

Helsinki Metropolia University of Applied Sciences
Degree Programme of Electronics

JESÚS CHOZAS ROBLEDO

HOW TO DESIGN A LOW COST UHF RFID READER

Bachelor's Thesis. May 24, 2010

Instructor: Janne Mäntykoski, Lecturer

Supervisor: Kenneth Kronkvist, Project Manager

Author	Jesús Chozas Robledo
Title	How to design a low cost UHF RFID reader
Number of Pages	139
Date	May 24, 2010
Degree Programme	Electronics
Degree	Bachelor of Engineering
Instructor Supervisor	Janne Mäntykoski, Lecturer Kenneth Kronkvist, Project Manager
<p>This Bachelor's Thesis studied the design of a low cost UHF RFID Reader. The goal was to get a low cost design; each decision made during this thesis project followed this idea. The thesis is divided in two parts, clearly differentiated but still related to each other, as it is explained below.</p> <p>A requisite of the UHF RFID Reader is having Ethernet interface, so this solution is treated after the first part. In other words, the first part of this thesis focuses on the Radio Frequency Identification world while the second part concentrates on computer networks and microcontroller programming.</p> <p>The aim of the first part (chapters 1-3) is to give a general view about the design of a low cost UHF RFID Reader. So a high level design is used to explain what the reader hardware design as well as the main materials and components would be like. In addition, the use of each element involved in the reader design has been explained and justified for a better understanding of the design.</p> <p>The second part of the thesis (chapters 4 and 5) focuses on the Ethernet interface. To carry out this part of the project, a microcontroller and an Ethernet controller were used in such a way that by connecting an Ethernet cable to a computer a communication test was carried out. For this reason, a TCP/IP stack was coded to allow making a 'ping' between both systems (computer and Ethernet Interface). This part has been documented at the end of this thesis.</p>	
Keywords	RFID, UHF, UHF RFID reader, passive tags, TPC/IP, Ethernet interface, ARM7

TABLE OF CONTENTS

1	Introduction.....	5
2	RFID System.....	6
2.1	History of Radio Frequency Identification (RFID)	6
2.2	Components of an RFID System	7
2.2.1	Reader, Transceiver or Interrogator	8
2.2.2	Transponder or Tag	8
2.2.3	Middleware or Reader Interface Layer	10
2.2.4	RFID System.....	11
2.3	Uses of RFID Systems	12
2.4	Regulations and Standards	14
2.4.1	RFID Frequency ranges	14
2.4.2	Regulations and Standards.....	15
2.4.3	Information about the Standards	16
2.5	UHF RFID Fundamentals	25
2.5.1	Analog Part	25
2.5.2	Digital Part	34
3	Design of an UHF RFID Reader.....	35
3.1	Components of the Analog Part.....	38
3.2	Components of the Digital Part.....	40
3.3	Power Supply	41
4	Reader to host interface.....	43
4.1	Introduction.....	43
4.2	Ethernet Interface.....	44
4.3	Problems in the Implementation	45
4.4	Materials Used	47
4.4.1	AT91SAM7s-EK Evaluation Kit.....	47
4.4.2	PICtail Ethernet Board (AC164121).....	48
4.4.3	IAR J-Link	49
4.4.4	IAR Embedded Workbench.....	50
4.5	Serial Peripheral Interface (SPI)	50
4.5.1	SPI Overview	50
4.5.2	ECN28J60 Memory	54
4.5.3	ECN28J60 SPI Instruction Set.....	58
4.6	LAN Overview.....	64

5	Pinging the Ethernet Interface	73
5.1	Ethernet Controller Connection	73
5.2	Steps before coding	73
5.3	Ping Utility Programming	76
5.3.1	setup_SPImaster.c	76
5.3.2	Timer0.c	79
5.3.3	Timer0_IrqHandler.c	85
5.3.4	ini_ecn28j60.c	86
5.3.5	ECN28J60_.c	89
5.3.6	arp.c	96
5.3.7	icmp.c	98
5.3.8	main.c	101
5.4	Problems during the Development of the Ping Application	102
6	Conclusions	108
7	References	109
8	Appendices	112
	Appendix 1... setup_SPImaster.c	112
	Appendix 2... IRQ1_Handler.c	113
	Appendix 3... timer0.c	114
	Appendix 4... timer0_IrqHandler.c	115
	Appendix 5... usart0.c	116
	Appendix 6... ini_ecn28j60.c	119
	Appendix 7... ecn28j60.c	121
	Appendix 8... arp.c	132
	Appendix 9... icmp.c	135
	Appendix 10..main.c	139

1 Introduction

The Thesis was done as a part of the Visual RFID project in the Electria department of Helsinki Metropolia University of Applied Sciences. The topic of this project was to design a low cost UHF RFID Reader by using a high level design. Therefore a general view is given about the main elements that compose the reader.

To understand how a UHF RFID Reader works, several skills across a wide spectrum of disciplines were required for the thesis project. Some examples of these disciplines are: Design in High Frequency, Physics, Radio Frequency, Telecommunication Systems, Modulation, Antennas, Microcontroller Programming and Computer Networks.

Radio Frequency Identification (RFID) has a wide field of applications. As we will see in later sections, the main application of an RFID system is oriented to the identification of objects. Due to their big advantages, RFID Systems are largely used to substitute the older bar codes. Nowadays their use also encompasses other applications such as the telemetry and medicine.

As RFID technology influences our lives significantly, it also has its detractors. Its use has generated controversy in one part of the population due to the lack of security in some applications, but the truth is that the advantages of RFID make life so much easier.

The idea of designing a low cost reader arose from the fact that commercial UHF RFID Readers cost around a thousand Euros or even more and they are designed to be used in standard applications. Therefore, it can be a good idea to design your own reader for your applications.

In short, this project gives a general view about designing UHF RFID reader. The different sections of the thesis aim to cover as comprehensively as possible all the aspects related to RFID and the RFID readers to give to the reader a good understanding of this kind of system.

2 RFID Systems

2.1 History of Radio Frequency Identification (RFID)

Although Radio Frequency is a fairly recent technology, the fact is that it has been used for a long time. The history of Radio Frequency dates back to the 19th century when the study of the electromagnetic waves started; this was the key in the development of this technology.

For this reason, the most important events since the 19th century have been summarized below:

- 1864: James Clerk Maxwell demonstrated that electric and magnetic fields travel through space in the form of waves and published his theory about electromagnetism.
- 1887: Heinrich Rudolf Hertz carried out experiments with radio waves (transmission and reception) in his laboratory.
- 1897: Guglielmo Marconi succeeded in transmitting a message over a distance of 6 km without any cable across the Bristol Channel. [1]

But it was not until the 20th century that the development of the modern radio communication was a fact, thanks to the development of the Radar.

- 1904 is considered as the year when the Radar was invented. One of the first experiments with the Radar was realized by Christian Hülsmeyer who detected a ship in the fog.
- By the 1930s, the Radar was employed during the World War II by the allies as a form of intercepting the enemies' planes. [2]

Its function has not changed with the years since it is possible to determine the position and distance of an object by the reflection of radio waves.

The world had to wait several years later, till 1948 to see the true “birth” of the RFID technology. In this year, engineer Harry Stockman published the first work known about the study of RFID, called “Communication by Means of Reflected Power” [1].

The study of RFID did not stop, but it continued to evolve thanks to other epoch inventions such as the transistor (1947), the integrated circuit (1958), the microprocessor (1971) and the development of communication networks.

- The 1960s were the prelude to the takeoff of RFID because it was used in commercial activities. It was promoted by such companies as Sensormatic, Checkpoint and Knogo. They developed the Electronic Article Surveillance (EAS) to prevent shoplifting which is still being used.
 - In 1975, Alfred Koelle, Steven Depp, and Robert Freyman published their work “Short-Range Radio- Telemetry for Electronic Identification Using Modulated Backscatter”. They carried out this study at Los Alamos Scientific Laboratory, and their findings were the beginning of short range passive tags (tens of meters).
 - In 1980s the RFID technology was implanted in products thanks to the invention of the personal computer. The computer enabled the harvesting and management of the data in RFID systems.
 - In 1990s the use of RFID was introduced in motorway tolls and the rail sector. Also, the first standards started to emerge and RFID become a part of everyday life.
- [1]

In the 21st century, the design of tags has advanced due to miniaturization (it is limited by the constraints of the antenna). A tag can now be manufactured using only two components: a single CMOS integrated circuit and an antenna. Nowadays the development of systems based on RFID continues its global expansion.

2.2 Components of an RFID System

After the reviewing the history of RFID, it is time to study this technology as well as the parts that compose an RFID system in more detail.

RFID, as the term indicates, is a communication system based on radio waves with the aim of identifying objects (animal, person or thing). First, the RFID elements that allow identifying the object are presented. After that, a brief introduction is provided about how the system works.

2.2.1 Reader, Transceiver or Interrogator

The reader is the most important element of the system. Basically, it can read and write into the tags by means of its antenna (in general, from one to four). The reader antenna or antennas can be integrated in the reader or be physically separated and connected with a cable. Since in this project is studied the design of a UHF RFID Reader, it will later be discussed in further detail (see chapter 3) [2, 7].

2.2.2 Transponder or Tag

The transponder consists of a coupling element (an integrated antenna) and at least one integrated circuit (IC). This IC contains the EPC (Electronic Product Code) and the logic necessary to understand the communication protocol between the tag and reader. As per the tag characteristics, can be classified in three groups: Passive, Semi-Passive and Active. Each tag group is explained in more detail below.

a. Passive tags

Passive tags do not have their own source of power (without battery). The tag uses the power supplied by the reader to activate the IC tag. That is, the power is obtained through the tag antenna when it enters in the interrogation zone of the reader. In other words, the tag is powered by rectification of the received power. Moreover, the tag returns a modified signal (signal backscatter) to the reader (tag's information). Passive tags have a maximum reading distance of 3 meters [3, 34]. Figure 1 illustrates how passive tags are built.

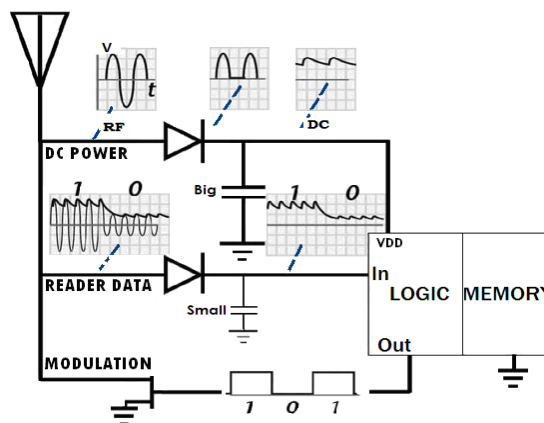


Figure 1. Schematic Depiction of a Simple Passive RFID Tag [3, 36].

As seen in Figure 1, a high-frequency signal (RF) is picked up when the antenna is influenced by an electromagnetic field of a nearby reader. This RF voltage that will be rectified by the diode and then smoothed by the capacitor to get a constant voltage that is able to power the tag's memory and logic circuitry.

A similar rectification is carried out, at the same time, with a smaller capacitor, with the aim of demodulating the information from the reader. This is made with an envelope detector that allows varying the voltage in the timescale of the reader data. Finally, to transmit the information back, the tag has to change the electrical characteristic of the antenna to modify the signal received. In Figure 1, this has been represented with a FET although in a real tag but the process is a bit more complex [3, 35-36].

b. Semi-Passive tags

Unlike passives tags, semi-passive tags have a battery for the power supply but still use the backscatter for the communicating with the reader (see Figure 2).

Semi-passive tags have a better operating range (from tens of meters to as much as 100 m) and give a better response to a valid interrogation than passive tags. Nevertheless, they are bigger due to the battery, cost more and are harder to maintain. Their applications are mainly oriented to automobile tolls and to tracking of high-value pieces, for example airplane parts [3, 37].

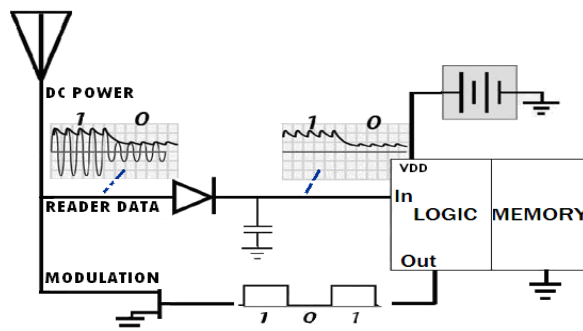


Figure 2. Schematic Depiction of a Semi-Passive RFID Tag [3, 38]

c. Actives tags

Active tags not only have batteries but also a transmitter, that is, they can be configured as a conventional bidirectional radio communications device. These tags can cover distances of more than 100 meters even kilometers.

Figure 3 illustrates the complexity of these tags. Active tags can use frequency-division multiplexing or different frequency channels when they want to communicate in the presence of other tags. Also, an active tag can communicate within a specific frequency band by the means of a Local Oscillator (LO) and a crystal reference (XTAL).

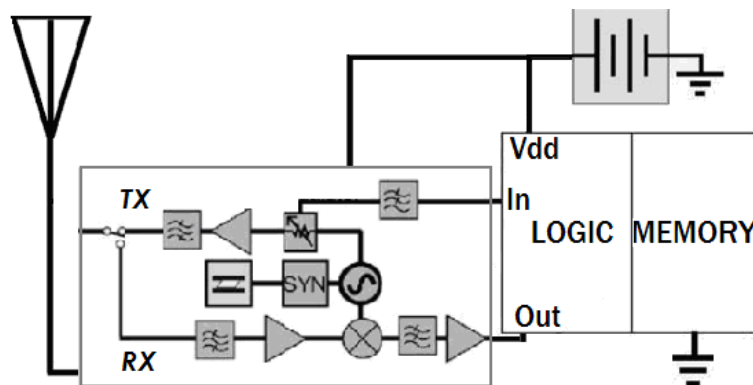


Figure 3 Schematic Depiction of an Active RFID Tag [3, 40].

Thanks to the use of Code-Division Multiple Access (CDMA) it is possible to reuse the same frequency band by multiple tags. So the active RFID tag can be successfully used in environments where the tag-reader path is significantly obstructed. One example is metal shipping containers that are stacked near each other without having a visual line from the reader to the tag [3, 40].

2.2.3 Middleware or Reader Interface Layer

An interface is necessary between the reader and the mainframe or host. Its function is to 'translate' the tags' signals into single identification data. Also, it acts as a link between the RFID hardware and the client's application.

The interface is often a software that runs on computers or servers and consists of middleware, which contains the logic of the RFID application and a backend database system (e.g. Oracle, SQL Server, MySQL, etc) for storing information about the tags (typically, the identification number and perhaps some item-specific information) [4, 16-17; 5, 5].

2.2.4 RFID System

This subsection explains how a Passive RFID system works.

As we have just seen, both tags and readers have their own antennas. When the transmitting antenna of the reader emits a Radio Frequency field, the nearby tags are interrogated when they pass through the RF field. The distance of this emitted field depends on the frequency and the power output used.

After the passive tag is activated, it sends (backscatters) the programmed information into its memory back. Next the reader antenna receives and detects this response. Then the backscatter signal is decoded with the purpose of sending the data to the host.

Sometimes, the reader can also broadcast special signals to a tag (e.g. to synchronize a tag with the reader for interrogating all or part of the tag's contents). Once the reader sends the data to the middleware, the data are kept to the host. In other words, the middleware handles the interface between the RFID hardware operations and the flow of data (e.g. different electronic product codes in a database).

The Middleware includes the following elements:

- *Reader and device management*

With a common interface, it is possible to configure, monitor or execute commands directly to reader.

- *Data management*

It is able to filter the captured information from readers (EPC or other data) and route it to its destination.

- *Applications integration*

It provides solutions for messaging, routing and connectivity. The RFID data can be integrated into a supply-chain management (SCM), enterprise resource planning (ERP), warehouse management (WMS), or customer relationship management (CRM) systems.

- *Partner integration*

It provides collaborative solutions like business-to-business (B2B). [6, 81]

In short, an RFID network can be defined as a peer-to-peer architecture able to send data to a central host by the middleware. The middleware is responsible for linking the data between different networks.

2.3 Uses of RFID Systems

Nowadays the use of RFID systems is growing rapidly. In fact, we can find this kind of technology in various everyday functions:

a. Payment by mobile telephone (named the In2pay Solution)

This solution was developed by DeviceFidelity in Dallas (Texas); it is based on a microSD card with an RFID module. This card is able to transform your mobile phone into a contactless payment device. Once the card is inserted, the mobile phone behaves like a Near Field Communication (NFC) passive tag and a reader. The expansion is expected in 2011 [7].

b. Transportation payment

- Toll motorway: the drivers do not need to stop to pay for the toll because an RFID tag is fixed to the car. When the car is next to the toll entrance, the tag is read by the toll readers and the toll gate rises.
- Public transport: the user is an owner of an RFID card that allows an easier and faster access. Moreover, some of the cards can be recharged at any time.

c. Product tracking

It is an application used with the aim of locating any shipment. That is, you can know in real-time where a certain product is.

d. Animal identification

In this application, the type and location of a tag depend on the type of the animal. For livestock, the tag is put in their ears (or in the paw in the case of fowls). In the case of the pets such as dogs or cats, the tag is a microchip which is implanted under their skin.

e. Libraries

RFID tags are used to store information related to books (e.g. title, author, genre, etc). Thus, the search of a book in a database is something easier.

f. Inventory systems

Companies use RFID for inventory control in an automated process, so time is saved and costs are reduced.

g. Human Identification

Nowadays, there are many examples of RFID applications in our lives, for example passports, race timing and ski resorts lift tickets.

h. Anti-thief device in shops (EAS)

Each article carries a tag that must be deactivated by the sales assistant after a client buys the product to avoid triggering the alarm system. The use of EAS is oriented to electronics devices, books, DVD's and cloths for example

• **Evolution and Innovation in the Uses of RFID**

One adaptation and improvement in the use of RFID is to combine the tags with different sensors. In this way, the tag delivers the identification information repeatedly, and the current data is picked up by the sensor. An example of this application in the alimentation market would be an RFID tag attached to a piece of meat that could report on the temperature readings ensuring that the meat is properly kept cool [8].

Other advancements in the RFID world are:

- The self-scanning technology allows reducing the waiting time in a store checkout line because the items selected will be charged automatically from your bank account.
- Applications in the field of medicine.
- Energy harvesting.

2.4 Regulations and Standards

Similarly to the worldwide expansion of RFID, the number of standards has risen. In principle, every country can set its own rules because nowadays there is no global public body which can regulate the frequencies used for RFID. Furthermore, the standards vary according to the tag type and the operating frequency.

2.4.1 RFID Frequency ranges

a. Low Frequency (LF) – [30 KHz ~ 300 KHz]

The typical frequencies used in RFID are 125~134.2 KHz and 140~148.5 KHz. LF is used in animal ID (standard ISO 14223/1) or in car applications. At this frequency, both the read range (less than 0.5 meter) and the data transfer rate (less than 1Kbit/s) are low.

b. High Frequency (HF) – [3 MHz ~ 30 MHz]

Typically, the HF value used for RFID is 13.56 MHz. Usually it is applied in smart tags (standard ISO 15693). The read range is higher than that of LF (up to 1.5 meter), but its data transfer rate is low (less than 25Kbit/s).

c. Ultra High Frequency (UHF) – [300 MHz ~ 3 GHz]

The frequency bands used in RFID are: 433 MHz, 865~960 MHz and 2,5GHz. Generally, these frequencies are employed for animal tracking or in logistic applications. The read range is 0.5 -5 meters for 865-960 MHz, but it can go up to the 100 meters to 433 MHz. Moreover, this frequency band has a data transfer rate larger than that of HF (30Kbit/s to 433 ~ 956 MHz).

d. Microwave – [2 GHz ~ 30 GHz]

The typical frequencies used in RFID are 2.45GHz and 5.8 GHz. It is used mainly in vehicle tolls because the read range is up to 10 meters. Moreover, it has a bigger data transfer rate than UHF (up to 100Kbit/s). [2, 161-166]

Apart from this division, the regulatory agencies of each country create their own standards for each frequency range. Some examples of the regulatory agencies are:

- Europe: ETSI (European Telecommunications Standard Institute).
- USA: FCC (Federal Communications Commission).
- China: SAC (Standardization Administration of China).
- Japan: MPHPT (Ministry of Public Management, Home Affairs, Post and Telecommunication). [9]

2.4.2 Regulations and Standards

As this thesis focuses on the use of UHF RFID Reader in Europe, the European standards will be reviewed below.

In October 2003 EPCglobal was founded with the purpose of regulating and unifying the different standards. EPCglobal is a joint venture between GS1 and GS1 US. GS1 is a private organization dedicated to the development of global standards which got the global adoption of the barcode in the 1980s [10]. Table 1 presents the global regulatory situation for UHF for the main countries.

Table 1. Frequency Regulations for UHF [9]

	Europe	North America	China	Japan	Australia	New Zealand
Band (MHz)	865~868	902~928	840,5 844,5	952~954	865,6 867,6	864~868
Power (EIRP)	2W	4W	2W	4W	2W	4W

The data of the table above were obtained from the latest report of EPCglobal done on March 18th, 2009.

Although Table 1 only shows some frequency regulations, the difference between countries is evident. This means that the same reader has to transmit at different frequencies and with more or less power depending on where it is installed. Moreover, a tag must be capable of answering to different frequencies to fulfill all regulations. In other words, an RFID reader designed for Europe can not be used in America unless its configuration and external components are changed.

The global standard defined by EPCGlobal for the UHF band is “EPCglobal UHF Class1 Gen2”. This standard defines the physical and logical requirements necessary in the communication protocols with passive tags for the frequency range between 860MHz - 960MHz. Apart from this standard, each region has its own normative or standard.

As seen in Table 1, the UHF range for Europe is 865-868 MHz. The standard that regulates this is the ETSI EN 302 208 standard [13]. It was adopted in 2008 by ETSI to regulate and describe the use of the UHF band for RFID applications.

To conclude this part, another important standard needs to be mentioned. It is the Low Level Reader Protocol Standard (LLRP) [14], an EPCglobal standard for the interface between the RFID reader and the client. [11]

2.4.3 Information about the Standards

This subsection summarizes the contents of the standards mentioned in the previous subsection. But evidently it does not mean no check them.

- ◆ “EPCTM Radio-Frequency Identity Protocols Class-1 Generations-2 UHF RFID Protocol for Communications at 860 MHz -960 MHz”
-

This document defines the specification for an RFID Air Interface. In other words, it describes the modulations and encodings used in RFID as well as the operating procedures and commands between the reader and the tag. The protocols used in this RFID communication are described below.

▣ Physical Layer Communications

Although this subsection distinguishes between the kinds of modulation and encoding of each physical layer, they are explained in detail in a later section. The physical layer can be divided into two types according to the direction of communication: reader to tag and tag to reader.

a. Reader to Tag

The reader uses modulation with an RF carrier to send information to one or more tags. There are three types of modulation:

- SSB-ASK (Single Side Band- Amplitude Shift Keying).
- DSB-ASK (Double Side Band- ASK).
- PR-ASK (Phase Reversed-ASK).

The encoding format is PIE (Pulse-Interval Encoding).

b. Tag to Reader

Tags answer to the reader by backscatter. For this reason, the tag modulates the amplitude and/or phase of an unmodulated RF carrier sent previously by the reader. There are now two types of modulation:

- DSB-ASK (Double Side Band- ASK).
- PR-ASK (Phase Reversed-ASK).

Also there are two types of encoding format:

- FM0
- Miller-modulated subcarrier.

□ Tag-Identification Layer Communication

This subsection describes the Tag-Identification layer communication. Note that the following information has been extracted from an abstract made by Texas Instrument called “UHF Gen 2 System Overview [12]”.

A tag memory is composed of four banks of non-volatile memory (Figure 4). The content of each bank is explained in the next page.

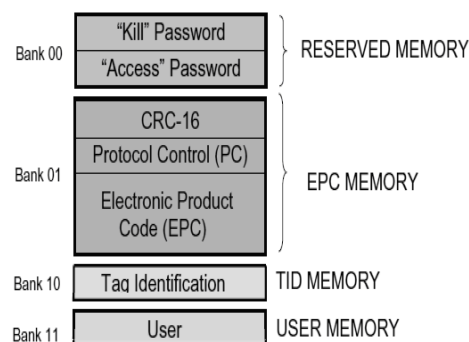


Figure 4. Scheme of a tag's memory banks [12, 22]

- a.* Reserved Memory (bank 00) contains the 32-bit tag's passwords.
- "Kill" password is used to silence a tag permanently if its value is zero.
 - "Access" password executes all access commands in a tag after having passed to a *secured* state.
- b.* EPC Memory(bank 01) contains:
- The actual EPC code.
 - A 16-bit Protocol Control (PC).
 - A 16-bit CRC calculated on the PC and EPC.
- c.* TID Memory (bank 10) has the information for the tag identification.
- 8-bit ISO 15963 allocation class identifier
 - A 12-bit Tag mask-designer ID.
 - A 12-bit Tag model number.
 - Possible manufacturer information.
- d.* USER Memory (bank 11) is an optional area where the user can keep any data.

The memory access is done by the reader by three operations (Select, Inventory and Access). Figure 5 illustrates the transition between these three operations before a reader can interact with tags.

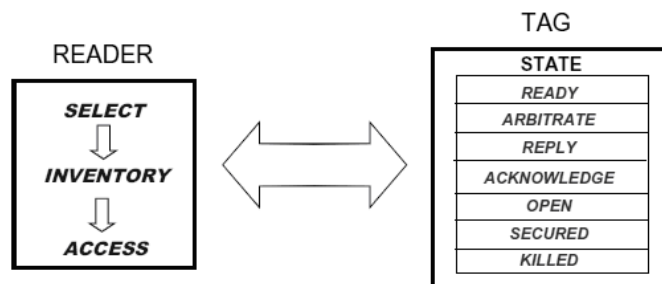


Figure 5. Tag-Identification Communication [12, 26]

The three different operations are explained below.

a. *Select operation*

As its own name indicates, it is employed to choose a tags population which will be part of the next inventory round. The reader chooses between one of four sessions (S0, S1, S2 and S3) and inventories the tags associated with that session. For each session the tag maintains an independent inventoried flag to indicate if it can answer to a reader with the possible flag value A or B.

b. *Inventory operation*

It uses a random algorithm for identifying (singulate) tags. It comprises five commands which are:

- Query: This command initiates the singulation process for selecting tags during the interrogation process. Moreover contains a slot-counter value ($0 \leq Q \leq 15$).
- Query Adjust: This command decrements the tag's slot-counter without changing any other parameters.
- QueryRep: This command repeats the last Query command.
- Ack: This command acknowledges a tag response.
- Nak: This command forces the *arbitrate* state.

Once the reader accesses the tag memory, then the tag can pass by different states. In other words, a tag works like a state machine (see Figure 6). The tag's stats are described below.

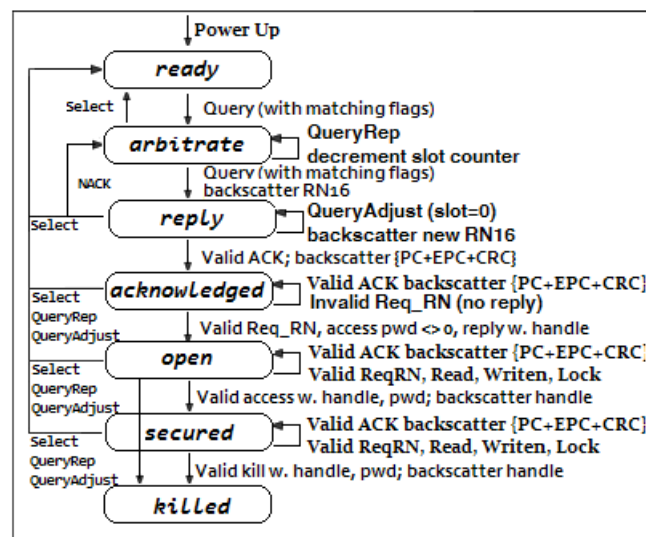


Figure 6. Tag's states diagram [3, 423]

A tag will pass to the *ready* state when it receives a Query command with its slot-count parameter Q (0 ~ 15) from the reader. So the command verifies if the tag belongs to the selected group. In such case, the tag picks a random value between 0 and $2^Q - 1$. Depending on its value the tag enters one state or another.

- When the value is zero, the tag will pass immediately to the *reply* state and backscatters a 16-bit (RN16) random number to the reader, in such way that it sends an ACK command with the same 16-bit random number. After that, the state of the tag changes to *acknowledged* state and the tag backscatters the content of its EPC memory. Then, the reader sends a QueryAdjust command so the identified tag inverts its inventoried flag (A→B or B→A) and the state transitions to ready state.
- When the value is not zero, then the tag will store the random number in its slot-counter and it will stay in *arbitrate* state until further commands.

Others issues to take into account are:

- If more than one tag responds at the same time and the reader cannot resolve the collision by sending a valid ACK command and each tag will return to *arbitrate* state. After that the reader sends a QueryAdjust command which causes a decrement in the slot counter of each unsolved tag. Only when the slot counter gets to zero, the tag will pass to *reply* state.
- At any time a reader can send a NAK command forces all tags back to *arbitrate*.

c. Access Command

Once the tags have been identified and can be located, the access operation is the last operation before the reader can read or write into tags. This operation mode can only be used when the tag is in *open* state (or *secured* state if its password is zero).

For the transition from *acknowledge* to *open* state, the reader has to send a request random number (Req_RN). Then the tag backscatters another random number (RN16) called 'handle' which is used later by the Access commands.

There are seven kinds of access commands which can be classified in two different groups (obligatory and optional) which are explained below.

- ◆ Obligatory

1. Read command allows to the reader to access the tag's memory .The tag response can be success, error or failure (timeout).
2. Write command: The data is sent encrypted with the aim of changing tag memory locations through the access to tag memory. It is necessary to request a new handle for each new command. The tag response can be success, error or failure (timeout).
3. Kill command disables a tag permanently. If its 32-bit encrypted password is zero, this command does not work. On the contrary, two Kill commands (each one of 16 bits) are sent. As with the Write command, a new handle is requested before other Kill command. The tag backscatters its 'handle' and remains silent, it does not, it indicates a command fail.
4. Lock command allows three actions:
 - Lock individual passwords
 - Lock individual memory banks
 - Permanently lock the tagAnd the tag response can be success, error or failure (timeout).

- ◆ Optional

1. Access command allows the tag to change its state from *open* to *secured* when the encrypted password (32bits) is not zero. In the opposite case, it is necessary to send two commands. The tag response can be success, error or failure (timeout).
2. Block Write allows writing multiple blocks into tag's Reserved, EPC, TID and User memory. The tag response can be success, error or failure (timeout).
3. BlockErase allows erasing multiple blocks from the tag memory (Tag's Reserved, EPC, TID or User memory). The tag response can be success, error or failure (timeout).

Finally to conclude this subsection, the other documents connected with the European regulations, EN 302 208 and LLRP, are presented below.

Although, this standard is composed by two volumes, the second volume makes reference to the first volume.

As seen in Table 1 on page 15, the bandwidth available is 3 MHz (865 ~868 MHz). Although it is divided in 15 channels, just four of them are of high-power (2W or 33dBm Effective radiated power [ERP] which is determined by subtracting system losses and adding system gains). Each one of the high-power channels has a bandwidth of 200 KHz and the center frequency of the lowest channel is located at 865.7 MHz. These high-power channels are separated from each other by equal intervals of 600 KHz, as can be seen in Figure 7.

