2006:048 CIV

MASTER'S THESIS

Enhanced Multi Media Adaptor (EMMA)

Richard Persson
Jordan Vitella

Luleå University of Technology

MSc Programmes in Engineering
Electrical Engineering
Department of Computer Science and Electrical Engineering
EISLAB

2006:048 CIV - ISSN: 1402-1617 - ISRN: LTU-EX--06/048--SE

Enhanced Multi Media Adaptor (EMMA) Project

by

Richard Persson
&
Jordan Vitella

A thesis performed at Ericsson Enterprise AB,
submitted to meet the requirements for a degree in

Master of Science

December 2005

Abstract

The Internet Protocol (IP) is moving communications into a new era of all-IP integrated systems. Enterprises are demanding Voice-over-IP (VoIP) services to save costs as well as to achieve communication convergence. A VoIP gateway is a device that interfaces analog telephones with an IP-based network. The primary functions of a VoIP gateway is the translation of the analog telephone signals into a control signaling protocol as well as the voice packetization. VoIP is an emerging market where System-on-Chip (SOC) is developed. A SOC is a complete system on an integrated circuit (IC) that combines software running on embedded processor cores and hardware intellectual property. Nowadays, vendors offer a set of powerful tools and intellectual property blocks for designing complete and flexible System-on-Programmable-Chip (SOPC) in a very short time.

This work presents the development of a low-cost and small VoIP gateway using the SOPC design methodology, and it also involves the study of different Power-over-Ethernet (PoE) solutions. This appliance is aimed to be used as a network test instrument within enterprise infrastructures. The Altera Cyclone II low-cost FPGA family and Si3210 ProSLIC are selected as the development platforms because this application is targeted to moderate volumes of productions, and also it is not able to afford risks and long design time.

The design flow starts with the definition of a clear system model, followed by hardware and software co-design, and eventually prototyping and verification. This design actually exploits the synergism of hardware and software inside a single FPGA platform. The processes described in the system model are split in hardware and software. Thereafter, dedicated digital hardware blocks are implemented with VHDL to accelerate voice packetization, whereas a C/C++ program implements the main VoIP signaling protocol – Session Initiation Protocol (SIP).

Altera SOPC Builder version 5.0 is employed to create the system architecture. The Nios II soft-processor core and set of on-chip peripherals are put together using a flexible interconnection bus structure – the Avalon Switch Fabric. Then, embedded software is designed to run over this architecture. It is built up and debugged using the Nios II integrated development environment (IDE). In addition, the program comprises of a group of tasks based on the popular MicroC/OS-II real time operating system (RTOS), which makes use of a lightweight TCP/IP stack (lwIP) for handling network applications.

Finally, in order to verify the system functionality, the system is prototyped with the *Altera Nios II Development Board* that has a Cyclone II FPGA and the *Si321xPPT-EVB*. First, the designed architecture is configured into the FPGA, and later the embedded software is also downloaded on the evaluation board. Additionally, a ProSLIC device is employed to interface the SOPC inside the FPGA with an analog telephone. This ProSLIC device contains some other functions, and it is provided by Silicon Labs with a handy evaluation board. The prototype is tested with a SIP-based telephone and a SIP proxy server within the Ericsson Enterprise network. The SIP signaling and the transmission of voice packets is verified and proven correct.

Acknowledgments

Our gratitude to the entire media gateway department at Ericsson Enterprise for the support provided during the development of this challenging project and for the nice time we shared every day during *fika*.

List of Contents

List of Figures

List of Tables

List of Abbreviations

API	-	Application Programming Interface
ASIC	-	Application Specific Integrated Circuit
CPU	-	Central Processing Unit
DHCP	-	Dynamic Host Control Protocol
DSP	-	Digital Signal Processor
DTMF	-	Dual Tone Modulation Frequency
EMMA	-	Enhanced Multi Media Adaptor
FIFO	-	First In First Out
FPGA	-	Field Programmable Gate Array
FSM	-	Finite State Machine
GUI	-	Graphical User Interface
HAL	-	Hardware Abstraction Layer
IC	-	Integrated Circuit
IDE	-	Integrated Development Environment
IETF	-	Internet Engineering Task Force
IMS	-	IP Multimedia Service
IP	-	Internet Protocol
ISR	-	Interrupt Service Routine
JTAG	-	Join Test Action Group
LAN	-	Local Area Network
LE	-	Logic Element
LIM	-	Line Interface Module
LUT	-	Look up Table
lwIP	-	Lightweight TPC/IP Stack
MAC	-	Medium Access Control
OSI	-	Open Systems Interconnection
PBX	-	Private Branch Exchange
PCB	-	Printed Circuit Board
PCM	-	Pulse Code Modulation
PD	-	Powered Device
PoE	-	Power over the Ethernet
POTS	-	Plain Old Telephone Systems
PSE	-	Power Sourcing Equipment
PSTN	-	Public Switched Telephone Network
RISC	-	Reduced Instruction Set Computer
RTCP	-	Real Time Control Protocol
RTP	-	Real Time Transport Protocol
RTOS	-	Real Time Operating System
SDP	-	Session Description Protocol
SIP	-	Session Initiation Protocol
SLIC	-	Subscriber Line Integrate Circuit
SOC	-	System on Chip
SOPC	-	System on a Programmable Chip
TCP	-	Transmission Control Protocol
UA	-	User Agent
UDP	-	User Datagram Protocol
VoIP	-	Voice over the Internet Protocol

1 Introduction

1.1 Trends in enterprise communications and Ericsson solution

There are several important trends occurring in the enterprise communications. One of the major trends indicates that the Internet Protocol (IP) will be used in almost all areas of communication. Enterprises need and expect their various communications systems to be able to interoperate. A common IP-based network enables a multitude of common functions and therefore reduces costs in the form of planning and operation [1]. Indeed, the advances in packet data technology are making possible the convergence of voice, data and video communications. Nonetheless, voice telephony is still the predominant form of communications for business users [2].

Having a slower start than many predicted, IP-based enterprise telephony is gaining ground fast. Enterprises are demanding IP telephony because of its perceived cost savings and future-proof architecture. It is also reinforced with the fact that new multimedia services and applications are easier to implement and manage in a converged infrastructure as the IP-based network. In addition, the increased availability of broadband connections enables the deployment of Voice-over-IP (VoIP) services. Moreover, it is expected an increase of the number of VoIP users in the global fixed line market. As one example, many enterprises today are looking to migrate their current circuit-switched telephony based systems smoothly to IP-enabled, hybrid private branch exchange (PBX) solutions – and ultimately all-IP based systems – to meet their communications needs into the future. [2]

Ericsson Enterprise presented to the market MD110, a multi convergence communication system that truly integrates fixed and mobile telephony, IP phones and IP gateways. MD110 has emerged as a hybrid solution that supports both the traditional circuit-switched technology as well as the innovative packet-switching (IP). Therefore, the system allows reusing as much as possible the exiting hardware inside a company; namely, the Plain Old Telephone Systems (POTS) – which are still broadly used – and the new IP phones can coexist [3]. Nowadays, MD110 is bringing true mobility to the enterprises, embracing technologies such as VoIP, GSM, and 3G, and novel features of mobile devices [4].

From its start, MD110 employed a revolutionary digital architecture based on a series of flexible components called Line Interface Module (LIM) [4]. The communication system is designed as a series of LIMs (figure 1.1), which are distributed and interconnected by other blocks called Group Switch (GS). More, this distributed architecture permits placing new modules wherever it is needed, being it centrally or spreading out over a wide network.

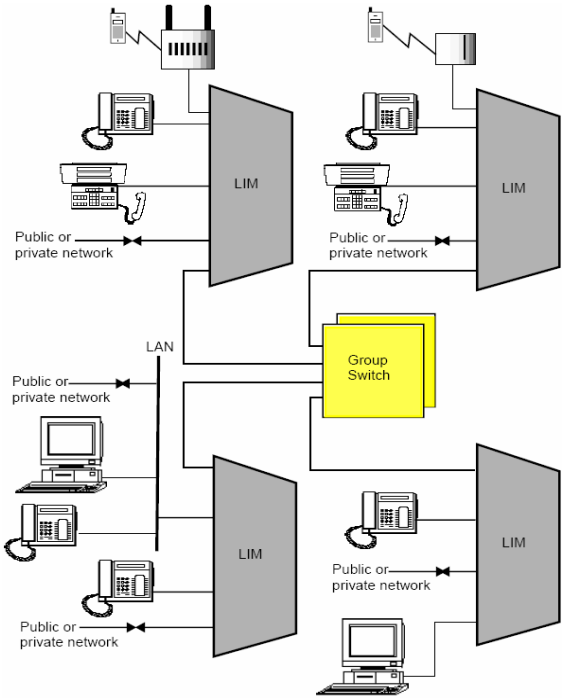

Figure 1.1 ***LIM-based distributed architecture [5]***

Resilience for IP telephony is also guaranteed due to the distributed architecture. MD110 has an integrated support for IP telephony, and it is one of the few IP telephony systems with no single point-of-failure. An alternative path is automatically found on failures in the network. If the network connection to one part of the distributed system is unavailable, or one IP system interface fails, then the IP-phones will automatically discover another system interface and register to it.

Every MD110 LIM is an autonomous system that contains all the functions required for signaling, controlling, and switching of calls to and from line/trunks connected to it. Each one contains a microprocessor-based control system, a nonblocking time switch, tone senders and receivers, and conference devices [6]. It is possible to equip the LIM with an arbitrary mixture of analog extensions lines, digital extension lines and IP trunk lines as well as I/O terminals [7].

1.2 Project motivation

The existing LIMs include boards that allow connecting 16 analog telephones, and thus extra analog boards are required to be included when more devices need to be connected [5]. It entails an extra cost, since each board is relatively expensive. Although the enterprise communications are smoothly trying to migrate to All-IP [8], there are still many analog POTS that are extensively employed. For that reason, the Product Development Unit in Ericsson Enterprise came up with a very interesting way to solve this requirement: Placing a very low-cost and small analog-to-IP adaptor next to each analog telephone in order to connect them to the IP network represents a promising solution (figure 1.2), which also would help to reuse the existing POTS as much as possible without employing costly extra analog boards at the LIM.

