycles)	
Implementation Type	**Speedup**
16 coefficients, 16KB, no opt.	1.91
16 coefficients, 16KB, -o1 opt.	1.80
32 coefficients, 16KB, no opt.	1.91
32 coefficients, 16KB, -o1 opt.	1.81
32 coefficients, 32KB, no opt.	1.91
32 coefficients, 32KB, -o1 opt.	1.81

www.manaraa.com

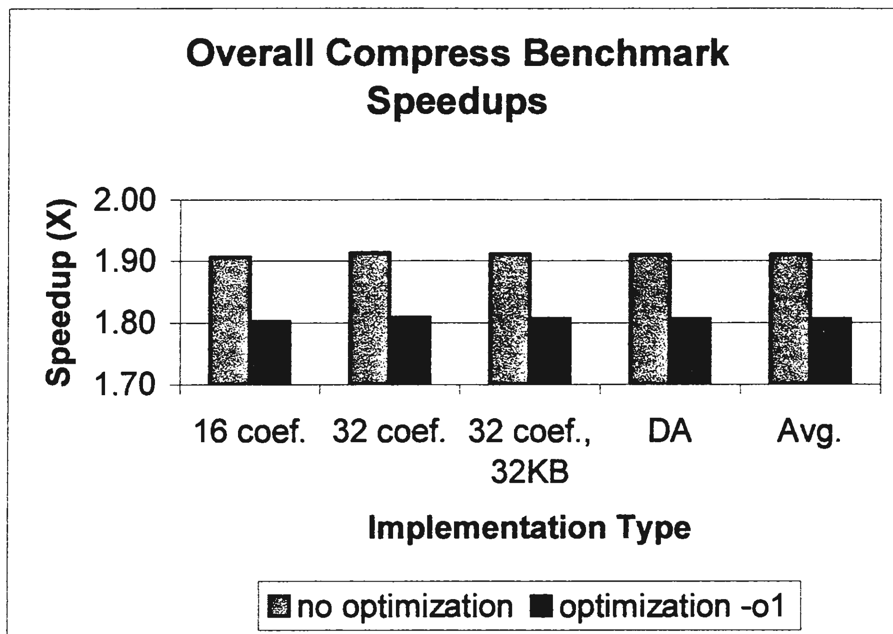

Figure 17. Overall Compress Benchmark Speedups

4.1.2 Power

The power consumption comparisons are presented in Figure 18. The results are reported as the percentage of average total conditional clocking 3 (cc3) power per cycle for Non-RFC used by the RFC version (RFC cc3 avg. total power per cycle/Non-RFC cc3 avg. total power per cycle*100). For each implementation the RFC benchmark consumed less power than the Non-RFC version with an average power consumption savings of approximately 11%. This is most likely due to the fact that registers and most existing functional units are not accessed while the simulator is in RFC mode. Additionally, when the simulator is performing table look-ups only the smaller, four-input decoders are used. For all of the implementations the compiler-optimized code used less power than the code with no compiler optimizations. The 32KB cache version used the least amount of power of the four implementations consuming 87% of the non-RFC power.

المنارة للاستشارات
www.manaraa.com

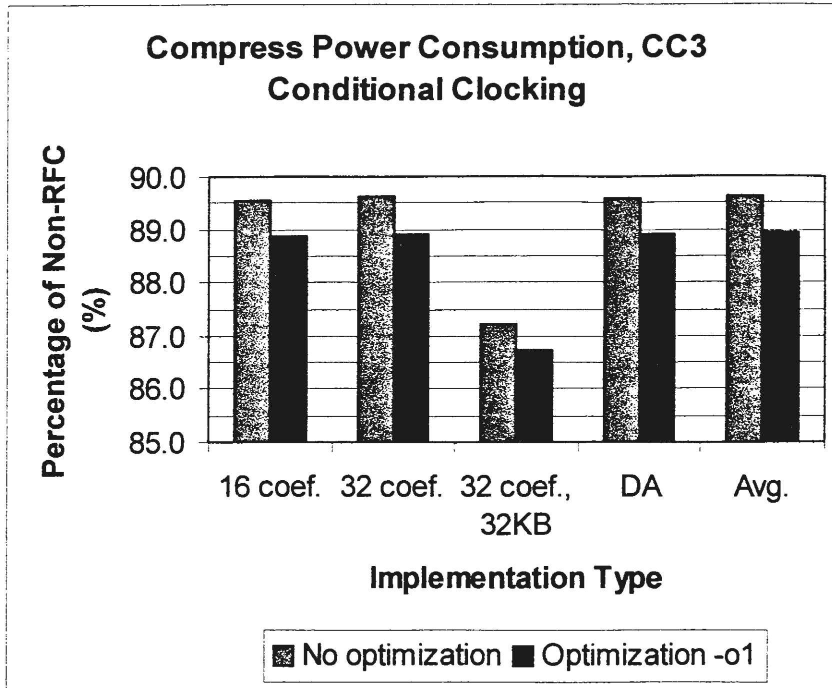

Figure 18. Percentage of Non-RFC power consumed by RFC Benchmark.

4.1.3 Energy Requirements

The energy requirements for the `Compress` benchmark are presented in Figure 19. A scaled energy value was computed to reflect the additional reduction in energy required due to the RFC version running for fewer cycles. The scaled energy value was computed with Equation 11.

$$E_{scaled} = f \times E_{orig.} \times \frac{T_{RFC}}{T_{orig.}} \tag{11}$$

In Equation 11 f is the fraction of non-RFC power consumed by the RFC version, E_{old} is the energy required by the non-RFC version and T is the number of clock cycles used. All RFC

المنارة للاستشارات

www.manaraa.com

implementations required less energy than their non-RFC counterparts, with RFC energy requirements ranging from 45.5 – 49% of non-RFC energy needs. The code that was not optimized required less energy to run than the optimized code for all implementations. This is often the case, as code that is optimized for speed often uses more instructions to implement a given function than code that is not optimized.

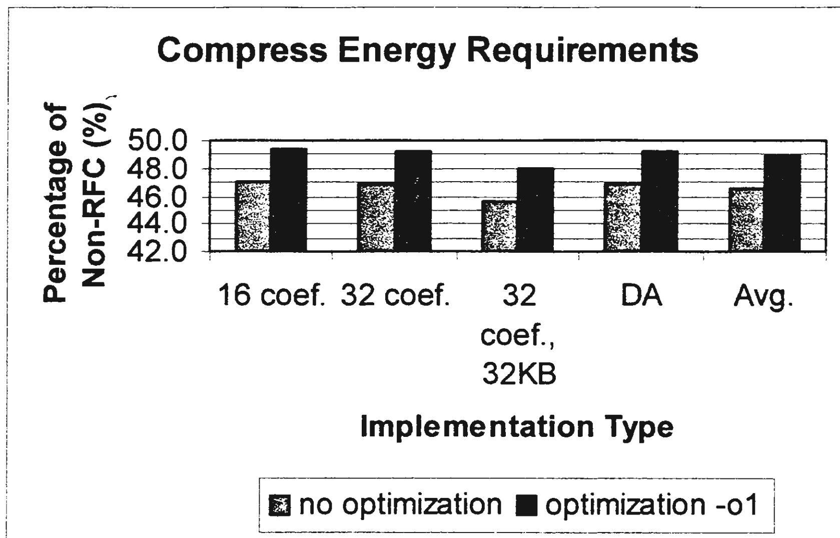

Figure 19. Percentage of Non-RFC Energy Required by Compress Benchmark.

The energy requirements are also shown as the percentage of energy saved by the RFC implementations. These results are shown in Figure 20. The energy savings were computed by subtracting the percentage of non-RFC energy required (shown in Figure 19) from 100. Therefore, energy savings were greatest for the code that was not optimized with the 32 coefficient, 32KB implementation achieving the greatest savings at 54.4%. The savings ranged from 50.7-54.4%.

