University College Cork, Ireland
Computer Science Department

FINAL YEAR PROJECT
April 2015

Caching in real-time and embedded systems

Benchmarking the ARM Cortex-M3 and Quark SoC X1000 processors

Student:
Pablo Pueyo Ramon
Student ID:
114102595

Supervisor:
Dr John Vaughan
Second Reader:
Professor John Morrison

Declaration of Originality.

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:-

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way towards an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Name: <Pablo Pueyo Ramon, 114102595>

Signed: _____

Date: <29/04/2015>

Contents

1 Introduction

Nowadays, the systems that are used in the field of the informatics are more and more different to each other. One of the most important systems are embedded system due to they are inside the devices that the current people use everyday such as cars, appliances, videoconsoles, other electronic devices.

That is why these devices should work properly, without any problem, because they can affect sometimes the people's security. So, the choice of a good embedded processor or a bad one, depending on the features, seems to be an important aspect.

Therefore, it is important to create a good benchmark suite which is able to test the processor features as well as their performance in order to choice the most suitable one depending to its application.

The end of this project is to compare two commercial processors, which will be described later, through a suite of benchmark created for the project.

2 Background

2.1 General view of the families and caching systems

The processors used in this project are the following:

-Intel Quark x1000 which is included into an Intel Galileo board and,
-ARM Cortex M3, included in an Arduino DUE board.
The processors will be described deeply in the following sections:

2.1.1 Description of the processor families

2.1.1.1 Quark x1000

General View of the chip

This chip was designed by Intel. The first one was announced in September 2013 and started to be sold on the first months of 2014.

The chip was developed to support embedded operating systems, with the goal of improving the consumption of power, but reaching good speed and performance too.

It has only one core, manufactured on 32nm technology, and supports Pentium instruction set. It is capable of reaching clock frequency up to 400 MHz .

These processors are slower and smaller than Intel previous one, Atom, but it consumes less power, feature which is more and more important to achieve a good embedded system-on chip nowadays.

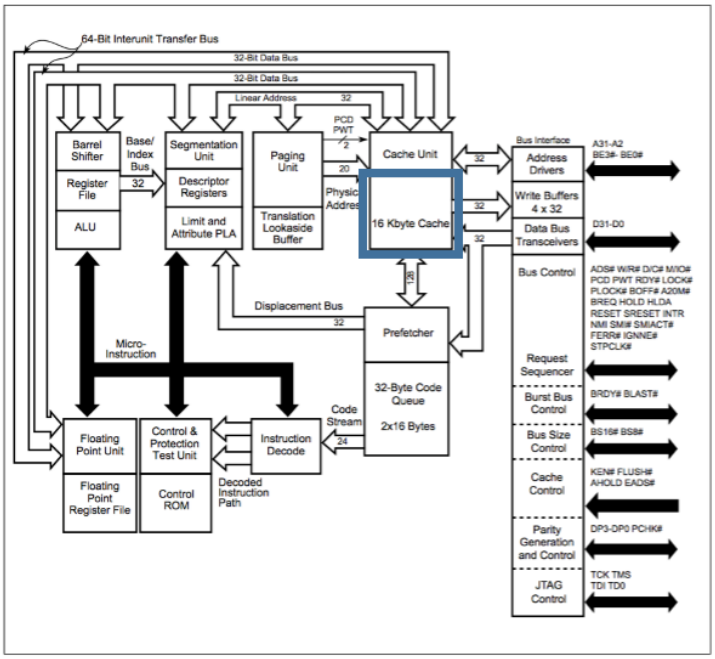

Figure 1: Overview of the Quark x1000 chip [1]

Memory System

This family of processors has a unified 16 Kb cache for instructions and data, in order to improve the performance of the whole processor, since caches are becoming one of the most important aspects of a chip to achieve better performances. In Fig.1 (in blue) it is easy to notice that two buses of data and instructions access to the same cache.

This cache is organize as a 4-way associative one. Therefore, it has 256 sets of 64 bytes each one. Each line of the cache, named blocks, has 16 bytes, 4 words. Thereby each set is comprised of 4 blocks.

That is to say that the whole cache is capable to store up to 1024 blocks.

This cache can either work with write-back mode or with write-through one, which can be set by the programmer or designer of the application that runs in the processor, depending of the well-known advantages and disadvantages of using each mode of writing from cache to main memory.

Write-back cache coherency

To preserve the write-back cache coherency of block, this processor uses a MESI protocol, which is founded on 4 states of each block, in order to know if a block can be used, if it should write on main memory if it is replaced, or it is necessary to access main memory to read the block. The four states are M (modified - can be read or written), E (exclusive - this cache is the only owner of this modified block), S (shared - One or more caches have this clean block), I (Invalid - The block cannot be used because do not respect the coherency with main memory)

2.1.1.2 ARM Cortex

General View of the chip

ARM is a 32 bits RISC (Reduced instruction set computing) instruction set architecture.

Its simplicity and reduced power consumption mean that the ARM is found in approximately 90% of the microcontrollers and microprocessors of the most of mobile and embedded systems, as smartphones, calculators, some computer devices, PDAs and videogames consoler processors.

Big companies implement their chips with ARM architecture, such as Nvidia, Samsung, Qualcomm, Apple and Atmel among others.

It is difficult to talk about the features of this processor, because there are a lot of subfamilies of ARM processor, each one with different features and properties.
The subfamily that is going to be described is Cortex Family.
Inside the Cortex subfamily, we can find up to 4 Cortex subfamilies. Those are, from the oldest to the newest, Cortex-M, Cortex-R, Cortex-A, Cortex A-50.

Memory System

The memory system depends of each processor again. There are Cortex processors with a high cache memory, separated into instruction and data ones, and Cortex processors without cache memory.
Therefore, it is better to describe one by one each family of processors in the

field of cache memory, even though the family with which we will work will be the Cortex M3.

Cortex-M

Inside of this family, we can find various architectures. They are intended for microcontroller use. The first is ARMv6-M. Cortex M1 and M0 belong to this family. This architecture does not have cache, what can affect chip performance.

After that, the next architecture inside this family is ARM v7-M. M3 and 4 processors do not have cache again, but the most recent processor in this family, Cortex M7, which was announced in 2014, may have cache. It meant a big improvement in the performance of its previous model, M4, with new features like the possibility of having cache. It may have one instruction cache from 0 to 64 Kb and one for data with the same size.

Cortex-R

The next step in ARM processors is Cortex-R. This was designed to work in real-time use, so its performance is better than the M ones. These processors are based on ARMv7 architecture, and may have cache.

The first is Cortex-R4, which can have again 0-64 Kb separated caches for instructions and data.

The next ones were optional - Multiple core processors. We can find 4–64 Kb separate caches for instructions and data again in Cortex R5 and Cortex R7 processors.

Cortex-A

The most known and used family of Cortex processors. They are designed to be used in an application level, so we can find Cortex-A processors in smartphones, videogames consoles, and others embedded systems.

Inside this family, there are a lot of models, which was released in different year and which have different features.

For example, there are 64 bit processors inside this subgroup

Every processor has cache in this subfamily, in order to improve this speed and performance. The cache of each one, is, by release date:

Cortex A5: L1 cache, one for data and one for instructions separately, each one with 4-64 Kb.

Cortex A7: Multiple Core processor. Each core has and 8-64 Kb different caches for instructions and data. It may have another L2 cache up to 1 Mb, shared by the two/four cores if they exist.

Cortex A8: Superscalar processor with 4-way set associative L1 cache of 16-32 Kib separated and 8 way set-associative L2 cache with 64k to 2Mb capacity.

Cortex A9: Multiple core processor. Separate data and instructions 4-way set-associative cache with configurable capacity as 16, 32 or 64 Kb. L2 128 Kb-8 Mb cache shared between every cpu.

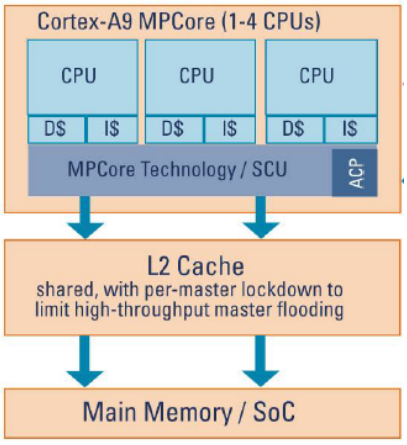

Figure 2: Cortex A9 memory schema[3]

Cortex A12: Another multiple core processor. Separated L1 caches I/D of 32-64 Kb each one. Optional L2 256Kb-8Mb cache to every cores.

Cortex A15: Multiple Core processor. 32 Kb cache of instructions and 32 for data per core. Another level, L2, up to 4 Mb per cluster.

Cortex A17: Last released processor. Multiple core. L1 separated 4 way-associative cache of 32-64 Kb for data and 32 Kb for instructions. Another

cache of L2 level, designed to work with every cores, 16 way-set associative, with variable capacity. Since 256 Kb to 8 Mb.

To sumarise the cache features of every Cortex processors, a general table is presented in figure 3.

	Processor	Cache L1(instruction/data)	Cache L2
Cortex M	Cortex M0	No cache	No L2 cache
	Cortex M1	No cache	No L2 cache
	Cortex M3	No cache	No L2 cache
	Cortex M4	No cache	No L2 cache
	Cortex M7	0-64 Kb / 0-64 Kb	No L2 cache
Cortex R	Cortex R4	0-64 Kb / 0-64 Kb	No L2 cache
	Cortex R5	0-64 Kb / 0-64 Kb	No L2 cache
	Cortex R7	0-64 Kb / 0-64 Kb	No L2 cache
Cortex A	Cortex A5	4-64 Kb / 4-64 Kb	No L2 cache
	Cortex A7	8-64 Kb / 8-64 Kb	0 – 1 Mb
	Cortex A8	16-32 Kb / 16-32 Kb	64 Kb – 2 Mb
	Cortex A9	16-64 Kb / 16-64 Kb	128 Kb – 8 Mb
	Cortex A12	32-64 Kb / 32-64 Kb	256Kb - 8Mb
	Cortex A15	32 Kb / 32 Kb	4 Mb
	Cortex A17	32 Kb / 32-64 Kb	256 Kb - 8 Mb

Figure 3: Table resume of memory system of ARM Cortex

2.2 Little changes in the features of the caches to determine the influence on the final performance

After investigating about the processors, some test about cache performance were done in CACTI. The test is important to know how can the variation of the cache parameters can affect to the power consumption and performance. The cache in which the tests will be done is the Arm Cortex R4 cache. This processor, can have cache, but it may not have one as well. The test will be run in the cached one, in the L1 cache.