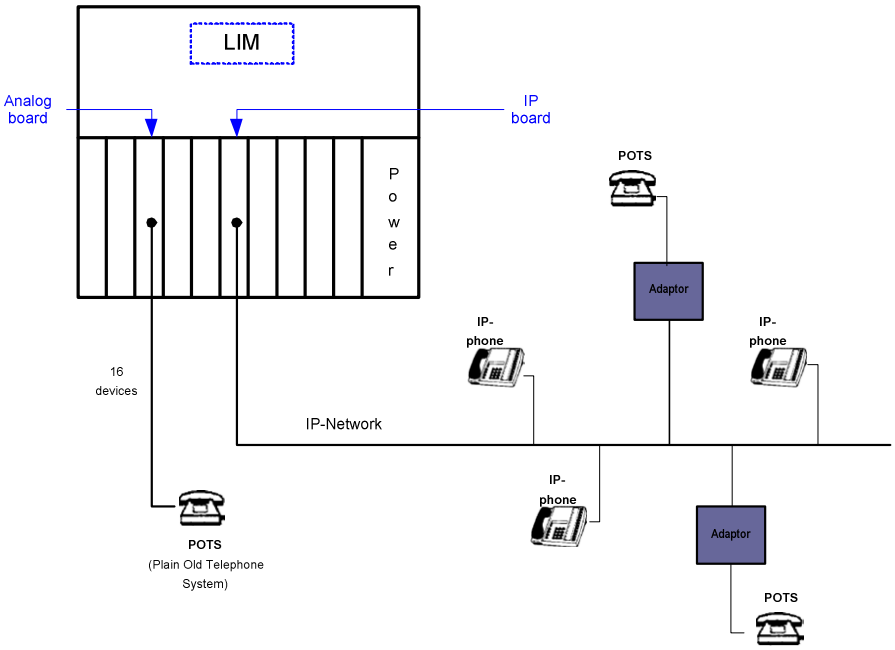

Figure 1.2 ***LIM connecting scheme for analog and IP devices***

This adaptor could also be used as a network test instrument. It would give Ericsson a possibility to analyze the networks before installing new equipment. By doing so Ericsson should be able to foresee some of the faults the customer will experience if/when they switch to a VoIP solution, and give better advices to the costumer about it. A typical scenario for this application would be: "Place a few clients at a customer site for a week. During that week the IP clients will make calls among themselves. After one-week performance test, statistics counters could be fetched from the clients and analyzed by Ericsson".

From the idea mentioned above, the Enhanced Multi Media Adaptor (EMMA) Project was created at Ericsson Enterprise having as its first goal to develop a small analog-to-IP telephone adaptor, which could be placed on a desk and not be considered as "an extra box on the table". It is preferred that the adaptor could be seen as part of a cable from the Ethernet port to the telephone (figure 1.3). Therefore, it implies to design a low power consumption system that could be energized through the network cable employing the Power-over-Ethernet (PoE) standard [9].

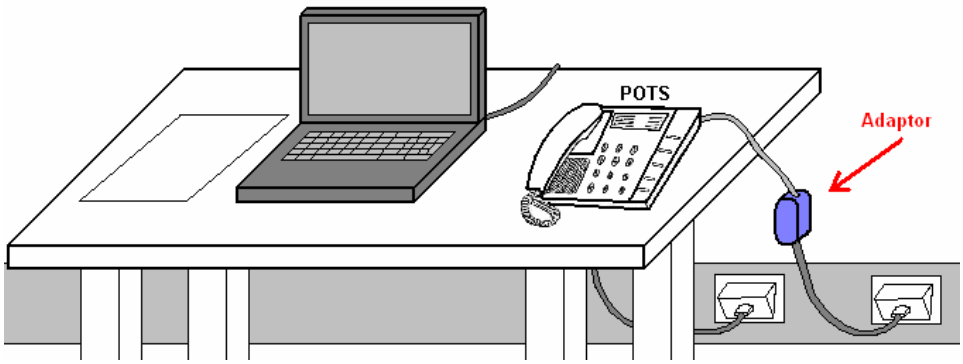

Figure 1.3 ***Description of the Adaptor***

In the beginning, the adaptor is aimed to work as a gateway between the analog POTS and the IP-based network, and later new more features can be added. Thus, it should perform analog signaling with the telephone and establish VoIP protocols with the network simultaneously (figure 1.4). These protocols need to be handled with a processing unit, which should be powerful enough to interpret and produce the messages following the typical logical sequence of a phone call. Besides, it must be able to digitalize and process the analog voice signals for sending them as packets in real time. In order to accomplish these requirements, a complete system that integrates an analog interface, processing units and a network adaptor must be developed. Furthermore, it should avoid using many hardware components since the final appliance has to be very small.

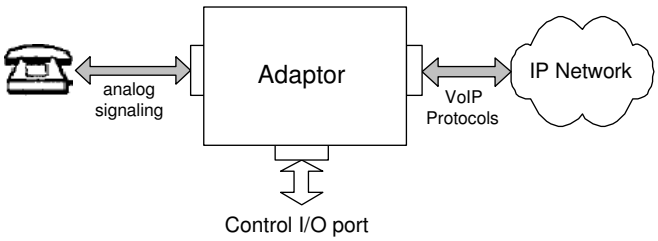

Figure 1.4 ***Functionality of the adaptor as an IP client***

1.3 A System-on-Programmable-Chip solution

The issues described above indicate that employing a System-on-Chip (SOC) for this application is the best solution. In general, a SOC is a complete system on an Integrated Circuit (IC) that combines software and hardware intellectual property using more than one design methodology, with the purpose of defining the functionality and behavior of the proposed system. A typical SOC contains multiple cores such as CPUs, DSPs, memories and peripherals. In most cases, the designed system is application specific.

VoIP applications are good examples of the emerging market where SOC is widely designed. Some examples are Telco Systems Access 241 [10] and Cisco ATA 186 [11], which are stand-alone boxes that provide VoIP functionalities for POTS. A VoIP SOC is a device used for functions such as vocoders, echo cancellation, data/fax modems, and VoIP protocols. Currently, there are a number of these devices – implemented as Application Specific Integrated Circuits (ASIC) – available on the market from several vendors. Typically they differ from each other by the type of functions and voice-processing algorithms they support [12]; among them are Audacity-T2 from Netergy Microelectronics and TNETV1050 from Texas Instrument [13].

While the VoIP products on the market are typically based on ASICs, the Field-Programmable-Gate-Array (FPGA) platforms are becoming very important and widely used in the industry for many applications due to their flexibility and the short design time that they permit. Furthermore, the FPGAs as components are becoming much cheaper and considerably more powerful. It is true that an ASIC achieves better performance and less power consumption than an FPGA for the same specific task, but designing an ASIC is significantly more time demanding. Also, the ASIC design flow is turning out to be even much more complicated and expensive, due to the fact that the transistor technology is shrinking. Designing an ASIC is only worthy when it is targeted

to products that are going to be sold in a very large number, and when special analog I/O analog interfaces are required for a specific design.

The main FPGA vendors, Altera and Xilinx, offer a complete set of embedded processors and Intellectual Property (IP) blocks as well as powerful tools to develop reliable SOCs inside the FPGAs. This makes possible to design fully programmable SOCs, including embedded software, with a very short design time. This methodology is called System-on-Programmable-Chip (SOPC) design. Therefore, the Ericsson EMMA project targets the FPGA as the development platform. In addition, it represents an interesting and promising start point to study the feasibility of implementing products based on fully programmable devices.

Since the success of the EMMA project is price-sensitive, the Altera Cyclone II low-cost FPGA family [14] is selected as the development platform. The cost per thousand of logic elements in small volumes is roughly $0.9, while $0.6 for big volumes. In addition, Altera provides interesting development platforms that permit implementing on this FPGA a wide variety of designs for industrial applications.

1.4 Outline

This thesis is focused on the following:

- Study the requirements for implementing a VoIP gateway and the initial scope of the project, chapter 2
- Describe the features of the device that provides an interface towards the analog POTS, chapter 3
- Design and build a System-on-Chip on an FPGA using the available Intellectual Property blocks, chapter 4
- Design and implement hardware acceleration blocks to handle real-time tasks that cannot meet the requirements by software, chapter 4
- Implement embedded software for the VoIP protocols, chapter 5
- Evaluate the design functionality within an enterprise communication infrastructure, chapter 6.
- Describe the Power-over-Ethernet standard, chapter 7
- Make an estimation of size and cost for building an actual product with the described specifications, chapter 8

2 Voice-over-IP System level overview

2.1 Introduction

IP telephony or VoIP is a technology for transmitting ordinary telephone calls over the Internet using packet-linked routes. A packet-switched network such as the Internet switches data through a network by splitting data into packets that are sent and routed independently; this is an ideal environment for non-voice data. In fact, the quality of VoIP does not yet match the quality of the Public Switched Telephone Network (PSTN).

The goal of VoIP is to provide the efficiency of a packet-switched network while rivaling the quality of a PSTN. For this reason, protocols are developed to manage the network resources more efficiently, thus making the phone calls able to share it in time properly. In this context, a protocol is a set of standards and rules that a machine's hardware and software must follow in order to be recognized and understood by other machine [15]. In the VoIP world, protocol functions are divided into layers; each protocol serves a particular function, allowing better software modularity, as well as system flexibility and extensibility [16].

In general, a gateway is a form of translator to make it possible to call or signal between different standards. A VoIP gateway is a device that converts voice calls - in real time - between the analog POTS and an IP network. The primary functions of a VoIP gateway include the translation of analog telephone signals into a control signaling protocol as well voice coding/decoding and packetization. The purpose of the control signaling protocol is to create, manage and terminate connections between endpoints in the IP network. Next, when the conversation initiates, the analog signal produced by the microphone from the human voice need to be encoded in a digital format suitable for transmission across an IP network [17]. The IP network must then ensure that the real-time conversation is transported across the available resources, achieving an acceptable voice quality.

The following section explains the process that is performed in order to transmit voice through a packet-based network. This process involves different stages that happen sequentially. The Session Initiation Protocol (SIP) [18] is selected as the control signaling protocol due its growing importance in the market and future applications related to communication convergence. An overview of SIP and the requirements for its implementation are provided in the next section.