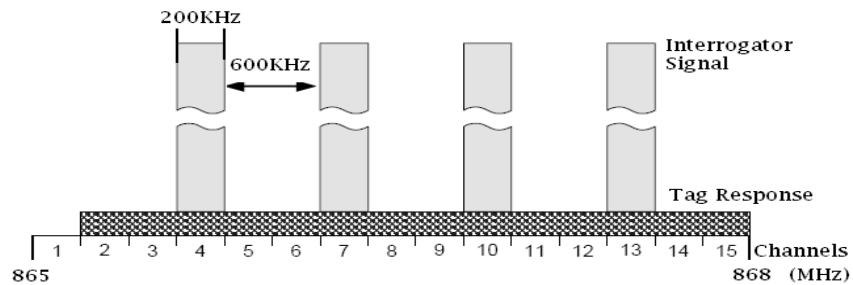


Figure 7. Diagram of Channel plan [13, 12]

RFID readers transmit in one of the high-power channels (4, 7, 10 or 13) using a modulated carrier, and preferably the tags respond in the adjacent low power channels (865~868 MHz) with a modulated signal. It is important take into account that we cannot exceed the power limits defined in the spectrum mask Figure 8).

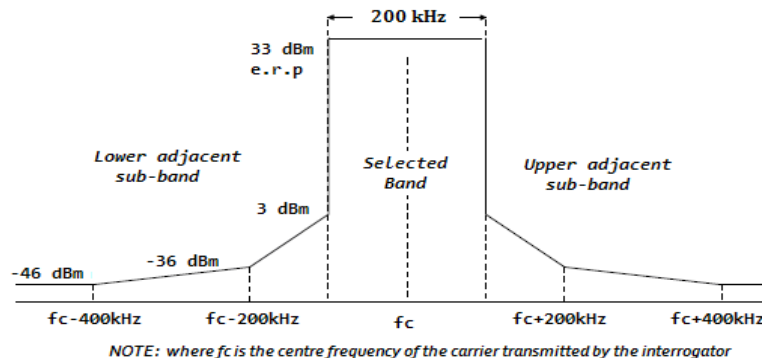


Figure 8. Spectrum Mask for modulated signals [13, 25]

In the current version, the use of “Listen Before Talk” (LBT) is optional, and its use and normative can be consulted in the annex B (volume 1) of this document.

LLRP (v.1.0.1)

In April 2007, EPCglobal ratified the Low Level Reader Protocol (LLRP) standard [14]. It is a specification for the network interface between the reader and its controlling software or hardware (client). It can be classified in three groups according to their function:

1. LLRP Reader Software is used to can communicate with readers by standard interface.
2. Client Software is used to read and write RFID data as to control the wireless aspects of readers.
3. LLRP Software Tools, this function helps to develop, administrate or maintain RFID systems using LLRP.

LLRP is an application layer protocol which is communicated through data units or messages. The characteristics of the data differ according to the direction of communication (reader-to-client or client-to-reader).

- From client to reader, the operations are the capability to find and configure readers, as well as to carry out the management of the select, inventory and access operations for the communication with tags.
- From reader to client, these messages are mainly reports, status notifications (inventory and access results) or keep lives only known by the client.

The major advantage of LLRP lies in that its iteration is not based on real-time because there is an asymmetric protocol between a client implementation of LLRP (application software) and the reader. In other words, a reader will perform time-critical functions in such a way that the application software can pass operational rules to the reader in non-real time and later trigger them to activate in real time.

This manner of operation allows the reader to be autonomous without having a real-time control interface from the host. It can be remote controlled through the same network. Moreover, it performs without any constraints caused by the network or host latency. This protocol has drawbacks, for example it does not have a retransmission facility.

Another drawback is that the LLRP is a binary protocol which needs to be learned but for this thesis the binary protocol is not such a big problem because as it will be seen in later chapters, in reader design is based on the use of a reader chip from Impinj. This company has created an open source programming toolkit for LLRP which can be found at www.llrp.org. This toolkit aims at facilitating a test tool for LLRP-based applications and simplifying the transactions between basic LLRP messages for example.

In short, the basics tasks of LLRP are as follows:

- Configuration of the reader according to the reader application.
- Sending Reader Operation Specification (ROSpec) commands to the reader. They contain a list of commands for reader operation called Antenna Inventory Specifications (AISpec).
- Sending AISpec commands to the reader. They tell the reader what type of data access operation (either read or write) has to perform on the tag.
- Getting the reports of information from the RFID reader.

Figure 9 is an example of the Command Sequencing between Client-Reader.

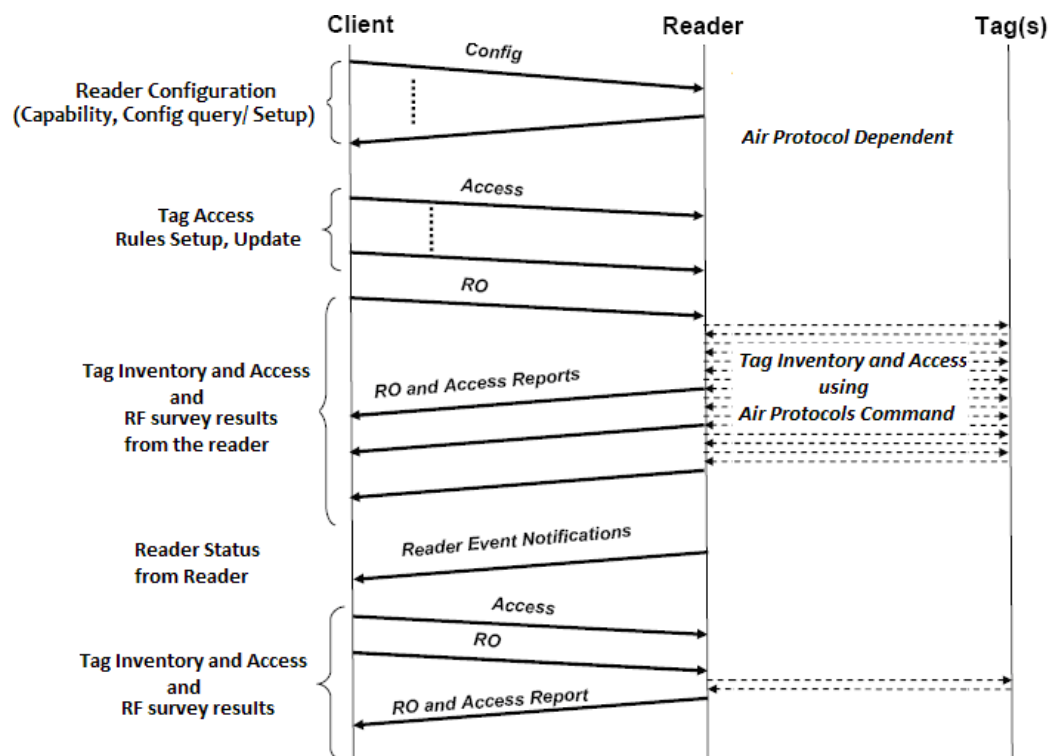


Figure 9. LLRP Timeline [14, 23]

2.5 UHF RFID Fundamentals

To understand how a system based on RFID and more concretely an RFID Reader really works, it is essential to know some theoretical concepts related to different areas. Therefore, this subsection tries to cover the topic in as much detail as possible.

First, the high level system architecture will be described. It could be said that the architecture of a UHF RFID Reader can be separated in two parts: analog and digital. The fact of doing this distinction is justified below.

The term analog refers to the part of the reader that is related to the radio transceiver, i.e. the part which makes possible the communication with tags by means of radio waves. The term digital refers to all the logic in charge of the communicating with the host and interacting with the reader chip. The theoretical concepts involved in each part of the reader are explained in more detail below.

2.5.1 Analog Part

As it was seen in section 2.2 (p. 7), the reader and tag communicate through air by means of their antennas. Depending on the type of reader antenna, the emission frequency will be higher or lower. For example, a near-field antenna which has magnetic component, is used for low frequencies and short range (e.g. LF and HF) whereas a far-field antenna which has both electric and magnetic components is used for high frequencies (bigger than 30 MHz) or long distances (e.g. UHF) [6, 84-85]. Figure 10 illustrates a comparison between the two types of antennas.

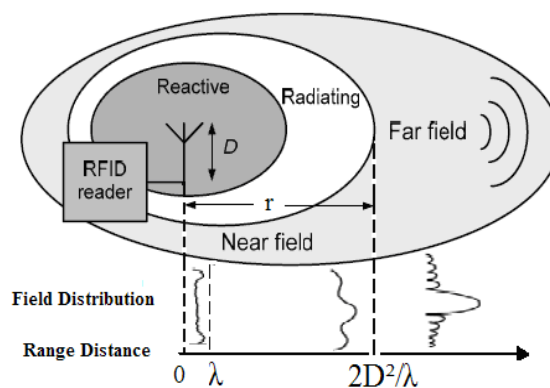


Figure 10. Near and Far Field [16]

As shown in Figure 10, it is considered a measure of far field when the difference between the transmitting and receiving antennas is larger than $r = 2D^2/\lambda$ (where, D is the maximum antenna dimension and λ the wavelength). Otherwise, it is a measure of near-field, if the distance is less than r .

In turn, the near field can be divided into two sub-regions:

- Radiating, where the angular field distribution is dependent on the distance.
- Reactive, where the energy is stored but not radiated.

Evidently, the far field covers more wavelength because the propagation distances are bigger. [16].

Although in theory the UHF belongs to the far field, there is a hybrid called Near Field UHF (NF UHF). The communication is based on the use of near E Field (electric capacitive) or near H Field (magnetic) instead of using the propagation of the electromagnetic wave such as the far field. The only difference is in the reader antenna because the reader's electronic and the tags are the same. Most UHF far field tags can operate in the near field, and its design only varies when you want to optimize it. [3, 284-285].

In this thesis project, the reader will use far field due to the bigger range. The typical UHF RFID reader antennas can be classified in two types.

a. Linearly polarized

- Energy is radiated linearly without variations in any direction (vertical or horizontal).
- Gives to rise to greatest ranges.
- Tends to generate a narrow beam.
- Requires alignment of both the transmitting and the receiving antenna.

b. Circularly polarized

- Energy is radiated circularly.
- Reduced range.
- Tends to generate a wider beam.
- The alignment of antennas is less critical
- Works much better with multi-path and scattering.

We can find in the market a wide variety of tag antenna models and shapes with different frequencies and manufacturers (see Figure 11). The antenna design will not be discussed in this report and also because it is not scope of this thesis, but there is a wide documentation about this area.

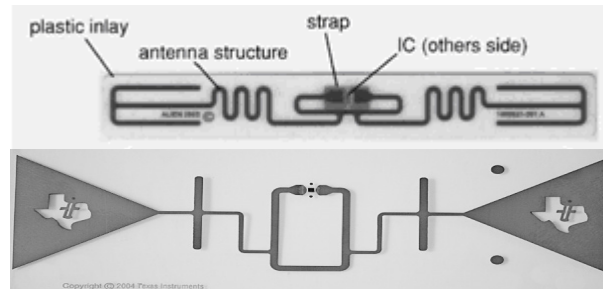


Figure 11. Typical Commercial Passive UHF Tags [3, 37; TI Model RI-UHF-00C01-03]

As the reader has to work in the far field and with passive tags, the communication with the tags is carried out by backscattering. In other words, it is a half-duplex communication. Basically, this technique is based on returning part of the signal emitted by the reader antenna, but with a lighter modification made by the tag.

As seen in subsection 2.2.2 (p. 8), a passive tag is composed of its own antenna and an application specific integrated circuit (ASIC) chip. This IC has a complex impedance (Figure 12) whose value allows the tags to send or not to send backscatter data to the reader. That is, when this chip is supplied by reader radio waves, the tag carries out a modulation of the reflected power by its antenna with the aim of sending data back. In order to do this, the tag switches its input impedance between two states (high and low) in such a way that when the impedance is low the tag can send a backscatter signal back and when it is high, the backscatter wave is negligible [15].

Since the communication is half duplex, the reader antenna can have two kinds of configuration:

- *Mono-static antenna*: The transmitter and receiver share the same antenna.
- *Bi-static antenna*: a separate antenna is used for reception and transmission.

[17, 284]

As the aim of the design is to get a low cost UHF RFID Reader, one way to reduce costs is opting for a the mono-static configuration. In a mono-static configuration a component known as circulator is used. It has three terminals, in such a way that the input signals pass from one port to the next in a rotational direction while the access in the opposite rotation is avoided (Figure 12).

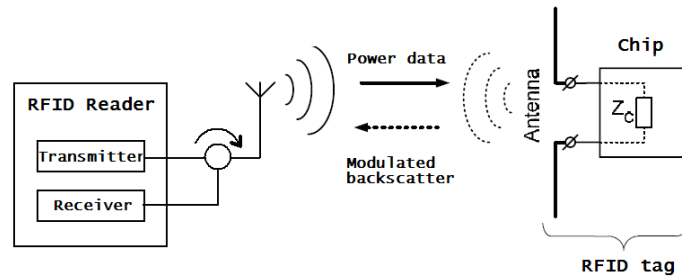


Figure 12. Communication by backscatter [15]

In theory this element prevents a transmitting signal from arriving to the receiver and vice versa, but in reality there are leaks. To solve this problem, a self-jammer canceller is needed. Although the receiver is much more sensitive if two separate antennas are used, there are still some power signals that leak directly from one to the other.

It is very important to cancel these leaks because the receiver will have to select only the tag information. Moreover the backscatter signal is much lower than the signals derived from reflection from other nearby objects (e.g. desks, tables, and people) [3, 70]. The reader is able to detect the backscatter signal thanks to different kinds of modulation and encoding (they are imposed by the EPCglobal standards) used during the data exchange between the reader and the tag. Before explaining in detail the modulation and encoding used by the tag and the reader, the basics concepts of a communication system will be reviewed.

A communication system is composed of a transmitter and a receiver which want to communicate with other via a channel or transmission medium (Figure 13). The transmitter has to adapt the signals to the medium and the receiver has to convert these signals to a form that can be used by the destination [18, 3-5].

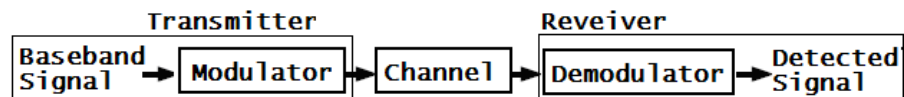


Figure 13. Basic communication system [19, 55]

As a rule, before the source can convey a 'message', the original signal has to be adapted to the channel because it is impossible to send this signal in its baseband of frequency. For this reason, the 'desired' signal is conveyed by a carrier signal with a higher frequency than that of the baseband signal. The carrier signal is a continuous wave (CW), that is, it is a periodic signal without changes in its amplitude, frequency or phase that will be modified by my signal (signal with information) [3, 58]. It is at this point when the modulation phenomenon happens.

Modulation can be defines as a process of modifying the characteristics of a carrier signal (amplitude, frequency or phase) in accordance with the variations of the desired signal to be transmitted (modulating signal); the resultant signal is known as modulated signal. The destination recovers the information by demodulating detecting the 'message' from the modulated carrier signal [18, 5-7; 19, 54-55].

Although there are two kinds of modulation (analog and digital), RFID uses digital modulation. In digital modulation the carrier signal is analogue, but the modulating signal is digital. The reason for modulating is that the antenna size which has to be the half wavelength ($\lambda / 2$) [19, 54] and the wavelength depends of the frequency as it is illustrated in formula 1.

$$\lambda = c/f \quad (1)$$

Where; $c = 3 \cdot 10^8 \text{ m} \cdot \text{s}^{-1}$ and $f = \text{frequency}$.

Consequently, although it would be conceptually possible to transmit signals directly in baseband, it is preferable to use the modulation. In this way, a better use is made of the available spectrum and bandwidth [20, 288].

As it has been explained in section 2.4.2 (page 15), EPCglobal establishes the standards used during the reader-to-tag (Forward Link) and tag-to-reader (Reverse Link) communication which are described in detail below.

□ Forward Link (Reader to Tag)

The reader sends the modified continuous wave. This modification is made by an amplitude shift key (ASK) modulation with pulse interval encoding (PIE).

Encoding is the process of converting a message into symbols. The PIE encoding is used to guarantee that the passive tag has enough power [3, 58-60].

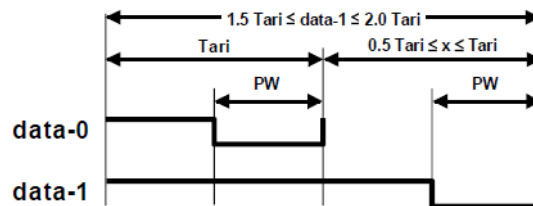


Figure 14. PIE Symbols [11, 24]

As seen in Figure 14 the data-0 has a transition to avoid the deactivation of the tag in the case of transmitting a stream of zeros. Moreover T_{ari} is the duration of data-0, and takes from $6.25\mu s$ up to $25\mu s$. The Pulse Width (PW) varies from $0.265T_{ari}$ to $0.525T_{ari}$. So the data rate is between 26.7Kbps and 128Kbps. [22, 2]

All reader-to-tag communication must start with a preamble (Figure 15).

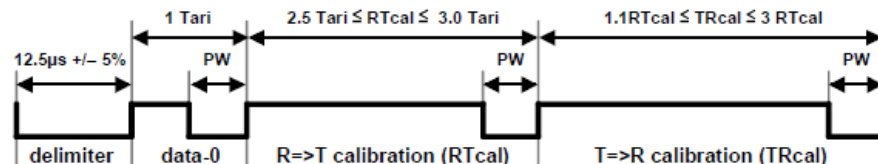


Figure 15. Preamble [11, 26]

And the subsequent commands can use a frame-synch (Figure 16).

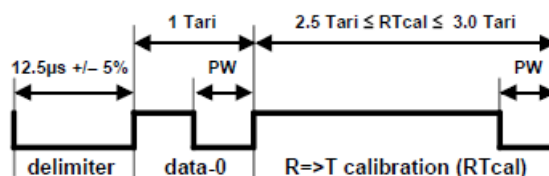


Figure 16. Frame Sync [11, 26]

As to the ASK modulation, there are three variants in the forward link:

- SSB-ASK (Single Side Band- ASK)
- DSB-ASK (Double Side Band- ASK)
- PR-ASK (Phase Reversed-ASK)

The only difference between them is that PR-ASK, is a variant of binary phase-shift keying (PSK) and similar to duobinary¹ data transmission. Otherwise, SSB-ASK remove one of the two sidebands present in an amplitude-modulated signal while DSB-ASK do not. That is, SSB-ASK or PR-ASK make a better use of the bandwidth.

An example of a transmission using is ASK modulation with PIE encoding is shown Figure 17. [3, 408]

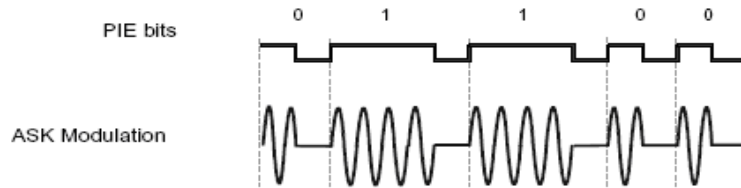


Figure 17. Example of Read-to-Tag Transmission. [22, 2]

□ Reverse Link (Tag to Reader)

The data is returned during one of the CW periods when the tag impedance modulates the backscattered signal. In this case, it is the reader that decides the encoding format which can be FM0 (default operating mode) or Miller-modulated subcarrier. The type of application determines which of the two formats is better.

The FM0 encoding is good to be used in low-noise environments (it allows obtaining high data rates). On the contrary, the Miller-subcarrier encoding has a larger number of transitions per bit (data read is slower) what makes easier to decode the signal in the presence of noise [22, 2]. Both types of encoding will be explained in more detail below.

As can be observed in Figure 18, the *FM0* encoding inverts the baseband value at the end of every bit period. Moreover, in case of a zero-bit, there is an additional transition in the middle.

¹ duobinary signal: A pseudobinary-coded signal in which a zero-bit is represented by a zero-level electric current or voltage and a one-bit is represented by a positive-level current or voltage if the quantity of "0" bits since the last "1" bit is even, and by a negative-level current or voltage if the quantity of "0" bits since the last "1" bit is odd. Duobinary signals require less bandwidth than NRZ and also permit the detection of some errors without the addition of error-checking bits. [original source: <http://www.its.bldrdoc.gov/fs-1037/dir-013/1844.htm>].

In this encoding, the data rate is equal to the backscatter link frequency (BLF) whose range varies from 40 kbps to 640 kbps.

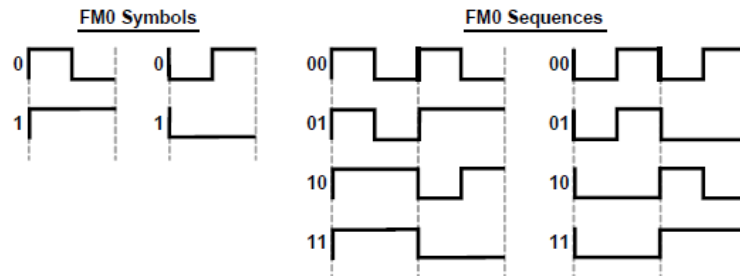


Figure 18. FM0 Symbols and Sequences [11, 30]

A FM0 message begins with one of the preambles in Figure 19.

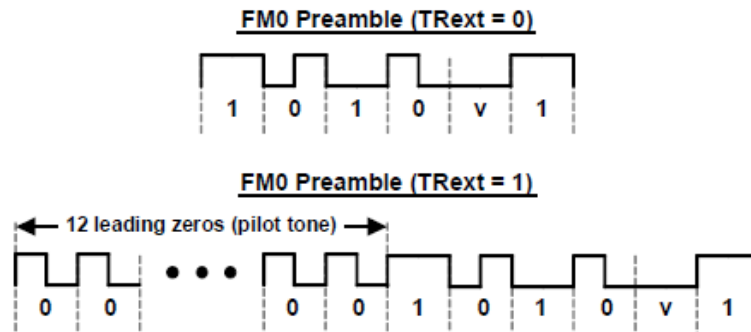


Figure 19. FM0 Preambles. [11, 31]

And it ends with one of the terminating sequences in Figure 20.

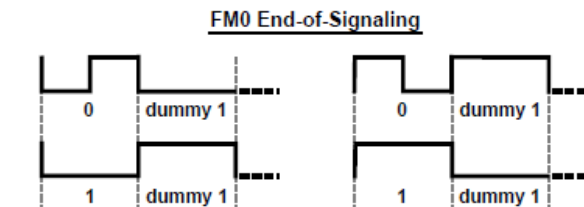


Figure 20. Termination FM0 Transitions [11, 31]

As already mentioned, in a *Miller Sub-carrier* encoding, the number of transitions per bit is larger. They occur between two zero data bits within a sequence and in the middle of data 1 bits. The resultant waveform is multiplied by the subcarrier square wave of M cycles per bit (where $M= 2, 4$ or 8). In other words, a Miller sequence can contain 2, 4 or 8 sub-carrier cycles/bit and are denoted Miller-2, Miller-4 and Miller-8 (see Figure 21). The M parameter is a parameter of Query command. Figure 21 below illustrates of a subcarrier sequence for the different M values. The range of bit rate varies from 5 to 320 Kbits/s [21, 2].

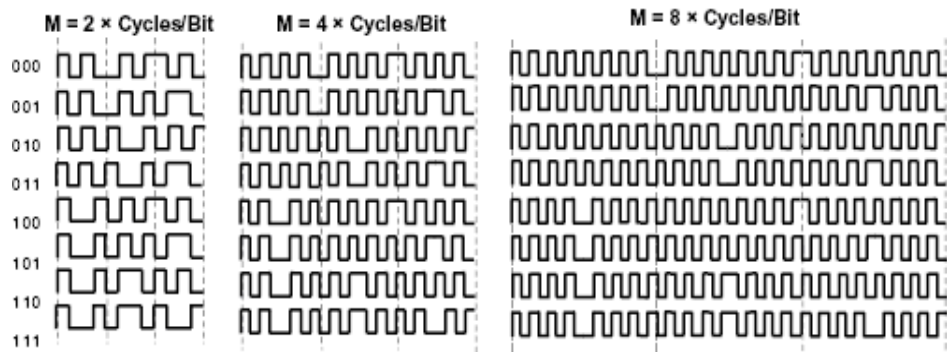


Figure 21. Subcarrier Sequences [11, 30]

There are 2 Miller Sub-carrier preambles. The Query command tells the Tag which to use. Figure 22 is an example for Miller-2 encoding.



Figure 22. Example of Preamble for Miller-2 [11, 33]

A Miller sequence terminates with a dummy. Figure 23 is an example of Miller-2 encoding (2 cycles/bit) would be:



Figure 23. Example of a Dummy for Miller-2 [11, 33]

The types of modulation used in tag-to-reader communication are presented below. The tag uses two backscattering modulation alternatives: amplitude shift keying (ASK) and phase shift keying (PSK).

In DSB-ASK modulation, the reflected power switches between two values with at a given rate whereas PSK modulation depends on the difference of phase between the reflected and sent signals at the antenna. The tag switches its input impedance between two values, to ensure the widest modulation angle between such waves [23, 49-50].

Figure 24 shows an example of Miller modulation using Miller-2 subcarrier encoding.

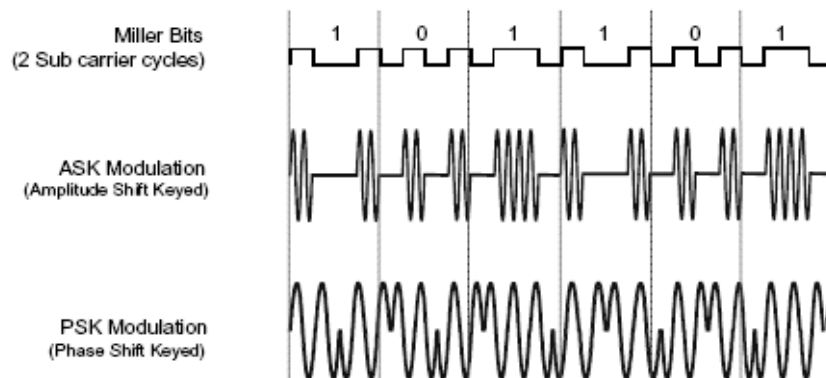


Figure 24. Example of Tag-to-Reader transmission with Miller-2. [22, 2]

2.5.2 Digital Part

This subsection presents the part of the reader that is in charge of interacting with the reader chip and the host. This task is carried out by a microcontroller. It is able to manage the Reader Chip functions (e.g reading and writing tags, setting power gain and choosing of antenna) by a Firmware that has previously been loaded in its memory. Therefore this microcontroller has to have properly configured the different interfaces that will be used for reader-microcontroller and microcontroller-Host communication.

As the interface will be explained in more detail in the next chapter, they are only briefly described here. For example, the connection between the microcontroller and the chip reader is imposed by the chip reader so it can not be selected freely.

The communication between the microcontroller and the host can be implemented by using any serial interface (UART, USB or Ethernet). The advantages that an Ethernet interface offer compared with a typical serial interface (RS232 or USB) are:

- It is not limited by the distance.
- It is not necessary to have a host exclusively for the communication. Typically, only the host which has the serial cable connected can communicate with the serial device.
- It offers bigger speeds.

3 Design of a UHF RFID Reader

Due to the huge complexity of the design, this chapter explains how to design a UHF RFID Reader from the theoretical point of view and describes the necessary components to carry out the design.

One of the most important components in this design is undoubtedly the reader chip. The reader chip that will be used is the Indy R2000. The first version of this chip (R1000) was developed by Intel, but it was sold to ImpInj.

This chip was selected because it integrates the 90% of the needed components for the hardware design of an RFID reader. In other words, it allows simplifying the design, size and cost of the reader. The elements included in the chip are: a complete UHF Gen 2 standard transmit, receive, demodulation and baseband functions. Apart from that, it includes protocol firmware, programming tools, radio drivers and schematics [24; 25]. Figure 25 shows a Top Level RF Block Diagram.

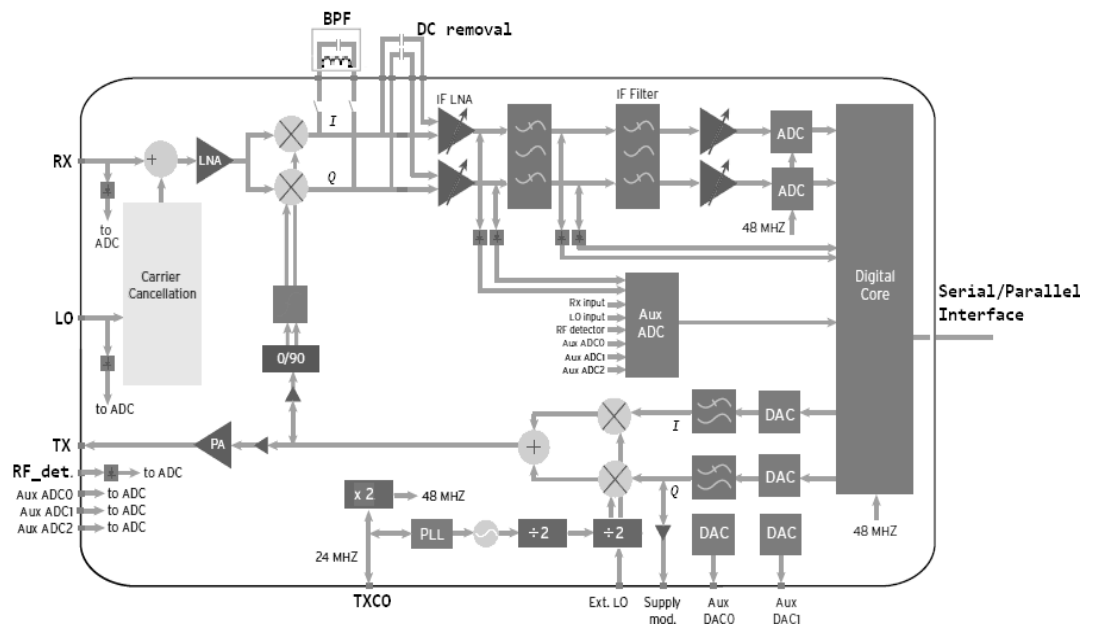


Figure 25. Indy R2000 Reader Chip [25, 2]

Figure 25 illustrates the type of receiver and transmitter that compose this IC. The receiver and transmitter are described in more detail below.

The receiver has a homodyne architecture that includes a self jammer or a carrier cancellation block. Thanks to this block is possible to avoid the saturation of receiver block and the degradation of sensitivity due to the leakages that occur when a single antenna configuration is used. Apart from this block, the receiver is contains the following elements:

- a. *Low noise amplifier (LNA)* amplifies the weak received signal without adding noise to the signal.
- b. *I/Q mixer* does the down-conversion. That is, it converts the RF signal to an Intermediate Frequency (IF) for later removing the DC part by the AC coupling capacitors (their values depend on the data rate of the tag).
- c. *IF LNA* has the same function as LNA.
- d. *IF filters* allows channel selectivity.
- e. *Analog-to-digital converter (ADC)* converts the filtered I and Q analog signals to digital.

The purpose of the transceiver is to convert the I/Q data digitals to RF. To this end, it includes the following elements:

- a. *Digital-to-analog converter (DAC)* converts the I/Q data digitals to I/Q analog signals.
- b. *Low pass filter (LPF)* eliminates the spurious signal to ensure the transmit spectrum fits into the mask.
- c. *I/Q modulator* converts the low frequency signal to the band of UHF (865MHz).
- d. *Power amplifier (PA)*, as its name indicates, is used to get more power to the output although to get the maximum power allowed (33dBm), it is necessary an external PA is needed. [21, 183-190]

Figure 26 on the next page illustrates the main characteristics of this chip in comparison with those of its predecessor R1000.

