SOURCE CODE OPTIMIZATIONS FOR LOW POWER CONSUMPTION ON MICROPROCESSOR-BASED SYSTEMS

By

David Andrés Ortiz López

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

November, 2007

Approved by:

Gladys O. Ducoudray, Ph.D Member, Graduate Committee	Date
Nelson Sepúlveda, Ph.D Member, Graduate Committee	Date
Nayda G. Santiago, Ph.D President, Graduate Committee	Date
Uroyoán R. Walker, Ph.D Representative of Graduate Studies	Date
Isidoro Couvertier, Ph.D Chairperson of the Department	Date

Abstract of Thesis Presented to the Office of Graduate Studies
of the University of Puerto Rico at Mayagüez in Partial Fulfillment of the
Requirements for the Degree of Master of Science

SOURCE CODE OPTIMIZATIONS FOR LOW POWER CONSUMPTION ON MICROPROCESSOR-BASED SYSTEMS

By

David Andrés Ortiz López

November 2007

Chair: Nayda G. Santiago
Major Department: Electrical and Computer Engineering

Power consumption is an important constraint in the design of battery-operated embedded systems. Minimizing power dissipation may be handled in terms of hardware or software optimizations. Source code-level optimization techniques have been used as an alternative to achieve low power consumption when programming embedded systems, however these techniques should be analyzed with statistical sound methods in order to reach strong conclusions about their actual impact on the power consumption. In this work, source code optimizations are applied on a set of representative benchmarks for embedded processors (MiBench), to analyze whether the techniques have or not an effect on the power dissipation of a set of microprocessor-based platforms. Design of experiments techniques (DOE) and analysis of variance (ANOVA) are used to achieve statistical sound conclusions. Results showed that not all optimizations have an effect on power consumption, moreover some techniques depend on the target platform where they are run.

Resumen de Tesis Presentada a la Oficina de Estudios Graduados
del Recinto Universitario de Mayagüez de la Universidad de Puerto Rico, como
Cumplimiento Parcial de los Requisitos para el Grado de Maestría en Ciencias

OPTIMIZACIÓN DE CÓDIGO PARA BAJO CONSUMO DE POTENCIA DE SISTEMAS BASADOS EN MICROPROCESADORES

Por

David Andrés Ortiz López

Noviembre 2007

Consejero: Nayda G. Santiago
Departamento: Ingeniería Eléctrica y Computadoras

El consumo de potencia es un factor de gran importancia en sistemas electrónicos basados en microprocesadores. Este problema puede ser resuelto con optimizaciones en términos de hardware y software. Diversas técnicas de optimización de software han sido utilizadas como alternativa para lograr obtener bajo consumo de potencia al diseñar este tipo de sistemas electrónicos, sin embargo estas técnicas deben ser analizadas con métodos estadísticos para obtener conclusiones sólidas sobre su efecto real. En este trabajo, tres técnicas de optimización aplicadas en lenguage de alto nivel son implementadas en un conjunto de benchmarks (MiBench) que representan aplicaciones típicas de sistemas basados en microprocesadores, con el objetivo de analizar si las técnicas de optimizatión tienen o no efecto en el consumo de potencia de distintas plataformas. Para realizar este análisis, métodos de diseño de experimentos (DOE) y análisis de varianza (ANOVA) fueron usados para obtener conclusiones basadas en hechos estadísticos. Los resultados mostraron que no todas las optimizaciones tienen efecto en el consumo de potencia, además algunas técnicas dependen de la plataforma en la cual son ejecutadas.

To my parents, Agueda and Humberto, my brothers Humberto José and Luis Felipe, my sister Diana Carolina, and my friends at the University of Puerto Rico, at Mayagüez.

"*David, do not doubt whether I have been a God's instrument or not... I really am...*"

Scott Tsai

ACKNOWLEDGMENTS

First, I would like to thank my advisor and committee chair Dr. Nayda G. Santiago, and my graduate committee, Dr. Gladys O. Ducoudray and Dr. Nelson Sepúlveda for their help and advice throughout this research. I would also like to thank professor David Gonzalez for his support and assistance in this work.

I want to thank all the people who were part of my life in Puerto Rico. Isabel, you are a magic person, thanks for believing in me and giving me the happiness to continue. Mariana, Juddy, and Miguel, I learned a lot from you, thanks for your support and friendship.

Sandy, you are an extraordinary woman, I hope that the magic of your heart may help other graduate students like me. Thanks for listening to me, and for your help in those difficult moments. Scott, thanks for your help, and for allowing me to see life from another perspective. Agnes, Luis, Roberto and Axel, thanks for all those funny moments, I was an undergraduate again with your company. To my friends, John, Omar, Edith, William, Pablo, Rafa, Iván, Víctor, and Alexis, thanks for sharing with me all those moments during the last year. IICOM members, Michael, Abigail, Aixa, Jomayra, and Debbie, thanks for being part of this beautiful group of friends. WIMS people, thank you all, because you gave me the support to continue as part of this amazing project.

Finally, I would like to thank the WIMS Center at the University of Michigan, Ann Arbor, for the sponsorship of this research. This work has been supported by the Engineering Research Centers Program of the National Science Foundation under Award Number ERC-9986866.

TABLE OF CONTENTS

LIST OF TABLES

Table page

LIST OF FIGURES

LIST OF ABBREVIATIONS

HLL	High-Level Language.
DOE	Design of Experiments.
RAM	Random Access Memory.
ROM	Read Only Memory.
ADC	Analog to Digital Converter.
DAC	Digital to Analog Converter.
SPI	Serial Peripheral Interface.
UART	Universal Asynchronous Receiver Transmitter.
SCI	Serial Communications Interface.
gcc	GNU Compiler Collection.
FFT	Fast Fourier Transform.
IFFT	Inverse Fast Fourier Transform.
ADPCM	Adaptive Differential Pulse Code Modulation.
GSM	Global System for Mobile Communications.
MPEG	Moving Picture Experts Group.
JPEG	Joint Photographic Experts Group.
RISC	Reduced Instruction Set Computer.
CISC	Complex Instruction Set Computer.
BIBD	Balanced Incomplete Block Design.

LIST OF SYMBOLS

I	Current.
α	Switching transition factor.
C	Transistor capacitance.
f	Switching frequency.
V_{cc}	Supply voltage.
I_{leak}	Leakage current.
P_{static}	Static power.
$P_{dynamic}$	Dynamic power.

CHAPTER 1
INTRODUCTION

Power consumption is one of the main design constraints for devices in embedded systems, such as wireless sensors, computer systems, and biomedical devices. The main reasons for analyzing power consumption in these systems is due to limited battery life time, heat dissipation, size constraints, and costs [4, 5]. The power dissipated in embedded systems can be reduced with multiple hardware optimization techniques, such as transistor resizing, low-voltage design techniques and frequency control methods [6]. There is a considerable amount of work done in hardware power optimization, however these techniques are only applied in early design steps [7, 8], such as VLSI design and synthesis.

Embedded software transformations are another way to reduce power consumption, since software is responsible for driving the circuits and components of the system. In terms of software optimization techniques, power dissipation can be reduced with compiler, instruction-level, and source code-level optimization methods. Most of the work done to reduce power consumption has been oriented to compilers optimization [9, 10], where several techniques have been created and incorporated into compilers [11, 12].

Source code and instruction-level optimizations appear as an alternative in low power consumption analysis [13–15]. Although instruction-level optimizations give excellent results with respect to low power consumption, source code optimizations have advantages in terms of portability, readability, and maintenance [16, 17]. Some

studies done in embedded software optimization have shown that source code optimization techniques tend to diminish power consumption [18, 19].

In this research, we investigated the impact of source code-level optimizations on the power consumption of different embedded platforms. We have selected a set of benchmarks to perform our study, and design of experiments (DOE) methods were used to track causes and effects of the power consumption of the system. We have shown that some source code-level optimizations have effect on the power consumption of embedded systems, however it is important to base the evaluation of the effects on statistical methods in order to attain robust conclusions. This document is organized as follows. In section 2, related work of software optimization techniques for embedded processors is discussed. Section 3 illustrates the methodology implemented. Section 4 shows the results obtained and the analysis performed. Finally, conclusions are presented.

CHAPTER 2
PREVIOUS WORK

The power consumed by a digital CMOS technology-based system can be considered as static or dynamic. Static power consumption is caused by leakage currents that appear when transistors are in cut-off or triode mode. Although the voltage at the gate of the transistor is lower than the threshold voltage, there is still current flow through the transistor from drain to source (subthreshold). Equation 2.1 describes static power consumption [9].

$$P_{static} = V_{supply} \cdot I_{leak} \tag{2.1}$$

where P_{static} is the static power dissipated by the system, V_{supply} is the supply voltage, and I_{leak} is the leakage current.

Dynamic power is caused by changing states when transistors switch operational modes. In that transition state, there is a period of time where the gate voltage V_G is greater or equal to the threshold voltage V_{th}, producing dynamic transient current through the circuit. Equation 2.2 describes dynamic power [9].

$$P_{dynamic} = \alpha C V^2 f \tag{2.2}$$

where $P_{dynamic}$ is the dynamic power, α is an activity factor related to the number of switching transitions that occur in an integrated circuit, C is the transistor capacitance, V is the supply voltage, and f is the switching frequency.

There are various ways to reduce static and dynamic power, such as lowering supply voltage, transistors resizing, cooling methods, and reduction of threshold voltage levels [7, 9], among others. Most power optimization techniques for embedded systems may be classified into hardware and software methods.

From a designer's perspective, an embedded system is composed of the code and associate hardware. Figure 2–1 shows the system components and the places where power optimization techniques can be applied. We are particularly interested in techniques applied at the highest level of abstraction.

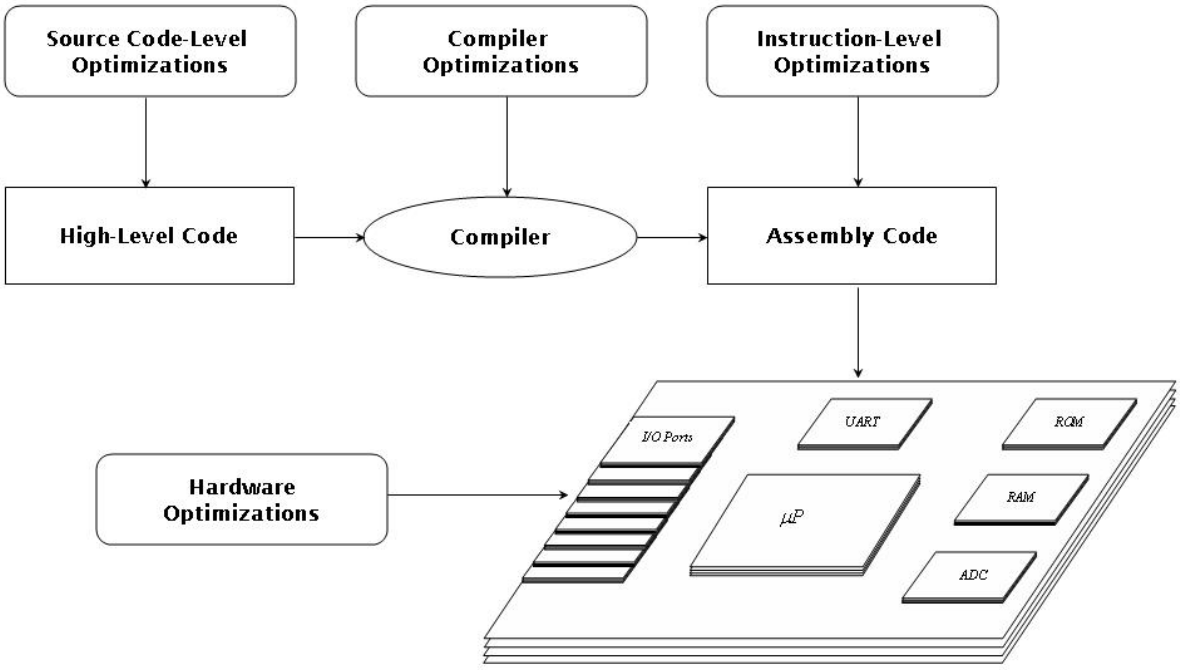

Figure 2–1: Embedded System Components

2.1 Hardware Optimization

Venkatachalam and Franz [9] described optimization techniques to lower power consumption at different hierarchies. At the circuit and logic level, transistor sizes, gates arrangements, clocks, and supply voltages were analyzed. At the interconnection level, some techniques as bus encoding, crosstalk, low swing buses, bus segmentation, and adiabatic buses were considered. In this survey, memory level was stated as one of the main sources of power consumption. Some ways to reduce memory effects on the power system are memory splitting and cache hierarchy structures, among others. At the architecture level, operation modes have been developed as a way to control power consumption when certain parts of the embedded processor are not active.

Other hardware-oriented techniques named dynamic voltage scaling (DVS) and dynamic power management (DPM) were discussed by [6, 8, 20]. In DVS, computation tasks run at different voltages and frequencies, while in DPM, system parts which are not in use are shut off in order to save energy. Jha [8] investigated a set of DVS and DPM scheduling techniques, and concluded that these techniques gave better results when used together. Hong *et al.* [6] studied dynamically variable voltage hardware techniques, where scheduling methods handle the voltage of the system as a variable. Zuquim *et al.* [20] investigated dynamic power management (DPM) techniques in applications with real-time constraints.

