PROJECTE DE FI DE CARRERA

TÍTOL DEL PFC: Benchmarks en diferents microprocessadors
TITULACIÓ: Enginyeria de Telecomunicació (segon cicle)
AUTOR: Josep Caubet Gomà
DIRECTOR: Bengt Koren. Royal Institute of Technology, Estocolm (Suècia)
DATA: 19 de juliol de 2006
**Títol:** Benchmarks en diferents microprocessadors

**Autor:** Josep Caubet Gomà

**Director:** Bengt Koren. Royal Institute of Technology, Estocolm (Suècia)

**Data:** 19 de juliol de 2006

---

**Resum**

L'objectiu d'aquest projecte és obtenir resultats de diferents benchmarks entre diferents plaques; RCM3720 de Rabbit Semiconductor i SNAP i IM3000 de Imsys Technologies.
S'han utilitzat tres llenguatges diferents de programació: C, Java i assembler. S'han executat un conjunt de diferents algoritmes en diferents microprocessadors, en un cas el processador va ser simulat a partir d'un simulador (El IM3000 Simulador de cicle real).
Els tests que s'han dut a terme tenien en compte diferents aspectes d'un microprocessador: el temps d'execució de diferents algoritmes, l'execució de benchmarks populars (Whetstone i Dhrystone) i el rendiment TCP/IP en l'SNAP.
L'anàlisi dels resultats obtinguts conclou que IM3000, programat en assembler, és la placa més ràpida. L'SNAP, programat en Java, és la més lenta. IM3000 triplica la velocitat de l'SNAP. RCM3720 és entre 5 i 10 vegades més lent que IM3000. La transferència de dades TCP/IP en l'SNAP està al voltant dels 4,7 Mbps (en ambdós sentits).
Overview

The objective of this thesis is to obtain results of benchmarking between different boards; RCM3720 of Rabbit Semiconductor and SNAP and IM3000 of Imsys Technologies.

Three different computer languages have been used: C, Java, and assembler. A set of algorithms have been run on different microprocessors and in one case the processor was simulated using a processor simulator (The IM3000 True cycle simulator).

The tests that have been carried out contain important different aspects of a microprocessor: the execution time of different algorithms, the execution of popular benchmarks (Whetstone and Dhrystone) and TCP/IP performances in SNAP.

The analysis of the obtained results concludes that IM3000, programmed in assembler is the quickest board. The SNAP, programmed in Java, is the slowest. IM3000 triplicates the SNAP speed. RCM3720 is around 5-10 times execution speed slower than IM3000. The transfer of data TCP/IP in SNAP is around 4,7 Mbps (both ways).
Acknowledgements

First of all, I would like to express my most sincere gratitude to Roger Sundman, my supervisor at Imsys Technologies AB for giving me the opportunity to perform this thesis, providing me with relevant information, and advice.

I would also like to thank all the people at Imsys for creating such a good atmosphere that made my work easier and more comfortable. Thanks to Christian Blixt, Anders Gradén and Per Alm for their technical help. Thanks to Yingqing Zhou and Sarat Samanta for their kind advice and support when I needed.

Many thanks also to Bengt Koren, my examiner and supervisor at the Royal Institute of Technology.

Last but not least. Many thanks to my parents for their love, unending support and encouragement over the years.


Josep Caubet Gomà
Acronyms

ARM – Acorn/Advanced RISC Machine

CAN – Controller Area Network

DMIPS – Dhrystone MIPS

EEMBC – Embedded Microprocessor Benchmark Consortium

HDLC – High-Level Data Link Control

I²C – Intelligent Interface Controller

ICE – In-Circuit Emulator

IDE – Integrated Development Environment

IP – Internet Protocol

IrDA – Infrared Data Association

JVM – Java Virtual Machine

MIPS – Million Instructions Per Second

POSIX – Portable Operating System Interface

RISC – Reduced Instruction Set Computing

SDLC – Synchronous Data Link Control

SPI – Serial Peripheral Interface

TCP – Transport Control Protocol

TDMI – Thumb Instruction, Debugger, Multiplier, ICE (ARM CPU features)

UART – Universal Asynchronous Receiver-Transmitter

USB – Universal Serial Bus

WIPS – Whetstone Instructions Per Second
# TABLE OF CONTENTS

INTRODUCTION .................................................................................................................. 9

CHAPTER 1. BACKGROUND ........................................................................................... 11
  1.1 Benchmarking .............................................................................................................. 11
  1.2 Embedded systems .................................................................................................... 11

CHAPTER 2. BOARD FEATURES .................................................................................... 13
  2.1 RCM3720 of Rabbit Semiconductor ........................................................................ 13
  2.2 SNAP of Imsys Technologies .................................................................................. 14
  2.3 IM3000 of Imsys Technologies ................................................................................. 16
  2.4 CerfBoard 270 Linux of Intrinsyc .......................................................................... 16
  2.5 Bitfire ARM of Arrow Engineering ........................................................................ 16
  2.6 Comparative table ................................................................................................. 17

CHAPTER 3. BENCHMARKS ........................................................................................... 19
  3.1 Execution time ........................................................................................................... 19
  3.2 Whetstone and Dhrystone ...................................................................................... 21
  3.3 TCP/IP performance .............................................................................................. 21

CHAPTER 4. RESULTS ..................................................................................................... 23
  4.1 Execution time ........................................................................................................... 23
  4.2 Whetstone and Dhrystone ...................................................................................... 25
  4.3 TCP/IP performance .............................................................................................. 26
  4.4 Comparing everything ......................................................................................... 28

CONCLUSIONS ................................................................................................................. 29

APPENDIX ......................................................................................................................... 31

APPENDIX A: OTHER STUDIED BOARDS...................................................................... 33
  A.1 CerfBoard 270 Linux of Intrinsyc .......................................................................... 33
  A.2 Bitfire ARM of Arrow Engineering ........................................................................ 34
  A.3 Comparative table ................................................................................................. 37

APPENDIX B: MEASURING EXECUTION TIME IN IMSYS DEVELOPER ...................... 39
  B.1 With `ftime()` function ........................................................................................ 39
  B.2 With the memory address ...................................................................................... 39

APPENDIX C: ENCOUNTERED PROBLEMS ................................................................ 43

APPENDIX D: DHRYSTONE’S CHARACTERISTICS ...................................................... 45

APPENDIX E: SET OF SIMPLE APPLICATIONS ......................................................... 49

REFERENCES ................................................................................................................... 63

BIBLIOGRAPHY ................................................................................................................ 65
INTRODUCTION

This report is the presentation of a Master Project Degree, part of Master of Science in Telecommunication Engineering at The Royal Institute of Technology, Stockholm, Sweden and the Technical University of Catalonia, Barcelona, Spain. The project was conducted between February and July of 2006 at Imsys Technologies AB “Imsys” in Upplands Väsby, Sweden.

The objective of this thesis is to obtain results in benchmarking of different microprocessors with an emphasis of Imsys proprietary architecture and comparison between an older versus a new processor design and with a special reference to power efficient computing. The organization of the report in chapters coincides with the steps that have been carried out to obtain the results.

The previous work and background information is described in the section Background followed by the section of Board Features, where the microprocessors are explained in more detail.

The chapter of Benchmarks shows the tests carried out and the different benchmark types that have been carried out using different programming languages (C, Asm and Java), including some generally well known benchmarks (Dhrystone and Whetstone). A specific TCP/IP test program was also included in the test suite.

Finally, after having carried out the planning and the execution of the benchmarks to fulfil the thesis, a conclusion with an analysis of the obtained results is made.
A large number of benchmarks have been carried out using more than 10 different algorithms or tasks. Three different computer languages have been used, (C, Asm and Java) and the combinations of algorithms and languages have been executed on several microprocessors and in one case the processor, the new Imsys IM3000, was simulated using a true cycle processor simulator. This was some of the basic prerequisite to carry out this project.

A large portion of time have been spent on learning and programming the different boards and languages used and also to set up the different units with their power supplies, cables, software, and other needed housekeeping functions. (For more information, see CHAPTER 2).

### 1.1 Benchmarking

To define benchmarking [1] is difficult since there are of many types (according to the objective that is pursued) but basically and without going into details, a benchmark [2] is the result of running an algorithm, or a set of algorithms, in order to assess the relative performance of an object, by running a number of standard tests and trials. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, but there are circumstances when the technique is also applicable to software, for example, efficiency of different compilers with respect to for example code size or execution time.

