ANALYSIS OF LOW-POWER SOFTWARE TECHNIQUES FOR REAL TIME OPERATING SYSTEMS

By

Daniel Ernesto Mera Romo

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

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

May, 2010

Approved by:

Rogelio Palomera, Ph.D Member, Graduate Committee	Date
Andres Díaz, Ph.D Member, Graduate Committee	Date
Nayda Santiago, Ph.D President, Graduate Committee	Date
Mercedes Ferrer, Ph.D Representative of Graduate Studies	Date
Hamed Parsiani, Ph.D Chairperson of the Department	Date

Abstract of Thesis Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ANALYSIS OF LOW-POWER SOFTWARE TECHNIQUES FOR REAL TIME OPERATING SYSTEMS

By

Daniel Ernesto Mera Romo

May 2010

Chair: Nayda Santiago
Major Department: Electrical and Computer Engineering

Power consumption is an important constraint in embedded systems running real time operating systems (RTOS). This study proposes an evaluation of the joint effect of possible factors in the power consumption of RTOS running on small and medium scale embedded systems. Design of experiments techniques (DOE) were used to identify the impact in the power consumption of the system. A case of study is presented with algorithms oriented to dynamic frequency scaling and memory management were applied to FreeRTOS. Experiments allowed us to find relationships between the type of architecture, the workload, and OS algorithms in power reduction.

Resumen de Tesis Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

ANALISIS DE TECNICAS DE SOFTWARE PARA BAJA POTENCIA EN SISTEMAS OPERATIVOS TIEMPO REAL

Por

Daniel Ernesto Mera Romo

May 2010

Consejero: Nayda Santiago
Departamento: Ingeniería Eléctrica y Computadoras

Consumo de potencia es una limitación importante en sistemas empotrados corriendo sistemas operativos tiempo real (RTOS). Este estudio propone una evaluación del efecto conjunto de posibles factores de interés en el consumo de potencia de RTOS corriendo en sistemas empotrados de pequeña y mediana escala. Técnicas de diseño de experimentos fueron utilizadas para identificar el impacto en el consumo de potencia del sistema. Un caso de estudio es presentado con algoritmos orientados a escalamiento de frecuencia dinámico y manejo de memoria aplicadas al FreeRTOS. Experimentos nos permitieron encontrar relaciones entre el tipo de arquitectura, la carga y las algoritmos de sistema opterativo en la reducción de potencia.

I dedicate this thesis to my family and God.

ACKNOWLEDGMENTS

Thanks to Dr. Nayda Santiago for giving me the opportunity of work with her, sharing her knowledge, and advising me at all times during my studies at UPRM.

Thanks to committee members for their help during the development of this work and for the revisions and comments on this document.

Specially I want to thank Sandy for giving me their friendship, affection, and support during my time at school. Thanks to all my friends, teachers, WIMS people, and housemates.

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

LIST OF FIGURES

Figure page

LIST OF ABBREVIATIONS

RTOS	Real Time Operating System.
DFS	Dynamic Frequency Scaling.
WCET	Worst Case Execution Time.
HLL	High-Level Language.
DOE	Design of Experiments.
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.

LIST OF SYMBOLS

I	Current.
pt	Probability of switching in power transition.
Cl	Loading capacitance.
Vdd	Supply voltage.
t	Time
$fclk$	Clock frequency.
μGC	Micro gas chromatograph.
Pav	Average Power.

CHAPTER 1
INTRODUCTION

Power consumption is an important constraint in embedded systems running real time operating systems (RTOS) due to limited battery life time, heat dissipation of electronic components, size constraints, and costs [2]. RTOS are widely used on the small-scale and medium-scale embedded systems applications [1]. Power consumption can be reduced via various hardware techniques such as transistor resizing, the design of dynamically variable voltage hardware [3], and VLSI techniques for low power and frequency control methods. Another way to reduce power consumption in embedded systems is through algorithms at the software level [4]. Instruction level [5], compiler level [6], operating system level [2] [7], and high level algorithms [8] have been proposed in literature. The related work in chapter 2 shows a brief description of the low power techniques level by level.

When an RTOS runs on an embedded system, several algorithms at the operating system level have been proposed to reduce power consumption such as I/O management, memory management schemes to make an energy efficient memory allocation [9], dynamic power management techniques (DPM) to dynamically scale the voltage and frequency on the hardware, low power modes, scheduling algorithms [10], and energy characterization of embedded RTOS [11] [12]. Our work differs from others since we evaluate the effect of variating simultaneously possible factors of interest: platforms, benchmarks, operating system level algorithms oriented to DFS and memory management, and their interaction in the power reduction of small

and medium scale embedded systems running RTOS. Statistically sound conclusions were obtained with design of experiments techniques (DOE) [13]. Results show that operating system algorithms impact the reduction of power consumption depending of the platform, and the workload, and their interaction. We found that there is significative interaction between all factors in the power consumption of the system. Moreover the operating system level algorithms impacted the power reduction depending of the relationship between architecture and the application. Additional results show that when general applications with diverse characteristics were tested, only certain architectures experienced reduction in power consumption when OS algorithms were applied.

1.1 Significance

Embedded systems are widely used in all aspects of our life: medical equipment, technology, communications, industry, and others. Power reduction allows to obtain benefits in terms of cost, increase of life time of battery in portable devices and the wear of the devices. Low power software techniques are very attractive because it is not necessary to make changes to hardware. Algorithms at the operating system level allow a significant saving in power due to better use of hardware resources, enabling applications to interact better with the hardware. Our study examines the interaction between types of architecture, types of applications and OS algorithms in reducing power in order to generalize on other architectures.

1.2 Contributions

- Our work was directed to medium to small embedded systems, which is different to most works found in literature.
- We have provided a methodology and a circuit to measure power in run time.
- We have evaluated the simultaneous effect of combining factors and algorithms in the analysis of power savings. Our work has found that there are interaction among all factors in the experiments.

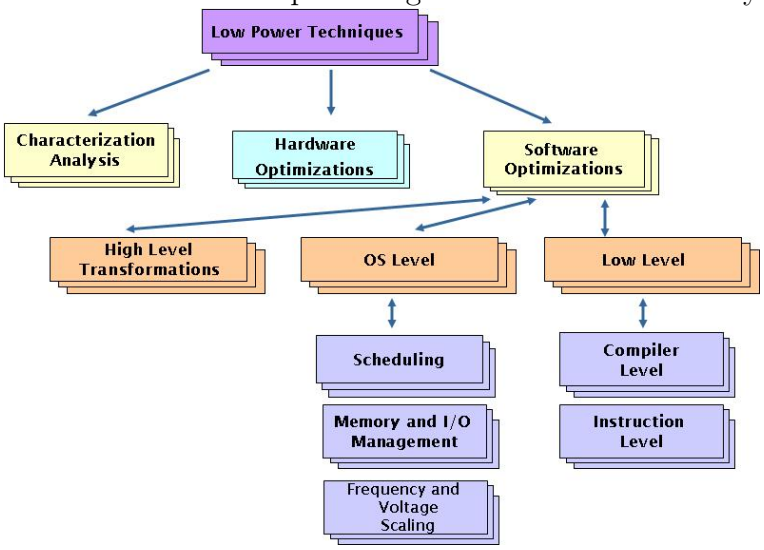
- We have found a relationship among the type of architecture, the type of load, and the RTOS algorithms on power reduction.

This thesis is organized as follows: in Chapter 2 discusses related work in low power techniques for embedded software, Chapter 3 illustrates the methodology implemented, and Chapter 4 shows the results obtained and the analysis. Finally, conclusions are presented.

CHAPTER 2
LITERATURE REVIEW

One of the major constraints of embedded systems is the power consumption which limits the lifetime of the battery. This problem can be approached at different levels of the design of embedded systems: hardware level [14][15], instruction level [16][5], compiler level [6], system level [7][2], and high level transformations [8]. Figure 2–1 shows the classification of the low power consumption techniques for embedded system at the different levels given by [7]. Our interest is oriented to analyze and minimize power consumption at the real-time operating systems (RTOS) that run on microcontrollers based systems.

Figure 2–1: Classification of low power algorithms for embedded systems by [7]



A real time operating system is an embedded software layer that performs functions at system level such as supervision of the application software, abstraction

of hardware layer, provides a mechanism to the processor to run a process doing scheduling, and do context-switch between multiple processes. It allows the system tasks access to resources in sequence. An RTOS is characterized by following a control plan for strictly meet the time deadlines for each task. Moreover it provides a number of services to an embedded system:

- ***Task management:*** is the major task, determines the order in which tasks in the present system will be implemented.
- ***Intertask communication and synchronization:*** allows exchange of information and synchronization between tasks.
- ***Timer:*** time basis for each task.
- ***Device I/O supervisor:*** supervises the protocols for input and output data through the peripherals.
- ***Memory allocation:*** allocates memory to each task and prevents fragmentation of memory.
- ***Other services:*** see table 2–1.

2.1 Power and Energy Analysis of RTOS

An analysis of the power consumption of a RTOS that runs on an embedded system has been done by Dick *et al.* [12]. The authors analyzed the interaction of the RTOS/application on the embedded system power consumption. Their experiments were accomplished with the uCOS-II RTOS running on the Fujitsu SPARClite processor with three software applications as example tests: an antilock braking system (ABS), a market composed of commodity trading agents, and a checksum computation and interfacing with an Ethernet controller with high per-access overhead. As results they developed a embedded system RTOS/application energy profile which can be used to rewrite the application software and use the RTOS in a more energy efficient way. Moreover their study showed that the input/output and intertasks

Table 2–1: RTOS services. Table extracted from [1].

Function	Activities
Basic OS Functions	Process Management, Resources Management, Memory Management, Device Management, I/O Devices subsystem and Network Devices and subsystems management.
RTOS Main functions	Real-time Task Scheduling and Interrup-latency control and use of timers and system clocks Process Management, Resources Management, Memory Management, Device Management, I/O Devices subsystem and Network Devices and subsystems management.
Time Management	Time allocation and de-allocation to attain efficiency in given timing constraints.
Predictability	A predictable timing behaviour of the system and a predictable task-synchronization.
Priorities Management	Priorities Allocation and Priorities Inheritance.
IPC Synchronisation	Synchronization of task with IPCs.
Time Slicing	Time-slicing of the processes execution.
Hard and soft real-time operatibility	Hard real-time and soft real-time operations [Hard real-time means strict adherence to each task schedule. Soft real-time means that only the precedence and sequence for the task-operations are defined.]

communication components consume a lot of power in the ethernet and ABS examples. Based on these results a few general guidelines to use a RTOS in a power efficient way were proposed:

- Rewrite the application to avoid unnecessary use of the RTOS scheduler.
- Remove the use redundant synchronization.
- If the power analysis show significant power consumption in the memory, then use others memory management schemes such as uniform block.
- Concentrate on special modes available in the processor, like using adequately the sleep mode.

  In [11] the authors proposed a methodology to analyze the energy overhead of an embedded operating system independently of the running application. They experimented with the I/O drivers and the context switch components. Their methodology is based in the energy characterization at different processor speeds. In the results they noticed that an increment of 10 KHz in the context switch frequency does not affect significantly the overall power consumption of the system. With respect to I/O drivers their experiments showed that when the CPU sends data to the I/O drivers in amount greater that buffer size, the power consumption increase considerably due to bottleneck. The authors also found a considerable amount of energy is consumed in the synchronization process. The conclusion is that the context switch mechanism is energy-efficient, and as suggestion they proposed that RTOS should send data into small fragments.

2.2 Algorithms at Hardware Level

Algorithms applied to bus encoding have been proposed for low power consumption. The authors in [17] proposed a strategy named bus-inverter code, which consists in coding the information that will be put in the bus to reduce the number of transitions. The algorithm takes into account the following consideration: the width of the bus is N, then the N/2 measure is compared with the hamming

distance among 2 successive patterns, if the distance hamming is bigger than N/2 then the current address is transmitted with inverse polarity, otherwise the address is transmitted without changes. Their results showed power reduction when they used random patterns in the time. In the case of irredundant codes the code gray was good option.

Venkatachalam and Franz in [17] described low power techniques at the circuit, logic, interconnection, and architectural level. At the circuit level they analyzed transistor sizes and at the logic level, number of gates arrangements and clocks. At the interconnection level, bus encoding techniques, crosstalk, low swing buses, bus segmentation, and adiabatic buses were considered. At the architecture level, hardware design with support of low power modes have been considered to control power consumption when certain parts of the embedded processor are inactive.

Another hardware-oriented techniques named dynamic voltage scaling (DVS) and dynamic power management (DPM) were considered by [18] [19], [20], [7]. In DVS, the operating system runs the tasks at different voltages and frequencies. In DPM, parts of the system as peripherals which are not in use are shut off in order to save power.

2.3 Algorithms at Instruction Level

At the instruction level, the work has been directed at developing a methodology for the power characterization of the instructions set for a particular architecture. Tiwari at el al. [16] [5] describe a alternative to analyze the power consumption at the instruction level. Their approach is based in placing each instruction into an infinite loop and measuring the current consumed using an digital ammeter. The experiments shown that the instructions involving memory access are much more expensive than instructions that involve just register accesses. They propose that a better use of registers will notably reduce the total power consumption in the

system. This work has allowed the development of the others algorithms at instruction level such as *Instruction Reordering* and *Energy Cost Driven Code Generation*. *Instruction Reordering* is focussed in diminishing the switching activity between consecutive instructions. The idea with *Energy Cost Driven Code Generation* is to select instructions based on their energy costs instead of the traditional metrics of code size or running time.

Ana-Maria Badulescu *et al.* in [21] designed a mechanism of fetch predictor with the purpose of eliminating the fetch state of the pipeline instructions which are not needed from a cache line. This mechanism predicts which instructions will be fetched before that they are put in the buffer. The prediction is used to search the useful part of the cache lines saving power. Here they use a cache with the ability for fetch any instruction from cache line, it was possible dividing the cache in subbanks then a control vector is used for fetch the subbanks that contain the required instructions. To eliminate the fetch of the instructions, they built a predicted-mask table, the predicted-mask table is indexed with the control vector, which is a bit mask whose length is equal to the number of instructions in a cache line. Each bit in the mask is used to enable or disable the subbanks with the words to be fetched in the next fetch cycle. The tests were carried out based on the simulator of the architecture MIPS, the simulator models a superscalar processor and with nonblocking caches. A subset of benchmark of SPEC95 is used. The results showed that the saving of power with regard to total power of the processor was of 4.4%.

In [22], Tomiyama *et al.* the authors proposed an low power algorithm to diminish the switching activity in the bus of data between the main memory and the cache. In systems based on microprocessors, the access to memory is one of the main problems in the consumption of energy. The algorithm uses the scheme of graphs DAC (directed acyclic graph) to determine all the possible paths of organization of the instructions. The algorithm determines the path with the less cost and therefore

the minor numbers of transitions due to the great quantity of possible paths. The algorithm uses two methods for a quicker search: the first method avoids redundancy in the search and the second one avoids the number of sub trees to be searched. The experiments were carried out on the SPARC architecture using several benchmarks: compress, laplace, linear, lowpass, sor, wavelet, compress (Unix). The results showed that the algorithm achieves a reduction of up to 28% in transitions on the data bus.

2.4 Algorithms at Compiler Level

In [23], Zambreno *et al.* analyzed the compiler level algorithms in the power consumption of the memory system on embedded devices. Their work is based in the interaction between compiler algorithms oriented to performance as loop unrolling and function inlining and the power consumption of the memory, the experiments were realized with the SPEC CPU 2000 and MediaBench benchmarks running on the MIPSR10000 processor. They found that performance techniques did not lead to power and energy savings.

In [6], Tiwari *et al.* had the hypothesis that since the switching activity in a circuit is a function of the present inputs and the previous state of the circuit, then the energy consumption of an instruction varies depending of the previous instruction. They proposed a low power reorganization of the instructions and analyzed the power consumption of the memory accesses. The experiments were carried out with the 486DX2 processor and shown that the power consumption of the memory access instructions is significantly high but the instructions reordering does not impact significantly the overall power consumption of the system, it showed an energy savings of only 2%. A suggestion given by the authors is to try to avoid the instructions with memory operands and making better use of the registers.

Another approach to reduce the power in the switching activity is given by Cheng *et al.* In [24], they present some issues involved in obtaining low power consumption for real-time system; they present a power reduction method at the

software level compilation. The idea is to implement a power reduction techniques inter instruction. The estimated power at instruction level is based on a change in the order to execute the sequence of instructions compiled to produce a significant change in the power consumption of the system.

2.5 High Level Transformations

Yingbiao *et al.* in [25] propose software techniques for low power consumption applied to a RISC core MP3 decoder. The idea is to reach high performance y memory space algorithm, to allow that the compiler creates energy efficient assembly code for a specific processor.

Vishal and *et al.* in [4] performed a study of source code and its effect on power consumption of embedded systems. They optimized high level code, and verified code size. They utilized the DPCM speech compression algorithm as vehicle to demostrate the following algorithms at high level: loop unrolling, function inlining, creation of function macros through #define preprocessor directive and transformation in the branching operations. Their experiments were carried out on the AMR 32 bits processor, in the results the branching transformation gave maximum improvements in power.

Li and *et al.* in [8] did an analysis at high level where a transformation-selection algorithm designed by the authors is applied to the source code of the program. The transformation selection algorithm applies loop unrolling and function inlining transformations and determines in which part of the source code to choose the combination and the order of the transformations that obtain the best energy improvement without violating the memory size limit. Experimental results have shown significant improvements (up to 95%) in energy dissipation.

In [26] a study on transformation techniques for high level synthesis of low power systems was done. The purpose of this work is to reduce the switching activity in the datapath through the loop holding technique applied to the input operands. To

evaluate this work a power analysis program called SPA was used. Their work is based on the fact that the consumption of power is proportional to the correlation of the input operands and the switching power diminishes if the correlation among two consecutive groups of input operands to the same functional unit is high. The loop folding algorithm improves the execution time of a loop through the use of the resources. The tests were made with several different order FIR filters. The results showed that is possible to obtain a reduction of power of up to 50%.

C. Chakrabarti in [27] proposes reducing the power consumption applying memory-optimizing loop transformations and a procedure to choose a fit cache and line size to satisfy the necessities of the system as for area, number of cycles and energy consumption. Among the transformations, the following were analyzed: loop reordering, loop fission, loop interchange enabled by loop skewing, loop fission enabled by loop normalization and loop peeling, loop tiling enabled by loop skewing and loop unrolling. The tests were carried out with the MPG2 encoder algorithm and varying the cache and cache line size. The results showed that for constant instruction cache size, the variation in energy due to variation in data cache size is small. In contrast, for constant data cache size, the variation in energy due to variation in instruction cache is significant.

Studies carried out in [28], [29] showed that 50%-80% of the total power is dissipated in the traffic with the memory and 25%-30% in the cache. Of the analysis they showed that an increment in the size of the cache diminishes the miss rate, cycle variation. Moreover the behavior of the power consumption with regard to the cache size is different: The energy consumption increases with the increase in line size for all cache sizes because a larger line size implies a bigger quantity of data to be fetched from main memory.

In [30] the authors propose a survey from the techniques at high level of modeling, estimation, and algorithms for low power consumption. Their research is based

in control-date-flow graph (CDFG) description of the design. The idea is to associate each operation of the graph with the time and therefore the resources that are not carrying out useful task then to disable them. Furthermore this provides an efficient way to manage the resources because if in some moment in the time several tasks are scheduled then the resources are reallocated again. The multiple supply voltage scheduling technique varies the feeding voltage of the circuit to high level; the advantage of this type of algorithm is assigning the appropriate feeding voltage for each task in the (CDFG).

2.6 Operating System Level Algorithms

Swaminathan *et al.* in [31] studied I/O device scheduling for hard real-time systems with the purpose of putting into sleep state devices that are not in use. They designed an energy efficient algorithm named, *LEDES* for dynamically managing power consumption of I/O devices in a hard real-time system. *LEDES* produces a sequence of sleep/working states for each device. The inputs of the algorithm are the task schedule and future knowledge of device requests. Their experiments showed that *LEDES* can reduce energy consumption by almost a 50%.

Vahdat *et al.* [32] worked in memory management for low power consumption, their study shows that the memory access is one significant cause of power consumption in the RTOS running on embedded systems. The authors propose an power efficient page allocation to prevent fragmentation. They conduced the experiments using two simulators, a trace-driven simulator and a detailed out-of-order execution-driven processor simulator. To evaluate this technique a set of programs from the integer SPEC2000 suite that place higher demands on the memory system was used. The results show that the power-aware page allocation allows a 6% to a 50% improvement in energy.

Voltage and frequency scaling are mechanisms by which energy consumption may be reduced. They are based on the fact that power consumption is a quadratic

function of the voltage and it is proportional to the clock frequency. Several works have followed this approach.