This cache is separated in Data and Instruction cache. Its capacity can vary from 4 to 64 Kb and it is a 4-way associative with a line length of 8 words.

However, the test will be run modifying this parameters, to prove how the features can affect to the cache performance.

Every test will be done in the CACTI tool. After doing the test, the results will be arranged in an array and presented as graphics.

The basic parameters will be set as specified in figure 4.

	40 nm G
Advantage Library	HS 12T
Target frequency	934 MHz
Standard cell area	0.48 mm2
Core power	0.08 mW/MHz

Figure 4: Basic parameters set in CACTI

2.2.1 Cache size

As the size of this cache may vary from 4 to 64 Kb, the first test will be done about this parameter. The others parameters that are requested in CACTI, will have been set with the values shown in figure 5. (The results of the test has been rounded to 3 decimals)

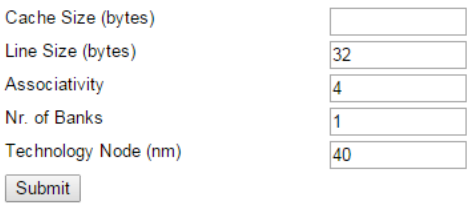

Figure 5: Parameters set in this section

The results of the test are shown in figure 6.

Cache size (KBytes)	Cache size (Bytes)	Access Time (ns)	Random cycle time (ns):	Total read dynamic energy per read port(nJ):	Total read dynamic power per read port at max freq (W):	Total area (mm^2):
4	**4096**	0,498	0,589	0,022	0,036	0,097
8	**8192**	0,498	0,589	0,022	0,037	0,143
16	**16384**	0,535	0,600	0,030	0,051	0,266
32	**32768**	0,595	0,609	0,032	0,052	0,318
64	**65536**	0,621	0,319	0,047	0,147	0,322

Figure 6: Table of results modifying the cache size

2.2.1.1 Plots with the results

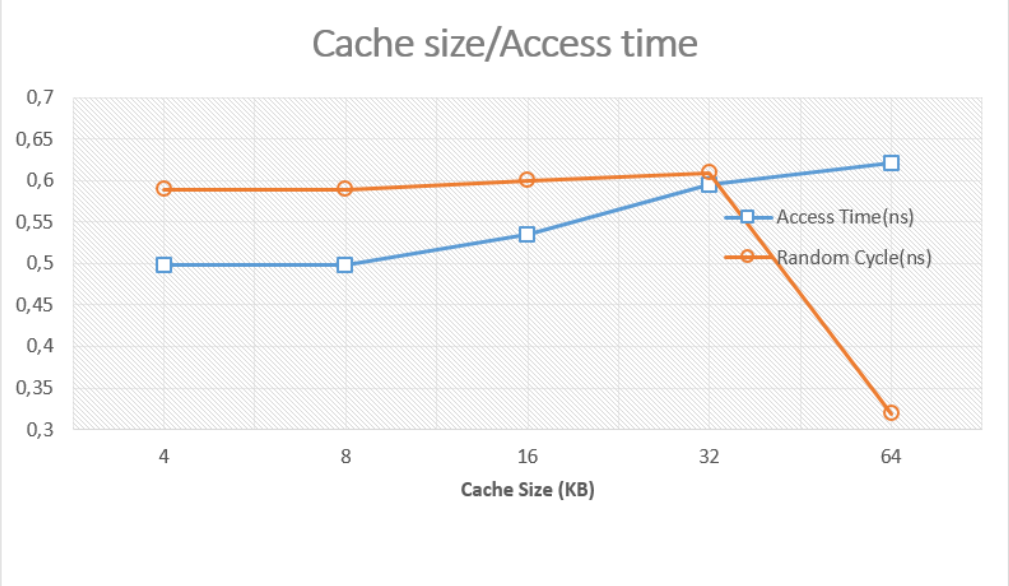

Figure 7: Plot Cache size/Access time

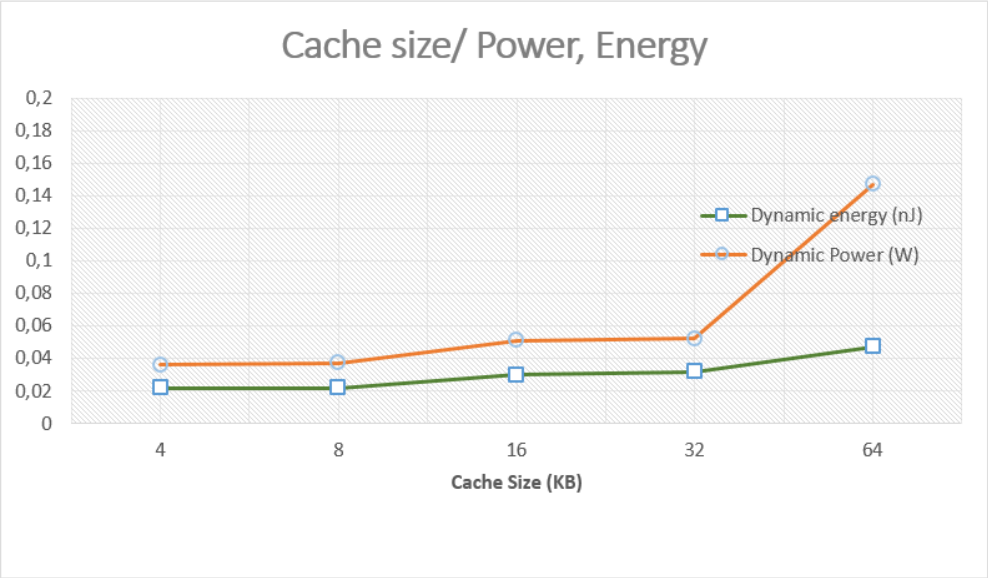

Figure 8: Plot Cache size/Power

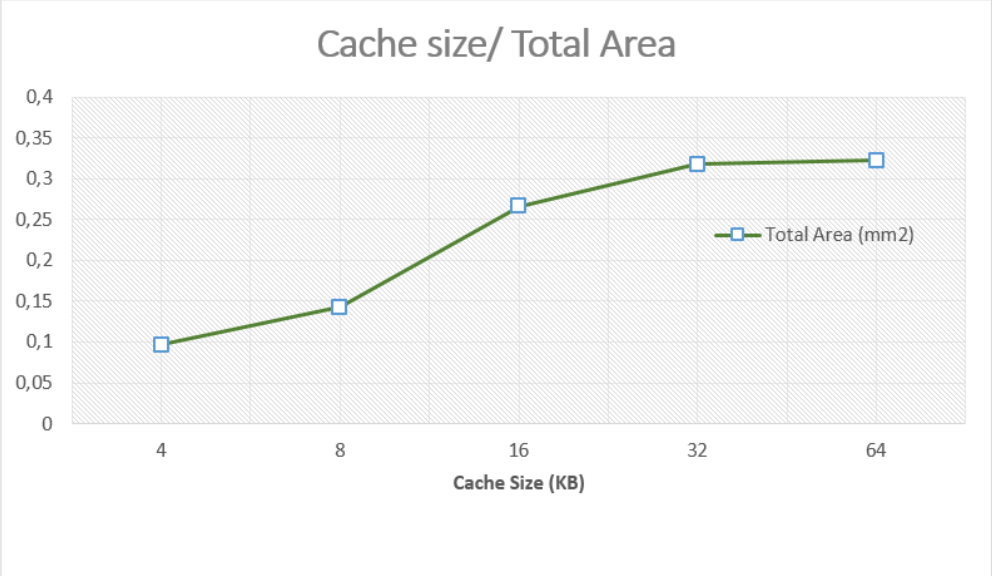

Figure 9: Plot Cache size/Power

It is easy to appreciate in the graphic found in Fig.7 that, the bigger the cache is, the more time it is needed to access to it. However, the quantity of

misses are incremented with a lower size of cache. That is why cache designers are always trying to find the best size of caches with which cache could achieve good access time and low miss rate too.

In the case of Random cycle time, there is a small difference between the four first cases. The most striking change is produced in the 64 KB cache. There is an important decrease in the Random Time Cycle with this cache, probably because the miss rate decreases significantly too.

According to ARM manual[4], this processor can run with a performance of above 934 MHz. That means that its cycle time is less than 1.07 ns.

About Fig.8, it can be noticed again that the consumption of energy/power rises when cache capacity rises.

An important fact should be highlighted in this graphic. That is the relationship between power and energy. Energy does not rise as fast as Power does. The value of the power is multiplied when the cache size is 64 KB. It happens because the frequency of the clock. The power is calculated as the Energy multiplied by the clock frequency. As the clock frequency rises a lot (in the previous graphic it is found access time, which is inversely proportional to clock frequency) when the cache size is 64 KB, the power rises too.

$$Power = Clock\ Frequency \times Energy$$

As can be seen in the plot found in Fig.9, the value of total area rises again when cache size does. There is no a hard difference between 32 and 64 KB cache, how it happened before, but know, the area of a 32 KB cache is tripled from the 4 KB. More cache size needs more physical space. That is for why the area increases too.

Following the data from Fig.4 every caches of the graphic could be inside of the chip, which area is $0.48mm^2$ whereas that the biggest cache has an area of $0.322mm^2$.

2.2.2 Line size

Now, the cache line size will be modified, in order to test how this can affect the whole cache performance. The rest of parameters will be fixed as Fig.10 shows.