Benchmarks provide a method of comparing the performance across different chip/system architectures.

There are many benchmarks types that can be carried out with the processors, for example of energy consumption, of algorithm execution time, code size, compiler optimization, communication efficiency and so on. The benchmarking makes sense against processors from other vendors to compare features. For this reason it is very important to make clear the context of the test (frequency of the processor, word length, bus width etc).

### 1.2 Embedded systems

An embedded system is usually designed in order to do a very specific function as efficient and cost effective as possible and unlike general-purpose computers such as the PC. An embedded system performs well and pre-defined tasks, usually with very specific requirements. Since the system is dedicated to a specific task, design engineers can optimize it, thereby reducing it to the least possible size and cost of the product.
The core of any embedded system is formed by one or more microprocessors or microcontrollers, programmed to perform well defined tasks. The software on an embedded system is usually semi-permanent; and it is often called "firmware" [6].

Embedded and real-time systems aren’t synonymous. Most embedded systems are real-time and most real-time systems are embedded [7].

Figure 1. Most embedded systems are real-time and most real-time systems are embedded
CHAPTER 2. BOARD FEATURES

In this chapter the features of the used boards will be explained: ARM2129 of Arrow Engineering, RCM3720 of Rabbit Semiconductor, CerfBoard 270 Linux of Intrinsyc, SNAP and IM3000 of Imsys Technologies.

In all case but one, the actual tests were run using available equipment and in one case a software simulator for the IM3000.

2.1 RCM3720 of Rabbit Semiconductor


The board used in this thesis is RCM3720 that belongs to the RCM3700 RabbitCore Family. This embedded system is characterized to be the lowest priced that incorporates the Rabbit 3000 processor.

![RCM3720 Ethernet Connection Kit](image)

Figure 2. RCM3720 Ethernet Connection Kit

Also, this board includes Dynamic C development system, an integrated C compiler, editor, loader, and debugger designed specifically for the Rabbit microprocessor. The Dynamic C integrated development environment provides a platform for developing applications.
The main characteristics of this board can be seen in Table 1.

Table 1. RabbitCore RCM3720 main specifications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manufacturer</strong></td>
<td>Rabbit Semiconductor</td>
</tr>
<tr>
<td><strong>Microprocessor</strong></td>
<td>Rabbit 3000 at 22.1 MHz</td>
</tr>
<tr>
<td><strong>Bits</strong></td>
<td>8</td>
</tr>
<tr>
<td><strong>Flash</strong></td>
<td>512 KB</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>256 KB SRAM</td>
</tr>
<tr>
<td><strong>Ethernet port</strong></td>
<td>10Base-T</td>
</tr>
<tr>
<td><strong>Interfaces</strong></td>
<td>IrDA</td>
</tr>
<tr>
<td></td>
<td>SDLC/HDLC</td>
</tr>
<tr>
<td></td>
<td>I2C</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>33 digital I/O</td>
</tr>
<tr>
<td><strong>Programming languages</strong></td>
<td>C Asm</td>
</tr>
</tbody>
</table>

2.2 SNAP of Imsys Technologies

Imsys designs and supplies networked microprocessor solutions to OEMs in the market of Embedded Control and Telematic systems. Imsys introduced the SNAP microprocessor module in 2003 and is now about to release to the market its new generation micro, the IM3000 (see IM3000 in 2.3).

SNAP is a high performance, low power and a network-ready, Java-powered plug & play reference platform. The processor, IM1000, can be programmed in C/C++, assembler and Java. For efficient Java execution, most Java byte codes are part of a special instruction sets. Moreover shell features provide easy development over Ethernet using Telnet and FTP.
Imsys Developer [Figure 5] is the Visual Development Environment from Imsys. It handles a simultaneous mix of Java, C and assembler code. Also included is an advanced simulator that allows application software to be developed and executed without the need of a hardware target.

The main characteristics of this board can be seen in Table 2.

Table 2. SNAP main specifications

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Imsys Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessor</td>
<td>IM1000 at 40 MHz</td>
</tr>
<tr>
<td>Bits</td>
<td>8</td>
</tr>
<tr>
<td>Flash</td>
<td>2 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>8 MB DRAM</td>
</tr>
<tr>
<td>Ethernet port</td>
<td>10/100Base-T</td>
</tr>
<tr>
<td>Interfaces</td>
<td>3 serial ports</td>
</tr>
<tr>
<td></td>
<td>Wire</td>
</tr>
<tr>
<td></td>
<td>I²C</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>CAN</td>
</tr>
<tr>
<td>Programming languages</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Asm</td>
</tr>
<tr>
<td></td>
<td>JAVA</td>
</tr>
</tbody>
</table>
2.3 IM3000 of Imsys Technologies

IM3000 is the new generation of processors of Imsys Technologies [see 2.2]. This processor incorporates important features in addition to IM1000.

The prototype of this new device is not yet available and therefore the tests made in this thesis are based on simulations carried out with Imsys Developer [see Figure 5].

<table>
<thead>
<tr>
<th>Table 3. IM3000 main specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
</tr>
<tr>
<td>Microprocessor</td>
</tr>
<tr>
<td>Bits</td>
</tr>
<tr>
<td>Flash</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Ethernet port</td>
</tr>
<tr>
<td>Interfaces</td>
</tr>
<tr>
<td>Programming languages</td>
</tr>
</tbody>
</table>

2.4 CerfBoard 270 Linux of Intrinsyc

The CerfBoard 270 Linux has been studied but it has not been able to execute any benchmark in it. In [APPENDIX C:] the problems are explained. The specifications of the board can see in [A.1].

2.5 Bitfire ARM of Arrow Engineering

The Bitfire ARM has been studied but it has not been included in the comparative table [see 2.6]. It works with 32 bits so doesn't make sense compare it with boards of 8 bits. The specifications of the board can see in [A.2].
### 2.6 Comparative table

In the [Table 4], the summary of comparison of different boards is presented.

Table 4. Boards specific features

<table>
<thead>
<tr>
<th></th>
<th>RCM3720</th>
<th>SNAP</th>
<th>IM3000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manufacturer</strong></td>
<td>Rabbit Semiconductor</td>
<td>Imsys Technologies</td>
<td>Imsys Technologies</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>RABBIT 3000</td>
<td>IM1000</td>
<td>IM3000</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>22.1 MHz</td>
<td>40 MHz</td>
<td>67 MHz</td>
</tr>
<tr>
<td><strong>Bits</strong></td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td><strong>Flash</strong></td>
<td>512 KB</td>
<td>2 MB</td>
<td>-</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>256 KB SRAM</td>
<td>8 MB DRAM</td>
<td>80 KB ROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40 KB SRAM</td>
</tr>
<tr>
<td><strong>Ethernet port</strong></td>
<td>10Base-T</td>
<td>10/100Base-T</td>
<td>Dual 10/100Base-T</td>
</tr>
<tr>
<td><strong>Development kit</strong></td>
<td>Dynamic C</td>
<td>Imsys Developer</td>
<td>Imsys Developer</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>Dynamic C</td>
<td>UCC</td>
<td>UCC</td>
</tr>
<tr>
<td><strong>Languages</strong></td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Asm</td>
<td>Asm</td>
<td>Asm</td>
</tr>
<tr>
<td></td>
<td>JAVA</td>
<td>JAVA</td>
<td>JAVA</td>
</tr>
<tr>
<td><strong>Processor price (10k)</strong></td>
<td>$9.25</td>
<td>$10</td>
<td>$15</td>
</tr>
</tbody>
</table>

For more precise information on these boards and others studied, please consult [APPENDIX A].
CHAPTER 3. BENCHMARKS

This chapter contains the benchmarks [see 1.1] carried out on the different boards [see CHAPTER 2].

Three sets of benchmarks was performed, one set of 11 applications mainly used in smaller, 8-bit, systems the Dhrystone and Whetstone benchmark, and a benchmark for determining the TCP/IP performances in the combination of the SNAP and a PC.

There are numerous aspects that influence directly the results, the architecture of the processor, the frequency, the code optimization through the compilers, etc.

The results of the benchmarks can be seen in [CHAPTER 4].

3.1 Execution time

Each algorithm (set of simple applications) [see APPENDIX E:] has been executed on the different boards with the aim of measuring the execution time. These algorithms have been executed using C, Java and Asm in all combinations except for the Rabbit since it lacks support for Java.