Firsthand [3] Krishna *et al.* worked on voltage scaling in embedded systems and built a voltage-sheduling algorithm based on the earlier deadline first (EDF) sheduling algorithm. In this algorithm the tasks are executed under the assumption of their worst case execution time (WCET), moreover the technique guarantees that all task deadlines will be met. The experiments considered the impact of worst-case task utilization. The results show that for low values of worst-case task set utilization, the system runs at the lowest allowed energy level and an energy saving of 56% is achieved.

Kumar *et al.* in [10] developed predictive strategy, based on workload and the current state of the processor in order to know the time that a task will start to be executed. The system is put in idle mode, or a voltage and frequency scaling can be applied during the intervals of inactivity. The mechanism was implemented on three applications: the Avionics tasks set, INS (Inertial Navigation system) task set, and CNC (computerized numerical control). The results based on simulation showed that the power gain increases, as well as BCET (Best Case Execution Time) gets smaller and the number of missed deadlines increase. Moreover, they noted that when the power conservation level gets bigger the number of tasks missing their deadlines also increases.

More recently, the reduction in power consumption is also focused towards the different enterprise environments, databases, and servers. For example in [33] the authors studied the power consumption at the components level and its variation in a blade server. They developed a hybrid model of hardware and software for the prediction of AC power named *mantis*. First, an analysis is done to find out which parts of the enterprise system consumes much power. Nowadays, two approaches are used to obtain a model for estimating power consumption; estimate the model from

actual power measurements at the hardware level and simulation of the behavior of the system.

The *mantis* model was used to predict power peaks and average power. Their study showed that the processor is the component that consumes more power, much more than the memory, adding to this more than 30-40% of the power is spent on network, disk I/O, peripherals, the power supply, regulators and the rest of the circuitry of the server. The *mantis* model must be calibrated with the appropriate benchmarks running on the server to stress the parts of interest of the system (memory, processing, I/O etc). Based on the output of the *mantis* model, it was possible to predict the power consumption of the server and its components (memory system, processor, I/O and network). The researchers used an AC power meter to measure the real power consumed, furthermore the metrics of measurement used by the model are low-overhead OS utilization and hardware performance counters. These OS metrics refer to CPU utilization and, rate of the I/O requests to the hard drive and network.

Another work focused on power consumption in data center is [34]. In this investigation, the authors seek to coordinate the different approaches to hardware and software levels together. This way of bringing together both approaches, can create conflict, to avoid this, they proposed a mathematical solution, based on control theory. In literature several research works are focused on reducing the average power, the management of peaks power for air conditioning and the costs in the power supply. In this study the researches propose an architecture for coordination and evaluation of peaks of power and the average power handling through the use of hardware and software for any complex enterprise environments. Their study suggests a mechanism of feedback control. Previous work showed a tradeoff between power consumption and performance, these included the tradeoff in their modeling.

The specific options of this approach in order to control power are: frequency and voltage scaling, sleep states, and system shut-down.

The system consists of the following parts:

- ***EC (controller efficiency)*** Optimizes the power consumption per server, the controller monitors the past use and adjusts the state of the processor.
- ***SM (Server Manager):*** Thermal power capping used, if power budget is violated.
- ***EM (Enclosure Manager) and GM (group manager):*** Implement thermal power capping at the blade enclosure and in the rack or data center levels, respectively.
- ***VMC (VM Controller):*** Seeks to put the machines in off power when they are not in use.

Their results based on large-scale simulations utilizing real data traces, showed a 64% in energy savings of 3% in performance degradation, and a 5% of power budget violation. In all scenarios, the coordinated solution resulted much better than the effect of the uncoordinated cases (All solutions working independently).

2.7 Benchmarking

Raj Jain [35] defines benchmarking as the process of performance comparison for two or more system by measurements. The workloads used in the measurements are called benchmarks. Although benchmarking has been focused to evaluate computing system performance, it can be used to analyze other types of metrics such as power consumption in embedded system. That is, stressing the system to simulate a real workload.

The four major considerations in selecting the workload are: the services exercised by the workload, the level of detail, representativeness, and timeliness. The exercised services refer to thought the system as a service provider, for this is necessary that in order for the system provide multiple services, the workload should

exercise as a complete set of services as possible. It is important to know the interface level at which the workload is directed:

1. Arithmetic-logic Unit
2. Central Processing Unit
3. Operating System
4. Applications

Our research is interested in stressing the system at the operating system level, moreover the workload is based in the services provided by the operating system, including the operating system commands and system services. The level of detail means the workload selection in recording and thus reproducing the requests for these services. The most common alternative used is to select as workload the most frequently request service. Representativeness refers to how a test workload should be representative of a real application, that is, the test workload and the real applications must match in the following characteristics: the arrival rate, resource demands, and resource usage profile; the resource usage profile relates to the sequence and the amounts of resources used in a application. The timeliness relates to time constraints, and here is important for the workload to represent the latest usage pattern.

A benchmark has the following important features:

1. Easy to use.
2. Small size.
3. Adaptable to run on different platforms.

In the literature several benchmarks have been proposed to analyze the performance of the embedded processors: sieve, ackerman function, whetstone, LINPACK, Dhrystone, Mediabench, Cpu2, the SPEC benchmark suite, among others. In [36] Guthaus *et al.*, proposed an open source set the benchmarks oriented to measure

performance in embedded systems named *MiBench*. These benchmarks are divided in six groups which represent different embedded systems markets:

1. Automotive
2. Consumers
3. Office
4. Networking
5. Security
6. Telecommunications

These benchmarks were written in C language, making them portable to any platform. MiBench showed different features with respect to SPEC 2000, however, they are suitable for the evaluation of performance in embedded systems platforms. *MiBench* presented a better behavior, including a good distribution of instructions, memory, and available parallelism.

CHAPTER 3
METHODOLOGY

For the purpose of identifying the effect of different factors and their interaction on the power consumption of RTOS running on embedded system, a case study was selected as research methodology.

Our research is divided into the following steps:

1. Three commercial microcontrollers were used for the experiments in order to analyze the operating system level algorithms for RTOS running on small and medium scale embedded system: the LM3S811 from LUMINARY MICRO, the MSP430 from TEXAS INSTRUMENT and, ATMEGA323 from ATMEL. These architectures were chosen because they are respresentative of the world of embedded applications. They allow us to compare differences in bus size and architecture and relate them to power consumption.
2. A set of benchmarks for embedded processors named MiBench [36] was chosen to stress the overall system. These benchmarks represent six groups of embedded applications: consumer, automotive, telecommunications, office, networking, and security. Then the worst case execution time (WCET) of the selected benchmarks was experimentally measured in order to apply the algorithms, next the benchmarks were run on the platforms and a measure of the consumed power of the system was taken.
3. Operating system level algorithms oriented to dynamic frequency scaling and memory management strategies were applied to FREERTOS, an open source, real time operating system designed for microcontrollers.

4. An electronic circuit was designed to measure the run time power and to make the data acquisition through the virtual interface (VI) from host PC.
5. Design Techniques of Experiments (DOE) were used to validate the results and to know whether if system level algorithms (dynamic frequency scaling and memory management strategies) can be used to reduce the power consumption of small and medium scale embedded system. Some of the constraints on these systems are memory and resources limitations.
6. Additional experiments were done with the benchmarks running individually on the platforms in order to find a relationship between type de workload, type of platform and, the OS algorithms in power reduction of the system.

3.1 Target Platforms

To perform the analysis of power consumption on medium and small scale embedded systems, three representative low power microcontrollers were chosen. Some features of the selected platforms are described in this section.

LM3S811

The LMS311 is an 32-Bit RISC architecture based on ARM platform. In terms of hardware power saving it has sleep and deep-sleep modes, on-chip Linear Drop-Out (LDO) voltage regulator, software controls shutdown of individual peripherals, up to 50-MHz of operation. In terms of memory, the microcontrollers has 64 KB single-cycle flash, 8 KB single-cycle SRAM. In terms of peripherals this platform is equipped with, two UARTs, a ADC with four 10-bit channels (inputs), three PWM generator blocks, 1 to 32 general purpose input/output (GPIOs), three 32 bits general-purpose timers, one independent integrated analog comparator, among others features. The applications include factory automation and control, industrial control power devices, building and home automation, brushless DC motors. The figure 3–3 shows the block diagram of the LMS3S811.

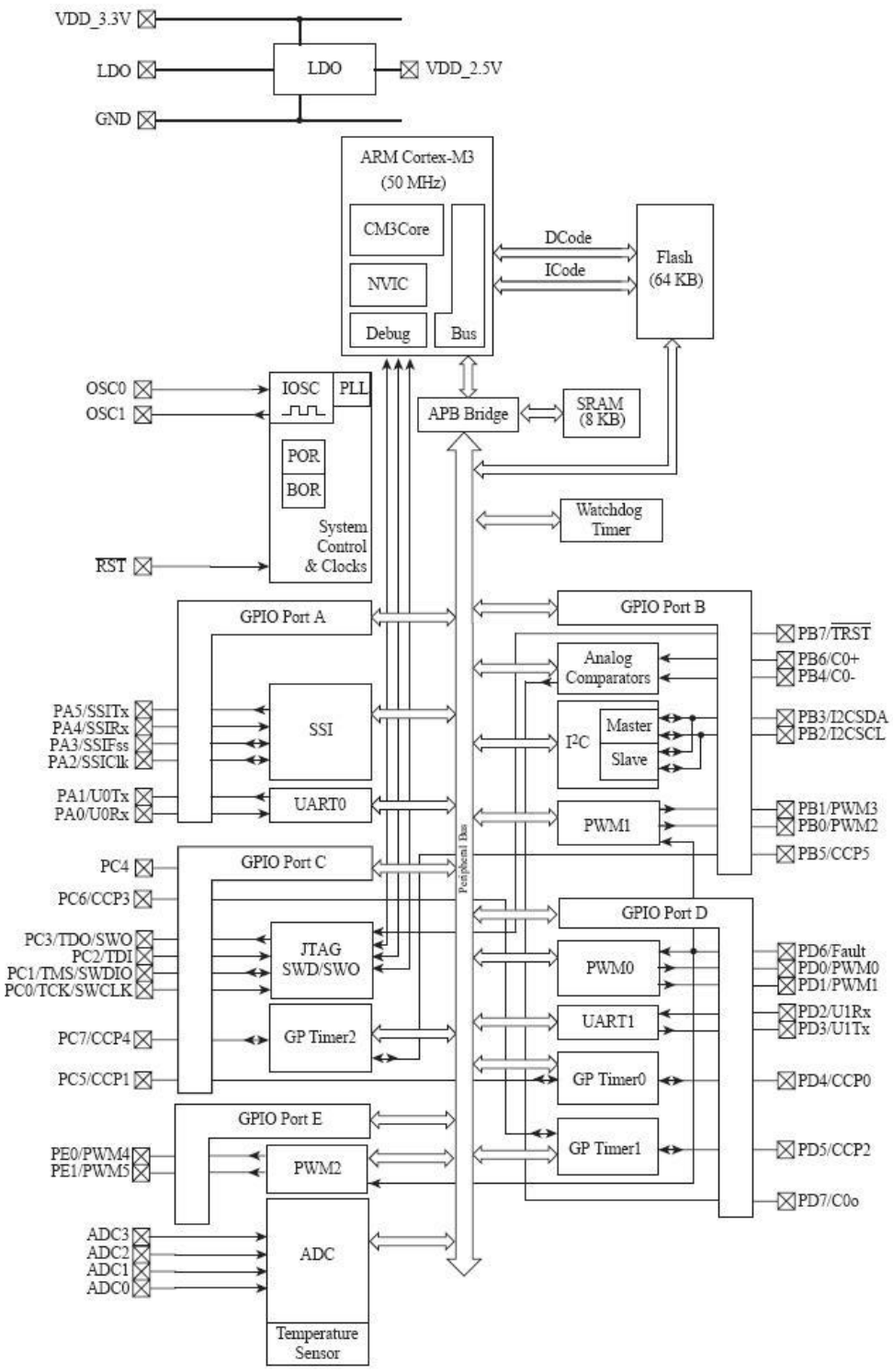


Figure 3–1: LM3S811 Block Diagram. (Figure courtesy from Luminary Micro)

MSP430F149

The MSP430F149 is a low power microcontroller, some features include: 16-Bit RISC architecture, with five power-saving modes. In terms of memory, the MSP430F149 has 60KB + 256B Flash Memory, 2KB RAM. In terms of peripherals it has a 12-Bit A/D Converter, two 16 bits timers, one on chip analog comparator, two serial communication interfaces, an internal digitally-controlled oscillator (DCO), a high frequency crystal oscillator among other characteristics. Typical applications include to capture analog signals of sensor systems converting them to digital values, and process and transmit the data to a host system, industrial control applications such as ripple counters, digital motor control, EE-meters, hand-held meters, etc. Figure 3–2 shows the block diagram of the MSP430F149 platform.

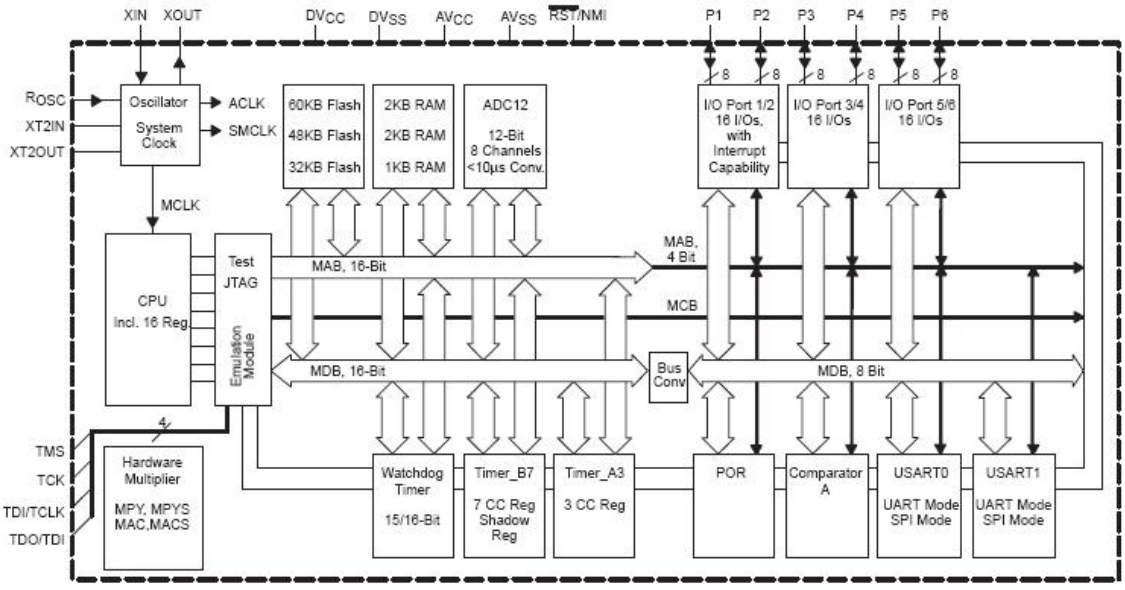


Figure 3–2: MSP430F149 Block Diagram. (Figure courtesy from Texas Instruments)

ATMEGA323

The ATMEGA323 is a high-performance, low-power AVR 8-bit microcontroller with RISC architecture and six sleep modes. In terms of memory it has 32K bytes of flash, 1K byte of EEPROM, 2K bytes of SRAM. In terms of peripherals it presents:

two 8-bit and one 16-bit timer/counters, four PWM Channels, one 8-channel 10-bit ADC, one programmable serial USART, one on-chip analog comparator among others additional features.

All the selected arquitectures have low power features that make them appropriate when designing systems with power constraints.

3.2 Benchmarking

In order to analyze power consumption of the selected platforms and stress the system, a set of benchmarks oriented to embedded applications was chosen. MiBench is a open source group of benchmarks developed by Guthaus et al. [35] and designed to measure performance in embedded processors, consists of 35 embedded applications distributed into six categories that represent commercial applications of embedded systems: automotive and industrial control, network, security, consumer devices, office automation, and telecommunications. These benchmarks are 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 and Industrial Control

This category represents the use of embedded processors in embedded control systems. Here the processors require performance in basic math abilities, bit manipulation, data input/output and simple data organization. Some applications are air bag controllers, engine performance monitors and sensor systems. The algorithms used in this tests are: a basic math test, a bit counting test, a sorting algorithm and a shape recognition program. Following is a brief description of each program.

- ***Basicmath*** performs simple mathematical calculations such as cubic function solving, integer square root and angle conversions from degrees to radians.
- ***Bitcount*** counts the number of bits in an array of integers for testing bit manipulation abilities of the processor.

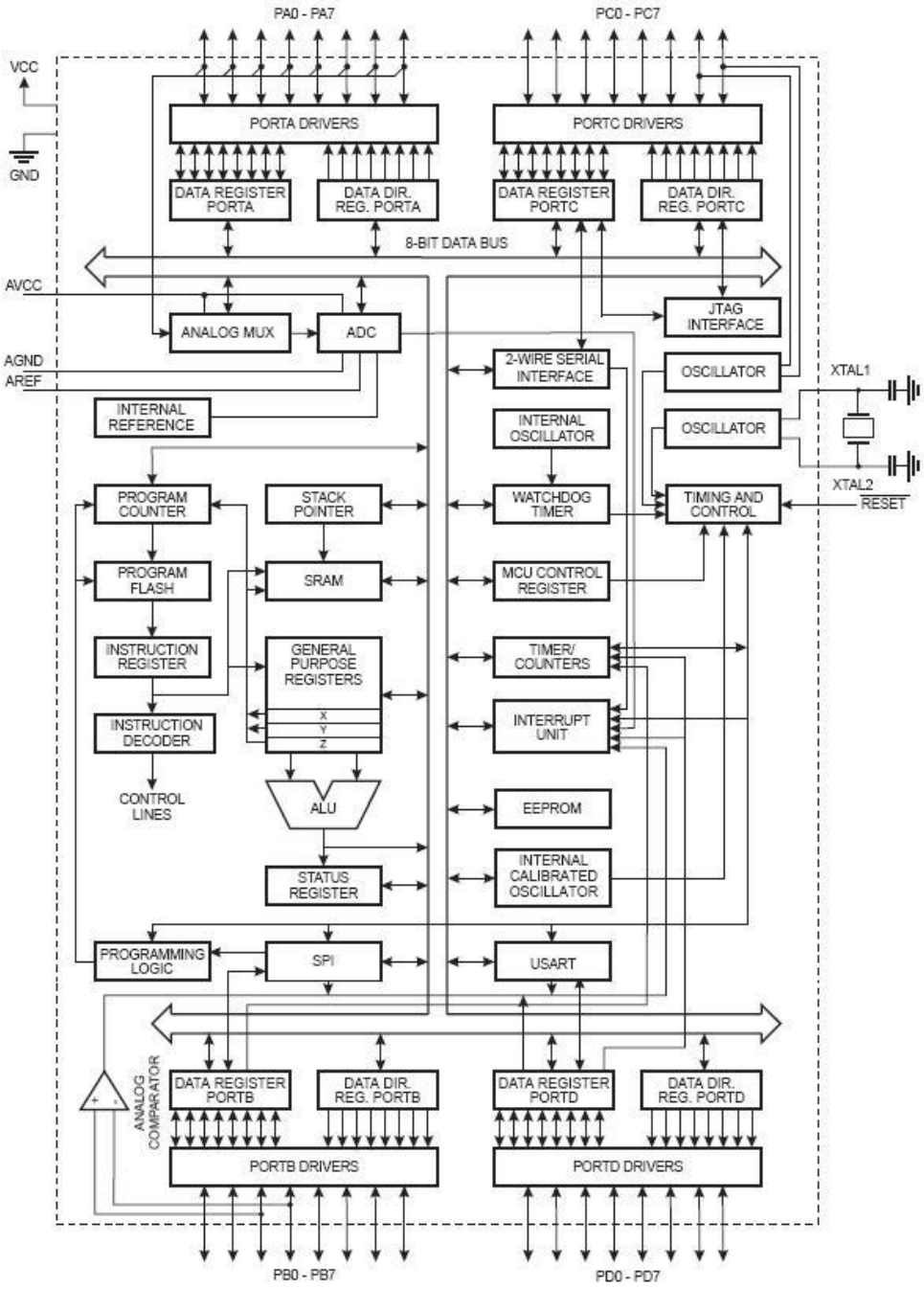


Figure 3–3: ATMEGA323 Block Diagram. (Figure courtesy from ATMEL)

- ***Qsort :*** This test sorts a large array of strings into ascending order using the well quick sort algorithm.
- ***Susan*** is an image recognition software. It is used in studies of the brain.