2.2 Voice Data flow: from analog signal to the network

2.2.1 Voice digitalization

The process of conveying voice through the Internet can be divided in four stages: voice digitalization, encoding, packetization and delivery to the packet-based network (figure 2.3). The transmitted voice flows forwardly through them, while the received voice goes backward. In the first step the analog voice signals generated in the microphone of the telephone handset are sampled 8000 times per second and coded in 64 kb/s stream.

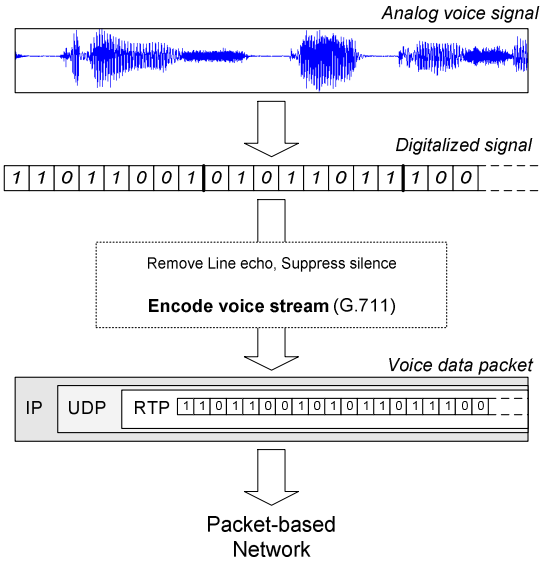

Figure 2.1 ***Basic steps for VoIP***

Consecutively, the digital voice signal is processed. Algorithms for line echo cancellation and silence compression are used. Each telephone conversation includes a lot of silence, and hence, a Voice Activity Detection (VAD) algorithm can be used in order to further reduce bandwidth by eliminating packets that contain silence. Since users are accustomed to "line-noise'' when their peer is not speaking, VoIP terminals can generate background "comfort-noise" to maintain the sensation of a constant stream of noise across the network; Comfort Noise Generation (CNG) can be based either on white-noise hum at a low volume or real noise sampled from the actual call [19].

On the other hand, echo is present in conventional analog POTS networks due to electrical causes, and the IP network also adds some delay that makes the echo problems worse. If the voice delay exceeds 50ms then line echo cancellation is essential for a VoIP gateway solution [20]. The echo from the telephone network can be reduced with digital adaptive filters. These features are quite complex, and their implementation is time demanding. For that reason, this digital signal processing stage is not included in the current version of the EMMA project, but it can be added later.

2.2.2 Encoding

In the second step, the digital voice stream is encoded with a specific algorithm. There are many voice coders (vocoders) in the market based on different algorithms that achieve different bit rates and qualities. A number of factors must be taken into account to select a vocoder, which should be able to achieve an optimal quality with a relatively low bandwidth. Toll quality [21], which is equivalent to the experience with an analog telephone conversation, can be achieved at 64 kb/s with the standard G.711 [22]. The G.711 describes a relatively simple way to encode data by using a semi-logarithmic scale – called the companded Pulse Code Modulation (PCM) – that increases the resolution of small signals, while large signals are treated proportionally. Other voice coders include compression to reduce the 64-kbit/s stream produced by G.711-PCM-encoding stream to a lower bit rate for more efficient transport across the network; among them are G.723.1 and G.729/G.729A [23, 25].

Packets are efficient for data transfer, but are not so attractive for real-time services such as voice. That is where the selection of an optimum voice coder is necessary. In fact, there is ordinarily a trade-off between voice quality and bandwidth used. Using the most efficient codec available today allows quasi-toll [21] quality bandwidth usage to be as low as 5kbps. As newer and more sophisticated algorithms are developed in the future, this figure will decrease, permitting more samples to be squeezed more efficiently while minimally sacrificing quality, if at all. In a fast and reliable enterprise communication infrastructure, G.711 is enough to achieve a good quality without having delays in the network.

2.2.3 Packet preparation

Once the encoding has occurred, the Real-Time Transport Protocol (RTP) [25] is utilized to convey the voice stream; in this stage, the voice stream is divided in certain intervals of time and grouped into packets. In essence, RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio (voice). RTP provides delivery monitoring of its payload types with sequence numbers and timestamps, and thus it allows the receiver to reconstruct the sender's packet order. The basic RTP header consists of only twelve bytes (figure 2.2), and it comprises basically: two bits for version, one-bit padding, another bit for extension, four-bit CSRC count, one bit for marker, seven bits for payload type, 16-bit sequence number, 32-bit timestamp, and 32 bits for the synchronization source identifier (SSRC).

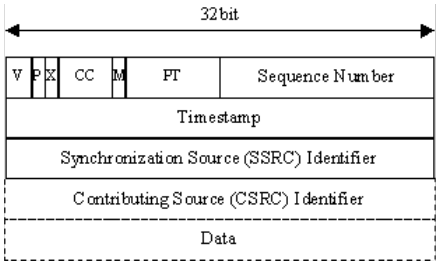

Figure 2.2 ***RTP header***

RTP does not address resource reservation and does not guarantee Quality of Service (QoS) for real-time services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a scalable way and provide minimal control and identification functionality. Essentially, RTCP is primarily useful as a heartbeat monitoring whether data delivery is occurring at all, and for endpoints to decide whether parts of the corresponding RTP stream are being lost in cases of network malfunction. Fortunately, RTP and RTCP are designed to be independent of the signaling protocol, encoding schemes, and network layers implemented [17].

Applications typically run RTP on top of User Datagram Protocol (UDP) to make use of its multiplexing and checksum services. UDP is a simple connection-less protocol for applications that do not want the Transmission Control Protocol (TCP)'s flow control; both protocols correspond to the transport layer according to the Open Systems Interconnection (OSI) [15]. TCP is preferred for call control and supplementary services, so we want it to be as reliable as possible [19]. The UDP header contains 8 bytes of Protocol Control Information: source and destination ports, UDP length and checksum.

Afterwards, an IP header is added. It corresponds to the Network OSI layer, and it contains information for routing purposes. The IP header consists of several fields, including version, its length, type of service, flags, time to live, protocol, header checksum, and source destination IP addresses. As a result, each voice packet incurs an uncompressed 40-byte header that comprises 20 bytes for the IP header, 8 bytes for the UDP header, and 12 bytes for the RTP header.

2.2.4 Delivery towards the Ethernet

Before sending the packet to a network physical interface, the Medium Access Control (MAC) protocol is used to provide the data link layer of the Ethernet LAN system. The MAC protocol encapsulates the data containing the voice packet (with the upper layer protocols) by adding a 14-byte header, called Protocol Control Information (PCI), and appending a 4-byte Cyclic Redundancy Check (CRC) as a tail (figure 2.3). This entire frame is preceded by a small idle period (the minimum inter-frame gap, 9.6 μs) and an 8-byte preamble [26]. The purpose of the idle time before the transmission starts is to give a small time interval for the receiver electronics in each of the nodes to settle after completion of the previous frame. The preamble allows time for the receiver in each node to synchronise the receive data clock to the transmit data clock.

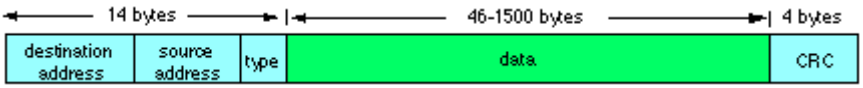

Figure 2.3 ***MAC encapsulation of a packet of data***

The header consists of three parts: a 6-byte destination address, 6-byte source address and 2-byte type field, which identify the type of protocol being carried (e.g. 0x0800 corresponds to the IP protocol). The CRC provides error detection in the case where line errors (or transmission collisions in Ethernet) result in corruption of the MAC frame. The MAC receiver discards any frame with an invalid CRC without further processing

There are many factors that impact voice quality in a VoIP network, and they are fairly well understood. Most of these can be mitigated with careful network design. Also, using control tools for good quality assurance in the network equipment can address these issues effectively [20]. The level of control over these factors will vary from network to network. For example, more control mechanisms for quality assurance are required in the internet, compared to a well-managed enterprise network.

2.3 SIP overview

2.3.1 Motivation

The SIP is an application-layer control protocol developed by the Internet Engineering Task Force (IETF), which gaining relevance in the communication market. Besides, the 3rd Generation Partnership Project (3GPP) has adopted SIP as a mandatory protocol for handling signaling in IP Multimedia Services (IMS) provided to 3G applications in the mobile phones [27]. For Ericsson, SIP is the protocol that will merge the mobile and fixed networks for Internet and multimedia services. Currently, some Ericsson

Enterprise products – such as IMS, MX-ONE, SME and IP phones – are facing the implementation of SIP.

The current MD110 IP telephony is based on the open H.323 standard for VoIP, however there is a strong motivation to implement applications based on the SIP standard. The SIP is considered to be a future powerful alternative to the H.323 standard as the signaling system for the dominant VoIP communications [16]. In fact, various vendors have been including SIP in their products without waiting for its full maturity. Therefore, one of the primary goals of the EMMA project is to implement the SIP standard for signaling and control.

2.3.2 Description

The most fundamental functionality of SIP is establishing, modifying and terminating multimedia sessions or telephony calls over a packet-based network. It operates on the top of UDP, but it can also operate over TCP. SIP is a text-based protocol, while H.323 is a binary-based protocol [28]; however, SIP is much simpler than H.323 when establishing multimedia sessions. A key feature of SIP is the personal mobility that it permits; i.e. a user should be able to access the services at any terminal in any location.

SIP is a client-server protocol, with requests by the client and responses returned by the server; the caller acts as client, and the callee acts as a server [29]. A client is any network element, such as a SIP phone, that sends SIP requests and receives SIP responses. A server is a network element that receives requests and sends back responses. The server can accept, reject or redirect the requests. [16]

In order to achieve the services of SIP there are a few distinct components defined such as User Agent, Proxy and Registrar, which must be present on a network. A User Agent (UA) acts both as a user agent client (UAC) and user agent server (UAS). The UAC is the calling user agent i.e. the application that initiates the SIP request. The UAS is the called user agent: A user agent server is a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user. The response accepts, rejects or redirects the request. A UA can be an IP phone or a softphone based on SIP. The EMMA is considered a UA in this context.