Indy Reader Chip Features		
	Indy R1000	Indy R2000
Air Interface Protocols	EPCglobal UHF Class 1 Gen 2 / ISO 18000-6C <ul style="list-style-type: none"> • DSB, SSB, and PR-ASK transmit modulation modes • Dense reader mode (DRM) 	EPCglobal UHF Class 1 Gen 2 / ISO 18000-6C <ul style="list-style-type: none"> • DSB, SSB, and PR-ASK transmit modulation modes • Dense reader mode (DRM) • Configurable for other protocols
Integrated Power Amplifier	Configurable. External power amplifier supported for high performance applications	Configurable. External power amplifier supported for high performance applications
Modem	Configurable digital baseband	Configurable digital baseband
Operating Frequencies	840–960 MHz	840–960 MHz
Package	56-pin 8 mm ² QFN	64-pin 9 mm ² QFN
Power	Advanced power management	Advanced power management
Process	0.18 μ m SiGe BiCMOS	0.18 μ m SiGe BiCMOS
RSSI	Configurable	Configurable
Sensitivity	-95 dBm (DRM) -110 dBm (LBT) -70 dBm (DRM) with a 10 dBm carrier at Rx port	-95 dBm (DRM) -110 dBm (LBT) -82 dBm (DRM) with a 10 dBm carrier at Rx port
Transmit Phase Noise (at 250 KHz offset)	-116 dBm/Hz	-126 dBm/Hz
Supported Regions	US, Canada and other regions following US FCC Part 15 regulations Europe and other regions following ETSI EN 302 208 with & without LBT regulations China, India, Japan, Korea, Malaysia, Taiwan	US, Canada and other regions following US FCC Part 15 regulations Europe and other regions following ETSI EN 302 208 with & without LBT regulations China, India, Japan, Korea, Malaysia, Taiwan

Figure 26. Comparison Between R1000 and R2000 [25, 4]

The present improvements of Indy R2000 are as follows:

- Carrier cancellation technology. This allows having bigger accuracy in the range read.
- Larger protocol configurability.
- Improved transmit phase.[24]

Figure 27 below displays a block diagram of the hypothetical connection between this chip and the rest of the external elements that compose the design of the UHF RFID Reader.

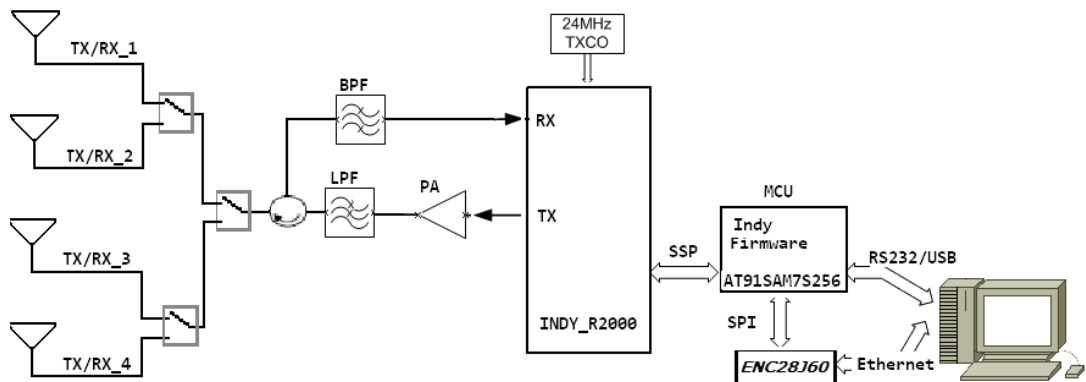


Figure 27. Block Diagram of the UHF RFID Reader.

The following sections are dedicated to analyzing some of the external components that are needed in the design.

3.1 Components of the Analog Part

The main components that compose the radio part are presented below.

Evidently, the path antennas have to be connected to SMA connectors (located in the designed reader board) by cables. Although this may seem trivial at first, the choice of the cable is important for getting the best read rate. The best choice is having a cable with low loss and a total cable length equal to a whole number of wavelengths (λ) of the frequency that is used (e.g. $\lambda \approx 0,34$ for 865 MHz). For example, IMPINJ offers RFID cables with an SMA TO R-TNC connector such as IPJ-A3002-000.

In general, the number of reader antennas in a typical commercial reader is four. As was commented in section 2.5.1 (p.28), a mono-static antenna configuration is used in the design so a circulator is needed to share the same antenna for reception and transmission. When the number of reader antennas is bigger than one, but you only are using a single circulator, then another component is needed.

As shown in Figure 27, the four antennas are joined by RFID Switches to a circulator or directional coupler. The advantages of using a directional coupler are cheaper and more compact although they have the same aim as will be explained later.

The RFID Switch is a single pole double throw (SPDT); it allows selecting the antenna by a control signal that has to be generated by the microcontroller using the firmware of the chip. In general, this device is an IC of six pins with two pins control pins, two inputs and one output. In order to make the right choice it is important to compare the electrical requirements and the losses between the various manufacturers, such as NEC, M/A-COM, Texas Instruments and Fairchild Semiconductor.

On the other hand, the circulator (Figure 28) as was explained in the subsection 2.5.1 (page 28) is a passive element of three or four ports (input, output, coupled and isolated ports). It passes the entering RF signal to any port, and then it transmits it to the next port in a rotational manner.

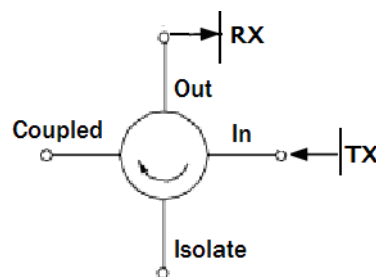


Figure 28. Circulator ports

The disadvantage of using a circulator or directional coupler is the produced losses between the ports input-output when a signal is transmitted and between the ports' isolate-coupled when a signal is received. This has to be taken into account when this component is chosen. Circulators are manufactured by for example Meca Electronics and Anaren.

An external band pass filter (BPF), is needed for the reception, which is generally a surface acoustic wave (SAW). It is really important to choose the band of frequencies where the reader will be used; the frequencies for UHF in Europe range from 865 MHz to 868 MHz. Other important factors in the selection of the filter are central frequency, maximum insertion attenuation and attenuation in the chosen band. Moreover, there are SAW Filters already matched to 50Ω. Examples of manufacturers are RF Monolithics, Filtronetics and Epcos.

An external power amplifier (PA) and a low pass filter (LPF) are used for the transmission. The first of them has to be matched to the input and output impedances of 50Ω by an input matching circuit. The PA has a control pin to transmit or not to transmit the maximum power (33dBm or 2W) which has to be controlled by the firmware. The main factors in the selection are a high IIP3/OIP3 (Third-order intercept point) and the gain. The M/A Com manufacturer also supplies this kind of component. A LPF is used to eliminate the spurious signal that the PA could have introduced. The typical insertion loss is 0.5 dB, and another important factor is the attenuation to $2xF_0$ and $3xF_0$ (where F_0 is the central Frequency). Some manufacturers of this filter are Murata and ACX.

Last element is the Temperature Compensated Crystal Oscillator (TCXO) used by the reader chip. It is used as a clock for the internal digital blocks (24 MHz for sigma-delta DAC's and 48 MHz for sigma-delta ADC's). It has to have low phase noise because has to be very accurate. Some oscillator manufacturers are Iplex Technologies and Taitien.

3.2 Components of the Digital Part

The choice of the microcontroller is determined by the manufacturer because the Firmware binary of the Indy R2000 reader chip can only run under the microcontroller AT91SAM7S256. This firmware has the function of configuring the different parameters involved in the reader chip (e.g. choice of the antenna, frequency and output power). In the case another microcontroller is desired, the only option is to purchase the source code license and then to try to port that code over to this microcontroller.

The AT91SAM7S256 is a Flash microcontroller based on the 32-bit ARM7TDMI RISC processor. It has an Internal High-speed Flash of 256K bytes and 64K bytes of SRAM. The embedded Flash can be programmed by the integrated proprietary SAM-BA Boot Assistant. The peripheral set available contains USB 2.0 Full Speed Device Port, two USART, SPI, SSC, TWI and an 8-channel 10-bit ADC. It can work up to 55 MHz at 1.65v and 85 °C (Worst Case Conditions). [26]

This microcontroller is a key element in the 'chain' of UHF RFID Reader because apart from managing the configuration of the reader chip has to maintain communication with the host.

The interaction with the host is carried out by means a serial interface since this offers larger advantages compared to a parallel interface, such as less use of pins and larger speeds. This is why the typical interfaces in commercial readers are RS232, USB and Ethernet. Chapter four describes this issue in more detail.

3.3 Power Supply

This module has to supply the power for the whole system. The voltage specifics used in the different parts of the design are as follows:

- Indy R200: 1.8 and 3.3V @ 1100 mW.
- ENC28J6: 3.3V @ 180 mA. [26]
- AT91SAM7s246: it has six types of power supply pins.

V_{DDIN} : It powers the integrate voltage regulator and the ADC; 3.3V

V_{DDOUT} : It is the output of the 1.8V integrate voltage regulator.

V_{DDIO} : It powers the I/O lines and the USB transceivers; Dual voltage range is supported; 3.3V or 1.8V.

$V_{DDFLASH}$: It powers a part of the Flash and is required for the Flash to operate correctly; 3.3V @ 10mA.

V_{DDCORE} : It powers the logic of the device; 1.8V @ 50mA. It is connected to the V_{DDOUT} pin with a decoupling capacitor.

V_{DDPLL} : It powers the oscillator and the PLL and is connected directly to the V_{DDOUT} pin.

To decrease the current consumption of the microcontroller when neither the voltage regulator and nor the ADC are used, V_{DDIN} , ADVREF, AD4, AD5, AD6 and AD7 should be connected to GND and V_{DDOUT} should be disconnected. [26]

- External PA: It depends on the manufacturer (e.g. 2.5V and 7.5V).

There are many ways to get the different voltages required. The easiest way is to use a constant unique voltage (obtained by an AC/DC transformer) and a lineal regulator for the rest of voltages.

A linear regulator regulates the output voltage or current by dissipating of the excess energy in form of heat; this makes the calculation of an adequate heat sink essential. In the market, there are many linear regulators for different voltages, for example: LM117-3.3 (3,3v), 7805 (5v) and 7812 (12v).

In spite of the fact that the use of Linear Power Supplies (as the mentioned above) is very popular and easy and cheap, there is another method for supplying circuits which uses the switching.

Switching regulators are more efficient and do not need an AC/DC transformer, so their size is much smaller. They can reduce, increase or invert From a DC input voltage (e.g. a battery). Their design is much more difficult and they are more vulnerable to noise, being more suitable for high power applications. To learn more about these converters, consult the book “Switching Power Supplies Design” written by Presman, Billing and Morey [27].

The selection of the type of power supply is determined by power losses. Generally, if the power losses are less than 0.5W, linear regulation is a good solution. When the power losses are of the order of some Watts, it is better to use switching regulation. This calculus can be done with formulae 2.

$$P_{WASTE} = (V_{INPUT} - V_{OUTPUT}) * I_{LOAD} \quad (2)$$

In brief, although the design of the power module seems ‘trivial’ at first sight, the fact is that for the correct design and choice of its components, different factors such as the power required for the system, the space required, noise and efficiency must to be taken into account.

4 Reader to host interface

4.1 Introduction

As it was noted in chapter 3, it would be unattainable for one single person to implement the whole system in a short time due to the complexity of its design. Therefore, the thesis now concentrates on the study of the part of the reader that is linked with the host.

Chapter 3 also explained that the microcontroller, apart from supporting the firmware of the reader, has to have its different interfaces configured to establish communication with both the reader chip and the host. In the case of the reader chip, the connection with the microcontroller is determined by the reader's own interface; nothing happens with the host.

Nowadays the typical way of connecting a host and a microcontroller is by serial interface. The most common option is the UART interface through the serial port of the computer, e.g. COM1, but some modern computers do not have this kind of port, so the USB interface is usually also used. Although they are widely used and easier to implement than other options, serial interfaces have the disadvantage of depending on wire length or being limited by the data speed.

In contrast, the use of an Ethernet interface has more advantages such as eliminating the distance limitations as well as allowing any host on the same network to control the device to its control. Besides, the majority of the commercial readers have this kind interface and some of them support power over Ethernet (PoE). For all these reasons, it is worth implementing this interface in the design.

In the next sections, the ins and outs of this interface are dealt with. Since it was not possible to get a reader chip for this project, the practical part of this thesis makes only reference to the communication between the host and the Ethernet controller in chapter 5.

4.2 Ethernet Interface

At present, we can find in the market a variety of solutions to implement the Ethernet interface. The easiest solution that you can find is some kind of RS-232-to-Ethernet or USB-to-Ethernet converter. Although these solutions are good and easier to implement, they are not practical if the idea is to commercialize the reader.

The other solution is looking for some Ethernet controllers. They are much cheaper than the serial-Ethernet converters mentioned before, but they have more complicated implementation. Some Ethernet controllers are presented bellow.

In spite of the fact that there are Ethernet microcontrollers such as the AX11015 (Asix), the PIC18F97J60 (Microchip), and the AT91SAM7X Series (Atmel), they are in my opinion, usually more expensive than a single Ethernet controller and its control and configuration is more complex because there is already another microcontroller in the system.

The RTL8029AS (Realtek), the CS8900A (Cirrus Logic), and the 82559 (Intel) are some examples of Ethernet controllers. The problem is that the majority these controllers do not meet the specifications desired because their main characteristics are that they use 100 or even more pins, they do not have the media access controller (MAC) layer and/or they use many pins for the connection (parallel interface).

Another controller, the CP220x (Silicon Labs), contains the MAC/PHY layers and has a smaller number of pins (28-pin QFN or 48-pin TQFP package), but it was discarded because it uses the parallel interface for the connection with a microcontroller.

Apart from taking into account easy implementation (MAC/PHY layers already implemented), it is necessary to choose a device with a low cost per chip because the goal of this design is to get a low cost reader.

Finally, the chip chosen was the ECN28J60 of Microchip [28] because apart from using only 28 pins and having integrated the MAC and 10Base-T Physical Layer, its cost is really low (around 2 €). Moreover, it is the only one in the market that can be configured by serial peripheral interface (SPI) whose operation is explained in section 4.5. Another advantage is that there are available many packages for the chip (SOIC - SPDIP - SSOP – QFN). The drawback is that upon using a PIC microcontroller, it is necessary to program your own TCP/IP stack software due to license problems (see section 4.3 below).

In the later sections, the use of this solution is addressed. First, the possible problems of not using a microchip device with the Ethernet controller are discussed. Next, the material used in the development of this part of the thesis is presented. Then, there is a brief introduction of the SPI interface, and after that, the memory and the SPI instruction set of the Ethernet controller are described. At the end of the chapter, the TCP/IP stack is discussed.

4.3 Problems in the Implementation

The implementation of the Ethernet interface without a Microchip microcontroller generates a software problem if you do not have so much experience in coding. The root of this problem is that Microchip has coded a complete TCP/IP stack with its different protocols, but it can not be used with another microcontroller due to software license agreement. Therefore, only the software driver source files ENC28J60.c, ENC28J60.h, can be modified. Consequently, it is necessary to program your own TCP/IP stack.

The reasons for not using a PIC microcontroller together with the Ethernet controller are that the RFID reader does not need to have a complex TCP/IP stack (a web server for example) implemented, and that a low cost design is pursued. Anyway, on the Internet there are some TCP/IP software stacks whose code could be reused already implemented with free license (e.g. uIP stack), but they were not used in this thesis project because a new one was created for the ping application.

Moreover, other problem to consider for whichever Ethernet Interface used is that a unique MAC address is needed. An Ethernet MAC address is a number of 48 bits (e.g. 00:04:A3 :--:--:--). The first 24 bits of a MAC address are the organizationally unique identifier (OUI), and the others 24 bits are the EI (Extension Identifier).

Any company can purchase an OUI number directly from IEEE at <http://standards.ieee.org/regauth/oui/index.shtml>. It costs around of 1,200€ because you can have 2^{24} different addresses. A cheaper option is to request for an individual address block (IAB). An IAB only costs 400€, but you will only get 4,096 unique addresses to use. But both options are really expensive if your number of manufactured interfaces is low.

In my design, I am going to use a Microchip Ethernet controller, and some ECN28J60 chips have a MAC address pre-loaded. Anyway, Microchip has MAC address chips. These chips can be accessed by SPI or I²C bus and the cost per chip is less than 1€ and you do not need a volume restriction.

Another future problem is that when the complete system is operative (reader chip, microcontroller and Ethernet controller), there can be a multitasking problem. This problem can be present because both the reader chip and the Ethernet controller are managed by an SPI interface, so the time control has to be divided. If a multitasking problem happens, it will be necessary to implement an RTOS (real time operative system).

On the internet you can find free RTOS for ATM7 with TCP/IP stack implemented, such as FreeRTOS, but you can also purchase RTOS programmed by other companies (e.g. Micro Digital).

4.4 Materials Used

As it was previously explained, the necessary elements to carry out the Ethernet interface implementation are: the AT91SAM7s256 microcontroller and the ECN28J60 Ethernet controller. Although the hardware for this part there could have been designed, the evaluation kits of the devices were used.

This decision was taken because the evaluation kit for the microcontroller had already been requested for. Moreover, once the software issue has been solved, it is easier to make your own prototype in a printed circuit board (PCB), in such a way that no mistake would be due to failure in the hardware design. The materials used for the development of the practical part are presented below.

4.4.1 AT91SAM7s-EK Evaluation Kit

Although this evaluation board allows the evaluation and the development of applications that run on AT91SAM7Sxx devices, the fact is that this board comes fitted with an AT91SAM7s56 that it is precisely the microcontroller used in the reader design. Figure 29 shows the evaluation board.

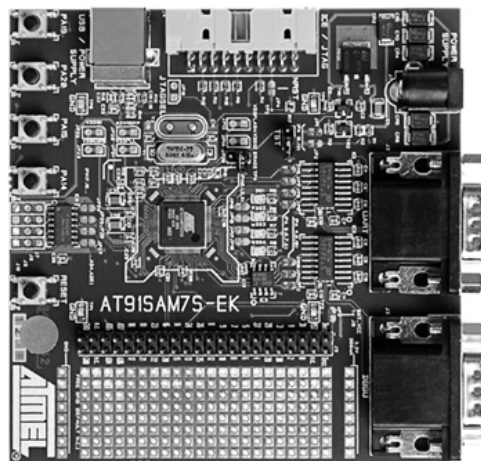


Figure 29. AT91SAM7s-EK Evaluation Board [26]

As the main characteristics of the AT91SAM7s256 microcontroller were described in section 3.2, the aspects of the evaluation kit are only summarized on next page.

One advantage of this board is that it can be supplied by an external power supply or by the USB port. The main characteristics of this board are presented in the following list:

- USB device port interface
- Two serial communication ports (USART and DEBUG)
- JTAG/ICE debug interface
- Four buffered analog inputs.
- Four general-purpose Led's and push-buttons
- Expansion connector with total access to the 32 I/O pins of the microcontroller
- Prototyping area

(For more information about this evaluation kit, visit the website of Atmel).

4.4.2 PICtail Ethernet Board (AC164121)

Despite the fact that this daughter board has been designed by Microchip to be plugged into others boards (PICDEM™), it is possible to use it joined to the AT91SAM7s-EK for the evaluation of the ECN28J60 Ethernet controller because as is illustrated in Figure 30, there is an accessible connector where the necessary wires for the SPI interface can be plugged.

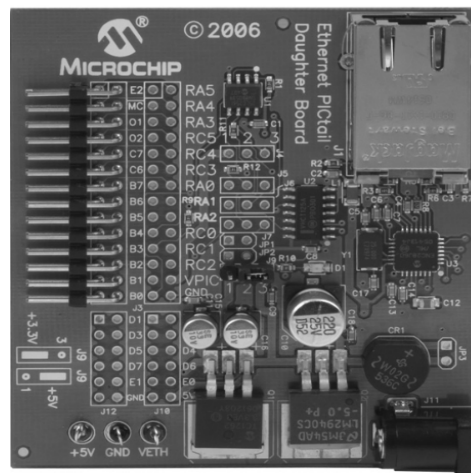


Figure 30. Daughter Board, Ethernet PICtail [27]

The decision to choose an evaluation kit for the Ethernet controller instead of designing a simple hardware was made to save time and cost. All hardware design implies doing measurements, choosing components, drawing schematic representation of the structure, making the board and soldering components.

Also the price of this board is low (25 €) and it is simple compared to other evaluation kits that are more complex with PIC microcontrollers or LCD's, for example.

The most important characteristics that this Ethernet board has are

- Integrated 28-pin ENC28J60 Ethernet controller,
- 10BASE-T Ethernet,
- Magnetic RJ-45 Ethernet connector with link and active led's,
- 256 Kbits SPI EEPROM (25LC256) for storing web pages and configurations,
- Media Access Control (MAC) Address, and
- Dedicated power supply (5V DC)

4.4.3 IAR J-Link

A JTAG debugger for ARM is connected via USB to any PC host running Windows. IAR J-Link is used together with the IAR Embedded Workbench tool. [28]



Figure 31. IAR Jlink [28]

Some of its key features are as follows:

- Any ARM7/ARM9/ARM11/Cortex-M0/M1/M3 core supported, including both JTAG and SWD.
- Max. JTAG speed 12 MHz
- No power supply required, powered through USB
- Automatic core recognition
- Auto speed recognition
- All JTAG signals can be monitored, and the target voltage can be measured
- Support for multiple devices on scan chain

(For more information about this Jlink debugger, visit the website of IAR).

4.4.4 IAR Embedded Workbench

This tool is an Integrated Development Environment (IDE) that contains a complete set of development tools for building and debugging embedded applications using assembler, C and C++. This tool allows creating your own source files and projects as well as to compile and debug your applications for later to be executed in the simulator or in hardware. The ARM core version used joined to the IAR J-Link can be downloaded on the website of IAR.

Key components of the software tool are as follows:

- Integrated development environment with project management tools and editor.
- Highly optimizing C and C++ compiler for ARM.
- Run-time libraries including source code.
- Relocating ARM assembler.
- Linker and librarian tools.
- C-SPY® debugger with ARM simulator, JTAG support and support for RTOS-aware debugging on hardware.
- Code templates for commonly used code construct. [28]

4.5 Serial Peripheral Interface (SPI)

As the SPI interface is the method of connection between the microcontroller and the Ethernet Controller, an overview of the SPI interface and the operation mode used for the configuring of the interface is given below.

Note: Part of the following information is based on the datasheets 26 and 28.

4.5.1 SPI Overview

SPI is a method of synchronous serial communication based on four wires (two data lines and two control lines), for linking with external devices in Master or Slave Mode. In this project, the master is the microcontroller and the slaves are the Ethernet controller and the chip reader. The master is in charge of driving the chip select line and the serial clock to the slaves.

All devices that support SPI devices have these same pins although sometimes the names of the signals change, as it happened in this case. Microchip defines the SPI lines as SI, SO, CS and SCK and Atmel defines its SPI lines as MOSI, MISO, NPCS and SPCK. Later on is tried this issue.

As the AT91SAM7s256 acts like master and it has to be programmed. The four SPI lines used are described below.

- *Master Out Slave In (MOSI)*: This data line is used for the transmission from the master to the slave.
- *Master In Slave Out (MISO)*: This data line is used for the transmission from the slave to the master. There may be no more than one slave transmitting at the same time.
- *Serial Clock (SPCK)*: This control line is supplied by the master to regulate the data flow, in case of using more than one slave with different clocks, the master must be reconfigured.
- *Slave Select (NPCSn)²*: is the control line that allows turn on or off the slaves by software. AT91SAM7s256 has four lines of selection (NPCS0- NPCS3), but that with external logic is able to select up to 15 external devices.

To initiate the data transfer, the Master has to program the clock with a frequency less than or equal to the maximum frequency supported by the slave device (up to 20 MHz for ECN28J60). This clock is only running during the time of transmission.

When there is no more data to transmit, the master stops toggling its clock and later disables the slave. Since the master can not select more than one slave at a time, all deactivated slaves must ignore the SPCK and MOSI signals, and must not drive the MISO signal.

² AT91SAM7s256 sometimes uses the letter 'N' at the beginning of its acronyms to indicate that the activation is low level.

In addition to setting the clock frequency, the master must also configure its polarity (CPOL) and phase (NCPHA) for the data transfer. AT91SAM7s256 supports the four possible combinations, but the master and slave must have the same polarity and phase to be able to communicate.

The ECN28J60 uses the mode 0 and Microchip defines this mode as CPOL=0 and CKE=1 so that the data of MOSI are captured on the rising edge and the data in MISO are driven out on the falling edge.

The Microchip CKE bit has the same functionality that CHPA, but is defined as inverted (NCPHA), that is, for the configuration of the master the values used are CPOL=0 and NCPHA=1.

The SPI is a full duplex communication, so when the master wants to read a byte sent by the slave on the MISO line, it has to write a ‘dummy’ byte on the MOSI line. A dummy byte is a byte without ‘value’ for the slave, usually is ‘zeros frame’ or ‘ones frame’ that it depends on the idle of the clock. In ECN28J60, the Idle is a low state, that is, the dummy byte is a padding of zeros.

The ECN28J60 Ethernet controller keeps the MISO line in High-Impedance State while the master (AT91SAM7s256) sends the commands and data configuring it through MOSI line. Similarly, when data is driven out by the ECN28J60 on the MISO line, the content of MOSI is not relevant.

The data exchange is done by the connection of shift registers. In general, in SPI transmission involves two shift registers with the same size (one for the master and other for the slave) and that are connected in a ring so that the stored data in some of the registers are shifted out by the most significant bit (MSB) first and the place is occupied by a new least significant bit.

When that register has shifted out all its bits, the master and slave will have exchanged register values. Then each device decides what to do with the data, for example writing it to memory. The AT91SAM7s256 has a single shift register and two holding registers (Transmit Data Register and Receive Data Register) as shown in Figure 32.

The behavior in the master mode can be summarized in the following actions: after enabling the SPI, the transmission starts when data is written to the SPI_TDR (Transmit Data Register). Automatically, this data is loaded into the shift register and transferred to MOSI line, and at the same time, the information of the slave is sent on the MISO line. The end of the transfer is indicated by the TXEMPTY flag in the Status Register (SPI_SR). As the shift registers of the Master and Slave are connected in a ring, data is received in the SPI_RDR (Receive Data Register) after a transfer, which is indicated by the RDRF flag in the SPI_SR.

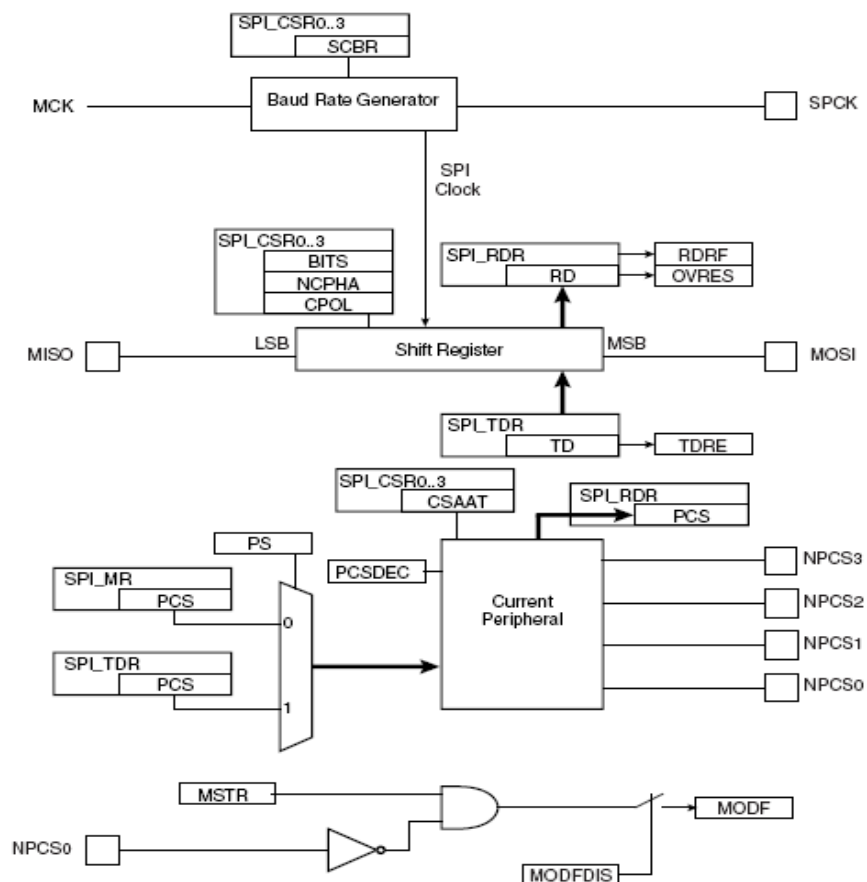


Figure 32. Master Mode Block Diagram [26, 269]

Although often, a SPI transmission uses blocks of 8 bits, other sizes are also common. However, this is not a problem for the AT91SAM7s256 microcontroller since it allows selecting the data length from 8 to 16 bits by configuring a suitable SPI Chip Select Register (SPI_CSR).

As we will see later, the microcontroller carries out the configuration of the Ethernet controller using an SPI Instruction Set. The commands defined in them allow accessing the memory of the Ethernet controller, but before seeing these commands the memory organization of the Ethernet controller is presented.

4.5.2 ECN28J60 Memory

All memory of the Ethernet Controller is implemented as a Static Random Access Memory (SRAM). The ECN28J60 has three types of memory:

- Control registers
This memory contains the register used for the initialization and configuration of the PHY and MAC levels.
- PHY registers
This memory contains the registers which take care of physical layer.
- Ethernet buffer
This memory stores the data which have been received or they are transmitted by the Ethernet cable.

Each memory type has its purpose, so they are explained in more detail below.

a. Control Registers' Memory

This memory is used for the configuration, control and status retrieval of the ECN28J60. It is split into four banks of 32 bytes addressed with 5 bits from 00h to 1Fh (Table 2) so it is necessary to change the bank to access a certain register.

As it can be observed in Table 2 on next page, the last five locations (1Bh to 1Fh) contain the same registers so they allow the control and monitoring of the state of the device without changing of bank. There are some locations unimplemented so the attempts of writing are ignored while reading return '0'. The 1Ah address is reserved address and it must not be written.

Table 2. ECN28J60 Control Register Map [27, 12]

Address	Name	Address	Name	Address	Name	Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR5
01h	ERDPTH	01h	EHT1	01h	Reserved	01h	MAADR6
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR4
04h	ETXSTL	04h	EHT4	04h	MABBIPG	04h	MAADR1
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR2
06h	ETXNDL	06h	EHT6	06h	MAIPGL	06h	EBSTSD
07h	ETXNDH	07h	EHT7	07h	MAIPGH	07h	EBSTCON
08h	ERXSTL	08h	EPMM0	08h	MACLCON1	08h	EBSTCSL
09h	ERXSTH	09h	EPMM1	09h	MACLCON2	09h	EBSTCSH
0Ah	ERXNDL	0Ah	EPMM2	0Ah	MAMXFLL	0Ah	MISTAT
0Bh	ERXNDH	0Bh	EPMM3	0Bh	MAMXFLH	0Bh	—
0Ch	ERXRDPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDPTH	0Dh	EPMM5	0Dh	Reserved	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMCSL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMCSH	11h	Reserved	11h	—
12h	EDMANDL	12h	—	12h	MICMD	12h	EREVID
13h	EDMANDH	13h	—	13h	—	13h	—
14h	EDMADSTL	14h	EPMOL	14h	MIREGADR	14h	—
15h	EDMADSTH	15h	EPMOH	15h	Reserved	15h	ECOCON
16h	EDMACSL	16h	Reserved	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	Reserved	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKTCNT	19h	MIRDH	19h	EPAUSH
1Ah	Reserved	1Ah	Reserved	1Ah	Reserved	1Ah	Reserved
1Bh	EIE	1Bh	EIE	1Bh	EIE	1Bh	EIE
1Ch	EIR	1Ch	EIR	1Ch	EIR	1Ch	EIR
1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT
1Eh	ECON2	1Eh	ECON2	1Eh	ECON2	1Eh	ECON2
1Fh	ECON1	1Fh	ECON1	1Fh	ECON1	1Fh	ECON1

The names of the registers start with different letters indicating the group they belong to; ‘E’ (ETH), ‘MA’ (MAC) and ‘MI’ (MII). It is possible to directly access these registers by SPI interface so that writing to them configures the different characteristics of the ECN28J60, such as the MAC address, duplex mode and the size of the receive and transmit Ethernet buffer, while reading them allows obtaining information about the right operation of the device such as read of flags.

b. PHY Registers

They are used for configuration and control of the PHY module state. Although they have a total of 32 PHY addresses (00h- 1Fh), only the first nine are implemented (00h -14h) while the rest are ignored (Table 3).