Consumer

The Consumer Devices benchmarks represents the variety of multimedia applications based on embedded systems like scanners, digital cameras and Personal Digital Assistants (PDAs). The algorithms of this group are: *JPEG*, *tiff2bw*, *tiff2rgba*, *tiffdither*, *tiffmedian*, *lame*, *mad*, and *Typeset* algorithms. Following is a description of each one:

- ***Jpeg encode/decode :*** Compression and decompression of the images in embedded documents.
- ***Tiff2bw*** converts a color TIFF image to black and white image.
- ***Tiff2rgba*** is a converter of color image in the TIFF format into a RGB color formatted TIFF image.
- ***Tiffdither*** reduces the resolution and size of a black and white TIFF image.
- ***Tiffmedian*** transforms an image to a reduced collection of colors.
- ***Lame*** is a MP3 encoder of small and large wave files.
- ***Mad*** is a MPEG audio decoder used in audio applications.
- ***Typeset*** is a tool of typesetting to process HTML documents. It is found in web browser applications.

Office

This group of benchmarks works with text manipulation algorithms and data organization. The following algorithm belong to this group: *Ghostscript*, *Stringsearch*, *Ispell*, *Rsynth*, and *Sphinx*. Below is a brief explanation of the algorithms.

- ***Ghostscript*** is a postscript language interpreter.
- ***Stringsearch*** searches words in a document.
- ***Ispell*** is spelling checker in several languages.

- ***Rsynth*** is a text used in speech synthesis programs.
- ***Sphinx*** is a speech decoder.

Network

The Network benchmarks represent embedded applications of network devices like switches, routers among others. In this group are *Dijkstra*, and *Patricia* benchmarks. Here is a brief description of the benchmarks.

- ***Dijkstra*** calculates the shortest path between every pair of nodes in the network.
- ***Patricia*** is an algorithm utilized to represent routing tables in the network applications.

Security

These algorithms are benchmarks for data encryption, decryption and hashing. The benchmarks of this group are *Blowfish*, *Sha*, *Rijndael*, and *Pgp*. Below is an explanation of each one.

- ***Blowfish*** is a encryption cipher with a variable length key.
- ***Sha*** is utilized for generating digital signatures.
- ***Rijndael*** is security code configurable for 128, 192 and 256 bits.
- ***Pgp*** is a key encryption algorithm that uses digital signatures and the RSA public key cryptosystem.

Telecommunications

This group represents the internet services and wireless communication applications. The benchmarks of this group are:

- ***FFT/IFFT*** is the Fast Fourier Transform and its inverse transform on an array of data.
- ***GSM*** is the standard for voice encoding/decoding in the mobile communication.
- ***ADPCM encode/decode*** is a type of modulation where the signal are represented by digital code.

- ***CRC32 :*** This algorithm detects errors in data transmission.

3.2.1 Selected Benchmarks

The selected benchmarks for our study were: bitcount, basicmath, and dikstra algorithm where the higher and lower priorities were assigned to bitcount and dijkstra respectively. The FreeRTOS ran simultaneously the benchmarks as tasks. The FreeRTOS simultaneously ran the benchmarks.

In embedded applications there are computation intensive, control intensive and I/O intensive applications. Where control intensive programs will have a much larger percentage of branch instructions, computation intensive applications will have a larger percentage of integer or floating point ALU operations, and I/O applications will have a larger percentage in manipulate transference of data. Bitcount and basicmanth are categorized like computation intensive applications, therefore they have many ALU operations. Dijsktra algorithm belong to I/O intensive applications since it performs many I/O control and memory operations. The previous benchmarks were chosen due to constraints in memory of the target platforms and the need to stress the overall system at computation, memory and I/O level. This gave us a better understanding of the behavior of the algorithms in the total power consumption of the system.

The uGC code

The uGC (micro gas chromatograph) is a project of the WIMS ERC; The WIMS ERC was created by the University of Michigan, Michigan State Unviersity, and Michigan Technological University with the goal of combining micro-power circuits, advanced packaging, and sensors into state of the art applications [37]. The micro gas chromatograph is a portable chromatograph used in analytic chemistry used for separating and analyzing compounds that can be vaporized without decomposition. The portable characteristic allows to minimize human exposure to

hazardous scenarios. The uGC helps to identify a compound. Other features include, real-time analysis, highly accurate measures and low power consumption, and micro electromechanical system (MEMS) technology.

The University of Puerto Rico in Mayagüez (UPRM) is working in the firmware with low power constraints which manages the uGC. It performs the data acquisition, data analysis, and finally send the results to a host computer by implementing wireless protocol. Figure 3–4 shows a description of the uCG Project.

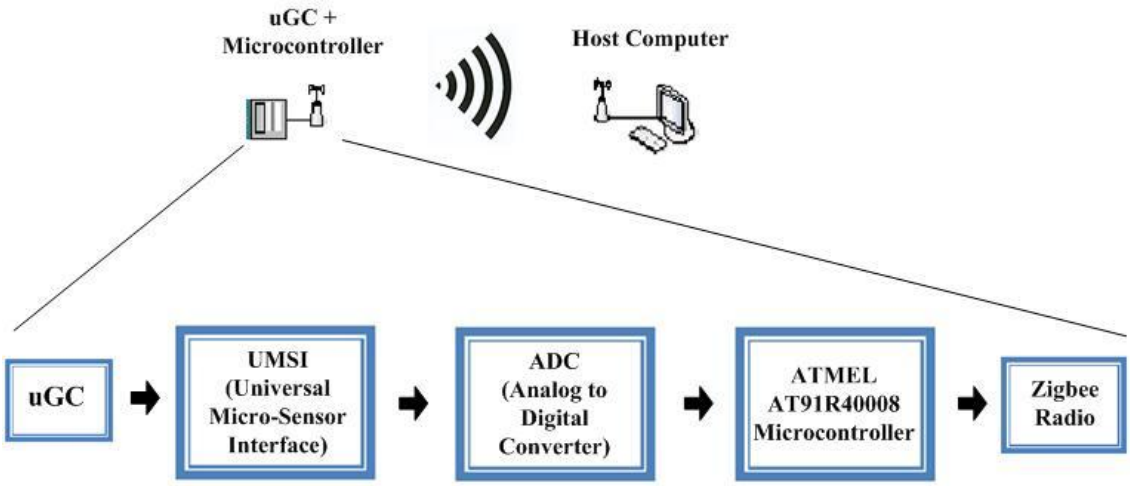


Figure 3–4: uGC System

The uGC firmware is embedded in the ATMEL AT91 commercial microcontroller because it has low power features. The firmware performs data acquisition, data analysis, and send the data via wireless to a host computer. The firmware has the following parts: *Control Code*, *Data Analysis Code*, *Memory Management*, and *Wireless Communication*.

3.3 The FreeRTOS Real Time Operating system

The FreeRTOS is a real time operating system with a scalable mini real time kernel designed specifically for embedded microcontrollers. Some features include preemptive, cooperative and hybrid configuration options, code structure written in C, support for tasks and co-routines, queues, binary semaphores, counting semaphores, recursive semaphores, and mutexes for communication and synchronization between

tasks, or between tasks and interruptions. Furthermore, these features make the FreeRTOS appropriate for our study.

3.4 The Operating System Level Algorithms

The algorithms oriented to DFS chosen were: DFS based on CPU usage, DFS predictive strategy based on WCET, and DFS based on tasks priority [38] [10]. In terms of memory management, static and dynamic memory allocation were considered [39]. Below is a brief description of the algorithms. In [38] the authors proposed a predictive dynamic voltage scaling (DVS) scheduling strategy, for this they have two approaches. The first one is based on assigning to each task a processor clock frequency according to the priority of the task, this is assigned at the task with the lowest priority with a low clock frequency, completing all task's deadlines. The second approach is based on reducing the clock frequency considering process usage, which measures the workload of the active tasks in a determined time. In [10] the authors took an approach based on simulations. A low power predictive scheduling strategy was proposed. The mechanism predicts the execution time of task instances doing an average of the previous instances, here the WCET (worst case execution time) is used to compare if the task will finish its execution before the WCET, then slow down the processor clock frequency (DFS) during the idle time intervals of the CPU.

To calculate the WCET of the tasks we have found in the literature, several ways to do it, through the *static methods* and *measurement-based methods*. We have chosen the *measurement-based methods* because they are good in terms that they have into account the target architecture and they produce WCET estimates more precise-closer to the exact WCET than the bounds from *static methods* [40]. *Measurement-based methods* imply experimental end-to-end measurements of a subset of all possible threads of executions of the tasks. Table 3–1 shows the WCET for the selected benchmarks.

Table 3–1: WCET for selected benchmarks

Platforms	**WCET (miliseconds)**			
	Basicmath	Bitcount	Dikstra	uGC
LM3S811 at (20MHz)	125.2	20.9	44	220
MSP430F149 at (4.9MHz)	162.7	27.1	57.3	160
ATMEGA323 at (2MHz)	420	70	148	950

In terms of memory management algorithms, we chose static and dynamic memory allocation [39]. Dynamic, differs from static memory allocation since runtime searches for a block big enough in the heap, to satisfy the request of store data for a given task.

We have used the previous approach taking into account dynamic frequency scaling (DFS), and tested experimentally the processor clock frequencies which allows for all deadlines to be completed. Dynamic frequency scaling (DFS) is used to adjust the working frequency according to the system workload in order to save the power consumption without degrading the system performance significantly beyond the user tolerance.

3.5 Instrumentation

Dynamic power is consumed due to the switching of gates and is still responsible for the largest percentage of the total power dissipated in current computing devices. The power dissipation in embedded system based on CMOS circuits satisfies approximately by the equation 3.1 [41]:

$$P_{system} \cong ptC_L V_{dd}^2 f_{clk} \tag{3.1}$$

Where pt is the probability of switching in power transition (the activity factor), C_L is the loading capacitance, V_{dd} the voltage supply and f_{clk} the clock frequency.

Given that we measured physically the current variations because the voltage is constant, our instrumentation circuit uses a resistor of 1 ohm in series with the power supply of microcontrollers. The signal was filtered using an instrumentation

amplifier (AD620). The data acquisition was done using a oscilloscope to a labview virtual interface (VI) in the host via the GPIB protocol, then the signal was discretized at 100 MHz of the oscilloscope. According to the Nyquist law, it allows obtain a reliable data to process. Finally at the end of the execution time of the benchmarks, the average power consumed in that interval was calculated as the equation 3.2 [42]:

$$P_{av} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} V(t) I(t)\, dt \tag{3.2}$$

For example, figure 3–5 shows the variations of the physical current waveform in the LM3S811 microcontroller. Figure 3–6 shows the instrumentation used to measure the current variation of the target architectures.

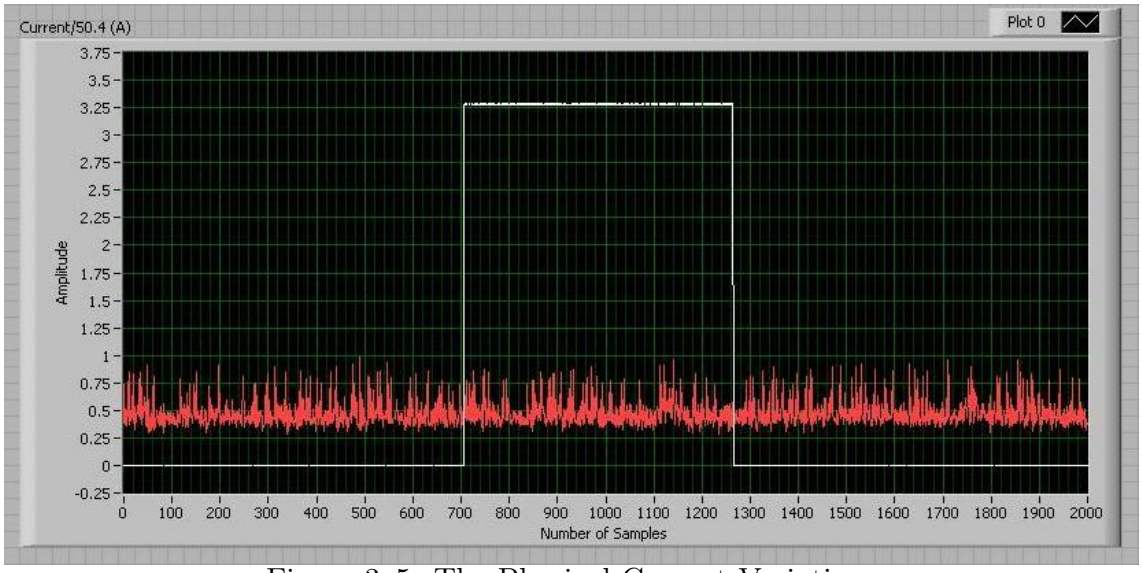


Figure 3–5: The Physical Current Variations

3.5.1 Validation of the Instrumentation

Before running the experiments we had in mind that equipment such as power supplies, the oscilloscope and the data acquisition card were properly calibrated for measurements of current variations in each of the experiments. As a general rule we always started with a multimeter measuring the output voltages of the power

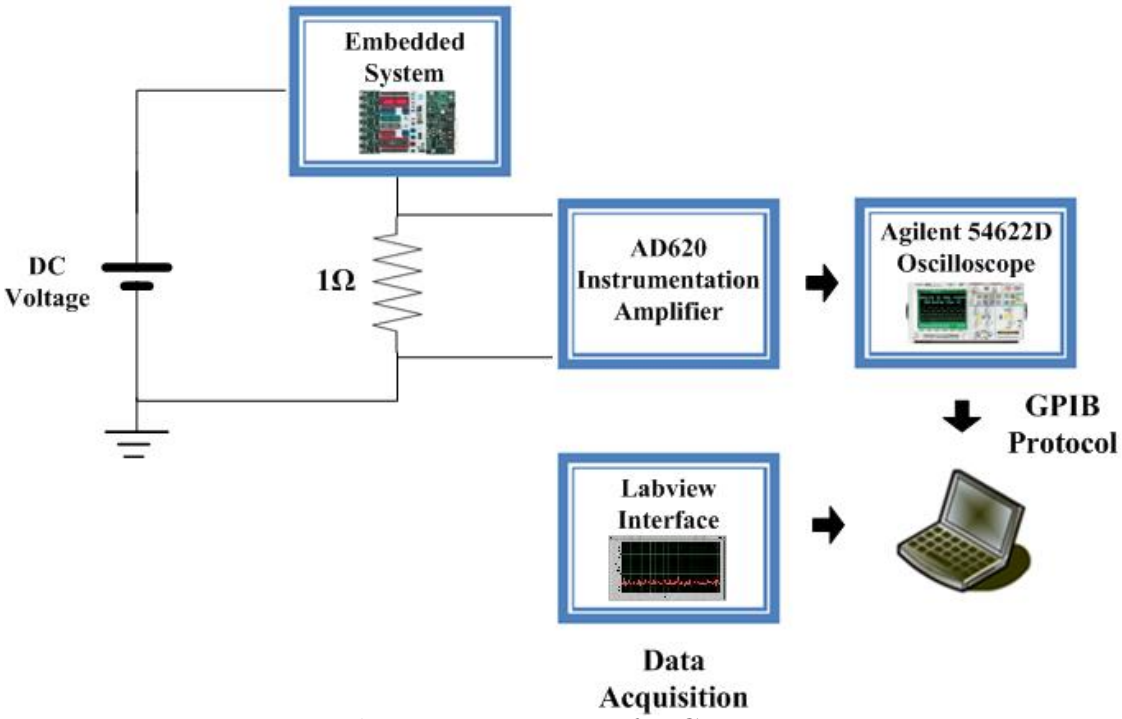


Figure 3–6: The Instrumentation for Current Measurement

supply and then chek the same level of voltage across the probes of the oscilloscope and he labview interface with data acquisition card.

3.6 Design of the Experiments and Statistical Analysis

An experiment is group of changes imposed to the input variables of a system or process to observe the changes caused in the output variable [13]. Design of experiments refers to the careful plan designed to obtain statistically significant data from the experiment with the minimum amount of effort. Taking this into account, our study intends to reach sound conclusions from statistical analysis when power consumption of embedded systems is studied under different algorithms at the operating system level software. In our case the two factors chosen were: algorithms oriented to DVS and algorithms oriented to memory management, after that the response variable selected was power consumption of the target platforms, after we chosen the experimental design. Then a full factorial design with two replicates was chosen, we mean that in each complete replication of the experiment all possible combination of the levels of the factors are investigated, one of the advantages of

this experiment is that it shows the impact of factors and a possible interaction between them, then we performed the experiment and the data was recollected, after of that a statistical analysis of the data using ANOVA was done and finally the conclusions and recommendations are presented.

The analysis of variance (ANOVA) is a statistical method used to analyze if the input factors are significant in the changes of the response of a system. ANOVA assumes that errors and observations are normal and independently distributed. This statistical analysis is based in *p-value* and the *null hipothesis*, the *p-value* is the minimum level of significance which the *null hipothesis* will be rejected. The significance level is selected according to the type of problem. The *null hipothesis* establishes that there is no significative impact of the factors or entry variable on the response. In our study a *p-value* of 0.05 was chosen, hence there is a 5% of probability to reject the *null hipothesis*. The *null hipothesis* we have is that the DFS and memory management algorithms do not have impact in the power reduction of medium and small scale embedded system. The analysis of variance will tell us if this assumption is statistically correct. In order to validate the results and find the behavior patterns of the algorithms, the test after ANOVA named Dunnet's test was used in each particular case. In this test each treatment is compared with the control (Static Memory Allocation - Constant Frequency) to see if the algorithm is statistically significant in terms of power savings. Dunnet's test provides a range of significance and compares it with each treatment.

The factors and levels of the experiments are:

1. Algorithms oriented to dynamic frequency scaling. The levels for this factor are: frequency constant, DFS based on CPU usage strategy, DFS predictive strategy based on WCET, and DFS with task priority.
2. Algorithms oriented to memory management. The levels for this factor are: static memory allocation and dynamic memory allocation.

3. Benchmarks or workloads. The levels for this factor are: basicmath, dijkstra, and bitcount.
4. Platforms. The levels for this factor are: LM3S811, MSP430F149, and ATMEGA323.

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.

A brief description of the experiments follows:

the first one consists of three platforms running simultaneously the MiBench, dijkstra, and bitcount benchmarks. It had the purpose of analyzing the behavior of the algorithms when the platforms are stressed simultaneously at I/O, processing, and memory level. In the second experiment, a code for a microgas chromatograph ran on all platforms to understand the effect that a real application load would have on the power consumption. Finally, the last experiment consists of running each algorithm individually, that is: MiBench, diskstra, and bitcount. The goal of this experiment is to find relationships and patterns between the type of benchmark, the architecture, and the algorithms on the power consumption.

4.1 Power Measurements

After performing the experiments, power consumption measurements for each platform were obtained. Tables A–1 to A–3 show the execution time and power consumption of four replicates taken when a subset of *MiBench* benchmarks are run, and the software techniques are applied on each platform. Tables A–4 to A–6 show the results with the *GC Code*. Tables B–1 to B–3 show the experiment with the MSP430F149 where the benchmarks were run individually. Tables B–4 to B–6 show the data with the ATMEGA323 where the benchmarks were run individually. Finally the results for LM3S811 are presented in the tables B–7 to B–9.

The results obtained from these experiments were used to perform a statistical analysis to find the relationship that exist between the type of architecture, the type of workload, and the OS algorithms in power reduction. A statistical analysis follows.

Figures 4–1 to 4–3 illustrate the data obtained showing a comparison between the averages the power consumed when *MiBench* and *uGC Code* are run on each platform versus the operating system level algorithms.

Figures 4–8 to 4–10 illustrate the averages the power consumed when the platforms run the benchmarks individually.