A Proxy Server is an intermediary program that acts as both as a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy interprets, and, if necessary, rewrites a request message before forwarding it. A Registrar is a server that accepts REGISTER requests, which allow the UAs to confirm their presence in the network. A registrar is typically collocated with a proxy and may offer location services. Ericsson Enterprise has a SIP Proxy Server based on a software PBX called Asterisk [30], which also behaves as a Registrar. Asterisk serves as a dedicated point at which all the UA are reachable.

2.3.3 SIP messages

SIP is a text-based protocol with syntax similar to HTTP. Message headers provide further information about a message and enable the message to be dealt correctly. SIP does not define the actual session attributes, treating them as opaque payload data in

order to remain independent of the communication media capabilities [31, pp. 1]. In addition, the Session Description Protocol (SDP) [32] is a mandatory part of the SIP standard. SDP is not really a "protocol", but a textual description (very simple language) of multimedia sessions. It is placed at the message body and it provides a description of the session to be established. It contains the type of media and codec as well as the sampling rate for negotiation with the called party. The SIP messages can be divided into two parts, request and responses.

Request messages:

INVITE	Invite the callee into a session
OPTIONS	Discover the capabilities of the receiver
BYE	Terminate a call or call request
CANCEL	Terminate an incomplete call request
ACK	Acknowledge a successful response
REGISTER	Register the current location of a user

Table 2.1 ***SIP request messages***

Responses are divided into different response classes identified by the first digit in a three-digit number.

Response Class	Meaning	Example
1XX	Information about the class status	180 Ringing
2XX	Success	200 OK
3XX	Redirection to another server	301 Moved Temporarily
4XX	Client did something wrong	401 Unauthorized
5XX	Server did something wrong	500 Internal Server Error
6XX	Global Failure (do not resend same request)	606 Not acceptable

Table 2.2 ***Summary of SIP response messages***

All SIP messages have headers of different types sometimes associated with their type or subtype of content. Typical message headers are the "Via", "To", "From", "Call-ID" and "CSeq" headers. The "To" and "From" contain the addresses of the callee and caller. A message can also contain a body (i.e. SDP) where the length, type and encoding are specified in different headers. Figure 2.4 shows an example of an INVITE request message and an OK response message.

```
INVITE sip:8626@130.100.26.140 SIP/2.0
Via: SIP/2.0/UDP 130.100.26.187:5060;branch=3Dz9h
From: <sip:8627@130.100.26.187>;tag=12060
To: <sip:8626@130.100.26.140>
Call-ID: 32225@130.100.26.187
CSeq: 6 INVITE
Contact: <sip:8627@130.100.26.187:5060>
Content-Type: application/sdp
Content-Length:   159

v=0
o=userX 2001 2001 IN IP4 130.100.26.187
s=session
c=IN IP4 130.100.26.187
t=0 0
m=audio 8000 RTP/AVP 0 8
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
```

(a)

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 130.100.26.140:5060;branch=z9hG4
From: "ip-phone2" <sip:8627@130.100.26.140>;tag=as17d1
To: <sip:8626@130.100.26.186:5060>;tag=17515
Call-ID: 620e512032dacb0e05cdf7d6119f8899@130.100.26.140
CSeq: 102 INVITE
Contact: <sip:8626@130.100.26.186:5060>
Content-Type: application/sdp
Content-Length:   167