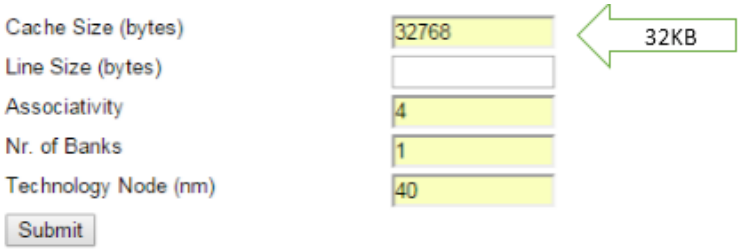

Figure 10: Parameters set in this section

The results are shown in Fig.11.

Line size (words)	Line size (Bytes)	Number of sets	Access Time (ns)	Random cycle time (ns):	Total read dynamic energy per read port(nJ):	Total read dynamic power per read port at max freq (W):	Total area (mm^2):
4	**16**	512	0,560	0,217	0,023	0,105	0,155
8	**32**	256	0,575	0,609	0,032	0,053	0,318
16	**64**	128	0,775	0,282	0,125	0,442	0,564
32	**128**	64	1,176	0,269	0,478	1,780	1,862
64	**256**	32	1,198	0,269	1,894	7,052	6,785

Figure 11: Table of results modifying the associativity

2.2.2.1 Plots with the results

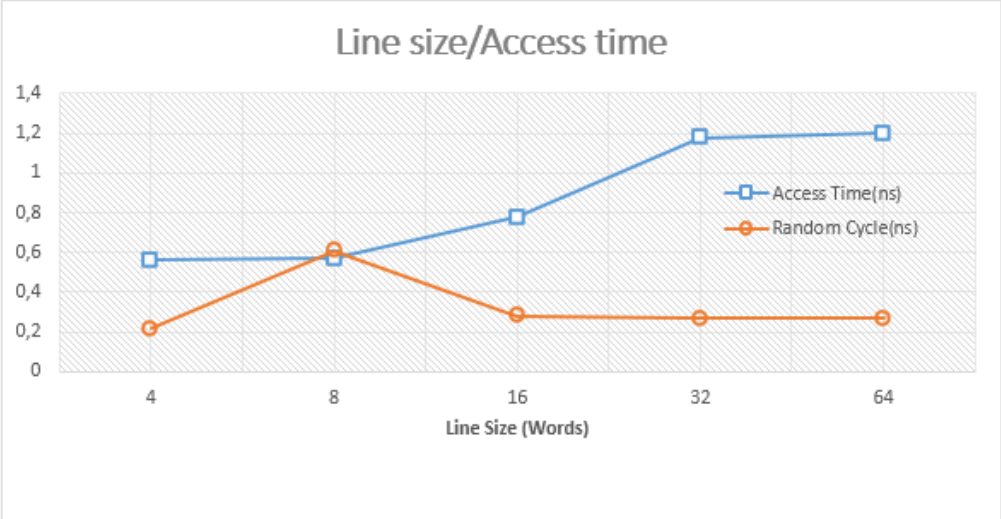

Figure 12: Plot Line size/Access time

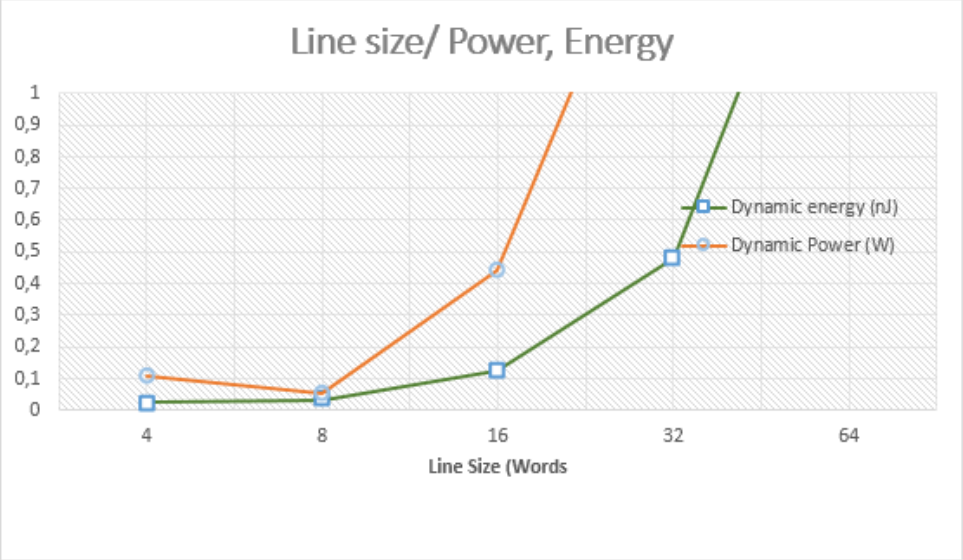

Figure 13: Plot Line size/Power

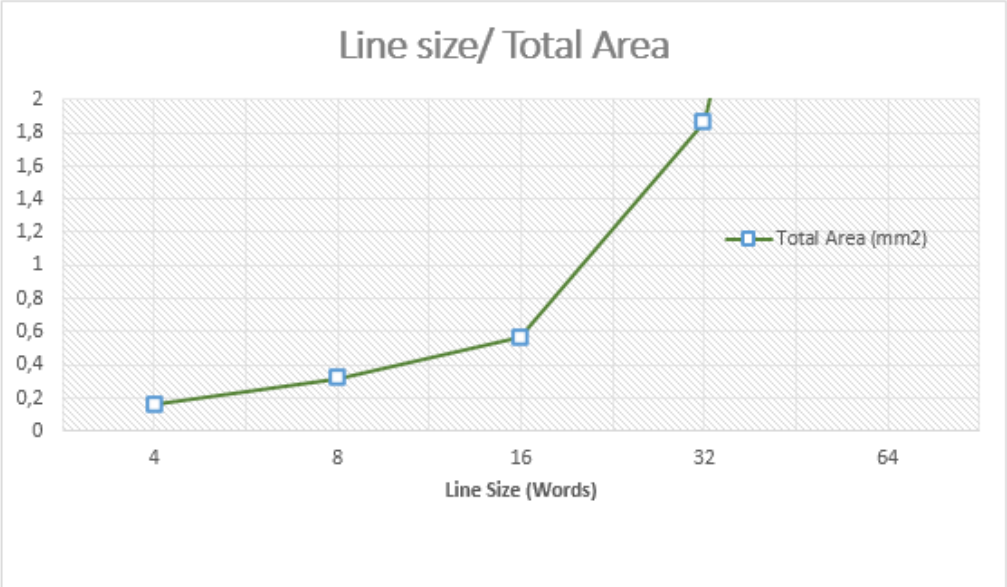

Figure 14: Plot Line size/Area

Now in the plot in Fig.12, the access time rises when Line size does. It occurs because the bigger the line size is, the more information should be recovered in a cache access, so the time is higher.

It is worth highlighting the striking change in the random cycle time when the line size is 8 words. It can be due to the relationship between the Cache size and the Line size.

Again, in Fig.13, it can be noticed that the consumption of power is higher when the line size gears up.

When the line size is 8 words, as it can be seen in the previous graphic, the clock frequency is low, so the power would be lower too.

When line size is more than 16 words, the power consumption is unfeasible, so line size cannot be higher than 16 words in this cache due to the power limitation, one of the most important things in all current processors.

As it can be noticed in the Fig.14, the value of total area rises again when line size does. Given the reference values, can be found in Fig.5, the whole processor should have an area of $0.488mm^2$ so, assuming that, the only line

sizes can be set in this cache should be less than 16 words.

It is totally impossible for current processors to have line size of 64 words, with a approximate area of $7mm^2$.

2.2.3 Associativity

It is shown in Fig.15 how the rest of parameters will be set:

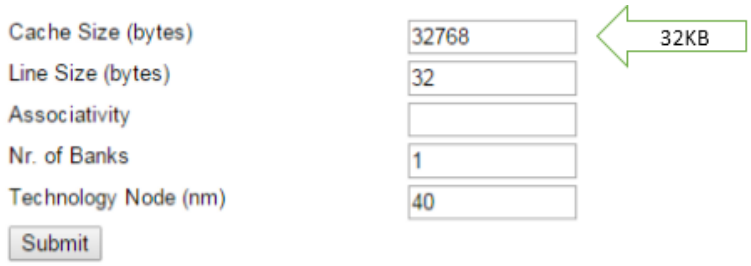

Figure 15: Parameters set in this section

The results are shown in Fig.16.

Associativity	Number of sets	Access Time (ns)	Random cycle time (ns):	Total read dynamic energy per read port(nJ):	Total read dynamic power per read port at max freq (W):	Total area (mm^2):
1	1024	0,394	0,273	0,035	0,129	0,277
2	512	0,544	0,217	0,031	0,141	0,181
4	256	0,575	0,609	0,032	0,053	0,318
8	128	0,774	0,588	0,066	0,112	0,463
16	64	1,118	0,588	0,208	0,354	0,782