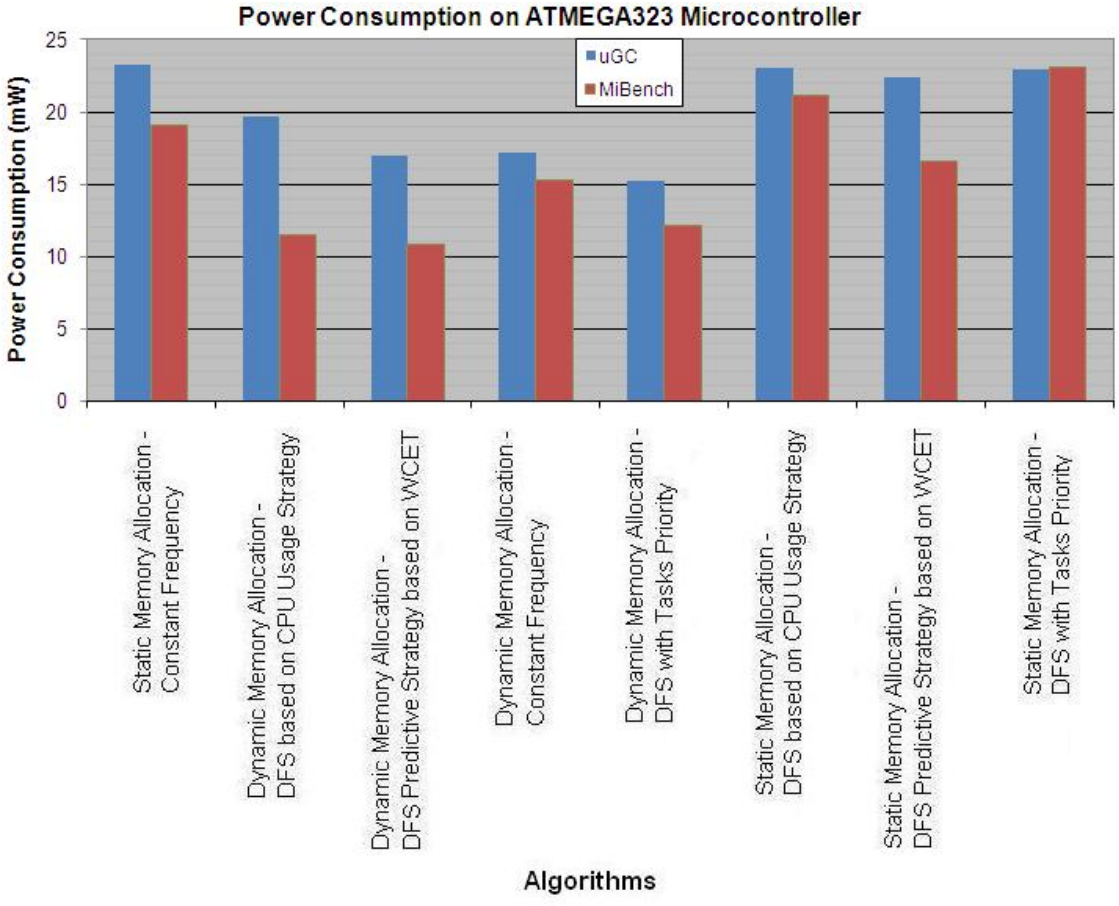


Figure 4–1: Power consumption averages on ATMEGA323 microcontroller

4.2 Full Factorial Design for each Platform

In this experiment, three factors were studied for each platform. The factors considered in the study were: algorithms oriented to dynamic frequency scaling,

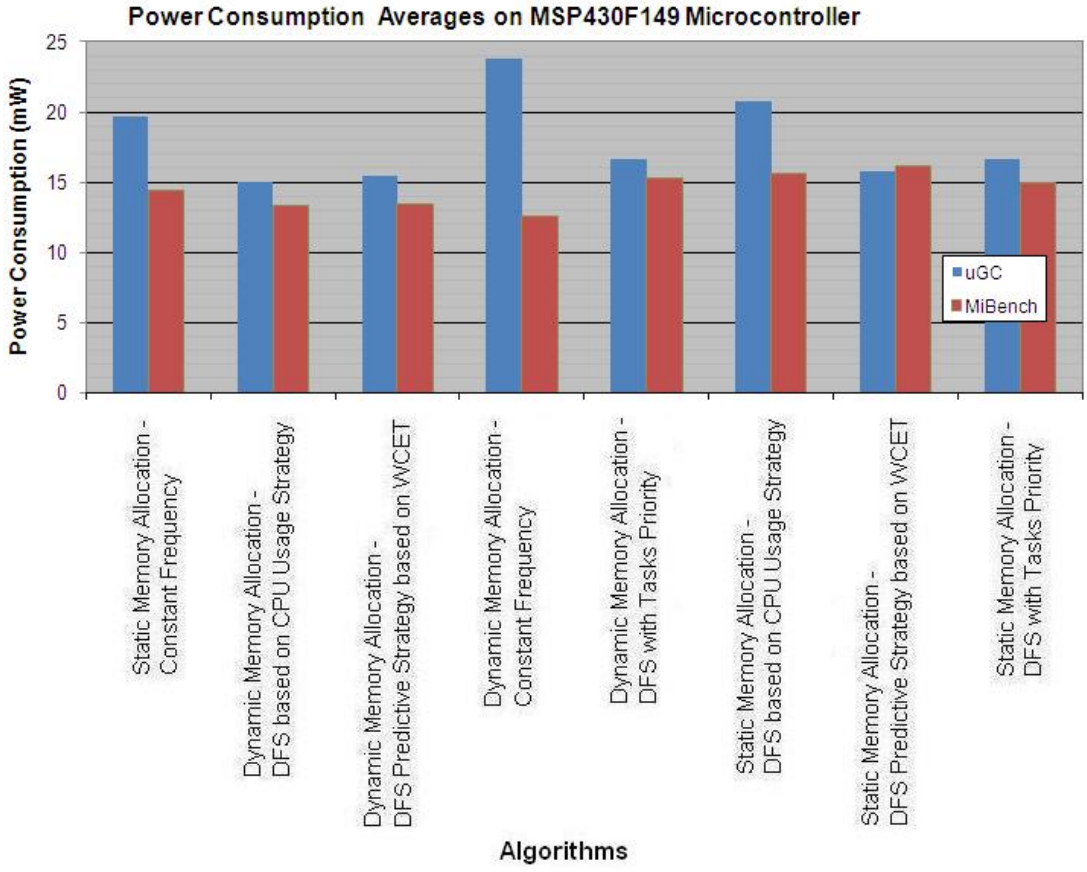


Figure 4–2: Power consumption averages on MSP430F149 microcontroller

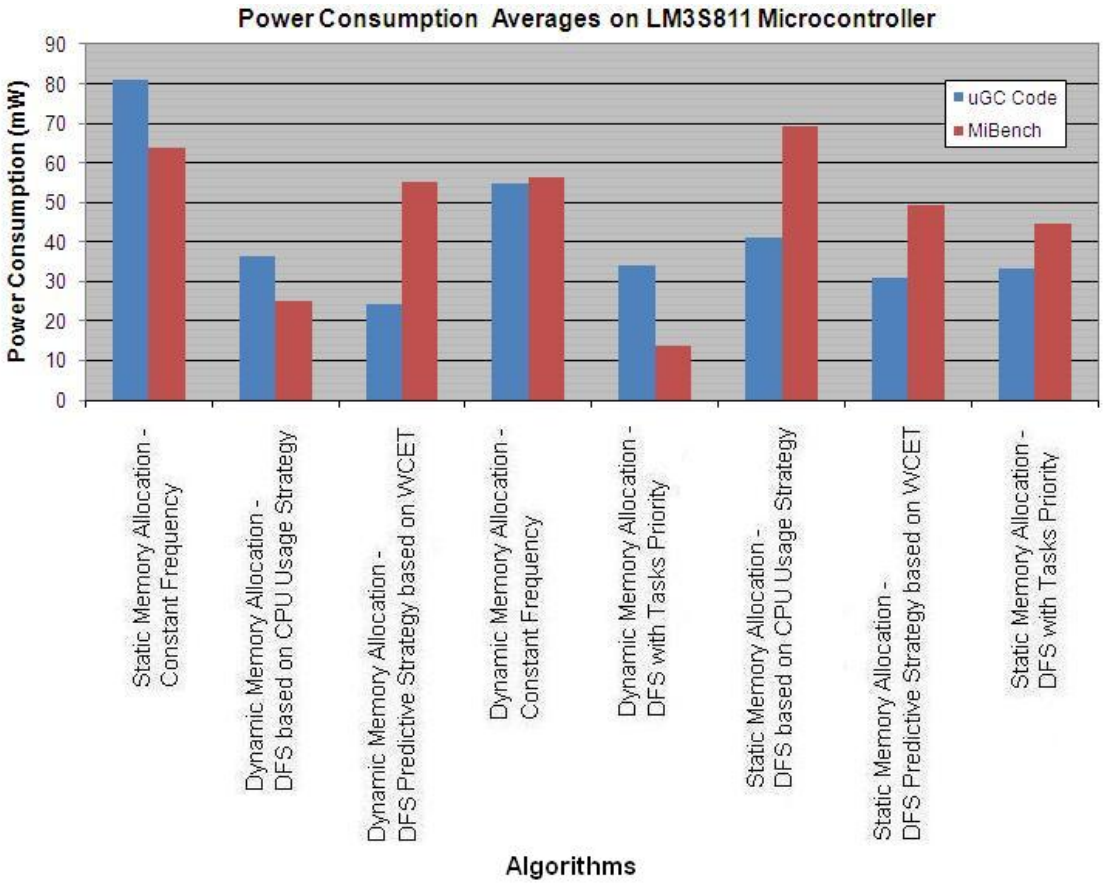


Figure 4–3: Power consumption averages on LM3S811 microcontroller

algorithms oriented to memory management, and the benchmarks selected. Due to the nature of the experiment, four replicates for experimental condition were taken, hence a full-factor factorial design with four replicates was the design chosen to make the analysis. One of the advantages of this experiment is that it shows the impact of factors and the possible interaction between them and allows conclude about the significance of the selected factors on each platform.

We performed the experiment and statistical analysis of the data using ANOVA was done. Figures 4–4 to 4–6 show the ANOVA analysis for each platform.

Later, we have chosen the Dunnet's analysis [13] to know the significance of each algorithm with respect to control (Static Memory Allocation - Constant Frequency (1)) in terms of power saving in each case. This allowed us to find patterns of behavior and to obtain more general conclusions. Finally the conclusions and recommendations are presented.

4.2.1 ANOVA for ATMEGA323 Microcontroller

The figure 4–4 shows the ANOVA for the experiment and the figure F–1 in the appendix F, shows the normal probability plot. Here all factors and their interaction have significance in the power consumption of the system. The *p-values* were 0.00. The factor workload or benchmarks is the same.

The figures D–1 and D–4 in the appendix D, show the Dunnet's tests for the ATMEGA323 running the MiBench and uGC, and the tables E–1 and E–4 in the appendix E, show the analysis in terms of the significance of the algorithms.

For example, the table 4–1 show the significant and not significant algorithms in power reducction, this is the result of dunnet's test for this experiment.

4.2.2 ANOVA for MSP430F149 Microcontroller

The figure 4–5 shows the ANOVA for the experiment and the figure F–2 (appendix F) shows the normal probability plot. The algorithm given by the interaction

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2
Workload	fixed	2	1, 2

Analysis of Variance for Power (mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	44.056	44.056	14.685	289.24	0.000
Memory Management Algorithms	1	703.473	703.473	703.473	13855.74	0.000
Workload	1	233.842	233.842	233.842	4605.79	0.000
DFS Algorithms * Memory Management Algorithms	3	46.816	46.816	15.605	307.36	0.000
DFS Algorithms *Workload	3	45.578	45.578	15.193	299.24	0.000
Memory Management Algorithms* Workload	1	13.864	13.864	13.864	273.07	0.000
DFS Algorithms* Memory Management Algorithms* Workload	3	40.730	40.730	13.577	267.41	0.000
Error	48	2.437	2.437	0.051		
Total	63	1130.795				

S = 0.225325 R-Sq = 99.78% R-Sq(adj) = 99.72%

Figure 4–4: ANOVA for ATMEGA323 microcontroller

Table 4–1: Significant algorithms in power reduction on ATMEGA323 running MiBench

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS with Tasks Priority (8)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - Constant Frequency (4)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	

between memory management and the workload was not significative, its *p-value* was $0.078 > 0.05$.

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2
Workload	fixed	2	1, 2

Analysis of Variance for Power (mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	49.952	49.952	16.651	10.22	0.000
Memory Management Algorithms	1	17.497	17.497	17.497	10.74	0.002
Workload	1	195.788	195.788	195.788	120.21	0.000
DFS Algorithms * Memory Management Algorithms	3	60.081	60.081	20.027	12.30	0.000
DFS Algorithms * Workload	3	132.862	132.862	44.287	27.19	0.000
Memory Management Algorithms * Workload	1	5.297	5.297	5.297	3.25	0.078
DFS Algorithms * Memory Management Algorithms * Workload	3	48.016	48.016	16.005	9.83	0.000
Error	48	78.175	78.175	1.629		
Total	63	587.668				

S = 1.27619 R-Sq = 86.70% R-Sq(adj) = 82.54%

Figure 4–5: ANOVA for MSP430F149 microcontroller

The figures D–2 and D–5 show the Dunnet's tests for the MSP430F149 running the MiBench and uGC, and the tables E–2 and E–5 show the analysis in terms of the significance of the algorithms.

4.2.3 ANOVA for LM3S811 Microcontroller

The figure 4–6 shows the ANOVA for the LM3S811 microcontroller and the figure F–3 shows the normal probability plot. Here all factors and their interaction have significance in the power consumption of the system. The *p-values* were <0.05.

The figures D–3 and D–6 show the Dunnet's tests for the MSP430F149 running the MiBench and uGC, and the tables E–3 and E–6 show the analysis in terms of the significance of the algorithms.

4.3 Results for the *Full Factorial Design* on the three Platforms

Here the platforms are considered as another factor of interest in the design. Then four factors were analyzed in the experiment: algorithms oriented to dynamic frequency scaling, algorithms oriented to memory management, the benchmarks

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2
Workload	fixed	2	1, 2

Analysis of Variance for Power (mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	8800.3	8800.3	2933.4	97.54	0.000
Memory Management Algorithms	1	3605.0	3605.0	3605.0	119.87	0.000
Workload	1	314.3	314.3	314.3	10.45	0.002
DFS Algorithms * Memory Management Algorithms	3	1018.1	1018.1	339.4	11.28	0.000
DFS Algorithms *Workload	3	1523.6	1523.6	507.9	16.89	0.000
Memory Management Algorithms * Workload	1	282.9	282.9	282.9	9.41	0.004
DFS Algorithms * Memory Management Algorithms * Workload	3	3807.8	3807.8	1269.3	42.20	0.000
Error	48	1443.6	1443.6	30.1		
Total	63	20795.7				

S = 5.48409 R-Sq = 93.06% R-Sq(adj) = 90.89%

Figure 4–6: ANOVA for LM3S811 microcontroller

selected, and the platforms. Due to the nature of the experiment, four replicates for experimental condition were taken, hence a full-factor factorial design with four replicates was the design chosen to make the analysis. One of the advantages of this experiment is that this shows the impact of factors and the possible interaction between them. We performed the experiment and statistical analysis of the data using ANOVA was done. Figure 4–7 shows the results of ANOVA for the full factorial design and the figure F–4 illustrates the normal probability plot.

We observed that statistically speaking, there is a significant effect of the platforms, benchmarks, operating system level algorithms oriented to DFS and memory management and their interaction in reduction of power consumption.

4.4 Results for the Experiment with the Benchmarks Running Individually

Appendix B presents the power consumption on selected platforms running the algorithms individually. The appendix C the ANOVA for each case when the platforms run the benchmarks individually. The figures D–7 to D–15 show the Dunnet's tests. Appendix E contents the results of the analysis for the Dunnet's

Analysis of Variance for Power (mW), using Adjusted SS for Tests

Source	P
DFS Algorithms	0.000
Memory Management Algorithms	0.000
Workload	0.042
Platform	0.000
DFS Algorithms* Memory Management Algorithms	0.000
DFS Algorithms*Workload	0.000
DFS Algorithms*Platform	0.000
Memory Management Algorithms* Workload	0.000
Memory Management Algorithms* Platform	0.000
Workload*Platform	0.000
DFS Algorithms* Memory Management Algorithms* Workload	0.000
DFS Algorithms* Memory Management Algorithms* Platform	0.000
DFS Algorithms*Workload*Platform	0.000
Memory Management Algorithms* Workload*Platform	0.003
DFS Algorithms* Memory Management Algorithms* Workload*Platform	0.000
Error	
Total	

S = 3.25345 R-Sq = 97.26% R-Sq(adj) = 96.36%

Figure 4–7: *Full Factorial Design* on the three platforms

tests and appendix F presents the normal probability plots for all experiments. The algorithms in appendix E are organized in order of highest to lowest power reduction for each algorithm.

The previous experiments with the benchmarks running simultaneously (see tables A–1 to A–3) showed that the combination of DFS algorithms and memory management algorithms in some cases of the MSP430F149, LM3S811, and ATMEGA323 were not significant in reducing power. For example in the table A–1, when the ATMEGA323 platform ran the MiBench benchmark, the algorithms Static Memory Allocation - DFS with Tasks Priority and Static Memory Allocation - DFS based on CPU Usage Strategy did not reduce power.

The OS algorithms in the past experiment with the MSP430F149 showed a poor or null impact in terms of power reduction. In order to understand the behavior of the system, we repeated experiments where each platform ran each benchmark

individually to stress specific parts of the architecture (I/O, computation, memory). This allowed us to find a relationship between the type of platforms, the type of workload, and the OS algorithms in the power reduction.

For this case we selected three benchmarks: basicmath (mathematical operations), dijkstra (I/O algorithm) and bitcount (count the number of bits of a string). Basicmath is categorized as computation intensive applications, therefore ALU operations dominate performance, and dijkstra algorithm belong to I/O intensive applications and the bitcount is memory stressing. Figures 4–8 to 4–10 show the averages of power consumption vs OS algorithms for each platform. For example the figure 4–8 shows the power consumption on MSP430F149, here all algorithms reduced power when the basicmath algorithm was run (see table E–7).

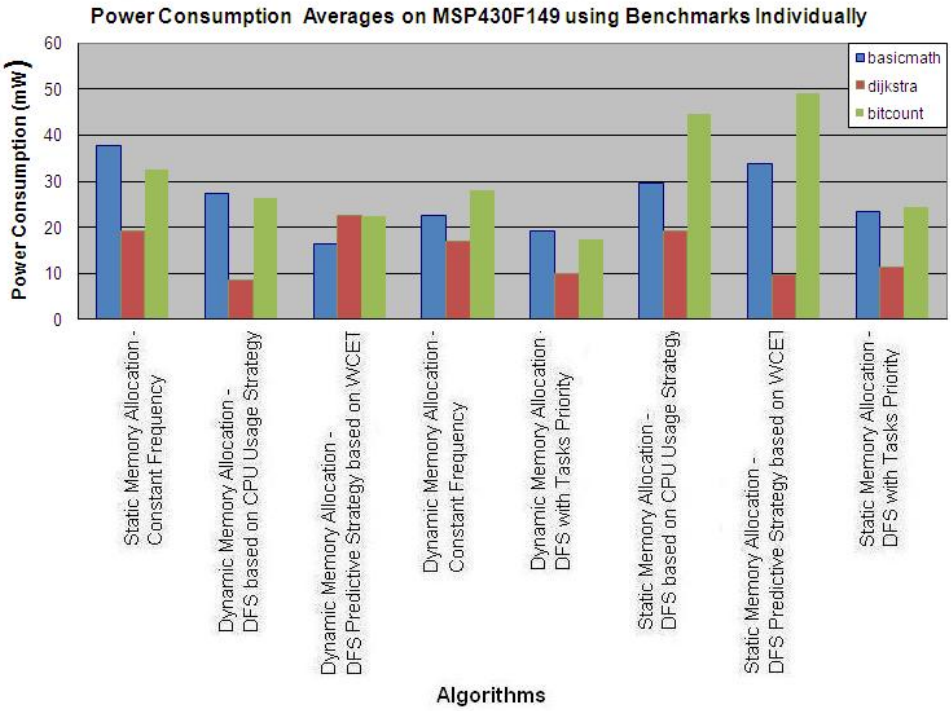


Figure 4–8: Power consumption averages on MSP430F149 microcontroller with the benchmarks running individually

ANOVA and Dunnet's (see appendix C, E) test demonstrated that OS algorithms impacted power consumption significantly (saving power) when the basicmath benchmark was running in MSP430F149 platform (see tables 4–8, E–7) and

dijkstra in LM3S811 and ATMEGA323 platforms (see tables 4–10, E–14, 4–9, and E–11). Basicmath is computation driven and the architecture MSP430F149 was designed for computation oriented applications. Dijkstra is I/O stressing and the LM3S811 and ATMEGA323 have an architectural organization I/O oriented.