2.2 Software Optimization

The consumption of power due to the software running on the microprocessor, can be modified by compiler, instruction, or source code-level optimizations. All of these have advantages and disadvantages depending on the target processor and architecture.

2.2.1 Compiler Optimizations

Compilers optimization were investigated in [10–12, 16, 21] for affecting both performance and power consumption. In terms of performance, Leupers [16] investigated the necessity of generating efficient assembly code, since many compilers are inefficient in terms of code quality, size, and performance. Also, architectural issues of embedded processors are not considered in many cases. The authors generated a survey of methods and techniques for code generation of embedded processors.

For power optimizations, Esakkimuthu *et al.* [21] made a comparison between hardware and software optimizations analyzing cache optimization mechanisms and compiler optimization techniques to lower power consumption. The results they obtained showed that compilers gave better results than cache optimizations in terms of energy savings. The authors concluded that compilers aid to lower power consumption, achieving good results compared with hardware optimizations.

Zambreno *et al.* [10] studied power consumption on portable devices. In this work an analysis of the effect of compiler optimizations on the memory energy was done. By their experiments they concluded that the best optimization approach may not give the best results in terms of power. Also, they observed that function inlining increased the power consumption of the system when applied, unlike loop unrolling, which showed decrease in energy. Ravindran *et al.* [11] proposed an approach for compiler-directed dynamic placement of instructions into a low-power code cache. Ravindran showed that when applying dynamic placement techniques, energy savings may be achieved on the WIMS microcontroller platform.

2.2.2 Instruction-Level Optimizations

Instruction-level optimizations can alter power consumption in embedded systems. These techniques have been studied in [5, 14, 22–25].

The first works in the area of power consumption at instruction-level were the studies done by Tiwari *et al.* [22–24] and Lee [25]. In their work, an instruction-level power model was developed in order to measure the power consumption of a microprocessor. Tiwari *et al.* [22, 24] performed a measurement-based instruction-level power analysis. Some energy optimizations presented were reduction of memory accesses, energy cost driven code generation, and instruction reordering. In [23], the authors made a power consumption analysis of a 32-bit microcontroller. The measurement-based instruction-level power analysis technique proposed by the authors in other studies is used in this work to evaluate the software power behavior of a RISC embedded processor. As conclusions of this work, the authors argued that the power model developed may be applied to any type of processor, providing important information of the power consumption of the system. Also, some suggestions on how to design efficient software in terms of low power were proposed. Lee *et al.* [25] developed a power analysis technique for a digital signal processor. The technique proposed was an instruction-level power model. The results obtained in their experiments showed important energy reductions using the proposed methodology.

Another significant study in this area is the work of Russell and Jacome [5]. Their work is focused on a statistical analysis to know if a parameter can model power consumption. In their work, they performed an instruction-level energy estimation model for embedded processors. They demonstrated the accuracy of such model, the linear dependency of power consumption and frequency in embedded systems, and the need for designers to minimize software execution for low power consumption benefits.

A final study that is important to mention is the work done by Oliver *et al.* [14]. Here, the authors analyzed some factors at the instruction-level that have effect on power consumption, such as cycles, branches, and instruction reordering, among others. With their experiments, they demonstrated that software optimization at

instruction-level is a good approach to minimize energy dissipation in embedded processors.

2.2.3 Source Code-Level Optimizations

Source code-level optimizations for execution time have been studied extensively by [15, 26–32]. Moreover programming style may have an effect on the power consumption of embedded systems. These optimizations were analyzed by [4, 13, 17–19, 32–40].

There are important works in terms of source code-level optimization for performance. Leupers [27], Sharma [17] and Aho *et al.* [28] classified optimizations as machine-independent and machine-dependent. Machine-independent optimizations are implemented at the source code and compiler-level, and they do not take into consideration the target platform. On the other hand, machine-dependent optimizations are implemented at the compiler-level, and are based on a specific processor architecture. Leupers subdivided machine-independent optimizations in four groups known as standard, address code, loops, and function inlining optimizations. Standard optimizations, and address code transformations are applied at the compiler-level, while loop transformations and function inlining, may be applied at the source code-level.

Kraeling [32] proposed different ways to optimize C source code. The topics that the author highlighted in this work were the importance of selecting an appropriate compiler for a target application, the analysis of fixed-point vs. floating-point operations on embedded processors, and different ways to conserve stack and memory resources. Since most of microprocessors do not have floating-point support, C compilers implement floating point operations by predefined support libraries, allowing faster execution.

Gupta *et al.* [29] proposed the evaluation of two address code transformation techniques named scope operation cost minimization complemented and non-linear operator strength reduction. Applying these optimizations they could achieve improvements in terms of performance. Bacon *at al.* [30] made a survey of important loop transformation techniques for C and Fortran. The authors described the goals achieved with each optimization, how to implement them, and some illustrative examples. Leupers and Marwedel [26] presented a methodology to implement optimum function inlining in embedded processors under code size constraints. They showed performance improvements with a low increment of the code size and also suggested that source code optimizations as function inlining should be considered when designing software for embedded processors.

In terms source code optimization for power reduction, Simunic *et al.* [18, 19] classified code optimization techniques in algorithmic, data, and instruction-flow optimizations. Algorithmic optimizations are in the highest stage of the optimization methodology proposed by the authors. In this type of optimizations, a profile of the source code is done in order to know what are the critical procedures, where most time and power is spent. After that, some algorithms are replaced by others with the same functionality, but with more efficient features in terms of time execution, and computation load. Data optimization is at a lower level layer in the optimization process. In this case, the data processed by the algorithms is related directly with some features of the target platform. Instruction-flow optimizations are in the lowest level of the optimization stages described by the authors. In this phase, the goal is to take advantage of specific instructions that exploit the features of the target processor. In this representation, algorithmic optimizations are machine-independent, while data and instruction-flow optimizations are machine-dependent. Also, the authors studied software optimizations and made simulations in order to analyze the power consumption of the SmartBadge ARM-based embedded system.

The tests done by the authors showed that source code optimizations reached better results than compiler optimizations in terms of energy savings. In [19], the authors presented a source code optimization methodology and a profiling tool as a way to optimize for performance and energy. As conclusions of these studies, the authors demonstrated that compiler optimizations are not sufficient for achieving low power consumption.

Weidong *et al.* [34] proposed a technique to optimize embedded software for performance and power through input space adaptive software synthesis and claimed 60% of power reduction in their experiments. Marculescu [37] studied the importance of tools to quantify the effect of performance and power optimizations for embedded software and proposed a technique that selects an optimal number of instructions to be fetched and executed in parallel by the processor to reduce energy consumption.

Dalal *et al.* [13] studied source code-level optimizations in terms of power consumption, analyzing optimum ways to program target applications. The software power components cited by the authors were arithmetic and logic circuits, address and data busses, and memories. One interesting conclusion is that source code optimization techniques should not be used simultaneously, because the improvements on the power consumption may not be significant. Sharma and Ravikumar [17] presented an efficient implementation of the ADPCM codec. In order to obtain more efficient code, optimization techniques were applied at the source code-level and compilation process. The authors made a classification of embedded software optimizations in two groups, structural transformations and machine-dependent optimizations. Chatzigeorgiou *et al.* [4] studied the impact of different software approaches on the power consumption of embedded processors. Since most of the microprocessors manufactured are used in embedded applications, power consumption becomes a problem to solve, because of battery life-time, weight, and heat

dissipation. Some examples of software decisions that may be applied are the utilization of fewer instructions, loop transformations, and functions calling methods. As conclusions, the authors proved that software design decisions may have effect on the power consumption.

Zotos *et al.* [33] explained the main sources of power consumption on processor-based systems, and the impact that source code optimization techniques have on processor and memory power. Dongkun *et al.* [35] proposed an energy monitoring tool with accurate and fast analysis. The tool studied was the SES (Seoul National University Energy Scanner). This takes power consumption data and then assigns it to certain high and low-level language code. With the utilization of this tool, the authors verified the development and simplification of low power embedded software. Peymandoust *et al.* [36] studied the quality of compiled code for embedded systems. They proposed a methodology based on symbolic manipulation of polynomials and energy profiling, thus reducing manual intervention on the optimization process. The methodology proposed automates the process of identifying code sections which may be benefited by algebraic optimizations, and then optimization is done using symbolic techniques. Chung *et al.* [39] presented a model for source code transformations in order to reduce energy cost in embedded software applications. The authors analyzed some low power techniques proposed in literature. As results of this work, the energy consumption of the embedded system was reduced applying the methodology proposed. Choi and Lee [40] studied power consumption on the implementation of G.723.1A/G.729AB on a RISC processor for personal IP telephony devices. The software optimization process had two phases, general techniques and specific approaches. In this implementation, the authors used different source code optimizations and fast algorithms to achieve low power consumption.

2.3 Benchmarking

Lilja [41] defined a benchmark as a program used as reference to make comparisons about the performance of a system. Benchmarks represent real applications which are run on computing systems. Although the concept of benchmarking is widely used in the area of computing systems performance, it can also be used when measuring other metrics such as power consumption. A benchmark has three important features:

1. Benchmark programs are easy to use.
2. Small size.
3. May run on different platforms.

Benchmarking of embedded processors is described in [42]. In this work, Guthaus *et al.* made a study of different benchmarks proposed in literature, such as Dhrystone, Linpack, Whetstone, CPU2, Mediabench, and SPEC2000, among others to evaluate performance of embedded systems. Since most processors are employed in embedded applications, and current embedded systems benchmarks are not reachable to academics, the authors proposed a set of benchmarks designed to measure performance in embedded systems named MiBench. These benchmarks were divided in six groups that represent embedded systems market:

- Automotive
- Consumers
- Office
- Networking
- Security
- Telecommunications

These benchmarks were written in high-level language, making them portable to any platform. MiBench showed different features with respect to SPEC2000, however they are suitable for evaluating performance in embedded systems platforms.

CHAPTER 3
OBJECTIVES AND METHODOLOGY

Our methodology is based in our belief that given a computing system consisting of the software driving a hardware-based embedded system, a transformation can be applied to the software such that power consumption of the embedded system may be affected. We claim that unless statistically sound methods to study cause and effect of these transformations are used, conclusions about whether they are effective on power consumption might not be reached.

This chapter explains the objectives, methodology, and tools employed in this research. Also, theoretical concepts of design of experiments (DOE) techniques, and source code optimization methods are presented.

3.1 Research Objectives

The main objective of this thesis is the analysis of the impact of machine-independent source code optimization techniques on the power consumption of an embedded system, when executing a target benchmark. In order to accomplish this goal, some specific objectives have to be achieved.

- Develop an analysis methodology of embedded software power consumption.
- Measure, and record current values (I) on the microprocessor-based system due to the code running on the embedded platform.
- Examine different source code-level optimization techniques and evaluate their impact in terms of the power consumption of the embedded system.

Three types of machine-independent optimizations are described and a statistical analysis is presented.

3.2 Methodology

Power consumption in embedded systems is an important topic of research, because most electronic devices present restrictions in terms of short battery lifetime. Since a lot of such devices are composed of embedded processors as core of their systems, power is diminished on the processor with hardware and software optimization techniques [9].

The methodology followed in this thesis was divided as follows. First, a selection of three target platforms for studying the impact of source code optimizations on different architectures was done. The platforms chosen for this study were an Intel 8051, a Motorola HC12, and an ARM7TDMI. Then, a set of benchmarks for embedded processors named MiBench [42] was chosen. These benchmarks represent six groups of the market and applications of embedded systems. The groups represented are automotive, telecommunications, office, networking, and security. After that, a profiling of the benchmarks selected was done, in order to know the critical code structures where the highest percentage of time was spent. This allowed us to identify which transformations can be applied to the existing code. Next, several code transformation techniques were selected for simple screening experimentation where a general sense of which techniques had an effect on the power consumption was developed. In our case, a set of loop-oriented optimization techniques designed for improving performance were analyzed in terms of the power dissipation of an Intel 8051 microprocessor-based platform [43]. Three groups of machine-independent source code optimization were chosen [17–19, 26], in order to analyze their impact on the power consumption of the embedded system. Since the analog peripherals of these platforms were not affected by this study, their contribution in terms of power

dissipation did not altere the analysis performed. Finally, instrumentation of the target platforms was done, and design of experiment techniques (DOE) [44] were used in order to perform a statistical analysis.

3.2.1 Target Platforms

To perform the analysis of power consumption on microprocessor-based systems, three representative embedded processors were chosen. Some features of the selected platforms are described in this chapter.

Intel® 8051

The Intel 8051 is an 8-bit CISC core developed for embedded devices. A special feature of the Intel 8051 is a boolean module that allows to perform logic operations. In terms of memory, the microcontroller has 128K of ROM, and 8K RAM memory. In terms of peripherals, the Intel 8051 is equipped with two UARTs, one SPI, a 12-bit and an 8-bit ADC with 8 channels each, two 12-bit DACs, two analog comparators, and five 16-bit timers, among additional features. Figure 3–1 shows the block diagram of the Intel 8051 platform.