All of the set of simple applications, except the last one, come from MSP430 Competitive Benchmarking of Texas Instrument [9].

The execution time for the algorithm is very small so a number of loops where used to get a time in the milliseconds range [see Figure 6]. [APPENDIX B:] explains how the time was measured in the SNAP and IM3000.

![Procedure to calculate the execution time](image)

Figure 6. Procedure to calculate the execution time

In [Table 5] the features of each algorithm are explained. Source code can be found in the [APPENDIX E:].
Table 5. Features of set of simple applications

<table>
<thead>
<tr>
<th>Source file name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit_math.c</td>
<td>Source file containing three math functions. One function performs addition of two 8 bit numbers, one performs multiplication, and one performs division. The “main()” function calls each of these functions.</td>
</tr>
<tr>
<td>16-bit_math.c</td>
<td>Source file containing three math functions. One function performs addition of two 16 bit numbers, one performs multiplication, and one performs division. The “main()” function calls each of these functions.</td>
</tr>
<tr>
<td>32-bit_math.c</td>
<td>Source file containing three math functions. One function performs addition of two 32 bit numbers, one performs multiplication, and one performs division. The “main()” function calls each of these functions.</td>
</tr>
<tr>
<td>floating_point_math.c</td>
<td>Source file containing three math functions. One function performs addition of two floating-point numbers, one performs multiplication, and one performs division. The “main()” function calls each of these functions.</td>
</tr>
<tr>
<td>8-bit_switch_case.c</td>
<td>Source file with one function containing a switch statement having 16 cases. An 8 bit value is used to select a particular case. The “main()” function calls the “switch” function with an input parameter selecting the last case.</td>
</tr>
<tr>
<td>16-bit_switch_case.c</td>
<td>Source file with one function containing a switch statement having 16 cases. A 16 bit value is used to select a particular case. The “main()” function calls the “switch” function with an input parameter selecting the last case.</td>
</tr>
<tr>
<td>8-bit_2-dim_matrix.c</td>
<td>Source file containing 3 two-dimensional arrays containing 8 bit values–one of which is initialized. The “main()” function copies values from array 1 to array 2, then from array 2 to array 3.</td>
</tr>
<tr>
<td>16-bit_2-dim_matrix.c</td>
<td>Source file containing 3 two-dimensional arrays containing 16 bit values–one of which is initialized. The “main()” function copies values from array 1 to array 2, then from array 2 to array 3.</td>
</tr>
<tr>
<td>fir_filter.c</td>
<td>Source file containing code that calculates the output from a 17-coefficient tap filter using simulated ADC input data.</td>
</tr>
<tr>
<td>matrix_multiplication.c</td>
<td>Source file containing code that multiplies a 3x4 matrix by a 4x5 matrix.</td>
</tr>
<tr>
<td>RSA_multiplication.c</td>
<td>Source file containing multiplication code.</td>
</tr>
</tbody>
</table>

The obtained results and their graphic representation are shown in the following section [see 4.1].
3.2 Whetstone and Dhrystone

The Whetstone and Dhrystone benchmark was introduced at the time of minicomputers and workstations. Today they have almost no meaning but companies like MIPS and ARM still publish this kind of number for each new processor design. When this benchmark was designed the PC was not even invented and different type of computations, integer or floating point was the main use for the equipment of the time. Today need is dominated by streaming data, driven by the Internet and TCP/IP.

However, they can be interesting to use such as to measure the performance of a given architecture family. So is the case with Imsys SNAP and the IM3000. In this case both the performance increase and power efficiency (MIPS/P), can be measured. The power efficiency is becoming more and more important with battery power devices.

3.2.1 Whetstone benchmark

The Whetstone benchmark was written in November of 1972 by Harold Curnow of CCTA in Algol 60, based on work by Brian Wichmann of the National Physical Laboratory [4]. The Whetstone benchmark primarily measures the floating-point arithmetic performance.

The Whetstone benchmark originally produced speed ratings in terms of Thousands of Whetstone Instructions Per Second (KWIPS), later produced speed ratings in MOPS (Millions of Operations Per Second) and MFLOPS (Floating Point) and overall rating in MWIPS (= 1000 KWIPS).

3.2.2 Dhrystone benchmark

Developed in 1984 by Rheinhold Weicker, Dhrystone is a benchmark program written in C or Pascal (and now even in Java) that tests a system's integer performance and string handling. The program is CPU bound, performing no I/O functions or operating system calls [3].

There are two Dhrystone versions 1.1 (1984) and the current version Dhrystone 2.1 (1988) [5]. Anyway Dhrystone doesn't take into account some details [see APPENDIX D].

Dhrystones per second is the typical metric used to measure the number of times the program can run in a second. Although there are derived calculations, most scores are reported as Dhrystone MIPS/megahertz (abbreviated as DMIPS/MHz) and/or as VAX Dhrystone MIPS (sometimes just called DMIPS) [5].

3.3 TCP/IP performance

It is possible to determine the TCP/IP performances of the SNAP that has Ethernet port. In this sense it has been developed a client and a server that data are sent among them, the purpose is to determine rate of transmission of data in uplink and in downlink.
The client is always a PC and the server is SNAP [see Figure 7]. Downlink refers when SNAP receives data and uplink when it sends data.

Figure 7. Scenario for determining TCP/IP performance

In the [Figure 8] the procedure for measure the TCP/IP performance is shown. Initially the client informs the server of the characteristics of the test that will be carried out sending the size of the packet, the number of packets to send and the number of times that will repeat the test (loops).

Figure 8. Diagram of the procedure of the benchmark

Immediately afterwards the client or the server – according to if it is uplink or downlink – it sends the number of packets of defined size. This size will fix to 1460 bytes (payload) and knowing that the headers are of 54 bytes, the total of the packet will be 1514 bytes, in fact the maximum size of a packet TCP [see Figure 9]. This way it is known that quantity of bytes is sent and its payload, let us remember that if the data overcome the 1514 bytes, the protocol TCP/IP cuts them in blocks of this same size.

Figure 9. TCP packet
CHAPTER 4. RESULTS

The results of executing the different benchmarks [see CHAPTER 3] are shown in this chapter.

### 4.1 Execution time

The [Table 6] briefly describe for each system the available languages, number of bits and frequency used.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Language</th>
<th>Bits</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCM3720</td>
<td>Rabbit semiconductor</td>
<td>C</td>
<td>8 bits</td>
</tr>
<tr>
<td>SNAP_C / J / ASM</td>
<td>Imsys Technologies</td>
<td>C / J / Asm</td>
<td>8 bits</td>
</tr>
<tr>
<td>IM3000_C / J / ASM</td>
<td>Imsys Technologies</td>
<td>C / J / Asm</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

The figure given in Table 10 is execution time in milliseconds for all boards except the IM3000 that are simulated.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Boards</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IM3000_ASM</td>
</tr>
<tr>
<td>8 bit Math</td>
<td>6</td>
</tr>
<tr>
<td>8 bit 2 Dim Matrix</td>
<td>221</td>
</tr>
<tr>
<td>8 bit Switch Case</td>
<td>2</td>
</tr>
<tr>
<td>16 Bit Math</td>
<td>7</td>
</tr>
<tr>
<td>16 Bit 2 Dim Matrix</td>
<td>221</td>
</tr>
<tr>
<td>16 Bit Switch Case</td>
<td>2</td>
</tr>
<tr>
<td>32 Bit Math</td>
<td>7</td>
</tr>
<tr>
<td>Floating-point Math</td>
<td>10</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>1794</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>283</td>
</tr>
<tr>
<td>RSA Multiplication</td>
<td>65262</td>
</tr>
</tbody>
</table>

The [Figure 11] shows the proportion of time that each board spends to execute an application. It is easy to see in this graph the quickest and slowest board.

In the [Figure 10] shows the order of the different boards in relation to the execution time.

![Figure 10. Orderly boards according to the execution time](image-url)
Figure 11. Proportion of time that each board spends to execute an application
4.2 Whetstone and Dhrystone

4.2.1 Whetstone

The obtained results of applying this benchmark can be seen in the [Table 8]. The execution of Whetstone in some boards has not been possible [see APPENDIX C:]

Table 8. Whetstone results

<table>
<thead>
<tr>
<th></th>
<th>KWIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCM3720</td>
<td>256,4</td>
</tr>
<tr>
<td>SNAP_C</td>
<td>322,6</td>
</tr>
<tr>
<td>IM3000</td>
<td>1021,03</td>
</tr>
</tbody>
</table>

In [Figure 12] one can see the graphic representation of the results.