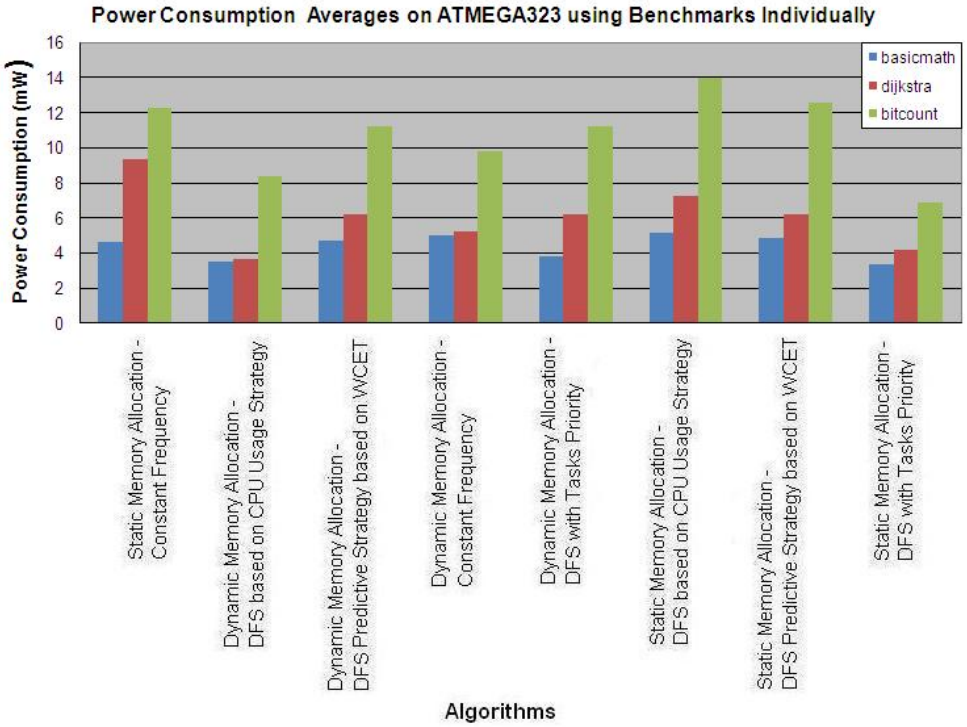


Figure 4–9: Power consumption averages on ATMEGA323 microcontroller with the benchmarks running individually

4.5 Analysis of the Results

In this work we evaluated the effect of variating simultaneously possible factors of interest: platforms, benchmarks, operating system level algorithms oriented to DFS and memory management, and their interaction in the power reduction of small and medium scale embedded systems running RTOS. Statistically sound conclusions were obtained with design of experiments techniques (DOE) [13].

From the data obtained in the first experiment, where the three platforms run simultaneously the benchmarks to stress the overall system at computation, I/O and memory level, results showed that operating system algorithms impacted the reduction of power consumption depending of the platform and the workload, and their

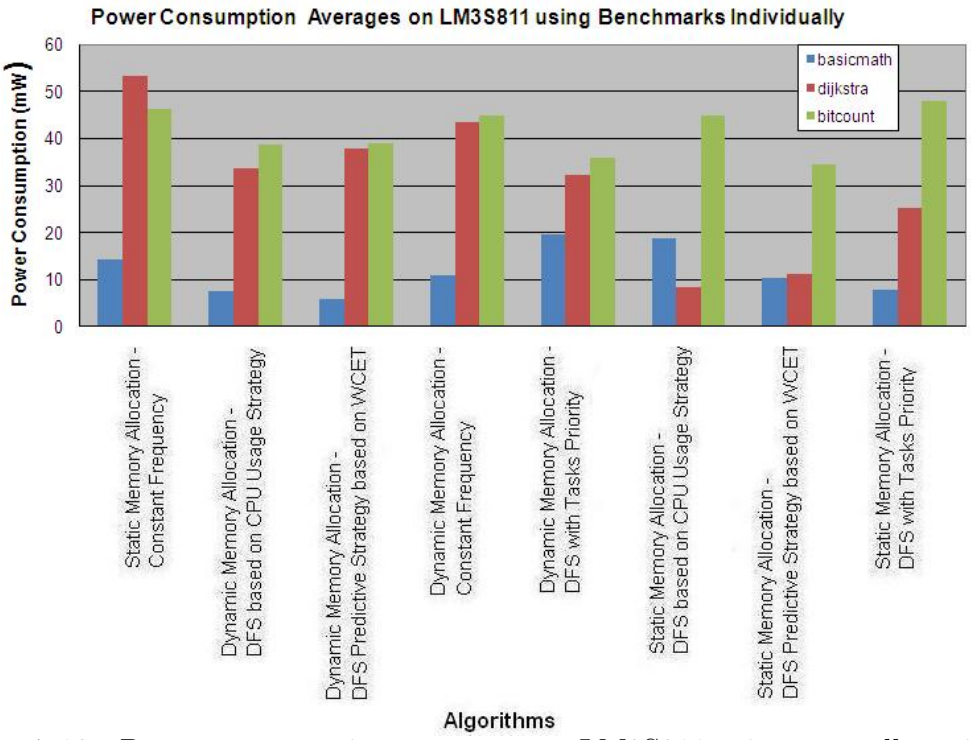


Figure 4–10: Power consumption averages on LM3S811 microcontroller with the benchmarks running individually

interaction (see figure 4–7). Dynamic memory allocation improved significantly the power reduction on ATMEGA323 platform (see figure 4–1), due that dynamic memory allocation tuned well with the architecture and the workload. The combination of DFS and memory management algorithms in some cases of the three platforms were not significant in reducing power. The results showed a interaction between all factors in power consumption of the system. In some cases the combination of OS algorithms worsened the power consumption, it was because the OS algorithms, and the application did not tune well with the characteristics of the platform. Accordingly it is not possible conclude that OS algorithms will impact the power reduction in any architecture running any workload, without having into consideration the existent interaction between all factors.

In order to find the relationship between the factors and generalize about the effect the algorithms in others architectures, it was necessary to make a more sound analysis of the relation between the types of workload and the type of architecture

and its effect on the OS power consumption. Therefore, in the next experiment, we stressed specific parts of the architecture.

The table 4–2 shows the profiling of the benchmarks and algorithms used. The Table 4–3 show the summary of results for the MSP430F149, ATMEGA323 and LM3S811 running individually the three types of workload. The above experiments (see figures 4–8 to 4–10) had shown the worst performance of the algorithms in MSP430F149 architecture and some algorithms did not have impact in power reduction in the others platforms. From results we concluded that all OS algorithms reduced power significatively when the application was of the same type that the target architecture. Consequently tuning between factors was necessary in order to saving power.

4.6 Observations

The table 4–4 shows each algorithm associated to a number, it with the purpose of simplify the references to a particular algorithm.

1. The algorithms oriented to frequency with dynamic memory allocation reduced power when the bitcount benchmarks was run.
2. Frequency reduction algorithm (2) worked good when it was associated with dynamic memory allocation.
3. All algorithm produced power reduction when the LM3S811 run dijkstra benchmark.
4. All algorithm produced power reduction when the ATMEGA323 run dijkstra benchmark.
5. All algorithm produced power reduction when the MSP430F149 run basicmath benchmark.
6. Dynamic memory allocation worked well with bitcount benchmark.
7. The MSP430F149 is the platform where the greatest number of times the algorithms worked (17 times).

Table 4–2: Profiling of the benchmarks and algorithms

Benchmarks and Algorithms	**Profiling**
Basicmath	720 operations of addition 720 operations of subtraction 720 operations of division Non I/O , Memory stressing to save data
Dijkstra	6 I/O operations 2 comparisons 2 operations of addition
Bitcount	96 memory accesses in LM3S811 192 memory accesses in MSP430F149 384 memory accesses in ATMEGA323
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	3 comparisons 1 operation of division 1 operation of mutiplication No I/O
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	7 operations of addition 1 operation of division No I/O
Dynamic Memory Allocation - DFS with Tasks Priority	3 comparisons No I/O
Static Memory Allocation - DFS based on CPU Usage Strategy	3 comparisons 3 operations of division 1 operation of multiplication No I/O
Static Memory Allocation - DFS Predictive Strategy based on WCET	7 operations of addition 1 operation of division No I/O
Static Memory Allocation - DFS with Tasks Priority	3 comparisons No I/O

Table 4–3: Summary of significant Algorithms on the selected platforms

Platforms	**Basicmath**	**Dijkstra**	**Bitcount**
MSP430F149	3,5,4,8,2,6,7	2,7,5,8,4	5,3,8,2,4
LM3S811	3,2,8,7,4	6,7,8,2,5,3,4	7,5,2,3
ATMEGA323	8,2,5	2,8,4,5,3,7,6	8,2,4,3,5

Table 4–4: Algorithms utilized in this research work

Algorithms
Static Memory Allocation - DFS based on CPU Usage Strategy(2)
Dynamic Memory Allocation - DFS with Tasks Priority (5)
Static Memory Allocation - DFS with Tasks Priority (8)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)
Dynamic Memory Allocation - Constant Frequency (4)
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)
Static Memory Allocation - DFS based on CPU Usage Strategy (6)

8. Static Memory Allocation - DFS based on CPU Usage Strategy presented the worst behavior in terms of power consumption.
9. Dynamic Memory Allocation - DFS based on CPU Usage Strategy was significant in power reduction in all cases.
10. The algorithms 2,8,5,4 reduced power in MSP430F149 whit the three benchmarks.
11. The algorithms 2,3,7 reduced power in LM3S811 whit the three benchmarks.
12. The algorithms 2,8,5 reduced power in ATMEGA323 whit the three benchmarks.

4.7 Recommendations

1. Dynamic Memory Allocation DFS based on CPU Usage Strategy reduced power in all architectures with the different benchmarks, in some cases presented the best power consumption savings. The condition necessary is that this algorithm must be associate with dynamic memory allocation.

2. Our results showed that it is possible associate the power reduction of the studied algorithms when the platforms run applications of same type of the application in a RTOS (computation, I/O, and memory).
3. The MSP430f149 is the platform where the greatest number of times the algorithms reduced power (17), its internal organization of the components connected to 3 buses allows better management of the data among the CPU, I/O, and memory. This produces reduction in the switching activity and power consumption
4. We have classified the algorithms in order of best performance to worst performance in terms of how many times they reduced power in all architectures (see table 4–5). Based in this classification the designer of embedded systems can use the algorithms associated with the greatest number of times.

Table 4–5: Number of times where the algorithms reduced power

Algorithms	**Number of times**
Static Memory Allocation - DFS based on CPU Usage Strategy(2)	9
Dynamic Memory Allocation - DFS with Tasks Priority (5)	8
Static Memory Allocation - DFS with Tasks Priority (8)	8
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	7
Dynamic Memory Allocation - Constant Frequency (4)	7
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	6
Static Memory Allocation - DFS based on CPU Usage Strategy (6)	3

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

This thesis has addressed the effect of different strategies used in software at the Real Time Operating System for reducing power consumption in middle to small size embedded systems. In particular, we have studied the effect of the combined use of these strategies on the consumption of the system and whether or not these strategies interact with each other.

We have found that Dynamic Frequency Scaling (DFS) based on CPU usage, DFS predictive strategy based on Worst Case Execution Time (WCET), and DFS based on tasks priority, Dynamic Memory Allocation, and Static Memory Allocation not necessarily behave as expected in all platforms. Moreover,these strategies interact among themselves, producing results which are not the sum or combination of previous results.

5.1 Contributions

- Our work was directed to medium to small embedded systems, which is different to most works found in literature.
- We have provided a methodology and a circuit to measure power in run time.
- We have evaluated the simultaneous effect of combining factors and algorithms in the analysis of power savings. Our work has found that there are interaction among all factors in the experiments.
- We have found a relationship among the type of architecture, the type of load, and the RTOS algorithms on power reduction.

Our future work includes the analysis of I/O strategies at the Real Time Operating System level in all the platforms and the analysis of the effect of all three interacting. In addition, we would like to study whether larger systems will have similar behavior of small and medium size systems. Another possible study is to study different types of operating systems and the study of a large number of architectures with the same bus size and architecture in order to identify if these affect power consumption.

APPENDICES

APPENDIX A
POWER CONSUMPTION ON SELECTED PLATFORMS RUNNING ALGORITHMS SIMULTANEOUSLY

Table A–1: Power consumption on ATMEGA323 platform using MiBench benchmarks as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.106	21.065
	0.106	21.258
	0.106	21.116
	0.107	21.197
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.638	16.177
	0.636	16.543
	0.640	17.067
	0.636	16.653
Static Memory Allocation - Constant Frequency	0.101	19.182
	0.101	19.248
	0.101	18.787
	0.101	18.964
Static Memory Allocation - DFS with Tasks Priority	0.160	23.157
	0.159	23.069
	0.160	23.318
	0.160	22.890
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.260	11.326
	0.260	11.698
	0.260	11.102
	0.260	11.800
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.300	10.902
	0.301	10.270
	0.300	10.847
	0.302	11.452
Dynamic Memory Allocation - Constant Frequency	0.338	15.398
	0.338	15.279
	0.337	14.970
	0.337	15.317
Dynamic Memory Allocation - DFS with Tasks Priority	0.366	12.184
	0.365	12.231
	0.365	12.050
	0.366	11.879

Table A–2: Power consumption on MSP430F149 platform using MiBench benchmarks as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.244	14.925
	0.243	16.042
	0.243	16.248
	0.244	14.985
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.160	16.025
	0.160	16.376
	0.160	16.050
	0.160	16.101
Static Memory Allocation - Constant Frequency	0.158	14.474
	0.158	14.524
	0.158	14.395
	0.158	14.356
Static Memory Allocation - DFS with Tasks Priority	0.180	15.424
	0.179	14.749
	0.179	15.139
	0.179	14.645
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.246	13.984
	0.246	12.736
	0.247	13.349
	0.247	13.170
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.167	13.861
	0.165	12.998
	0.167	13.615
	0.166	13.297
Dynamic Memory Allocation - Constant Frequency	0.160	12.709
	0.160	12.700
	0.160	12.712
	0.160	12.285
Dynamic Memory Allocation - DFS with Tasks Priority	0.210	15.545
	0.209	15.049
	0.209	15.343
	0.209	15.167

Table A–3: Power consumption on LM3S811 platform using MiBench benchmarks as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.160	52.530
	0.160	74.072
	0.160	81.494
	0.161	69.692
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.190	38.598
	0.190	47.784
	0.190	48.165
	0.193	43.910
Static Memory Allocation - Constant Frequency	0.160	52.798
	0.151	60.566
	0.160	68.383
	0.151	73.715
Static Memory Allocation - DFS with Tasks Priority	0.168	47.035
	0.167	55.107
	0.168	44.934
	0.168	49.812
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.168	25.442
	0.170	28.376
	0.169	24.724
	0.168	22.076
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.170	55.273
	0.168	66.839
	0.168	45.192
	0.169	53.345
Dynamic Memory Allocation - Constant Frequency	0.160	54.059
	0.159	51.430
	0.160	61.016
	0.160	59.234
Dynamic Memory Allocation - DFS with Tasks Priority	0.170	14.710
	0.180	13.259
	0.165	12.915
	0.175	13.259

Table A–4: Power consumption on ATMEGA323 platform using uGC code as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.150	15.935
	0.150	15.289
	0.150	14.857
	0.152	14.864
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.920	14.800
	0.920	14.743
	0.920	14.819
	0.920	14.792
Static Memory Allocation - Constant Frequency	0.130	15.349
	0.135	15.358
	0.128	15.360
	0.135	15.330
Static Memory Allocation - DFS with Tasks Priority	0.180	15.119
	0.175	15.222
	0.185	15.281
	0.178	15.065
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.380	12.692
	0.370	15.544
	0.360	10.911
	0.360	12.779
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.260	11.124
	0.260	11.168
	0.260	11.096
	0.260	11.243
Dynamic Memory Allocation - Constant Frequency	0.220	11.327
	0.220	11.385
	0.220	11.342
	0.220	11.344
Dynamic Memory Allocation - DFS with Tasks Priority	0.380	9.994
	0.378	9.992
	0.376	10.022
	0.377	10.029

Table A–5: Power consumption on MSP430F149 platform using uGC code as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.110	19.911
	0.110	19.876
	0.110	23.447
	0.110	19.757
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.104	17.035
	0.104	15.375
	0.104	15.189
	0.104	15.359
Static Memory Allocation - Constant Frequency	0.105	16.394
	0.104	20.016
	0.105	19.881
	0.105	22.327
Static Memory Allocation - DFS with Tasks Priority	0.160	14.881
	0.160	15.753
	0.160	16.438
	0.160	19.575
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.108	15.034
	0.108	15.367
	0.108	14.260
	0.108	15.480
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.105	17.312
	0.105	16.904
	0.105	14.296
	0.105	13.215
Dynamic Memory Allocation - Constant Frequency	0.104	23.486
	0.104	24.724
	0.104	22.284
	0.104	24.609
Dynamic Memory Allocation - DFS with Tasks Priority	0.158	14.598
	0.158	14.852
	0.158	18.988
	0.158	18.280

Table A–6: Power consumption on LM3S811 platform using uGC code as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.110	35.638
	0.113	44.239
	0.113	42.138
	0.113	43.015
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.150	30.095
	0.150	41.664
	0.150	40.859
	0.150	44.970
Static Memory Allocation - Constant Frequency	0.111	76.064
	0.111	78.871
	0.111	84.980
	0.111	83.642
Static Memory Allocation - DFS with Tasks Priority	0.145	34.522
	0.145	28.756
	0.145	28.651
	0.145	32.290
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.114	35.842
	0.114	34.807
	0.113	38.573
	0.113	36.949
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.220	23.923
	0.220	24.741
	0.220	25.451
	0.220	22.194
Dynamic Memory Allocation - Constant Frequency	0.115	52.827
	0.115	56.653
	0.115	54.572
	0.115	54.521
Dynamic Memory Allocation - DFS with Tasks Priority	0.165	30.691
	0.173	31.666
	0.170	36.566
	0.175	37.533

APPENDIX B
POWER CONSUMPTION ON SELECTED PLATFORMS RUNNING ALGORITHMS INDIVIDUALLY

Table B–1: Power consumption on MSP430F149 running basicmath as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.170	29.838
	0.170	29.451
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.150	34.295
	0.150	33.231
Static Memory Allocation - Constant Frequency	0.155	39.544
	0.155	35.995
Static Memory Allocation - DFS with Tasks Priority	0.200	24.535
	0.200	22.495
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.170	27.768
	0.170	27.125
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.155	16.676
	0.155	16.291
Dynamic Memory Allocation - Constant Frequency	0.110	22.981
	0.110	22.281
Dynamic Memory Allocation - DFS with Tasks Priority	0.180	19.070
	0.180	19.137

Table B–2: Power consumption on MSP430F149 running dijkstra as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.035	19.115
	0.035	19.020
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.048	10.898
	0.048	8.279
Static Memory Allocation - Constant Frequency	0.028	23.347
	0.028	22.307
Static Memory Allocation - DFS with Tasks Priority	0.04	11.341
	0.04	11.354
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.042	8.574
	0.042	8.651
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.18	22.574
	0.18	22.651
Dynamic Memory Allocation - Constant Frequency	0.035	16.816
	0.035	16.944
Dynamic Memory Allocation - DFS with Tasks Priority	0.054	9.999
	0.054	9.985

Table B–3: Power consumption on MSP430F149 running bitcount as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.018	45.100
	0.017	44.200
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.015	50.000
	0.014	48.000
Static Memory Allocation - Constant Frequency	0.012	33.000
	0.012	32.000
Static Memory Allocation - DFS with Tasks Priority	0.02	24.000
	0.019	25.000
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.02	26.300
	0.021	26.400
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.015	22.000
	0.015	23.000
Dynamic Memory Allocation - Constant Frequency	0.018	28.000
	0.018	28.000
Dynamic Memory Allocation - DFS with Tasks Priority	0.025	18.230
	0.026	17.120

Table B–4: Power consumption on ATMEGA323 running basicmath as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.25	5.158
	0.25	5.188
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.19	4.861
	0.19	4.871
Static Memory Allocation - Constant Frequency	0.11	4.650
	0.11	4.700
Static Memory Allocation - DFS with Tasks Priority	0.27	3.353
	0.27	3.373
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.22	3.502
	0.22	3.652
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.19	4.722
	0.18	4.820
Dynamic Memory Allocation - Constant Frequency	0.28	5.019
	0.28	5.029
Dynamic Memory Allocation - DFS with Tasks Priority	0.30	3.849
	0.30	3.839

Table B–5: Power consumption on ATMEGA323 running dijkstra as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.035	7.280
	0.035	7.220
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.030	6.240
	0.030	6.550
Static Memory Allocation - Constant Frequency	0.042	11.320
	0.042	11.410
Static Memory Allocation - DFS with Tasks Priority	0.060	4.160
	0.060	4.190
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.040	4.160
	0.040	3.998
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.078	6.240
	0.080	6.365
Dynamic Memory Allocation - Constant Frequency	0.028	5.200
	0.028	5.298
Dynamic Memory Allocation - DFS with Tasks Priority	0.070	6.240
	0.07	6.119