Table 3 PHY Register Summary [27, 20]

Addr	Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00h	PHCON1	PRST	PLOOPBK	—	—	PPWRSV	r	—	PDPXMD ⁽¹⁾	r	—	—	—	—	—	—	—
01h	PHSTAT1	—	—	—	PFDPX	PHDPX	—	—	—	—	—	—	—	—	LLSTAT	JBSTAT	—
02h	PHID1	PHY Identifier (OUI3:OUI18) = 0083h															
03h	PHID2	PHY Identifier (OUI9:OUI24) = 000101						PHY P/N (PPN5:PPN0) = 00h						PHY Revision (PREV3:PREV0) = 00h			
10h	PHCON2	—	FRCLNK	TXDIS	r	r	JABBER	r	HDLDIS	r	r	r	r	r	r	r	r
11h	PHSTAT2	—	—	TXSTAT	RXSTAT	COLSTAT	LSTAT	DPXSTAT ⁽¹⁾	—	—	—	PLRITY	—	—	—	—	—
12h	PHIE	r	r	r	r	r	r	r	r	r	r	r	PLNKIE	r	r	PGEIE	r
13h	PHIR	r	r	r	r	r	r	r	r	r	r	r	PLNKIF	r	PGIF	r	r
14h	PHLCON	r	r	r	r	LACFG3	LACFG2	LACFG1	LACFG0	LBCFG3	LBCFG2	LBCFG1	LBCFG0	LFRQ1	LFRQ0	STRCH	r

Legend: x = unknown, u = unchanged, — = unimplemented, q = value depends on condition, r = reserved, do not modify.

This memory cannot be directly accessed by the SPI interface, but it is only accessible through media independent interface management (MIIM) implemented in the MAC, also referred to as the MII registers.

As we have just to see, these registers are located in the control register. Table 4³ shows the MII registers involved in the read and write of the PHY layer. The steps necessary to carry out these operations are explained later.

Table 4. MII Register used for the access to PHY registers

Register name	Length (bits)	Contained MSB ----- LSB	Bank Location
MIREGADR	5	Address[4:0]	Bank 2 (14h)
MICMD	2	MIISCAN; MIIRD	Bank 2 (12h)
MISTAT	4	reserve; NVALID; SCAN; BUSY	Bank 3 (0Ah)
MIRD_L	8	Data	Bank 2 (18h)
MIRD_H	8	Data	Bank 2 (19h)
MIWRL	8	Data	Bank 2 (16h)
MIWRH	8	Data	Bank 2 (17h)

³ All the registers managed by the ECN28J60 have 8 bits, but not all of them are implemented and are read as '0'.

c. Ethernet Buffer

The buffer length is 8 Kbytes (0000h-1FFFh) although it is divided into two separate areas because it contains transmit and receive memory used by the Ethernet controller. This buffer can be accessed by the SPI interface and some ETH registers located into the control memory.

For example, the receive buffer length is programmed by the ERXST and ERXND pointers so that any space outside of the receive buffer pointers is considered as transmit buffer. As the pointers managed by the buffer use 13 bits and the registers of the control memory are 8 bits wide, two registers are needed (one for the low part and the other for the high part). Figure 33 shows some of the pointers used by the buffer.

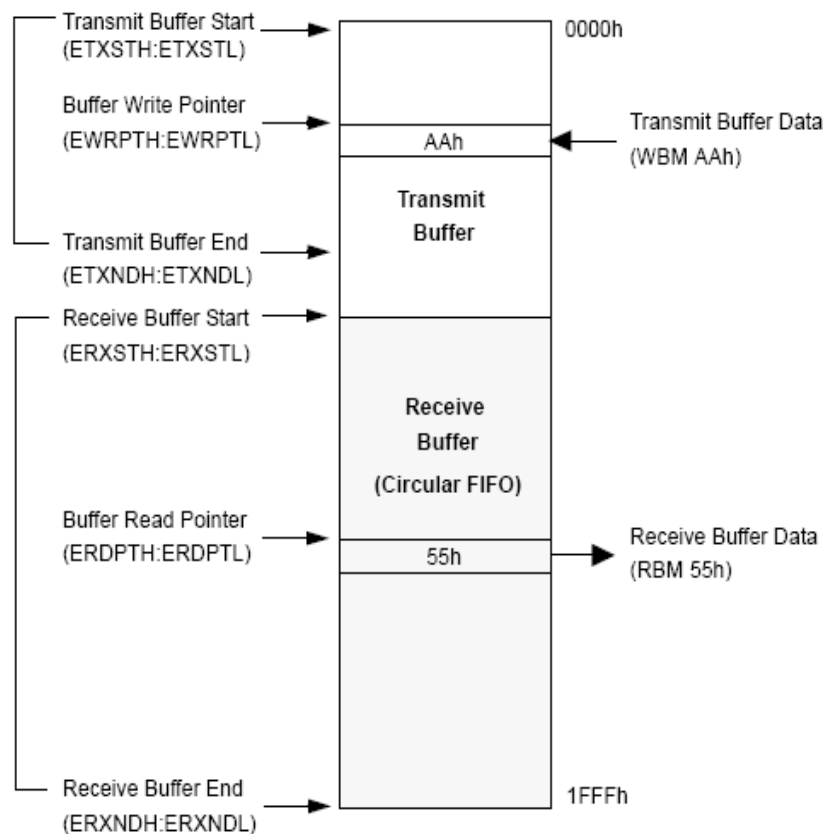


Figure 33. Ethernet Buffer Organization [27, 18]

As it will be explained in the next subsection, the read buffer memory (RBM) and the write buffer memory (WRM) SPI commands, together with the Buffer Pointers (ERDPT and EWRPT) are used for reading and writing into the buffer.

4.5.3 ECN28J60 SPI Instruction Set

As it was mentioned in the previous section, using the SPI Instruction Set (Table 5) defined for the ECN28J60 is necessary to access the different types of memory. Therefore the microcontroller has to manage these commands to be able to configure the Ethernet controller properly.

Table 5. SPI Instruction Set for the ENC28J60 [26, 26]

Instruction Name and Mnemonic	Byte 0		Byte 1 and Following
	Opcode	Argument	Data
Read Control Register (RCR)	0 0 0	a a a a a	N/A
Read Buffer Memory (RBM)	0 0 1	1 1 0 1 0	N/A
Write Control Register (WCR)	0 1 0	a a a a a	d d d d d
Write Buffer Memory (WBM)	0 1 1	1 1 0 1 0	d d d d d
Bit Field Set (BFS)	1 0 0	a a a a a	d d d d d
Bit Field Clear (BFC)	1 0 1	a a a a a	d d d d d
System Reset Command (Soft Reset) (SRC)	1 1 1	1 1 1 1 1	N/A

Legend: **a** = control register address, **d** = data payload

As it can be observed in Table 5, the SPI Instruction Set is only composed by seven instructions which are described one-by-one below.

1. Read Control Register (RCR) Command

This command allows reading any of the ETH, MAC and MII registers in any order. Once the chip select is enabled, the microcontroller has to send on the MOSI line the first byte composed by the RCR opcode (000) followed by a 5 bit address that identifies to the register of the current bank which I want to read. The only difference between ETH or MAC/MII registers is the number of dummy zeros that the microcontroller has to send after the first byte.

As shown in Figure 34, for the MAC or MII registers it is necessary to send two dummy bytes before getting the content of the register on the MISO line. Once I have the content of the register and chip select is disabled, the operation ends.

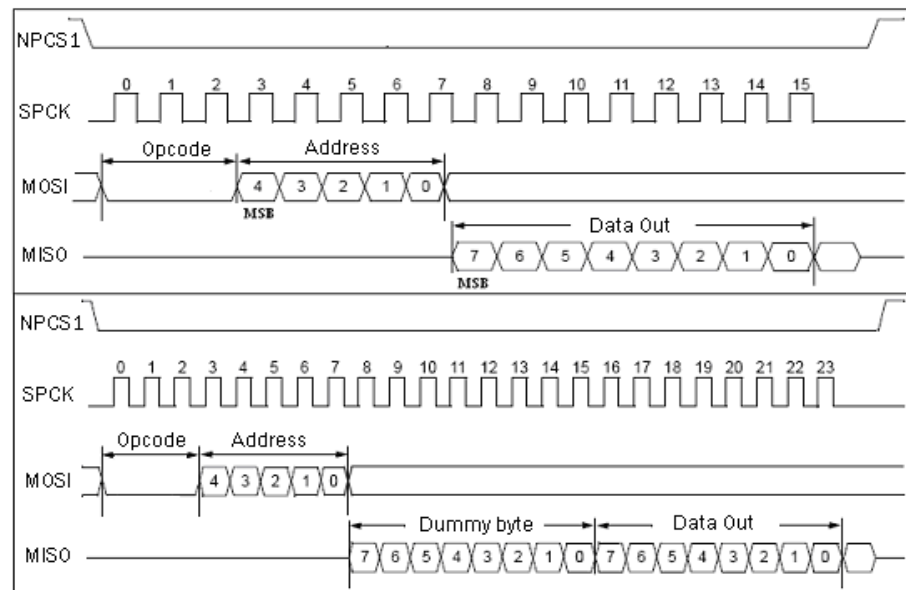


Figure 34. RCR Command Sequence for ETH, MAC and MII Registers [26, 27]

As it was said previously, when I want to read the content of PHY registers, I need to use the MII registers, but moreover the following steps are necessary.

- a. The 5 bit address of the desired PHY register is written in MIREGADR
- b. The MIIRD bit of the MICMD register is set and the read operation starts (Flag MISTAT.BUSY=1).
- c. The time that MAC takes to obtain the content of the PHY register selected is 10.24 μ s; during this time the BUSY flag can poll to know if the operation has been completed.
- d. The MIIRD bit is clear.
- e. Finally, the content of PHY register is read from MIRDH and MIRDH.

2. Read Buffer Memory (RBM) Command

This command allows the reading of the Ethernet buffer and more concretely the receive buffer memory although as it will be seen in chapter 5, it can be used for knowing the content of the transmit buffer during debugging.

There is a bit in the ECON2 register called AUTOINC that, when set, allows automatically the increment of the ERDPT Pointer to the next address after reading the last bit of data. The receiver buffer is a circular FIFO; if the address pointed by ERDPT is different to that pointed by ERXND, then ERDPT is incremented by one unity. Otherwise ERDPT points to the beginning of the buffer (ERXST). In the case that ERXND does not point to 1FFFh, but the data is read of this address, the ERDPT will point to 0000h when it was incremented.

As is shown in Figure 35, once that chip select is enabled; the microcontroller sends the first byte composed by the RBM and a 5 bit constant (1Ah). After sending this first byte (0x3A), the data storage at the address pointed to by ERDPT is shifted out on the MISO line to be read. The operation ends when the microcontroller obtains the data and the chip select is disabled.

In the case of the master decides to keep the clock signal and chip select enabled, the byte pointed to by ERDPT will shift out on the SO line, which means that when the bit AUTOINC=1, I can read the continuously the receive buffer.

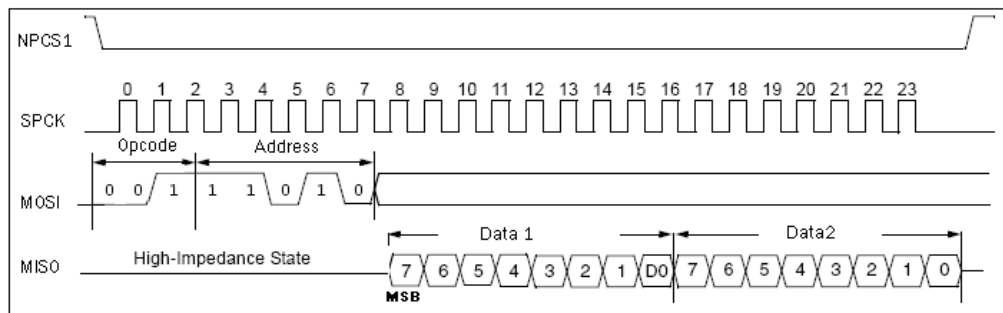


Figure 35. RBM Command Sequence

3. Write Control Register (WCR) Command

This command allows writing any of the ETH, MAC and MII registers in any order. Unlike the RCR command, in this command there is not distinction between the ETH and MAC/MII registers.

As can be seen in Figure 36, once the chip select is enabled, the microcontroller has to send the first byte composed by the WCR opcode, followed by a 5 bit address that identifies the register of the current bank which I want to write.

After sending the first byte, the microcontroller will send a second byte whose value will be written in the selected register. The operation ends when the last bit of the data is sent and the chip select is disabled. If it was disabled before sending the 8 bits, the write command will be aborted.

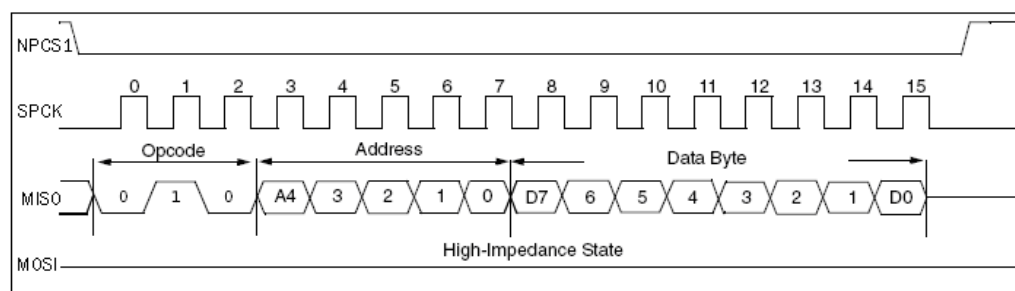


Figure 36. RCR Command Sequence for ETH, MAC and MII Registers [26, 28]

On the contrary, if the registers that I want to write are the PHY registers, the steps to accomplish this are as follows:

- a. The 5 bit address of the desired PHY register is written in the MIREGADR register.
- b. The lower 8 bits of data has to be written into the MIWRL register.
- c. The upper 8 bits of data has to be written into the MIWRH register. When MIWRH is written, the flag MISTAT.BUSY= '1', and it clears itself when the writing operation has finished. This takes 10.24 μ s.

4. Write Buffer Memory (WBM) Command

This command allows the writing of the Ethernet buffer and more concretely the transmit buffer memory. That is, I write the data into the transmit buffer that I want to send later to the host through Ethernet cable.

Similarly to the receive buffer, when the bit AUTOINC of the ECON2 register is set, the EWRPT Pointer is automatically incremented to the next address after the last bit of each data byte is written. That is, in the case of the keeping the clock signal and chip select enabled, I can write continuously into the buffer without any other WBM opcode. In the case that EWRPT point to 1FFFh, the write pointer will be incremented to 0000h.

Figure 37 shows that in the same manner as with the WCR command, once the chip select is enabled, the microcontroller sends the first byte composed by the WBM opcode and followed by a 5 bit constant (1Ah). After this first byte (0x7A) has been sent, the microcontroller will be the byte which will be written into transmit buffer at the address pointed to by EWRPT. The operation ends when the data are sent, and chip select is disabled.

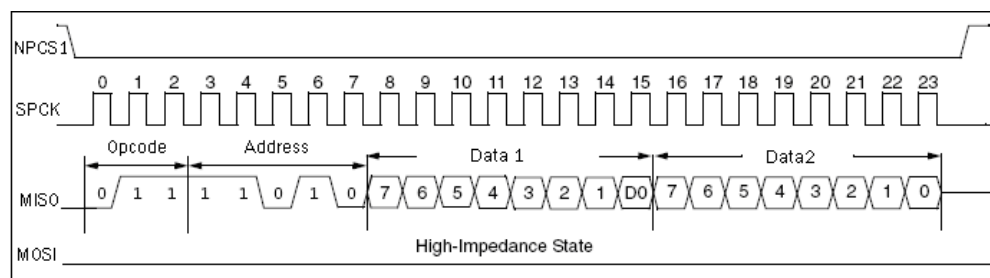


Figure 37. RCR Command Sequence for ETH, MAC and MII Registers [26, 29]

5. Bit Field Set (BFS) Command

This command can be used only with ETH registers since it does not work with MAC or MII registers. It provides a bit-wise OR operation between the supplied data in the command and the content of ETH addressed register. It is usually used to set any bit since it is better than the WCR command. As seen in Figure 38, once the chip select is enabled; the BFS opcode is sent by MOSI line, followed by a 5 bit address that identifies to the ETH register of the current bank. After sending this first byte, it sends the data (MSB first) by means of which the bit-wise OR operation will be performed.

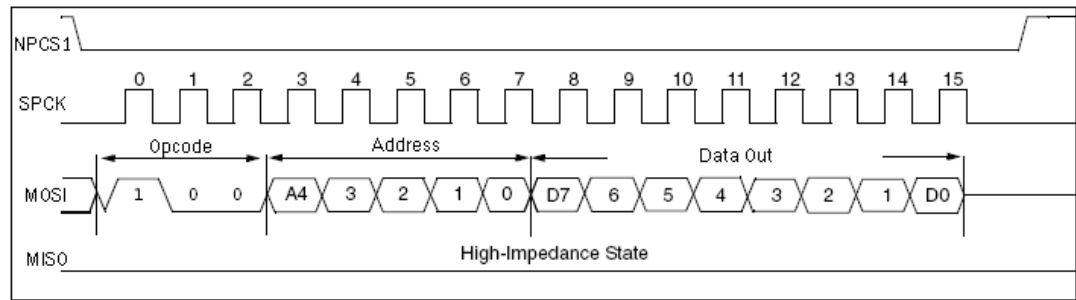


Figure 38. BFS Command Sequence

If chip select is disabled before carrying out the operation in the 8 bits, the operation is aborted.

6. Bit Field Clear (BFC) Command

Just like the BFS command, BFC command can be only used with ETH registers. This command provides a bit-wise NOTAND operation between the data supplied and the content of ETH addressed register. The operation is simple. First the ECN28J60 makes the inversion of the supplied data (second byte). Then, the AND operation is carried out among the data inverted and the content of ETH addressed register. It is usually used to clear any bit since it is better than the WCR command.

As shown in Figure 39, once the chip select is enabled; the BFS opcode is sent by MOSI line, followed by a 5 bit address that identifies the ETH register of the current bank. After sending this first byte, it is sent the data (MSB first) by means of which the bit-wise NOTAND operation will be performed.

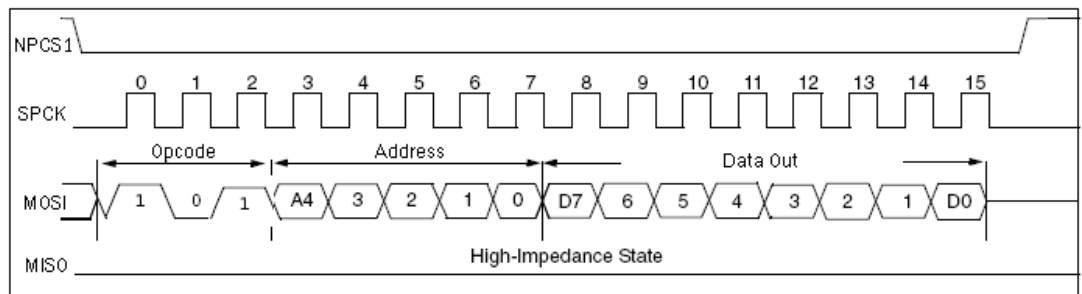


Figure 39. BFC Command Sequence

If chip select is disabled before carrying out the operation in the 8 bits, the operation is aborted.

7. System Reset Command (SRC)

Unlike other SPI commands, the SRC does not operate on any register. Otherwise, it allows the host to do a reset with software. Figure 40 illustrates as is the command sequence. The microcontroller only needs sending a single data of 8 bits on the MOSI line.

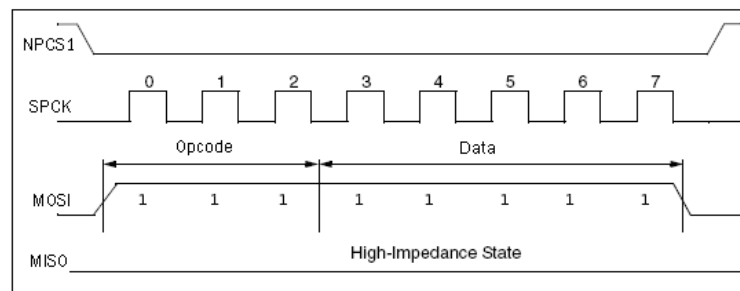


Figure 40. System Reset Command Sequence

4.6 LAN Overview

Since I need to program my own TCP/IP stack, this section is oriented to reviewing of the concepts used in Computer Networks. Like the Radio Frequency, the Computer Networks is a wide field and it could be a topic for another thesis. A brief overview of the matters must be known in order to understand how my TCP/IP works is made below. The majority of the information was extracted from the books written by Tanenbaum [30] and Stallings[31] although more information about this topic can be found on the internet.

The TCP/IP stack can be really complex depending of the application that has to support DHCP, LL A, NTP, HTTP, Telnet and SSH for example. As the goal of this part of the thesis to focus on showing the communication of the network formed between a host and the microcontroller together with the Ethernet controller, my TCP/IP is less complex. The easiest and fastest way to test the communication in a network is making a “ping” between both systems. This is explained in more detail in chapter 5.

In short, as the Ethernet interface is based on the use of a TCP/IP Stack, the main concepts and protocols involved in a computer network, more concretely a Local Area Network (LAN) are reviewed first.

A LAN is a private network of a few kilometers of length that can be connected using different topologies (line, ring, bus, etc) which are not explained here. Besides, different LAN's can be interconnected by routers and/or switches as is illustrated in Figure 41. In this case, each subnet will have to properly write the routing table of its router, that is, different tracks for one packet can arrive from the source to destination through the other routers of the network.

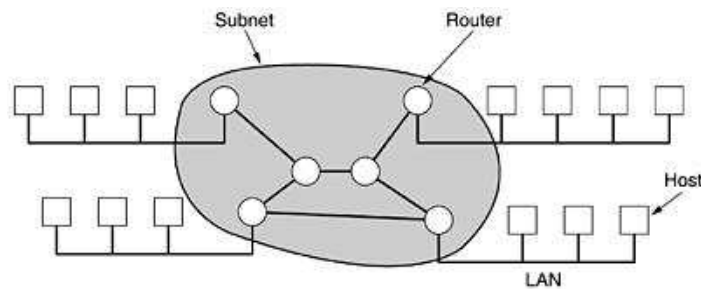


Figure 41. LAN Interconnections [30, 20]

The solution showed above can be found in large companies and universities for example, where the number of host or departments is high. As for the reader case this network would be a little simpler because we would only have the UHF RFID Reader plugged to a computer through a router. The reason for using this device of interconnection is the type of Ethernet cable used in the communication.

Mainly, this kind of cable is unshielded twisted pair (UTP) Category 5, and it is composed by 8 wires. Twisted pair wires avoid electromagnetic interferences and the category is a measure of quality, meaning that the wires support traffic up to 100Mb/s.

The problem is due to the fact that the majority of the Ethernet cables are Straight-Through Cables, that is, the Pin Out (Figure 42) is the same in both sides and the use of a intermediate interconnection element (router, hub or switch) is required.

There are also cross-over cables. In contrast to straight-through cables, their Pin Out is different in each extreme, in such a way that the Transmit+ of the one side is connected to the Receive+ of the other, and the same applies to the Transmit- and Receive- [32].

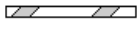
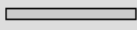
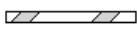
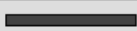
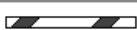



RJ45 Pin #	Wire Color (T568B)	Wire Diagram (T568B)	10Base-T Signal 100Base-TX Signal
1	White/Orange		Transmit+
2	Orange		Transmit-
3	White/Green		Receive+
4	Blue		Unused
5	White/Blue		Unused
6	Green		Receive-
7	White/Brown		Unused
8	Brown		Unused

Figure 42. Straight-Through Cable Pin Out for T568B [32]

Nowadays there are network cards, switches and routers that have a mechanism called auto-medium dependent interface crossover (MDIX), which detects the kind of cable plugged into Ethernet port which is why a cross over cable is not needed. That is, if a Straight-Through cable is detected between two hosts, internally the hardware is in charge of the TX/RX crossing although the use of a router is usually recommended.

My network is composed of a computer, the group formed by the microcontroller and the Ethernet controller plugged directly through an Ethernet cable.

Once the physical connection has been established, the communication within a network is based on the use of protocols, services and primitives among the different layers of the stack.

A service consists mainly of transferring messages between two layers (request and offer it). That is, an upper layer requests services to the layer immediately lower while a layer N-1 offers services to layer N.

However, the communication between the same levels is carried out with primitives (operations) and protocols (set of rules regulating the format and meaning of the packets or messages that are exchanged between entities of a layer). [30, 24-32; 31, 46]

Although in network architecture the two reference models managed are OSI and TCP/IP, actually only the TCP/IP model is used, but the theory of OSI model is already valid.

The model TCP/IP is used on the internet and for all the communication tasks. It is based on a model of five levels although some texts talk about a model of four levels (see Figure 43).

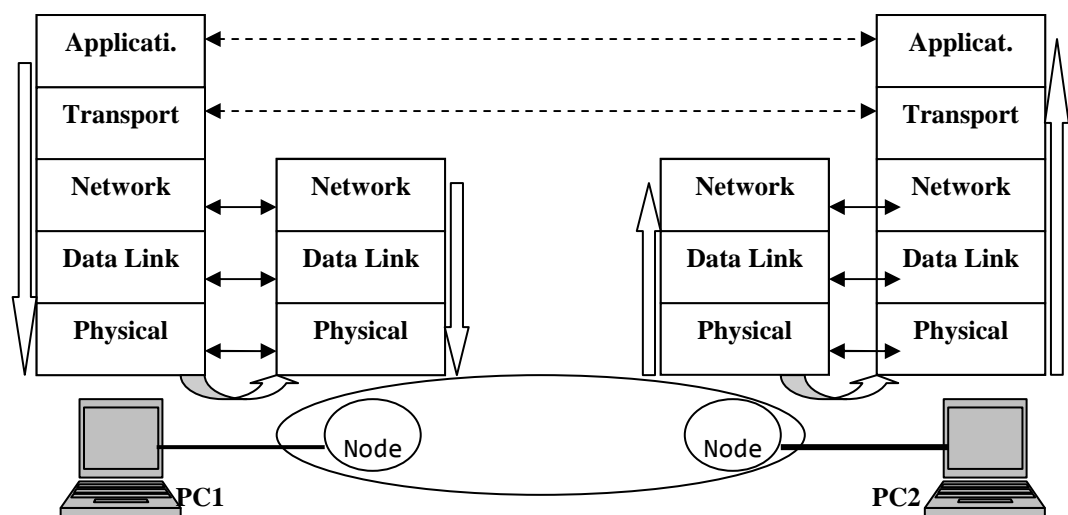


Figure 43. TCP/IP Architecture Model

Figure 43 above shows the whole TCP/IP architecture model and an example of the direction of the communication (level to level by the services) from PC1 to PC2. The nodes in the physical layer represent routers (only working until level 3) while the space between them represents the transmission medium within subnet.

Each level will be briefly explained on next page to help to the reader to understand the role played by them within the stack [31, 34-42]. Evidently, each layer has its own purpose so that the highest layer contains only the user data, and a protocol header is added in each lower layer as it is shown in Figure 44.

- *Application layer (Level 5)*

It is considered the upper layer and provides the services used for the user's different applications, e.g. HTTP, FTP, SMTP and DNS.

- *Transport layer (Level 4)*

It is a secure connection between extremes. It is composed of two protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

- TCP is a connection-oriented protocol that has to be complex to do a reliable transmission. It uses the datagram protocol.

- UDP is a not connection-oriented protocol. It does not guarantee the delivery but makes it possible to send the messages between applications.

- *Internet layer (Level 3)*

The IP Level or routing is not connection-oriented protocol. There is a unique Protocol Data Unit (PDU) called Datagram.

The IP protocol uses this level to offer the routing service through several networks. It is implemented in the final system by means of which intermediate routers can be interconnected to the network where we want to go.

- *Network access layer (Level 2)*

This level is used for error control between the host and the network. Moreover, it is responsible for the data exchange between the end system (e.g. server and working station) and the network. The protocol used depends on the kind of Network.

- *Physical layer (Level 1)*

It is the lower level and defines how I am connected to the network. That is, it defines the physical interface between the transmission medium and the network (e.g. Ethernet, ATM, X25, etc). This level carries out the specification of the transmission medium, e.g. the signal nature and TX speed.

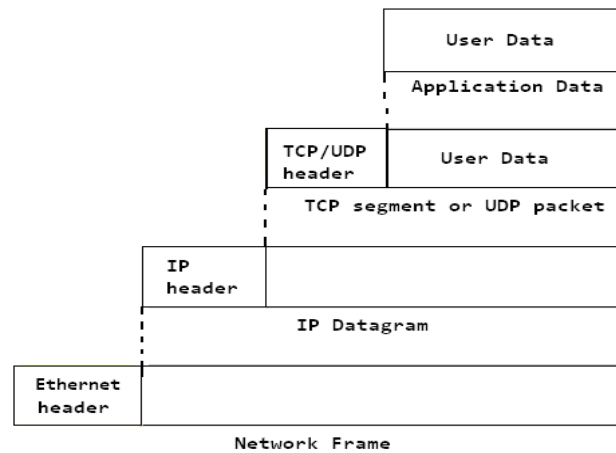


Figure 44. Protocol Data Units (PDUs) in the TCP/IP Architecture [31, 37]

As it is shown above, each layer includes its own header in addition to the desired data. That is, identical layers keep the header of its level and the rest is delivered to the upper level until the desired data remains.

In short, the coding of a TCP/IP stack requires building the right frames of each level composed mainly by the header and the data. The different headers and protocols used in the TCP/IP stack are regulated as Internet standards and they are described in the Request for Comment (RFC). In the case of the Physical layer, the standard is the IEEE 802.3 and the packet format is illustrated in Figure 45 below [30, 223].

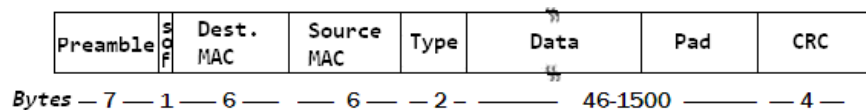


Figure 45. Ethernet Packet Format (version 2) [30, 223]

The Ethernet frame format (see Figure 45) is the lowest frame that I can manage in the TCP/IP stack, so the data field contains the information of the upper levels (IP header, TCP header and data application).

Generally, just the header and data compose the network frame. The preamble and Start of Frame (SOF) do not belong to network header else that it is composed of the destination address, source address and type. The CRC field is like an “ending header” and is neither included in the network frame representation.

As to the data field, the maximum size for the data payload is 1500 octets. When sending data larger than this value, fragmentation is required. That is, the host or router split the datagram in several smaller packets and are reassembled in the destination. However, as the minimum size of the frame (header + data) is 60 bytes, sometimes it is necessary to add pad octets (mainly zeros) to reach this minimum.

In the case of the MAC level of the ECN28J60, both the preamble and SOF are generated automatically while the padding field and checksum can also be generated but you need to configure them first. The rest of fields have to be filled into the ECN28J60 transmit buffer before sending the Ethernet frame.