IM3000 has three more times KWIPS faster that SNAP_C. RCM3720 is a little below SNAP.

4.2.2 Dhrystone

The obtained results of applying this benchmark are visualized in the [Table 9]. The execution of Whetstone in some boards has not been possible [see APPENDIX C:]

Table 9. Dhrystone 2.1 results (LOOP = 100.000, C code)

<table>
<thead>
<tr>
<th></th>
<th>Dhrystones per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAP_C</td>
<td>4.347,8</td>
</tr>
<tr>
<td>IM3000</td>
<td>12.853,5</td>
</tr>
</tbody>
</table>

In [Figure 13] one can see the graphic representation of the results.
The relation among boards of the [Figure 10] it is like in Whetstone benchmark. IM3000 has three more times Dhrystones per second that SNAP_C.

### 4.3 TCP/IP performance

To obtain results of this benchmark it has been developed a client and a server. Initially it was developed a client and a server in Java. It was checked their correct operation starting from Ethereal – a network protocol analyzer – with which the sent and received packets were studied. Nevertheless, the results were not satisfactory because the obtained rate didn't fulfil the predictions. They were obtained around some 400 Kbps of maximum rate in the uplink and about 40 Kbps in the downlink.

The reason that was identified in its moment is that Java consumes many resources and this affects to the performances of this type of tests, more in an embedded system, since the same performance tests worked without any problem when being executed between two PCs [Figure 14].

To solve this problem another client-server it was developed in C with which the wanted results were obtained.

In the [Table 10] TCP uplink performance is shown, where SNAP sends packets (total packet size: 1514 bytes payload: 1460 bytes) to a PC that finally calculates Rate payload and Rate total data.
Table 10. TCP rate uplink performance

<table>
<thead>
<tr>
<th>Size packets</th>
<th>Num Packets</th>
<th>SNAP SEND DATA</th>
<th>PC RECEIVE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total Payload (Mbits)</td>
<td>Total data (Mbits)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>11,68</td>
<td>12,11</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>23,36</td>
<td>24,22</td>
</tr>
<tr>
<td></td>
<td>5000</td>
<td>58,40</td>
<td>60,56</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>116,80</td>
<td>121,12</td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>233,60</td>
<td>242,24</td>
</tr>
<tr>
<td></td>
<td>40000</td>
<td>467,20</td>
<td>484,48</td>
</tr>
<tr>
<td></td>
<td>100000</td>
<td>1168,00</td>
<td>1211,20</td>
</tr>
</tbody>
</table>

In the [Table 11] TCP downlink performance is shown, where SNAP receives packets (total packet size: 1514 bytes payload: 1460 bytes) of a PC.

Table 11. TCP rate downlink performance

<table>
<thead>
<tr>
<th>Size packets</th>
<th>Num Packets</th>
<th>PC SEND DATA</th>
<th>SNAP RECEIVE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total Payload (Mbits)</td>
<td>Total data (Mbits)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>11,68</td>
<td>12,11</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>23,36</td>
<td>24,22</td>
</tr>
<tr>
<td></td>
<td>5000</td>
<td>58,40</td>
<td>60,56</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>116,80</td>
<td>121,12</td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>233,60</td>
<td>242,24</td>
</tr>
<tr>
<td></td>
<td>40000</td>
<td>467,20</td>
<td>484,48</td>
</tr>
<tr>
<td></td>
<td>100000</td>
<td>1168,00</td>
<td>1211,20</td>
</tr>
</tbody>
</table>

The [Figure 15] shows the uplink and downlink transmission rates.

![Figure 15. TCP rates uplink and downlink](image)

The maximum rate of transmission in uplink and in downlink is around 4,7 Mbps. In downlink the rate is constant independently of the number of sent packets, on the other hand in uplink, when SNAP sends data, the rate increases with the number of sent packets. The relation is proportional until the 5000 packets.
### 4.4 Comparing everything

In this section, the obtained results of the previous sections will be compared among them. The [Figure 16] shows a graph that relates benchmark performance and price.

In the graph there are as many points as boards and programming languages [see Table 6].

In the [Figure 16], the benchmark performance showed in the X axis is the execution time obtained from the set of simple applications [see 3.1].

The [Figure 16], obtained from [Table 7], have been done in the following way:

- First, the percentage of time that each board invests in executing an algorithm, has been calculated.
- Then, with these results, the total percentage of time (axis X), that each board spend in executing the algorithms, has been calculated.

The percentage of prices (in relation of the addition of all prices) is showed in the axis Y.

![Figure 16. Comparison among boards](image-url)
CONCLUSIONS

- Conclusions of the benchmarks

This thesis shows the results of the benchmarking carried out to different boards: RCM3720 of Rabbit Semiconductor, SNAP and IM3000 of Imsys Technologies. The tests that have been carried out contain important different aspects of a microprocessor, the execution time, the execution of popular benchmarks (Whetstone and Dhrystone) and TCP/IP performances in SNAP.

A large number of benchmarks have been carried out using more than 10 different algorithms or tasks. Three different computer languages have been used, (C, Java and Asm) and the combinations of algorithms have been running on different microprocessors and in one case the processor was simulated using a processor simulator (The IM3000 True cycle simulator).

The analysis of the obtained results concludes that IM3000 (programmed in Asm) is the quickest board and SNAP (programmed in Java) is the slowest.

IM3000, the third generation of processors of Imsys, triplicates the speed of SNAP (programmed in Asm or C). With RCM3720 is around 5-10 times execution speed faster.

Two popular benchmarks, Whetstone and Dhrystone have been executed too. Both are programmed in C and they show the Whetstone Instructions Per Second and Dhrystones per second units respectively.

The Dhrystone relation between SNAP and IM3000 is approximately 3. This relation in Whetstone benchmark stays and the number of Whetstones Instructions per second in RCM3720 is a little below SNAP.

The transfer of data TCP in SNAP, around 4,7 Mbps (both ways), fulfil for example the necessary specifications to be installed in a printer without any problem.

This thesis has also required important knowledge about networks and operating systems since it has been had to develop a client and a server TCP with sockets, first in Java and later in C on different platforms (Windows and Linux).

Finally, the difference between boards and its development environments (except SNAP and IM3000) has added complication to the project. It has been programmed in JAVA, C (with different variants according to the board) and Asm (with also different variants, for example the RCM3720 didn't have the division function). The adaptation of Dhrystone and Whetstone have been difficult, in some boards it has been impossible to execute them. Therefore, the arisen problems have gone adapting the objectives throughout the project.
• Future work

This thesis leaves the way open to enlarge and to go more deeply into the obtained benchmarks. Below follow a few suggestions on further investigations.

- Code size. Establishing some calculation approaches before.
- Dhrystone and Whetstone in other boards.
- UDP performances.
- TCP performances in other boards.
- To try to solve the encountered problems [see APPENDIX C;].
APPENDIX
APPENDIX A: OTHER STUDIED BOARDS

Other boards have been studied but it has not been included in the comparative table [see 2.6] with different reasons.

Bitfire ARM works with 32 bits so didn't make sense compare it with boards of 8 bits. On the other hand, any benchmark had not been able to execute in the CerfBoard 270 Linux. In [APPENDIX C:] the problems are explained.

A.1 CerfBoard 270 Linux of Intrinsyc

CerfBoard 270 Linux is a new board developed by Intrinsyc Software, mobility software and services company, based in Vancouver, Canada.

The CerfBoard 270 for Linux [see Figure 17] is a high-performance, low-power design that is a starting point for developing Internet devices. The system is based on the Intel XScale platform, and includes a preloaded Linux v2.6 Kernel.

The board incorporates 2 Compact Flash connectors that can be used to add Bluetooth, Wireless LAN support or additional memory as well as a Secure Digital / MultiMedia Card connector. Also includes 10/100 Ethernet, 2 Serial Ports, three USB ports, a VGA LCD connector, audio connectors, etc.

Figure 17. CerfBoard 270 Linux of Intrinsyc

The terminal emulation program for Windows [see Figure 18] allows controlling all that happens in the board, besides to download and to upload files.
With this board, the results of the benchmarks have not been obtained. The problems that have seen are explained in the [APPENDIX C:].