المنارة للاستشارات
www.manaraa.com

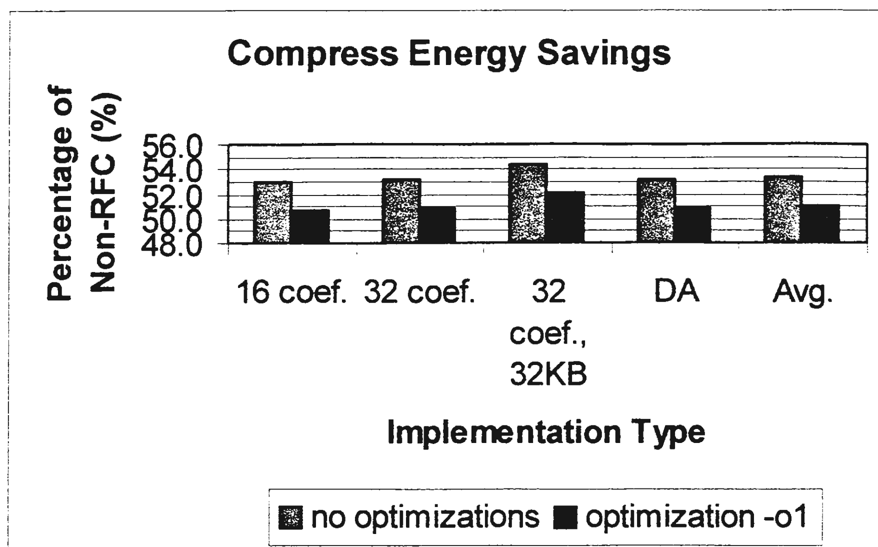

Figure 20. Energy Savings for RFC Compress Benchmark.

4.1.4 L1 Data Cache Miss Rates

Since the RFC reduces the amount of L1 caching area by one-half it is expected that miss rates would increase—perhaps even double. In fact, for the unoptimized code the overall L1 data cache miss-rate for both RFC versions that used a 16KB cache was double that of the non-RFC versions that used a 16KB cache. However, the overall miss rate for these two RFC implementations was only 0.02%, therefore this increase in the miss rate did not hinder performance too greatly. The optimized code and both the optimized and unoptimized versions that used a 32KB cache did not show any increase in miss rates. This lack of increase most likely accounts for the performance improvements seen for the optimized kernels in section 4.1.1. The lack of increase in miss rate for the optimized code was likely due to changes in the way data was loaded and stored locally due to the register and local code optimizations. Overall, the low miss rates achieved by the C64x, even when increased due to the RFC, especially when combined with lower power usage and improved

المنارة للاستشارات
www.manaraa.com

performance do not warrant limiting the use of RFC to avoid the increased miss rates. The miss rates for the `Compress` benchmark are shown in Figure 21. They are presented as the RFC miss rate increase over Non-RFC (RFC L1 data cache miss rate/Non-RFC L1 data cache miss rate).

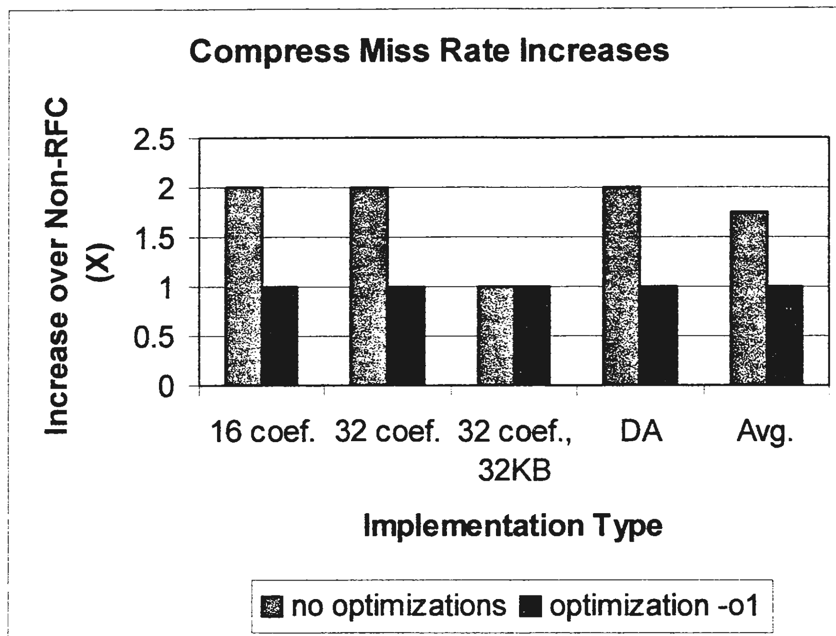

Figure 21. Increases in Miss Rates for RFC Compress Benchmarks.

4.2 Convolution

4.2.1 Performance

The RFC implementations of convolution for the Edge Detect benchmark were promising. While not as high as those for DCT, they were still worthy of implementation, especially if code size is of concern. The higher optimizations showed less speedup. This is due to the compiler's ability to successfully optimize the code's performance for the current DSP. However, even with an optimization of –o2 the RFC showed a speedup of 12X over the non-

www.manaraa.com

RFC code. If the –o3 optimized code had run successfully, it probably would have shown performance improvements less than 12X for the RFC as the improvements steadily decreased from no optimizations to –o2 optimizations. Figure 22 shows the kernel speedups for the different optimizations.

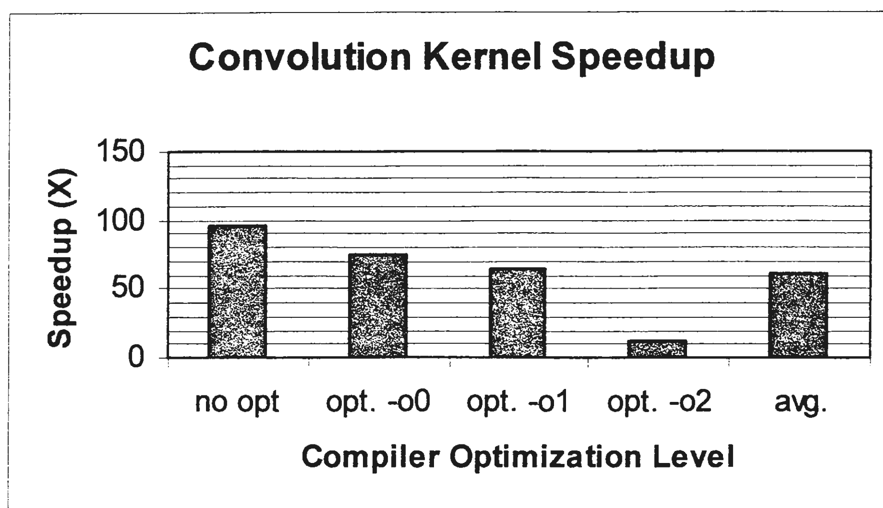

Figure 22. Speedup of the RFC Convolution Kernel for Edge Detect.

Like the `Compress` benchmark, there is some overall speedup of the `Edge Detect` benchmark. These values are lower than those seen for `Compress` due to two facts. One, the kernel speedups themselves were not as great as they were for `Compress`. Two, the convolution kernel only accounts for approximately 13% of the cycles required to complete benchmark execution whereas DCT accounts for approximately 45% of the cycles required to complete benchmark execution. The benchmark speedup values are listed in Table 8 and shown in Figure 23. The kernel speedups and the overall benchmark speedups are consistent with what was expected due to Amdahl's Law.

المنارة للاستشارات
www.manaraa.com

Table 8. Overall Benchmark Speedups for Edge Detect.

Overall Edge Detect Benchmark Speedups (Total Non-RFC Benchmark cycles/Total RFC Benchmark Cycles)	
Implementation Type	**Speedup**
No optimization	1.22
-o0 optimization	1.18
-o1 optimization	1.16
-o2 optimization	1.02

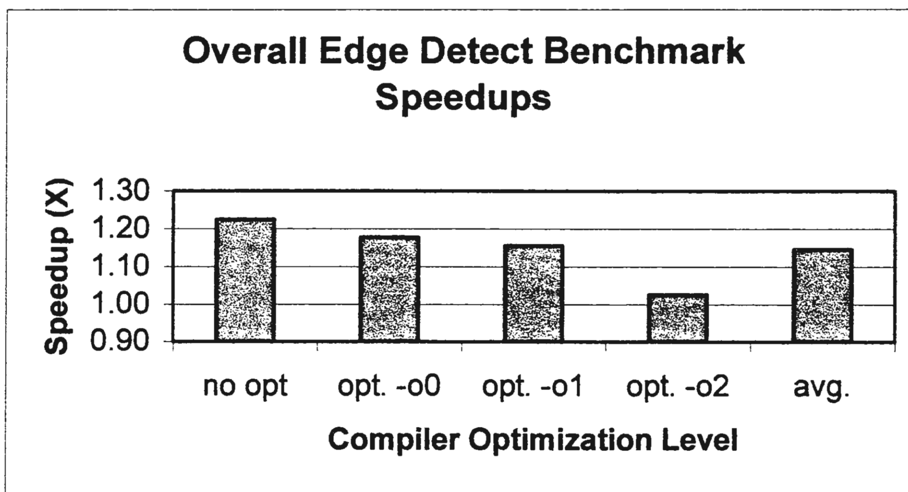

Figure 23. Overall Benchmark Speedups for Edge Detect.