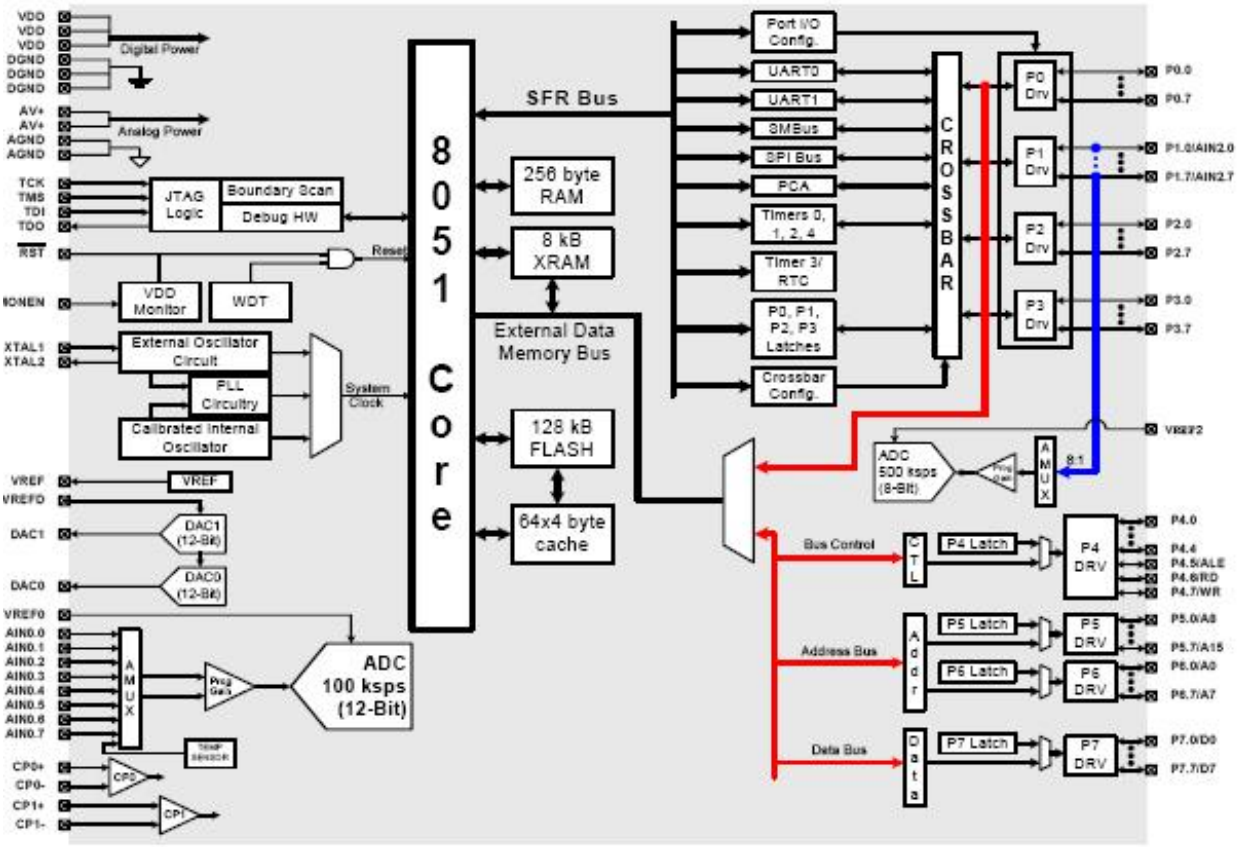

Figure 3–1: Intel 8051 Block Diagram. (Figure courtesy of Silicon Laboratories [1])

Motorola^{TM} HC12

The Motorola HC12 is a 16-bit CISC core for high performance embedded applications. This processor presents features of low power consumption, cost, and code-size effectiveness. The platform has 512K of ROM and 20K of RAM memory. The HC12 microcontroller has three SPIs, a 10-bit ADC with eight channels, and four independent timers, among other characteristics. Figure 3–2 presents the block diagram of the Motorola HC12 platform.

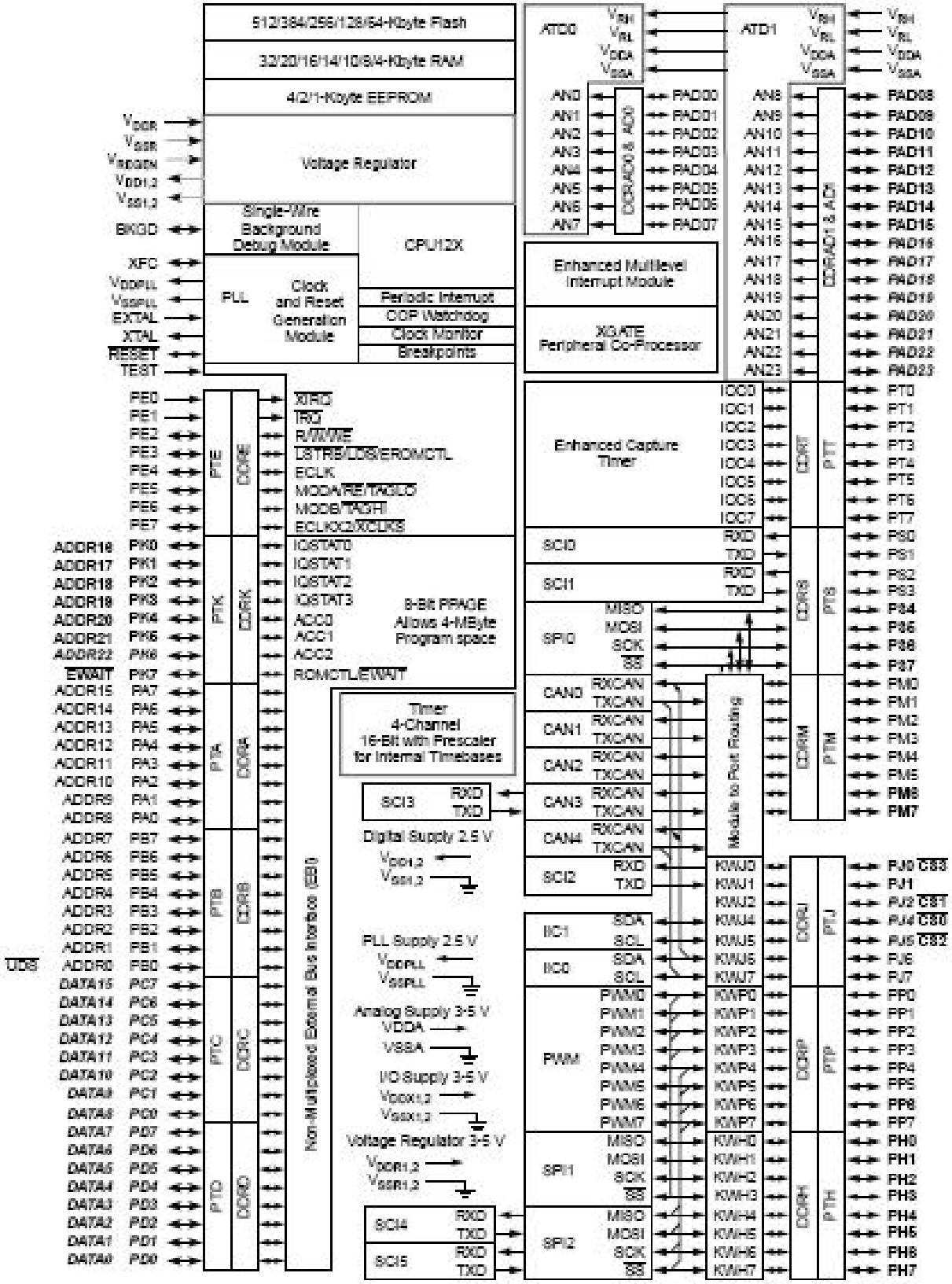

Figure 3–2: Motorola HC12 Block Diagram. (Figure courtesy of Freescale Semiconductor [2])

ARM® ARM7TDMI

The ARM7TDMI is a 32-bit RISC core for high performance purposes. This unit includes characteristics of high speed operation, high throughput, code efficiency, and low costs. This processor has 1M of ROM and 64K of RAM memory. In terms of peripherals, the ARM7TDMI has two SPIs, a 10-bit ADC with 12 channels, a timer with 12 programmable channels, three SCI for serial communication, among others. Figure 3–3 illustrates the block diagram of the ARM7TDMI.

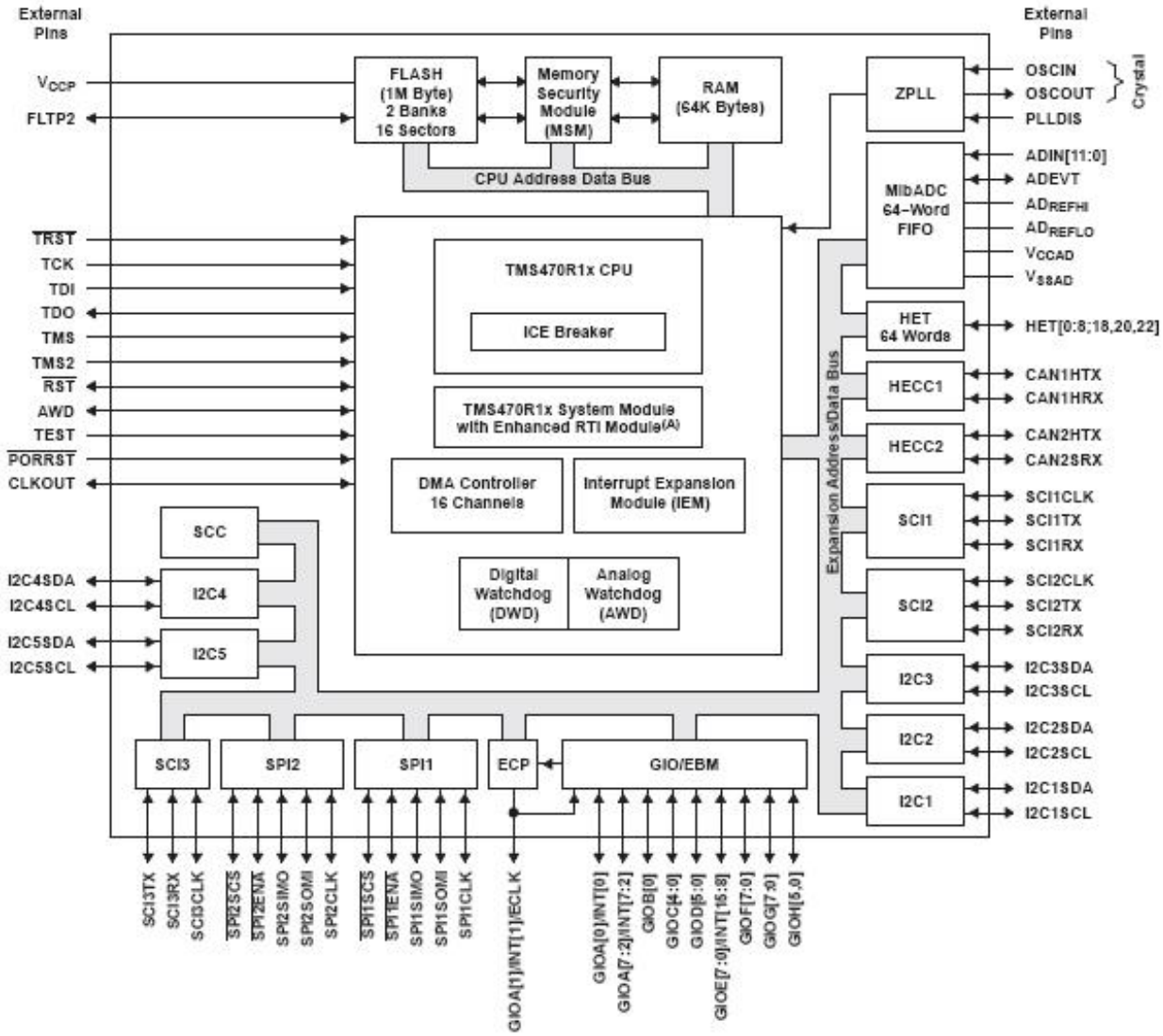

Figure 3–3: ARM7TDMI Block Diagram. (Figure courtesy of Texas Instruments [3])

3.2.2 Benchmarks

In order to analyze the power consumption of the three platforms selected, a set of representative benchmarks for embedded processors applications were selected. Guthaus *et al.* [42] developed a group of benchmarks named MiBench, with the purpose of measuring performance on embedded processors. Since most of traditional benchmarks are designed to make performance analysis on desktop computers, MiBench appears as an alternative to measure performance and other metrics on microprocessor-based systems. Another reason to chose MiBench to perform our power consumption analysis was their availability to academics, allowing the research community to apply them in different studies of embedded software.

Mibench is composed of 35 applications, divided in six groups that represent commercial applications of embedded systems, such as automotive, consumer, office, network, security, and telecommunications. Since these benchmarks were written in C language, they have features of portability and readability, making them adaptable to any embedded platform. A brief description of each one of the six groups of MiBench follows.

Automotive

The automotive benchmarks have the purpose of showing the different uses of embedded systems in industrial and control applications. This group is composed of four programs named *basicmath*, *bitcount*, *qsort*, and *susan*. Following is a brief description of each program.

- *Basicmath* makes mathematical computations that are common in automation, such as the cubic function, integer square root, and angle conversions, among others.
- *Bitcount* counts an array of bits, allowing to test the bit handling features of the embedded processor.

- *Qsort* arranges an array of strings using the quick sort algorithm.
- *Susan* is an image recognition software used for shapes recognition in brain images.