Figure 16: Table of results modifying the associativity

2.2.3.1 Plots with the results

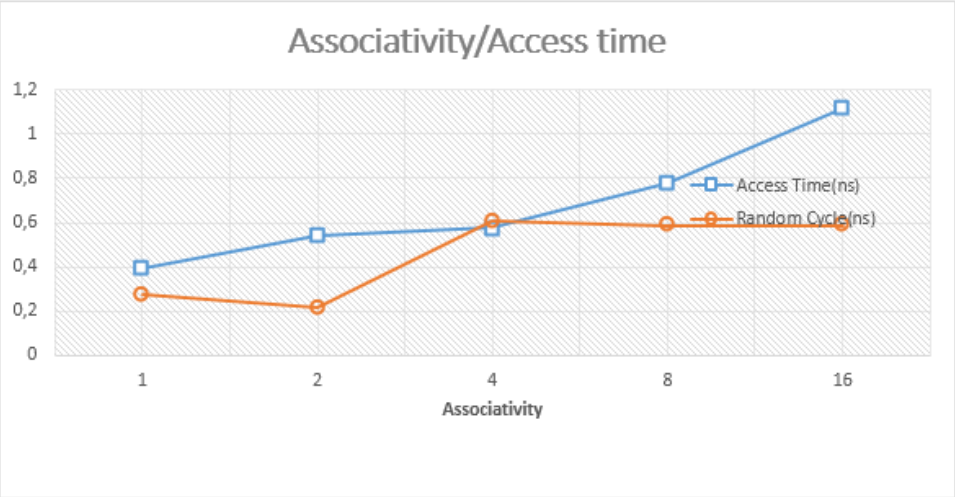

Figure 17: Plot Associativity/Access time

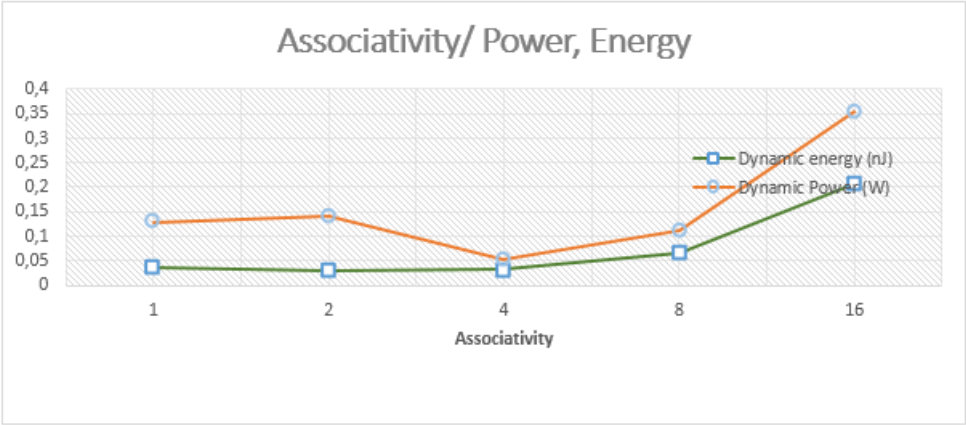

Figure 18: Plot Associativity/Power

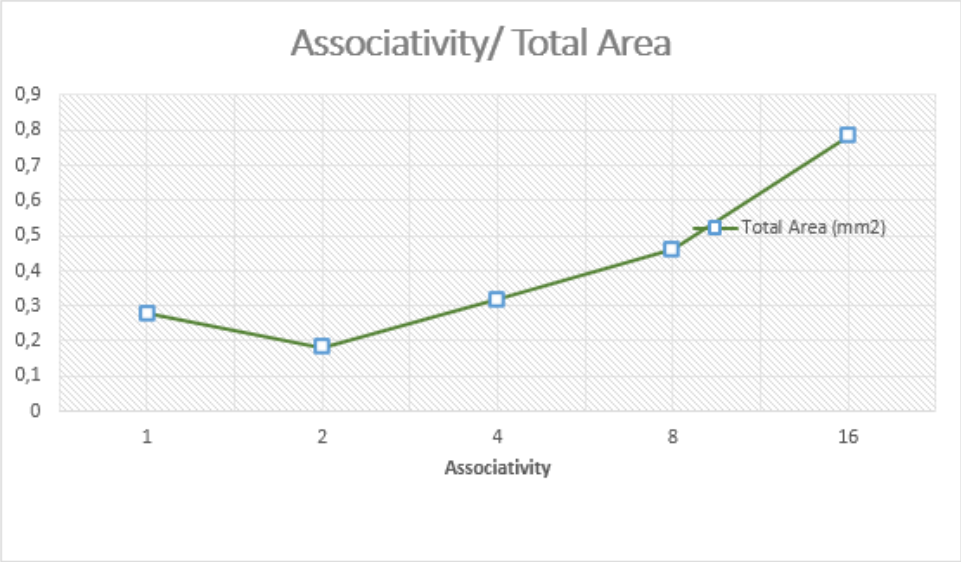

Figure 19: Plot Associativity/Area

Once again, in the Fig.17, everything rise when associativity rises. Those results are not really realistic, because increasing the associativity, a better performance of caches is achieved even though the access time is higher.

Again, it can be noticed in Fig.18 that the power is proportional to the clock frequency.

The energy is higher when the associativity rises. This test shows a less abrupt difference in the power than in the previous one.

Seeing the Fig.19, the value of total area rises again when line size does. An associativity with value of 16, will not be accepted to this cache, because the area is higher than 0.48 mm2. The others values can be accepted. Now, it is understandable that the area is proportional to the associativity because the higher the associativity is, the more logic and physical hardware is needed to maintain the coherence between sets.

2.3 Systematic literature review about caching

Caching is becoming one of the most important features in the most recent and updated processor devices such as real-time and embedded systems There are many different approaches of how caching can affect to the processor performance and energy consumption.

While the performance is usually determined by the behavior of the application which is running, depending on the use of the spatial locality for example, cache energy consumption can affect up to 50% of the final processor energy use. We can find embedded systems everywhere -cars, small devices and appliances amog others-, they are becoming an important aspect in recent computing studies. As power consumption is more and more important in this field, because the increasingly dependence of batteries, there are a large number of papers and reviews which approach that subject, through loads of studies and experiments, trying to find the equilibrium between cache power consumption and performance.

That is for why, this Systematic Literature Review will be made, to look for the best papers which are describing the cache model of a processor. The text will be found on the internet doing some searches on a searching engine such as google, doing queries related to the subject.

2.3.1 Questions for the analysis

As a systematic literature review, a few question should be made in order to facilitate the searching of relevant documents. That documents will be chosen following the answer of the questions below, looking for this reports which can achieve the answer "yes" in the three proposed questions. Those documents will be the most important ones between all the selected at first in relation with the subject of this report "Caching in real-time and embedded systems". In the first table, the questions are presented, followed by their possible answers. In the second table, all the documents found are placed, with its adequate answer for questions.

Question	*Answer*
Q1. Does the text speak about the caching model in embedded/real-time systems?	Yes/No
Q2. Are some kind of experiments or comparisons about caching in real-time embedded systems presented?	Yes/No
Q3. Is any new method or improvement of the cache presented in the document?	Yes/No

2.3.2 Answers to the questions applied to the documents

No	Author	Year	Topic Type	Q1	Q2	Q3
[5]	A. Marti Campoy, E. Tamura, S. Saez, F. Rodriguez, and J. V. Busquets-Mataix	2005	On Using Locking Caches in Embedded Real-Time Systems	Y	Y	Y
[6]	Paul Markin	2003	A Case-Study On The Impact Of L2 Cache Size On CPU Performance	N	Y	Y
[7]	Rogue Wave Software	2011	CPU Cache Optimization: Does It Matter? Should I Worry? Why?	N	Y	Y
[8]	Dan Nicolaescu, Alex Veidenbaum, Alex Nicolaux	-	Reducing Power Consumption for High-Associativity Data Caches in Embedded Processors	Y	Y	Y
[9]	Andhi Janapsatya, Aleksandar Ignjatovic, Sri Parameswaran	2006	Finding Optimal L1 Cache Configuration for Embedded Systems	Y	Y	Y
[10]	Yujia Jin and Rong Chen	-	Instruction Cache Compression for Embedded Systems	Y	Y	Y
[11]	Bruce Jacob	1999	Cache Design for Embedded Real-Time Systems	Y	Y	Y
[12]	Norman P. Jouppi	1991	Cache Write Policies and Performance	Y	Y	Y
[13]	Bruce Jacob	1998	Software-Managed Caches: Architectural Support for Real-Time Embedded Systems	Y	N	N
[14]	Bach D. Bui, Marco Caccamo, Lui Sha, Joseph Martinez	2007	Impact of Cache Partitioning on Multi-Tasking Real Time Embedded Systems	Y	Y	Y

No	Author	Year	Topic Type	Q1	Q2	Q3
[15]	Prabhat Jain	2008	Software-assisted Cache Mechanisms for Embedded Systems	Y	Y	Y
[16]	Emre Ozer, Resit Sendag and David Greggrr	2005	Multiple-Valued Caches for Power-Efficient Embedded Systems	Y	Y	Y
[17]	Abu Asaduzzaman	2009	Cache optimization for real-time embedded systems	Y	Y	Y
[18]	Abu Asaduzzaman	-	Block Cache for Embedded Systems	Y	Y	Y
[21]	Frank Vahid, Walid Najjar and Chuanjun Zhang	2005	A Highly Configurable Cache for Low Energy Embedded Systems	Y	Y	Y

2.3.3 Discussion

In this section, the previous research questions will be answered, regarding the selected papers.

RQ1. How important is the caching model to the processor performance?

All text are speaking about this subject. We can start describing how the cache model can affect to the processor energy consumption, as we can see the importance of the caches energy consumption in many papers.

Cache memories account for about 50% of the total energy consumed in the system, as we can read in [21]. Moreover, varying some cache parameters, the use of energy can be varied too. As [8] says, the difference between set-associative and non set-associative caches could be a critic decision if the designer wants to save energy, for example, for embedded systems, where energy consumption is one of the most important features.

That is the same case in processor performance and speed. "Cache memories are crucial to obtain high performance on contemporary processors." says [5]. And it is right. For example, following [6], doubling the L2 cache in the processor it talks about, can suppose an improvement of the computational performance of 4-5%, an important improvement in terms of processor performance. However, some texts says that the performance of the cache system is highly determined by the program behavior.[21]

Therefore, cache memories is a thing to take into account when a new processor is designed.

To finish with this section, it is important to remark that cache parameters such as associativity, the total cache size, and the cache line size, among others, can affect the hit rate of the cache. The hit and miss rates are the most determinant parameters in terms of cache performance and energy consumption.[21]

RQ2. Which are the improvements that can be done to the current caches?

Some of the texts try to find the optimum set of cache parameters to improve the cache yield. Some of them will be presented in this answer section. In [21] it is presented a highly configurable cache, in order to modify dynamically the most important parameters such as cache line size, cache size, among others, which affect significantly to the whole processor, as it was presented in the previous answer.
In other texts, the parameters are varied too to show the improvement of cache results doing that. [6] talks about cache size, [8] about associativity, in [9], they try to find the best L1 cache configuration varying those parameters, among others.