v=0
o=userX 20000001 20000001 IN IP4 130.100.26.186
s=session
c=IN IP4 130.100.26.186
t=0 0
m=audio 8000 RTP/AVP 0 8
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
```

(b)

Figure 2.4 ***Example of SIP messages (a) INVITE request, (b) 200 OK response***

In SIP, addresses are typically SIP URLs of the form sip:user@host, where usernames can also be telephone-numbers or an alias. However SIP terminals can also support other URLs schemes (for example “tel:”). The implementation studied only supports the “sip:” URL scheme. SIP URLs contain sufficient information to initiate and maintain communication session with the resource. For interworking with a traditional circuit-switched network, SIP enables the “user” portion of the SIP address to be a telephone number.

2.3.4 Implementing SIP on a telephony application

In order to implement SIP for a simple telephony application, two main functions need to be created: a Finite State Machine (FSM); and, a message parsing and generation unit.

First, in every telephone conversation there is a specific sequence of events that depends on the actions that the user take; for instance, starting a phone call, waiting for an answer, answering the phone, putting down the phone when waiting or after the conversation, etc. These events affect the telephone conversation’s status, and some informational events may occur, such as ringing and tones; or SIP messages may be generated and sent. These status changes are modeled and implemented with an FSM.

On the other hand, a SIP message is in essence a continuous string of characters that needs to be processed and logically understood. This function is called message parsing, and it is the process of analyzing the string of characters in order to determine the actual SIP message structure. A parser transforms the input text into a logical data structure that is suitable for later processing. Message generation is the inverse process, and it consists to generate a string of characters from a given logical data structure.

Implementing these parsing algorithms requires a lot of effort, since it must be done manually [31, pp. 10]. For that reason, it is better to reuse existing libraries that implement them. In the EMMA project, an open source library called OSIP [33] is employed for message parsing and generation.

There are a few extra considerations that should be taken into account for an embedded application. In order to simplify the implementation while retaining the basic functionality, a limited amount of SIP headers are supported. Also, when using SIP over UDP, the entire message should fit within a single packet. If a SIP message is fragmented into multiple datagrams, the probability of losing the entire message increases with the number of fragments [34].

3 EMMA System design description

3.1 Scope

There are many features and services that VoIP gateways in the market typically offer, and thus they include more protocols and complexity in their designs. Nevertheless, the current version of the EMMA project embraces the most basic functionalities of a VoIP gateway. It makes possible to establish a reliable phone communication within a well-managed network. As mentioned, only the SIP standard is included for signaling and control of the phone calls, while the PCM G.711 is used to encode the voice data. Neither echo canceling nor silence suppression are included, due to their design complexity and no need in a fast Local Area Network (LAN). The RTCP is also out of this scope, since RTP is able to convey media without it.

The VoIP gateway acts as a translator of both signaling and media simultaneously. As a signaling gateway, the system is able to translate analog telephone signals into the SIP standard. It detects the Dual Tone Modulation Frequency (DTMF) produced when dialing numbers as well as the on-hook/off-hook states from the analog POTS. The dialed numbers are translated into user names and IP addresses. Consecutively, the system produces and sends a set of SIP text messages towards the network. Also, ringing and tones are generated at the analog POTS in order to indicate the status of the communication in a traditional way. Figure 3.1 shows two different sequences of events that occur during phone calls through the VoIP gateway. These events involve the delivery of SIP messages towards a proxy server as well as the detection and generation of analog signaling at the telephone.

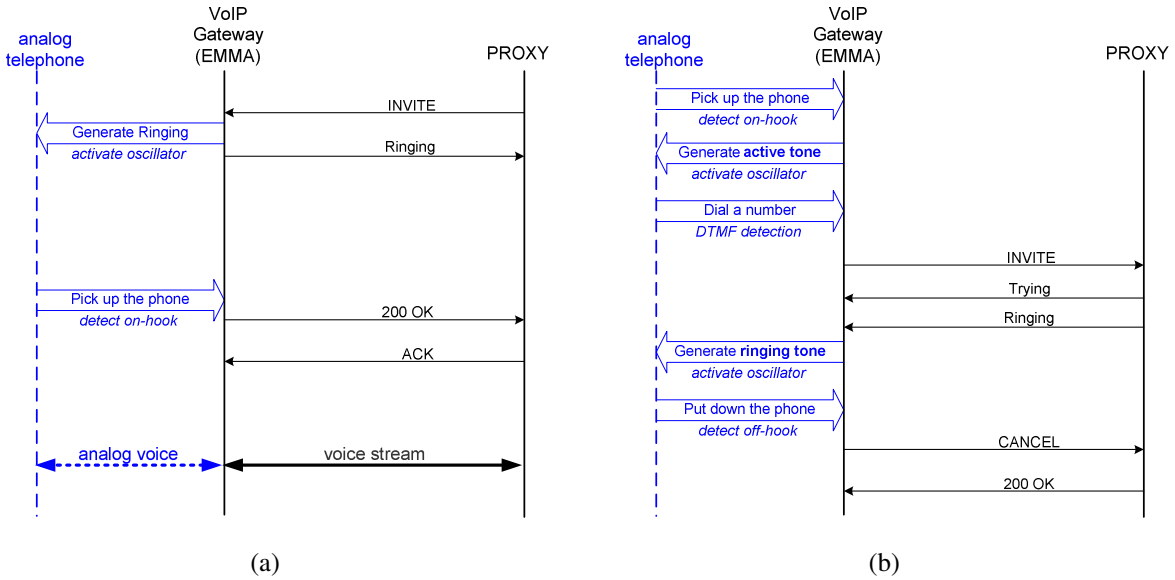

Figure 3.1 ***Functionality of a SIP signaling gateway. (a) Receiving a phone call and initiating the conversation, (b) Initiating a phone call and later canceling it***

As a media gateway, the system converts analog voice signals into RTP packets and vice versa. The system digitalizes and encodes the voice, and it creates the RTP packets for a proper transport. The sequence number and the timestamp, included in the RTP

header, are incremented according to the delivery of the packets. When RTP voice packets are received, the system stores and puts them in order, and then the voice streams contained in those packets are decoded and converted into an analog signal again.

Additionally, there might be an option to enter the gateway's IP addresses with a user interface. However, it is not very comfortable for a user to configure IP addresses manually all the time. Therefore, the VoIP gateway is also able to get it automatically with the Dynamic Host Control Protocol (DHCP) [15], which is an application-layer protocol that provides a plug-and-use functionality to a host. DHCP is a simple request-response UDP application that allows the host to get the IP address automatically during the system boot. Then, the system contains three application protocols running on top of UDP and IP. Figure 3.2 shows a layer-based diagram of all the protocols involved in this scope.

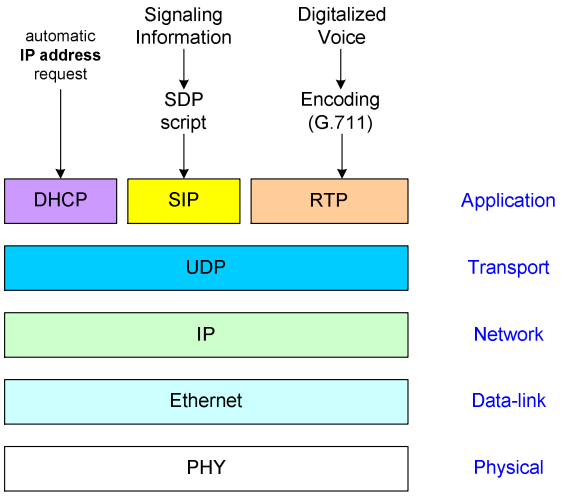

Figure 3.2 ***Network Protocols involved in the design***

3.2 System Model Specification

The whole system is modeled with the six abstract blocks (figure 3.3) that have specific tasks: interface to the analog POTS, RTP output/jitter buffer, main control unit, SIP message block, Packet-handling unit, and Ethernet MAC unit with a network physical interface. This scheme gives a better understanding of the different modules that are included in the system. Some of them are able to be implemented in hardware or software, while others reuse Intellectual Property because of their complexity.

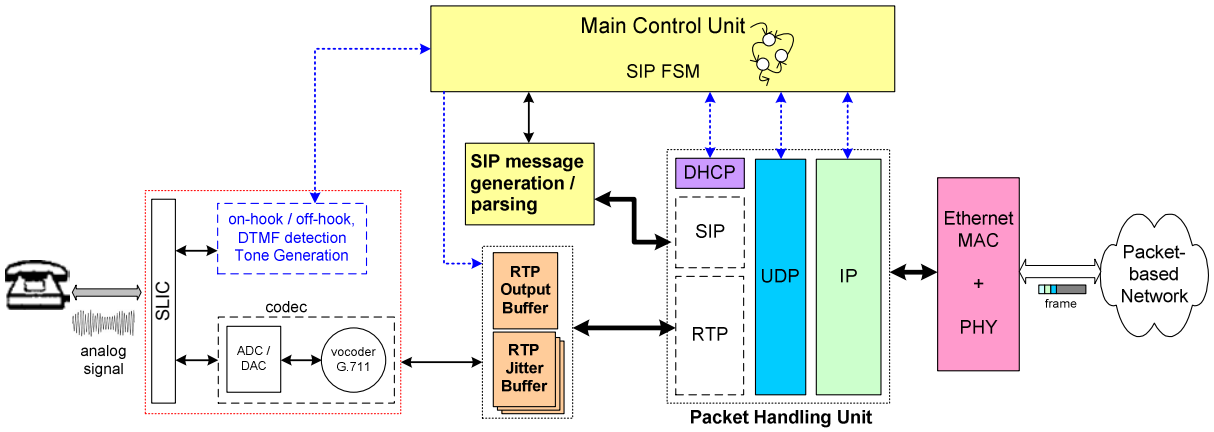

Figure 3.3 ***EMMA System block diagram***

A Subscriber Line Integrate Circuit (SLIC) device [19] is used to interface the analog telephone with the other blocks of the system. This device emulates the PSTN network's voltage levels, and its main function is to combine the analog voice signal with the PSTN voltages. Following this SLIC device, there are two units that handle analog signaling and voice digitalization. The first of them is responsible of detecting DTMF, on-hook, and off-hook as well as ringing and tone generation towards the phone. The other is a codec that is comprised of an ADC/DAC and a G.711 encoder/decoder.

The digitalized voice stream, produced after the codec, is then buffered and transmitted periodically. The RTP output buffer stores encoded voice data and adds an RTP header. When it has accumulated 30 ms of voice, the data contained in the buffer is sent towards the packet-handling unit. Simultaneously, the RTP Jitter buffer queues incoming voice packets before playing them out.