As it was previously mentioned, my TCP/IP stack has to be able to do a ping between both systems. So let's go to start from the beginning. A ping is a computer network utility used to check the communication within a network by the exchange of Internet Control Message Protocols (ICMP). As its name suggests, this protocol provides a medium to transfer messages between two systems, usually between a router and a host or two hosts.

The ICMP protocol belongs to IP level so the messages are delivered like IP Datagram and the delivery can no be guaranteed (only TCP level can do it). In other words, ICMP will be made up of an IP header (see Figure 46) and an ICMP payload.

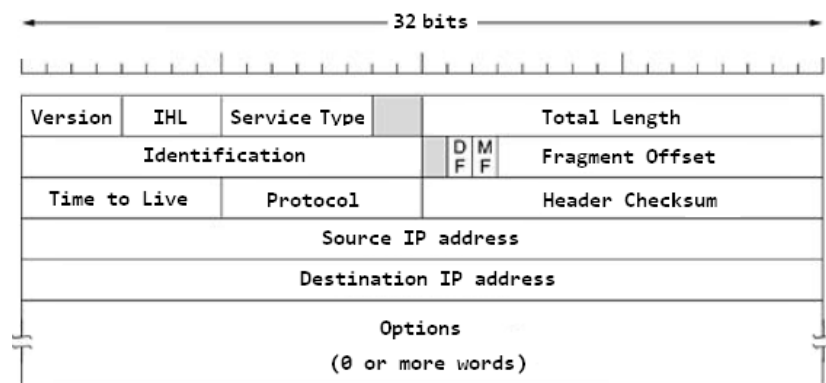


Figure 46. IPv4 Header [31, 578]

Although there are several kinds of ICMP messages [31, 582], the ping utility only uses two of them: echo and echo reply. One of the extremes sends an ICMP request and the other answers with an ICMP reply. The message format (ICMP payload) is the same for an ICMP echo or echo reply but the content is different (see subsection 5.3.7) and is illustrated below.

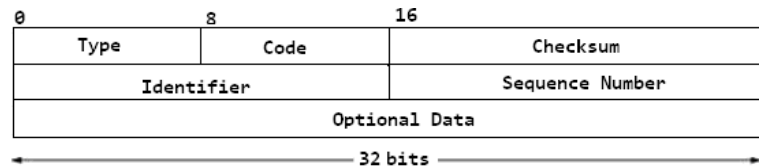


Figure 47. ICMP Echo [31, 583]

As we can see in Figure 46, an IP header manages IP addresses, but before the source can send an ICMP request it needs to know to whom the destination IP address belongs. This problem arises from the fact that only the MAC addresses are unique, (as it was explained in the section 4.3, pages 45-46) while the IP addresses are configured by the user and can be change at any moment.

For this reason, the first time that a host tries to make a ping, an ARP packet is sent. Like the ICMP echo messages, ARP also has ‘request’ and ‘reply’ packets with the same format (Figure 48) but different content.

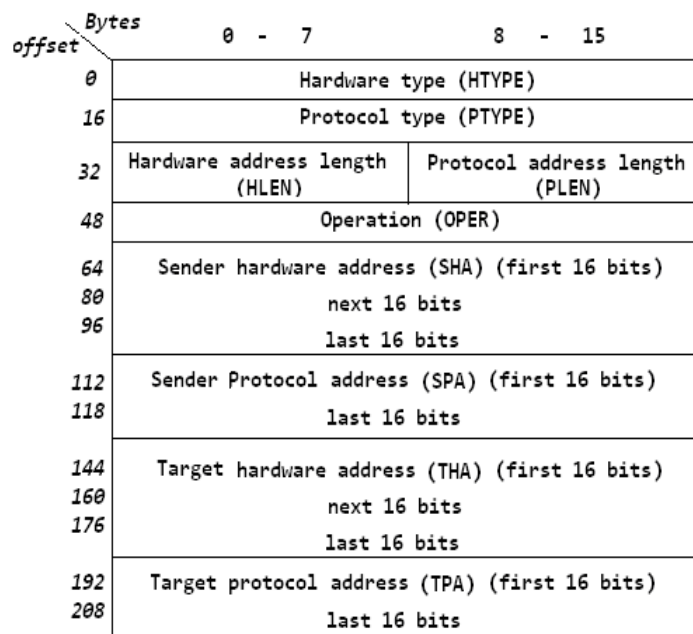


Figure 48. ARP Packet Format

The acronym ARP comes from Address Resolution Protocol. It resides in layer two of the TCP stack and its function is assigning a determinate IP address to a specific MAC or physical address [30,450]. Once the source host receives this relation then it is written into an ARP table. In this way, each time that source host asks for the same IP address, an ARP packet does not need to be sent again. Figure 49 illustrates the packet exchange during the first attempt of ping from the host to the Ethernet controller.

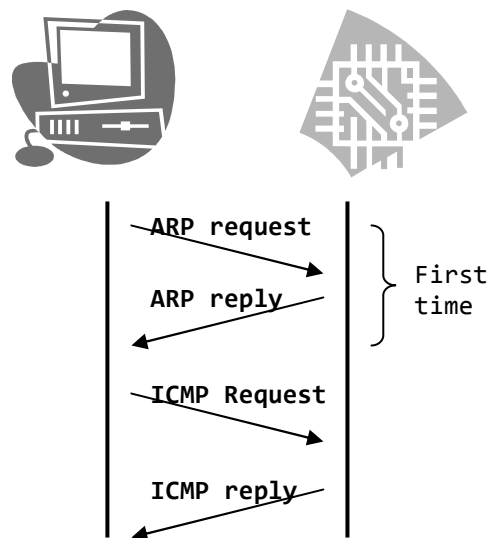


Figure 49. Example of Packet Exchange for Ping Utility

If a network can exchange ARP packets in the communication between the tested systems, this network is working perfectly regardless of the ICMP packets exchange. This is because ARP manages physical addresses in place of IP addresses. So it can occur that a ping fails (e.g. due to firewall block) but the ARP packet exchange has been done.

In short, a TCP/IP stack based on a ping application needs only three levels and has to be able to manage ARP and ICMP packets. Its coding is explained in the next chapter.

The other levels (TCP and application) and protocols have been omitted and they will not be explained because they are not used in this thesis. For more information, the books “Computer Networks [30]” and “Data and Computer Communications [31]” can be consulted.

5 Pinging the Ethernet Interface

This chapter focuses on the software used in the communication test. Apart from showing the coding of the TCP/IP stack, it explains the configuration of the Ethernet controller as well as that of the microcontroller and its interfaces used for this purpose.

First, the physical connection is introduced, next the software used is presented and then the software is explained. At the end of this chapter a list of the problems found during the development of the coding is presented.

5.1 Ethernet Controller Connection

Since it is not necessary to have any development board to carry out this communication test, Figure 50 illustrates the general connection diagram with the pins used in each device although some details have been omitted

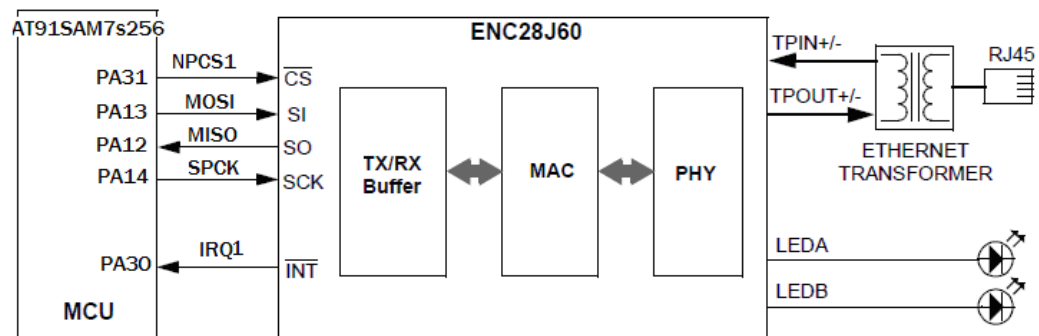


Figure 50. Connection between the Ethernet Controller and the microcontroller [28, 4]

5.2 Steps before coding

As it was mentioned previously, the software used for the programming of the microcontroller is the IAR Embedded Workbench tool. Anyway, on the internet there are numerous programs with free license to compile, debug and load your applications and their use is described in the reference “Using Open Source Tools for AT91SAM7S Cross Development [33]” written by Lynch .

The software is coded in the C# Programming Language and lately loaded in the RAM or FLASH of the Atmel microcontroller. Since the programming of this kind of microcontroller is different to that of the typical 8051 microcontroller, reading Atmel's document 'ARM-based Software Packages [34]' which briefly describes the organization and contents of the AT91 packet software, is recommended.

Before explaining the codes which are located in the appendices, there is a brief "tutorial" about the IAR Embedded Workbench tool with the main steps carried out. Since on the internet you can find many manuals or user guide about this tool, in this "tutorial" screen captures has been omitted to save space.

In summary, once the software has been installed and the program is already opened, the following actions are needed:

- ◆ Create a new project

Project/Create New Project/ARM (Empty Project)/name of project.ewp

Before creating a new project, it is useful create a files structure with the aim of tying the different files that will be used in the program. The structure showed in Figure 51 is commonly used in the examples programs on the website of Atmel.

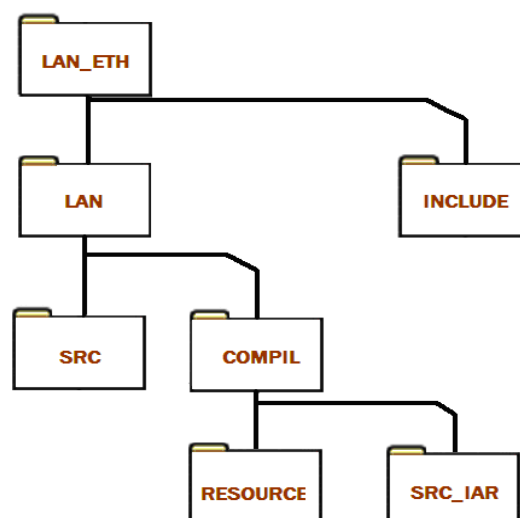


Figure 51. Example of the Tree of Files Used in the Ping Program

The content of the resource and src_iar files as well as the header files used by the AT91SAM7s256 microcontroller can be got from the examples available on Atmel's website. The information contained in them (header files, start-up files, macro files) is essential for the configuring of the microcontroller.

◆ Save the workspace

File/Save Workspace/name.eww

Similarly to the tree of files created previously, in this project there can be different groups with the purpose of dividing the Atmel's files and the files coded during project.

- *Project/Add group*

It is used to arrange the files by type (e.g. src and src_iar).

- *Project/Add files*

The different files are aggregated inside each group. So that the C# files coded by myself are put in src file and the start_up files are added in the src_iar group.

◆ Configuration

Before the program can be debugged and loaded in the RAM of the microcontroller, it is necessary to configure some aspects of the tool.

Project/Options

The options to modify are as follows:

- General Options/Device/AT91SAM7s256
- C/C++Compiler/Preprocessor/
 - Additional include: \$PROJ_DIR\$\src_iar\
\$PROJ_DIR\$\..\..\
- Assembler/Preprocessor/
 - Include Paths: \$PROJ_DIR\$\..\..\include\
- Linker/Config
 - Override default/AT91SAM7s256_64RAM.xcl
- JLink/use macro file/SAM7RAM.mac

5.3 Ping Utility Programming

The C# files used by the ping program are explained in this section, but the whole source code with their respective comments can be found in the appendix section. Figure 52 is a screen capture taken from the IAR tool where the different files are listed.

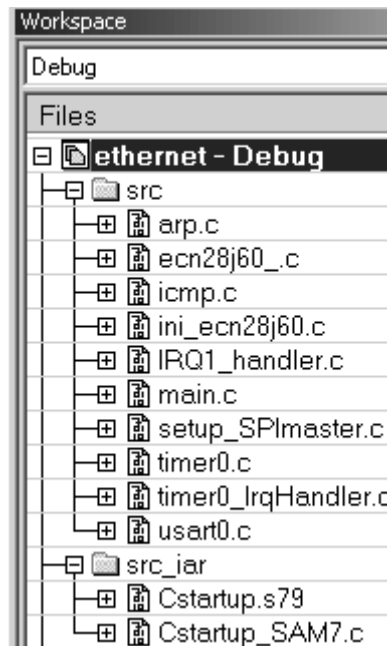


Figure 52. Ping program IAR Screen Capture

Each function belonging to the src group (Figure 52) is explained below, but the order is different than in the above figure.

5.3.1 setup_SPImaster.c

This file contains the function in charge of configuring the AT91SAM7s256 in the master mode with the requirements imposed by the slave (frequency, mode, etc).

As shown in Figure 50, the chip select pin of the Ethernet controller was plugged to NPCS1 pin (PA31) of the microcontroller in this project so that the NPCS0 (PA11) is left for the future reader chip connection. As the entire microcontroller pins are I/O lines, it is necessary to configure the PIO controller to assign the SPI pins to their peripheral functions before using the SPI lines.

Another important thing is enabling the peripheral clock for the SPI interface in the power management controller (PMC). Both operations can be carried out by making use of the inline functions `AT91F_SPI_CfgPeriph` and `AT91F_SPI_CfgPMC` (read Atmel's document [34]). They can be found in the 'lib_AT91SAM7S256.h' Atmel's header file. These functions are very useful because they are already coded by Atmel and you only need to pass the parameters.

After that, the configuration of the SPI interface can start so the SPI registers of the microcontroller have to be properly programmed. The SPI registers are mapped from 0xFFFFE 0000 address, and the registers used are explained in more detail below.

a. Mode Register

Name:		SPI_MR							
Access Type:		Read-write							
31	30	29	28	27	26	25	24		
DLYBCS									
23	22	21	20	19	18	17	16		
PCS									
15	14	13	12	11	10	9	8		
-									
7	6	5	4	3	2	1	0		
LLB	-	-	MODFDIS		PCSDEC	PS	MSTR		

Figure 53. SPI Mode Register [26, 277]

Figure 53 shows the content of the 32 bit register used for the configuration of the SPI operation mode. In this project, this operation mode has the following characteristics:

- ◆ Master Mode (MSTR=1).
- ◆ Variable Peripheral (PS=1) → the peripheral selection has to be defined in the Peripheral Chip Select (PCS) field of the Transmit Register for each new data.
- ◆ Without external decode (PCSDEC=0).
- ◆ Disable mode Fault (MODFDIS=1).
- ◆ Without Local Loopback (LLB=0).
- ◆ Peripheral Chip Select (PCS=X): This field is only used in Fixed mode (PS=0).
- ◆ Delay Between Chip Select (DLYBCS=0) →(Default value is zero)

b. Chip Select Register

Name:		SPI_CSR0... SPI_CSR3									
Access Type:		Read-write									
31	30	29	28	27	26	25	24				
DLYBCT											
23	22	21	20	19	18	17	16				
DLYBS											
15	14	13	12	11	10	9	8				
SCBR											
7	6	5	4	3	2	1	0				
BITS				CSAAT		-	NCPHA	CPOL			

Figure 54. SPI Chip Select Register [26, 286]

Figure 54 shows the content of the 32 bit register where the each chip select is configured. In this project, the chip select 1 was configured because the NPCS1 pin was used as the chip select of the ECN28J60. The chip select 0 is reserved for the reader and at the moment it is not defined.

- ◆ Mode 0 → CPOL=0, NPCHA=1
- ◆ Chip Select Active After Transfer (CSAAT=1) → The chip select is enabled until it is requested for another different chip select or Last Transfer (LASTXFER) field of the Transmit Register is set. That is, two or more frames can be sent consecutively without disabling the chip select.
- ◆ Bits Per Transfer (BITS=0000) → 8 bits are send in each transfer.
- ◆ Serial Clock Baud Rate (SCBR) → Its value establishes the frequency of the SPI clock signal (SPCK). It is calculated by the formulae 3.

$$SPCK = \frac{MCK}{SPCK} = \frac{47923200}{10\text{ MHz}} = 4,79232 \approx 5 \quad (3)$$

- ◆ Delay Before SPCK (DLYBS=0) → Default valued = $\frac{1}{2}T_{SPCK}$
- ◆ Delay Between Consecutive Transfers (DLYBCT=1) → (see section 5.4, p.104)

The SPI interface of the microcontroller has the option of handling the SPI interrupt but this requires programming the Advanced Interrupt Controller (AIC) before using the SPI interrupt. In this project, this option was not used.

In conclusion, the software flowchart for the function **void setup_SPImaster (void)** is shown in Figure 55 and its code can be consulted in Appendix 1.

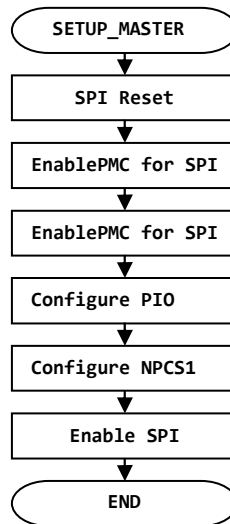


Figure 55. Software Flowchart of the Function `setup_SPImaster`

5.3.2 Timer0.c

This file contains the function put in charge of configuring the channel 0 of the microcontroller's timer ("timer0") with the purpose of coding a wait time. The values assigned are able to generate an interrupt of 1 ms each. As the wait time only has to occur after a software reset in the ECN28J60, the enabling of the timer0 is not done in this function but in the `SystemCommandReset` external function (see Figure. 70, page 92).

The AT91SAM7s256 microcontroller has three timer counters (TC). They are identically and can be independently programmed. This project used just one in this application (Timer0). (For more information about the Timer Counter can be consulted the pages 439-472 of the AT91SAM7s256 Datasheet [26]).

Similarly the SPI interface, before using the timer0, the PMC clock must be enabled. Once, this is done, the configuration of timer0 is carried out by the programming of the TC registers of the microcontroller. The TC registers are mapped from 0xFFFFA 0000 address and the registers used are described below.

d. Channel Mode Register

This register is different as per the operation mode of the channel (WAVE bit value). In such a way that each channel can independently operate in two modes:

- Capture Mode (for the measurement on signals). (WAVE=0)
- Waveform Mode (WAVE=1).

In this project, I decided to configure the “timer0” in capture mode since I only want to program a wait time which is shown in Figure 59.

Register Name: TC_CMRx [x=0..2] (WAVE = 0)
Access Type: Read-write

31	30			19	18	17	16
-	-	-	-	LDRB		LDRA	
15	14	13	12	11	10	9	8
WAVE	CPCTRG	-	-	-	ABETRG	ETRGEDG	
7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI		TCCLKS	

Figure 59. TC Channel Mode Register: Capture Mode [26, 459].

The configuration of the timer0 is imposed by the characteristics of the counter. In this project, the Ethernet controller needs to wait at least 1ms after doing a reset command before the MAC level is ready. The values used in each field are explained below.

- ◆ Clock Selection (TCCLKS): In this field the clock used during the count of the Timer is chosen. This project used the internal clock. As the interrupt of the timer0 is for 1ms, the minimal pre-scaler required has to be calculated by formula 4.

$$Div_{MIN} = t_{desired} \frac{MCK}{2^{16}} = 1ms \frac{47923200}{65535} = 0,731 \quad (4)$$

The internal clock can be prescaled by 2, 8, 32, 128, 1024. Although any division larger than 0,731 is valid, the bigger value was chosen. In other words, the internal clock for the counter is the master clock, and it is divided by 1024 (prescaler 5) →TCCLKS=100.

- ◆ Clock Invert (CLKI=0). Default value
- ◆ Burst Signal Selection (BURST=0). The clock is not gated by an external signal.
- ◆ Counter Clock Stopped with RB Loading (LDBSTOP=0). Default value.
- ◆ Counter Clock Disable with RB Loading (LDBDIS=0). Default value.
- ◆ External Trigger Edge Selection (ETRGEDG=0).
- ◆ TIOA or TIOB External Trigger Selection (ABETRG=0). Default value.
- ◆ RC Compare Trigger Enable (CPCTRG=1).

The counter is stopped when the count arrives to the RC value and it is initialized again. The RC value has to be loaded in the TC Register C, which will be discussed later.

- ◆ WAVE: It has the zero value to indicate the capture mode.
- ◆ RA Loading Selection (LDRA=0).
- ◆ RB Loading Selection (LDRB=0).

e. TC Register C

This register keeps the RC value which will be compared with the timer count. This value is calculated as per formula 5 and then it loaded in the register shown in Figure 60.

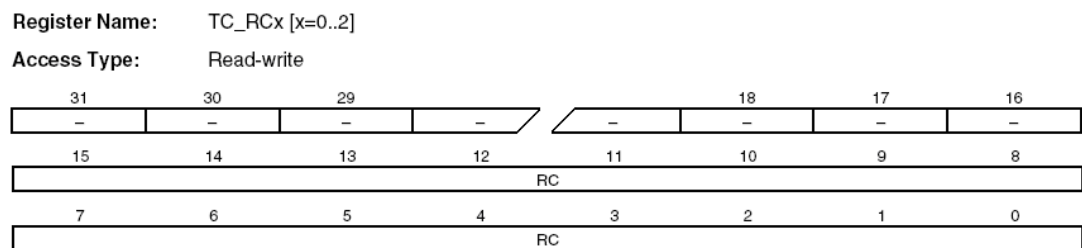


Figure 60. TC Register C [26, 467]

$$RC = \left[t_{desired} f_{TIMER0} \right] - 1 = \left[1ms \frac{MCK}{1024} \right] - 1 = \left[1ms \frac{47923200}{1024} \right] - 1 = 45,8 \approx 46 \quad (5)$$

f. TC Interrupt Enable Register.

Figure 61 shows the content of this register which in this project only enabled the RC compare interrupt (CPCS=1). Before using the timer interrupt, it is necessary to configure the advanced interrupt controller (AIC).

The AT91SAM7s256 has 32 possible sources of interrupt, but only the timer interrupt configuration is explained [26, 159-184].

Register Name: TC_IERx [x=0..2]

Access Type: Write-only

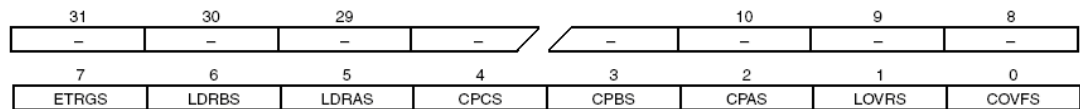


Figure 61. TC Interrupt Enable Register [26, 470].

Just as before, for the configuration of the timer interrupt, the inline functions defined in the 'lib_AT91SAM7S256.h' header file were used in this project. The function showed in Figure 62 is used for the initialization of the interrupt. The more important parameters passed to the function are:

- ◆ Address of the interrupt handler: void Timer0_IrqHandler (void)
- ◆ Priority- there are 8 levels of priority from 0 (lowest) to 7 (highest). I choose an intermediate level (4)
- ◆ Type of activation – Edge-triggered or level-sensitive. This project used the first one.

```

/*-----
/* \fn    AT91F_AIC_ConfigureIt
/* \brief Interrupt Handler Initialization
/*-----
inline unsigned int AT91F_AIC_ConfigureIt (
    AT91PS_AIC pAic, // \arg pointer to the AIC registers
    unsigned int irq_id, // \arg interrupt number to initialize
    unsigned int priority, // \arg priority to give to the interrupt
    unsigned int src_type, // \arg activation and sense of activation
    void (*newHandler) (void) ) // \arg address of the interrupt handler
{
    unsigned int oldHandler;
    unsigned int mask ;

    oldHandler = pAic->AIC_SVR[irq_id];

    mask = 0x1 << irq_id ;
    /* Disable the interrupt on the interrupt controller
    pAic->AIC_IDCR = mask ;
    /* Save the interrupt handler routine pointer and the interrupt priority
    pAic->AIC_SVR[irq_id] = (unsigned int) newHandler ;
    /* Store the Source Mode Register
    pAic->AIC_SMR[irq_id] = src_type | priority ;
    /* Clear the interrupt on the interrupt controller
    pAic->AIC_ICCR = mask ;

    return oldHandler;
}

```

Figure 62. Function for the Initialization of the Interrupt

The most relevant registers involved with the *AT91F_AIC_ConfigureIt* function are explained below.

g. AIC Source Vector Register

As shown in Figure 63, there are 32 AIC Source Vector Registers different (one for each possible interrupt). Each interrupt has its own Peripheral ID which is defined in the ‘AT91SAM7S256.h’ file and can be found in the page 34 of the datasheet [26]. In the case of the timer interrupt the value of this number is 12, so AIC_SVR [12] tells where the ‘Timer0_IrqHandler ()’ function address is stored.

Register Name: AIC_SVR0..AIC_SVR31
 Access Type: Read/Write
 Reset Value: 0x0

31	30	29	28	27	26	25	24
VECTOR							
23	22	21	20	19	18	17	16
VECTOR							
15	14	13	12	11	10	9	8
VECTOR							
7	6	5	4	3	2	1	0
VECTOR							

Figure 63. AIC Source Vector Register [26, 176]

h. AIC Source Mode Register

Similarly to the source vector register, there is a source mode register for each interrupt. Figure 64 shows the content of this register. In it, the priority and type of activation of the interrupt are defined.

Register Name: AIC_SMR0..AIC_SMR31
 Access Type: Read/Write
 Reset Value: 0x0

31	30	29			10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	SRCTYPE		-	-	PRIOR		

Figure 64. AIC Source Mode Register [26, 175]

The flowchart in Figure 65 describes the behavior of the function **void Timer0Setup (void)** of the file *Timer0.c* and its code can be found in Appendix 3.

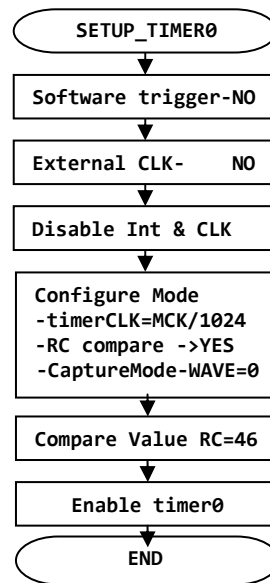


Figure 65. Software Flowchart of the Function Timer0Setup

5.3.3 Timer0_IrqHandler.c

This file contains the timer interrupt service routine (ISR), so that when an interrupt occurs the program enters automatically this function. The software flowchart in Figure 66 illustrates the operations carried out in the ISR by the **void Timer0_IrqHandler (void)** function whose code can be found in the Appendix 4.

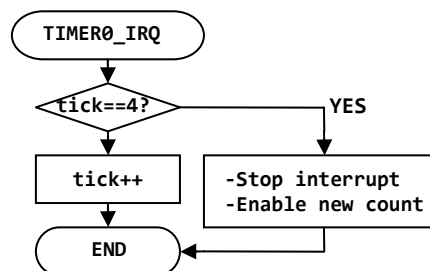


Figure 66. Flowchart for the Function Timer0_IrqHandler

In this service routine, an external variable called ‘tick’ was created with the aim of disabling the timer0 when the wait time has expired. The counter is stopped by the Channel Control Register (TC_CCR) and is re-enabled again to leave the Timer0 in a ‘standby-mode’. When the SystemResetCommand external function is called again, a new count starts, and the Timer0 will generate a new interrupt of 1 ms each.

5.3.4 ini_ecn28j60.c

This file contains the function of initialization of the Ethernet Controller as directed on the datasheet of the ECN28J60 (pages 33-38) [28] as well as the operations to fix the silicon errata present in all the chips. Each chip has a revision identifier (1, 4, 5 or 7), which is located in the EREVID register at the address 0x12 of the bank three. The silicon errata document which describes the actions to fix the possible problems can be downloaded from Microchip's website.

All the registers managed in this function belong to ECN28J60 memory. During the initialization of the ECN28J60 different tasks which have been coded in the **void ini_ECN28J60 (void)** function are carried out (Appendix 6). These tasks are explained below.

a. Receive and Transmit Buffer

During the buffer initialization is configured the size of the receive buffer by the ERXST and ERXND Pointers to determine its length. The ERDPT Pointer has to be programmed with the same value of ERXST for tracking purposes. The receive buffer size depends on the type of the application the memory requirements can be larger or smaller. One of the errors has to do with the receive buffer, Microchip advises the user to start the receive buffer from the 0000h address.

In this project, the same space was used for both receive and transmit buffer. That is, the receive buffer is defined from 0000h-0FFFh being the rest of space considered as transmit buffer.

b. Receive Filters

The ECN28J60 incorporates different receive filters with the purpose of allowing the access of desired packets and excluding the rest. They are selected in the ERXFCON Register.

In this project, three filters were enabled to make a ping, the broadcast filter allows the ARP packets, the unicast filter, filters the MAC address and the pattern match filter.

This last filter selects up to 64 bytes from the incoming packets and then calculates an IP checksum of these bytes. If this checksum has the same value as the EPMCS registers, the packet meets the criteria.

To use the Pattern Mach filter, three different registers were programmed.

1. EPMOH=0x00, EPMOL=0x00. In these registers the offset is programmed. In this project is zero.
2. EPMCSH=0xF9, EPMCSL=0xF7. The checksum is programmed in these register. An IP checksum is calculated like the 16 bit one's complement of the sum of all 16 bit words (see Table 6). In this project, an ARP Packet (0x0806) has to be filtered. (it must be stored in bigger endian → 0x0608) and my address destination is the Broadcast (FF: FF :FF :FF :FF :FF).

Table 6. Checksum Calculation Example

Reg 1	0x0608
Reg 2	0xFFFF
Reg 3	0xFFFF
Reg 4	0xFFFF
$\sum_{i=1}^4 \text{Reg}$	0x0003 0605 high low
Reg a	0x0003
Reg b	0x0605
$\sum_{i=a}^b \text{Reg}$	0x0608
One's complement	0xF9F7 <u>CHEKSUM</u>

3. EPMM1=0x30, EPMM0=0x3F. The Pattern Match Mask is programmed in theses registers. Although the mask can have programmed up to 64 bits, in this project only 16 are used. The mask is calculated in similar way to an IP mask. In this project, a specific MAC address is filtered, as is shown in Table 7.

Table 7. Pattern Match Mask Example

Type	Source MAC	Destination MAC																												
0x0608	00:04:A3:01:01:01	FF:FF:FF:FF:FF:FF																												
<table border="1"> <tr> <td>1</td> <td>1</td> </tr> <tr> <td colspan="2">3</td> </tr> </table>	1	1	3		<table border="1"> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td colspan="6">0</td> </tr> </table>	0	0	0	0	0	0	0						<table border="1"> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td colspan="3">3</td> <td colspan="3">F</td> </tr> </table>	1	1	1	1	1	1	3			F		
1	1																													
3																														
0	0	0	0	0	0																									
0																														
1	1	1	1	1	1																									
3			F																											

c. Waiting for OST

Before modifying any MAC or PHY registers, it is necessary to wait until the CLKRDY flag is set. So in this project, before doing the steps (d) and (e), this flag was polled.

d. MAC Initialization

The configuration of the MAC level includes these operations:

- Enable the MAC to receive frames (MACON1=0x05).
- Choose duplex mode (Half-duplex) and auto padding (MACON3=0x30).
- Enable the conformance for the IEE 802, 3 standard (MACON4=0x40).
- Configure the maximum frame length → typically 1518 =0x05EE.
(MAMFLH=0x05; MAMFLL=0xEE).
- Configure the Back-to-Back Inter-Packet Gap Register (MABBIPG=0x12)
- Configure the Non-Back-to-Back Inter-Packet Gap Register (MAIPGH=0x0C;
MAIPGL=0x12)
- Configure the Retransmission and collision windows (Default value)
- Program the local MAC address (00:04:A3:01:01:01 → (MAADR1-MAADR6)

e. PHY Initialization

Although some registers of this level are configured with the external circuitry, some changes can be done in this level are as follows:

- Configure the control of the Ethernet Led's by PHLCON register, for example to display link status, collision, receive or transmit activity. In this project, one of the led is configured to display the receive activity and the other the transmit activity (PHLCON=0x0912).
- Avoid automatic loopback when half duplex is used (PHCON2= 0x0100).