Consumer

The consumer benchmarks characterize a variety of applications based on embedded systems, such as cameras and PDAs, among others. In this group there are programs commonly used in consumer devices, such as *JPEG*, *lame*, *mad*, *tiff2bw*, *tiff2rgba*, *tiffdither*, and *tiffmedian* algorithms. Following is a description of each one of the programs mentioned before.

- *JPEG* is an image compression algorithm used to manage pictures in documents.
- *Lame* is an MP3 encoder, used in media applications.
- *Mad* is an MPEG audio decoder.
- *Tiff2bw* is a color to black and white TIFF image converter.
- *Tiff2rgba* is a color to RGB TIFF image format converter.
- *Tiffdither* reduces the size and resolution of a black and white TIFF image.
- *Tiffmedian* transforms an image to a condensed color setting.

Office

The office benchmarks have been designed to represent algorithms used in office devices, such as printers and fax machines, among other applications. This group is formed by a set of algorithms, such as *ghostscript*, *stringsearch*, *ispell*, *rsynth*, and *sphinx*. Here is a explanation of the algorithms.

- *Ghostscript* is a postcript language interpreter.
- *Stringsearch* looks for words within a document.
- *Ispell* is a spelling checker.

- *Rsynth* is a text to speech processing program.
- *Sphinx* is a speech decoder.

Network

The network benchmarks were developed to exemplify applications of embedded processors in networking devices, such as routers, switches, and modems, among others. The programs that form this group are *Dijkstra*, and *patricia*. This is a description of network benchmarks.

- *Dijkstra* is an algorithm that calculates the shortest path between a pair of nodes in a network.
- *Patricia* is a flexible algorithm for storing, indexing, and retrieving information, used for networking purposes.

Security

The security benchmarks are different algorithms intended for data encryption and decryption. The benchmarks of this group are *blowfish*, *pgp*, *rijndael*, and *sha*. This is an explanation of each one.

- *Blowfish* is a security code based on a 32 to 448 bits key, developed by Bruce Schneider.
- *Pgp* is an encryption algorithm, designed by Phil Zimmerman.
- *Rijndael* is a security code with different key bits (128, 192, 256 bits).
- *Sha* is a hash algorithm for production of digital signatures.

Telecommunications

The telecommunication benchmarks characterize applications where portable systems incorporate internet services and wireless communication. Some programs

of this group include encoding and decoding algorithms, such as *CRC32*, *FFT*, *IFFT*, *ADPCM*, and *GSM*. Descriptions of each benchmark are given here.

- *CRC32* is an algorithm that performs cyclic redundancy check on a file.
- *FFT/IFFT* Fast fourier and inverse fast fourier transform in an array of data.
- *ADPCM* is a type of modulation where an analog signal is represented into a digital code.
- *GSM* is an algorithm that describes the standard for voice encoding in mobile communications.

Selected Benchmarks

For the purposes of this thesis, a subgroup of the benchmarks explained before was chosen. The subset of benchmarks selected consisted of those programs that could fit in the memory of the platforms evaluated. The benchmarks used in this study are shown in table 3–1.

Table 3–1: Benchmarks selected from MiBench for the study of Power Consumption

Benchmark	**Group**
Basicmath	Automotive
Bitcount	Automotive
Qsort	Automotive
Stringsearch	Office
Dijkstra	Network
ADPCM Coder	Telecommunications
ADPCM Decoder	Telecommunications
FFT	Telecommunications

3.2.3 Profiling

To apply source code-level optimization techniques on the benchmarks selected, it was important to obtain a profile of each program in order to identify critical code

structures of the benchmarks. This characterization was done with a profiler provided by the *gcc* compiler. Some statistics achieved with the profiler are shown in table 3–2.

Table 3–2: Profiling of the Benchmarks

Benchmark	**Function**	**% Time**
Basicmath	usqrt	98.01
Bitcount	bit_shifter	31.10
	bit_count	26.83
	ntbl_bitcnt	12.50
	bitcnt	11.59
Qsort	main	88.52
Stringsearch	main	41.67
	init_search	27.78
	str_search	25.00
Dijkstra	dijkstra	62.50
	enqueue	25.00
	print_path	12.50
ADPCM Coder	adpcm_coder	98.60
	main	1.40
ADPCM Decoder	adpcm_decoder	88.12
	main	10.54
FFT	fft	97.53

From these results, optimizations were applied to those parts where the program spent most of the time.

3.2.4 Machine-Independent Optimizations

Source code-level optimization techniques are divided in two main groups named machine-dependent and independent optimizations [17], [27]. Machine-dependent optimizations are based on the features of each architecture, moreover machine-independent optimizations can be compiled to different target platforms. Based on the previous work done by Simunic, Leupers, and Sharma *et al.*, machine-independent optimizations can be classified in algorithmic optimizations [17], [18],

loop transformations [16], and function inlining methods [27].

Algorithmic Optimizations

In algorithmic optimizations, the target program is profiled in order to identify critical code structures where most power and time could be spent and after profiling the code, substitute algorithms that perform the same process are used to replace the original one. This may affect the outcome since the initial algorithm may consume more power or spend more time making certain work. Simunic [19] argues that algorithmic optimizations present good potential to obtain high performance and low power consumption on general and embedded processors. Some algorithmic optimizations are cited next [45], [46].

- Division and remainder
- Conditional execution
- Boolean expressions
- Switch statement
- Variable types

From this group we selected variable declaration to be applied to the benchmarks which is appropriate in this case. Variables declaration are used to replace variable types by others which tend to lower power consumption.

Loop Transformations

Bacon *et al.* made a survey of general purpose-program transformations [30]. The authors emphasized loops because most of execution time is spent on them. Loop transformations are applied to a program in order to exploit the high performance features of the target processor. Some advantages of using loop transformations on embedded software are cache performance improvement, and efficient

utilization of parallel processing features of the platform, among others. Following are cited some common loop transformations.

- Loop interchange
- Loop peeling
- Loop fusion
- Loop fission
- Loop unrolling
- Loop unswitching
- Loop inversion
- Loop invariant code motion
- Loop reversal
- Loop skewing

Loop unrolling is an optimization technique where the body of a loop is copied several times. Since loop unrolling has shown good results in terms of low power consumption for the Intel 8051 platform [43], this was the loop transformation technique chosen to optimize the proposed benchmarks. Table 3–3 shows the results obtained applying a group of eleven loop transformation techniques on different code structures.

Table 3–3: Loop Transformations and Power Consumption

Loop Transformation	**Optimization Phase**	
	$P(mW)$ Non-Opt.	$P(mW)$ Opt.
Statement Reordering	89.37	91.71
Unswitching	90.81	91.26
Loop Peeling	90.63	90.73
Scalar Expansion	93.96	94.23
Loop Fusion	94.05	94.32
Loop Alignment	91.98	90.72
Loop Fission	91.80	94.86
Nested Loops	90.90	90.99
Loop Reversal	93.87	94.95
Loop Interchanging	90.72	90.81
Loop Unrolling	89.28	87.39

Function Inlining

Leupers [16], [26] and Simunic [45] studied function inlining as a machine-independent optimization technique for embedded processors. In function inlining, a function call is replaced by the body of the function in order to increase the performance of the system and reduce the calling overhead (parameter passing, and call and return instructions). However, function inlining tend to increase code size, becoming a limitation for embedded processors with restrictions in terms of memory capabilities.

Table 3–4 shows which source code-level optimization technique can be applied to each benchmark selected.

Table 3–4: Source Code-Level Optimizations applied on the Subset of Benchmarks Selected

Benchmark	**Function**	**Code Structure**	**Source Level Optimizations**		
			Variable Types	Loop Unrolling	Function Inlining
Basicmath	usqrt	Functions			√
Bitcount	bit_shifter bit_count ntbl_bitcnt bitcnt	Algorithms	√		
Qsort	main	Loops Algorithms	√	√	
Stringsearch	main init_search str_search	Loops Algorithms	√	√	
Dijkstra	dijkstra enqueue print_path	Loops Functions		√	√
ADPCM Coder	adpcm_coder main	Functions Algorithms	√		√
ADPCM Decoder	adpcm_decoder main	Functions Algorithms	√		√
FFT	fft	Functions			√

3.2.5 Instrumentation

The power consumption of an embedded system can be calculated evaluating the supply voltage and the average current drawn by the platform [24]. This calculation is given by equation 3.1.

$$P_{system} = V_{cc} I \tag{3.1}$$

where P_{system} is the power consumption of the embedded system, I is the average current and V_{cc} is the supply voltage. In order to determine the power consumption, we measured the current through the embedded system. This measurement was performed connecting an ammeter between the power supply and the microprocessor board, while running non-optimized and optimized versions of the benchmarks within an infinite loop. This was done in order to perform the statistical analysis of the impact of the source code optimization techniques selected on the power consumption.

3.2.6 Design of the experiment and Statistical Analysis

Montgomery [44] defines an experiment as a test where a set of variations are performed on a system in order to observe its response under certain conditions. In design of experiments (DOE), an analysis of a test is done to choose an appropriate design, depending on the factors and response variable that are going to be studied in a process. After identifying an appropriate experiment, statistical analysis of the data is done to reach strong and unbiased conclusions about the data.

Before performing the experiment it is important to plan a completely randomized experimental design. In this type of experiment, the measurements of the test are taken randomly, thus avoiding biased effects due to factors not considered or

nuisance factors. Another aspect to have in mind is a design technique called blocking, where variability from undesirable factors is eliminated from the experiment, evading wrong conclusions. A third characteristic to think about in DOE is replication, where the experiment is repeated in order to obtain an experimental error, which will conclude if the factors of the test have a statistically significant impact on the experiment under study.

A statistical analysis gives information about whether the results of an experiment arose by coincidence, notifying if there is significance effect of the factors on the outcomes of the process. In this study, analysis of variance (ANOVA) was the statistical approach used. When applying ANOVA on an experiment, it is important to assume that the data follows a normal distribution.

A statistical hypothesis is a statement that reveals some inference about certain condition or situation. In statistical analysis, there are two hypothesis, the null hypothesis and the alternative hypothesis, where null hypothesis is assumed to be true. To test an hypothesis the significance level or *p-value* is analyzed. The *p-value* is a measure of how much evidence exists to evaluate a hypothesis. Based on literature [47], *p-values* of 0.01 and 0.05 are commonly employed in statistics, thus depending on the *p-value* chosen the null hypothesis will be accepted or rejected. To reject the null hypothesis for the alternative hypothesis, enough evidence should be present. When the *p-value* obtained is more than the *p-value* selected for the study, the null hypothesis is accepted, otherwise it is rejected. Besides analyzing the *p-value*, it is a good approach to confirm normal distribution assumption of the residuals through a normal probability plot. This is a graphical method useful to determine the normal distribution of the residuals, based on their allocation along a straight line. If the points in the graph are close to the line it can be concluded that the normal distribution is an appropriate model for describing the data.

In order to perform the tests of this research, a guideline for designing experiments proposed by Montgomery [44] was followed. The target problem of this study was the investigation of the actual effect of the optimizations selected on the power consumption of different embedded systems without biased interpretations about results. After this, we selected the factors to be considered in this experiment. In this case, the factors were the optimization phase of each technique and the benchmarks. Then, power consumption of embedded platforms was identified as the response variable of the analysis. To establish the effect of optimization techniques on power consumption, a *p-value* of 0.05 was chosen to accept or reject the null hypothesis. In this study, the null hypothesis was that the optimization phases and the selected benchmarks do not impact the power consumption of the selected platforms. The factors of the experiment are explained next. For our study the optimization phases were non-optimized and optimized, where a benchmark is optimized when at least one optimization technique has been applied on it. The benchmarks were the eight programs selected from MiBench to perform the analysis. Once the factors and the response variable of the experiment were identified, two design of experiments were selected in order to perform a screening analysis of the data. The designs chosen were a balanced incomplete block design (BIBD), and a two-factor factorial design. Following is a description of each design.

A balanced incomplete block design (BIBD) is a special case of a randomized incomplete block design where it is not possible to collect all the combinations of data in each trial. A BIBD is an incomplete block design where any group of two levels of a factor arises the same number of times than any other pair of treatments. This design was selected as a screening of the analysis to be performed.

Factorial designs are those that include two or more variables whose treatments are including related. A two-factor factorial design consists of two factors, each one

with different levels and n replicates. Moreover a two-factor factorial design with one observation per cell is where the design has one single replicate.

After selecting the design of the experiment, the test was performed, the statistical analysis of the data was done, and conclusions and recommendations were obtained.

CHAPTER 4
EXPERIMENTAL RESULTS

This chapter shows the results and the analysis performed on the data obtained. The design of the experiment is illustrated and the ANOVA is analyzed for each case.

4.1 Power Measurements

After performing the experiments, power consumption measures for each platform were obtained. Tables 4–1, 4–2, and 4–3 show the data taken when the benchmarks selected are run, and the optimizations techniques are applied on each platform.