4.2.2 Power

The total average power consumed per cycle for conditional clocking 3 was lower for all of the RFC benchmarks compared to the corresponding non-RFC benchmarks. This was expected, for the same reasons as were listed in section 4.1.2. Oddly, the versions that were optimized with –o1 and with –o2 consumed more average power per cycle overall than those with no optimizations. The benchmark optimized with –o0 consumed the least amount of average power per cycle overall. However, percentage wise, the –o0 and –o1 versions consumed a higher percentage of their corresponding non-RFC version than did the version

المنارة للاستشارات
www.manaraa.com

with no optimizations and the version with –o2 optimization. The percentage of average total power per cycle of the non-RFC version consumed by its corresponding RFC version is shown in Figure 24. These values ranged from 85-89% with an average of 87% of the non-RFC power being consumed by the RFC versions.

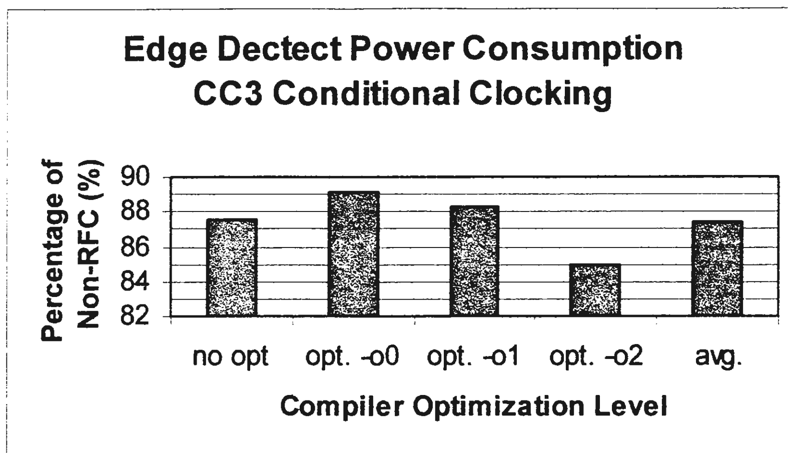

Figure 24. Percentage of Non-RFC power consumed by RFC Benchmarks.

4.2.3 Energy Requirements

The energy requirements are again shown as the percentage of non-RFC energy required by the RFC version. These results are displayed in Figure 25. In keeping with Amdahl's Law, the percentage of energy required by the RFC versions of the `Edge Detect` benchmark are higher than those seen for the `Compress` benchmark (and thus energy savings are lower). The percentage of non-RFC energy required for the RFC implementations ranged from 71.5-82.9% with an average of 76.6%. As seen with the previous benchmark, the code that was not optimized required less energy than the optimized versions, with energy requirements increasing as optimization levels increased. The energy savings are shown in Figure 26. The savings ranged from 17.1-28.5% with an average of 23.4%.

المنارة للاستشارات
www.manaraa.com

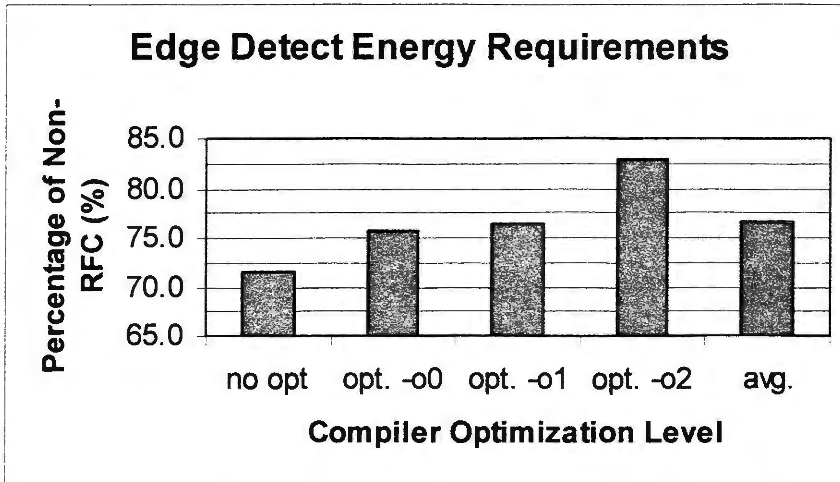

Figure 25. Percentage of Non-RFC Energy Required by the Edge Detect Benchmark.

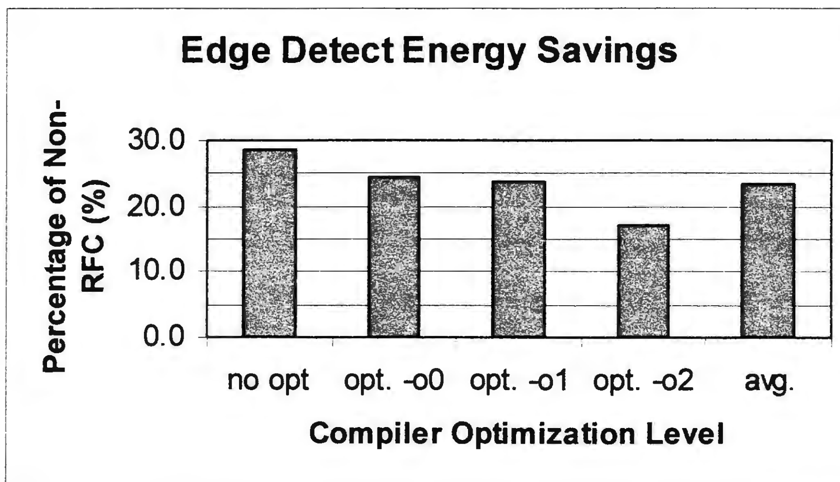

Figure 26. Energy Savings for Edge Detect Benchmark.

4.2.4 L1 Data Cache Miss Rates

Another interesting phenomenon was observed with the L1 data cache miss rates for the RFC `Edge Detect` benchmarks—they were actually lower than their non-RFC counterpart for all except the unoptimized code. The decrease in miss rates was most likely due to blocking effects caused by differences in how the data was retrieved for processing between the RFC

and non-RFC versions. This is the only benchmark that this was observed with. The increase/decrease in miss rates are displayed in Figure 27. Again, these are presented as the result of RFC miss rate divided by non-RFC miss rate. The values that are less than one show a decrease in the RFC miss rate.

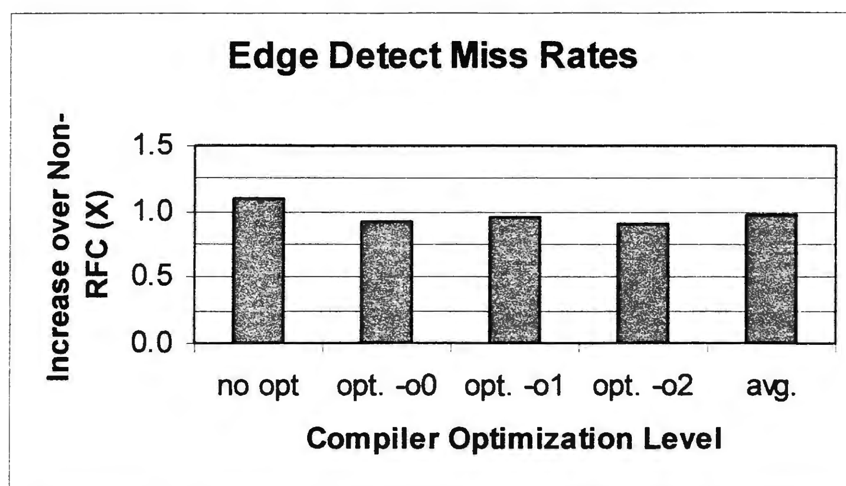

Figure 27. Miss Rate Increase (Decrease) for RFC over Non-RFC.

4.3 FFT

4.3.1 Performance

The speedups achieved by the RFC FFT kernels were not as great as those achieved by DCT or convolution. However, they did show the same trend as convolution in terms of speedups achieved for different optimization levels. With FFT, like with convolution, the higher the optimization, the lower the speedup. Again, the code generated with level –o3 optimizations would not execute on the simulator, so its results cannot be given. For FFT the speedup ranged from 36X for the unoptimized code down to 10X for level –o2 optimized code. These results are presented in Figure 28.