5.3.5 ECN28J60_.c

In this file the ECN28J60 memory reading and writing functions based on the SPI instruction set (see subsection 4.5.3) are coded. The C# code of this file is located in Appendix 5.

As it was mentioned in section 4.3, Microchip provides all the needed files for the implementation of the TCP/IP stack in PIC microcontroller. Evidently, Atmel's microcontrollers do not work although on the internet there is an adaptation for an AVR microcontroller from Atmel (Atmega168). In this project, same a new TCP/IP stack, was coded from the beginning.

The SPI commands can be easily identified since the functions defined in this file have the same name as they have. As the SPI commands were previously explained, the functions are illustrated by flowcharts or pseudo code. In fact, they can be classified according to the kind of memory used (control registers, PHY register or Buffer).

a) Functions used by the control register.

- **int BankSelect (int BankSelect)**

This function is used to change to one of the three banks of the Control Register. As the flowchart in Figure 67 illustrates, if the bank number is bigger than 3 the function returns FALSE. The bank selection is made by the ECON1 register and the BFC and BFS commands.

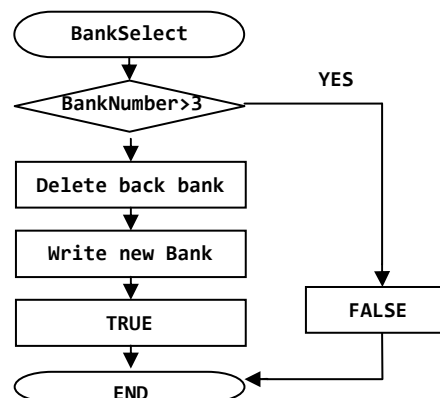


Figure 67. Flowchart for the BankSelect Function

- **void BitFieldClear (u08 Address, u08 Data)**
- **void BitFieldSet (u08 Address, u08 Data)**

Both functions use the same algorithm described by the flowchart in Figure 68 but evidently the opcode is different. As it is observed in the algorithm of below, before writing the data into Transmit Data Register (TDR), it is necessary to wait until the register is empty. Moreover, after sending the data on the MOSI line, it is necessary to empty the Receive Data Register (RDR) to avoid an overrun error.

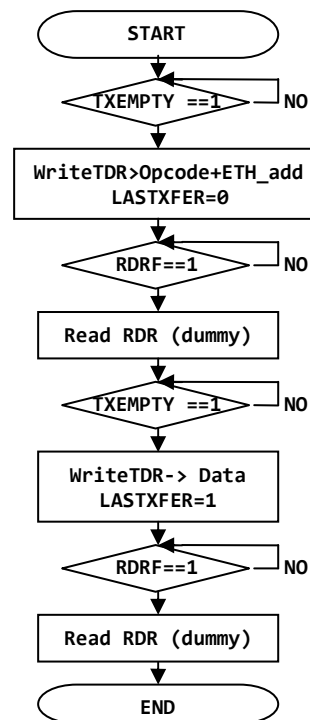


Figure 68. Flowchart for the BitFieldSet and BitFieldClear Functions

- **WriteCtrReg (u08 Address, u08 Data)**

This function uses the same algorithm seen in Figure 68 but there are some differences apart from a different opcode. These differences are related to the address parameter passed to the function.

First of all, the address can belong to any kind register of the control register's memory (ETH, MAC or MII) and secondly the address is checked, so if the address is larger 0x1F, the function returns a FALSE value.

- u16 ReadCtrReg (u08 address, u08 BankNumber)

The flowchart in Figure 69 summarizes the operations carried out to read any control register.

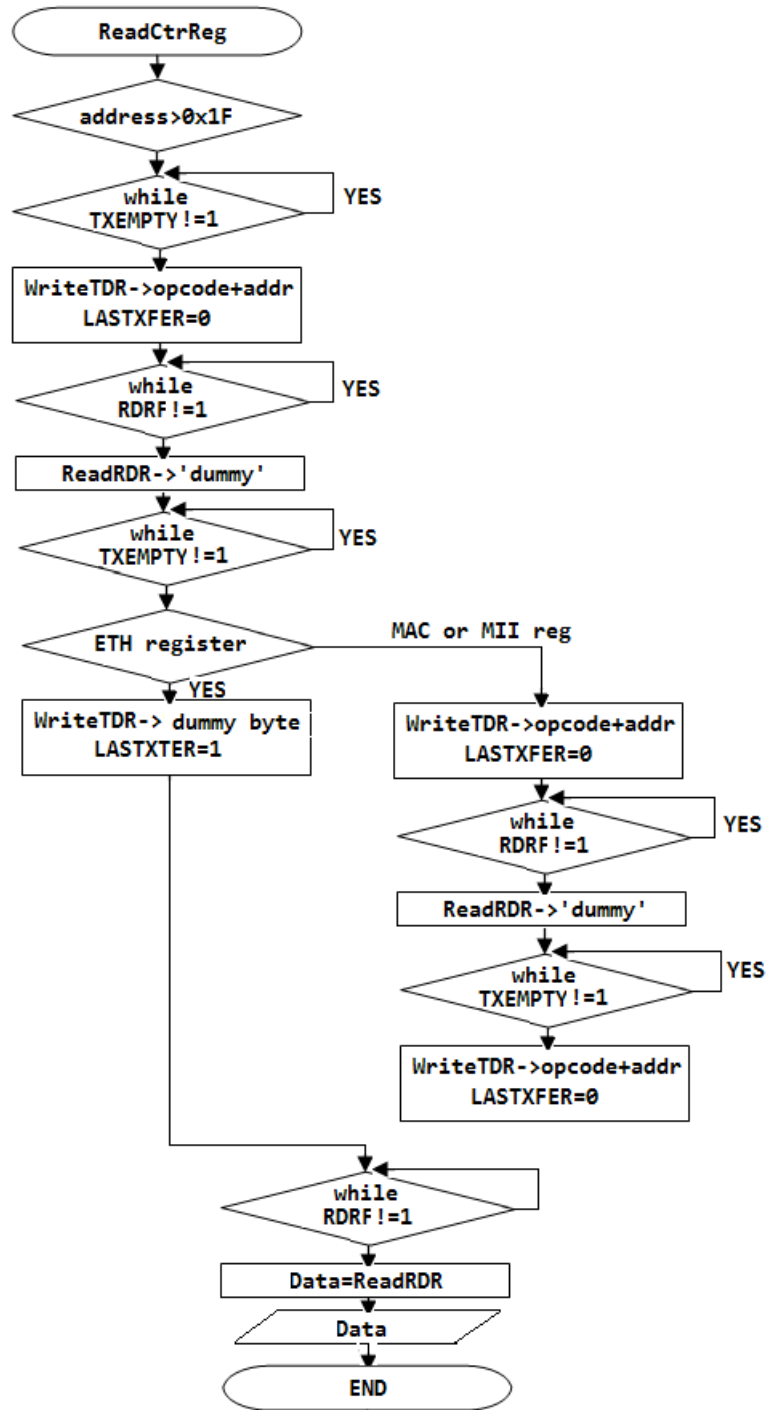


Figure 69. Flowchart for the ReadCtrReg Function

- **void SystemResetCommand (void)**

Apart from carrying out a software reset, this function enables the timer to interrupt in order to start the waiting time.

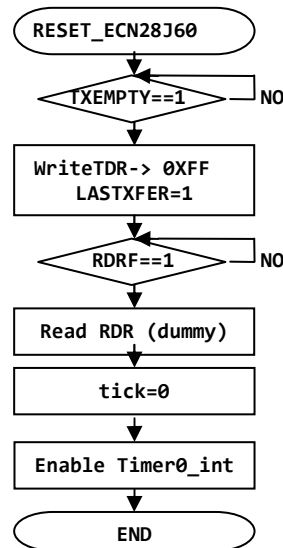


Figure 70. Flowchart for the SytemResetCommand Function

b) Functions used by the PHY registers.

As it was commented on earlier, the functions that allow reading and writing in the PHY registers use the MII registers because they can not be accessed directly by SPI port. For this reason, to understand faster the algorithm of these functions, the pseudo code is used instead the flowchart.

- **u16 ReadPhyReg (u08 address)**

Begin

BankSelect(2);

Write PHY address register into MIREGADR register.

Write the low part of the data into MIWRL register.

Write the high part of the data into MIWRH register.

End

- **int WritePhyReg (u08 address, u16 data)**

Begin

Bankselect (2);

Write PHY address in the MIREGADR register.

Set MICMD.MIIRD=1

BankSelect (3);

Do {

Read MISTAT register

} while (MISTAT.BUSY=0);

Clear MICMD.MIIRD=0

Read the low part of the data into MIWRL register.

Read the high part of the data into MIWRH register.

End;

c) Functions used by the Ethernet buffer.

In the same manner as with the functions used by the PHY registers, the used functions by the buffer are explained with pseudo code. As the Ethernet buffer has to be split in to a transmit and a receive buffer, each case is individually boarded.

◆ *Packet reception [28, 43-44]*

Before receiving any data packet from computer, the MAC level and the receive filters have to be configured. Since it was not known when an ARP or ICMP packet will be received, the external interrupt of the ECN28J60 was configured. The ECN28J60 has seven interrupt sources configured by the EIE and EIR control registers. In this project, the receive packet pending interrupt was only used.

So when a packet is received, an external interrupt occurs. In the interrupt service routine (Appendix 2), there is a function call to read the data received.

The Ethernet frame is written after six bytes (Next Packet Pointer and RX status vector [28, 44]) as can be seen in Figure 71.

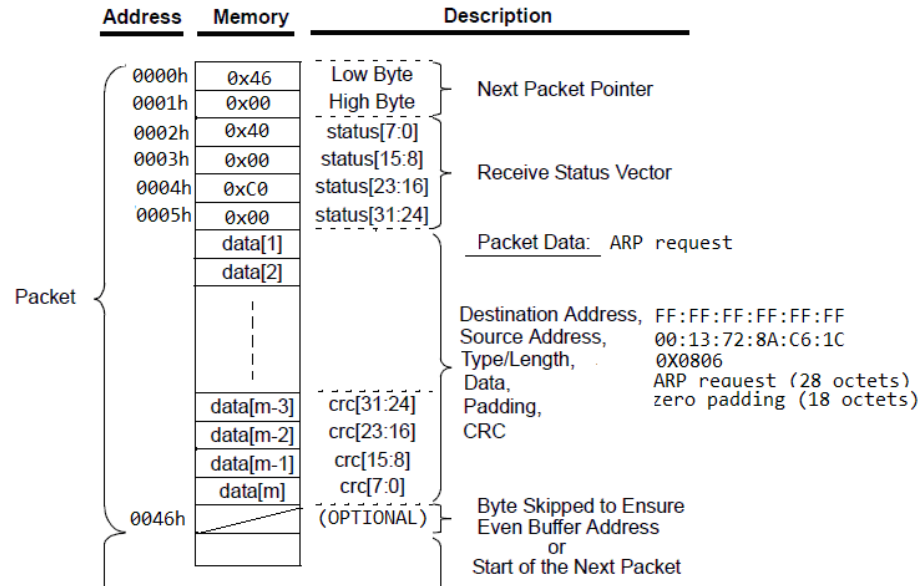


Figure 71. Content of the Receive Buffer after receiving an ARP request packet

Once the packet is written into the receive buffer, it remains there until the buffer space is freed (ERXRDPT has to be reprogrammed). The content of the receive buffer is read by two coded functions. The algorithm for the read the buffer is described below.

- **void ReceivePacket (void)**

Begin

Save NextPacketPointer to update the ERDPT later.

Save length of the received frame (two first octets of status vector).

Ensure ERXRDPT address has to be odd to avoid corrupt circular FIFO.

Read the contained of the Ethernet frame -> ReadBufferMem function call

ReadBufferMem(length);

Begin

Write RBM opcode on the MOSI line (0x3A).

Save Ethernet header (Destination and source address and type).

What kind of frame is?

if (type=IP) then

Read the type of IP protocol

if (protocol = ICMP) then

Read kind of ICMP protocol

if (ICMP protocol = ICMP request)

Send ICMP reply

endif

endif

endif

if (type=ARP) then

Read the type of ARP opcode

if (ARP opcode = request & dest IP = 192.168.0.2) then

Send ARP request

elseif (ARP opcode = reply) then

Send ICMP request

endif

End_ ReadBufferMem

Update ERDPT pointer with the NexPacketPointer value.

Freeing receive buffer (optional)

End_ ReceivePacket

◆ *Packet Transmission [28, 39-42]*

Before sending any data packet from ECN28J60, this packet has to be written first into the transmit buffer using the adequate SPI commands. First, an octet (Per packet control) is written and then the data packet. After that, the hardware of the ECN28J60 will write automatically a 7 byte status vector defined in [28, 41].

The algorithm carried out to transmit a packet is described below. It is composed of two functions. The parameters used by the TransmitPacket function are passed to WriteBufferMem function during the call.

- **int TransmitPacket (ETHframe macframe, int DataLen)**

Begin

Write ETXST pointer in an even address

Write EWRPT pointer at the beginning of the transmit buffer.

Write Ethernet Frame into the transmit buffer → function call.

address=WriteBufferMem (macframe, DataLen);

Begin

Write WBM opcode on the MOSI line (0x7A).

Write PerPacketControl (0x00).

Write macframe (Destination add, source add, type and data)

return EWRPT value

End_ WriteBufferMem

Keep the current address into ETXND.

Reset transmit logic (Set and Clear ECON1.TXRST)

Start transmission (ECON1.TXRTS=1) -> ETH frame is sent to host.

If (ESTAT.TXABRT==1)

The transmission was aborted → return FALSE

End_ TransmitPacket

5.3.6 arp.c

This file contained the part of the TCP/IP stack related with levels 2 and 3. As it was explained in section 4.6 (p.64), in a computer network, the ARP packets exchange is carried out during the first attempts of communication between source and destination.

The file is composed of two functions whose algorithm is based on building the Ethernet packet to send to host later. These functions are explained below to justify such frame construction.

- **ETHframe WriteARPrequest (void)**

This function generates an ARP request packet, if all goes well then the computer answers with an ARP reply packet. Moreover, it assembles the Ethernet packet that will be sent to the host as is illustrates below.

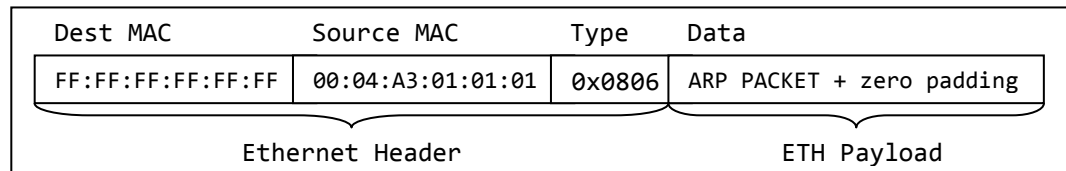


Figure 72. Ethernet Frame Sent by the ECN28J60

Figure 72 is the Ethernet frame that has to be written in the transmit buffer to be sent to the host later.

As at the beginning, the destination MAC is unknown and a broadcast packet is sent. That is, this packet is sent to all the host of the network but only the host with the sought IP answers. The source MAC is the physical address of the ECN28J60, in this project the 00:04:A3:01:01:01 MAC address was chosen for the Ethernet controller, the first three octets belong to Microchip and the others number were chosen by me. But as was discussed in section 4.3 (p.46), for commercial applications, this number has to be unique, so it cannot be randomly chosen.

The value of the type field is 0x0806 because indicates an ARP packet and into of the data field has to go encapsulated the ARP packet. As can be seen in Figure 48 (page 71), an ARP packet has only 28 octets. That is, it is necessary to fill the data field with zero padding to get the minimum Ethernet frame size (60 octets).

This padding can be done manually but in this project the MACON3 register was configured to get an automatic padding. Table 8 on the next page, illustrates the content of the ARP request.

Table 8. ARP Request Packet

ARP Field	Value	BiggerEndian format
htype	0x0001(Ethernet)	0x0100
ptype	0x0800 (IPv4)	0x0008
hlen	0x06 (Eth. size)	0x06
plen	0x04 (Ipv4 size)	0x04
Operation	0x0001 (request)	0x0100
SHA	00:04:A3:01:01:01 (ECN28J60 MAC)	MSB is written first
THA	00:00:00:00:00:00 (ignored for req)	MSB is written first
SPA	192.168.0.2 (ECN28J60 IP)	MSB is written first
TPA	192.168.0.3 (Host IP)	MSB is written first

Notice that the protocols managed in a TCP/IP stack are in bigger endian so the data pertinent to the stack have to be written in this format.

- **ETHframe WriteARPreply (ETHframe rxframe)**

This function generates an ARP reply packet which is sent when a previous ARP request packet has been received. In such a way this function assembles the Ethernet packet with the data of the received packet because the only parameter unknown is the physical address of the ECN28J60. Evidently, although the data managed are the same as in the ARP request, the source and destination are exchanged.

(The code for both functions can be found in Appendix 8 of this thesis).

5.3.7 icmp.c

This file contained the part of the TCP/IP stack related with IP levels. As it was explained in section 4.6 (p.64), a ping command in a computer networks is based on the ICMP packets exchange. This file is composed of two functions. The algorithm is patterned on the ARP packets, so the functions of this file are explained in the same way.

- **ETHframe WriteICMPrequest (ETHframe rxframe)**

This function generates an ICMP request packet, if the firewall of the host allows the reception of ICMP packets, then the computer answers with an ICMP reply packet. As it was previously said, only the ARP packets exchange guarantees the perfect communication between two systems of a computer network.

Moreover this function assembles the Ethernet packet that will be sent to the host which is shown below.

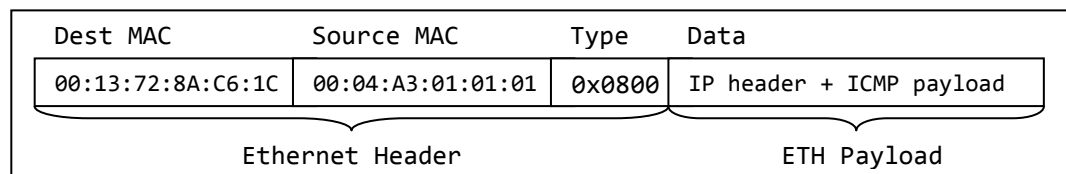


Figure 73. Ethernet Frame Sent by the ECN28J60

As can be observed in Figure 73, after receiving the ARP reply from the host, the physical address of the computer is known. In this case, the type of the frame has 0x0800, which indicates that it is an IP frame.

The other difference in the ARP packet is the data field. As it was explained in section 4.6 (p.64), an ICMP packet consists of an IP header and an ICMP payload. The next page shows the content of the ICMP packet which will be encapsulated in the data field of the Ethernet frame (see Table 9). In this case the auto padding is not necessary because the minimum size is achieved.

As can be observed in Table 9, the first octet is formed by the field header length and version. Further, the IP datagram has a total length of 64 bytes that is 20 bytes of IP header (without data) + 40bytes ICMP data payload.

The IP checksum is calculated using the data of the IP header in the same way as was explained in Table 6 (p.87). Similarly, the ICMP checksum is calculated, but the ICMP data payload is taken into account

Table 9. Ethernet payload for ICMP request

IP header Field	Value	BiggerEndian format
Header length	5 (20 bytes header)	-
version	4	0x45
Type of service	0x00 (ICMP)	0x00
Total length	0x003C = 60 bytes	0x3C00
identification	0x0101(for example)	0x0101
Flags/offset	0x0000	0x0000
Time to live	128 (typical)	128
Protocol	1 (ICMP)	1
IP checksum	0xB86A	0x6AB8
Source IP	192.168.0.2	MSB is sent first
Destination IP	192.168.0.3	MSB is sent first
ICMP Payload	-----	-----
type	0x08(ICMP request)	0x08
code	0x00 (ICMP)	0x00
ICMP checksum	0x495C(seqnumber=0)	0x5C49
identifier	0x0400 (Windows)	0x0004
Sequence numb	0x0000	0x0000
Data (32 bits)	'a'-'w' letters	MSB is sent first

As to ICMP data, the only parameters that are not constant are the sequence number and the ICMP checksum. Usually, the sequence number has to be incremented by one in each request. But in this project these data are fixed to simplify the algorithm of this function. The reason is that a normal ping uses the command prompt of windows. In other words, the ping from the ECN28J60 to the host has to be coded. However, the ping from the computer to the ECN28J60 is made using the command prompt (e.g. ping 192.168.0.2) because the ECN28J60 will be as another host.

- **ETHframe WriteICMPreply (ETHframe rxframe)**

This function generates an ICMP reply packet which is sent when a previous ICMP request packet has been received. In this way this function assembles the Ethernet packet with the data of the received packet.

In this case, the only parameters that vary from the ICMP reply packet are the ICMP type whose value is 0x00 for the echo reply and the checksum. Unlike the previous function (`WriteICMPRequest`), this function calculates a new ICMP checksum by each received ICMP request packet. Since any data related to the TCP/IP protocols has to be written in bigger endian format, the checksum calculation algorithm created in this project takes into account this issue.

(The code for both functions can be found in the Appendix 9 of this thesis)

5.3.8 main.c

Evidently, this file contains the main program where the different interfaces are configured by the function calls but it will not be discussed in more detail here because the functions used by this file have been described before.

5.4 Problems during the Development of the Ping Application

This section focuses on enumerating the different snags that were presented during the development of the ping application.

Since the configuration of the ENC28J60 is based on sending commands through of the SPI interface, this interface has to work properly before the TCP/IP can be tested. The best way to test the SPI interface is using some kind of device to visualize the SPI signals (CS, CLK, MOSI, and MISO).

At the beginning of this project, a four channel oscilloscope was used. The model used was the LC584A from LeCroy with a bandwidth of 1GHz which is shown in Figure 74.

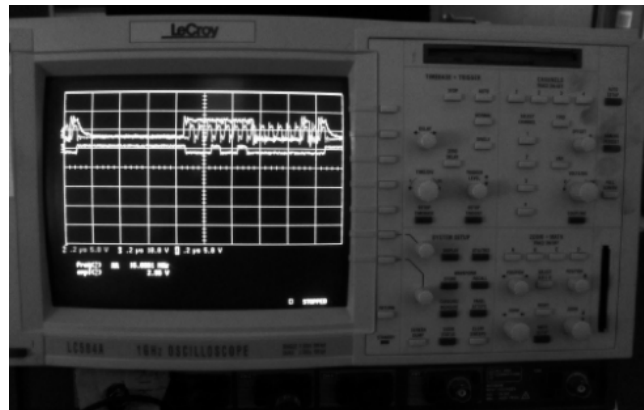


Figure 74. Oscilloscope LeCroy, Model LC584A

Before any SPI signals could be visualized, the coupling of the oscilloscope had to be configured. After doing that, the MISO signal was inactive although the rest of the signals were not because I was taken the ground of evaluation board.

Figure 74 shows the capture of a RCR command where the content of the EREVID is represented. Although, the oscilloscope help you to visualize the SPI signals as can be seen in above figure, this method is quite annoying if you need to know the values of the frames sent on MOSI or MISO lines. For this reason, another way to visualize SPI signals is a logic analyzer.

This device allows visualizing the desired frames in different formats (decimal, binary, hexadecimal, etc) apart from saving the different captures. Moreover, there is the possibility of using a protocol interpreter (I2C, RS-232, SPI, etc). In the market, there are logic analyzers really expensive and complex to use but there are also PC- Logic analyzer cheaper and easier to use than the first one. In this project, a PC-logic analyzer of 34 channels from Intronix (LA1034 model) was used (Figure 75). The software can be downloaded from the website of the manufacturer: <http://www.pctestinstruments.com/index.htm>.



Figure 75. PC-Logic Analyzer from Intronix

Another more important problem occurred with the MAC/MII registers. When an attempt to read any MAC or MII registers was made after writing them, their reset values were always seen but this did not happen with the ETH registers.

Figure 76 is a screen capture taken from the Intronix Logic Port software after writing a 0xFF value in the MACLCON2.

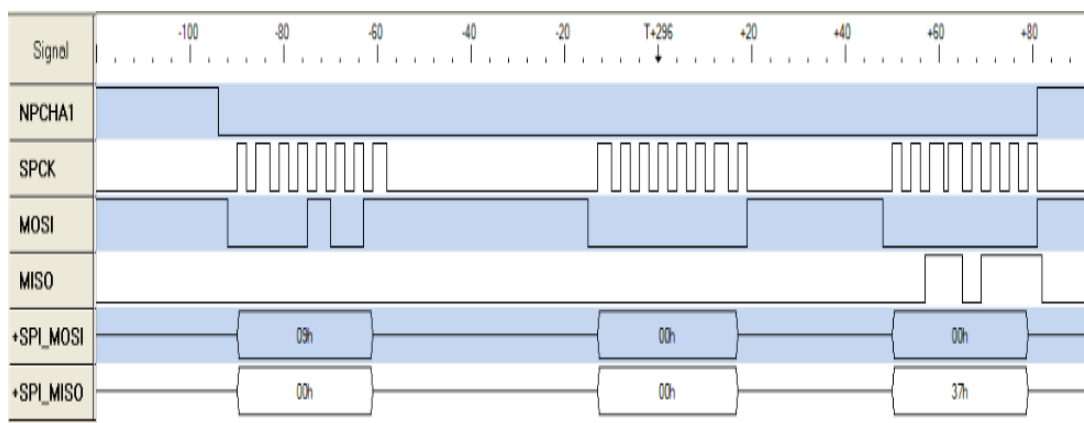


Figure 76. Read of the MACLCON2 register

Unfortunately, one day the PIC Tail started to work in a weird mode because no register be visualized. A possible hardware failure was suspected but the CLKOUT signal output of the ECN28J60 worked whereas no answer was received on the MISO line.

The board could have fixed changing the ECN28J60 chip of the board, but the chip packet is QFN and it is very difficult to unsolder without the proper material so another Ethernet board was acquired.

By change, I found an English website where send Ethernet board based on the ECN28J60 chip (<http://www.hotsolder.co.uk/ethernet-modules-10-c.asp>). This board cost around 20€ and is simpler than the PICTal board because is not designed to be used with other PIC boards. A picture of this board can be seen in Figure 77.



Figure 77. UCEthernet 2 board

The first time that this board was plugged, the problems with the MAC/MII registers happened again. Finally, the problem was solved introducing a little delay between continuous transfers because this field has a zero value by default. In the DLYBCT field of the Chip Select Register of the SPI interface of the microcontroller is configured this delay.

From this moment, the SPI signals could be tested as well as some of the SPI functions programmed in the 'ECN28J60.c' file. But the logic analyzer is not useful for testing functions with a big amount of called to other SPI functions. For example, to check up if a PHY register has been truly written, to observe the ReadPhyReg function, the logic analyzer has to show more than 10 frames and the tracking is tiring. So I had to find another way for testing these functions.

In the C# programming language there is the function 'printf' which shows in the screen computer the desired data. But this kind of functions is complicated to use in embedded systems. So in this project a 'printf' function was coded. For this purpose, the serial port (USART) of the microcontroller was used.

In this way, the content of the any register can be visualized by the USART0 and a terminal program (e.g. HyperTerminal, Real Term, etc). Appendix 5 contains the coded file called usart.c which contains the function of configuration of the USART0 and the 'print' functions used in the debugging of the ping program.

The Atmel's USART interface is really sophisticated since it can works in many different modes, such as asynchronous, synchronous, RS-485, Smart Card protocol (ISO 7816) or Infra-red protocol [26,353-400].

For this project, the configuration of the USART0 is the easiest possible. The USART0 operates in asynchronous mode at 9600 baud with 8 data bits, 1 stop bit, and no parity. As the 'print' application is designed for the transmission, the USART interrupt is not used. Below some registers used in the USART0 configuration are described.

As in the case of the SPI interface, it is necessary to enable of the PMC clock and configuring the PIO lines used by the USART0. Although the USART0 has five pins (RXD0, TXD0, SCK0, RTS0 and CTS0), for this application the PA5 (RXD0) and PA6 (TXD0) ports are used. The SK0 pin (baud rate clock) is not used in an asynchronous serial application and the RTS0 (request to send) and CTS0 (clear to send) pins are not used in a simple RS-232 connection. Figure 78 illustrates the register where the USART0 is configured. The most important fields are explained below.

Name: US_MR
Access Type: Read-write

31	30	29	28	27	26	25	24
-	-	-	FILTER	-	MAX_ITERATION		
23	22	21	20	19	18	17	16
-	-	DSNACK	INACK	OVER	CLKO	MODE9	MSBF
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

Figure 78. USART Mode Register [26, 386]

- ◆ USART Mode: Normal Mode (USART_MODE =0x0000)
- ◆ Clock Selection: The main clock is chosen (MCK=47923200 Hz) as the baud rate generator clock to the transmitter (UCLKS=MCK).
- ◆ Character Length: 8 bits like the memory registers size of ECN28J60(CHRL=0)
- ◆ Synchronous Mode Select: The parity is not used so this field has to have a value 10X. (e.g. SYNC=100)
- ◆ Number of Stop Bits: 1 stop bit by default (NBSTOP=00).
- ◆ The rest of the bits are programmed with their default values.

Figure 79 shows the content of all the control registers involved in the initialization of the Ethernet Interface (ini_ecn28j60.c).

```

RealTerm: Serial Capture Program 2.0.0.57
ERXSTH: 0x00 |
ERXSTL: 0x00 |
ERXNDH: 0x0F |
ERXNDL: 0xFF |
ERXRDPH: 0x00 |
ERXRPTL: 0x00 |
ERXFCON: 0xB0 |
EPMM0: 0x3F |
EPMM1: 0x30 |
EPMCSH: 0xF7 |
EPMCSL: 0xF9 |
MACON1: 0x01 |
MACON3: 0x32 |
MACON4: 0x40 |
MAMXFLH: 0x05 |
MAMXFL: 0xEE |
MABBIPG: 0x12 |
MAIPGH: 0x0C |
MAIPGL: 0x12 |
MAADR1: 0x00 |
MAADR2: 0x04 |
MAADR3: 0xA3 |
MAADR4: 0x01 |
MAADR5: 0x01 |
MAADR6: 0x01 |
ECOCON: 0x00 |
EREVID: 0x04 |
ESTAT: 0x01 |
ECON1: 0x07 |
ECON2: 0x80 |
PHID1H: 0x00 |
PHID1L: 0x83 |
PHID2H: 0x14 |
PHID2L: 0x00 |
PHLCONH: 0x34 |
PHLCONL: 0x22 |
PHCON2H: 0x01 |
PHCON2L: 0x00 |

Display Port Capture Pins Send Echo Port
Display As
  Ascii [x]
  Ansi
  Hex[space]
  Hex + Ascii
  Half Duplex [x]
  newLine mode
  Invert Data
  Big Endian

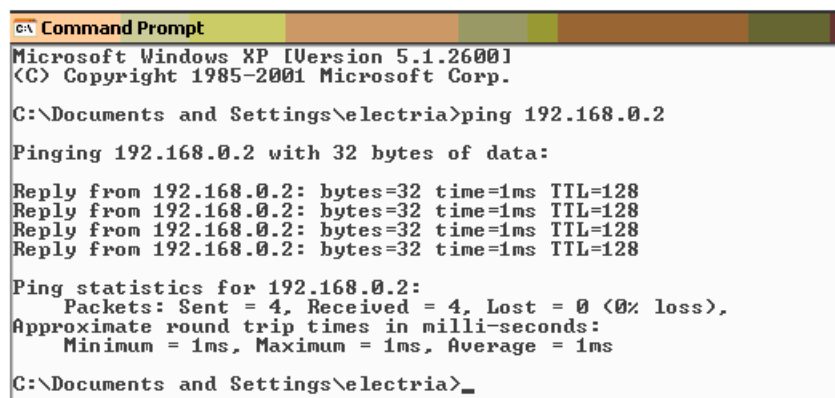
```

Figure 79. Control Registers after the initialization

Once the PHY and MAC levels are configured and the TCP/IP stack has been coded, the last step is testing the ping application. To know what is happening in a computer network, it is convenient to the use of a Network Protocol Analyzer, for example ‘Wireshark’.