Block cache for embedded systems are described in [18]. Using a dynamically instruction blocks can reduce the area, power use and performance of the caches, according to the authors.
The authors of [10] speak about what they call "Instruction Cache Compression".

In [7], some experiments are made in order to show how can the code affect to the cache performance. Locking caches, which consists in locking the cache in order to avoid unnecessary block replacements, which can be configured in execution time to lock the blocks are presented in [5].
To complete this section, a new kind of software management is introduced in [13]. we can avoid to save not important software code as initialization code in caches with the end of saving space to necessary data.

RQ3. How can we validate the effectiveness of the proposed techniques?

There are a huge number of experiments and results presented in each report. Some of them are going to be presented now as examples, but all reports have data which they can support their new techniques with.

For instance, in [21], [6], [8] or [9] among others, loads of graphics, numeric results and experiments were done in order to support how dynamically varying of parameters affect positively cache performance.
In [5] they prove the excellent results of their "locking caches" showing a lot of graphic and numeric data again.

Or for example, the visual graphic and experiments presented in [18] to show the reliability to Block caches for embedded systems.

3 Building the benchmark suite

Once the investigation about embedded systems and caching there have been done, it is important to focus the work on the benckmarking suite of this project.
First of all, the description of tools that are disposed:

3.1 Boards

The two boards, and their features that will be used to the experiments are the following:

ARDUINO DUE [25]
-Processor: ARM Cortex-M3 32 bits
-CPU clock: 84 Mhz
-Flash memory for code: 512 KBytes
-SRAM memory: 96 KB: Two banks: 64 KB and 32 KB

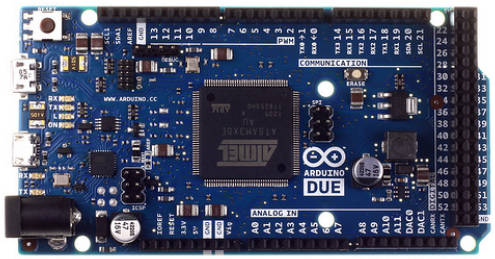

Figure 20: Arduino DUE board front

INTEL GALILEO [26]
-Processor: Intel Quark x1000, 32 bits.
-CPU clock: Up to 400 Mhz
-SRAM memory: 512 KB
-Cache memory: 16 KByte L1 Cache

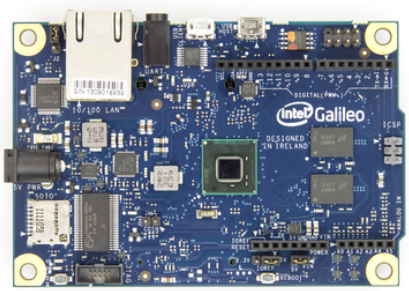

Figure 21: Intel Galileo board front

3.2 Environment and materials

For the development in the boards, basic circuit materials, such as cables, breadboard, buttons and other electronic stuff have been used.
The environment has been the Arduino for windows. It is an environment which facilitate the upload of the program to the board using its own compiler and uploader. The programs are written in C and upload to the board trough an USB cable. It is the basic program for the Arduino board. However, an emulator of Arduino is used in the Galileo, what should be taken into account with the numeric results, thus it can reduce its performance.

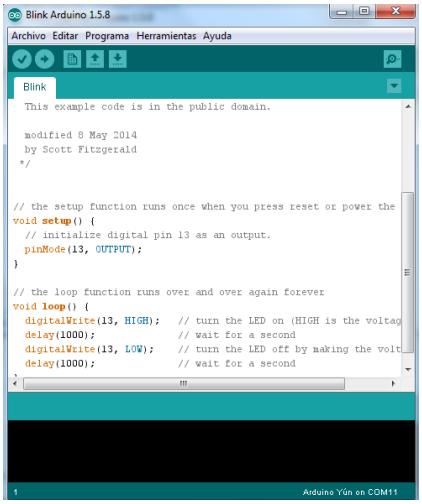

Figure 22: Arduino program

3.3 Oscilloscope and wave generator

For taking times, two devices were used. They are the function (wave) generator, which generates waves whose shape can be chosen among square, sinusoidal... This device generates the waves with a determinate frequency. It helps to take time, because these waves can cause an interruption or change of pins into the board.
On the other hand we have the oscilloscope, "a screen" that shows the changes produced in a pin or an electrical signal. With both devices, it is possible to take times. For example, with the wave generator we can create an interruption each time the wave goes up, and make the board to process this interruption, showing a wave in the oscilloscope. The time gap between the input and the output, is the time that the board needs to process a simple interruption.

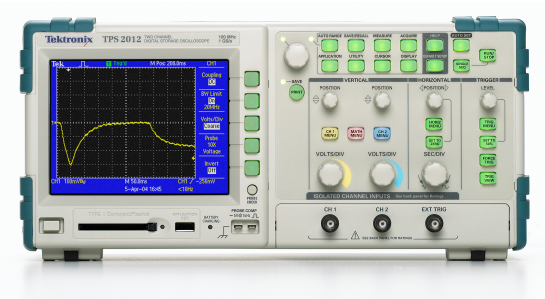

Figure 23: Oscilloscope simple. Source: [28]

Figure 24: Wave generator. Source: [29]

3.4 Looking for other benchmark programs

After doing the first approaches with the board, it is worth to look for information about how to do a benchmark in a real-time/embedded system and specifically, in Arduino.

There are a lot of codes and ways to benchmark the boards. The examples seen follow the same structure. They do some hard mathematical operations, e.g. calculate the value of e using an infinite series, matrix multiplication, or numerical series, among others.

The most important thing to demonstrate a processor performance is taking times.

Before doing these mathematical operations, it is needed to take the time. The time is taken again when the operation is concluded and the overall operation time is calculated doing a deduction of the times taken. After that, the operation is done in another board to make a comparison between them, looking at the execution times.

The kind of the operation affects to the time. It is different to execute a floating point operation and an integer operation. A board can execute an integer operation faster than another, but it can execute floating point operations slower than other board.

3.5 How the benchmark programs for real-time systems should be?

Once the reading of these documents, the problem to create a benchmarking set of programs to a real-time/embedded systems is facing. The difference between a real-time and a traditional system are that the real-time should be able to respond to external and internal events in a fixed period of time[27], being the response time the most critical feature to them.

The Input/Output services, carried out by the sensors and actuators are also a thing to take into account when doing benchmarking. The time of response of that, the interrupt processing or the time to treat the interruption are also added to the list.

The interruptions and inputs can have a order of priorities. For example, inside a car, supposing that an accident happens, it is more important to open the airbag than the radio continue playing. So the airbag must be more priority than the radio player, and it can use the processor overlapping other running tasks.

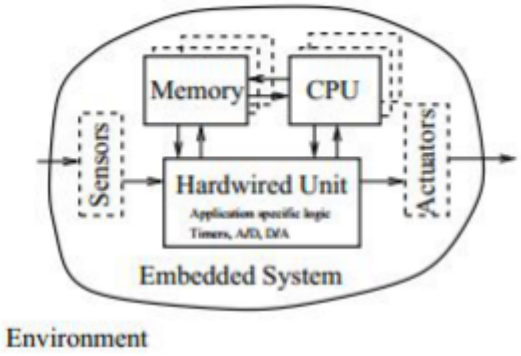

Figure 25: Essential part of an embedded system. Source: [27]

Therefore, what I want to measure is meanly the time.
But, what tasks should I take the time for?
Summarizing, The kinds of process which are measuring are those that are focused on, for example:
-Multitasking
-I/O
-Memory management
-Interrupt processing
-Scheduling latency
-Interprocess communications

4 The benchmark suite

Thus the major features of embedded systems wanted to be tested, the suite of benchmarking programs of this project is divided in sections, depending on the feature of the system that it tests. The sections are the followings:

4.1 Performance in mathematical calculations

The speed calculating and working with floating point numbers is remarkable in the effectiveness of an embedded system. For that, the program Pi was created:

4.1.1 Pi

This program calculates the value of Pi using the Newton Approximation, though a big number of iterations using a loop. The number of iterations can be chosen, the more iterations you choose, the more accurate the value of Pi you will obtain, but it will take more time.
The time taken in this program is that it uses since the first iteration to the value of Pi is calculated.

```
//Pablo Pueyo

// Program which calculates the approximation of the pi
// number using the Newton's approximation.
// Pi is more accurate doing more iterations, but it
// makes the processor to decrease the speed and to take
// more time to execute the code.

#define ITERATIONS 200000      // number of iterations
#define FLASH 1000             // blink LED every 1000
                               // iterations


void setup() {
 pinMode(13, OUTPUT);
 Serial.begin(9600);
}

void loop() {

 unsigned long start, time;
```

```
 unsigned long niter=ITERATIONS;
 unsigned long i, count=0;
 double x = 1.0;
 double temp, pi=1.0;

 start = millis();

 Serial.print("Beginning ");
 Serial.print(niter);
 Serial.println(" iterations...");
 Serial.println();

 for ( i = 2; i < niter; i++) {
   x *= -1.0;
   pi += x / (2.0*(double)i-1);
 }

 time = millis() - start;

 pi = pi * 4.0;

 Serial.print("Number of iteras = ");
 Serial.println(niter);
 Serial.print("Estimate of pi = ");
 Serial.println(pi, 10);

 Serial.print("Time: "); Serial.print(time);
 Serial.println(" ms");

 delay(10000);
}
```

4.2 Performance in mathematical calculations and memory performance

The two following programs have been created to test the performance of the memory management of the system. Moreover, it does some calculations with the arrays created, so the mathematical speed is also tested in these programs.