Table 4–1: Benchmarks, Optimization and Power Measurements for the Intel 8051 Platform

Benchmark	**Optimization Technique**	**Optimization Phase**	
		$P(mW)$ Non-Opt.	$P(mW)$ Opt.
Basicmath	Function Inlining	91.17	89.94
Bitcount	Variable Types	94.05	90.76
Qsort	Loop Unrolling	93.42	92.23
Dijkstra	Loop Unrolling	91.44	80.66
	Function Inlining		88.11
Stringsearch	Loop Unrolling	95.4	92.18
	Variable Types		94.66
ADPCM Coder	Function Inlining	93.87	93.58
	Variable Types		93.87
ADPCM Decoder	Function Inlining	103.59	89.433
	Variable Types		87.06
FFT	Function Inlining	94.59	93.55

Table 4–2: Benchmarks, Optimization and Power Measurements for the Motorola HC12 Platform

Benchmark	**Optimization Technique**	**Optimization Phase**	
		$P(mW)$ Non-Opt.	$P(mW)$ Opt.
Basicmath	Function Inlining	670.23	668.61
Bitcount	Variable Types	668.25	666.99
Qsort	Loop Unrolling	668.97	667.71
Dijkstra	Loop Unrolling	667.89	666.81
	Function Inlining		666.72
Stringsearch	Loop Unrolling	668.88	667.53
	Variable Types		668.07
ADPCM Coder	Function Inlining	669.24	668.07
	Variable Types		668.25
ADPCM Decoder	Function Inlining	670.41	668.97
	Variable Types		668.61
FFT	Function Inlining	669.15	703.08

Table 4–3: Benchmarks, Optimization and Power Measurements for the ARM7TDMI Platform

Benchmark	**Optimization Technique**	**Optimization Phase**	
		$P(mW)$ Non-Opt.	$P(mW)$ Opt.
Basicmath	Function Inlining	1361.88	1350.99
Bitcount	Variable Types	982.71	968.85
Qsort	Loop Unrolling	1300.05	1268.01
	Variable Types		1281.06
Dijkstra	Loop Unrolling	973.17	956.88
	Function Inlining		949.77
Stringsearch	Loop Unrolling	1292.94	1268.19
ADPCM Coder	Function Inlining	1292.49	1266.57
	Variable Types		1268.19
ADPCM Decoder	Function Inlining	1293.84	1245.24
	Variable Types		1302.93
FFT	Function Inlining	1283.49	1287.18

Figures 4–1, 4–2, and 4–3 illustrate the data obtained showing a comparison between non-optimized and optimized versions of the benchmarks, when measuring power consumption on each platform. Also a contrast between the optimization techniques applied is done.

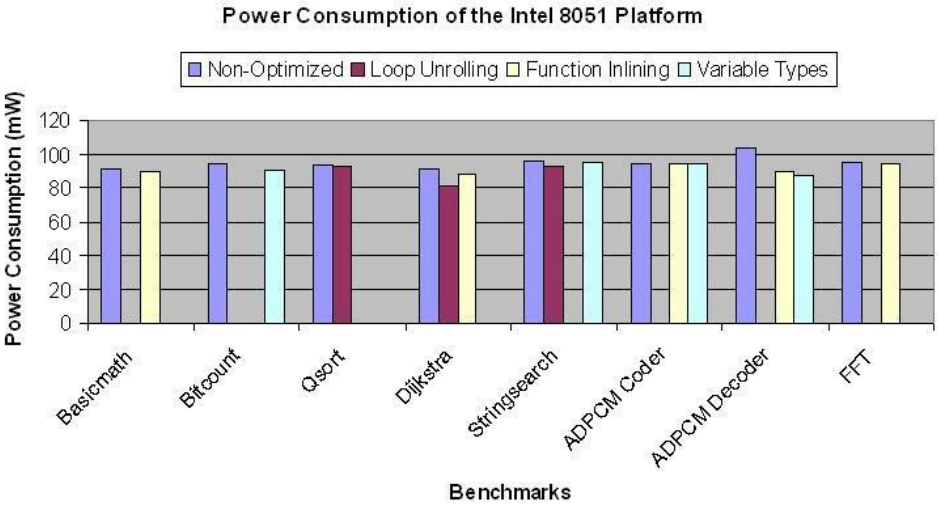

Figure 4–1: Power Consumption Chart for the Intel 8051 Platform

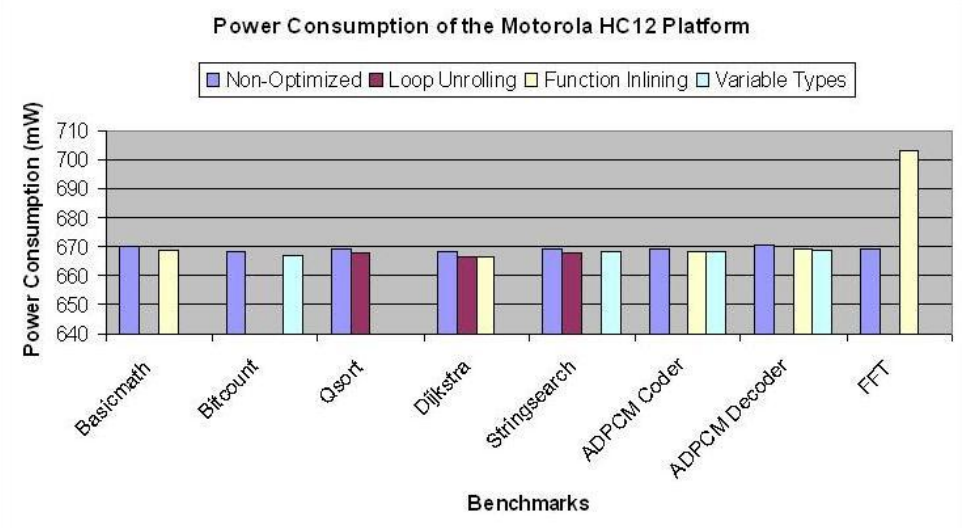

Figure 4–2: Power Consumption Chart for the Motorola HC12 Platform

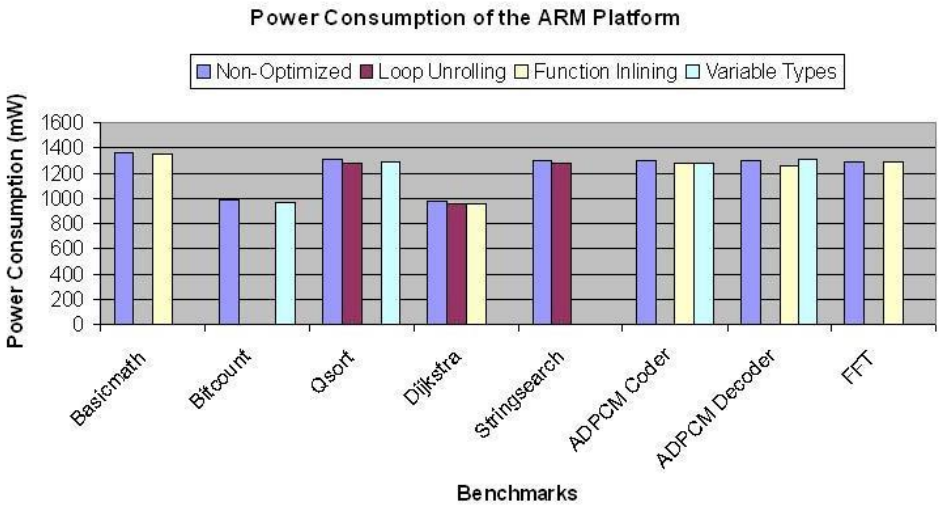

Figure 4–3: Power Consumption Chart for the ARM7TDMI Platform

4.2 Balanced Incomplete Block Design

An initial approach for analyzing the experiments performed was through a balanced incomplete block design (BIBD). This type of experiment is used when the treatment combinations are not run in each block. The factor studied in this case was each optimization technique implemented, and the block was the benchmarks run on the platforms. Tables 4–4, 4–5, and 4–6 show the BIBD for each platform.

Table 4–4: Balanced Incomplete Block Design for Optimization Techniques on the Intel 8051 Platform

Optimization Technique	**Benchmarks**		
	Dijkstra	Stringsearch	ADPCM Coder
Loop Unrolling	80.66	92.18	_
Function Inlining	88.11	_	93.58
Variable Types	_	90.63	90.73

Table 4–5: Balanced Incomplete Block Design for Optimization Techniques on the Motorola HC12 Platform

Optimization Technique	**Benchmarks**		
	Dijkstra	Stringsearch	ADPCM Coder
Loop Unrolling	666.81	667.53	_
Function Inlining	666.72	_	668.07
Variable Types	_	668.07	668.25

Table 4–6: Balanced Incomplete Block Design for Optimization Techniques on the Intel ARM7TDMI Platform

Optimization Technique	**Benchmarks**		
	Qsort	Dijkstra	ADPCM Coder
Loop Unrolling	1268.01	956.88	_
Function Inlining	949.77	_	1266.57
Variable Types	_	1281.06	1268.10

The analysis of variance of each BIBD experiment was performed. From the ANOVA of each experiment, we could observe that the *p-value* of each test was more than 0.05, therefore the initial hypothesis was not rejected, which means that the application of different optimization techniques on the selected benchmarks do not impact the power consumption of the platforms studied. For this reason, we decided to consider other design of experiment to analyze the impact of each technique when the benchmarks are non-optimized and optimized. Figures 4–4, 4–5, and 4–6 show the ANOVA of the BIBD for each platform.

General Linear Model: Power versus Optimization Technique, Benchmarks

```
Factor                  Type   Levels  Values
Optimization Technique  fixed       3  1, 2, 3
Benchmarks              fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source                  DF   Seq SS  Adj SS  Adj MS     F      P
Optimization Technique   2   25.170  26.710  13.355  2.12  0.437
Benchmarks               2   75.017  75.017  37.508  5.95  0.278
Error                    1    6.304   6.304   6.304
Total                    5  106.491


S = 2.51073   R-Sq = 94.08%   R-Sq(adj) = 70.40%
```

Figure 4–4: ANOVA for Intel 8051 Platform

General Linear Model: Power versus Optimization Technique, Benchmarks

```
Factor                  Type   Levels  Values
Optimization Technique  fixed       3  1, 2, 3
Benchmarks              fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source                  DF   Seq SS   Adj SS   Adj MS      F      P
Optimization Technique   2  1.07730  0.13230  0.06615   1.96  0.451
Benchmarks               2  1.15290  1.15290  0.57645  17.08  0.169
Error                    1  0.03375  0.03375  0.03375
Total                    5  2.26395


S = 0.183712   R-Sq = 98.51%   R-Sq(adj) = 92.55%
```

Figure 4–5: ANOVA for Motorola HC12 Platform

General Linear Model: Power versus Optimization Technique, Benchmarks

```
Factor                  Type   Levels  Values
Optimization Technique  fixed       3  1, 2, 3
Benchmarks              fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source                  DF  Seq SS  Adj SS  Adj MS     F      P
Optimization Technique   2   35999   34729   17365  0.25  0.815
Benchmarks               2   30209   30209   15105  0.22  0.833
Error                    1   68457   68457   68457
Total                    5  134665


S = 261.642   R-Sq = 49.17%   R-Sq(adj) = 0.00%
```

Figure 4–6: ANOVA for ARM7TDMI Platform

4.3 Two-Factor Factorial Design with One Observation per Cell

In this type of design, two factors are studied on the experiment. For the tests, the factors are the optimization phase and the benchmarks selected. Due to the experiment's nature, one observation per level on each factor was taken. Hence, a two-factor factorial design with one observation per cell was the design chosen to make the analysis. The results were obtained when analyzing loop unrolling, function inlining, and variable types declaration techniques on the three platforms considered.

4.3.1 Intel 8051 Platform

Tables 4–7, 4–8, and 4–9 show the design of each experiment, and figures 4–8, 4–10, and 4–12 show the normal probability plot for each test performed on the Intel 8051 platform.

Loop Unrolling

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for loop unrolling on the Intel 8051 platform are presented below.