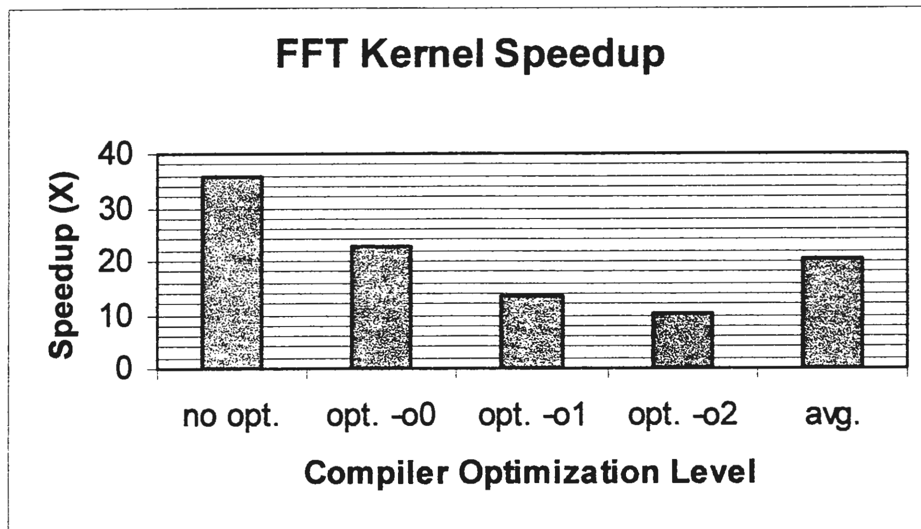

Figure 28. RFC Kernel Speedup over Non-RFC for FFT.

The overall speedups for the benchmarks are actually slightly better than those seen for `Edge Detect`, despite the fact that the kernel speedups were not as high. This is because FFT accounts for about 20% of the total cycles required to complete benchmark execution, so any improvements seen in the FFT kernel will have a larger impact on the overall performance. These values are listed in Table 9 and shown in Figure 29.

Table 9. Overall Benchmark Speedups Observed for the Spectral Benchmark.

Overall Spectral Benchmark Speedups (Total Non-RFC Benchmark cycles/Total RFC Benchmark Cycles)	
Implementation Type	**Speedup**
No optimization	1.44
-o0 optimization	1.28
-o1 optimization	1.17
-o2 optimization	1.13

المنارة للاستشارات
www.manaraa.com

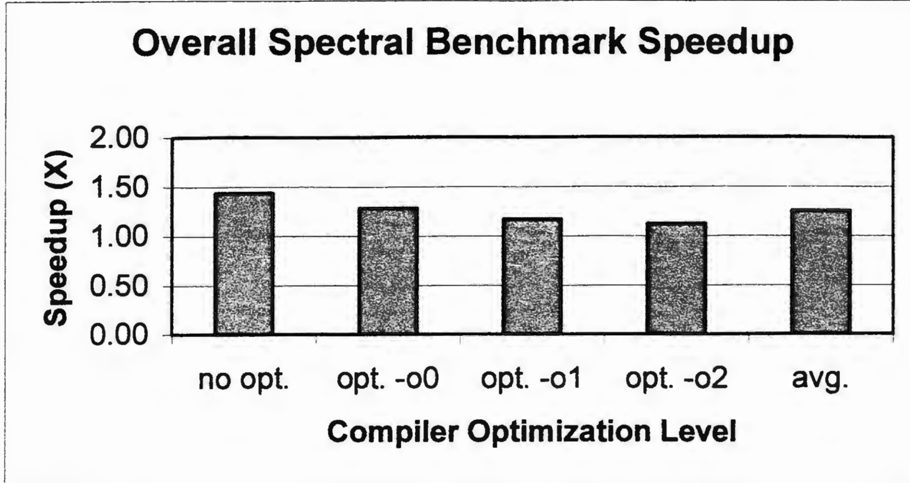

Figure 29. Overall Benchmark Speedups for the Spectral Benchmark.

4.3.2 Power

For the `Spectral` benchmark the values used to compare power performance were again the average total power consumed per cycle for cc3. Overall, the unoptimized RFC version used the least amount of power per cycle, with the –o2 RFC version using the second least amount overall. However, when looking at the percentage of non-RFC average power per cycle consumed by its corresponding RFC version, the percentage of non-RFC power consumed steadily decreases from the unoptimized version to the –o2 version. These results are displayed in Figure 30.

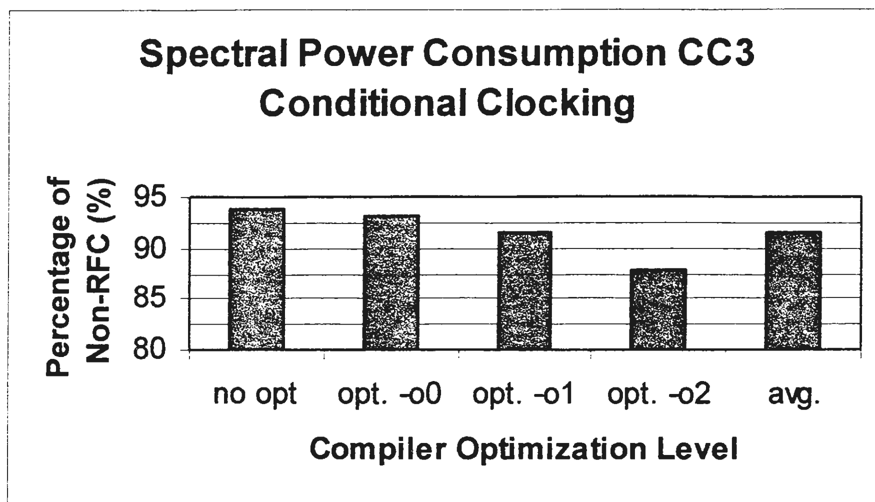

Figure 30. Percentage of Non-RFC Power Consumed by RFC.

4.3.3 Energy Requirements

The percentage of non-RFC energy used by the RFC implementations of Spectral are shown in Figure 31. These percentages ranged from just over 65% to nearly 78.5% with an average of 73.6%. As seen with the previous two benchmarks, the unoptimized code required the least percentage of energy with percentages increases as optimization levels increased. For this benchmark optimization level –o1 used a slightly higher percentage of energy than level –o2, but the energy requirements between the two levels were not significantly different. The energy savings are presented in Figure 32. Again, the savings are greatest for the unoptimized code at 34.7%, with an average energy savings of 26.4%.

www.manaraa.com

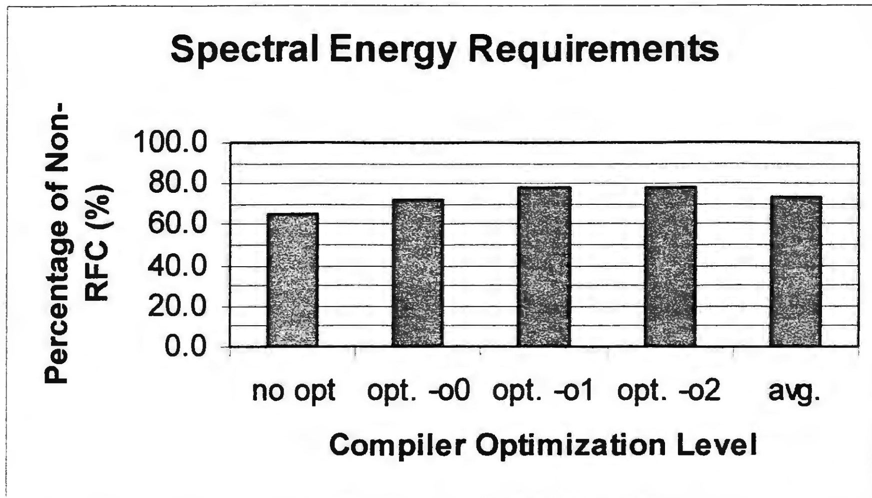

Figure 31. Percentage of non-RFC Energy Required by the Spectral Benchmark.