The purpose of the main control unit is to handle the telephone call, and it is based on an FSM that implements the basic requirements of the SIP standard. State changes occur depending on the signals produced by the analog POTS and the received SIP messages from the network. For that reason, the main control unit and the SIP message block work closely, and both constitute a "SIP stack"; the SIP message block executes the text parsing and generation functions that are described in section 2.3. The main control unit is not affected by the information contained in the RTP packets; however, it is able to start and stop the transmission of them. Furthermore, it sets port numbers and IP addresses in the packet-handling unit.

The Packet handling unit gets data from the SIP message block and the RTP output buffer, and it provides UDP and IP headers to that data. Subsequently, an Ethernet MAC unit concludes the encapsulation of the data and sends it to the network with a physical interface. The Ethernet MAC is included in the system as an Intellectual Property hardware block because its implementation entails much effort.

3.3 Design Platform

The system described above is classically implemented on the market with a combination of a high-performance RISC (reduced instruction set computer) processor and a Digital Signal Processor (DSP). In this approach, the RISC processor handles the network related software, from the signaling protocols to the Ethernet controller driver; whereas the DSP executes the vocoders and accompanying media processing algorithms [19]. Vendors also offer entirely VoIP packaged solutions that meet stringent objectives. The problem illustrated here is product differentiation and also vendor dependence. Furthermore, the only possibility for this architecture to evolve is to wait for future generations of the chosen processors.

One interesting development during the last decade which is perfectly suitable to utilize to reduce the development time and cost is reprogrammable hardware, specifically the FPGA. Reconfigurable hardware has been progressively replacing the application specific hardware in small volume designs. The use of reprogrammable hardware allows the system to be implemented and upgraded without non-recurring engineering costs, so the system can be reconfigured in the field after deployment. This flexibility permits the hardware structures to be specialized and optimized to achieve a good performance; this performance can be greater than the one that is achieved within most traditional processing systems.

Moreover, FPGA vendors offer new capabilities that are well suited to SOC design, such as RISC processors and application-specific cores. These processors are designed to interface with standard buses, simplifying system design with the availability of third-party Intellectual Property cores. A complete, customized design can therefore be created on these platforms by integrating a RISC processor with a number of off-the-shelf cores, minimizing development time for the application designer. The software design of such a system has fewer restrictions than the classical approach in which the hardware dictates how the software must be implemented. With an FPGA-oriented solution, the hardware can be adapted to the requirements of the software designers.

For the reasons mentioned, the EMMA project has chosen the FPGA solution. A fully programmable SOC, which contains an embedded RISC processor and specific purpose on-chip peripherals interconnected with a bus, is designed and implemented on an Altera Cyclone II FPGA. External memories are essential to store the software running on the embedded processor, and a network adapter is also considered necessary. Therefore, a development board, which gathers an FPGA and some other useful components, is employed. Additionally, a fully equipped programmable SLIC/CODEC device [35] (i.e. ProSLIC) that incorporates extra features for analog signaling is employed. This device serves as interface between the designed system on the FPGA and the analog telephone (figure 3.4).

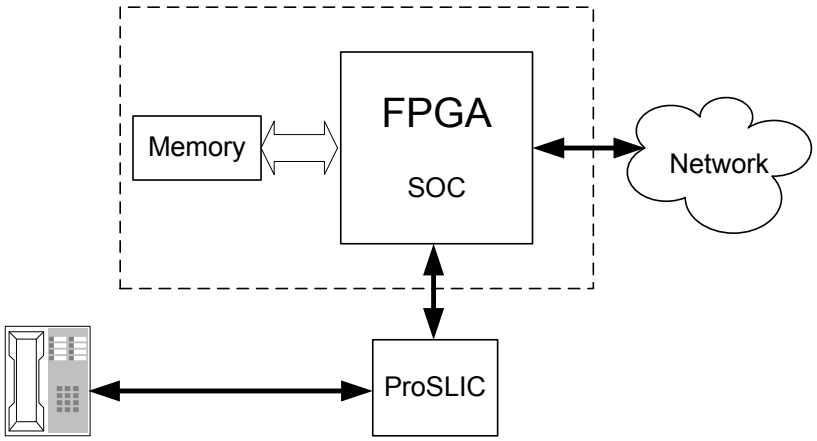

Figure 3.4 ***Hardware Architecture employed***

Further details about the hardware architecture and the embedded software that are implemented on the FPGA as well as the SOPC design methodology are provided in the following chapters.

4 Specifications of the Analog Interface

The analog interface consists of two blocks. One block that takes care of the high voltages, and the other block takes care of DTMF, Tone Generation, Ringing Generation and PCM Interface.

When setting up the criteria for the analog interface there where some criteria that were more important than the others. One of them was that it should be as small as possible, and another important criteria were that it should be as cheap as possible.

Regarding to these two criteria there was almost just one solution for the analog interface and that was the Silicon laboratories ProSLIC family of microcontrollers. This conclusion were drawn since Ericsson has really good prices on this microcontrollers make them the cheapest alternative. They are also the smallest microcontrollers on the market today with the features that were needed for the design, such as DTMF detection and Ringing Generation etc.

The ProSLIC microcontroller family provides both the microcontroller for high voltages and the microcontroller for the POT interface. The microcontroller that provides the solution for high voltages is Si3201 and the microcontroller that provides the solution for the POT interface is Si3210 ProSLIC.

4.1 Si3201

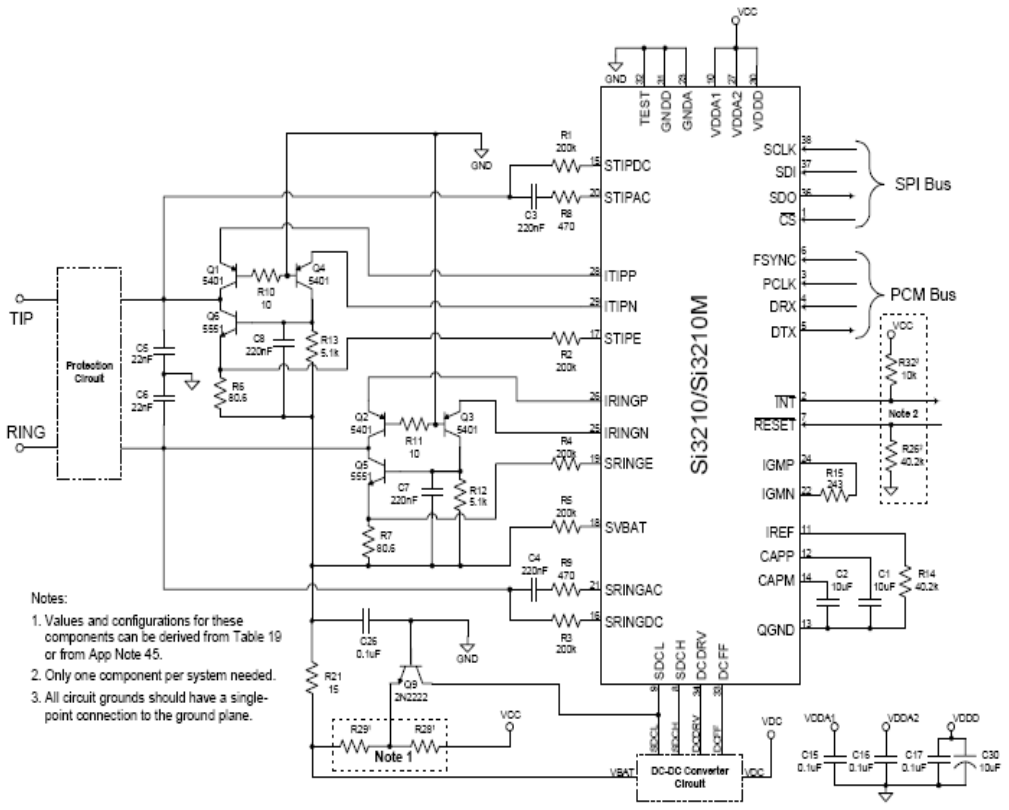

Figure 4.1 ***Circuit with discrete components [35]***

The Si3201 is protecting the Si3210 ProSLIC from high voltages that can appear from the telephone line. The Si3201 is also used to perform all high voltage functions such as on-hook and ringing signals. By using the Si3201 in the high voltage protection, instead of discrete components, a big advantage is gained in designing such a small circuit as possible. The difference is shown in figure 4.1 and figure 4.2.

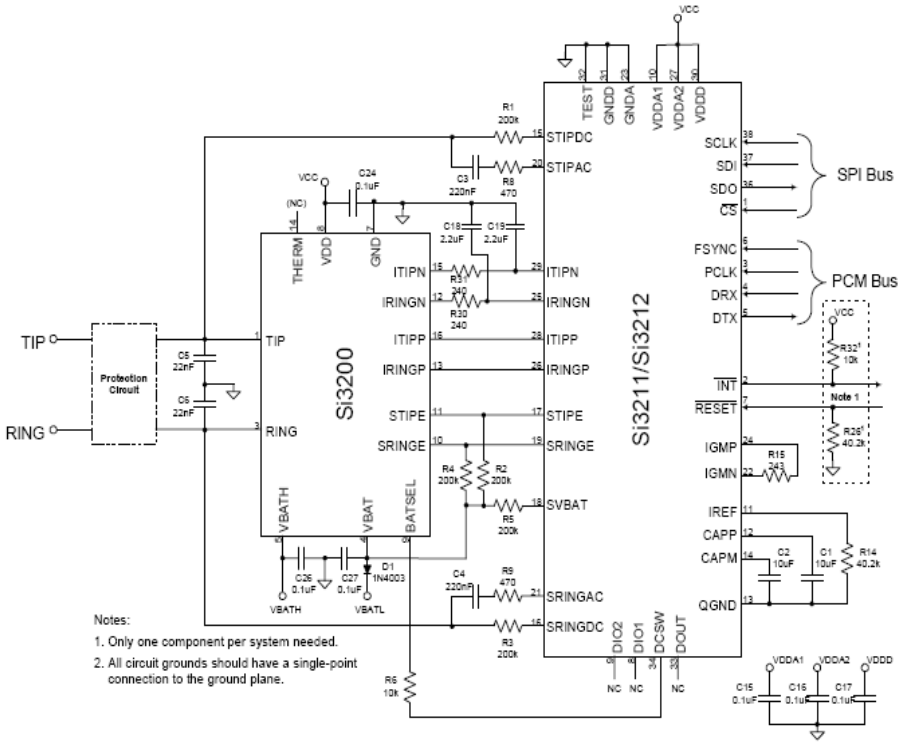

Figure 4.2 ***Circuit with IC solution [35]***

4.2 Si3210 ProSLIC

Si3210 ProSLIC is a powerful integrated circuit that provides a complete analog telephone interface. The most useful functions that Si3210 ProSLIC provides are codec and battery generation functionality. Since the EMMA-design should have as low power consumption as possible the battery generation was really useful. It continuously adapts its output voltage to minimize power, which and that made it possible to power the entire Si3210 ProSLIC solution from a single 3.3V supply. Si3210 ProSLIC also has software configurable features like DTMF generation and decoding.

4.2.1 DTMF

The dual-tone-multi-frequency (DTMF) tone signalling standard is also known as touch tone. It is an in-band signalling system used to replace the pulse-dial signalling standard. In DTMF two tones are used to generate a DTMF digit. One tone is chosen from four possible row tones and the other tone is chosen from four possible column tones. The sum of these tones constitutes one of 16 possible DTMF digits as shown in table 3.1.

4.2.2 DTMF detection is based on the modified Goertzel algorithm [35]

Goertzel algorithm is used to compute the dual frequency tone (DTF) for each of the eight DTMF frequencies as well as theirs second harmonics. At the end of the DTF computation, the squard magnitudes of the DTF results for the eight DTMF fundamental tones are computed.

After this is done, the row results are sorted to determine the strongest row frequency, the column frequencies are sorted as well. When this process is completed there are a number of checks made to determine if the strongest row and column tones are DTMF digits or not. The process to detect the DTMF digits is performed twice within the 45 ms minimum tone time. The criteria it must fulfil to be recognized as a new DTMF digit is that it has to be detected on two consecutive tests following a pause. When it has passed all tests there is an interrupt generated and the DTMF digit value is loaded into the DTMF register.