At the beginning, the problems related to the part of the TCP/IP stack were caused by the impossibility of writing into transmit buffer. To solve this problem, I coded others 'print' functions with the aim of seeing the content of the buffer memory. These functions are located in the ECN28J60 file (Appendix 7) whose algorithm is not explained because they are debugging functions.

The problem was due to a pointer's mistake. Inside the ECN28J60 buffer you can only use the pointers defined in the control registers, and you can not create a pointer to track the buffer. Another problem was the bad coding of the stack and the use of little endian format in the Ethernet frames. Once all the problems were solved, the ping command was carried out with success. This is shown in Figure80.



```

CA Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\electria>ping 192.168.0.2

Pinging 192.168.0.2 with 32 bytes of data:

Reply from 192.168.0.2: bytes=32 time=1ms TTL=128
Reply from 192.168.0.2: bytes=32 time=1ms TTL=128
Reply from 192.168.0.2: bytes=32 time=1ms TTL=128
Reply from 192.168.0.2: bytes=32 time=1ms TTL=128

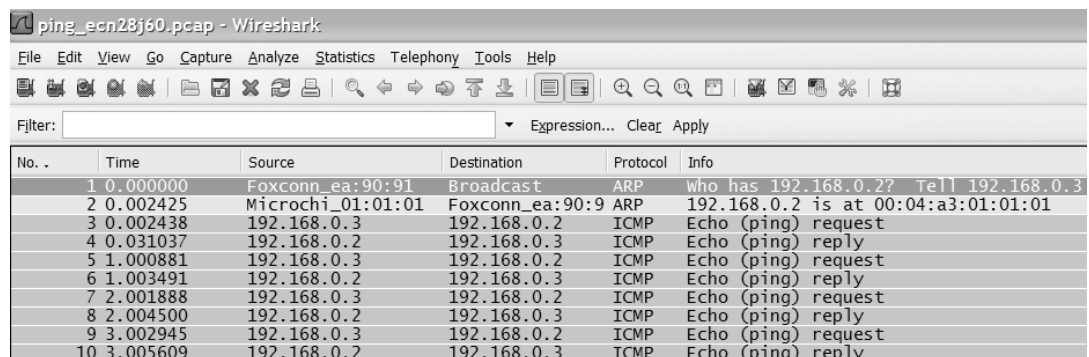
Ping statistics for 192.168.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 1ms, Average = 1ms

C:\Documents and Settings\electria>_

```

Figure 80. Ping command to ECN28J60 IP address

Figure 81 illustrates a 'Wireshark' capture after making a ping from the computer to the Ethernet Controller. As it can be seen, the packet exchange is the same that was described in Figure 49 (page 72).



No. .	Time	Source	Destination	Protocol	Info
1	0.000000	Foxconn_ea:90:91	Broadcast	ARP	who has 192.168.0.2? Tell 192.168.0.3
2	0.002425	Microchi_01:01:01	Foxconn_ea:90:9	ARP	192.168.0.2 is at 00:04:a3:01:01:01
3	0.002438	192.168.0.3	192.168.0.2	ICMP	Echo (ping) request
4	0.031037	192.168.0.2	192.168.0.3	ICMP	Echo (ping) reply
5	1.000881	192.168.0.3	192.168.0.2	ICMP	Echo (ping) request
6	1.003491	192.168.0.2	192.168.0.3	ICMP	Echo (ping) reply
7	2.001888	192.168.0.3	192.168.0.2	ICMP	Echo (ping) request
8	2.004500	192.168.0.2	192.168.0.3	ICMP	Echo (ping) reply
9	3.002945	192.168.0.3	192.168.0.2	ICMP	Echo (ping) request
10	3.005609	192.168.0.2	192.168.0.3	ICMP	Echo (ping) reply

Figure 81. Packet Exchange during ping to ECN28J60 IP address

6 Conclusions

Since the thesis can be read by anyone who has or has not acknowledged in RFID technology, the first chapters of the thesis have focused on explaining the operating principles of a RFID system as well as the elements that compose it. Once some theoretical background has been presented, the reader design can be better approached; the design of UHF RFID reader has been explained.

The main purpose of the project was to explain how to design a low cost reader. When selecting the components, both their characteristics and their price were taken into account (e.g. the Ethernet controller). Also their effect on the final price was considered (e.g. the configuration for the reader antennas)

Moreover, the function of each component (internal and external) involved in a typical commercial UHF RFID reader has been explained and justified. Some examples of manufacturer that design these components have been mentioned.

Lastly, since one of the typical interfaces in a commercial reader is the Ethernet interface. The last chapters have mainly focused on the implementation of this kind of interface in the system.

As it was not in the scope of this thesis to build the UHF RFID reader, the Ethernet interface was only connected to a computer. So the test carried out is based on showing the communication between the Ethernet interface and the host by ping. For this reason, the last chapters focus on the coding of a TCP/IP stack to get a ping between both systems. Once this stack has been understood, a similar TCP/IP stack as well as the other levels could be later coded with less effort.

The appendix section is reserved for the files sources that have been coded for the TCP/IP stack.

7 References

- [1] Landt J. The history of RFID [online]. New Mexico, USA: Los Alamos Nat. Lab; October/November 2005.
URL: http://autoid.mit.edu/pickup/RFID_Papers/008.pdf. Accessed 27 Jan 2010.
- [2] Finkenzeller K. RFID Handbook: Fundamentals and Applications in Contactless Smart Card and Identification. 2nd ed. West Sussex, England: ; 2003.
- [3] Dobkin D. The RF in RFID: Passive UHF RFID in Practice. Massachusetts, USA: Newnes; 2008
- [4] Thornton F, Haines B. RFID Security: Protect the Supply Chain. Massachusetts, USA: Syngress; 2006.
- [5] Rieback, Simpson, Crispo, Tanenbaum. The RFID Threat [online]. March 2006.
URL: www.rfidvirus.org/media/Line56.pdf
- [6] Sweenelly II P. RFID for Dummies. Indianapolis, Indiana: Wiley; 2003.
- [7] Swedberg C. MicroSD Card Brings NFC to Phones for Credit Card Companies, Bank [serial online]. RFID Journal, USA; November 2009.
URL: <http://www.rfidjournal.com/article/view/7224>
- [8] Collins J. RFID for Meat Eaters [serial online]. RFID Journal, New York City, USA; July 2004.
URL: <http://www.rfidjournal.com/article/articleview/1036/>
- [9] EPC Global. Regulatory Status for using RFID in the UHF spectrum. [Online]; March 2009.
URL: http://www.epcglobalinc.org/tech/freq_reg/RFID_at_UHF_Regulations_2009_0318.pdf
- [10] GS1. Global System, Global Standard or Global Solution, (“1” means the number one position in the global standards). [Online]
URL: <http://www.gs1.org/>
- [11] EPC Global. Specification for RFID Air Interface: EPCTM Radio-Frequency Identify Protocols Class-1 Generation-2 UHF RFID, Protocol communications at 860 MHz – 960 MHz. v.1.2.0 [online]; October 2008.
URL: <http://www.epcglobalinc.org/standards/uhfclg2>
- [12] Texas Instruments (JAG). UHF Gen 2 System Overview [online]; March 2005.
URL: http://rfidusa.com/superstore/pdf/UHF_System_Overview.pdf

[13] ETSI EN 302 208-1. Radio Frequency Identification Equipment operating in the band 865 MHz to 868 MHz with power levels up to 2 W, Part 1: Technical requirements and methods of measurement. v.1.3.1 [online]; July 2009.

URL: http://webapp.etsi.org/action/V/V20100206/en_30220801v010301v.pdf

[14] EPC Global. Low Level Reader Protocol (LLRP) Standard LLRP Low Level Reader Protocol. v.1.0.1 [online]; August 2007.

URL:http://www.epcglobalinc.org/standards/llrp/llrp_1_0_1-standard-20070813.pdf

[15] Nikitin P and Rao K. Measurement of Backscattering from RFID Tags. IEEE Xplore. [Serial online] Washington, USA; 2005.

URL: http://www.ee.washington.edu/faculty/nikitin_pavel/papers/AMTA_2005.pdf

[16] Nikitin P. An Overview of Near Field UHF RFID. IEEE [serial online]. Texas, USA; February 2007.

URL: http://www.ee.washington.edu/faculty/nikitin_pavel/papers/RFID_2007.pdf

[17] Hagen J. Radio-Frequency Electronics: Circuits and Applications. 2nd ed. New York, USA: Cambridge; 2009.

[18] Chitode J. Communication Theory. 3rd ed. India: Technical Publication Pune; 2008.

[19] Razari B. RF Microelectronics. Los Angeles: University of California, USA: Prentice Hall; 1998.

[20] Pozas D. Microwave and RF Design of Wireless Systems. University of Massachusset at Amherst, USA: Wiley; 2001.

[21] Turcu C. Development and Implementation of RFID Technology. Croatia: In Tech; January 2009.

[22] Yuan C, Huang K, Li H, Huang Y. The Design of Encoding Architecture for UHF RFID Applications. IEEE Xplore. [Serial online]. Kaohsiung, Taiwan: Department of Electrical Engineering; 2008.

URL:<http://sciencestage.com/d/5836437/the-design-of-encoding-architecture-for-uhf-rfid-applications.html>

[23] Pascal Curty J, Declercq M, Dehoilain C, Joehl N. Design and Optimization of Passive UHF RFID Systems. Ecole Polytechnique Federale de Lausanne, Switzerland: Springer; 2007.

[24] IMPINJ. Indy Family Brochure [online]

URL: <http://www.impinj.com/products/SubTwoToOneCol.aspx?id=3273>

[25]IMPINJ. Indy Family Brochure [online]

URL: <http://www.impinj.com/WorkArea/showcontent.aspx?id=3271>

[26] Datasheet and Information of AT91SAM7s256 microcontroller [online].
URL: http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

[27] Pressman A, Billings K, Morey T. Switching Power Supplies Design. 3rd ed.
USA: Cambridge; 2009.

[28] ENC28J60 Ethernet Controller with SPI Interface [online].
URL (ENC28J60 Datasheet)

<http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf>

URL (Ethernet PICTail Daughter Board)

http://ww1.microchip.com/downloads/en/DeviceDoc/Ethernet%20PICtail%20Info%20Sheet_51569b.pdf

[29] Website of IAR [online].

URL (IAR JLINK): <http://www.iar.com/website1/1.0.1.0/369/1/>

URL (IAR Embedded Workbench): <http://www.iar.com/website1/1.0.1.0/50/1/>

[30] Tanenbaum A. Computer Networks 4th. New Jersey, USA: Prentice Hall;
2003.

[31] Stallings W. Data and Computer Communications 8th. USA: Prentice Hall;
2007.

[32] Nikkel S. How to wire Ethernet Cables [online]

URL: http://www.ertyu.org/steven_nikkel/ethernetcables.html

[33] Lynch J. Using Open Source Tools for AT91SAM7S Cross Development.
[Serial online] Grand Island, New York, USA; May 2007.

URL: <http://www2.amontec.com/sdk4arm/ext/jlynch-tutorial-20061124.pdf>

[34] Atmel. *ARM-based Software Packages* [Serial online]

URL: www.atmel.com/dyn/resources/prod_documents/doc6016.pdf

Appendix 1

```

/*****
* File:      setup_SPImaster.c
* Overview: configuration of the SPI interface in Master mode to manage the ECN28J60
*           and the reader
* Author:    Jesus Chozas Robledo
*****/

#include "board.h"
#include "include/ECN28J60.h"

// CONSTANTS
#define DLYBCS ((unsigned int)0x0A<< 24) //Delay =2Tclk = 0,2us
#define SCBR0  ((unsigned int)0x5 << 8)  //Serial clock baud rate for 10MHz
#define DLYBS0 ((unsigned int) 0x00 << 16) //Default delay= 1/2 SPCK clock period
#define DLYBCT0 ((unsigned int) 0x01<< 24) //Delay =3 clock cycles
/* Defined in SPI.h
#define LAN_SLAVE 1 //It is Chosen the slave 0 for the ECN28J60
#define READER_SLAVE 0 //It is Chosen the slave 1 for the INDY R2000
*/
//External functions
extern void setup_SPImaster.c (void);
extern void IRQ1_Handler (void);

/*****
// Function name: setup_SPImaster
// Description: This function carries out the right configuration of the SPI interface.
//             The requisites are written as per the characteristics of the slave.
// input param: none
// output param: none
*****/
void setup_SPImaster (void)
{
    AT91PS_SPI pSPI = AT91C_BASE_SPI; // Pointer to SPI structure
    AT91PS_PIO pPIO = AT91C_BASE_PIOA; // Pointer to PIOA structure
    AT91PS_AIC pAIC = AT91C_BASE_AIC; // Pointer to AIC structure
    unsigned int mode, confcs1;

    pSPI->SPI_CR=AT91C_SPI_SWRST; //SPI RESET

    //(1) Enable SPI in the PMC
    AT91F_SPI_CfgPMC(); // Enable the Peripheral Clock for SPI

    //(2) Initialization of AT91SAM7s256
    // It assigns the pins that are used at SPI port.
    AT91F_PIO_CfgPeriph (pPIO, (AT91C_PA11_NPCS0|
                                AT91C_PA12_MISO |
                                AT91C_PA13_MOSI |
                                AT91C_PA14_SPCK |
                                AT91C_PA31_NPCS1|
                                AT91C_PA30_IRQ1),
                        0);
}

```



```

// (3) Configuration of Master mode by Mode Register
mode= AT91C_SPI_MSTR | AT91C_SPI_PS_VARIABLE | AT91C_SPI_MODFDIS | DLYBCS;
AT91F_SPI_CfgMode(pSPI, mode);

// Conf_chipSelects
// Ethernet controller --> Chip Select 1 (ECN28J60 Mode0, 0)
confcs1= AT91C_SPI_NCPHA | AT91C_SPI_CSAAT | AT91C_SPI_BITS_8 | SCBR0 | DLYBS0 |
DLYBCT0;
AT91F_SPI_CfgCs (pSPI, LAN_SLAVE, confcs1);

/* Indy R2000 --> Chip Select 0 -- NOT DEFINED YET -- */

// (4) Enable SPI to transfer and receive data (pSPI->SPI_CR.SPIEN=1)
AT91F_SPI_Enable (pSPI);

// Configuration Advanced Interrupt Controller (AIC) registers for external interrupts
// Function IRQ1_Handler is assigned to IRQ1 interrupt
// Set the interrupt source type (external to low level) and priority =4

AT91F_AIC_ConfigureIt(pAIC, AT91C_ID_IRQ1, 4, AT91C_AIC_SRCTYPE_EXT_LOW_LEVEL, IRQ1_Handler);

// Enable the SPI interrupt in AIC Interrupt Enable (0X00000020)
AT91F_AIC_EnableIt (pAIC, AT91C_ID_IRQ1);

// Interrupt Enable Register
/* it is used by the ECN28J60. see ini_ecn28j60.c */
}

```

Appendix 2

```

/*****
* File: IRQ1_Handler.c
* Overview: Interrupt service routine for the external interrupt
* Author: Jesus Chozas Robledo
*****/
#include "include/typedef.h"
#include "include/ecn28j60.h"

void IRQ1_Handler (void)
{
    // Clear the global interrupt before servicing it
    BitFieldClear (EIE, EIE_INTIE);
    // Decrement package count
    BitFieldSet (ECON2, ECON2_PKTDEC);
    BitFieldClear (ECON2, ECON2_PKTDEC);
    // Read Packet (see ecn28j60.c)
    ReceivePacket ();
    // enable global interrupt_
    BitFieldSet (EIE, EIE_INTIE);
}

```

Appendix 3

```

/*****
* File:      timer0.c
* Overview: Configuration of the timer0 to get an interrupt each 1ms
* Author:   Jesus Chozas Robledo
*****/
#include "include/AT91SAM7S256.h"
#include "board.h"

extern void Timer0_IrqHandler (void);

/*****
// Function name: Timer0setup
// Description: The function configures the timer0 interrupt to generate 1ms wait time
// input param: none
// output param: none
*****/
void Timer0Setup(void)
{
    AT91PS_TCB pTCB= AT91C_BASE_TCB;    // Pointer to TC Global Register structure
    AT91PS_TC pTC0 = AT91C_BASE_TC0;    // Pointer to channel 0 Register structure
    AT91PS_PMC pPMC= AT91C_BASE_PMC;    // Pointer to PMC reg. structure
    AT91PS_AIC pAIC = AT91C_BASE_AIC;    // Pointer to AIC data structure

    //Enable clock for timer0
    pPMC->PMC_PCER = (1<<AT91C_ID_TC0);    //enable clock

    // Timer Counter Interface
    pTCB->TCB_BCR = 0; // SYNC trigger not used
    pTCB->TCB_BMR= AT91C_TCB_TC0XC0S_NONE| // external clocks not used
                  AT91C_TCB_TC1XC1S_NONE|
                  AT91C_TCB_TC2XC2S_NONE;

    //Timer Counter Interface
    pTC0->TC_CCR = AT91C_TC_CLKDIS;    // Disable the Clock Counter
    pTC0->TC_IDR = 0xFFFFFFFF;
    pTC0->TC_SR;

    pTC0->TC_CMR = AT91C_TC_CLKS_TIMER_DIV5_CLOCK | //f=46800Hz
                  AT91C_TC_CPCTRG;                // RC Compare resets
                                                    //WAVE=0 Capture Mode

    pTC0->TC_CCR = AT91C_TC_CLKEN;    // enable the clock
    pTC0->TC_RC = 46;                //Value to get 1ms for prescaler 1024
    pTC0->TC_IER =AT91C_TC_CPCS;    //Enable RC compare interrupt

    // Set up the Advanced Interrupt Controller AIC for Timer 0
    AT91F_AIC_ConfigureIt (pAIC, AT91C_ID_TC0, 4, AT91C_AIC_SRCTYPE_INT_LEVEL_
                          SENSITIVE, Timer0_IrqHandler);

    //Timer0 enable is done in the SystemResetCommand function (see ECN28J60.h)
}

```

Appendix 4

```

/*****
* File:      timer0_IrqHandler.c
* Overview:  Interrupt service routine for Timer0
* Author:    Jesus Chozas Robledo
*****/
#include "include/AT91SAM7S256.h"
#include "board.h"

extern int tick=0; //External variable use to disable and enable the Timer0 interrupt.

/*****
// Function name: Timer0_IrqHandler
// Description: This function makes the interrupt service routine for the Timer0
// input param: none
// output param: none
*****/
void Timer0_IrqHandler (void)
{
    AT91PS_TC pTC0 = AT91C_BASE_TC0; // pointer to timer channel 0 register structure
    AT91PS_PIO pPIO = AT91C_BASE_PIOA; // pointer to PIO register structure

    /* only for debug purpose (pTC0->TC_RC=0xB6CF (1seg))*/
    //Led blinking
    if ((AT91F_PIO_GetInput(pPIO)& LED2) == LED2)
        AT91F_PIO_ClearOutput (pPIO, LED2); // turn LED2 on
    else
        AT91F_PIO_SetOutput (pPIO, LED2); // turn LED2 off

    pTC0->TC_SR; // read TC0 Status Register to clear it

    //Poll if the time = 4ms
    if (tick == 4)
    {
        pTC0->TC_CCR = AT91C_TC_CLKDIS; // Disable the Clock Counter
        pTC0->TC_CCR = AT91C_TC_CLKEN; // Re-enable the clock
    }
    else
        tick++;
}
//end timer0

```

Appendix 5

```

/*****
* File:      usart0.c
* Overview:  This files contains the function for the Configuration of the usart0 as well as
*           two functions to help in the visualization of the content of the memory
*           registers of the ECN28J60. The desired data is sent by serial port to be
*           shown in the screen using a terminal software like Realterm.
* Author:    Jesus Chozas Robledo
*****/
#include "Board.h"
#include "include/typedef.h"

// Function declaration and function prototipes
void      usart0Setup (void);
extern void print      (char Data1[10], u08 Data2);
extern void print16bits (char Data1[10], u16 Data2);
/*****

// Function name: usart0setup
// Description:  The function configures the usart0 interface in asynchronous mode:
//              8 bits, baud rate=115200, 1stopbit, no parity.
//              The usart0 interrupt is not used because I decided when I want to transmit,
//              and the reception is disable.
// parameters:  none
*****/
void usart0Setup (void)
{
    unsigned int mode=0;
    AT91PS_USART pUSART0 = AT91C_BASE_US0; // create a pointer to USART0 structure
    AT91PS_PIO pPIO = AT91C_BASE_PIOA;      //Pointer to PIO structure

//Configuration PIO lines
    pPIO->PIO_PER = AT91C_PIO_PA5 | AT91C_PIO_PA6;
//Assigns peripheral
    AT91F_PIO_CfgPeriph (pPIO, (AT91C_PA5_RXD0 | AT91C_PA6_TXD0),0);
//Configuration USART0
    AT91F_US0_CfgPMC ( ); //USART Clock has to be enabled before use USART0
// Usart Configure
    mode = ( AT91C_US_USMODE_NORMAL| AT91C_US_CLKS_CLOCK|
            AT91C_US_CHRL_8_BITS| AT91C_US_PAR_NONE|
            AT91C_US_NBSTOP_1_BIT | AT91C_US_CHMODE_NORMAL);
    AT91F_US_Configure (pUSART0, MCK, mode, 9600, 0);
//Reset and enable of transmitter
    AT91F_US_ResetTx (pUSART0);
    AT91F_US_DisableRx (pUSART0);
}
/*****
// Function name: print
// Description:  This function makes the function of printf of C# but for ARM7. Basically,
// it is used to show the value of memory registers of the ECN28J60.
// input param: -Data1 is used to write the name of the register.
//              -Data2 contains the hexadecimal value to show (8 bits).
// output param: none
*****/

```

```

void print (char Data1 [10], u08 Data2)
{
  AT91PS_USART pUSART0=AT91C_BASE_US0;
  static int i;
  char* pchar=Data1;
  u08 hex[4];

  //send the data1 (name of register to visualize)
  do{
    //Wait until Transmission is ready
    while (!(pUSART0->US_CSR & AT91C_US_TXRDY));
    pUSART0->US_THR = *pchar; //Send character on the
    pchar ++;
  }while (*pchar != ' ');

  /* Show content of the register (Data2) in ASCII format (0x--)
  I showed the numbers like their ASCII character, example
  45h = '4' and '5' --> Conversion of the value to its ASCII character */

  hex[0]='0';
  hex[1]='x';
  hex[2]= (Data2 & 0xF0)>>4;
  hex[3]= Data2 & 0x0F;

  for (i=0; i<4; i++)
  {
    if(hex[i]<= 0x09)
      hex[i] += 0x30; //Conversion to ascii
    else if (hex[i]>= 0x0A && hex[i]<=0x0F)
      hex[i] += 0x37; //Conversion to ascii

    while (!(pUSART0->US_CSR & AT91C_US_TXRDY)); //Wait until TX ready
    pUSART0->US_THR = hex[i]; //send data
  }
  // Next register will start in a new line
  while (!(pUSART0->US_CSR & AT91C_US_TXRDY));
  pUSART0->US_THR ='\r';
}

/*****
// Function name: print16bits
// Description: It works like print function but visualizes data of 16 bits.
// input param: -Data1 is used to write the name of the register.
//              -Data2 contains the hexadecimal value to show (16 bits).
// output param: none
*****/
void print16bits (char Data1 [10], u16 Data2)
{
  AT91PS_USART pUSART0=AT91C_BASE_US0;
  int i;
  char* pchar=Data1;
  u08 hex[6];

```

```

//send the data1 (name of register to visualize)
do{
    //Wait until Transmission is ready
    while (!(pUSART0->US_CSR & AT91C_US_TXRDY));
    pUSART0->US_THR = *pchar;
    pchar++;
}while (*pchar != ' ');

/* Show content of the register (Data2) in ASCII format (0x--)
I showed the numbers like their ASCII character, example
C45Ah = 'C', '4', '5' and 'A' */
hex[0]='0';
hex[1]='x';
hex[2]=(Data2 & 0xF000)>>12;
hex[3]=(Data2 & 0x0F00)>>8;
hex[4]=(Data2 & 0x00F0)>>4;
hex[5]=(Data2 & 0x000F);

for (i=0; i<6; i++)
{
    if(hex[i]<= 0x09)
        hex[i] += 0x30;
    else if (hex[i]>= 0x0A && hex[i]<=0x0F)
        hex[i] += 0x37;

    while (!(pUSART0->US_CSR & AT91C_US_TXRDY));
    pUSART0->US_THR = hex[i];
}

// Next register will start in a new line
while (!(pUSART0->US_CSR & AT91C_US_TXRDY));
pUSART0->US_THR ='\r';
}

```

Appendix 6

```

/*****
* File:      ini_ecn28j60.c
* Overview:  Initialization of Ethernet Controller ECN28J60
* Author:    Jesus Chozas Robledo
*****/
#include "include/AT91SAM7S256.h"
#define __inline inline
#include "include/lib_AT91SAM7S256.h"
#include "include/ecn28j60.h"

/*****
// Function name: ini_EC29J60
// Description: This function carries out the right configuring of the Ethernet Controller.
// input param: none
// return:      none
*****/
void ini_EC28J60 (void)
{
// (1) Receive Buffer
// Program the ERXST and ERXND Pointers to determinate the buffer length.
BankSelect (0); //It is selected the bank to use.
WriteCtrReg (ERXSTL, (u08) (ERXSTART & 0x00FF));
WriteCtrReg (ERXSTH, (u08) ((ERXSTART & 0xFF00) >> 8));
WriteCtrReg (ERXNDL, (u08) (ERXEND & 0x00FF));
WriteCtrReg (ERXNDH, (u08) ((ERXEND & 0xFF00) >> 8));

//ERDPT (Buffer Read Pointer) has to point to ERXST for tracking purposes
WriteCtrReg (ERDPTL, (u08) (ERXSTART & 0x00FF));
WriteCtrReg (ERDPTH, (u08) ((ERXSTART & 0xFF00) >> 8));

//(2) Transmit Buffer
/* Not explicit action is required to initialize the transmission buffer */

//(3) Receive Filters
BankSelect (1);
WriteCtrReg (ERXFCON, ERXFCON_UCEN| ERXFCON_CRCEN| ERXFCON_PMEN);
WriteCtrReg (EPMM0, 0x3F);
WriteCtrReg (EPMM1, 0x30);
WriteCtrReg (EPMCSL, 0xF9);
WriteCtrReg (EPMCSH, 0xF7);

//(4) Wait for OST (Oscillator Start-up time) before changing MAC/MII registers.
while (!(ReadCtrReg (ESTAT, 0) & ESTAT_CLKRDY));

//(5) MAC Initialization Settings
BankSelect (2);
WriteCtrReg (MACON1, MACON1_MARXEN); //enable MAC to receive frames
WriteCtrReg (MACON3, MACON3_PADCFG0| MACON3_TXCRCEN); //auto padding
WriteCtrReg (MACON4, MACON4_DEFER); //IEEE 802.3
//Maximum Frame length is configured (1518 typical = 0x05EE)
WriteCtrReg (MAMXFLL, (u08) (MAXFRAMELEN)); // the lower 8 bits
WriteCtrReg (MAMXFLH, (u08) ((MAXFRAMELEN >> 8))); //The higher 8 bits

```

```

//Conf. Back-to-Back Inter-Packet Gap -Typically, for half-duplex the value is 12h.
WriteCtrReg (MABBIPG, 0x12); // Data are written in the ERXSTL
//Conf No Back-to-Back Inter-Packet Gap
WriteCtrReg (MAIPGL, 0x12); // Data are written in the MAIPGL (typical value)
WriteCtrReg (MAIPGH, 0x0C); // Data are written in the MAIPGH (typical value)
//MAC Address Configuration (ECN28J60 MAC address= 00:04:A3:01:01:01)
BankSelect (3);
WriteCtrReg (MAADR1, 0X00); //OUI 1 (Microchip)
WriteCtrReg (MAADR2, 0X04); //OUI 2
WriteCtrReg (MAADR3, 0XA3); //OUI 3
WriteCtrReg (MAADR4, 0X01);
WriteCtrReg (MAADR5, 0X01);
WriteCtrReg (MAADR6, 0X01);

//Disable of CLKOUT of the ECN28J60 because is not used like SPI clock
WriteCtrReg (ECOCON, 0x00);

//(6) PHY Initialization Settings
//Conf PHLCON to control the output of LEDA, LEDB
WritePhyReg (PHLCON, 0x0330); //(to display collision status) /*ERRATA FIXED*/
// Conf PHCON1. Mode half duplex with MACON3.FULLDPX=0
WritePhyReg (PHCON1, ~PHCON1_PDPXMD);
//Conf PHCON2. To avoid automatic loopback when TX in half duplex
WritePhyReg (PHCON2, PHCON2_HDLDIS);

} //end ini_ecn28j60

```


Appendix 7

```

/*****
* File:      ecn28j60.c
* Overview: This file contains all the functions used for the configuring of the registers of
*           the ECN28J60 (Controller memory and PHY memory), as well as the
*           Functions to carry out the communication with PC (Ethernet buffer).
* Author:    Jesus Chozas Robledo
*****/
#include "include/AT91SAM7s256.h"
#include "include/ecn28j60.h"
#include "include/tcpipstack.h"

extern void print16bits (char Data1[10], u16 Data2); //DEBUG
extern void print (char Data1[10], u08 Data2);      //DEBUG
extern int tick; // External Variable to manage the timer0

// AT91SAM7s256 SPI_TDR : (SPI Offset: 0xc) Transmit Data Register
// The followings constants are used if Variable Peripheral Select is active (PS=1)
#define PCS ((unsigned int) 0xD) //Peripheral Chip Select-> Select SPI Slave ECN28J60
#define LASTXFER ((unsigned int) 0x1) // Last Transfer-> '1' deactivate Slave after TX

/*****
// Function name: BankSelect
// Description: The function writes the bank selected on the ECON1 reg of the ECN28J60
//              but it doesn't change the other flags value.
// input param: Desired bank number
// return      : TRUE when had not problem
*****/
int BankSelect (u08 BankNumber)
{
    if (BankNumber > 3)
        return FALSE;
    // Delete current bank
    BitFieldClear (ECON1, ECON1_BSEL1 | ECON1_BSEL0);
    // Write new bank
    BitFieldSet (ECON1, BankNumber);
    return TRUE;
}

/*****
// Function name: ReadCtrlReg
// Description: The function reads any ETH, MAC, MII of the control register of the
//              ECN28J60.
// input param : register address and bank number to belong to the register
// return      : contained of the register or FALSE is had a problem
*****/
u08 ReadCtrlReg (u08 address, u08 BankNumber)
{
    AT91PS_SPI pSPI = AT91C_BASE_SPI; //Pointer to SPI structure
    u32 RDframe=0; //Received frame from MISO line