Table B–6: Power consumption on ATMEGA323 running bitcount as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.055	14.000
	0.055	14.456
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.068	12.600
	0.068	12.460
Static Memory Allocation - Constant Frequency	0.060	12.600
	0.060	11.900
Static Memory Allocation - DFS with Tasks Priority	0.050	7.000
	0.050	6.720
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.065	7.790
	0.065	7.800
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.058	11.200
	0.058	10.987
Dynamic Memory Allocation - Constant Frequency	0.053	8.400
	0.053	8.394
Dynamic Memory Allocation - DFS with Tasks Priority	0.068	11.200
	0.068	11.109

Table B–7: Power consumption on LM3S811 running basicmath as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.100	18.200
	0.098	18.500
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.073	10.400
	0.075	10.400
Static Memory Allocation - Constant Frequency	0.078	14.300
	0.077	14.300
Static Memory Allocation - DFS with Tasks Priority	0.103	7.170
	0.103	8.297
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.065	7.500
	0.062	7.500
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.080	6.500
	0.082	5.200
Dynamic Memory Allocation - Constant Frequency	0.114	11.000
	0.115	11.000
Dynamic Memory Allocation - DFS with Tasks Priority	0.122	19.500
	0.122	19.500

Table B–8: Power consumption on LM3S811 running dijkstra as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.029	8.432
	0.029	8.566
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.031	11.003
	0.031	11.200
Static Memory Allocation - Constant Frequency	0.018	53.600
	0.017	53.800
Static Memory Allocation - DFS with Tasks Priority	0.025	25.200
	0.025	25.200
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.038	33.100
	0.037	33.226
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.034	36.400
	0.035	36.200
Dynamic Memory Allocation - Constant Frequency	0.030	42.000
	0.028	42.400
Dynamic Memory Allocation - DFS with Tasks Priority	0.041	33.800
	0.040	33.600

Table B–9: Power consumption on LM3S811 running bitcount as workload

Algorithms	**Execution Time(s)**	**P(mW)**
Static Memory Allocation - DFS based on CPU Usage Strategy	0.019	45.078
	0.018	45.000
Static Memory Allocation - DFS Predictive Strategy based on WCET	0.016	33.000
	0.015	34.234
Static Memory Allocation - Constant Frequency	0.011	47.871
	0.011	45.098
Static Memory Allocation - DFS with Tasks Priority	0.018	48.233
	0.018	48.000
Dynamic Memory Allocation - DFS based on CPU Usage Strategy	0.014	38.83
	0.014	38.83
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET	0.015	39.542
	0.015	39.712
Dynamic Memory Allocation - Constant Frequency	0.012	45.619
	0.012	45.300
Dynamic Memory Allocation - DFS with Tasks Priority	0.020	36.002
	0.020	36.100

APPENDIX C
ANOVA FOR THE PLATFORMS RUNNING EACH BENCHMARK INDIVIDUALLY

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2

Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	897.17	897.17	299.06	1496.72	0.000
Memory Management Algorithms	1	119.51	119.51	119.51	598.11	0.000
DFS Algorithms* Memory Management Algorithms	3	695.26	695.26	231.75	1159.88	0.000
Error	8	1.60	1.60	0.20		
Total	15	1713.53				

S = 0.446998 R-Sq = 99.91% R-Sq(adj) = 99.83%

Figure C–1: ANOVA for MSP430F149 platform running basicmath

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2

Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	229.221	229.221	76.407	22386.05	0.000
Memory Management Algorithms	1	25.962	25.962	25.962	7606.42	0.000
DFS Algorithms* Memory Management Algorithms	3	4.048	4.048	1.349	395.31	0.000
Error	8	0.027	0.027	0.003		
Total	15	259.258				

S = 0.0584222 R-Sq = 99.99% R-Sq(adj) = 99.98%

Figure C–2: ANOVA for MSP430F149 platform running dijkstra

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power (mW), using Adjusted SS for Tests

Source                           DF   Seq SS  Adj SS  Adj MS       F      P
DFS Algorithms                    3   841.25  841.25  280.42  249.26  0.000
Memory Management Algorithms      1   529.00  529.00  529.00  470.22  0.000
DFS Algorithms*                   3   306.50  306.50  102.17   90.81  0.000
  Memory Management Algorithms
Error                             8     9.00    9.00    1.12
Total                            15  1685.75


S = 1.06066   R-Sq = 99.47%   R-Sq(adj) = 99.00%
```

Figure C–3: ANOVA for MSP430F149 platform running bitcount

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source                           DF  Seq SS  Adj SS  Adj MS         F      P
DFS Algorithms                    3  4.2829  4.2829  1.4276  13652.29  0.000
Memory Management Algorithms      1  0.4451  0.4451  0.4451   4256.53  0.000
DFS Algorithms*                   3  2.3823  2.3823  0.7941   7593.78  0.000
  Memory Management Algorithms
Error                             8  0.0008  0.0008  0.0001
Total                            15  7.1111


S = 0.0102260   R-Sq = 99.99%   R-Sq(adj) = 99.98%
```

Figure C–4: ANOVA for ATMEGA323 platform running basicmath

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power(MW), using Adjusted SS for Tests

Source                           DF  Seq SS  Adj SS  Adj MS        F      P
DFS Algorithms                    3  22.691  22.691   7.564   692.27  0.000
Memory Management Algorithms      1  13.598  13.598  13.598  1244.54  0.000
DFS Algorithms*                   3  37.890  37.890  12.630  1155.96  0.000
  Memory Management Algorithms
Error                             8   0.087   0.087   0.011
Total                            15  74.266


S = 0.104527   R-Sq = 99.88%   R-Sq(adj) = 99.78%
```

Figure C–5: ANOVA for ATMEGA323 platform running dijkstra

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source                           DF  Seq SS  Adj SS  Adj MS       F      P
DFS Algorithms                    3  23.752  23.752   7.917  149.10  0.000
Memory Management Algorithms      1   7.352   7.352   7.352  138.46  0.000
DFS Algorithms*                   3  47.607  47.607  15.869  298.84  0.000
  Memory Management Algorithms
Error                             8   0.425   0.425   0.053
Total                            15  79.136


S = 0.230437   R-Sq = 99.46%   R-Sq(adj) = 98.99%
```

Figure C–6: ANOVA for ATMEGA323 platform running bitcount

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source                           DF   Seq SS   Adj SS   Adj MS        F      P
DFS Algorithms                    3  386.620  386.620  128.873  8930.17  0.000
Memory Management Algorithms      1   42.478   42.478   42.478  2943.46  0.000
DFS Algorithms*                   3  166.614  166.614   55.538  3848.45  0.000
  Memory Management Algorithms
Error                             8    0.115    0.115    0.014
Total                            15  595.828


S = 0.120130   R-Sq = 99.98%   R-Sq(adj) = 99.96%
```

Figure C–7: ANOVA for LM3S811 platform running basicmath

```
Factor                          Type   Levels  Values
DFS Algorithms                  fixed       4  1, 2, 3, 4
Memory Management Algorithms    fixed       2  1, 2


Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source                           DF   Seq SS   Adj SS  Adj MS        F      P
DFS Algorithms                    3  1781.29  1781.29  593.76  26940.23  0.000
Memory Management Algorithms      1   549.02   549.02  549.02  24910.25  0.000
DFS Algorithms*                   3   898.75   898.75  299.58  13592.73  0.000
  Memory Management Algorithms
Error                             8     0.18     0.18    0.02
Total                            15  3229.25


S = 0.148459   R-Sq =  99.99%   R-Sq(adj) = 99.99%
```

Figure C–8: ANOVA for LM3S811 platform running dijkstra

Factor	Type	Levels	Values
DFS Algorithms	fixed	4	1, 2, 3, 4
Memory Management Algorithms	fixed	2	1, 2

Analysis of Variance for Power(mW), using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
DFS Algorithms	3	232.389	232.389	77.463	417.51	0.000
Memory Management Algorithms	1	3.101	3.101	3.101	16.71	0.003
DFS Algorithms* Memory Management Algorithms	3	185.160	185.160	61.720	332.66	0.000
Error	8	1.484	1.484	0.186		
Total	15	422.135				

S = 0.430740 R-Sq = 99.65% R-Sq(adj) = 99.34%

Figure C–9: ANOVA for LM3S811 platform running bitcount

APPENDIX D
DUNNET'S TEST FOR THE PLATFORMS RUNNING THE BENCHMARKS

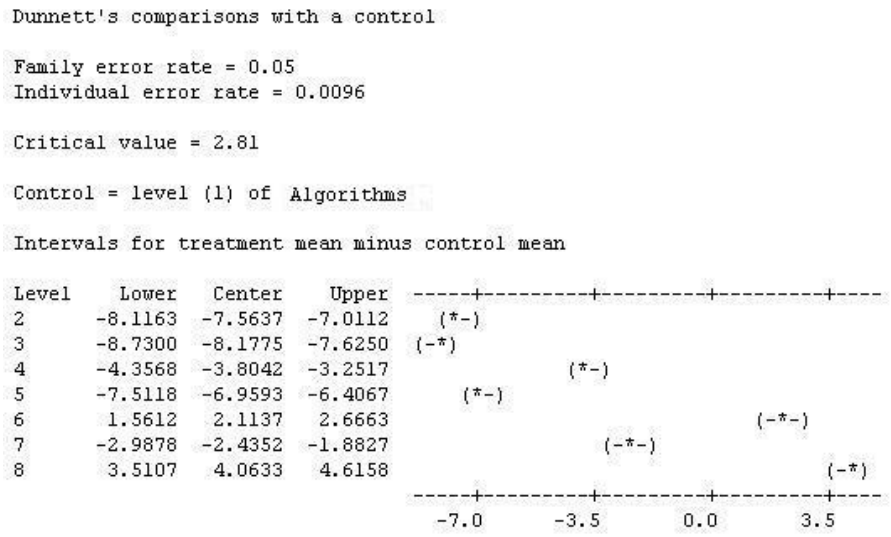


Figure D–1: Dunnet's analysis for the algorithms on ATMEGA323 using MiBench

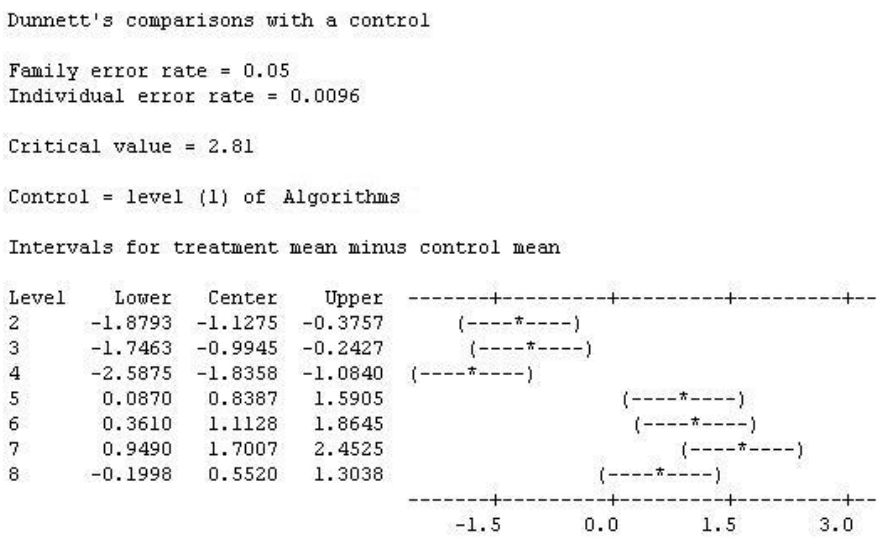


Figure D–2: Dunnet's analysis for the algorithms on MSP430F149 using MiBench

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0096

Critical value = 2.81

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ------+---------+---------+---------+---
2      -52.436  -38.711  -24.985       (-----*----)
3      -22.429   -8.703    5.022                  (-----*----)
4      -21.156   -7.431    6.295                   (----*-----)
5      -64.055  -50.329  -36.604  (-----*----)
6       -8.144    5.582   19.307                          (----*-----)
7      -32.976  -19.251   -5.525               (----*-----)
8      -28.368  -14.643   -0.917                 (----*-----)
                                 ------+---------+---------+---------+---
                                     -50       -25         0        25
```

Figure D–3: Dunnet's analysis for the algorithms on LM3S811 using MiBench

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0096

Critical value = 2.81

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower  Center   Upper  ---------+---------+---------+---------+
2       -5.702  -3.587  -1.473              (-----*-----)
3       -8.465  -6.351  -4.236       (-----*-----)
4       -8.174  -6.060  -3.946        (-----*-----)
5      -10.205  -8.091  -5.977  (-----*-----)
6       -2.286  -0.171   1.943                       (------*-----)
7       -2.964  -0.850   1.265                      (-----*-----)
8       -2.383  -0.269   1.846                       (-----*-----)
                                 ---------+---------+---------+---------+
                                       -7.0      -3.5       0.0       3.5
```

Figure D–4: Dunnet's analysis for the algorithms on ATMEGA323 using uGC code

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0096

Critical value = 2.81

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower  Center   Upper     +---------+---------+---------+---------
2       -8.132  -4.619  -1.107     (-------*--------)
3       -7.735  -4.223  -0.711      (-------*--------)
4        0.609   4.121   7.633                        (-------*--------)
5       -6.487  -2.975   0.537        (--------*-------)
6       -2.419   1.093   4.605                  (--------*--------)
7       -7.427  -3.915  -0.403      (--------*--------)
8       -6.505  -2.993   0.519         (--------*-------)
                                   +---------+---------+---------+---------
                                 -8.0      -4.0       0.0       4.0
```

Figure D–5: Dunnet's analysis for the algorithms on MSP430F149 using uGC code

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0096

Critical value = 2.81

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ---+---------+---------+---------+------
2      -51.405  -44.346  -37.288            (-----*-----)
3      -63.870  -56.812  -49.753  (-----*-----)
4      -33.305  -26.246  -19.188                          (-----*-----)
5      -53.834  -46.776  -39.717          (-----*-----)
6      -46.690  -39.632  -32.573                (-----*-----)
7      -48.551  -41.492  -34.434              (----*-----)
8      -56.893  -49.835  -42.776        (----*-----)
                                    ---+---------+---------+---------+------
                                     -60       -48       -36       -24
```

Figure D–6: Dunnet's analysis for the algorithms on LM3S811 using uGC code

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper     -+---------+---------+---------+--------
2      -13.778  -11.323   -8.868                      (---*---)
3      -24.741  -22.286  -19.831     (---*---)
4      -18.594  -16.139  -13.684               (---*---)
5      -22.121  -19.666  -17.211         (---*---)
6      -11.580   -9.125   -6.670                        (---*---)
7       -7.462   -5.007   -2.552                              (---*---)
8      -17.710  -15.255  -12.800                (----*---)
                                    -+---------+---------+---------+--------
                                   -24.0     -18.0     -12.0      -6.0
```

Figure D–7: Dunnet's analysis for the algorithms on MSP430F149 using basicmath

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ------+---------+---------+---------+---
2      -13.038  -10.714  -8.391  (----*---)
3        1.176    3.500   5.824                              (----*----)
4       -4.771   -2.447  -0.123                    (----*----)
5      -11.659   -9.335  -7.011     (---*----)
6       -2.583   -0.259   2.064                         (---*----)
7      -12.062   -9.738  -7.415    (----*---)
8      -10.303   -7.979  -5.656       (----*----)
                                   ------+---------+---------+---------+---
                                     -10.0       -5.0        0.0        5.0
```

Figure D–8: Dunnet's analysis for the algorithms on MSP430F149 using dijkstra

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  -------+---------+---------+---------+--
2       -8.620   -6.150   -3.680          (--*-)
3      -12.470  -10.000   -7.530       (-*-)
4       -6.970   -4.500   -2.030            (--*-)
5      -17.470  -15.000  -12.530  (-*-)
6        9.530   12.000   14.470                              (-*-)
7       14.030   16.500   18.970                                  (--*-)
8      -10.470   -8.000   -5.530         (-*-)
                                  -------+---------+---------+---------+--
                                       -10         0         10         20
```

Figure D–9: Dunnet's analysis for MSP430F149 platform running bitcount

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ----+---------+---------+---------+-----
2      -1.2544  -1.0980  -0.9416     (--*-)
3      -0.0603   0.0961   0.2525                          (--*-)
4       0.1934   0.3498   0.5062                              (--*-)
5      -0.9872  -0.8308  -0.6743           (-*--)
6       0.3422   0.4986   0.6550                                 (-*--)
7       0.0346   0.1911   0.3475                            (-*--)
8      -1.4683  -1.3119  -1.1555  (-*--)
                                  ----+---------+---------+---------+-----
                                   -1.20     -0.60      0.00      0.60
```

Figure D–10: Dunnet's analysis for ATMEGA323 platform running basicmath

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ------+---------+---------+---------+---
2      -7.6301  -7.2860  -6.9419  (--*---)
3      -5.4066  -5.0625  -4.7184                    (--*---)
4      -6.4601  -6.1160  -5.7719          (---*--)
5      -5.5296  -5.1855  -4.8414                  (--*---)
6      -4.4591  -4.1150  -3.7709                           (---*--)
7      -5.3141  -4.9700  -4.6259                    (--*---)
8      -7.5341  -7.1900  -6.8459   (--*---)
                                  ------+---------+---------+---------+---
                                    -7.0      -6.0      -5.0      -4.0
```

Figure D–11: Dunnet's analysis for ATMEGA323 platform running dijkstra

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  -----+---------+---------+---------+----
2      -5.2136  -4.4550  -3.6964      (--*--)
3      -1.9151  -1.1565  -0.3979                (--*--)
4      -4.6116  -3.8530  -3.0944        (--*--)
5      -1.8541  -1.0955  -0.3369                 (--*--)
6       1.2194   1.9780   2.7366                          (--*--)
7      -0.4786   0.2800   1.0386                     (--*--)
8      -6.1486  -5.3900  -4.6314  (--*--)
                                  -----+---------+---------+---------+----
                                    -5.0      -2.5       0.0       2.5
```

Figure D–12: Dunnet's analysis for ATMEGA323 platform running bitcount

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper   --+---------+---------+---------+-------
2      -7.2454  -6.8500  -6.4546       (*)
3      -8.8454  -8.4500  -8.0546   (*)
4      -1.5954  -1.2000  -0.8046                    (*)
5       4.8046   5.2000   5.5954                                  (*)
6       3.6546   4.0500   4.4454                             (*)
7      -4.2954  -3.9000  -3.5046             (*)
8      -6.8804  -6.4850  -6.0896        (*)
                                     --+---------+---------+---------+-------
                                    -8.0       -4.0        0.0        4.0
```

Figure D–13: Dunnet's analysis for LM3S811 platform running basicmath

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ------+---------+---------+---------+---
2      -21.026  -20.537  -20.048                           *)
3      -17.889  -17.400  -16.911                              (*
4      -11.989  -11.500  -11.011                                    (*
5      -20.489  -20.000  -19.511                            *
6      -45.690  -45.201  -44.712  (*
7      -43.087  -42.599  -42.110     *)
8      -28.989  -28.500  -28.011                    *)
                                  ------+---------+---------+---------+---
                                      -40       -30       -20       -10
```

Figure D–14: Dunnet's analysis for LM3S811 platform running dijkstra