The main characteristics of this board can be seen in Table 2.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Intrinsyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessor</td>
<td>PXA270 by XScale Intel family at 416 MHz</td>
</tr>
<tr>
<td>Bits</td>
<td>32</td>
</tr>
<tr>
<td>Flash</td>
<td>16 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>64 MB SRAM</td>
</tr>
<tr>
<td>Ethernet port</td>
<td>10/100Base-T</td>
</tr>
<tr>
<td>Interfaces</td>
<td>2 serial ports</td>
</tr>
<tr>
<td></td>
<td>2 USB 2.0</td>
</tr>
<tr>
<td></td>
<td>1 USB 1.1 client &amp; host support</td>
</tr>
<tr>
<td></td>
<td>4 wire, I2C</td>
</tr>
<tr>
<td>Programming languages</td>
<td>C</td>
</tr>
</tbody>
</table>

**A.2 Bitfire ARM of Arrow Engineering**

This board belongs to the Bitfire project which was started by the "ARMSchool" project group at Arrow Engineering.

Bitfire has an ARM7TDMI from Philips with 256KB Flash and 16KB RAM. It also includes an FPGA from Altera and a big high intensity bicolour LED-matrix (total 1280 LEDs).
Bitfire provides the advanced embedded systems developer with a low-cost ARM7 based development kit. The kit also supports a low-cost FPGA, which opens possibilities for a wide variety of applications, and to run and debug algorithms from RAM and Flash.

No additional software or hardware is required to develop or debug tools. It’s all based around the GNU GCC/GDB tool suite, and it runs on all platforms supported by GCC. To install and set up the environment under Windows, Cygwin is used [see Figure 20] that implements a POSIX layer (Portable Operating System Interface) on Windows, therefore any application that fulfils this standard will be able to execute under Windows. A unique source code [8] allows the portability in different platforms.

The Bitfire platform also offer Eclipse, an open extensible IDE that requires Java to run. It is used for project management, build control and debugging. It sits on-top of the GCC/GDB tools and uses Makefiles for building.

To download the hex files in Flash the Philips Flash Utility is used. In [Figure 21] the configuration options are shown for the correct transfer of the data.
The main characteristics of this board can be seen in [Table 13].

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Arrow Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessor</td>
<td>ARM7TDMI at 60 MHz</td>
</tr>
<tr>
<td>Bits</td>
<td>32</td>
</tr>
<tr>
<td>Flash</td>
<td>256 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>16 KB RAM</td>
</tr>
<tr>
<td>Ethernet port</td>
<td>10Base-T</td>
</tr>
<tr>
<td>Interfaces</td>
<td>2 serial ports</td>
</tr>
<tr>
<td></td>
<td>2 CAN</td>
</tr>
<tr>
<td></td>
<td>I²C</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
</tr>
<tr>
<td>Programming languages</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Asm</td>
</tr>
</tbody>
</table>

**A.2.1 Execution time results**

The [Table 14] shows the execution time [see 3.1] obtained of running a set of simple applications [see APPENDIX E:].

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Bitfire ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit Math</td>
<td>4</td>
</tr>
<tr>
<td>8 bit 2 Dim Matrix</td>
<td>92</td>
</tr>
<tr>
<td>8 bit Switch Case</td>
<td>1</td>
</tr>
<tr>
<td>16 Bit Math</td>
<td>5</td>
</tr>
<tr>
<td>16 Bit 2 Dim Matrix</td>
<td>96</td>
</tr>
<tr>
<td>16 Bit Switch Case</td>
<td>1</td>
</tr>
<tr>
<td>32 Bit Math</td>
<td>6</td>
</tr>
<tr>
<td>Floating-point Math</td>
<td>7</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>1100</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>236</td>
</tr>
<tr>
<td>RSA Multiplication</td>
<td>26000</td>
</tr>
</tbody>
</table>
A.3 Comparative table

The [Table 15] shown all the boards studied in this thesis.

Table 15. All boards specific features

<table>
<thead>
<tr>
<th></th>
<th>RCM3720</th>
<th>Bitfire ARM</th>
<th>SNAP</th>
<th>IM3000</th>
<th>CerfBoard 270 Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Rabbit Semiconductor</td>
<td>Arrow Engineering</td>
<td>Imsys Technologies</td>
<td>Imsys Technologies</td>
<td>Intrinsyc</td>
</tr>
<tr>
<td>Processor</td>
<td>RABBIT 3000</td>
<td>ARM7TDMI</td>
<td>IM1000</td>
<td>IM3000</td>
<td>PXA270 XScale Intel family</td>
</tr>
<tr>
<td>Frequency</td>
<td>22.1 MHz</td>
<td>60 MHz</td>
<td>40 MHz</td>
<td>67 MHz</td>
<td>416 MHz</td>
</tr>
<tr>
<td>Bits</td>
<td>8</td>
<td>32</td>
<td>8</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>Power Supply</td>
<td>4.75 to 5.25 V</td>
<td>10 to 23 V</td>
<td>5 V</td>
<td>3.0 to 3.6 V</td>
<td>6 to 12 V</td>
</tr>
<tr>
<td>Temperature range</td>
<td>-40° to +70° C</td>
<td>-20° to +70° C</td>
<td>-40° to +85° C</td>
<td>0° to +50°C</td>
<td></td>
</tr>
<tr>
<td>Flash</td>
<td>512 KB</td>
<td>256 KB</td>
<td>2 MB</td>
<td>-</td>
<td>16 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>256 KB SRAM</td>
<td>16 KB RAM</td>
<td>8 MB DRAM</td>
<td>80 KB ROM</td>
<td>64 MB SDRAM</td>
</tr>
<tr>
<td>Ethernet port</td>
<td>10Base-T</td>
<td>-</td>
<td>10/100Base-T</td>
<td>Dual 10/100Base-T</td>
<td>10/100Base-T</td>
</tr>
<tr>
<td>Interfaces</td>
<td>IrDA</td>
<td>2 serial ports</td>
<td>3 serial ports</td>
<td>3 serial ports</td>
<td>2 serial ports</td>
</tr>
<tr>
<td></td>
<td>SDLC/HDLC</td>
<td>2 CAN</td>
<td>Wire</td>
<td>I²C</td>
<td>2 USB 2.0</td>
</tr>
<tr>
<td></td>
<td>I²C</td>
<td>I²C</td>
<td>I²C</td>
<td>SPI</td>
<td>1 USB 1.1 client &amp; host support</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
<td>SPI</td>
<td>SPI</td>
<td>CAN</td>
<td>4 wire, I²C</td>
</tr>
<tr>
<td></td>
<td>33 digital I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development kit</td>
<td>Dynamic C</td>
<td>Eclipse</td>
<td>Imsys Developer</td>
<td>Imsys Developer</td>
<td>HyperTerminal Shell</td>
</tr>
<tr>
<td>Compiler</td>
<td>Dynamic C</td>
<td>GCC Compiler</td>
<td>UCC</td>
<td>UCC</td>
<td>GCC</td>
</tr>
<tr>
<td>Languages</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>Asm</td>
<td>Asm</td>
<td>Asm</td>
<td>Asm</td>
<td>Asm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>4 PWM output channels</td>
<td>Big high intensity bicolor LED-matrix</td>
<td>4 canals A/D</td>
<td>8 channel A/D (16 bit) 2 channel D/A (16 bit)</td>
<td>Linux environment 2 CompactFlash connec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Audio connectors</td>
</tr>
<tr>
<td>Processor price (10k)</td>
<td>$9.25</td>
<td>$7.4</td>
<td>$10</td>
<td>$15</td>
<td>$32</td>
</tr>
</tbody>
</table>
APPENDIX B: Measuring execution time in Imsys Developer

Measuring the time of execution of an algorithm [see Figure 6] in SNAP and IM3000 it is possible to make it in several ways how it explains next.

B.1 With `ftime()` function

The library time.h has the function `ftime(&tb)` that returns the current time in seconds `[tb.time]` and milliseconds `[tb.millitm]`.