DTMF char	***Low (Hz)***	***High (Hz)***
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
*	941	1209
0	941	1336
#	941	1477
A	697	1633
B	770	1633
C	852	1633
D	941	1633

Table 4.1 ***DTMF frequencies [35]***

4.2.3 Tone Generation

The tone generation is made through the two digital tone generators that are provided in the Si3210 ProSLIC. These two digital tone generators allow a wide variety of single or dual tone frequency and amplitude combinations. Each of the two-tone generators contains a two-pole resonate oscillator circuit with programmable frequency and amplitude. The clock frequency for the two oscillators is 8000 Hz.

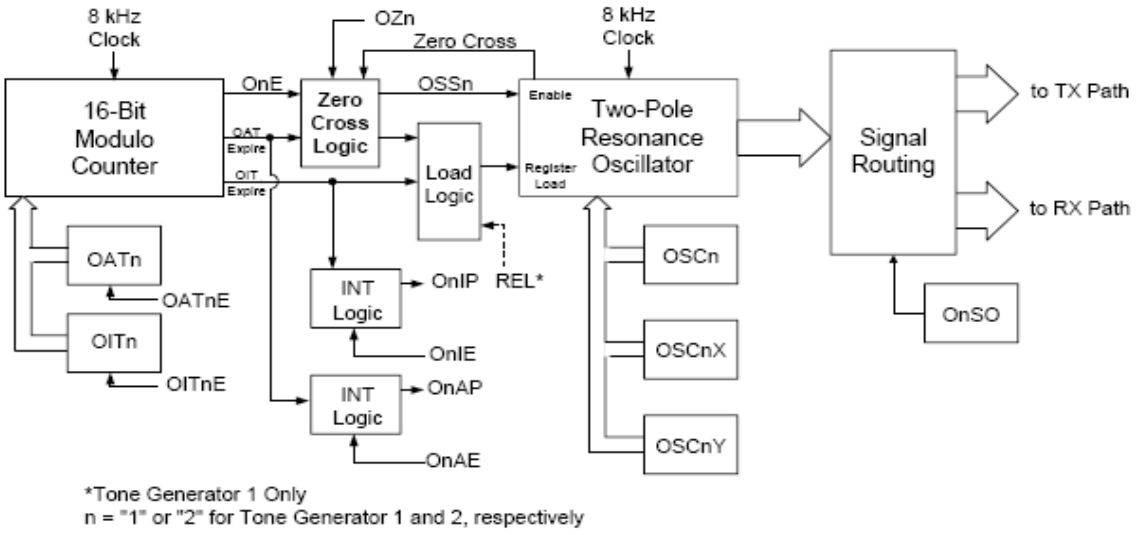

Figure 4.3 ***ProSLIC Tone Generation Architecture [35]***

4.2.4 Ringing Generation

The ringing generator architecture is nearly identical to that of the tone generator. The internal two-pole resonance oscillator circuit that has programmable frequency and amplitude creates the sinusoidal ringing waveform. But there is a big difference, since ringing frequencies are very low compared to the audio band signaling frequencies, the ringing waveform is generated at a 1000 Hz clock rate instead of 8000 Hz.

4.2.5 Control Interface

The 4-wire control interface consists of a clock (SCLK), chipselect (CS), serial data input (SDI), and serial data output (SDO). Data is transferred one byte at a time with each register access consisting of a pair of byte transfers. Figure 4.4 and 4.5 illustrate write and read operation in the SPI bus. The first byte of the pair is the address byte. The MSB of this byte indicates a register read when 1 and a register write when 0. The remaining seven bits of the address byte indicates the address of the register to be accessed. The second byte of the pair is the data byte.

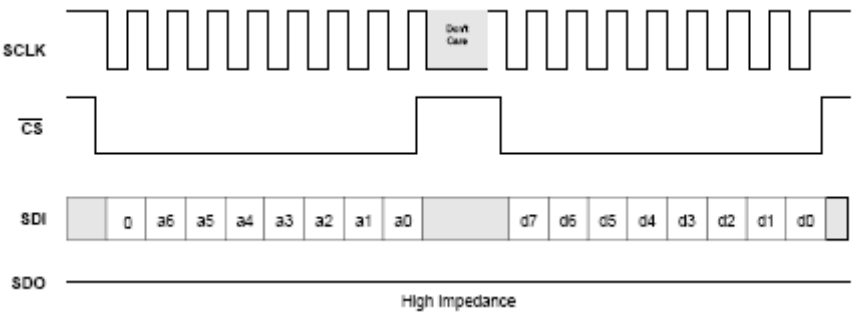

Figure 4.4 ***Serial write [35]***

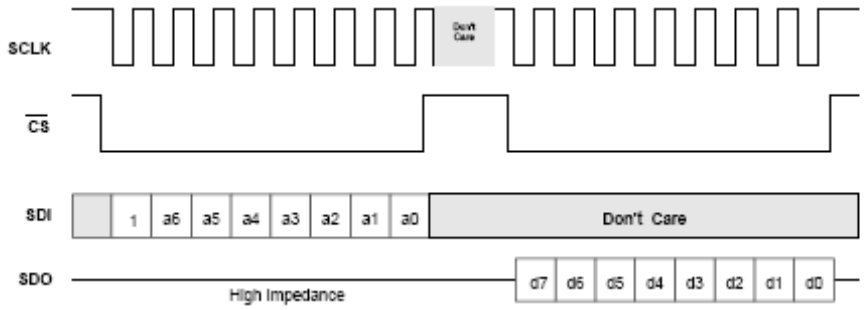

Figure 4.5 ***Serial read [35]***

4.2.6 PCM Interface

The Si3210 ProSLIC contains a programmable interface for the transmission and reception of digital PCM samples. PCM data transfer is controlled via the PCLK and FSYNC inputs as well as the PCM Mode Select that gives the opportunity to set some different parameters like A-law or µ-law for example. PCM Transmit Start count and PCM Receive Start count sets after how many bites transmit or receive should start.

5 System-on-Chip architecture design

5.1 Design strategy

FPGA vendors provide configurable soft processor cores that can be synthesized onto their FPGA products. Altera offers Nios II, while Xilinx offers MicroBlaze [36], both 32-bit general-purpose soft processor cores, and they grant a tremendous flexibility during a design process. They also help the designers to configure the processor to meet the needs of their systems (e.g., adding custom instructions or including/excluding particular datapath coprocessors) and to quickly integrate the processor within any FPGA. Besides, several soft processors can be included in one single FPGA [37].

Designing the architecture of a real-time system based on a soft processor core often demands a careful hardware/software co-design strategy. Soft processor cores implemented using FPGAs typically have higher power consumption and decreased performance compared with hard-core processors [38]. To alleviate the performance and power overhead of the system, a designer can employ hardware/software partitioning, which consists in a separation of the system functionality into application-specific hardware and software executed on the soft processor. This partitioning is the major problem in hardware/software co-design, and it has a first order impact on the cost/performance characteristics of the final design [39].

Not only do the FPGAs provide great flexibility for partitioning a real-time application into hardware and software, but also they allow implementing it with a high degree of parallelism. Once the critical processes within the real-time application are identified, they are implemented as hardware blocks that work in parallel with the soft processor; thus, it improves its efficiency. The implementation of the system involves hardware synthesis and software compilation, and finally it is complemented with integration and verification [40].

The EMMA project proposes a one-level hardware/software partition (see figure 5.1) in order to accelerate the processing of voice packets, since they need to be treated in real time. The system model described in chapter 2 is implemented with a bus-based architecture that has Nios II as the central processing unit (CPU). The main control unit, the SIP message block and the packet-handling unit (see section 2.4.2) are carried out by embedded software (explained in the next chapter) running on Nios II, while the RTP output/jitter buffer as well as the interface towards ProSLIC are handled by an application-specific hardware block. The SOC architecture and its main components are described in the following section. Further details about the hardware acceleration block that is included as a component of the SOC architecture are given in section 3.3. The system is later prototyped and verified (chapter 7) on the *Altera Nios II Development Board* that includes a Cyclone II FPGA [41]. The last section of this chapter shows the utilization of the Altera Cyclone II FPGA, which is obtained after synthesizing the architecture with Quartus II version 5.0.

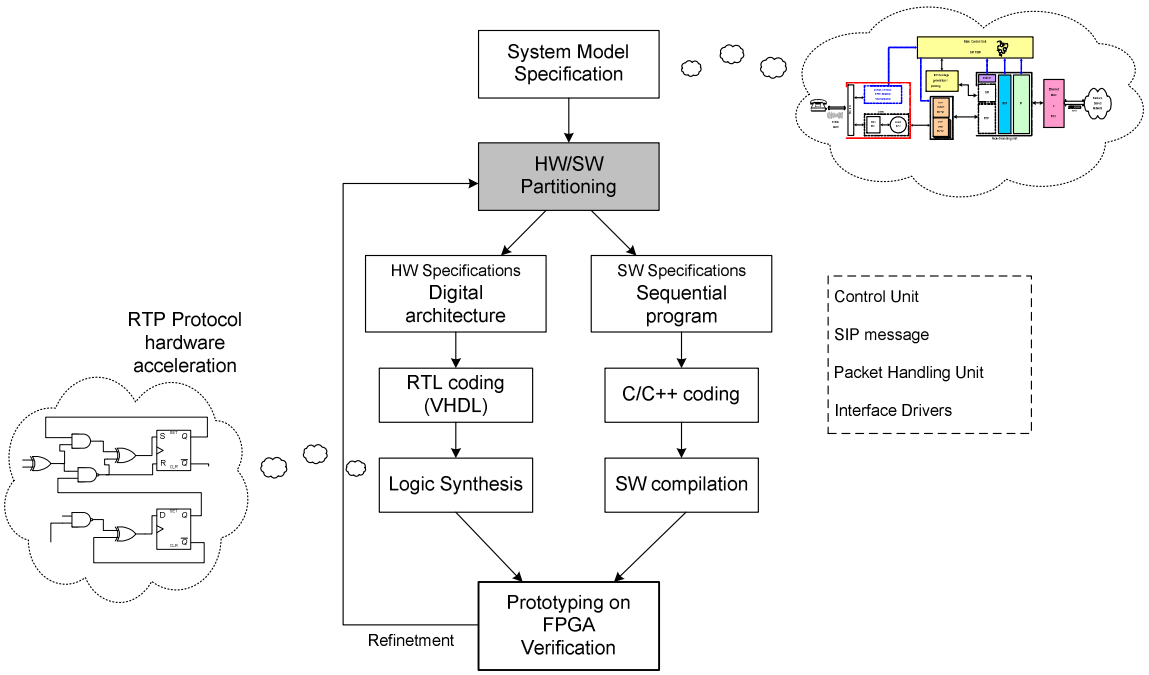

Figure 5.1 ***Hardware/Software Partitioning***

5.2 Architectural Description

The EMMA system architecture resembles a “computer on a chip” because it includes a CPU and a combination of peripherals and memory on a single chip. It contains one Nios II, a set of specific purpose on-chip peripherals and memory as well as interfaces to others off-chip components. These components are selected according to the requirements of the system design, and they are joined with a flexible on-chip bus structure called Avalon Switch Fabric [42]. Altera’s SOPC Builder design tool [43] is employed to configure the system’s features and generate a hardware architecture that can be programmed into an FPGA.

SOPC Builder is included in Quartus II and is a powerful tool for creating systems based on processors, peripherals and memories. It makes possible to define and generate a complete SOPC in much less time than using traditional, manual integration methods. The purpose of SOPC Builder is to abstract away the complexity of interconnect logic, allowing designers to focus on the details of their custom components and the high-level system architecture [44]. Using the SOPC Builder graphical user interface (GUI), the EMMA system architecture is straightforwardly created putting together the required set of Intellectual Property hardware blocks (see figure 5.2). SOPC Builder outputs VHDL files that define all components of the system, and a top-level HDL design file that connects all the components together.

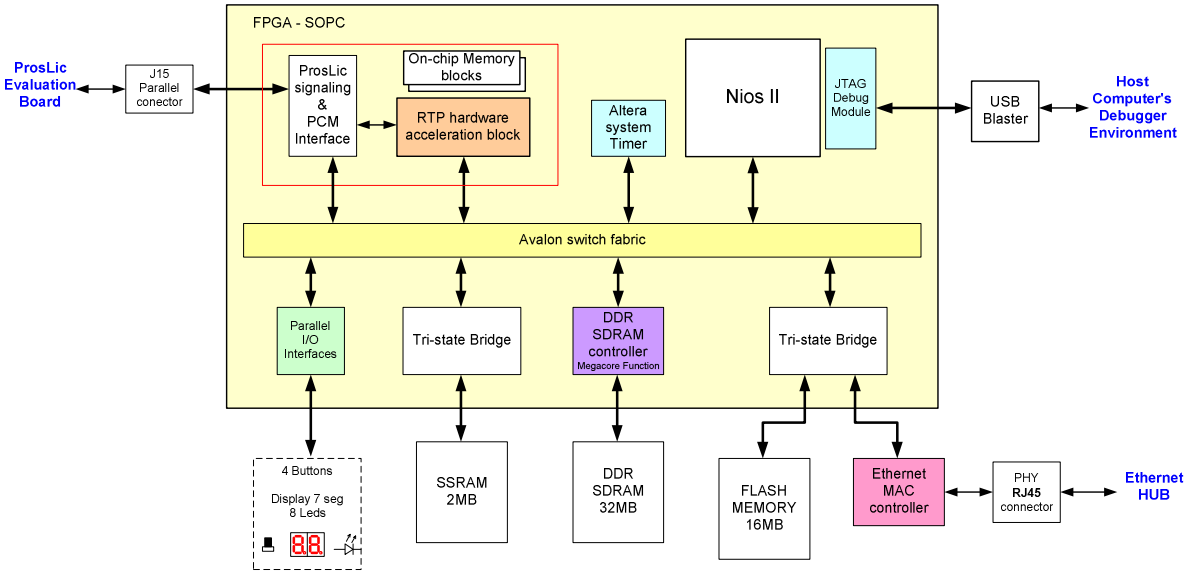

Figure 5.2 ***System architecture block diagram***

The Altera Nios II is a general-purpose RISC soft-processor core, which provides a full 32-bit instruction set, data path, and address space. It contains 32 general-purpose registers and accepts 32 external interrupt sources. Nios II is configurable, since it allows that some features – such as different cache sizes or custom instructions – can be added or removed in order to meet the desired performance or price goals. SOPC Builder provides readymade Nios II system designs that designers can reuse; if these designs meet the system requirements, there is no need to configure the design further. A standard configuration of Nios II [44] is employed, since it consumes fewer resources on the FPGA than a full featured Nios II and keeps a good system performance.