```
Dunnett's comparisons with a control

Family error rate = 0.05
Individual error rate = 0.0110

Critical value = 3.29

Control = level (1) of Algorithms

Intervals for treatment mean minus control mean

Level    Lower   Center    Upper  ------+---------+---------+---------+---
2       -8.797   -7.538   -6.279          (--*-)
3       -7.120   -5.861   -4.602              (-*--)
4       -1.288   -0.028    1.231                          (--*-)
5      -10.696   -9.437   -8.178       (-*--)
6       -1.708   -0.449    0.810                          (-*--)
7      -13.130  -11.871  -10.612  (-*--)
8        1.369    2.629    3.888                                  (-*--)
                                  ------+---------+---------+---------+---
                                    -10.0       -5.0        0.0        5.0
```

Figure D–15: Dunnet's analysis for LM3S811 platform running bitcount

APPENDIX E
DUNNET'S ANALYSIS FOR THE PLATFORMS RUNNING THE BENCHMARKS

Table E–1: Significant algorithms in power reduction on ATMEGA323 running MiBench

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS with Tasks Priority (8)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - Constant Frequency (4)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	

Table E–2: Significant algorithms in power reduction on MSP430F149 running MiBench

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Dynamic Memory Allocation - Constant Frequency (4)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	Dynamic Memory Allocation - DFS with Tasks Priority (5)
	Static Memory Allocation - DFS with Tasks Priority (8)

Table E–3: Significant algorithms in power reduction on LM3S811 running MiBench

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS with Tasks Priority (5)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Dynamic Memory Allocation - Constant Frequency (4)
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)
Static Memory Allocation - DFS with Tasks Priority (8)	

Table E–4: Significant algorithms in power reduction on ATMEGA323 running uGC code

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS with Tasks Priority (5)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS with Tasks Priority (8)
Dynamic Memory Allocation - Constant Frequency (4)	Static Memory Allocation - DFS Predictive Strategy based on WCET (7)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	

Table E–5: Significant algorithms in power reduction on MSP430F149 running uGC code

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Dynamic Memory Allocation - Constant Frequency (4)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	Dynamic Memory Allocation - DFS with Tasks Priority (5)
	Static Memory Allocation - DFS with Tasks Priority (8)

Table E–6: Significant algorithms in power reduction on LM3S811 running uGC

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	
Static Memory Allocation - DFS based on CPU Usage Strategy (6)	
Dynamic Memory Allocation - Constant Frequency (4)	

Table E–7: Significant algorithms in power reduction on MSP430F149 running basicmath

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - Constant Frequency (4)	
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	
Static Memory Allocation - DFS based on CPU Usage Strategy (6)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	

Table E–8: Significant algorithms in power reduction on MSP430F149 running dijkstra

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - Constant Frequency (4)	

Table E–9: Significant algorithms in power reduction on MSP430F149 running bitcount

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS with Tasks Priority (5)	Static Memory Allocation - DFS Predictive Strategy based on WCET (7)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	
Dynamic Memory Allocation - Constant Frequency (4)	

Table E–10: Significant algorithms in power reduction on ATMEGA323 running basicmath

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Static Memory Allocation - DFS with Tasks Priority (8)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Dynamic Memory Allocation - Constant Frequency (4)
Dynamic Memory Allocation - DFS with Tasks Priority (5)	Static Memory Allocation - DFS Predictive Strategy based on WCET (7)
	Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)

Table E–11: Significant algorithms in power reduction on ATMEGA323 running dijkstra

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - Constant Frequency (4)	
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	
Static Memory Allocation - DFS based on CPU Usage Strategy (6)	

Table E–12: Significant algorithms in power reduction on ATMEGA323 running bitcount

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Static Memory Allocation - DFS with Tasks Priority (8)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Static Memory Allocation - DFS Predictive Strategy based on WCET (7)
Dynamic Memory Allocation - Constant Frequency (4)	
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	
Dynamic Memory Allocation - DFS with Tasks Priority (5)	

Table E–13: Significant algorithms in power reduction on LM3S811 running basicmath

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	Dynamic Memory Allocation - DFS with Tasks Priority (5)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Static Memory Allocation - DFS with Tasks Priority (8)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	
Dynamic Memory Allocation - Constant Frequency (4)	

Table E–14: Significant algorithms in power reduction on LM3S811 running dijkstra

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Static Memory Allocation - DFS based on CPU Usage Strategy (6)	
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	
Static Memory Allocation - DFS with Tasks Priority (8)	
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	
Dynamic Memory Allocation - DFS with Tasks Priority (5)	
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	
Dynamic Memory Allocation - Constant Frequency (4)	

Table E–15: Significant algorithms in power reduction on LM3S811 running bitcount

Algorithms significantly different from control algorithm	**Not significantly different from control algorithm**
Static Memory Allocation - DFS Predictive Strategy based on WCET (7)	Static Memory Allocation - DFS with Tasks Priority (8)
Dynamic Memory Allocation - DFS with Tasks Priority (5)	Dynamic Memory Allocation - Constant Frequency (4)
Dynamic Memory Allocation - DFS based on CPU Usage Strategy (2)	Static Memory Allocation - DFS based on CPU Usage Strategy (6)
Dynamic Memory Allocation - DFS Predictive Strategy based on WCET (3)	

APPENDIX F
NORMALITY PLOT FOR THE SELECTED PLATFORMS

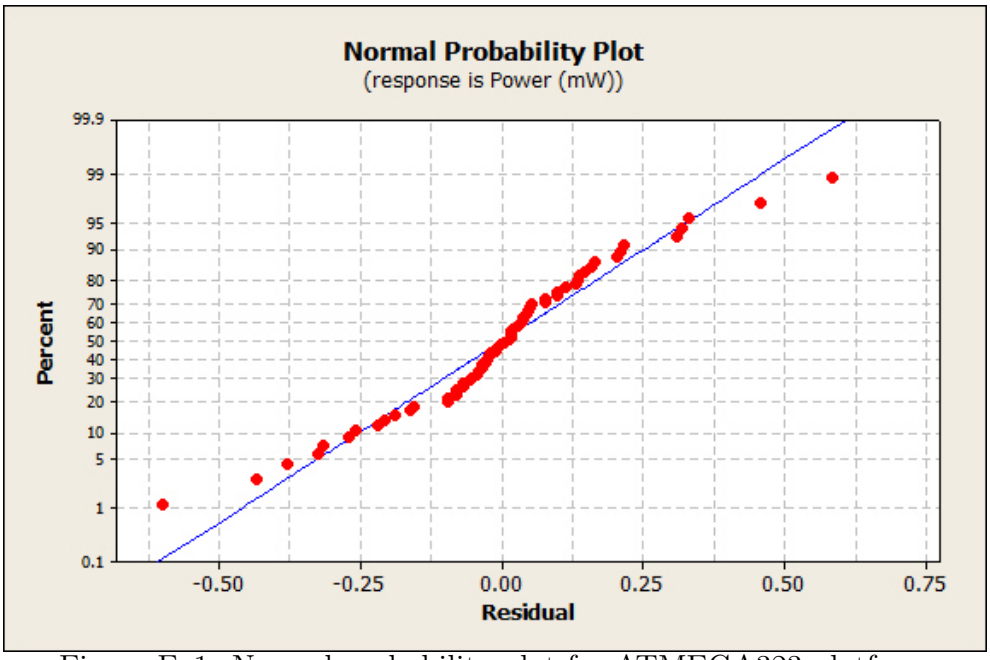


Figure F–1: Normal probability plot for ATMEGA323 platform

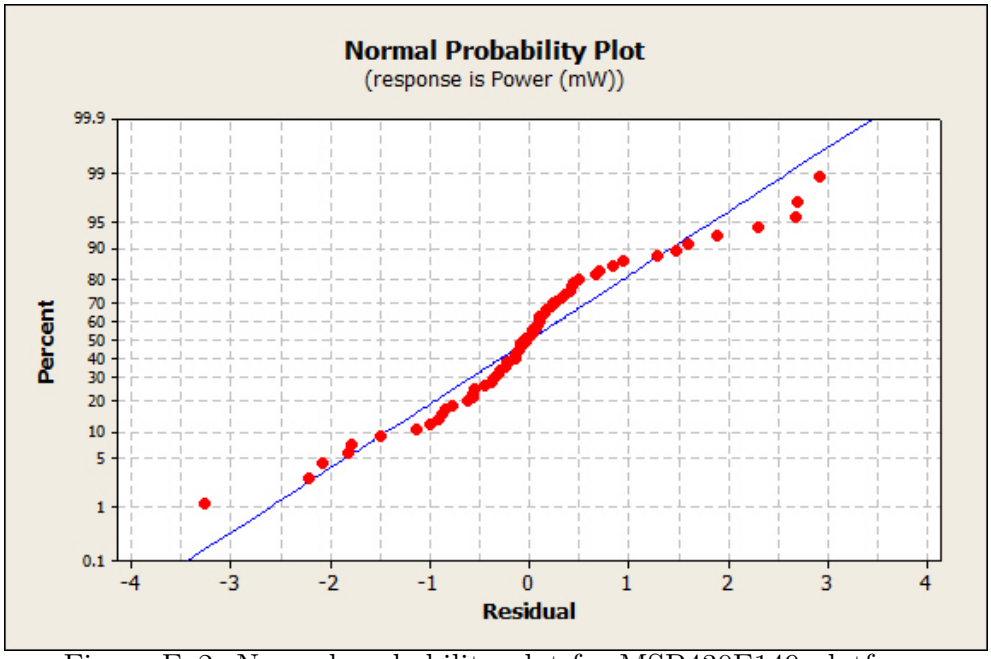


Figure F–2: Normal probability plot for MSP430F149 platform

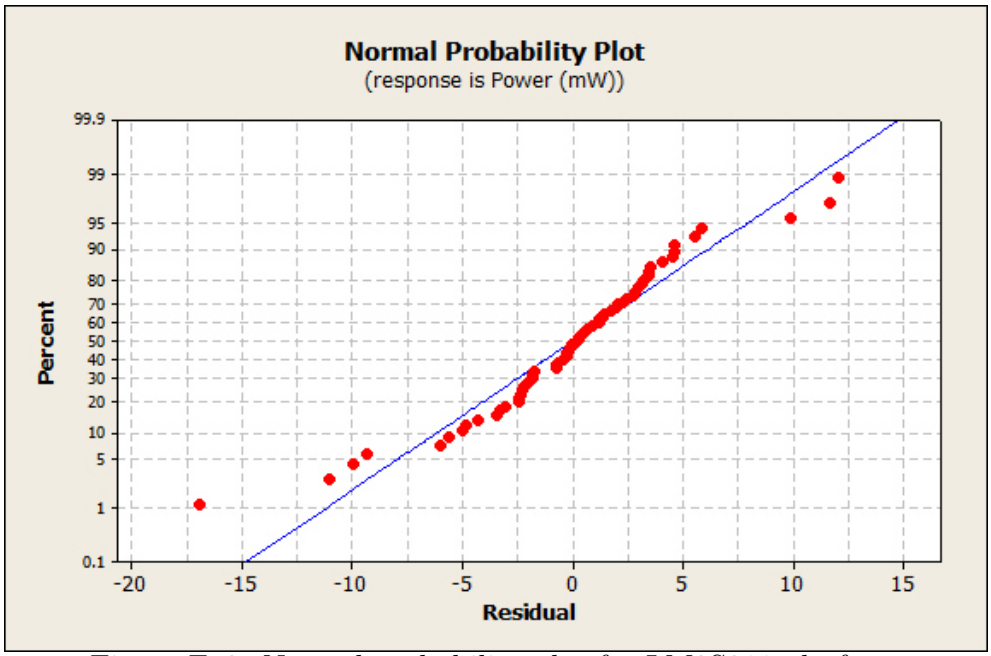


Figure F–3: Normal probability plot for LM3S811 platform

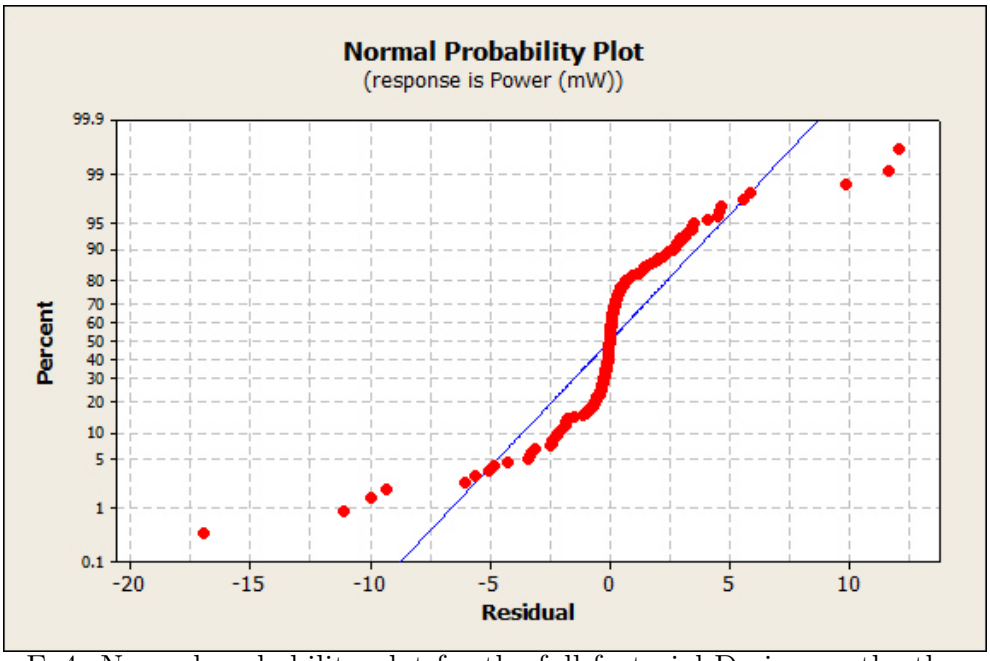


Figure F–4: Normal probability plot for the full factorial Design on the three Platforms

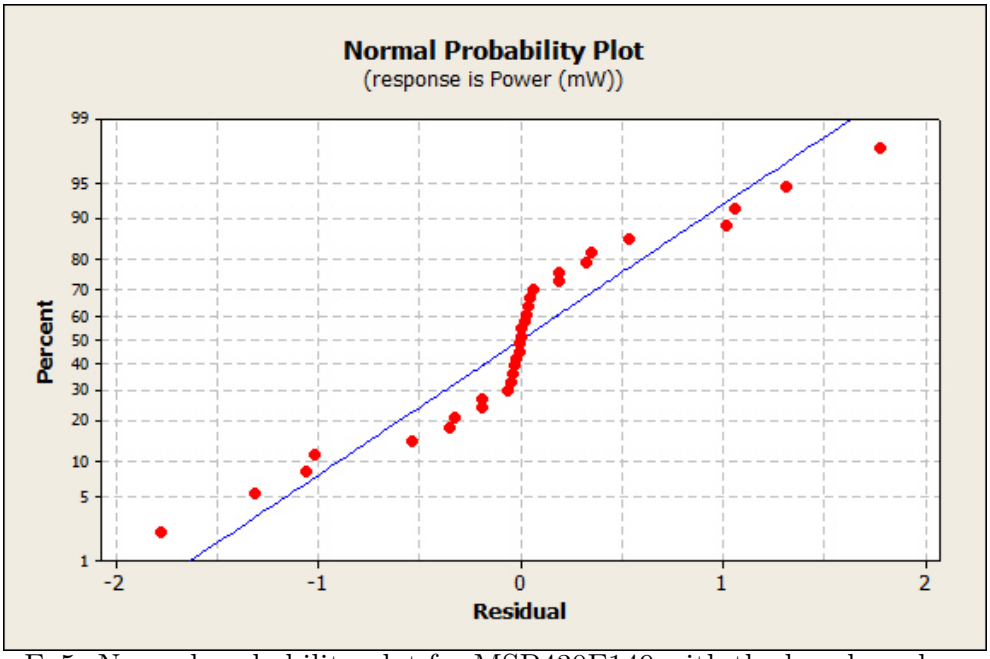


Figure F–5: Normal probability plot for MSP430F149 with the benchmarks running individually

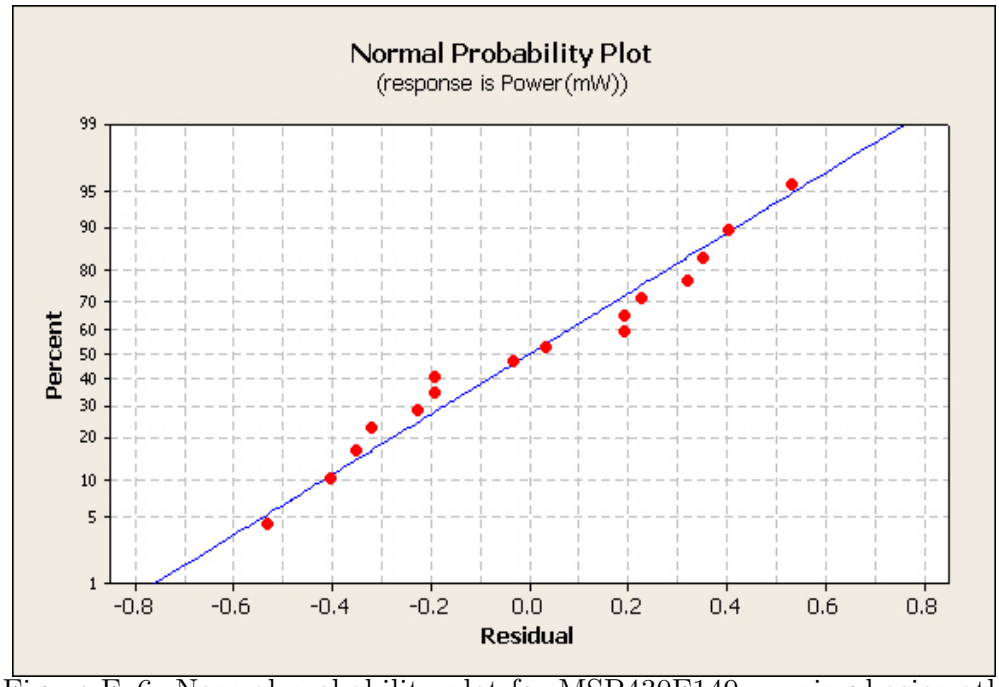


Figure F–6: Normal probability plot for MSP430F149 running basicmath

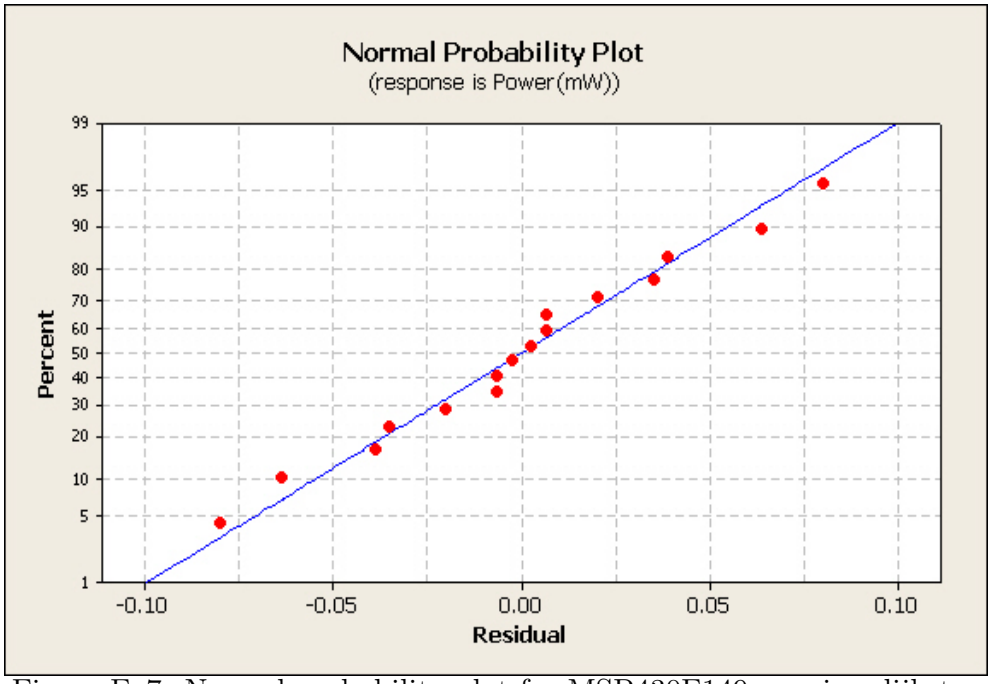


Figure F–7: Normal probability plot for MSP430F149 running dijkstra

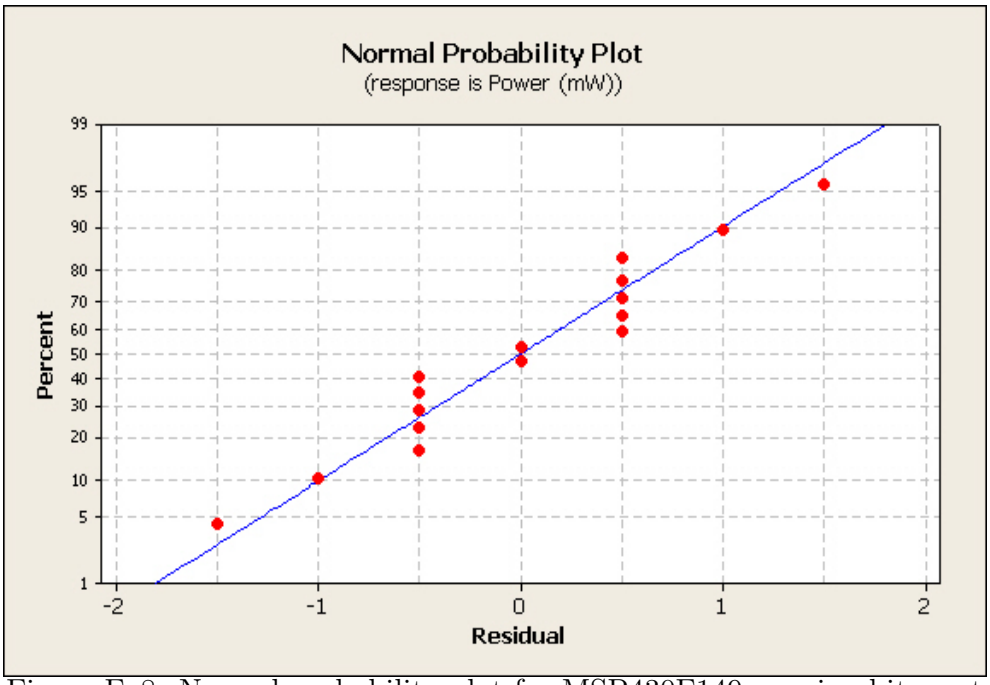


Figure F–8: Normal probability plot for MSP430F149 running bitcount

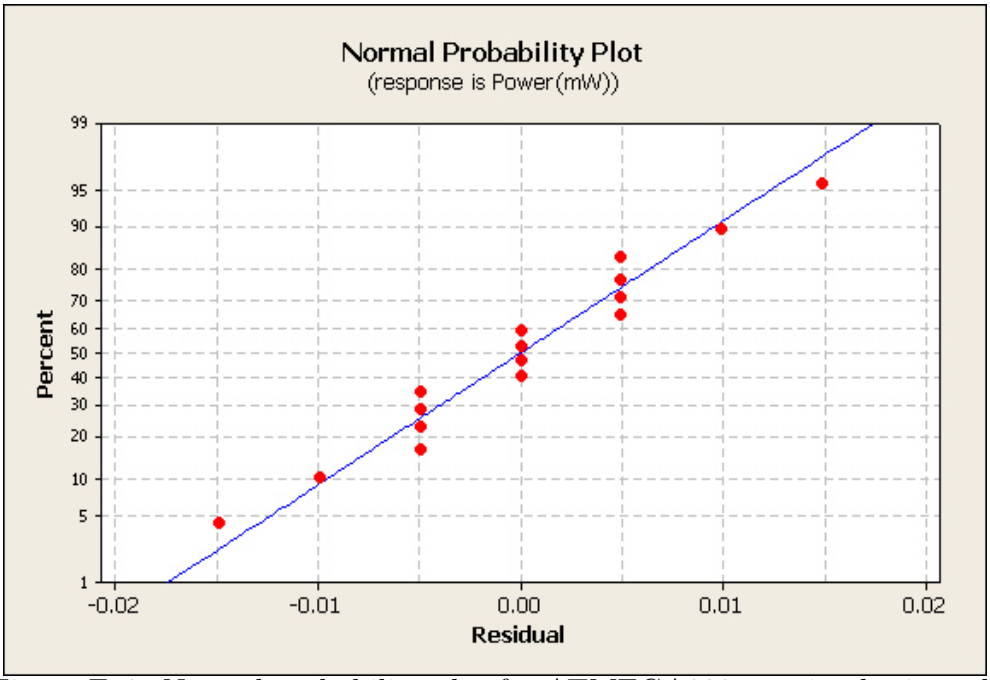


Figure F–9: Normal probability plot for ATMEGA323 running basicmath