Table 4–7: Factorial Design for Loop Unrolling Technique on the Intel 8051 Platform

Optimization Phase	**Benchmarks**		
	Qsort	Dijkstra	Stringsearch
Non-Optimized	93.42	91.44	95.4
Optimized	92.23	80.66	92.18

General Linear Model: Power versus Optimization Phase, benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
benchmarks          fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS     F      P
Optimization Phase   1   38.46   38.46   38.46  3.01  0.225
benchmarks           2   71.16   71.16   35.58  2.79  0.264
Error                2   25.54   25.54   12.77
Total                5  135.16


S = 3.57354   R-Sq = 81.10%   R-Sq(adj) = 52.76%
```

Figure 4–7: ANOVA for Loop Unrolling on the Intel 8051 Platform

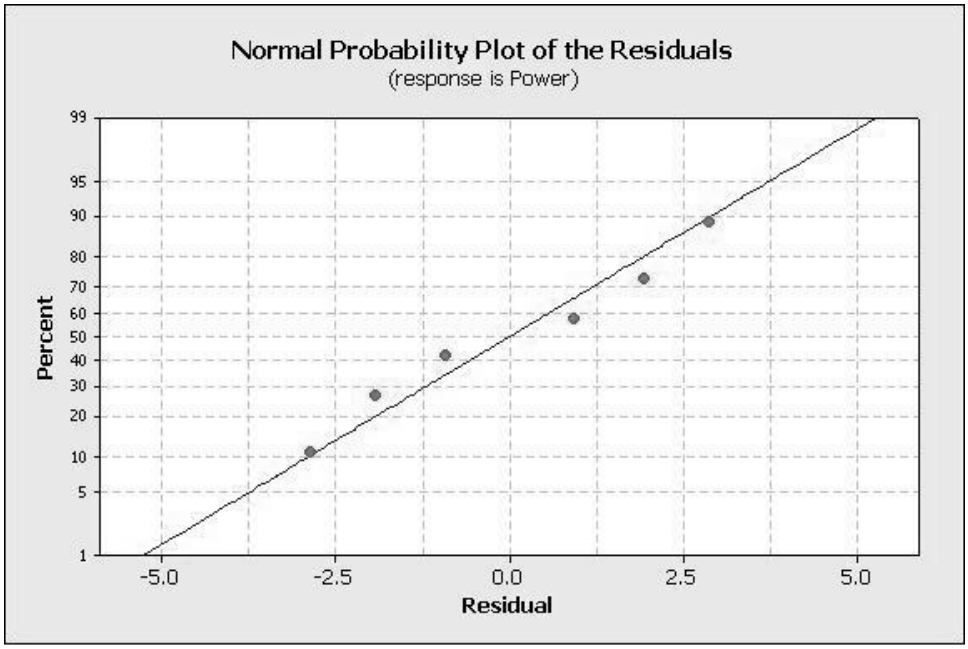

Figure 4–8: Normal Probability Plot for Loop Unrolling on the Intel 8051 Platform

For loop unrolling, the normal probability plot shows that the initial hypothesis assumption is validated. However the *p-value* for this technique on the Intel 8051 platform was > 0.05, therefore the hypothesis is not rejected.

Function Inlining

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for function inlining on the Intel 8051 platform are presented here.

Table 4–8: Factorial Design for Function Inlining Technique on the Intel 8051 Platform

Optimization Phase	**Benchmarks**				
	Basicmath	Dijkstra	ADPCM Coder	ADPCM Decoder	FFT
Non-Optimized	91.17	91.44	93.87	103.59	94.59
Optimized	89.94	88.11	93.58	89.43	93.55

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       5  1, 2, 3, 4, 5


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS     F      P
Optimization Phase   1   40.20   40.20   40.20  2.40  0.196
Benchmarks           4   60.69   60.69   15.17  0.91  0.537
Error                4   66.94   66.94   16.73
Total                9  167.82


S = 4.09073   R-Sq = 60.11%   R-Sq(adj) = 10.26%
```

Figure 4–9: ANOVA for Function Inlining on the Intel 8051 Platform

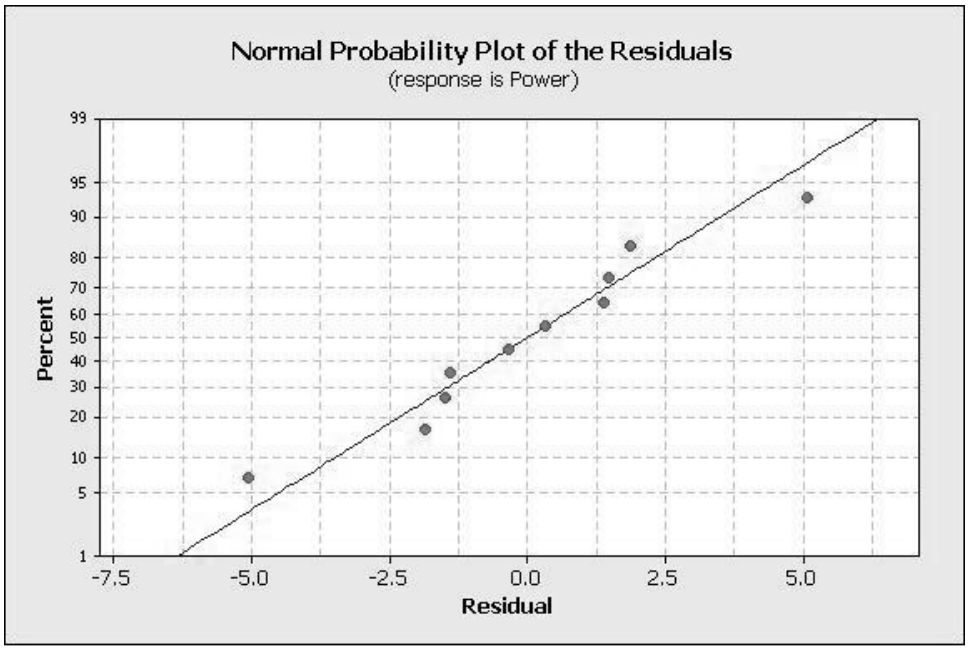

Figure 4–10: Normal Probability Plot for Function Inlining on the Intel 8051 Platform

Here, the normal probability plot shows that the initial hypothesis assumption is validated, but the *p-value* for this technique was > 0.05, hence the hypothesis is not rejected.

Variable Types Declaration

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for variable types declaration on the Intel 8051 platform are presented next.

Table 4–9: Factorial Design for Variable Types Technique on the Intel 8051 Platform

Optimization Phase	**Benchmarks**			
	Bitcount	Stringsearch	ADPCM Coder	ADPCM Decoder
Non-Optimized	94.05	95.4	93.87	103.59
Optimized	90.76	94.66	93.87	87.06

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       4  1, 2, 3, 4


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS     F      P
Optimization Phase   1   52.84   52.84   52.84  1.77  0.275
Benchmarks           3   10.56   10.56    3.52  0.12  0.944
Error                3   89.47   89.47   29.82
Total                7  152.86


S = 5.46099   R-Sq = 41.47%   R-Sq(adj) = 0.00%
```

Figure 4–11: ANOVA for Variable Types on the Intel 8051 Platform

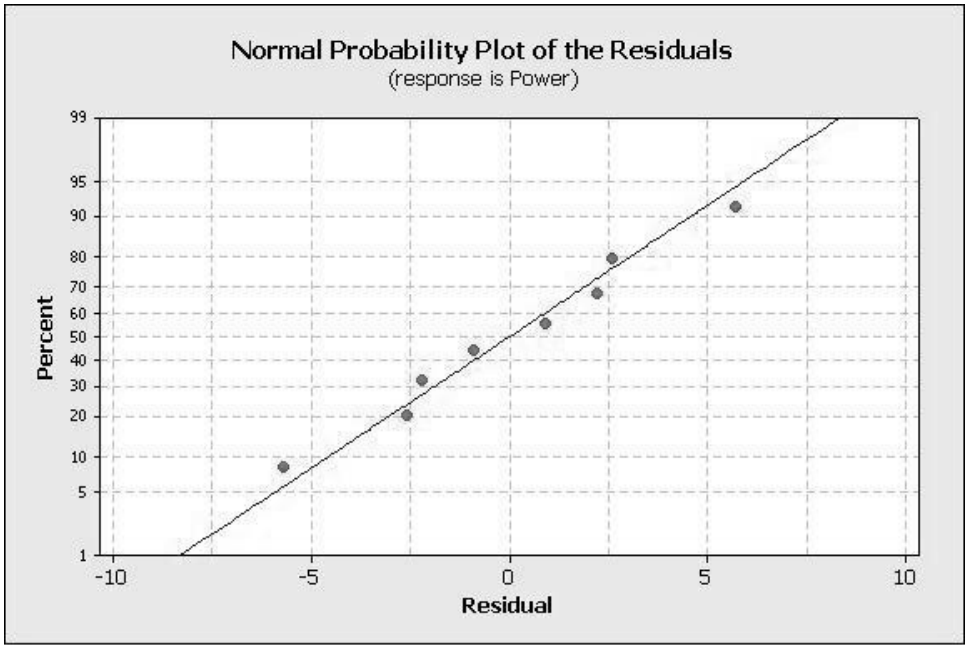

Figure 4–12: Normal Probability Plot for Variable Types on the Intel 8051 Platform

Although the normal probability plot shows that the initial hypothesis assumption is validated for this case, the *p-value* for this technique was > 0.05, and the hypothesis is not rejected.

4.3.2 Motorola HC12 Platform

Tables 4–10, 4–11, and 4–12 show the design of each experiment, and figures 4–14, 4–16, and 4–18 show the normal probability plot for each test performed on the Motorola HC12 platform.

Loop Unrolling

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for loop unrolling on the Motorola HC12 platform are presented below.

Table 4–10: Factorial Design for Loop Unrolling Technique on the Motorola HC12 Platform

Optimization Phase	**Benchmarks**		
	Qsort	Dijkstra	Stringsearch
Non-Optimized	668.97	667.89	668.88
Optimized	667.71	666.81	667.53

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS       F      P
Optimization Phase   1  2.2694  2.2694  2.2694  240.14  0.004
Benchmarks           2  1.1529  1.1529  0.5765   61.00  0.016
Error                2  0.0189  0.0189  0.0094
Total                5  3.4412


S = 0.0972111   R-Sq = 99.45%   R-Sq(adj) = 98.63%
```

Figure 4–13: ANOVA for Loop Unrolling on the Motorola HC12 Platform

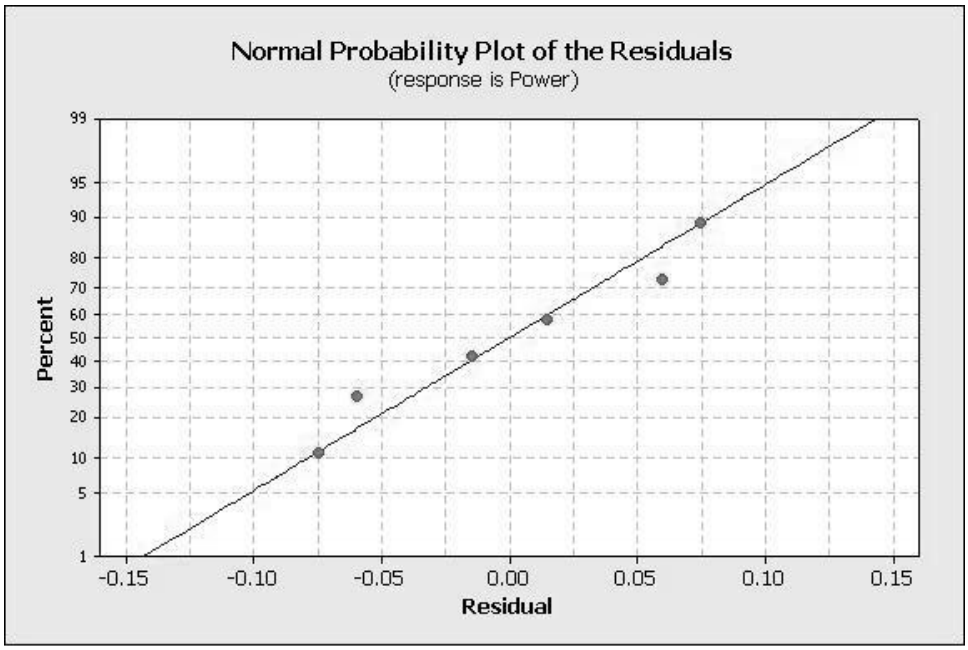

Figure 4–14: Normal Probability Plot for Loop Unrolling on the Motorola HC12 Platform

In this case, the normal probability plot shows that the initial hypothesis assumption is validated. Moreover the *p-value* for this technique evaluated on the Motorola HC12 platform was < 0.05, therefore the hypothesis is rejected.

Function Inlining

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for function inlining the HC12 platform are presented here.

Table 4–11: Factorial Design for Function Inlining Technique on the Motorola HC12 Platform

Optimization Phase	**Benchmarks**				
	Basicmath	Dijkstra	ADPCM Coder	ADPCM Decoder	FFT
Non-Optimized	670.23	667.89	669.24	670.41	669.15
Optimized	668.61	666.72	668.07	668.97	703.08

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       5  1, 2, 3, 4, 5


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS     F      P
Optimization Phase   1    81.4    81.4    81.4  0.65  0.464
Benchmarks           4   488.4   488.4   122.1  0.98  0.507
Error                4   497.9   497.9   124.5
Total                9  1067.7


S = 11.1573   R-Sq = 53.36%   R-Sq(adj) = 0.00%
```

Figure 4–15: ANOVA for Function Inlining on the Motorola HC12 Platform

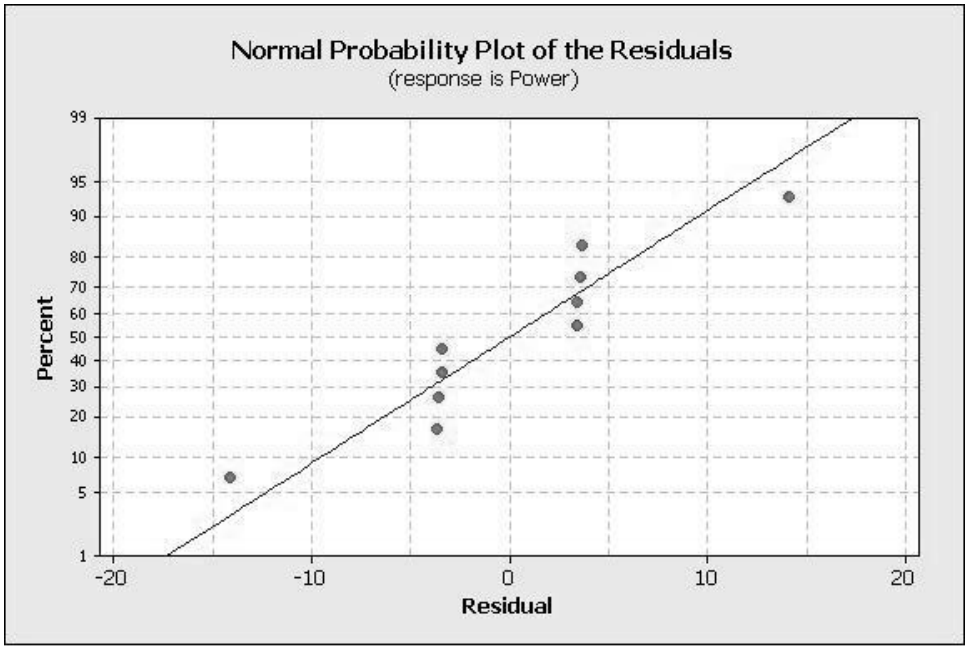

Figure 4–16: Normal Probability Plot for Function Inlining on the Motorola HC12 Platform

Here, the normal probability plot does not show the initial hypothesis assumption, furthermore the *p-value* > 0.05 suggests that the hypothesis is not rejected.

Variable Types Declaration

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for variable types declaration on the HC12 platform are presented next.

Table 4–12: Factorial Design for Variable Types Technique on the Motorola HC12 Platform

Optimization Phase	**Benchmarks**			
	Bitcount	Stringsearch	ADPCM Coder	ADPCM Decoder
Non-Optimized	668.25	668.88	669.24	670.41
Optimized	666.99	668.07	668.25	668.61

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       4  1, 2, 3, 4


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS      F      P
Optimization Phase   1  2.9524  2.9524  2.9524  31.70  0.011
Benchmarks           3  3.6490  3.6490  1.2163  13.06  0.032
Error                3  0.2794  0.2794  0.0931
Total                7  6.8809


S = 0.305205   R-Sq = 95.94%   R-Sq(adj) = 90.52%
```