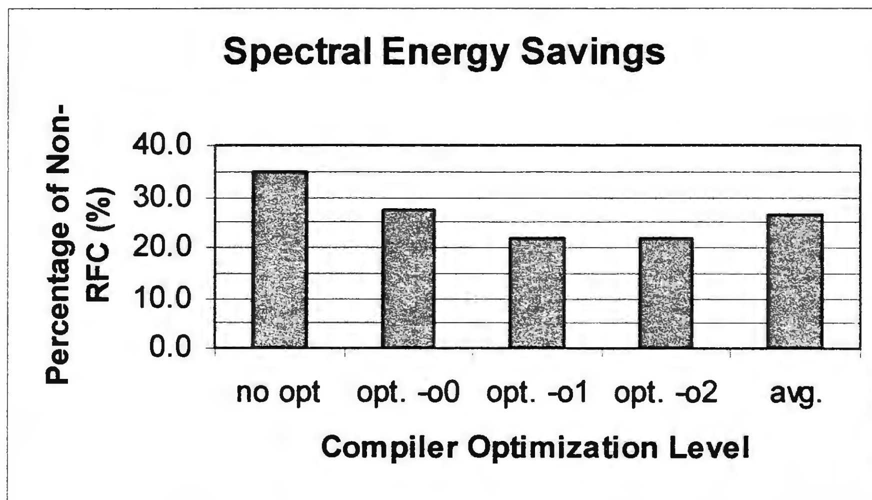

Figure 32. Energy Savings for Spectral Benchmark.

4.3.4 L1 Data Cache Miss Rates

The miss rates of the level one data cache were all slightly higher for the RFC versions than the non-RFC versions, as was expected. However, none of the increases were twice the non-RFC rate. The largest increase was 1.6X for the version with no compiler optimizations.

This was a lower increase than was observed for Compress. But an increase in the miss rate for the level –o1 optimization was observed, whereas for Compress, the –o1 optimization did not experience an increase in the miss rate. Again, though, these miss rate increases are not so great as to warrant avoiding RFC implementation. The level one data cache miss rate increases are shown in Figure 33.

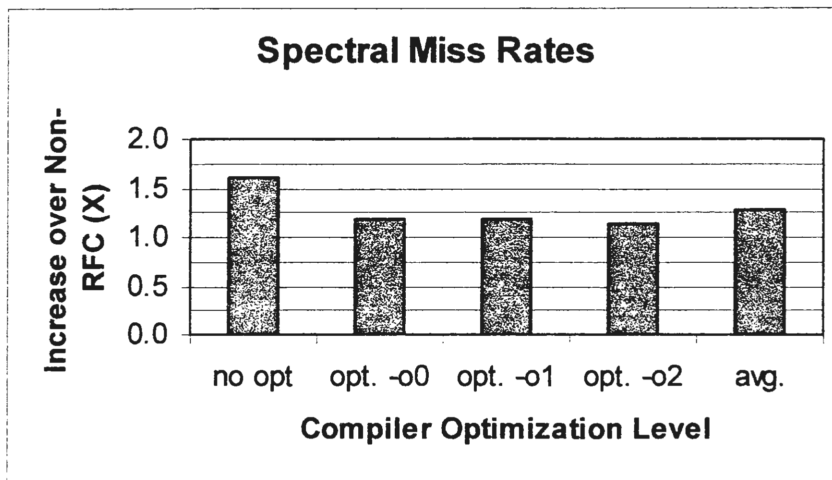

Figure 33. Increases in Miss Rates for the RFC Spectral Benchmarks.

4.4 FIR, 32-Tap and 256-Tap

4.4.1 Performance

The TI C6000 compiler does a good job of optimizing FIR kernels. In fact, for the 256-tap FIR only RFC runs with inputs of 2048 or higher were able to show any improvement over the non-RFC performance. Part of the lack of speedup is definitely due to the high overhead of reconfiguration for a 256-tap FIR in RFC, but it doesn't help the RFC that the compiler alone can improve the performance of the FIR kernel about 10 times over that of code with no optimizations. The 32-tap FIR kernel did slightly better than the 256-tap at achieving

www.manaraa.com

speedups with fewer inputs. Even so, the best speedup achieved for the RFC 32-tap FIR is just over three times faster than the non-RFC version when –o2 compiler optimized code is used. Based on these results, if large numbers of inputs (greater than 2048) are going to be processed, then the reconfiguration overhead will not diminish the benefits of implementing RFC for FIR. If, however, the number of inputs to be processed will be less than 2048, better performance will be achieved by simply using the TI compiler to optimize the code. The speedups for the RFC 32-tap FIR are given in Figure 34 and those for the RFC 256-tap FIR are given in Figure 35. Due to the fact that the FIR code was not an entire benchmark, but rather just the FIR kernel plus I/O, the overall speedup was negligible (at most 1.01X for the unoptimized 32-tap FIR). There was no overall speedup for the optimized RFC 32-tap FIR and, of course, overall performance decreases were seen for the optimized, 256-tap FIR RFCs with inputs fewer than 2048.

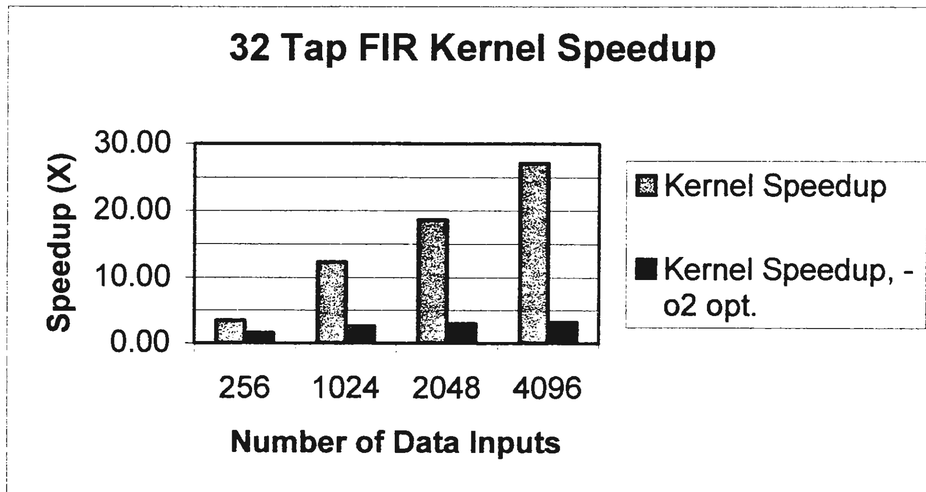

Figure 34. FIR, 32-tap, Kernel Speedup for RFC Implementation.

www.manaraa.com

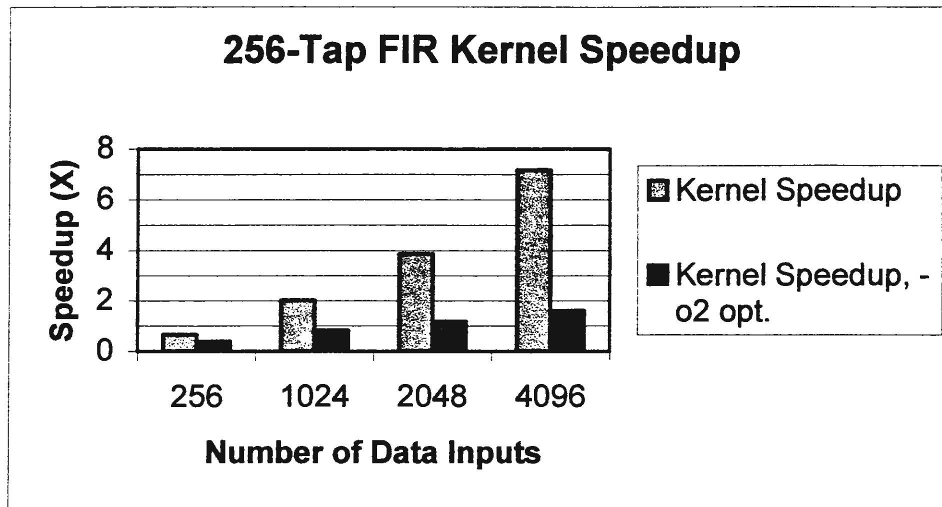

Figure 35. FIR, 256-tap, Kernel Speedup (Decrease) for RFC.

4.4.2 Power

All of the reconfigurable FIR kernels consumed less average power per cycle than the non-RFC versions. This is expected since fewer of the components that contribute to the total power each cycle are used when the kernel is in reconfigurable mode. The average amount consumed per cycle did not appear to be dependent upon the number of inputs. This seems logical because nothing in terms of hardware would be different per cycle to affect the power. The overall total power consumption would be expected to be higher simply due to the longer processing time, but when averaged over each cycle, a difference would not be expected. The average amount consumed per cycle did not seem to be dependent upon the number of taps either, with both the 32-tap and the 256-tap reconfigurable FIRs consuming about 85% of their counterpart non-RFC FIRs. Interestingly, the optimized RFC versions for less than 2048 inputs for both the 32-tap and the 256-tap appeared to consume more power per cycle than the unoptimized versions. However, considering the unknown amount of error

المنارة للاستشارات
www.manaraa.com

within the power measurements, these differences were not significant enough to conclusively state that the optimized versions consumed more power for inputs fewer than 2048. Due to the lack of variation in the percentage of non-RFC average power per cycle consumed by the RFC a graph is not given.