4.2.1 Quicksort

It sorts a totally unsorted array, that is to say, an array sorted inversed, from the highest to the lowest number. The length of the array can be modified. After the creation of the array, it sorts the elements low to hight using the method called quicksort. Again, the longer the array is, the more time to complete the action it lasts. In this program, both the time of creating the array and of sorting the elements are taken.

```
// QuickSort for benchmarking
//
// Pablo Pueyo
// February 2015
//
// This program creates a new array, with length
// indicated in the constant LENGTH
// It fills it with random numbers, between 0 and
// LENGHT and utilises the method
// quicksort to sort the array.
// After that, it shows the times which the board
// has used to create and fill the table,
// and the time used to sort the elements. It also
// shows the first 50 elements of the table before
// and after the sort.
// The led is on when the program is sorting the
// elements
//
//Quicksort method obtained from the webpage
// http://www.cprogramto.com/c-program-quick-sort/
//with some modifications.

    #define LENGTH 2000     // length of the table

void setup() {
 pinMode(13, OUTPUT);
 Serial.begin(9600);
}

void loop() {
 unsigned long start, timeCreate, timeSort, show;
 int table[LENGTH];
```

```
  int i = 0;

  start = millis();
  digitalWrite(13, HIGH);
  while (i != LENGTH - 1){
    table[i] = LENGTH - i - 1;
    i++;
  }
  digitalWrite(13, LOW);
  timeCreate = millis() - start;
  if(LENGTH < 50){
        show = LENGTH;
     }
  else{
     show=50;
   }

  Serial.print("Unsorted elements: [");
  for(i=0;i<show;i++){
     Serial.print(table[i]);
     Serial.print(", ");}
  Serial.println("] ");
  digitalWrite(13, HIGH);
  start = millis();
  quicksort(table, 0, LENGTH);
  timeSort = millis() - start;
  digitalWrite(13, LOW);
  Serial.print("Sorted elements: [");

  for(i=0;i<show;i++){
    Serial.print(table[i]);
    Serial.print(", ");}
  Serial.println("]");
  Serial.print("Time to create:  ");
  Serial.println(timeCreate);
  Serial.print("Time to sort:  ");
  Serial.println(timeSort);

  delay(10000);
}
```

```
void quicksort(int x[],int first,int last){
    int pivot,j,temp,i;

     if(first<last){
          pivot=first;
          i=first;
          j=last;

          while(i<j){
               while(x[i]<=x[pivot]&&i<last)
                    i++;
               while(x[j]>x[pivot])
                    j--;
               if(i<j){
                    temp=x[i];
                     x[i]=x[j];
                     x[j]=temp;
               }
          }

          temp=x[pivot];
          x[pivot]=x[j];
          x[j]=temp;
          quicksort(x,first,j-1);
          quicksort(x,j+1,last);

    }
}
```

4.2.2 Determinant

This program creates a matrix NxN (N indicated by the user) and calculates its determinant and matrix transpose using a recursive function. The time is again taken to make comparisons.

```
 //Pablo Pueyo Ramon

//This program creates a matrix NxN (N indicated by the
//user) and calculates its determinant and matrix
//transpose using a recursive function. The time is
//again taken to make comparisons.


// constants will not change. They are used here to
// set pin numbers:
const int ledPin =  13;      // the number of the LED pin

// variables will change:

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
   Serial.begin(9600);
}

void loop() {
  int n = 9;
  double mat[10][10], tras[10][10];
  double inv[10][10];
  //digitalWrite(ledPin, LOW);
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
      mat[i][j] = (rand() % 3) +1;
    }
  }

  long start = millis();
  digitalWrite(ledPin, LOW);
  double det1 = det(n, mat);
  digitalWrite(ledPin, HIGH);
  Serial.println(det1);
```

```
  //digitalWrite(ledPin, LOW);
  for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
        tras[i][j]=mat[j][i];
  }
  //digitalWrite(ledPin, HIGH);
}
Serial.println(millis() - start);
delay(1000);
}


double det(int n, double mat[10][10])
{double d = 0;
    int c, subi, i, j, subj;
    double submat[10][10];
    if (n == 2)
    {
        return( (mat[0][0] * mat[1][1]) -
        (mat[1][0] * mat[0][1]));
    }
    else
    {
        for(c = 0; c < n; c++)
        {
            subi = 0;
            for(i = 1; i < n; i++)
            {
                subj = 0;
                for(j = 0; j < n; j++)
                {
                    if (j == c)
                    {
                        continue;
                    }
                    submat[subi][subj] = mat[i][j];
                    subj++;
                }
                subi++;
            }
        d = d + (pow(-1 ,c) * mat[0][c] *
```

```
            det(n - 1 ,submat));
        }
    }
    return d;
}
```

4.3 Performance in Input/Output

Probably the most important feature in the recent embedded processors. They should use the sensors/actuators to interact with the environment. The programs that take into account this function are:

4.3.1 IRreceiver

Program created to benchmark the i/o of the board. In it, I used the infrared receiver as well as the remote control. Pushing 3 buttons of the remote, 3 leds of the board are turned on and off. Pushing the shutdown button, the three leds are turned off. In this case, the time since the signal is received and the led are turn off/on is taken.

```
/* Author: Pablo Pueyo Ramon
 * IRremote. Using the IR remote, the led 1, 2, 3,
 * connected to the pins 4, 5, 6, turn on when the
 * 1, 2, 3 buttons of the remote are pushed. Pushing
 * the on/off button, the three leds are turned off.
 * The time is taken to know how fast is the board
 * with the I/O since it receives the Ir signal until
 * the led is on.
 */

#include <IRremote2.h>

int RECV_PIN = 18;

int aPin = 4;
int bPin = 5;
int cPin = 6;

int aOn = 0;
int bOn = 0;
int cOn = 0;

decode_results results;

IRrecv recept(RECV_PIN);

void setup()
{
```

```
  Serial.begin(9600);
  recept.enableIRIn(); // Start the receiver
  pinMode(aPin, OUTPUT);
  pinMode(bPin, OUTPUT);
  pinMode(cPin, OUTPUT);
}

void loop() {
  if (recept.decode(&results)) {
    long start = millis();
    long int decCode = results.value;
    Serial.println(decCode);
    switch (results.value) {
      case 16718055:
        Serial.println("1 pushed");
        if(aOn){
            digitalWrite(aPin, LOW);
            aOn = 0;
          }
          else{
        digitalWrite(aPin, HIGH);
        aOn = 1;
        }
        break;
      case 16724175:
        Serial.println("2 pushed");
        if(bOn){
            digitalWrite(bPin, LOW);
             bOn=0;
          }
          else{
        digitalWrite(bPin, HIGH);    // sets the LED on
        bOn=1;
        }
        break;
      case 16743045:
        Serial.println("3 pushed");
      if(cOn){
            digitalWrite(cPin, LOW); // sets the LED off
            cOn=0;
          }
```

```
            else{
          digitalWrite(cPin, HIGH);    // sets the LED on
          cOn=1;
          }
          break;
        case 16753245:
          Serial.println("Stop");
          digitalWrite(aPin, LOW);    // sets the LED off
          digitalWrite(bPin, LOW);    // sets the LED off
          digitalWrite(cPin, LOW);    // sets the LED off
          aOn = 0;
          bOn = 0;
          cOn = 0;
          break;
        default:
          Serial.println("Waiting ....");
      }
long time = millis() - start;
Serial.println(time);
      recept.resume(); // Receive the next value
    }
}
```

4.3.2 Interruption

With the help of an oscilloscope, which will take the times of the board output, and a function generator, which will create square waves to generate interruptions in the board.
The pin 2 is attached to an interruption routine which will change a variable called "state" each time the square generated by the wave generator goes up. When it happens, the output wave, attached to the pin 13, will change its value (1 - 0) in order to take the time of processing the interruption. With the oscilloscope, we can measure the delay between the time the function generator generates the wave and the time that the board processes it.

```
 //Pablo Pueyo Ramon

int ledPin = 13; // LED is attached to digital pin 13

 int state=0;

void setup() {
  Serial.begin(9600);
  pinMode (13, OUTPUT);
  pinMode (2, INPUT);
  attachInterrupt(2, increment,    CHANGE);
}

void loop() {
  if(!state) {
  digitalWrite(13, HIGH);
  }
  else{
    digitalWrite(13, LOW);
  }
}

// Interrupt service routine for pin A5
void increment() {
  if(state) {
  state=0;
  }
  else{;
  state=1;}
}
```

4.3.3 Potentiometer

With the potentiometer, used as an input device, three leds are turned on or off depending on the position of the potentiometer. As I said, I do not know how to take times of these programs, either because the time between interruptions is close to 0 or because I do not know which time to be taken.

```
int ledPin = 8;       // LED connected to digital pin 8
int led1Pin = 9;       // LED connected to digital pin 9
int led2Pin = 10;      // LED connected to digital pin 10
int analogPin = 3;  // potentiometer connected to pin 3
int val = 0;          // variable to store the read value

void setup()
{
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);    // sets the pin as output
    pinMode(led1Pin, OUTPUT);   // sets the pin as output
    pinMode(led2Pin, OUTPUT);   // sets the pin as output
    }

    void loop()
    {
      val = analogRead(analogPin); // read the input pin
    if(val <400){
      digitalWrite(ledPin, LOW);
      digitalWrite(led1Pin, HIGH);
      digitalWrite(led2Pin, HIGH);
    }
    else if (val>=400 && val<=800){
        digitalWrite(led1Pin, LOW);
      digitalWrite(ledPin, HIGH);
      digitalWrite(led2Pin, HIGH);
    }

    else if(val>800){
      digitalWrite(led2Pin, LOW);
      digitalWrite(led1Pin, HIGH);
      digitalWrite(ledPin, HIGH);
```

```
    }
    Serial.println(val);
}
```

5 Results on the boards

Once the programs are already implemented, it is time to test it on the boards. With the results, a comparison between both processors can be done.

5.1 IRReceiver and Potentiometer

These programs cannot have been used to make comparisons because the IRReceiver only can be used in Arduino due to the pins and connections and in the potentiometer there is no time to be taken.

5.2 Pi

A number of 200000 iterations was set in this program. The accuracy of the program achieved to reach a number π with 10 decimals.
The times taken are the following:

Arduino DUE (ARM Cortex-M3): 3280 ms
Intel Galileo (Quark x1000): 63.7 ms

The time of calculating the π approximation in Quark x1000 is almost 50 times faster than ARM M3. Of course, the clock frequency of the the processors is related to the results. 84 Mhz vs 400 Mhz, so it is logic that Galileo is faster doing mathematical calculation. Moreover, Quark has a floating point unit, with a floating point calculation unit, floating point registers and so on, while Cortex-M3 only emulates the floating point operation. Those reasons make the Quark x1000 faster working with floating point operations than the Cortex-M3.