```

```

if (address > 0x1F)
    return FALSE; //There is not address of Control Register bigger than 0x1F

while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
pSPI->SPI_TDR= ((PCS << 16)|(address));
while (!(pSPI->SPI_SR & AT91C_SPI_RDRF));
RDframe=pSPI->SPI_RDR;
while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));

if ((BankNumber ==2)|| (BankNumber == 3 && (address <= 0x05 )|| (address == 0x0A)))
{ //The register belong to MAC or MII group
    pSPI->SPI_TDR= (PCS << 16); //Dummy 1 (zeros frame)
    while (!(pSPI->SPI_SR & AT91C_SPI_RDRF));
    RDframe=pSPI->SPI_RDR;
    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR= ((LASTXFER << 24)|(PCS << 16)); //Dummy 2 (zeros frame)
}
else
    pSPI->SPI_TDR= ((LASTXFER << 24)|(PCS << 16)); //Dummy 1 (zeros frame)

while (!(pSPI->SPI_SR & AT91C_SPI_RDRF));
RDframe=pSPI->SPI_RDR;
return ((u08)RDframe);
}
/*****
// Function name: WriteCtrReg
// Description: The function writes a data in an ETH, MAC or MII Control Register
//               of the ECN28J60
// input param: register's address and desired data to write into register
// return:      TRUE if it had not problem
*****/
int WriteCtrReg (u08 address, u16 Data)
{
    AT91PS_SPI pSPI = AT91C_BASE_SPI; //Pointer to SPI structure
    if (address > 0x1F) //ECN28J60 cannot have a control register add bigger than 0x1F
        return FALSE;

    address |= WCR_OP; //the opcode is added to address byte

    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR= ((PCS << 16)| address);
    while (!(pSPI->SPI_SR & AT91C_SPI_RDRF));
    pSPI->SPI_RDR; //Dummy received is not stored but I empty the receive register

    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR= ((LASTXFER << 24)|(PCS << 16)|Data);
    while (!(pSPI->SPI_SR & AT91C_SPI_RDRF));
    pSPI->SPI_RDR;

    return TRUE;
}

```

```

/*****
/ Function name: SystemResetCommand
/ Description: The function allows a software reset in the ECN28J60
/ input param: none
/ return: none
*****/
void SystemResetCommand (void)
{
    AT91PS_SPI pSPI = AT91C_BASE_SPI; //Pointer to SPI structure
    AT91PS_TC pTC0= AT91C_BASE_TC0; //Pointer to Timer0 reg

    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR=((LASTXFER << 24)|(PCS << 16)|(SRC_OP));
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
    pSPI->SPI_RDR;
    //After reset wait at least 2 ms /*ERRATA FIXED*/
    tick=0;
    pTC0->TC_CCR = AT91C_TC_SWTRG; // enable the clock
}

/*****_*_*_*_*_*_*_*_* EXCLUSIVE FUCTIONS FOR ETH REGISTERS *_*_*_*_*_*_*_*_*/
/*****
/ Function name: BitFieldSet
/ Description: It does a bitwise OR operation between the contained of the addressed
// register with the supplied data.
/ input Param: ETH reg's address and data to make the OR.
/ return: none
*****/
void BitFieldSet (u08 Address, u08 Data)
{
    AT91PS_SPI pSPI = AT91C_BASE_SPI; //Pointer to SPI structure

    Address |= BFS_OP; //the opcode is added to address byte
    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR= ((PCS << 16)| Address);
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
    pSPI->SPI_RDR;

    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
    pSPI->SPI_TDR= ((LASTXFER << 24)|(PCS << 16)|Data);
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
    pSPI->SPI_RDR;
}

/*****
/ Function name: BitFieldClear
/ Description: It does an bitwise NOTAND operation between the contained of the
// addressed register with the supplied data. Supplied data is inverted (NOT)
// and the result is bitwise AND with the addressed register content.
/ Input param: ETH reg's address and data to make the NOTAND
/ return: none
*****/
void BitFieldClear (u08 Address, u08 Data){
    AT91PS_SPI pSPI = AT91C_BASE_SPI;

```

```
Address |= BFC_OP;

while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
pSPI->SPI_TDR= ((PCS << 16)| Address);
while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
pSPI->SPI_RDR;
while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY));
pSPI->SPI_TDR= ((LASTXFER << 24)|(PCS << 16)|Data);
while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
pSPI->SPI_RDR;
}
/*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_**/
EXCLUSIVE FUCTIONS FOR PHY REGISTER
/*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_**/
/*****
// Function name: WritePhyReg
// Description: It is necessary doing the following steps:
//                1) Write address of phy reg to MIREGADR.
//                2) Write lower 8 bits of data to MIWRL.
//                3) Write upper 8 bits of data to MIWRL.
// Param:        PHY address and data to be written in this address
// return:       TRUE is OK.
*****/
int WritePhyReg(u08 address, u16 data)
{
    if (address > 0x14)
        return FALSE; //There isn't PHY address bigger than 0x14h
    BankSelect (2);
    WriteCtrReg (MIREGADR, address);        // Write address of Phy reg
    WriteCtrReg (MIWRL,(u08)data);         // Lower PHY data are written
    WriteCtrReg (MIWRH,((u08)(data >>8))); // Upper PHY data are written
    return TRUE;
}
/*****
// Function name: ReadPhyReg
// Description: No direct access allowed to phy registers so the following process must
//                take place.
//                1) Write address of PHY reg to read from into MIREGADR.
//                2) Set the MICMD.MIIRD bit to start read operation
//                3) Wait 10,24us and Poll MISTAT.BUSY bit.
//                4) Clear the MICMD.MIIRD bit.
//                5) Read data from MIRDH and MIRDH reg.
// Param:        PHY address to be read
// return:       content of the PHY address
*****/
u16 ReadPhyReg(u08 address)
{
    u16 Data=0X0000;
    u08 bytStat;

    /*1*/ BankSelect(2);
        WriteCtrReg(MIREGADR, address);

    /*2*/ BitFieldSet(MICMD, MICMD_MIIRD);
```



```

data=(u08)pSPI->SPI_RDR;

if (i>=0 && i<6)           // destination MAC
    rxframe.dest_add[i]=data;
else if (i>=6 && i<11)    // Source MAC
    rxframe.sour_add[i-6]=data;
else if (i==12)           //Type of frame
    rxframe.type=(u16)data <<8; //save high part of type
else if (i==13)
    rxframe.type|=(u16)data;    //save low part of type
else
    rxframe.data[i-14]=data;
} //for

// As per the kind of frame, I need to do one thing or another.
if (rxframe.type == 0x0800)
{
    //see protocol field (ICMP=1,IP=4,TCP=6,UDP=17, etc)
    if (rxframe.data[9] == 0x01)           // ICMP protocol (60octects)
    {
        //see kind of message ICMP
        if (rxframe.data[20]==0x08)//ICMP REQUEST was received
        {
            macframe= WriteICMPreply (rxframe);
            TrasmitPacket( macframe, 60);
        }
        /*else if (rxframe.data[20]==0x01)           //ICMP REPLY was received*/
    } //if
} //if
else if (rxframe.type == 0x0806)           //an ARP request packet was received (28octect)
{
    ARPopcode=(u16)rxframe.data[6] <<8;
    ARPopcode|=(u16)rxframe.data[7];

    if (ARPopcode == 0x0001)               //ARP REQUEST was received
    {
        if (rxframe.data[24]==192 && //Only the IP assigned to ECN28J60 answer to request
            rxframe.data[25]==168 &&
            rxframe.data[26]==0 &&
            rxframe.data[27]==2)
        {
            macframe= WriteARPreply (rxframe);
            TrasmitPacket( macframe, 28);
        }
    }
    else if (ARPopcode == 0x0002)           //ARP REPLY was received
    {
        macframe= WriteICMPrequest (rxframe);
        TrasmitPacket( macframe, 60);
    }
} //elseif
} //end

```

```

/*****
// Function name: ReceivePacket
// Description: The function is used to receive a packet when an RX interrupt is caused.
// Input param: none
// return : none
*****/
void ReceivePacket (void)
{
    AT91PS_SPI pSPI=AT91C_BASE_SPI;
    static u16 pNextPacket, pRXstart, pRXend, length;
    u08 data;
    static int i;
    //Enabling Reception
    /* BitFieldSet( ECON1, ECON1_RXEN); (Done during the initialization)*/

    //Wait until the transmit data register is empty
    while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY) );
    pSPI->SPI_TDR= ((PCS << 16)|RBM_OP); //the OPCODE=0x3A is sent on the MOSI line
    while (!(pSPI->SPI_SR & AT91C_SPI_RDRF) );
    pSPI->SPI_RDR;

    //- 1st. Save the NextPacketPointer (2 octets)
    //- 2nd. Receive Status Vector (4 octets) -> save Length of the received frame
    for(i=0;i<6;i++)
    {
        while (!(pSPI->SPI_SR & AT91C_SPI_TXEMPTY) );
        if (i<5)
            pSPI->SPI_TDR=(PCS<<16);
        else
            pSPI->SPI_TDR=(LASTXFER<<24 | PCS << 16);
        while (!(pSPI->SPI_SR & AT91C_SPI_RDRF) );
        data=(u08)pSPI->SPI_RDR;
        switch (i)
        {
            case 0: // LSB NextPacketPointer
                pNextPacket=(u16)data;
                break;
            case 1: // MSB NextPacketPointer
                pNextPacket|= (u16)(data <<8);
                break;
            case 2: // RXvector1: LSB frame Length
                length=(u16)data;
                break;
            case 3: // RXvector2: MSB frame Length
                length|= (u16)(data <<8);
                break;
            default:
                break;
        }
    }
}

//Ensure ERXRDPT address has to be odd to avoid corrupt circular FIFO /*ERRATA
FIXED*/

```

```

BankSelect(0);
pRXstart= (u16)ReadCtrReg(ERXSTH,0)<<8;
pRXstart|= (u16)ReadCtrReg(ERXSTL,0);
pRXend= (u16)ReadCtrReg(ERXNDH,0)<<8;
pRXend|= (u16)ReadCtrReg(ERXNDL,0);

if (pNextPacket-1 <pRXstart | pNextPacket-1>pRXend)
{
WriteCtrReg (ERXRDPTL,(u08)(ERXEND & 0x00FF));
WriteCtrReg (ERXRDPTH,(u08)((ERXEND & 0xFF00) >> 8));
}
else
{
WriteCtrReg (ERXRDPTL,(u08)((pNextPacket-1) & 0x00FF));
WriteCtrReg (ERXRDPTH,(u08)((pNextPacket-1) & 0xFF00) >> 8);
}

//2. Read the content of the present packet
ReadBufferMem(length);
// print16bits("NEXT_: ",pNextPacket); //debug

//3. Move ERDPT for next read
BankSelect(0);
WriteCtrReg (ERDPTL, (u08)(pNextPacket & 0x00FF));
WriteCtrReg (ERDPTH, (u08)((pNextPacket & 0xFF00) >> 8));
/*
to free up ENC memory I have to adjust the RX Read pointer (ERXRDPT)
*/
}
/***** TRANSMISSION *****/
// to send a data packet from ECN28J60. First is necessary write this data into
// transmit buffer using the adequate SPI command.
// First a octet (PerPacketControl) is written and then data packet. The Hardware will
// write automatically a 7 byte status vector.
/*****
// Function name: WriteBufferMem
// Description: The function allows the host to write from the 8Kbyte buffer of the
// ECN28J60. If ECON2.AUTOINC=1, ERDPT Pointer is automatically
// increment.
// Input Param: Mac header and length of data payload
// return:
*****/
u16 WriteBufferMem (ETHframe macframe, int datalen )
{
AT91PS_SPI pSPI = AT91C_BASE_SPI; //Pointer to SPI structure
u08 PerPacketControl=0x00; //MACON3 determinate the conf. tx packet
static int i;
u16 ptrbuffer;

while (!(pSPI->SPI_SR) & AT91C_SPI_TXEMPTY);
pSPI->SPI_TDR= (PCS << 16 | WBM_OP); //Opcode=0x7A
while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF);
pSPI->SPI_RDR; //received dummy byte

```



```

//Starting to write into Transmission Buffer
for(i=0;i<15;i++)
{
while (!((pSPI->SPI_SR) & AT91C_SPI_TXEMPTY)) ;
if (i==0)
    pSPI->SPI_TDR= (PCS << 16| PerPacketControl);
else if (i>=1 && i<=6)
    pSPI->SPI_TDR= (PCS << 16| macframe.dest_add[i-1]);
else if (i>=7 && i<=12)
    pSPI->SPI_TDR= (PCS << 16 |macframe.sour_add[i-7]);
else if (i==13)
    pSPI->SPI_TDR= (PCS << 16|(u08)((macframe.type&0xFF00)>>8)); //High part
else if (i==14)
    pSPI->SPI_TDR= (PCS << 16|(u08)(macframe.type & 0x00FF)); //low part

while (!((pSPI->SPI_SR) & AT91C_SPI_RDRF)) ;
pSPI->SPI_RDR; //dummy byte
}
for(i=0;i<datalen;i++)
{
while (!((pSPI->SPI_SR) & AT91C_SPI_TXEMPTY)) ;
if(i < datalen-1)
    pSPI->SPI_TDR= (PCS << 16| macframe.data[i]);
else
    pSPI->SPI_TDR= ((LASTXFER <<24)|PCS << 16|macframe.data[datalen-1]);
while (!((pSPI->SPI_SR) & AT91C_SPI_RDRF)) ;
pSPI->SPI_RDR; //dummy byte
}
//Send pointer position
BankSelect(0);
ptrbuffer= (u16)(ReadCtrReg(EWRPTH,0)<<8);
ptrbuffer|= ((u16)ReadCtrReg(EWRPTL,0));
ptrbuffer--; //Last increment is not needed
return ptrbuffer;
} //end
/*****
// Function name: WriteBufferMem
// Description: The function carries out the transmission of the packet to PC.
// If ECON2.AUTOINC=1, EWRPT Pointer is automatically increment.
// Input Param: Mac header and length of datapayload
// return: TRUE is it was OK.
*****/
int TrasmitPacket ( ETHframe macframe, int DataLen)
{
    u16 address =ETXSTART; //store the actual value of EWRPT

//1) Program ETXST Pointer in a even address that is not being used.
    BankSelect (0); //It is selected the bank to use.
    WriteCtrReg (ETXSTL, (u08)(ETXSTART & 0x00FF));
    WriteCtrReg (ETXSTH, (u08)((ETXSTART& 0xFF00) >> 8));
// Set write buffer to point to start of Tx Buffer
    WriteCtrReg (EWRPTL, (u08)(ETXSTART & 0x00FF));
    WriteCtrReg (EWRPTH, (u08)((ETXSTART& 0xFF00) >> 8));

```

```

//2) Write the data at the buffer memory
// BitFieldClear (ECON2, ECON2_AUTOINC); //(Set by default)
address=WriteBufferMem (macframe, DataLen);

//3) Program the ETXND Pointer to point to the last byte in the data payload
BankSelect (0); //It is selected the bank to use.
WriteCtrReg (ETXNDL,((u08) address & 0x00FF));
WriteCtrReg (ETXNDH,((u08) ((address & 0xFF00)>>8)));

//4) Start the transmission.the contents of the transmit buffer is sent to the network
//-----ERRATA FIXED _TRANSMIT LOGIC
//RESET Transmit logic
BitFieldSet (ECON1, ECON1_TXRST); //ECON1_TXRST=1
BitFieldClear (ECON1, ECON1_TXRST); // ECON1_TXRST=0

//5) Start the transmission
BitFieldSet (ECON1, ECON1_TXRTS); //ECON1_TXRTS=1

// * Transmission finished when ECON1_TXRTS=0
// * Status vector is written from ETXND+1 and interrupt is generate (EIR.TXIF=1)
if(ReadCtrReg(ESTAT, 0) & ESTAT_TXABRT)
{
    printTXbuffer (ETXSTART, 43); //For debugging only
    return FALSE; // the transmissionpacket was aborted
}
return TRUE;
}
/*_*_*_*_*_*_ EXCLUSIVE FUCTIONS FOR DEBUGGING BUFFER MEMORY *_*_*_*_*_*_*/
/*****
// Function name: printTXbuffer
// Description: The function uses the usart0 to visualize the content of TX buffer.
//              similarly to print and print16bits (see usart0.c), the function was only used
//              during the debug of the program join to real term.
// Input params: ptrbuffer -> Indicates where I start to read the TX buffer
//              length -> Indicates how many addresses I want to see
// return:      none
*****/
void printTXbuffer (u16 ptrbuffer, int length)
{
    AT91PS_SPI pSPI=AT91C_BASE_SPI;
    static int i;
    u08 data;

    while (ReadCtrReg(ECON1, 0) & ECON1_TXRTS); //Wait until data has been transmitted
    BankSelect(0);
    WriteCtrReg (ERDPTL, (u08)( ptrbuffer & 0x00FF));
    WriteCtrReg (ERDPH, (u08)((ptrbuffer & 0xFF00) >> 8));

    while (!(pSPI->SPI_SR) & AT91C_SPI_TXEMPTY);
    pSPI->SPI_TDR= (PCS << 16| RBM_OP); //send opcode=0x3A
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF);
    pSPI->SPI_RDR;

```

```

for(i=0;i<length;i++)          //ETHheader+ETHdata; statusVextor 50
{
    while (!(pSPI->SPI_SR) & AT91C_SPI_TXEMPTY) ;
    if(i<length)
        pSPI->SPI_TDR= (PCS << 16);
    else
        pSPI->SPI_TDR= ((LASTXFER <<24)|PCS << 16);
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
    data=(u08)pSPI->SPI_RDR;
    //Data is send by the serial port to be visualize in a terminal
    print("TXbuffer: ",data);
} //for
} //end
/*****
// Function name: printRXbuffer
// Description: The function uses the usart0 to visualize the content of RX buffer.
// Input params: ptrbuffer -> Indicates where I start to read the RX buffer
//               length -> Indicates how many addresses I want to see
// return:      none
*****/
void printRXbuffer (u16 ptrbuffer, int length)
{
    AT91PS_SPI pSPI=AT91C_BASE_SPI;
    static int i;
    u08 data;

    BankSelect(0);
    WriteCtrReg (ERDPTL, (u08)( ptrbuffer & 0x00FF));
    WriteCtrReg (ERDPTR, (u08)((ptrbuffer & 0xFF00) >> 8));

    while (!(pSPI->SPI_SR) & AT91C_SPI_TXEMPTY) ;
    pSPI->SPI_TDR= (PCS << 16| RBM_OP);
    while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
    pSPI->SPI_RDR; //Read dummy byte

    for(i=0;i<length;i++)
    {
        while (!(pSPI->SPI_SR) & AT91C_SPI_TXEMPTY) ;

        if(i<length)
            pSPI->SPI_TDR= (PCS << 16);
        else
            pSPI->SPI_TDR= ((LASTXFER <<24)|PCS << 16);

        while (!(pSPI->SPI_SR) & AT91C_SPI_RDRF) ;
        data=(u08)pSPI->SPI_RDR;

        //Data is send by the serial port to be visualize in a terminal
        print("RXbuffer: ",data);
    } //for
}

```

Appendix 8

```

/*****
* File: arp.c
* Overview: This file contains the part of the TCP/IP stack related of levels2 and 3
*           to get pinging between the PC and the ECN28J6.
* Author: Jesus Chozas Robledo
*****/

#include "include/typedef.h"
#include "include/ecn28j60.h"
#include "include/tcpipstack.h"
#include <string.h>

const u08 ecn28j60MAC[6]={0x00, 0x04,0xA3,0x01,0x01,0x01};
/*****
// Function name: WriteARPrequest
// Description: This function generates an ARP packet request with the aim of that the PC
// answer with an ARP request. In this case the ping is made from the ECN28J60 to thePC.
// input param: none
// return      : ETHframe -> Frame generated to send to PC
*****/
ETHframe WriteARPrequest (void)
{
    static int i;
    const u08 sourceIP [4] = {192,168,0,2}; // ECN28j60 IP address
    const u08 targetIP [4] = {192,168,0,3}; //host IP address
    ARP_packet arp;
    ETHframe macframe ;

    //1.Fill ARP Packet (28 octets). The configuration is in big endian (MSB first)
    arp.Htype= 0x0100; //Link layer protocol type (Ethernet=1)
    arp.Ptype= 0x0008; //Upper layer protocol which ARP request (Ipv4=0x0800)
    arp.Hlen=0x06;    //Hardware length (eth. size=6)
    arp.Plen=0x04;    //Protocol length (IPv4 size=4)
    arp.oper=0x0100; //Operation that sender is performing (1-request)

    //Hardware origin and destination MAC
    for (i=0; i<6; i++)
    {
        arp.sha[i]= ecn28j60MAC[i]; //Source MAC
        arp.tha[i]=0x00;           //Dest MAC (Ignored for ARPrequest)
    }
    //origin and destination IP
    for (i=0; i<4; i++)
    {
        arp.spa[i]=sourceIP[i];
        arp.tpa[i]=targetIP[i];
    }
}

```

```

//2.Fill the Ethernet FRAME with ARP Packet encapsulated
//Ethernet HEADER      (Dest add, Source add and type)
for(i=0;i<6;i++)
{
    macframe.dest_add[i]= 0xFF;          //ETH Source Address (BROADCAST)
    macframe.sour_add[i]= ecn28j60MAC[i]; //ETH Target Address
}
    macframe.type = 0x0806;              //Type (0x0806=ARP)
//Ethernet Data Payload (max 1500 bytes)
    memset (macframe.data,0,1500*sizeof(u08)); //Initalization of data field to zero
    memcpy(macframe.data, &arp, sizeof(arp));  //Copy ARPpacket (28 octects)
    return macframe;
} //end
/*****
// Function name: WriteARPreply
// Description: This function generates a ARP packet reply with the aim of the PC answers
// with an ICMP request. In this case the ping is made from the PC to
// ECN28J60 by command promt of Windows.
// input param: ETHframe -> Frame received previously (ARP request)
// output param: ETHframe -> Frame generated to send to PC
*****/
ETHframe WriteARPreply (ETHframe rxframe)
{
    static int i;
    ARP_packet ARPreply;
    ETHframe MACframe;

//1.Build the ARP reply with the information of ARP request.
//Link layer protocol type (Ethernet=1)
    ARPreply.Htype=(u16)rxframe.data[1] <<8;
    ARPreply.Htype|=(u16)rxframe.data[0];
//Upper layer protol which ARP request (Ipv4=0x0800)
    ARPreply.Ptype=(u16)rxframe.data[3] <<8;
    ARPreply.Ptype|=(u16)rxframe.data[2];
    ARPreply.Hlen=rxframe.data[4];
    ARPreply.Plen=rxframe.data[5];
    ARPreply.oper=0x0200; //Operation that sender is performing (2-reply)

//Hardware origin and destination MAC
    for (i=0; i<6; i++)
    {
        ARPreply.sha[i]= ecn28j60MAC[i]; //Source MAC
        ARPreply.tha[i]= rxframe.data[i+8]; //Dest MAC
    }
//origin and destination IP
    for (i=0; i<4; i++)
    {
        ARPreply.spa[i]= rxframe.data[i+24]; //sourIP[i];
        ARPreply.tpa[i]= rxframe.data[i+14]; //destIP[i];
    }
}

```

```
//2.Fill the Ethernet FRAME with ARP Packet encapsulated
//Ethernet HEADER      (Dest add, Source add and type)
for(i=0;i<6;i++)
{
    MACframe.dest_add[i]= rxframe.sour_add[i]; //ETH Target Address
    MACframe.sour_add[i]= ecn28j60MAC[i];    //ETH Source Address
}
MACframe.type = 0x0806;                      //Type (0x0806=ARP)

//Ethernet DataPayload
memset (MACframe.data,0,1500*sizeof(u08));
//Copy ARPpacket (28 octects) into data field
memcpy(MACframe.data, &ARPreply, sizeof(ARPreply));

return MACframe;
} //end
```

Appendix 9

```

/*****
* File: icmp.c
* Overview: This file contains the part of the TCP/IP stack related of IP level. To get
*           a ping between the PC and the ECN28J6.
* Author: Jesus Chozas Robledo
*****/
#include "include/typedef.h"
#include "include/ecn28j60.h"
#include "include/tcpipstack.h"
#include <string.h>

// function declaration
extern void print16bits (char Data1[10], u16 Data2); //Debug
/*****
// Function name: WriteICMPrequest
// Description: This function generates an ICMP packet request with part of the
//              information received previously.
// input param: ETHframe -> Received frame during ARP reply
// output param: ETHframe -> Generated frame to send to the PC
*****/
ETHframe WriteICMPrequest (ETHframe rxframe)
{
    static int i;
    u08 ICMPdata='a';
    ICMPPacket icmpacket;
    ETHframe macframe;

// The configuration is in little endian _(LSB first) but the protocol is bigger endian

    icmpacket.IPheader.headerLength=5; //20 bytes header without options or data
    icmpacket.IPheader.version=4; //IPv4-> 4

    icmpacket.IPheader.type_service=0x00; //for ICMP
    icmpacket.IPheader.lenght=0x3C00; // lenght 60 bytes=20Ipheader+40ICMP
    icmpacket.IPheader.ident=0x0101;

// icmpacket.IPheader.flags=0x0000; //Flags are the 3 MSB, the rest belong to offset
    icmpacket.IPheader.offset=0x0000; // Offset has the 5 bits LSB,
    icmpacket.IPheader.timetolive=128; //Typical TTL for ICMP request
    icmpacket.IPheader.protocol=1; //1 for ICMP, 6 TCP, 7 UDP...
    icmpacket.IPheader.checksum=0x6AB8; //Checksum of the IP header 0xB86A

    for (i=0; i<4;i++)
    {
        icmpacket.IPheader.sourceIP[i]=rxframe.data[i+24]; //sourceIP[i];
        icmpacket.IPheader.destIP[i]=rxframe.data[i+14]; //destIP[i];
    }
    icmpacket.Payload.type=0x08; //message type REQUEST
    icmpacket.Payload.code=0x00; //zero for ICMP
    icmpacket.Payload.checksum=0x5C49; //ICMPchecksum calculate for seqnumber=0
    icmpacket.Payload.identifier=0x0004; //For Windows O.S this value is fixed (0x0400)
    icmpacket.Payload.seqnumber=0x0000; //e.g. 0 (it has to be increment by 1 each time)

```

```

for(i=0; i<32;i++) //For make ping, this data field is filling with character 'a'-'w'
{
    icmpacket.Payload.data[i]=ICMPdata;
    if (ICMPdata!='w')
        ICMPdata++;
else
    ICMPdata='a';
}

/* Encapsulation of the IP datagram to the Ethernet Frame */
//Ethernet HEADER      (Dest add, Source add and type)
    for(i=0;i<6;i++)
    {
        macframe.sour_add[i]= rxframe.dest_add[i]; //ETH Target Address
        macframe.dest_add[i]= rxframe.sour_add[i]; //ETH Source Address
    }
    macframe.type = 0x0800; //Type (0x0800=IP)

//Ethernet DataPayload
    memset (macframe.data,0,1500*sizeof(u08)); //Initialize to zero
    memcpy(macframe.data, &icmpacket, sizeof(icmpacket));
    return macframe;
} //end
/*****
// Function name: WriteICMPreply
// Description: This function generate an ICMP packet reply with part of the information
//              received previously.
// input param: ETHframe -> Received frame during ARP request
// output param: ETHframe ->Generated frame to send to PC
*****/
ETHframe WriteICMPreply(ETHframe rxframe)
{
    ETHframe macframe;
    ICMPacket icmpacket;
    static int i;
    static u16 data1=0, data2=0;
    static u32 sum=0;

// The configuration is in little endian _(LSB first) but the protocol is bigger endian
//IP_HEADER
    icmpacket.IPheader.headerLength=rxframe.data[0] & 0x0F;
    icmpacket.IPheader.version=(rxframe.data[0]&0xF0) >>4;
    icmpacket.IPheader.type_service=rxframe.data[1];
    icmpacket.IPheader.lenght= (u16)rxframe.data[3] <<8;
    icmpacket.IPheader.lenght |= (u16)rxframe.data[2];
    icmpacket.IPheader.ident= (u16)rxframe.data[5] <<8;
    icmpacket.IPheader.ident |= (u16)rxframe.data[4];
    icmpacket.IPheader.offset= (u16)rxframe.data[7] <<8;
    icmpacket.IPheader.offset |= (u16)rxframe.data[6];
    icmpacket.IPheader.timetolive=rxframe.data[8];
    icmpacket.IPheader.protocol=rxframe.data[9];
    icmpacket.IPheader.checksum= (u16)rxframe.data[11] <<8;
    icmpacket.IPheader.checksum|= (u16)rxframe.data[10];

```



```

for (i=0; i<4;i++)
{
    icmpacket.IPheader.destIP[i]=rxframe.data[i+12];
    icmpacket.IPheader.sourceIP[i]=rxframe.data[i+16];
}

//ICMP DATAPAYLOAD= ICMP header + data
icmpacket.Payload.type=0x00;           //message type REPLY
icmpacket.Payload.code=rxframe.data[21]; //zero

//ICMPchecksum=ICMPheader+data (same algorithm as the IP checksum)
sum=0;                                //avoid wrong checksum after first calculation
for (i=24; i<59;)
{
    data1=(u16)rxframe.data[i]<<8;
    data1+=(u16)rxframe.data[i+1];
    data2=(u16)rxframe.data[i+2]<<8;
    data2+=(u16)rxframe.data[i+3];
    sum+=(u32)data1+(u32)data2;
    i=i+4;
} //for
data1= (u16) ((sum &0xFFFF0000)>>16);
data2= (u16) (sum &0x0000FFFF);

//The sum is carried out until obtaining a single data of 16 bits
while (sum > 0x0000FFFF)
{
    data1= (u16)((sum &0xFFFF0000)>>16);
    data2= (u16)(sum &0x0000FFFF);
    sum=(u32)data1+(u32)data2;
}
data1= (u16) (sum &0x0000FFFF); //result of sum groups of 16 bits
data1=~data1;                 //one's complement
data2=data1;                  //save the checksum in another register
//bigger-endian
icmpacket.Payload.checksum= ((data1 & 0xFF00)>>8);
icmpacket.Payload.checksum|=((data2 & 0x00FF)<<8);

icmpacket.Payload.identifier= (u16)rxframe.data[25] <<8;
icmpacket.Payload.identifier|=(u16)rxframe.data[24];

icmpacket.Payload.seqnumber=(u16)rxframe.data[27] <<8;
icmpacket.Payload.seqnumber|=(u16)rxframe.data[26];

for(i=0; i<32;i++)
{
    icmpacket.Payload.data[i]=rxframe.data[i+28];
}

```

```
//Ethernet HEADER (Dest add, Source add and type)
for(i=0;i<6;i++)
{
    macframe.dest_add[i]= rxframe.sour_add[i]; //ETH Target Address
    macframe.sour_add[i]= rxframe.dest_add[i]; //ETH Source Address
}
    macframe.type = 0x0800; //Type (0x0800=IP)

//DataPayload (max 1500 bytes)
    memset (macframe.data,0,1500*sizeof(u08)); //Initialize to zero
//Copy ICMPpacket (28 octects) into data field
    memcpy (macframe.data, &icmpacket, sizeof(icmpacket));

return macframe;
}
```

Appendix 10

```

/*****
* File:      main.c
* Overview:  This file contains the main program which calls to the other setup functions
*            and configures both the microcontroller and the ECN28J60.
* Author:    Jesus Chozas Robledo
*****/

#include "include/AT91SAM7S256.h"
#define __inline inline
#include "include/ECN28J60.h"
#include "include/tcpipstack.h"
#include <string.h>
#include "Board.h"  //(only used by evaluation board AT91SAM7s-EK)

extern void LowLevelInit(void);
extern void setup_SPImaster (void);
extern void Timer0Setup(void);

void main( void )
{
    ARP_packet arp;
    ETHframe macframe ;
    AT91PS_AIC pAIC=AT91C_BASE_AIC;

     // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
     // call low-level init - not here - already done from Assembler-Init (see Cstartup.S)
     /* AT91F_LowLevelInit ( ); */

     /* Initialization of the timer0 interrupt
        Timer0Setup ( );

     /* Initialization of the SPI
        setup_SPImaster ( );

     /* Reset of ECN28J60
         // SystemResetCommand ( );

     /* Initialization of the ECN28J60
        ini_ECN28J60 ( );

     //Send frame to the PC and wait response after replying
        AT91F_AIC_DisableIt (pAIC, AT91C_ID_IRQ1);
        macframe =WriteARPrequest ( );
        TrasmitPacket( macframe, sizeof(arp));
        AT91F_AIC_EnableIt (pAIC, AT91C_ID_IRQ1);

        while(1);  //Infinite loop Waiting for ping from PC
    }  //end main

```