If you proceed in the following way the time of execution of the algorithm it is obtained:

```c
#include <stdio.h>
#include <time.h>

void main(void)
{
    unsigned short aa_mm,bb_ms;
    long aa_s,bb_s;
    struct timeb tb;

    ftime(&tb); //Start
    aa_s = tb.time;
    aa_ms = tb.millitm;

    //Code to measure the time of execution
    ftime(&tb); //Stop
    bb_s = tb.time;
    bb_ms = tb.millitm;

    printf("Time: \%d s and \%d ms",(bb_s-AA_s),(bb_ms-aa_ms));
}
```

B.2 With the memory address

It is possible to visualize the time of execution of the algorithm with the memory address 0xFE4, but not without before to activate a bit:

*(unsigned char *)0xFD0 |= 0x08;

The time in milliseconds is shown of 0xFE4 - 0xFE7. If one wants more precision in 0xFE8 - 0xFE9 parts of 1/65536 are visualized [see Figure 22].
To calculate the time, it is only necessary to subtract the time START and time STOP [see Figure 6]. To make it simpler, the time START will be erased and we will only notice time STOP to know the time of execution.

The function in Asm `eraseTime` erases these memory addresses:

```asm
._eraseTime:
    ldw.a 0FE4H
    ld 0
    st
```

The code would be in the following way:

```c
#include <stdio.h>

void main(void)
{
    *(unsigned char *)0xFD0 |= 0x08;
    eraseTime();
    //Code to measure the time of execution
    //Look time lapsed in the memory
}
```

In any case a deviation of time takes place since when erasing these addresses by memory it lapses time and also times of Imsys Developer also influences. The best option to measure the time is the Second option [B.2.2].

**B.2.2 Second option**

This option solves the problems explained in the previous section.
The code to execute would be:

```c
#include <stdio.h>

void main(void)
{
    unsigned int t1,t2;
    *(unsigned char *)0xFD0 |= 0x08;
    t1 = *(unsigned int*)0xFE4; //Start
    //Code to measure the time of execution
    t2 = *(unsigned int*)0xFE4; //Stop
    printf ("Time: %d ms",(t2-t1));
}
```
APPENDIX C: Encountered problems

This section lists and explains the problems that have been found throughout the thesis and they have not found solution.

- **TTCP benchmark** [10]

TTCP is a benchmarking tool for measuring TCP and UDP performance. It was possible to execute the application satisfactorily between two PCs, but not in SNAP, due to problems related with unknown functions.

- **CerfBoard 270 Linux of Intrinsyc**

This board works with the operating system Linux. It was configured the net connection satisfactorily. According to the manufacture’s recommendations they set the packages of development IPKG but it was not possible to compile any program.

As alternative, Cygwin (explained previously in this thesis) was used that allows to work with an environment Linux in Windows. I settled the development software for processors ARM Intel XScale with the objective of building in the PC and later to download the executable to CerfBoard 270 Linux. The result neither was satisfactory.

- **Dhrystone benchmark**

It was not possible the execution of this benchmark in RCM3720 of Rabbit Semiconductor, due to problems related with unknown functions.
APPENDIX D: Dhrystone’s characteristics

To use a popular benchmark has their advantages but sometimes also some disadvantages like Dhrystone.

The [Table 16] provides a concise summary of Dhrystone’s characteristics and corresponding strength or weakness [5]:

Table 16. Dhrystone’s strength and weakness

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Strength and Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Written in C language code</td>
<td>Strength: Allows code to be ported to a large number of platforms and architectures.</td>
</tr>
<tr>
<td>Very small size</td>
<td>Strength: An engineer can quickly master Dhrystone.</td>
</tr>
<tr>
<td></td>
<td>Weakness:</td>
</tr>
<tr>
<td></td>
<td>- A compiler writer, or architect, can quickly defeat Dhrystone and &quot;design to a benchmark.&quot;</td>
</tr>
<tr>
<td></td>
<td>- Minimizes or eliminates stress on memory subsystems and easily fits inside L1 caches.</td>
</tr>
<tr>
<td></td>
<td>- Cannot hope to mimic the breadth of applications encountered by a processor-based system.</td>
</tr>
<tr>
<td></td>
<td>- Is based on a single benchmark comprised of three files: dhry_1.c, dhry_2.c, and dhry.h. There is only one set of functions.</td>
</tr>
<tr>
<td>Single, easy-to-report score</td>
<td>Strength: Reported as a single figure of merit, similar to the ‘marks’ used by EEMBC, has allowed it to gain industry traction. Dhrystone is formally reported as “Dhrystone 2.1 MIPS”.</td>
</tr>
<tr>
<td></td>
<td>Weakness:</td>
</tr>
<tr>
<td></td>
<td>- Dhrystone users employ confusing and ambiguous terminology such as DMIPS, DMIPS/MHz, Rounded Dhrystones/second, and Dhrystones/CPU cycle. Furthermore, a &quot;MIP&quot; is actually 1.75 DEC VAX MIPS.</td>
</tr>
<tr>
<td>Synthetic</td>
<td>Weakness: Dhrystone only measures a few mathematical and basic operations.</td>
</tr>
<tr>
<td>Integer only code</td>
<td>Strength:</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>- This makes it potentially useful for simple 8 and 16-bit microcontrollers, assuming people don't care about relating anything to real world applications.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Does not measure multiply accumulate, floating-point, SIMD, or any other type of operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library dependent performance</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Dhrystone's execution is largely spent in standard C library functions, such as <code>strcmp()</code>, <code>strcpy()</code>, and <code>memcp()</code>. Compiler vendors generally provide these libraries that are typically optimized and hand-written in assembly language. While you may think you are benchmarking a processor, you are really benchmarking are the compiler writer’s optimizations of the C library functions for a particular platform.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Evolution</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Compiler writers have long ago determined Dhrystone's functionality. The secret to good benchmarks, as SPEC and EEMBC have shown, is to stay ahead of the compiler writers to ensure that the processor and system is benchmarked, not just the compiler.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Third-Party Certification</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Dhrystone's lack of an official certification process has eliminated this benchmark's credibility. Certification can only come from inherent value, and there is very little value in Dhrystone to modern processors or compilers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Source Control</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Dhrystone is available from multiple sources, and while most companies attempt to use Weicker's original source, some servers have &quot;gone dark&quot; as the age of the Web increases. There is great potential that a company, or an individual, has modified the code to its advantage. Some companies report Dhrystone 1.1 scores - an even older version of the code.</td>
</tr>
</tbody>
</table>

| No Standard Run Rules            | Weakness: |
Due to the lack of a standards organization, Dhrystone’s original runtime rules have eroded into a state of confusion, thereby turning it into a performance measurement that is easily circumvented.

<table>
<thead>
<tr>
<th>No Disclosure of Benchmark Environment</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The benchmarking environment, including processor and memory clock speed, compiler switches, and libraries, are not disclosed nor required.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inlining or excessive compiler optimization destroys the benchmark</th>
<th>Weakness:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Instructing the compiler to inline the code, greatly increasing the benchmark's susceptibility to code elimination, typically breaks Dhrystone's apocryphal &quot;rules&quot;. The benchmark essentially vanishes and scores get unrealistically good.</td>
</tr>
</tbody>
</table>
APPENDIX E: Set of simple applications

The algorithms that have been used in the benchmarks are included in this section. All of them, except the last one, come from *MSP430 Competitive Benchmarking* of Texas Instrument [9].

8-bit Math.c

```c
/************************************************************************
* Name : 8-bit Math
* Purpose : Benchmark 8-bit math functions.
* 
* ************************************************************************/

typedef unsigned char UInt8;

UInt8 add(UInt8 a, UInt8 b)
{
    return (a + b);
}

UInt8 mul(UInt8 a, UInt8 b)
{
    return (a * b);
}

UInt8 div(UInt8 a, UInt8 b)
{
    return (a / b);
}

void main(void)
{
    volatile UInt8 result[4];
    result[0] = 12;
    result[1] = 3;

    result[2] = add(result[0], result[1]);
    result[1] = mul(result[0], result[2]);
    result[3] = div(result[1], result[2]);

    return;
}
```
8-bit 2-dim Matrix.c