4.4.3 Energy Requirements

Due to the fact that the percentage of non-RFC power consumed by the 32-tap and 256-tap FIR filters was fairly consistent for various numbers of data inputs, the percentage of non-RFC energy required by these filters was fairly consistent as well. The percentage of non-RFC energy used by the 32-tap RFC filter ranged from 84.4%-84.7% for the code that was not optimized and from 85.5-85.7% for the –o2 optimized code. As the number of data inputs increased, the percentage of energy required decreased. Likewise, the percent savings were fairly consistent as well ranging from 15.3 to 15.6% for the unoptimized code and from 14.3 to 14.5% for the –o2 optimized code. The energy requirements for the 32-tap FIR filter are shown in Figure 36 and the savings are shown in Figure 37.

The range of energy required by the RFC 256-tap FIR was almost identical to the range for the 32-tap filter. The percentage of non-RFC energy used for the 256-tap filter (code not optimized) with 256 data inputs was 85.4% with the percentage used for numbers of data inputs 1024 and greater being 85.2%. With the level –o2 optimizations the range was 85.7-85.8%. Likewise, the percent savings range was 14.6-14.8% for the code with no compiler optimizations and 14.2-14.3% for the code with level –o2 compiler optimizations. Due to the similarity of these results to those for the 32-tap filter, graphs are not given.

المنارة للاستشارات
www.manaraa.com

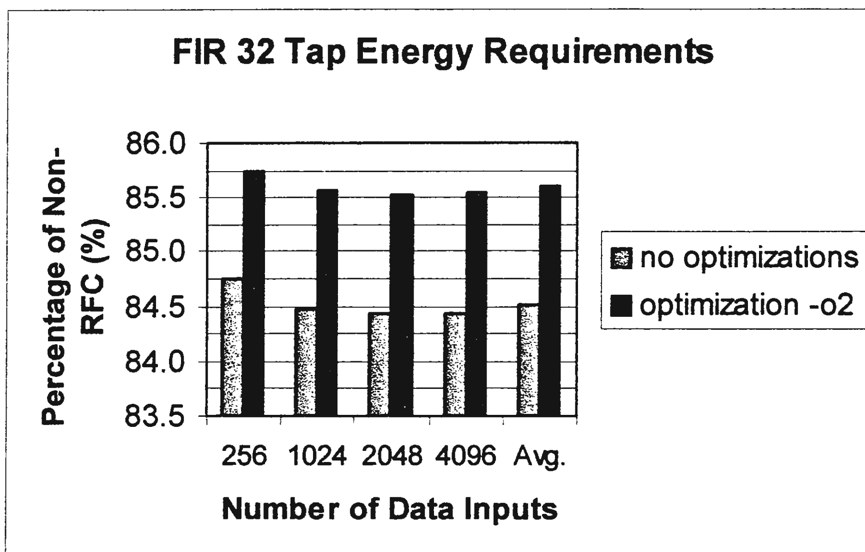

Figure 36. Percentage of Non-RFC Energy Required by the 32-Tap FIR Kernel.

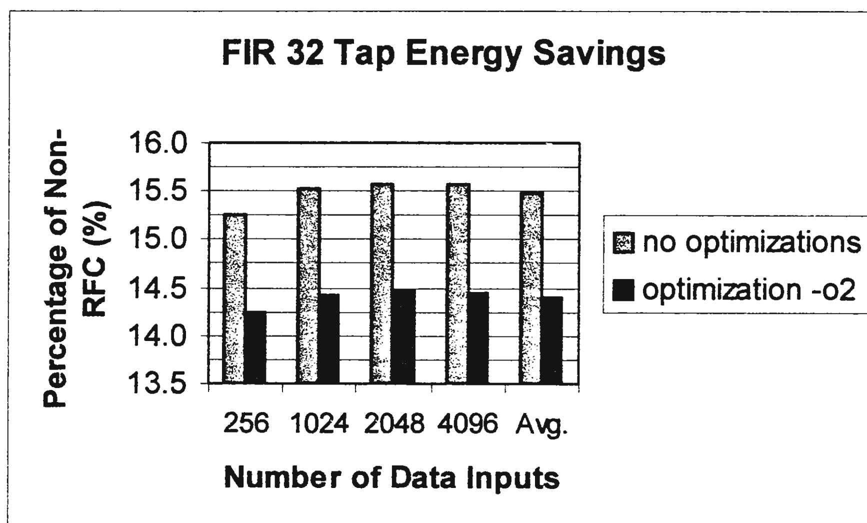

Figure 37. Energy Savings for the 32-Tap FIR Filter Kernel

4.4.4 Level One Data Cache Miss Rates

The L1 data cache miss rate increases for both the 32-tap and the 256-tap reconfigurable FIRs are difficult to interpret. For most of the input levels the amount of compiler

المنارة للاستشارات
www.manaraa.com

optimization did not affect the miss rate compared to the miss rate of the unoptimized code. Only for the 256-tap FIR with 1024 inputs was a difference observed for the –o2 optimized code. That run experienced no increase in the miss rate compared to the non-RFC version, whereas the unoptimized RFC version doubled the miss rate compared to its non-RFC counterpart. Even more confusing is the miss rate behavior of the reconfigurable 32-tap FIR with 1024 inputs. Both the optimized and unoptimized codes for the reconfigurable 32-tap FIR had a miss rate twice that of the non-RFC FIRs with 1024 inputs. The only other 32-tap input level to experience a miss rate increase was the 256 input and that increase was only 1.3 times greater than the non-RFC 256 input FIR. Both the 32-tap and 256-tap FIRs did not experience any change in the miss rate when 4096 inputs were used. They also had the lowest miss rates of the FIR runs with a miss rate of 0.0001%. Perhaps this is because a cache can take advantage of more temporal and spatial locality when the input array is that large, thus resulting in fewer misses. The miss rate increases for the 32-tap reconfigurable FIR are given in Figure 38 and the increases for the 256-tap reconfigurable FIR are given in Figure 39.

المنارة للاستشارات
www.manaraa.com

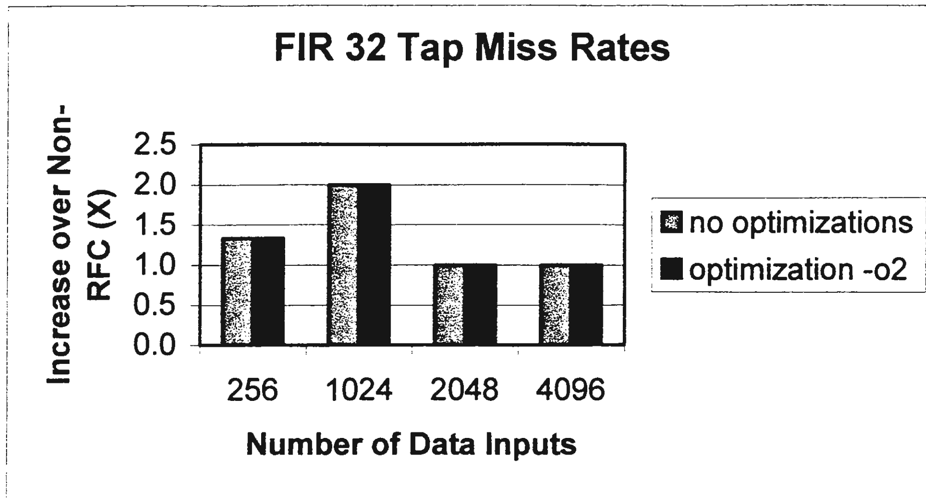

Figure 38. Miss Rate Increases for 32-tap Reconfigurable FIR.