5.3 Quicksort

The limited memory of the Cortex-M3 allows to use a maximum length of the array of 2000 elements. However, it is a good number to make comparisons looking at the results.
Here, the results are split in two times, the time of creation the array and the time used to sort the table:

Creating the table:
Arduino DUE (ARM Cortex-M3): 280 μs
Intel Galileo (Quark x1000): 2.2 ms

The captions of the oscilloscope of this program are the following:

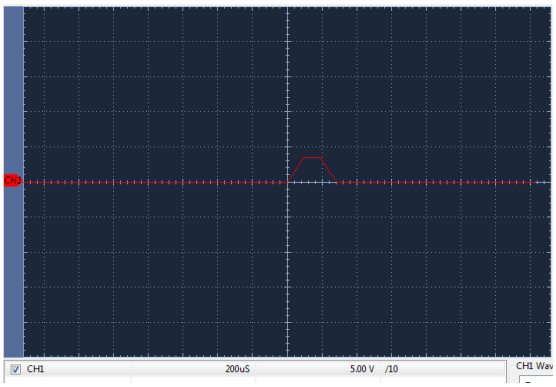

Figure 26: Time creating the array to quicksort on Arduino

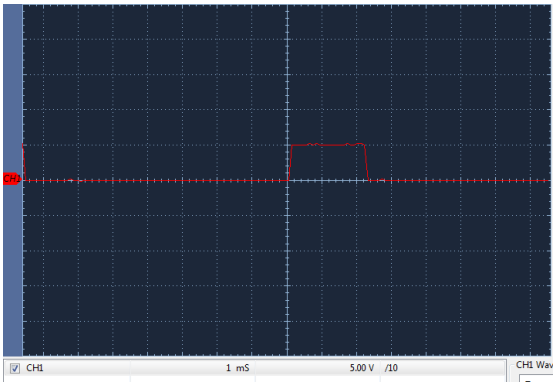

Figure 27: Time creating the array to quicksort on Galileo

Now, the time to create the table and write the 2000 elements on Cortex-M3 is much lower than on Quark x1000, 1000 times faster. That is to say that the memory management in the first processor is better than in the other one. It is curious because Quark counts with a cache, thus the memory access should be faster, and its RAM memory is much bigger, 512 KB vs 96 KB. The emulation of Arduino in the intel Galileo can affect this time.

Sorting the table:
Arduino DUE (ARM Cortex-M3): 270 ms
Intel Galileo (Quark x1000): 85 ms

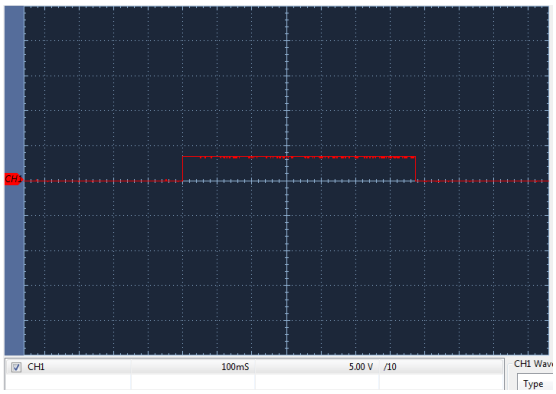

Figure 28: Time sorting the array to quicksort on Arduino

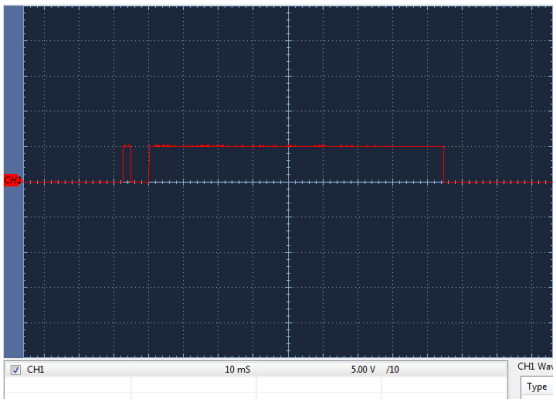

Figure 29: Time sorting the array to quicksort on Galileo

Now, the time sorting the elements is less in the Galileo Board, due to the faster clock frequency of the processor.

5.4 Determinant

Again, the memory of the Cortex-M3 doesn't allows to use more length of the matrix than 81 elements 9x9. But the results are big enough to make comparisons.
Again, the results are separated in two different times, the times of the creation of the matrix and the time used to calculate the determinant of the matrix:

Creating and storing the array:
Arduino DUE (ARM M3): 300 μs
Intel Galileo (Quark x1000): 2.4 ms

The captions of the oscilloscope of this program are the following:

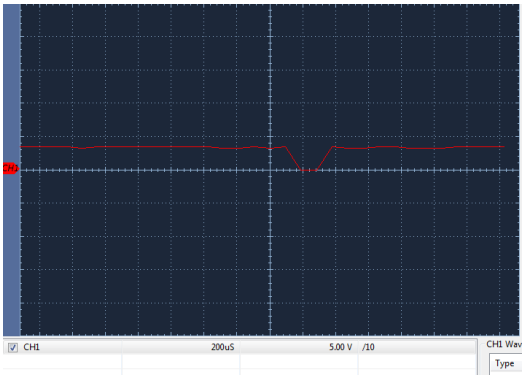

Figure 30: Time creating the matrix on Arduino

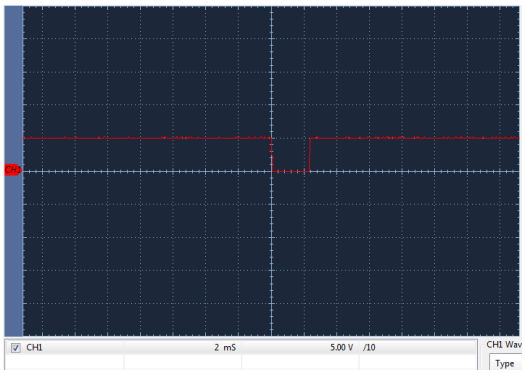

Figure 31: Time creating the matrix on Galileo

Again, the time to create a matrix 9x9 is 1000 times faster in Arduino than Galileo. It confirms that the memory management is better in Cortex-M3 than Quark x1000 that the results of the qsort experiment showed.

Sorting the table:
Arduino DUE (ARM M3): 1000 ms
Intel Galileo (Quark x1000): 380 ms

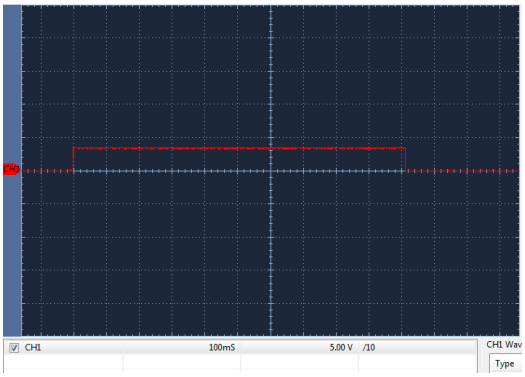

Figure 32: Time calculating the determinant of the matrix on Arduino

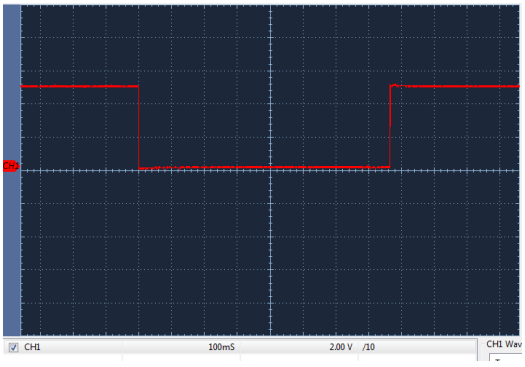

Figure 33: Time calculating the determinant of the matrix on Galileo

And, once again, the time calculating the determinant, working with floating point numbers again, is much less in Galileo than Arduino. It also confirms that Quark x1000 is faster doing mathematical calculations than Cortex-M3.

5.5 Interruption

Now, we want to see which processor is better with the interruption management, one of the most important features of the embedded processors. For that, the program previously explained is executed in both boards with the next results:

5.5.1 Arduino

First of all, experiments in Arduino board will be done. The board generates an output which variate depending on the branches and the machine instructions it should execute, as it can be seen in the following pictures (scale: 2μs per square)

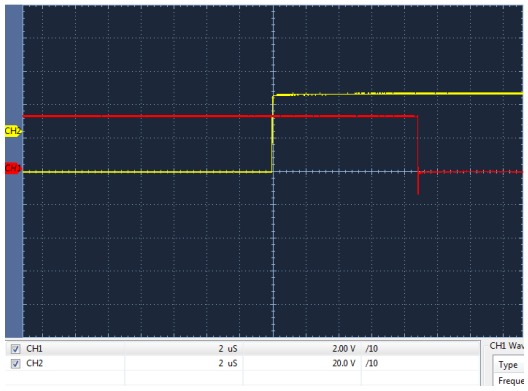

Figure 34: Minimum gap between waves in Arduino board (9μs)

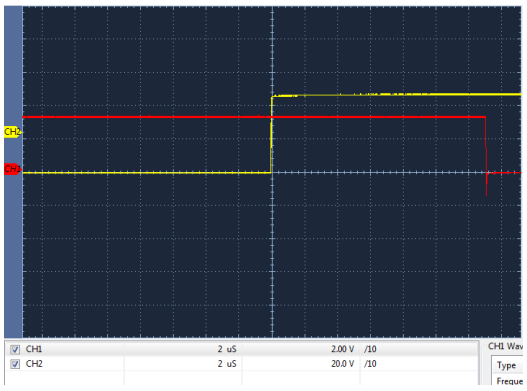

Figure 35: Maximum gap between waves in Arduino board (13μs)

As it can be seen in the pictures, the gap between waves goes from 9 to 13 μ. That is why, the time for the arduino boards will be the average between them, 11 μs.
This is the time the board uses to process an interruption.
According to the manual, interrupt latency lasts 12 cycles. As the processor frequency is 84 Mhz (cycle time 0.0119 μs), 12 cycles would be 0,1547 μs.
In this case, the information do not match with the information done by the board developer.