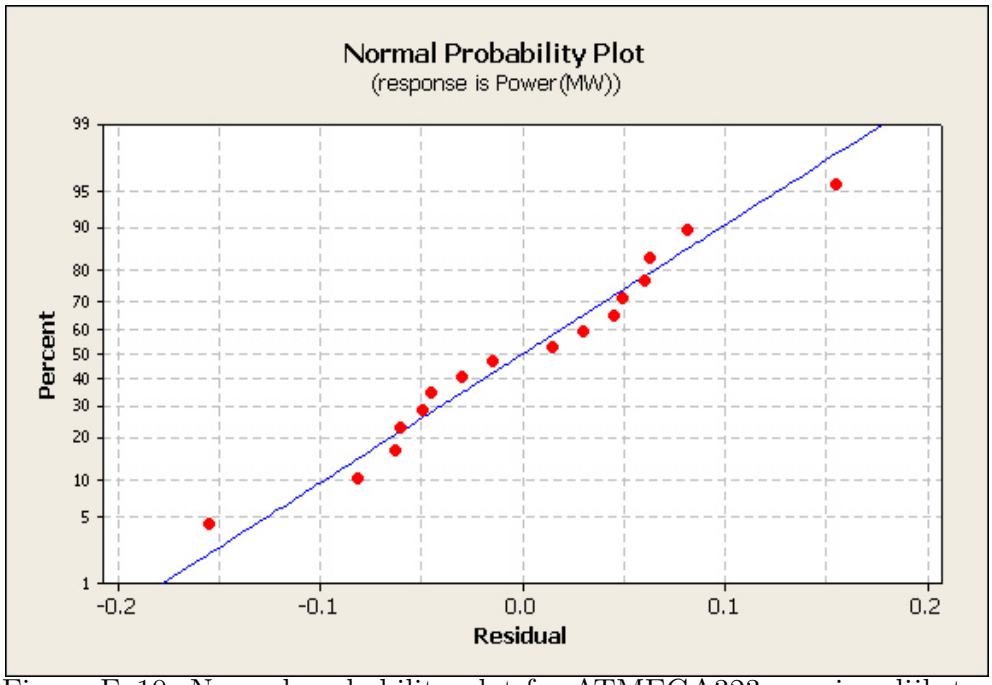


Figure F–10: Normal probability plot for ATMEGA323 running dijkstra

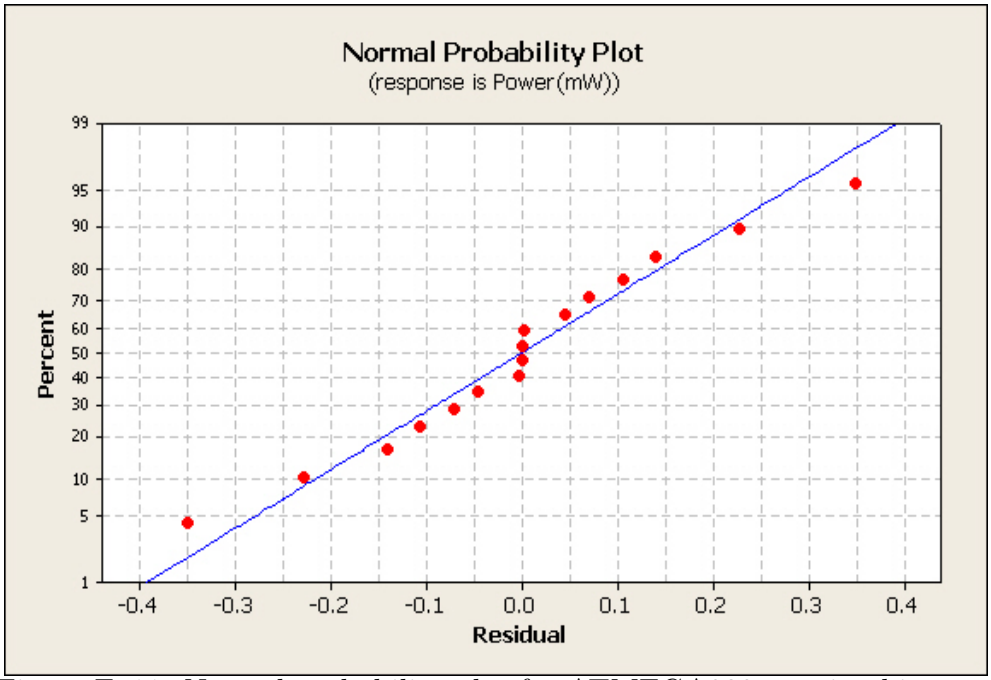


Figure F–11: Normal probability plot for ATMEGA323 running bitcount

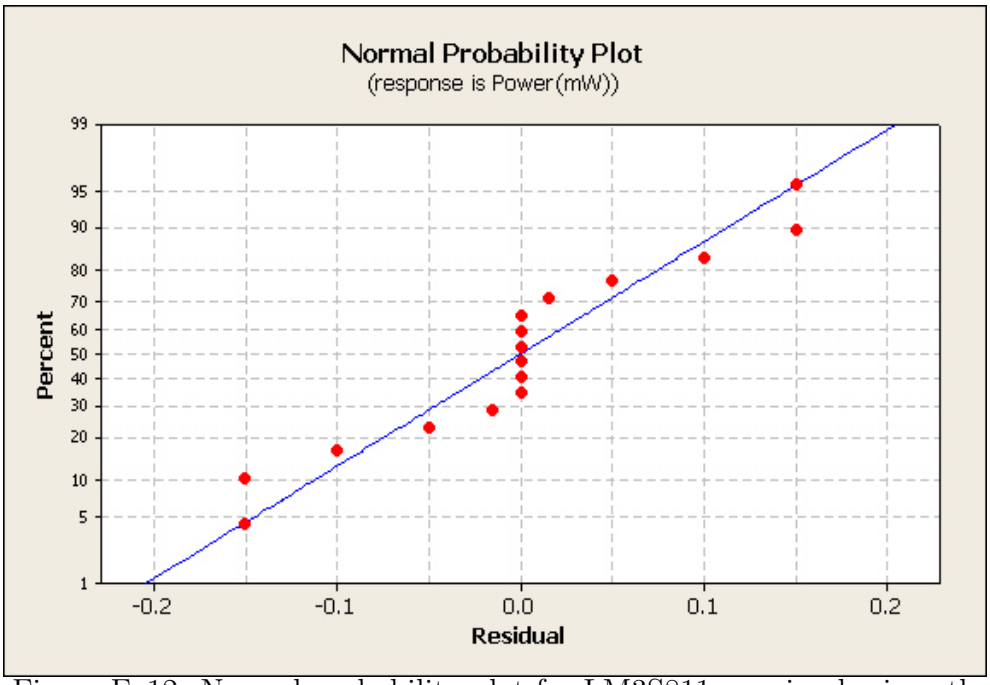


Figure F–12: Normal probability plot for LM3S811 running basicmath

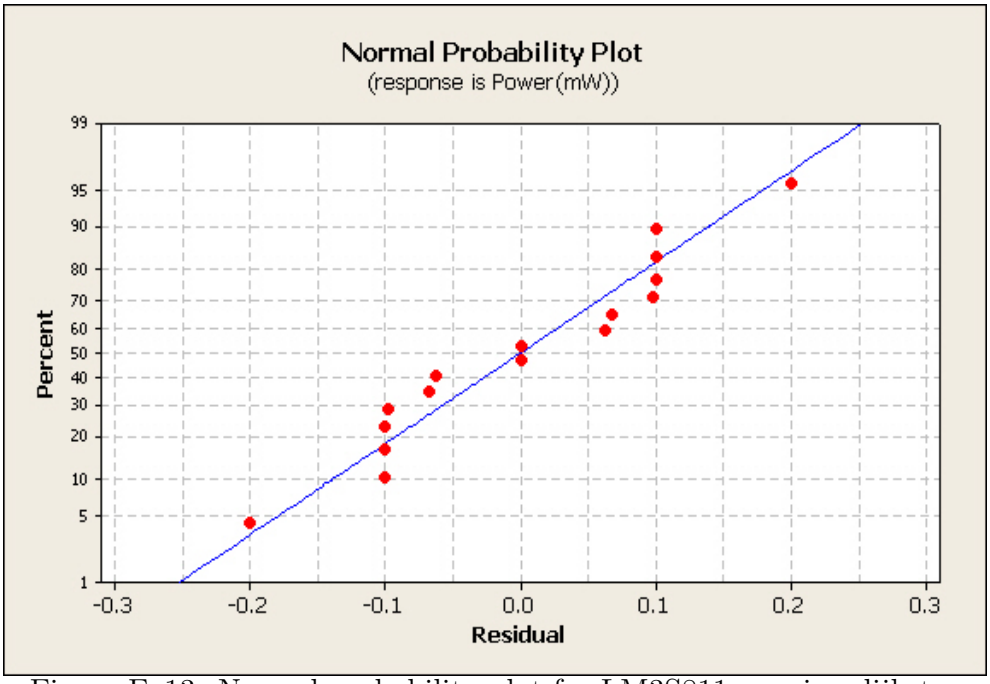


Figure F–13: Normal probability plot for LM3S811 running dijkstra

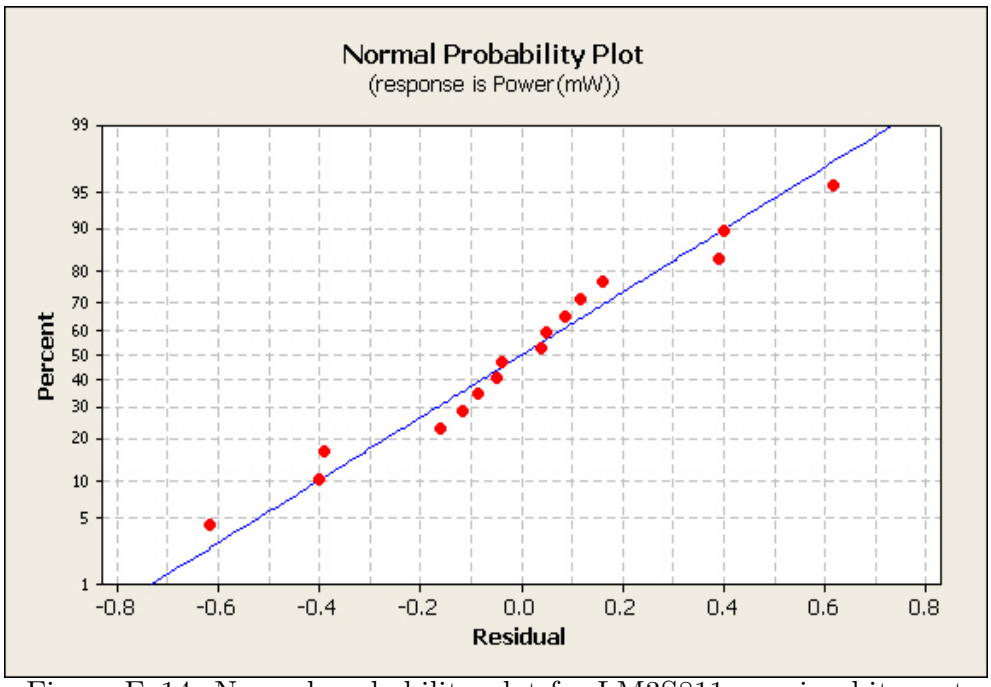


Figure F–14: Normal probability plot for LM3S811 running bitcount

REFERENCE LIST

[1] R. Kamal. *Embedded System: Architecture, Programming, and Design.* McGraw-Hill, New Delhi, India, 2008.

[2] O. S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE*, 91(7):1055–1069, July 2003.

[3] C. M. Krishna and Y. H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. *IEEE Trans. Comput.*, 52(12):1586–1593, 2003.

[4] V. Dalal and C.P. Ravikumar. Software power optimizations in an embedded system. In *VLSID '01: Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*, pages 254–259, Washington, DC, USA, 2001.

[5] V. Tiwari, S. Malik, A. Wolfe, and M. T. C. Lee. Instruction level power analysis and optimization of software. In *Proceedings of the Ninth International Conference on VLSI Design*, pages 326–328, Jan 1996.

[6] V. Tiwari, S. Malik, and A. Wolfe. Compilation techniques for low energy: an overview. In *IEEE Symposium on Low Power Electronics. Digest of Technical Papers.*, pages 38–39, Oct 1994.

[7] L. Benini and G. D. Micheli. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):115–192, April 2000.

[8] Y. Li and J. Henkel. A framework for estimating and minimizing energy dissipation of embedded hw/sw systems. *Proceedings of the Design Automation Conference*, pages 188–193, 1998.

[9] W. Wolf and M. Kandemir. Memory system optimization of embedded software. *Proceedings of the IEEE*, 91(1):165–182, Jan 2003.

[10] P. Kumar and M. Srivastava. Predictive strategies for low-power rtos scheduling. In *ICCD '00: Proceedings of the 2000 IEEE International Conference on Computer Design*, pages 343–348, Los Alamitos, CA, USA, 2000.

[11] A. Acquaviva, L. Benini, and B. Riccó. *Energy characterization of embedded real-time operating systems*, chapter 4, pages 53–73. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[12] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Analysis of power dissipation in embedded systems using real-time operating systems. *IEEE transactions on computer-aided design of integrated circuits and systems*, 22(5):615–627, May 2003.

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

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

[15] M.R. Minhas. An evolutionary algorithm for low power vlsi cell placement. In *Proceedings of the 46th International Midwest Symposium on Circuits and Systems*, volume 3, pages 1540–1543, Dec. 2003.

[16] Vivek Tiwari and Mike Tien-Chien Lee. Power analysis of a 32-bit embedded microcontroller. In *Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95. Design Automation Conference, IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific*, pages 141–148, 29 Aug - 1 Sep 1995.

[17] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, 2005.

[18] I. Hong, D. Kirovski, G. Qu, 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.

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

[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. In *IEEE Conference on Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03.*, volume 1, pages 496–505, Sept. 2003.

[21] A. M. Badulescu and A. Veidenbaum. Power efficient instruction cache for wide-issue processors. In *IWIA '01: Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'01)*, pages 12–15, Washington, DC, USA, 2001. IEEE Computer Society.

[22] H. Tomiyama, T. Ishihara, A. Inoue, and H. Yasuura. Instruction scheduling for power reduction in processor-based system design. In *Proceedings of the conference on Design, automation and test in Europe*, pages 855–860, Washington, DC, USA, Feb 1998.

[23] J. Zambreno, M. T. Kandemir, and A. N. Choudhary. Enhancing compiler techniques for memory energy optimizations. In *EMSOFT '02: Proceedings of the Second International Conference on Embedded Software*, pages 364–381, London, UK, 2002.

[24] S. T. Cheng, C. M. Chen, and J. W. Hwang. Low-power design for real-time systems. *Information, Communications and Signal Processing, 1997. ICICS.,*

Proceedings of 1997 International Conference on, 3:1746–1750, Sep 1997.

[25] Y. Yao, Q. Yao, P. Liu, and Z. Xiao. Embedded software optimization for mp3 decoder implemented on risc core. *IEEE Transactions on Consumer Electronics*, 50(4):1244–1249, Nov. 2004.

[26] D. Kim and K. Choi. Power-conscious high level synthesis using loop folding. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pages 441–445, New York, NY, USA, 1997.

[27] C. Chakrabarti. Cache design and exploration for low power embedded systems. In *IEEE International Conference on Performance, Computing, and Communications*, pages 135–139, Apr 2001.

[28] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man. Global communication and memory optimizing transformations for low power signal processing systems. In *VII Workshop on VLSI Signal Processing*, pages 178–187, Oct. 1994.

[29] F. Catthoor, E. de Greef, and S. Suytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[30] E. Macii, M. Pedram, and F. Somenzi. High-level power modeling, estimation, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1061–1079, Nov 1998.

[31] V. Swaminathan, K. Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 237–242, New York, NY, USA, 2001.

[32] A. Vahdat, A. Lebeck, and C. S. Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36, New

York, NY, USA, 2000. ACM.

[33] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. June 2006.

[34] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the data center. *Proceedings of the 13th international conference on Architectural support for programming languages and operating system*, 42(2):48–59, 2008.

[35] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, Littleton, Massachusetts, 1991.

[36] 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*, pages 3 – 14, Dec. 2001.

[37] H. Kim, W.H. Steinecker, S. Reidy, G.R. Lambertus, A.A. Astle, K. Najafi, E.T. Zellers, L.P. Bernal, P.D. Washabaugh, and K.D. Wise. A micropump-driven high-speed mems gas chromatography system. In *International Conference on Solid-State Sensors, Actuators and Microsystems, TRANSDUCERS 2007*, pages 1505–1508, June 2007.

[38] C. Lim, H. T. Ahn, and J. T. Kim. Predictive dvs scheduling for low-power real-time operating system. In *Proceedings of the 2007 International Conference on Convergence Information Technology*, pages 1918–1921, Washington, DC, USA, 2007.

[39] D. Atienza, S. Mamagkakis, M. Peon, F. Catthoor, J.M. Mendas, and D. Soudris. Power aware tuning of dynamic memory management for embedded real-time multimedia applications. In *Proceedings of the XIX Conference on Design of Circuits and Integrated Systems, DCIS'04*, volume 2, pages 375–380, Bourdeaux, France, Nov. 2004.

[40] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008.

[41] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low power cmos digital design. *IEEE Journal of Solid State Circuits*, 27:473–484, 1995.

[42] R. W. Erickson and D. Maksimovic. *Fundamentals of Power Electronics*. Springer, New Delhi, India, 2nd edition, 2001.

ANALYSIS OF LOW-POWER SOFTWARE TECHNIQUES FOR REAL TIME OPERATING SYSTEMS

Daniel Ernesto Mera Romo
(787) 529-8124
Department of Electrical and Computer Engineering
Chair: Nayda Santiago
Degree: Master of Science
Graduation Date: May 2010