Figure 4–17: ANOVA for Variable Types on the Motorola HC12 Platform

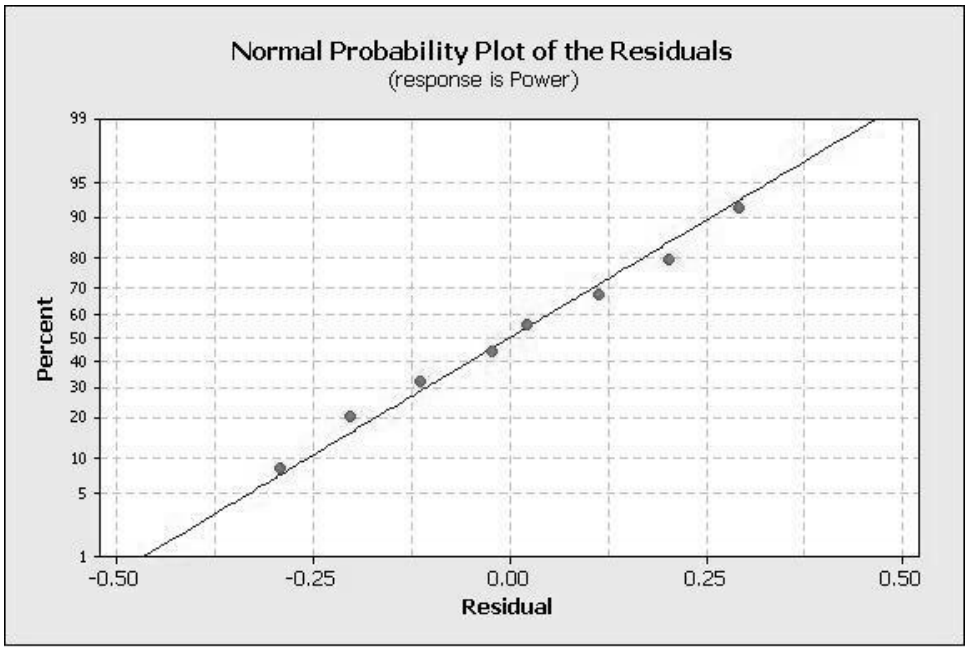

Figure 4–18: Normal Probability Plot for Variable Types on the Motorola HC12 Platform

For variable types declaration on the Motorola HC12 platform, the hypothesis is rejected due to the $p\text{-}value < 0.05$, and the normal probability plot validates the assumption.

4.3.3 ARM7TDMI Platform

Tables 4–13, 4–14, and 4–15 show the design of each experiment, and figures 4–20, 4–22, and 4–24 show the normal probability plot for each test performed on the ARM7TDMI platform.

Loop Unrolling

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for loop unrolling on the ARM7TDMI platform are presented below.

Table 4–13: Factorial Design for Loop Unrolling Technique on the ARM7TDMI Platform

Optimization Phase	**Benchmarks**		
	Qsort	Dijkstra	Stringsearch
Non-Optimized	1300.05	973.17	1292.94
Optimized	1268.01	956.88	1268.19

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       3  1, 2, 3


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS        F      P
Optimization Phase   1     890     890     890    28.65  0.033
Benchmarks           2  134228  134228   67114  2160.44  0.000
Error                2      62      62      31
Total                5  135180


S = 5.57359   R-Sq = 99.95%   R-Sq(adj) = 99.89%
```

Figure 4–19: ANOVA for Loop Unrolling on the ARM7TDMI Platform

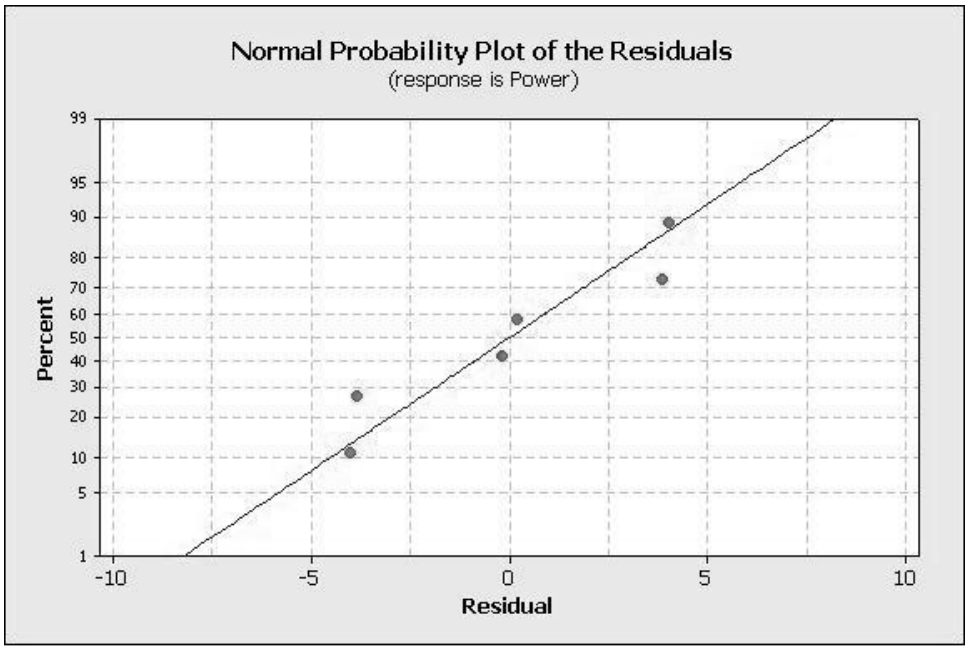

Figure 4–20: Normal Probability Plot for Loop Unrolling on the ARM7TDMI Platform

For loop unrolling on the ARM7TDMI platform, the hypothesis is rejected since the *p-value* is < 0.05.

Function Inlining

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for function inlining on the ARM7TDMI platform are presented here.

Table 4–14: Factorial Design for Function Inlining Technique on the ARM7TDMI Platform

Optimization Phase	**Benchmarks**				
	Basicmath	Dijkstra	ADPCM Coder	ADPCM Decoder	FFT
Non-Optimized	1361.88	973.17	1292.49	1293.84	1283.49
Optimized	1350.99	949.77	1266.57	1245.24	1287.18

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       5  1, 2, 3, 4, 5


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS       F      P
Optimization Phase   1     799     799     799    2.58  0.184
Benchmarks           4  192171  192171   48043  154.95  0.000
Error                4    1240    1240     310
Total                9  194210


S = 17.6085   R-Sq = 99.36%   R-Sq(adj) = 98.56%
```

Figure 4–21: ANOVA for Function Inlining on the ARM7TDMI Platform

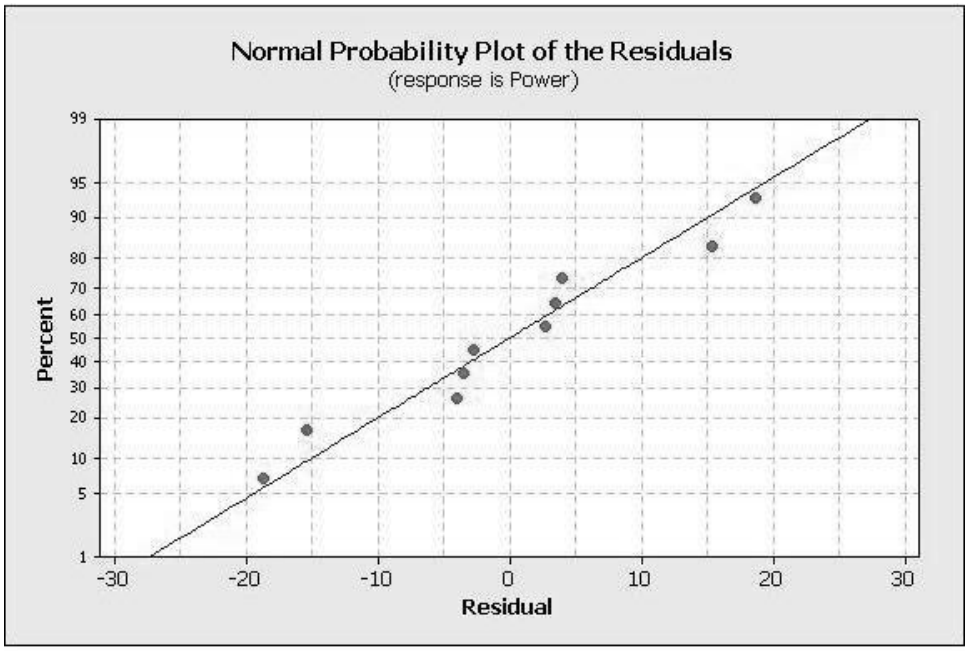

Figure 4–22: Normal Probability Plot for Function Inlining on the ARM7TDMI Platform

In this case, for function inlining technique applied on the ARM7TDMI, the initial hypothesis is not rejected, since the *p-value* is > 0.05. Hence this technique does not have impact on power consumption for the ARM platform.

Variable Types Declaration

The design of the experiment, the normal probability plot of the residuals, and the ANOVA for variable types declaration on the ARM7TDMI platform are presented next.

Table 4–15: Factorial Design for Variable Types Technique on the ARM7TDMI Platform

Optimization Phase	**Benchmarks**			
	Bitcount	Qsort	ADPCM Coder	ADPCM Decoder
Non-Optimized	982.71	1300.05	1292.49	1293.84
Optimized	968.85	1281.06	1268.19	1302.93

General Linear Model: Power versus Optimization Phase, Benchmarks

```
Factor              Type   Levels  Values
Optimization Phase  fixed       2  1, 2
Benchmarks          fixed       4  1, 2, 3, 4


Analysis of Variance for Power, using Adjusted SS for Tests

Source              DF  Seq SS  Adj SS  Adj MS       F      P
Optimization Phase   1     289     289     289    2.67  0.201
Benchmarks           3  148203  148203   49401  457.14  0.000
Error                3     324     324     108
Total                7  148816


S = 10.3955   R-Sq = 99.78%   R-Sq(adj) = 99.49%
```

Figure 4–23: ANOVA for Variable Types on the ARM7TDMI Platform

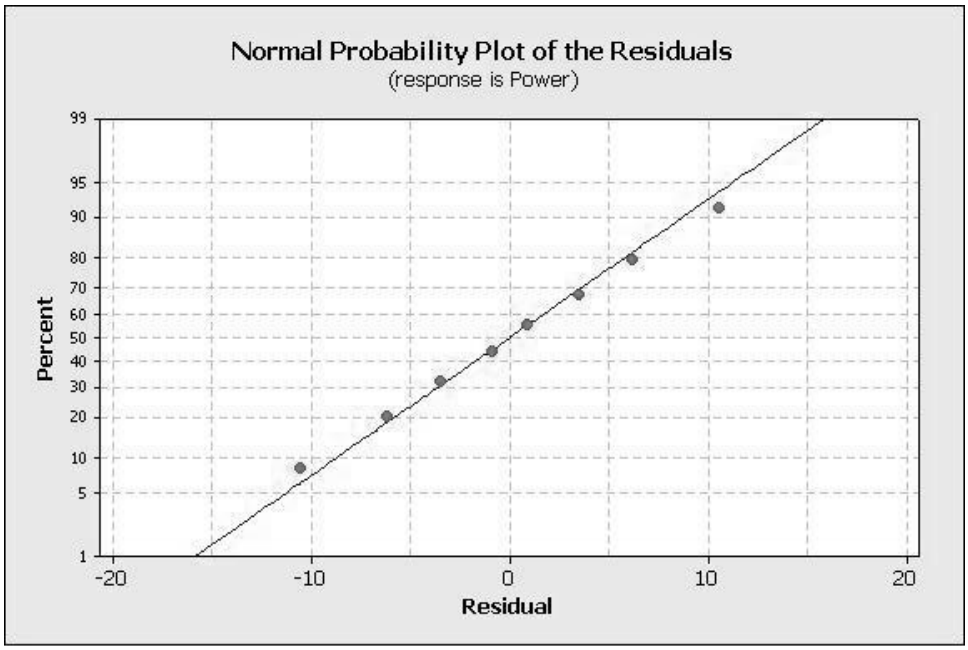

Figure 4–24: Normal Probability Plot for Variable Types on the ARM7TDMI Platform

Finally, for variable types declaration, the normal probability plot validates the initial hypothesis assumption. Moreover the hypothesis is rejected because the *p-value* is less than 0.05.