5.5.2 Galileo

Now, the experiments will be executed in the Galileo board:

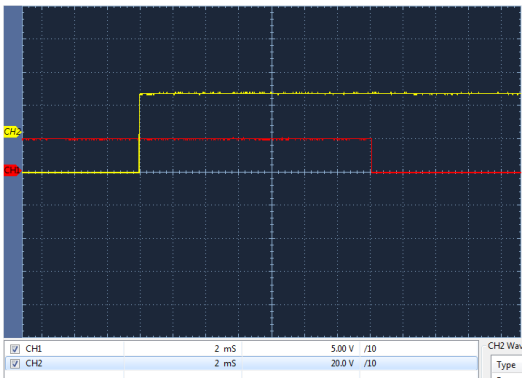

Figure 36: Minimum gap between waves in Galileo board (14 ms)

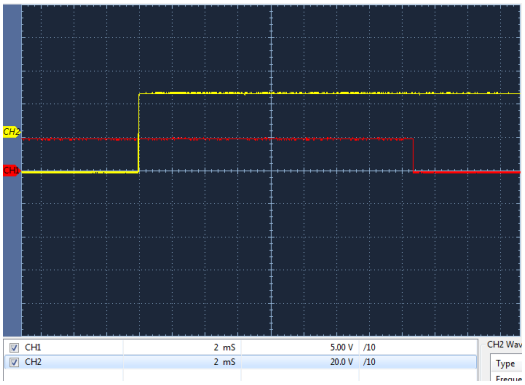

Figure 37: Maximum gap between waves in Galileo board (17.5 ms)

It is important to remark that now, the scale is set as 2 ms per square, so the time taken will be much bigger.
Again, the time taken will be the average between 14 and 17.5, so the time that Galileo needs to process a simple interruption would be 15,75 ms.

5.5.3 Comparisons of this program

This experiment is very useful to know which board takes more time to process interruptions. Arduino lasts 11 μs, Galileo 15,75 ms.
The result of this test is unusual. Galileo takes more than 1000 times more than Arduino Due does to process a simple interruption. It is strange because Galileo is a more powerful, newer board than Arduino. All the experiments done show that Galileo board takes much less time than Galileo processing data, even though it is less powerful processing interruptions.

6 Final highlights and conclusions

6.1 Tables with the results

The overall results sorted in two tables (Table 1 and Table 2), after doing 10 iterations.

Program	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Average
[1] Pi (ms)	3280	3280	3280	3280	3280	3280	3280	3280	3280	3280	3280
[2] Ir-Receiver(ms)	18	19	18	19	19	19	18	19	18	18	18,5
[3.1] Qsort(μs)	-*	-	-	-	-	-	-	-	-	-	280
[3.2] Qsort2(ms)	270	270	270	270	270	270	270	270	270	270	270
[4.1] Determinant (μs)	-*	-	-	-	-	-	-	-	-	-	300
[4.2] Determinant(ms)	-*	-	-	-	-	-	-	-	-	-	1000
[5] Interruption (μs)	-*	-	-	-	-	-	-	-	-	-	11

Table 1: Board with the results of the programs in the Arduino DUE (ARM Cortex-M3)

Program	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Average
[1] Pi(ms)	64	63	64	64	64	63	64	64	63	64	63.7
[3.1] Qsort (ms)	-*	-	-	-	-	-	-	-	-	-	2,2
[3.2] Qsort 2(ms)	-*	-	-	-	-	-	-	-	-	-	85
[4.1] Determinant(ms)	-*	-	-	-	-	-	-	-	-	-	2.4
[4.2] Determinant(ms)	-*	-	-	-	-	-	-	-	-	-	380
[5] Interruption(ms)	-*	-	-	-	-	-	-	-	-	-	15.75

Table 2: Board with the results of the programs in the Intel Galileo (Quark X1000)

The times taken are explained here:

1 - Pi: Time used calculating the Newton approximation of Pi, in 2000 loops.
3.1 - Qsort: Time needed to create the list of elements, which length is 2000.
3.2 - Qsort2: Time used to sort the array before created, totally unsorted.
4.1 - Determinant: Time taken to create and save in memory a matrix 9x9.
4.2 - Determinan2: Time taken to calculate the determinant of a matrix 9x9.
5 - Interrupt: Time to process a simple interruption.
(*Taken with the oscilloscope, there is no average)

6.2 Final conclusion

With these boards, it is possible to make final comparisons between both boards/processors. With these experiments, it is shown that the processor Quark X1000 is more powerful doing mathematical calculations and processing data in general. However, the ARM M3 is much more powerful and faster processing simple interruptions, what would be taken into account whether the interruptions in the embedded system where the processor will work are important or not. The Cortex-M3 is also faster in the programs in which the interaction with the memory play an important role.

Therefore, it is difficult to choose only one processor. The election would depend on the application the processor is required to execute. Thus, if the embedded system is designed to interact with the I/O via interruptions, or the speed with the memory management is important, Cortex-M3 would be chosen.
However, if the system has to work with mathematical calculations or calculations in general, and it is wanted that these calculations are fast, the election would be the Quark X1000.

References

[1] Intel staff: *Intel Quark Core HW Reference Manual.* Site: `http://caxapa.ru/thumbs/497461/Intel_Quark_Core_HWRefMan_001.pdf` . October 2013.

[2] Intel staff: *Intel Quark Core developer's Manual.* Site: `http://download.intel.com/support/processors/quark/sb/intelquarkcore_devman_001.pdf`. October 2013.

[3] Prof. Dan Grigoras: *Slides from Mobile Devices and Systems* 2015: University College Cork.

[4] ARM staff: *Cortex R4 datasheet* Site: `http://www.arm.com/products/processors/cortex-r/cortex-r4.php`

[5] A. Marti Campoy, E. Tamura, S. Saez, F. Rodriguez, and J. V. Busquets-Mataix *On Using Locking Caches in Embedded Real-Time Systems* 2005: Departamento de Informatica de Sistemas y Computadores: Universidad Politecnica de Valencia, Grupo de Automatica y Robotica: Pontificia Universidad Javeriana - Cali, Colombia

[6] Paul Markin *A Case-Study On The Impact Of L2 Cache Size On CPU Performance* 2003: www.overlockers.com

[7] Rogue Wave Software *CPU Cache Optimization: Does It Matter? Should I Worry? Why?* 2011: United States of America

[8] Dan Nicolaescu, Alex Veidenbaum, Alex Nicolau *Reducing Power Consumption for High-Associativity Data Caches in Embedded Processors* Dept. of Information and Computer Science, University of California at Irvine

[9] Andhi Janapsatya, Aleksandar Ignjatovic, Sri Parameswaran *Finding Optimal L1 Cache Configuration for Embedded Systems* 2006: School of Computer Science and Engineering, The University of New South Wales at Sydney, Australia

[10] Yujia Jin and Rong Chen *Instruction Cache Compression for Embedded Systems* Berkeley

[11] Bruce Jacob *Cache Design for Embedded Real-Time Systems* 1999: Electrical & Computer Engineering Department, University of Maryland at College Park

[12] Norman P. Jouppi *Cache Write Policies and Performance* 1991: Western research laboratory at University Avenue Palo Alto, California

[13] Bruce Jacob *Software-Managed Caches: Architectural Support for Real-Time Embedded Systems* 1998: Electrical Engineering Department, University of Maryland, College Park

[14] Bach D. Bui, Marco Caccamo, Lui Sha, Joseph Martinez *Impact of Cache Partitioning on Multi-Tasking Real Time Embedded Systems* 2007: University of Illinois at Urbana-Champaign, Lockheed Martin Aeronautics Company

[15] Prabhat Jain *Software-assisted Cache Mechanisms for Embedded Systems* 2008:Massachusetts institute of technology

[16] Emre Ozer, Resit Sendag and David Greggr *Multiple-Valued Caches for Power-Efficient Embedded Systems* 2005: University of Rhode Island, Trinity College Dublin

[17] Abu Asaduzzaman *Cache optimization for real-time embedded systems* 2009: Florida Atlantic University

[18] Dominic Hillenbrand, Jorg Henkel *Block Cache for Embedded Systems* University of Karlsruhe

[19] Raull Mata Botana *Tablas en Latex* 2008.

[20] David R. Wilkins *Getting started with Latex*

[21] Chuanjun Zhang, Frank Vahid and Walid Najjar *A Highly Configurable Cache for Low Energy Embedded Systems* 2005.

[22] Venkat Rao, Gaurav Singhal, Anshul Kumar , Nicolas Navet *Battery Model for Embedded Systems* 2005: Random House, N.Y.

[23] Software Engineering Group, Department of Computer Science *Guidelines for performing Systematic Literature Reviews in Software Engineering* 2007: Keele University and University of Durham.

[24] Nabilah Kamarul Bahrin, Radziah Mohamadi *A Systematic Literature Review of Test Case Generator for Embedded Real Time System* 1986: Faculty of computing of Universiti Teknologi Malaysia 2005.

[25] www.arduino.cc *Arduino Due Board information* Site: `http://www.arduino.cc/en/Main/arduinoBoardDue`

[26] www.arduino.cc *Intel Galileo Board information* Site: http://www.arduino.cc/en/ArduinoCertified/IntelGalileo

[27] Sorin Cotofana, Stephan Wong, Stamatis Vassiliadis *Embedded Processors: Characterictics and Trends*

[28] Image from: https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcT0UgHy9HvB-81lyS7JMIEJ-0BWnAkMZLoPJ2fg5ne4yFJDKtzg

[29] Image from: https://www.testequipmentconnection.com/images/products/Array_Electronic_3400A.JPG