/*************************************************************************
 * Name : 8-bit 2-dim Matrix
 * Purpose : Benchmark copying 8-bit values.
 **************************************************************************/

typedef unsigned char UInt8;

const UInt8 m1[16][4] = {
    {0x12, 0x56, 0x90, 0x34},
    {0x78, 0x12, 0x56, 0x90},
    {0x34, 0x78, 0x12, 0x56},
    {0x90, 0x34, 0x78, 0x12},
    {0x12, 0x56, 0x90, 0x34},
    {0x78, 0x12, 0x56, 0x90},
    {0x34, 0x78, 0x12, 0x56},
    {0x90, 0x34, 0x78, 0x12},
    {0x12, 0x56, 0x90, 0x34},
    {0x78, 0x12, 0x56, 0x90},
    {0x34, 0x78, 0x12, 0x56},
    {0x90, 0x34, 0x78, 0x12},
    {0x12, 0x56, 0x90, 0x34},
    {0x78, 0x12, 0x56, 0x90},
    {0x34, 0x78, 0x12, 0x56},
    {0x90, 0x34, 0x78, 0x12}
};

void main (void)
{
    int i, j;
    volatile UInt8 m2[16][4], m3[16][4];

    for(i = 0; i < 16; i++)
    {
        for(j=0; j < 4; j++)
        {
            m2[i][j] = m1[i][j];
            m3[i][j] = m2[i][j];
        }
    }

    return;
}
8-bit Switch Case.c

/************************************************************************
* Name : 8-bit Switch Case
* Purpose : Benchmark accessing switch statement using 8-bit value.
*************************************************************************/

typedef unsigned char UInt8;

UInt8 switch_case(UInt8 a)
{
    UInt8 output;

    switch (a)
    {
        case 0x01:
            output = 0x01;
            break;

        case 0x02:
            output = 0x02;
            break;

        case 0x03:
            output = 0x03;
            break;

        case 0x04:
            output = 0x04;
            break;

        case 0x05:
            output = 0x05;
            break;

        case 0x06:
            output = 0x06;
            break;

        case 0x07:
            output = 0x07;
            break;

        case 0x08:
            output = 0x08;
            break;

        case 0x09:
            output = 0x09;
            break;

        case 0x0a:
            output = 0x0a;
            break;
    }

    return output;
}
case 0x0b:
    output = 0x0b;
    break;

case 0x0c:
    output = 0x0c;
    break;

case 0x0d:
    output = 0x0d;
    break;

case 0x0e:
    output = 0x0e;
    break;

case 0x0f:
    output = 0x0f;
    break;

case 0x10:
    output = 0x10;
    break;
} /* end switch*/

    return (output);

} /* end switch*/

void main(void)
{
    volatile UInt8 result;

    result = switch_case(0x10);

    return;
}
typedef unsigned short UInt16;

UInt16 add(UInt16 a, UInt16 b)
{
    return (a + b);
}

UInt16 mul(UInt16 a, UInt16 b)
{
    return (a * b);
}

UInt16 div(UInt16 a, UInt16 b)
{
    return (a / b);
}

void main(void)
{
    volatile UInt16 result[4];
    result[0] = 231;
    result[1] = 12;
    result[2] = add(result[0], result[1]);
    result[1] = mul(result[0], result[2]);
    result[3] = div(result[1], result[2]);

    return;
}
16-bit 2-dim Matrix.c

/************************************************************************
* Name : 16-bit 2-dim Matrix
* Purpose : Benchmark copying 16-bit values.
*************************************************************************/

typedef unsigned short UInt16;

const UInt16 m1[16][4] = {
    {0x1234, 0x5678, 0x9012, 0x3456},
    {0x7890, 0x1234, 0x5678, 0x9012},
    {0x3456, 0x7890, 0x1234, 0x5678},
    {0x9012, 0x3456, 0x7890, 0x1234},
    {0x1234, 0x5678, 0x9012, 0x3456},
    {0x7890, 0x1234, 0x5678, 0x9012},
    {0x3456, 0x7890, 0x1234, 0x5678},
    {0x9012, 0x3456, 0x7890, 0x1234},
    {0x1234, 0x5678, 0x9012, 0x3456},
    {0x7890, 0x1234, 0x5678, 0x9012},
    {0x3456, 0x7890, 0x1234, 0x5678},
    {0x9012, 0x3456, 0x7890, 0x1234},
    {0x1234, 0x5678, 0x9012, 0x3456},
    {0x7890, 0x1234, 0x5678, 0x9012},
    {0x3456, 0x7890, 0x1234, 0x5678},
    {0x9012, 0x3456, 0x7890, 0x1234}
};

void main(void)
{
    int i, j;
    volatile UInt16 m2[16][4], m3[16][4];

    for(i = 0; i < 16; i++)
    {
        for(j = 0; j < 4; j++)
        {
            m2[i][j] = m1[i][j];
            m3[i][j] = m2[i][j];
        }
    }

    return;
}
16-bit Switch Case.c

/*************************************************************************
* Name : 16-bit Switch Case  
* Purpose : Benchmark accessing switch statement using 16-bit value.  
**************************************************************************/

typedef unsigned short UInt16;

UInt16 switch_case(UInt16 a)
{
  UInt16 output;

  switch (a)
  {
  case 0x0001:
    output = 0x0001;
    break;

  case 0x0002:
    output = 0x0002;
    break;

  case 0x0003:
    output = 0x0003;
    break;

  case 0x0004:
    output = 0x0004;
    break;

  case 0x0005:
    output = 0x0005;
    break;

  case 0x0006:
    output = 0x0006;
    break;

  case 0x0007:
    output = 0x0007;
    break;

  case 0x0008:
    output = 0x0008;
    break;

  case 0x0009:
    output = 0x0009;
    break;

  case 0x000a:
    output = 0x000a;
    break;
  }
case 0x000b:
    output = 0x000b;
    break;

case 0x000c:
    output = 0x000c;
    break;

case 0x000d:
    output = 0x000d;
    break;

case 0x000e:
    output = 0x000e;
    break;

case 0x000f:
    output = 0x000f;
    break;

case 0x0010:
    output = 0x0010;
    break;
} /* end switch*/

return (output);

} /* end main */

void main(void)
{
    volatile UInt16 result;

    result = switch_case(0x0010);

    return;
}
32-bit Math.c

/*************************************************************************
* Name : 32-bit Math
* Purpose : Benchmark 32-bit math functions.
**************************************************************************/

#include <math.h>

typedef unsigned long UInt32;

UInt32 add(UInt32 a, UInt32 b)
{
    return (a + b);
}

UInt32 mul(UInt32 a, UInt32 b)
{
    return (a * b);
}

UInt32 div(UInt32 a, UInt32 b)
{
    return (a / b);
}

void main(void)
{
    volatile UInt32 result[4];

    result[0] = 43125;
    result[1] = 14567;

    result[2] = add(result[0], result[1]);
    result[1] = mul(result[0], result[2]);
    result[3] = div(result[1], result[2]);

    return;
}
Floating-point Math.c

/************************************************************************
* Name : Floating-point Math
* Purpose : Benchmark floating-point math functions.
* 
************************************************************************/

float add(float a, float b)
{
    return (a + b);
}

float mul(float a, float b)
{
    return (a * b);
}

float div(float a, float b)
{
    return (a / b);
}

void main(void)
{
    volatile float result[4];

    result[0] = 54.567;
    result[1] = 14346.67;

    result[2] = add(result[0], result[1]);
    result[1] = mul(result[0], result[2]);
    result[3] = div(result[1], result[2]);

    return;
}
FIR Filter.c

/************************************************************************
* Name : FIR Filter
* Purpose : Benchmark a FIR filter. The input values for the filter
* is an array of 51 16-bit values. The order of the filter 17.
* */
/************************************************************************/

#include <math.h>

#define FIR_LENGTH 17

const float COEFF[FIR_LENGTH] = {
    -0.000091552734,
    0.000305175781,
    0.004608154297,
    0.003356933594,
    -0.025939941406,
    -0.044006347656,
    0.063079833984,
    0.290313720703,
    0.416748046875,
    0.290313720703,
    0.063079833984,
    -0.044006347656,
    -0.025939941406,
    0.003356933594,
    0.004608154297,
    0.000305175781,
    -0.000091552734
};