4.4 Analysis of the Results

From the obtained data from the ANOVA of each one of the experiments mentioned above, we may observe that some techniques have impact on power consumption on certain platforms. Table 4–16 shows a summary of the obtained results.

From the obtained results when performing the statistical analysis, we may conclude that some techniques have impact on power consumption, however it is important to consider the target platform when performing this analysis.

Tables 4–1, 4–2, and 4–3 illustrate power reductions on the platforms when applying the optimization techniques on the benchmarks selected, however it was important to design an experiment and use ANOVA, in order to establish if the impact of the factors considered in the experiment have significant influence on the outcomes. The first impression when analyzing the data obtained was that all optimization techniques have effect on power consumption, nevertheless it was necessary to avoid random results.

From the two-factor factorial design, we could observe that loop unrolling and function inlining had a real impact on the power consumption of the Motorola HC12 platform. Moreover, only loop unrolling presented a significant effect on power consumption for the ARM7TDMI platform, as illustrated in table 4–16.

Table 4–16: Impact of Optimization Techniques on each Platforms: *p-value*

Optimization Technique	**Platforms**		
	Intel 8051	Motorola HC12	ARM7TDMI
Loop Unrolling	No impact	Impact	Impact
Function Inlining	No impact	No impact	No impact
Variable Types	No impact	Impact	No impact

The statistical analysis done showed that depending on the target platforms, an optimization technique may or may not have an impact on power consumption. Based on the results obtained, the Intel 8051 did not show any change in power dissipation, while the Motorola HC12 and the ARM7TDMI showed power reductions when applying the techniques mentioned before.

We may notice that power consumption due to machine-independent optimizations may be related to the hardware features of each platform. The Intel 8051, which did not show improvements in terms of low power consumption, is an 8-bit microprocessor designed for automotive and control applications. Moreover, the Motorola HC12, and the ARM7TDMI platforms are 16 and 32-bit cores respectively, designed for data processing purposes, hence this may have consequences on the optimization techniques evaluated on this work.

CHAPTER 5
CONCLUSIONS

In this work three machine-independent source code optimizations were evaluated on three microprocessor-based platforms, in order to know if power consumption can be diminished when optimizing at the highest level of abstraction. For this, design of experiments (DOE) techniques and statistical analysis (ANOVA) were used to reach statistically significant conclusions about the actual impact of such optimizations on the power consumption of the embedded systems analyzed.

The results obtained showed that power savings due to machine-independent optimizations depend on the target platforms where the optimizations are performed. Moreover, although there were reductions in power consumption in all platforms when applying the three optimizations selected, the analysis performed showed that these results were statistically significant only when applying loop unrolling on the Motorola HC12 and the ARM7TDMI, and variable types declaration on the Motorola HC12 platform.

The main contributions of this work were:

- Evaluation of the power consumption of three commercial platforms when applying source code-level optimization techniques.
- Measurement of power consumption directly from the platform, allowing getting real data from the systems studied, thus avoiding erroneous results.
- Analysis of the data obtained through a statistical method (ANOVA), which led us to reach strong conclusions about the actual impact of the optimization techniques studied on power consumption of the selected platforms.

- A methodology to study power consumption on embedded systems without biased conclusions.

CHAPTER 6
FUTURE WORK

As future work, a study of different ways of analyzing code for embedded systems from the software architecture point of view may be done. Depending on the architecture implemented, different results in terms of power consumption can be reached. Also, a comparison of the obtained results with those given by a power simulator can be done in order to identify which methodology may present better results in terms of the data accuracy.

Other platforms, such as DSPs or microcontrollers with different features in terms of hardware resources can be used to understand the relationship between the architecture and the software running on the processor. Additional benchmarks may be implemented depending on the characteristics of the new platforms considered.

REFERENCES

[1] Silicon Laboratories Inc. *C8051F120 Datasheet*, 2005. http://www.silabs.com/.

[2] Freescale Semiconductor. *MC9S12XDP512 Datasheet*, 2007. http://www.freescale.com/.

[3] Texas Instruments Incorporated. *TMS470R1B1M Datasheet*, 2005. http://focus.ti.com/lit/ds/symlink/tms470r1b1m.pdf/.

[4] A. Chatzigeorgiou and G. Stephanides. Energy issues in software design of embedded systems. *2nd WSEAS International Conference on Applied Informatics, Rethymnon, Crete, Greece*, Jul. 2002.

[5] J.T. Russell and M.F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. *International Conference on Computer Design: VLSI in Computers and Processors*, pages 328 – 333, Oct. 1998.

[6] I. Hong, D. Kirovski, Qu Gang, M. Potkonjak, and M.B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(12):1702 – 1714, Dec. 1999.

[7] F. Gruian. Low power directed system design. *International Symposium on Low Power Electronics and Design*, pages 9 – 14, 2000.

[8] N.K. Jha. Low power system scheduling and synthesis. *IEEE/ACM International Conference on Computer Aided Design*, pages 259 – 263, Nov. 2001.

[9] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37:195 – 237, Sept. 2005.

[10] J. Zambreno, M.T. Kandemir, and A. Choudhary. Enhancing compiler techniques for memory energy optimizations. *Embedded Software. Second International Conference, EMSOFT 2002*, 2491:364 – 381, 2002.

[11] R.A. Ravindran, P.D. Nagarkar, G.S. Dasika, E.D. Marsman, R.M. Senger, S.A. Mahlke, and R.B. Brown. Compiler managed dynamic instruction placement in a low-power code cache. *International Symposium on Code Generation and Optimization*, pages 179 – 190, Mar. 2005.

[12] H. Mehta, R. Owens, M. Irwin, R. Chen, and D. Ghosh. Techniques for low energy software. *ISLPED - International Symposium on Low Power Electronics and Design*, pages 72 – 75, 1997.

[13] V. Dalal and C.P. Ravikumar. Software power optimizations in an embedded system. *Fourteenth International Conference on VLSI Design*, pages 254 – 259, Jan. 2001.

[14] J. Oliver, O. Mocanu, and C. Ferrer. Energy awareness through software optimization as a performance estimate case study of the MC68HC908GP32 microcontroller. *4th International Workshop on Microprocessor Test and Verification: Common Challenges and Solutions*, pages 111 – 116, May. 2003.

[15] Y. Yingbiao, Y. Qingdong, L. Peng, and X. Zhibin. Embedded software optimization for MP3 decoder implemented on RISC core. *IEEE Transactions on Consumer Electronics*, 50(4):1244 – 1249, Nov. 2004.

[16] R. Leupers. Code generation for embedded processors. *The 13th International Symposium on System Synthesis*, pages 173 – 178, Sept. 2000.

[17] A. Sharma and C.P. Ravikumar. Efficient implementation of ADPCM codec. *Thirteenth International Conference on VLSI Design*, pages 456 – 461, Jan. 2000.

[18] T. Simunic, L. Benini, and G. de Micheli. Energy-efficient design of battery-powered embedded systems. *IEEE Transactions on Very Large Scale Integration*

Systems, 9(1):15 – 28, Feb. 2001.

[19] T. Simunic, G. de Micheli, L. Benini, and M. Hans. Source code optimization and profiling of energy consumption in embedded systems. *International Symposium on System Synthesis*, pages 193 – 199, Sept. 2000.

[20] A.L.A.P. Zuquim, L.F.M. Vieira, M.A. Vieira, A.B. Vieira, H.S. Carvalho, J.A. Nacif, Jr. Coelho C.N., Jr. da Silva D.C., A.O. Fernandes, and A.A.F. Loureiro. Efficient power management in real-time embedded systems. *IEEE Conference on Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03*, 1:496 – 505, Sept. 2003.

[21] G. Esakkimuthu, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. Memory ystem energy: Influence of hardware-software optimizations. *Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000. ISLPED '00*, pages 244 – 246, Dec. 2000.

[22] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437 – 445, Dec. 1994.

[23] V. Tiwari and M. Tien-Chien Lee. Power analysis of a 32-bit embedded microcontroller. *Design Automation Conference. Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95. IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific*, pages 141 – 148, Aug., Sept. 1995.

[24] V. Tiwari, S. Malik, A. Wolfe, and M.T. Lee. Instruction level power analysis and optimization of software. *Proceedings of 9th International Conference on VLSI Design, Bangalore, India*, pages 326 – 328, Jan. 1996.

[25] M. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and low-power scheduling techniques for embedded DSP software. *Fujitsu Scientific and Technical Journal, vol.31:215-229, 1995*, 1995.

[26] R. Leupers and P. Marwedel. Function inlining under code size constraints for embedded processors. *1999 IEEE/ACM International Conference on Computer-Aided Design, 1999. Digest of Technical Papers*, pages 7 – 11, Nov. 1999.

[27] R. Leupers. Code optimization techniques for embedded processors. *Kluwer Academic Publishers*, 2000.

[28] A.V. Aho, R. Sethi, and J.D. Ullman. Compilers - principles, techniques and tools. *Adison-Wesley*, 1986.

[29] S. Gupta, M. Miranda, F. Catthoor, and R. Gupta. Analysis of high-level address code transformations for programmable processors. *Proceedings on Design, Automation and Test. Conference and Exhibition 2000*, pages 27 – 30, Mar. 2000.

[30] D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4):345–420, 1994.

[31] Byeongdo Kang, Young-Jik Kwon, and R.Y. Lee. A design and test technique for embedded software. *Third ACIS International Conference on Software Engineering Research, Management and Applications, 2005.*, pages 160 – 165, Aug. 2005.

[32] M.B. Kraeling. Optimization of C code in a real-time environment. *WESCON/96*, pages 574 – 580, Oct. 1996.

[33] K. Zotos, A. Litke, A. Chatzigeorgiou, S. Nikolaidis, and G. Stephanides. Energy complexity of software in embedded systems. *ACIT - Automation, Control, and Applications*, 2005.

[34] W. Weidong, A. Raghunathan, G. Lakshminarayana, and N.K. Jha. Input space adaptive embedded software synthesis. *Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design*, pages 711 – 718, Jan. 2002.

[35] S. Dongkun, S. Hojun, J. Yongsoo, Y. Han-Saem, K. Jihong, and C. Naehyuck. Energy-monitoring tool for low-power embedded programs. *IEEE Design and Test of Computers*, 19(4):7 – 17, Aug. 2002.

[36] A. Peymandoust, T. Simunic, and G. De Micheli. Low power embedded software optimization using symbolic algebra. *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings.*, pages 1052 – 1058, Mar. 2002.

[37] D. Marculescu. Profile-driven code execution for low power dissipation. *Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000. ISLPED '00.*, pages 253 – 255, 2000.

[38] T.K. Tan, A. Raghunathan, and N.K. Jha. Software architectural transformations: a new approach to low energy embedded software. *Design, Automation and Test in Europe Conference and Exhibition, 2003.*, pages 1046 – 1051, 2003.

[39] Eui-Young Chung, L. Benini, and G. De Micheli. Source code transformation based on software cost analysis. *The 14th International Symposium on System Synthesis, 2001. Proceedings.*, pages 153 – 158, 2001.

[40] Yong-Soo Choi and Gyoo-Soo Lee. Real-time implementation of g.723.1a/g.729ab on a risc processor for personal ip telephony devices. *Proceedings of the Ninth International Symposium on Consumer Electronics, 2005. (ISCE 2005).*, pages 20 – 24, Jun. 2005.

[41] D.J. Lilja. *Measuring Computer Performance, A practitioner Guide*. 2000.

[42] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A free, commercially representative embedded benchmark suite. *IEEE International Workshop on Workload Characterization, 2001. WWC-4. 2001*, pages 3 – 14, Dec. 2001.

[43] D.A. Ortiz and N.G. Santiago. High-level optimization for low power consumption on microprocessor-based systems. *50th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'07)*, pages 1265 – 1268, Aug.

2007.

[44] D.C. Montgomery. Design and analysis of experiments. *Wiley, New York, 6th edition*, 2004.

[45] T. Simunic, L. Benini, and G. de Micheli. Energy-efficient design of battery-powered embedded systems. *International Symposium on Low Power Electronics and Design*, pages 212 – 217, 1999.

[46] ARM Application note 34. Writing efficient C for ARM, ARM DA10034A.

[47] J.R. Turner and J.F. Thayer. Introduction to analysis of variance. *Sage Publications*, 2001.

BIOGRAPHICAL SKETCH

David A. Ortiz was born in Barranquilla, Colombia, in 1980. He received a B.S. in Electronic Engineering from National University of Colombia, Manizales Campus, in 2003. As an undergraduate, he was involved in research with the definition of software requirements for a microwave propagation losses simulator, based on Digital Elevation Models (DEM). He was a student at the University of Puerto Rico, Mayagüez Campus from 2005 until 2007, where he was pursuing a M.S. in Electrical Engineering under the supervision of professor Nayda G. Santiago. During his masters, he was interested in embedded software optimization techniques for low power consumption on microprocessor-based systems. His work was published in the IEEE International Midwest Symposium on Circuits and Systems *(MWSCAS 2007)* held in Montreal, Canada.