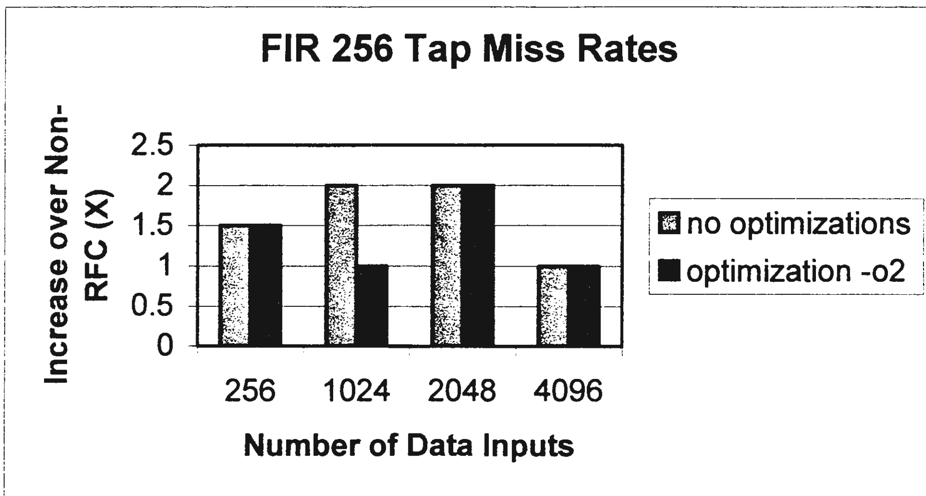

Figure 39. Miss Rate Increases for 256-tap Reconfigurable FIR.

www.manaraa.com

CHAPTER 5. CONCLUSIONS

This research has looked at both the power and performance measurements of a reconfigurable functional cache within a TMS320C64X digital signal processor. The use of a reconfigurable functional cache with a general purpose processor has been shown to be beneficial and feasible by other researchers. This work has expanded upon that idea by implementing the reconfigurable functional cache in one of the ways of the 2-way set associative level one data cache on the C64x. To test the performance and power consumption of the RFC on the C64x a simulator, the C6400, was created. The C6400 was made by merging together parts from three other simulators and expanding upon them to implement the entire C64x ISA within the simulator. Using the simulator with a normal level one data cache established a baseline to compare the reconfigurable C6400 to. Three benchmarks that contained computationally intensive kernels were used. The benchmarks used were Compress, Edge Detect and Spectral from the UTDSP benchmark suite. The kernels within these benchmarks were DCT, convolution and FFT. These kernels, along with a 32-tap FIR and a 256-tap FIR (also from UTDSP) were implemented in the reconfigurable cache. Additionally, since the TI C6000 DSP family has a compiler that is capable of advanced optimizations, at least two different levels of optimizations were tested for each benchmark/kernel. Results showed that the TI compiler is capable of improving the performance of these kernels on its own. However, even with –o2 levels of optimization, the results that were obtained were promising. Speedups were most impressive for DCT, which normally consumes about 45% of the cycles required to execute the benchmark it is in. The DCT speedups ranged from 128X up to over 350X depending upon the RFC implementation and the level of optimization. The performance results for convolution and FFT, while not as

المنارة للاستشارات
www.manaraa.com

great, were still encouraging. The speedups for convolution ranged from 12X to 95X depending upon the optimization level and the speedups for FFT ranged from 10X to 36X. The performance results for the 32-tap and 256-tap FIR filters illustrated how configuration overhead can diminish the RFC advantage. These results also depicted the compiler's ability to optimize MAC operations. Small speedups were shown for reconfigurable FIR filters as long as the number of data inputs was greater than 2048.

The relative power performances for all of the RFC implementations clearly showed that a reconfigurable functional cache is viable in an embedded environment. None of the RFC implementations consumed more power than the non-reconfigurable implementations. Most RFC simulations required about 85-90% of the power required by the standard configurations.

The last statistic that was compared for the reconfigurable and standard simulations was the level one data cache miss rate. Since the RFC removes one of the ways from the eligible caching area, it effectively reduces the level one data cache size in half when in RFC mode. If the RFC were to cause huge increases in miss rates, the increase in cycle latencies experienced for accessing main memory would offset any performance gains. The most that the RFC increased the miss rates was 2X. But, even in this worst case the miss rate was only 0.0002% which did not hinder performance greatly. One benchmark, Edge Detect, even experienced miss rate decreases for the RFC simulations.

المنارة للاستشارات
www.manaraa.com

In conclusion, these findings support the initial hypothesis that a reconfigurable computing cache could enhance the performance of a DSP while maintaining flexibility and still being energy-efficient enough to be suitable for an embedded environment.

المنارة للاستشارات

www.manaraa.com

REFERENCES

[1] W. Strauss, "DSP Market Alert", *Forward Concepts*, www.fwdconcepts.com/press32.htm, Oct. 2001 (Date accessed: 15 June 2002).

[2] A. Singhal. "Reconfigurable Cache Architecture", M.S. Thesis, Dept. of Computer Science, Iowa State University, Ames, IA, May 2000.

[3] J.L. Hennessy and D.A. Patterson, Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers, Inc, San Franciso, CA, Second Edition, 1996.

[4] J. Brenner. "Cache Usage in High-Performance DSP Applications With the TMS320C64x", Application Report SPRA 756, Available at www.ti.com, Dec. 2001 (Date accessed: 28 Mar. 2002).

[5] Michael J. Lee, "Cache Justification for DSP Processors", EE382C Embedded Software Systems, www.ece.utexas.edu/~bevans/courses/ee382c/projects/fall99/index.html, Oct. 1999 (Date accessed: 15 June 2002).

[6] Hue-Sung Kim, "Towards Adaptive Balanced Computing (ABC) using Reconfigurable Functional Caches (RFCs)", Ph.D. Dissertation, Dept. of Computer Engineering, Iowa State University, Ames, IA, 2001.

[7] H. Kim, A.K. Somani and A. Tyagi. "A Reconfigurable Multi-function Computing Cache Architecture", *IEEE Transactions on VLSI Systems*, Vol. 9, No. 4, pp. 509-523, Aug. 2001.

[8] A. Singhal, A. Somani and A. Tyagi, "A Reconfigurable Cache Module Architecture", *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '01*, 2001.

[9] P. Ranganathan, S. Adve and N.P. Jouppi, "Reconfigurable Caches and Their Application to Media Processing", *Proceedings of the 27th International Symposium on Computer Architecture (ISCA-27)*, pp. 214-224, June 2000.

[10] P. Graham and B. Nelson, "Reconfigurable Processors for High-Performance, Embedded Digital Signal Processing", *Proceedings of the Ninth International Workshop on Field Programmable Logic and Applications*, Aug. 1999.

[11] J.M. Rabaey, "Reconfigurable Processing: The Solution to Low-Power Programmable DSP", *Proceedings of the 1997 ICASSP Conference*, Vol. 1, pp. 275-278, Munich, Apr. 1997.

المنارة للاستشارات

www.manaraa.com

[12] A. Abnous, K. Seno, Y. Ichikawa, M. Wan and J. Rabaey, "Evaluation of a Low-Power Reconfigurable DSP Architecture", *Proceedings of the Reconfigurable Architecture Workshop*, Orlando, Florida, Mar. 1998.

[13] M. Wan, Y. Ichikawa, D. Lidsky, J. Rabaey, "An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPS", *Proceedings of the Custom Intergrated Circuit Conference*, pp.111-117, Santa Clara, CA, May 1998.

[14] J. Rabaey and M. Wan, "An Energy-Conscious Exploration Methodology for Reconfigurable DSPs", *Proceedings of the 1998 Design Automation and Test in Europe*, pp. 341-342, 1998.

[15] H. Zhang, M. Wan, V. George and J. Rabaey. "Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs", *Proceedings of the Workshop on VLSI*, Orlando, Florida, pp. 2-8, Apr. 1999.

[16] S. Li, M. Wan and J. Rabaey. "Configuration Code Generation and Optimizations for Heterogeneous Reconfigurable DSPs", *Proceedings of SIPS*, pp. 169-180, 1999.

[17] M. Wan, H. Zhang, M. Benes and J. Rabaey, "A Low-Power Reconfigurable Data-Flow Driven DSP System", *Proceedings of SIPS*, pp. 191-200, 1999.

[18] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu and J. Rabaey, "Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System", *Journal of VLSI Signal Processing*, 2000.

[19] "A Low-Energy Heterogeneous Reconfigurable DSP IC", *DAC Design Contest*, bwrc.eecs.berkeley.edu/Research/Configurable_Architectures/papers.html, 2000 (Date accessed: 16 May 2001).

[20] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous and J.M. Rabaey, "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications", *Proceedings of ISSCC*, pp. 68 -69, 448, 2000.

[21] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous and J.M. Rabaey, "A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing", *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 11, pp. 1697-1704, Nov. 2000.