/* The following array simulates input A/D converted values */

const unsigned int INPUT[] = {
    0x0400, 0x0800, 0x0C00, 0x1000, 0x1400, 0x1800, 0x1C00, 0x2000,
    0x2400, 0x2000, 0x1C00, 0x1800, 0x1400, 0x1000, 0x0C00, 0x0800,
    0x0400, 0x0400, 0x0800, 0x0C00, 0x1000, 0x1400, 0x1800, 0x1C00,
    0x2000, 0x2400, 0x2000, 0x1C00, 0x1800, 0x1400, 0x1000, 0x0C00,
    0x0800, 0x0400, 0x0400, 0x0800, 0x0C00, 0x1000, 0x1400, 0x1800,
    0x1C00, 0x2000, 0x2400, 0x2000, 0x1C00, 0x1800, 0x1400, 0x1000,
    0x0C00, 0x0800, 0x0400
};

void main(void)
{
    int i, y; /* Loop counters */
    volatile float OUTPUT[36];

    for(y = 0; y < 36; y++)
    {
        for(i = 0; i < FIR_LENGTH/2; i++)
        {
            OUTPUT[y] = COEFF[i] * ( INPUT[y + 16 - i] + INPUT[y + i] );
        }
    }
}
OUTPUT[y] = OUTPUT[y] + ( INPUT[y + 16 - i] * COEFF[i] );

return;
Matrix Multiplication.c

/************************************************************************
* Name: Matrix Multiplication
* Purpose: Benchmark multiplying a 3x4 matrix by a 4x5 matrix.
* Matrix contains 16-bit values.
* ************************************************************************/

typedef unsigned short UInt16;

const UInt16 m1[3][4] = {
    {0x01, 0x02, 0x03, 0x04},
    {0x05, 0x06, 0x07, 0x08},
    {0x09, 0x0A, 0x0B, 0x0C}
};

const UInt16 m2[4][5] = {
    {0x01, 0x02, 0x03, 0x04, 0x05},
    {0x06, 0x07, 0x08, 0x09, 0x0A},
    {0x0B, 0x0C, 0x0D, 0x0E, 0x0F},
    {0x10, 0x11, 0x12, 0x13, 0x14}
};

void main(void)
{
    int m, n, p;
    volatile UInt16 m3[3][5];

    for(m = 0; m < 3; m++)
    {
        for(p = 0; p < 5; p++)
        {
            m3[m][p] = 0;
            for(n = 0; n < 4; n++)
            {
                m3[m][p] += m1[m][n] * m2[n][p];
            }
        }
    }

    return;
}
RSA_Multiplication.c

/************************************************************************
* Name : RSA Multiplication
* Purpose : Benchmark multiplying vectors.
* {}
************************************************************************/

unsigned char f1[256];
unsigned char f2[256];
unsigned char p[256];

void main(void)
{
    unsigned long i,j;
    unsigned long acc;
    unsigned long loop;

    t1 = *(unsigned int*)0xFE4; //Start

    for (i=0; i<256; i++) {
        f1[255-i] = (unsigned char)i;
        f2[i] = (unsigned char)i;
    }

    for (loop=0; loop<1000; loop++) {
        acc = (f1[0] * f2[0]);
        p[0] = (unsigned char)acc;
        acc >>= 8;
        for (j=1; j<256; j++) {
            for (i=0; i<=j; i++) {
                acc += (f1[i] * f2[j-i]);
            }
            p[j] = (unsigned char)acc;
            acc >>= 8;
        }
    }
}
REFERENCES


BIBLIOGRAPHY


IEARobotics. *Desarrollo de aplicaciones multiplataforma Linux/Windows, para consola, con Cygwin.*

The Embedded Microprocessor Benchmark Consortium (EEMBC)
http://www.eembc.com [Date of access: 12/3/2005]

EEMBC Certification Laboratory
http://www.ebenchmarks.com [Date of access: 12/3/2005]

SPEC - Standard Performance Evaluation Corporation
http://www.specbench.org [Date of access: 12/3/2005]

Linux man pages online
http://man.he.net [Date of access: 17/6/2005]

Alphabetical Directory of Linux Commands
http://www.oreillynet.com/linux/cmd [Date of access: 22/6/2005]


http://iteso.mx/~jluis/sd/diapositivas/02-Sockets.pdf [Date of access: 10/6/2006]

http://focus.ti.com/lit/an/slaa205a/slaa205a.pdf [Date of access: 29/2/2006]
# FIGURE LIST

## INTRODUCTION

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Most embedded systems are real-time and most real-time systems are embedded.</td>
</tr>
</tbody>
</table>

## CHAPTER 1. BACKGROUND

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>RCM3720 Ethernet Connection Kit.</td>
</tr>
<tr>
<td>3.</td>
<td>Rabbit Development kit (Dynamic C).</td>
</tr>
<tr>
<td>4.</td>
<td>SNAP board with IM1000.</td>
</tr>
<tr>
<td>5.</td>
<td>Imsys Developer.</td>
</tr>
</tbody>
</table>

## CHAPTER 2. BOARD FEATURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.</td>
<td>Procedure to calculate the execution time.</td>
</tr>
<tr>
<td>8.</td>
<td>Diagram of the procedure of the benchmark.</td>
</tr>
<tr>
<td>9.</td>
<td>TCP packet.</td>
</tr>
</tbody>
</table>

## CHAPTER 3. BENCHMARKS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.</td>
<td>Orderly boards according to the execution time.</td>
</tr>
<tr>
<td>11.</td>
<td>Proportion of time that each board spends to execute an application.</td>
</tr>
<tr>
<td>12.</td>
<td>Whetstone results.</td>
</tr>
<tr>
<td>14.</td>
<td>Results of the execution of the client-server between two PCs.</td>
</tr>
<tr>
<td>15.</td>
<td>TCP rates uplink and downlink.</td>
</tr>
</tbody>
</table>

## CHAPTER 4. RESULTS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.</td>
<td>CerfBoard 270 Linux of Intrinsyc.</td>
</tr>
<tr>
<td>18.</td>
<td>HyperTerminal, terminal emulation program of Windows.</td>
</tr>
<tr>
<td>20.</td>
<td>Shell Cygwin.</td>
</tr>
</tbody>
</table>

## CONCLUSIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## APPENDIX A: OTHER STUDIED BOARDS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.</td>
<td>CerfBoard 270 Linux of Intrinsyc.</td>
</tr>
<tr>
<td>24.</td>
<td>HyperTerminal, terminal emulation program of Windows.</td>
</tr>
<tr>
<td>25.</td>
<td>Bitfire board by Arrow engineering.</td>
</tr>
<tr>
<td>26.</td>
<td>Shell Cygwin.</td>
</tr>
<tr>
<td>27.</td>
<td>Philips Flash Utility.</td>
</tr>
</tbody>
</table>

## APPENDIX B: MEASURING EXECUTION TIME IN IMSYS DEVELOPER

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## APPENDIX C: ENCOUNTERED PROBLEMS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## APPENDIX D: DHRYSTONE’S CHARACTERISTICS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## APPENDIX E: SET OF SIMPLE APPLICATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## REFERENCES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>

## BIBLIOGRAPHY

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.</td>
<td>SNAP and IM3000 memory.</td>
</tr>
</tbody>
</table>
TABLE LIST

INTRODUCTION .................................................................................................................. 9

CHAPTER 1. BACKGROUND ............................................................................................. 11

CHAPTER 2. BOARD FEATURES ...................................................................................... 13
  Table 1. RabbitCore RCM3720 main specifications ...................................................... 14
  Table 2. SNAP main specifications .............................................................................. 15
  Table 3. IM3000 main specifications ........................................................................... 16
  Table 4. Boards specific features .............................................................................. 17

CHAPTER 3. BENCHMARKS ............................................................................................ 19
  Table 5. Features of set of simple applications ............................................................. 20

CHAPTER 4. RESULTS ................................................................................................... 23
  Table 6. Features of different boards ........................................................................... 23
  Table 7. Execution time in milliseconds ...................................................................... 23
  Table 8. Whetstone results ........................................................................................... 25
  Table 9. Dhrystone 2.1 results (LOOP = 100.000, C code) ............................................ 25
  Table 10. TCP rate uplink performance ....................................................................... 27
  Table 11. TCP rate downlink performance .................................................................. 27

CONCLUSIONS ............................................................................................................... 29

APPENDIX ...................................................................................................................... 31

APPENDIX A: OTHER STUDIED BOARDS ..................................................................... 33
  Table 12. CerfBoard 270 Linux Specifications ............................................................... 34
  Table 13. Bitfire main specifications ............................................................................ 36
  Table 14. Execution time in milliseconds in Bitfire ARM .............................................. 36
  Table 15. All boards specific features ....................................................................... 37

APPENDIX B: MEASURING EXECUTION TIME IN IMSYS DEVELOPER .................... 39

APPENDIX C: ENCOUNTERED PROBLEMS ................................................................ 43

APPENDIX D: DHRYSTONE’S CHARACTERISTICS ..................................................... 45
  Table 16. Dhrystone’s strength and weakness ............................................................... 45

APPENDIX E: SET OF SIMPLE APPLICATIONS ............................................................ 49

REFERENCES ............................................................................................................... 63

BIBLIOGRAPHY ........................................................................................................... 65