[22] V. Cuppu, "Cycle Accurate Simulator for TMS320C62x, 8 way VLIW DSP Processor", Graduate Student, Electrical Engineering, University of Maryland, www.ece.umd.edu/~ramvinod/c6xsim.pdf (Date accessed: 21 June 2001).

[23] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical Report #1342, University of Wisconsin, Madison, June 1997.

المنارة للاستشارات

www.manaraa.com

[24] D. Brooks, V. Tiwari, and M. Martonosi. "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", *27th International Symposium on Computer Architecture (ISCA)*, pp. 83-94, Vancouver, British Columbia, June 2000.

[25] V.S. Pai, P. Ranganathan and S. Adve, "RSIM: A Simulator for Shared-Memory Multiprocessor and Uniprocessor Systems that Exploit ILP", *Proceedings of the Third Workshop on Computer Architecture Education*, 1997.

[26] J.R. Haueser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor", *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM '97), pp. 12-21, Apr. 16-18, 1997.

[27] R.D. Wittig and P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", *The Fourth Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, pp. 126-135, Mar. 1996.

[28] J.A. Jacob and P. Chow, "Memory Interfacing and Instruction Specification for Reconfigurable Processors", International Symposium on Field-Programmable Gate Arrays (FPGA '99), pp. 145-154, ACM/SIGDA, Feb. 1999.

[29] S. Hauck, T.W. Fry, M.M. Hosler and J.P. Kao, "The Chimaera Reconfigurable Functional Unit", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 87-96, 1997.

[30] G. Lu, M. Lee, H. Singh, N. Bagherzadeh, F.J. Kurdahi and E.M. Filho, "MorphoSys: a Reconfigurable Processor Targeted to High Performance Image Application", *6th Reconfigurable Architectures Workshop, at 13th International Parallel Processing Symposium*. Puerto Rico, Apr. 1999.

[31] H. Singh, M. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh and E.M. Chaves Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel Computation-Intensive Applications", *IEEE Transactions on Computers*, Vol. 49, No. 5, pp. 465-481, May 2000.

[32] C.A. Moritz, D. Yeung and A. Agarwal, "Exploring Optimal Cost-Performance Designs for Raw Microprocessors", *Proceedings of the International IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 98*, pp. 12-27, Apr. 1998.

[33] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to Software: Raw Machines", *IEEE Computer*, Vol. 30, No. 9, pp. 86-93, Sept. 1997.

[34] M.B. Taylor, J. Kim, J. Miller, F. Ghodrat, B. Greenwald, P. Johnson, W. Lee, A. Ma, N. Shnidman, V. Strumpen, D. Wentzlaff, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw Processor - A Scalable 32-bit Fabric for Embedded and General Purpose Computing", presented by Michael Bedford Taylor at *Hotchips XIII*, Palo Alto, California, Aug. 2001.

المنارة للاستشارات
www.manaraa.com

[35] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-Chip Communications Architecture", *Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques*, pp. 37-46, Philadelphia, PA, Oct. 2000.

[36] W. Burleson, R. Tessier, D. Goeckel, S. Swaminathan, P. Jain, J. Euh, S. Venkatraman and V. Thyagarajan, "Dynamically Parameterized Algorithms and Architectures to Exploit Signal Variations For Improved Performance and Reduced Power", *The Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, 2001 (ICASSP'01)*, Vol. 2, pp. 901-94, Salt Lake City, Utah, May 2001.

[37] G. Sassatelli, G. Cambon, J. Galy and L. Torres, "A Dynamically Reconfigurable Architecture for Embedded Systems", *12th International Workshop on Rapid System Protyping,2001*, pp. 32-37, 2001.

[38] T. Nishitani, "An Approach to a Multimedia System on a Chip", *IEEE Workshop on Signal Processing Systems, 1999, SIPS 99*, pp. 13-22, 1999.

[39] J. Becker, T. Pionteck and M. Glesner, "An Application-tailored Dynamically Reconfigurable Hardware Architecture for Digital Baseband Processing", *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, 2000*, pp. 341-346, 2000.

[40] F. Barat, M. Jayapala, P.O. de Beeck and G. Deconinck, "Reconfigurable Instruction Set Processors: An Implementation Platform for Interactive Multimedia Applications", *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, 2001*, Vol. 1, pp. 481-485, 2001.

[41] V. George, H. Zhang and J. Rabaey, "The Design of a Low Energy FPGA", *Proceedings of ISLPED 1999*, pp. 188-193, San Diego, CA, 1999.

[42] "TMS320C6000 Peripherals Reference Guide", Literature No. SPRU190D, available at www.ti.com, Feb. 2001 (Date accessed: 20 June 2001).

[43] "TMS320C6000 Instruction Set Reference Guide", Liturature No. SPRU189F, available at www.ti.com, Oct. 2000 (Date accessed: 20 June 2001).

[44] J. Sankaran, "Reed Solomon Decoder: TMS320C64x Implementation", Application Report SPRA686, available at www.ti.com, Dec. 2000 (Date accessed: 18 Feb. 2002).

[45] C6000 Applications Team, "How to Begin Development Today with the TMS320C6414, TMS320C6415, and TMS320C6416 DSPs", Application Report SPRA718, available at www.ti.com, Feb. 2001 (Date accessed: 20 June 2001).

[46] T. Hiers, C6000 Applications Team, "TMS320C6414/15/16 Power Consumption Summary", Application Report SPRA811A, available at www.ti.com, Mar. 2002 (Date accessed: 18 May 2002).

المنارة للاستشارات
www.manaraa.com

[47] M. Bhatnagar, L. Kondur, A. Nandi, B. Siravara, "Simulation of the Texas Instruments TMS320C64x Digital Signal Processor", Final Report for Computer Architecture, University of Texas, available at www.utdallas.edu/~siravara/finalreport.pdf, Fall 2000 (Date accessed: 15 June 2002).

[48] V. Tiwari, S. Malik, A. Wolfe and M.T.-C. Lee, "Instruction Level Power Analysis and Optimization of Software", *9th International Conference on VLSI Design*, pp. 326-328, Jan. 1996.

[49] M.T-C. Lee, V. Tiwari, S. Malik and M. Fujita, "Power Analysis and Low-Power Scheduling Techniques for Embedded DSP Software", *Proceedings of the Eighth International Symposium on System Synthesis*, pp. 110 –115, 1995.

[50] P. Landman, "High-Level Power Estimation", *International Symposium on Low Power Electronics and Design*, pp. 29-35, Monterey, CA, 1996.

[51] C.H. Gebotys and R.J. Gebotys, "Designing for Low Power in Complex Embedded DSP Systems", *Proceedings of the 32nd Hawaii International Conference on System Sciences*, pp. 1-8, 1999.

[52] R. Muresan and C.H. Gebotys, "Current Consumption Dynamics at Instruction and Program Level for a VLIW DSP Processor", *ISSS'01*, Vol. 14, pp. 130-135, Montreal, Quebec, Canada, Oct. 2001.

[53] S.J.E. Wilton and N.P. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches", *WRL Research Report 93/5*, Western Research Laboratory, Palo A lto, CA, July, 1994.

[54] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems", *30th Internal Symposium on Microarchitecture*, pp. 330-335, 1997.

[55] "The Role of Distributed Arithmetic in FPGA-based Signal Processing", Xilinx Application Notes, available from http://www.xilinx.com/appnotes/theory1.pdf, Oct. 2000 (Date accessed: 30 July 2001).

[56] "Video Compression Using DCT", Xilinx Application Note: Virtex-II Series, XAPP610, Ver. 1.2, available from http://www.xilinx.com/xapp/xapp610.pdf, Apr. 2002 (Date accessed: 17 June 2002).

[57] A.K. Somani and A. Tyagi, Instructors, H. Kim, Scribe, "Implementation of Functional Units using LUTs", Lecture 4&5, CprE/ComS 583x Adaptive Computing Systems, Iowa State University, available from http://class.ee.iastate.edu/somani/cpre583/lecturenotes.html, Fall 1998 (Date accessed: 17 July 2002).

المنارة للاستشارات

www.manaraa.com

[58] D. Cross, "FFT Fast Fourier Transforms", http://www.intersrv.com/~dcross/fft.html#section3, Feb. 2000 (Date accessed: 17 June 2002).

[59] "TMS320C6000 Optimizing Compiler User's Guide", Literature Number SPRU187, available at www.ti.com, Apr. 2001 (Date accessed: 14 May 2002).

المنارة للاستشارات

www.manaraa.com