The soft-core nature of the Nios II permits designers to debug a system in development using a join-test-action-group (JTAG) debug module, which provides on-chip emulation features to control the processor remotely from a host PC. The debug module connects to the JTAG circuitry in an Altera FPGA, and thus external debugging probes can access the processor via the standard JTAG interface on the FPGA. For trace data collection, the debug module stores trace data in memory either on-chip or in the debug probe [44]. For the release version of a product, the JTAG debug module functionality can be reduced, or removed altogether.

SOPC Builder provides a set of peripherals commonly used in microcontrollers, such as timers, serial communication interfaces, general-purpose I/O, SDRAM controllers, bridges and other interfaces. Some of them are selected according to the available off-chip devices included in the *Altera Nios II Development Board* [41]. A DDR SDRAM controller is used to interface the system with a fast 32 MB SDRAM, while tri-state bridges adapt a 2MB SSRAM as well as a 16MB Flash Memory and an Ethernet MAC/PHY device. The last one allows Nios II to access Ethernet networks via an RJ-45 connector, but it shares address and data connections with the Flash memory in the evaluation board. General purpose I/O interfaces are employed to control 8 LEDs, a seven-segmented display, an LCD screen and 4 buttons, which are also very useful for analysis and debugging purposes.

The SDRAM (volatile) stores the application program code during its execution on Nios II, while a Flash nonvolatile memory is used for the permanent storage of the application program. The application code is downloaded to the nonvolatile storage, and then copied in the SDRAM during the system power-up. In addition, a new program can be stored into the external Flash by downloading it from the PC without having to reload the FPGA's hardware configuration. This scheme provides the advantages of permanent storage and fast execution, and the ability to modify the application program when needed.

The RTP acceleration block and the ProSLIC interface are added to the system as one custom component. Both blocks interact very closely, since the voice samples gathered from the ProSLIC's PCM interface [40] are placed in buffers, conforming RTP voice packets. Additionally, the ProSLIC interface uses some of the general purpose pins on the FPGA Development Board in order to communicate with the ProSLIC's evaluation board [41]. This custom component is designed with a special purpose, and it offers a double performance benefit: the hardware implementation is faster than software; and the processor is free to perform other functions in parallel while the custom peripheral operates on data. Its design is described with more detail in the following section.

The system architecture utilizes advanced features of the Avalon bus to interconnect most components [42]. The Avalon switch fabric is the glue logic that binds all the system's components together. It is actually the collection of signals and logic that connects master and slave components; including address decoding, data-path multiplexing, wait-state generation, arbitration, interrupt controller, and data-width matching. The bus is generated from SOPC builder, and it provides a switch fabric that allows the components to communicate with each other simultaneously. Each component is assigned an address range, according to which they are accessed. The interface from the components to the on-chip bus follows the Avalon bus specification [42]. The Avalon interface is an open standard, and no license is required to produce and distribute custom peripherals using the Avalon interface.

5.3 Hardware acceleration block

5.3.1 Description of the Custom Component

The RTP acceleration block and the ProSLIC are included inside an SOPC Builder custom component [43]. As with any logic design process, the development of this

SOPC Builder component begins by specifying its functional requirements. Thereafter, this component is described with VHDL and simulated with Modelsim tool; coding VHDL is an iterative process, since verifications of the code should be performed against the functional specification. Afterwards, the designed VHDL entity is provided to SOPC Builder as a design entry, and then SOPC Builder automatically creates and integrates the component within the system architecture. Nonetheless, a few considerations has been taken when designing the VHDL entity.

In SOPC Builder a peripheral can contain many ports with different characteristics, and it makes possible to include different functionalities in one single block. Every port gathers a set of Avalon signals that are required to perform bus transactions in the system. This flexibility permits to include four slave ports (figure 5.3) in this hardware acceleration block: PCM Control, Read Buffer, Write Buffer, Signaling control.

The PCM control port is defined as "slave 0" at SOPC Builder, and it contains a number of registers that configure parameters of the other hardware blocks. Also, this block gives the status of the execution of the system. The Read and Write buffer ports (slave 1 and 2 respectively) are seen as on-chip memory blocks by Nios II, since they are based on the *LPM_RAM* hardware IP blocks from the *Altera Megafunctions library* [45] at Quartus II. Both ports provide support to low-level software instructions. In fact, they are aimed to avoid Nios II to read/write every PCM sample coming from ProSLIC. As a result, any program running under Nios II only reads/writes a bunch of PCM data starting from the corresponding memory position; the UDP and IP header are added or removed by a program running under Nios II, which is mentioned in the next chapter.

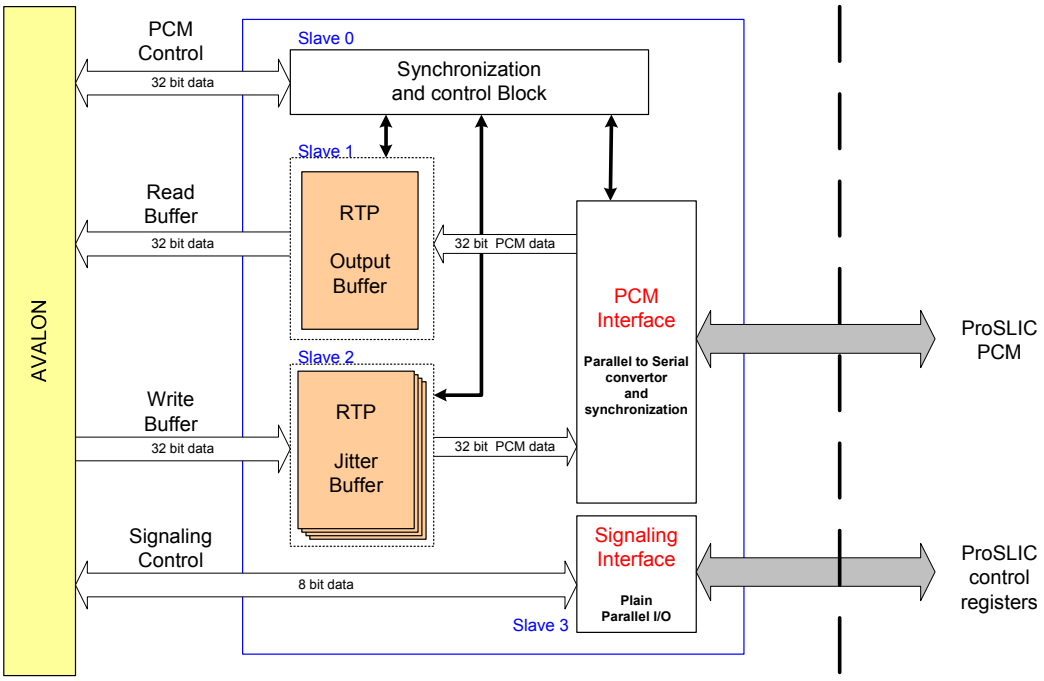

Figure 5.3 ***Block Diagram of the Avalon custom peripheral designed for hardware acceleration***

Since this block acts also as an interface to an off-chip device, it contains other signals – in addition to the Avalon interface – that connect to logic outside the system. Therefore, its top-level VHDL entity clearly specifies a complete set of Avalon signals

corresponding to every slave port as well as those connecting to the logic outside the system.

5.3.2 ProSLIC Interface

As described in the previous chapter, encoded voice samples are gathered from the ProSLIC device. It provides a straightforward interface that transmits and receives serial 8-bit PCM samples, with the DTX and DRX signals respectively, and two periodic signals for synchronization: FSYNC at the PCM sampling frequency (8 KHz), and PCLK at the frequency of the transmitted serial data [46]. Therefore, the designed ProSLIC PCM interface is actually a digital parallel-to-serial converter, which also provides synchronization to the system. A few counters keep track of the transmission/reception of the data (figure 5.4). The transmitted/received 8-bit samples are first converted into parallel by a shift register of 8 bits. Later, the information gathered by this register is sorted into 4 registers of the same size with the purpose of matching the 32-bit data path of Nios II.

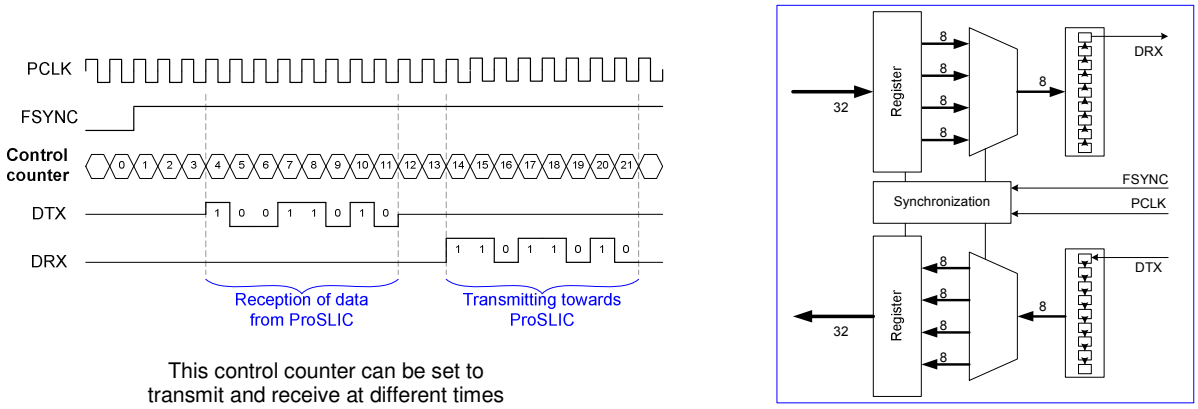

Figure 5.4 ***Transmission of PCM data***

The Signaling control port (figure 5.3) is implemented as a simple parallel I/O interface (slave 3) in order to access the control registers inside the ProSLIC device. Through these control registers, it is possible to generate informational signals (ringing and tones) at the analog POTS as well as gathering the information from it (line status, DTMF detection).

5.3.3 RTP Output Buffer

The encoded PCM samples are grouped and transported with the RTP protocol. It implies to store certain amount of those samples in a buffer (on-chip memory), and then a consistent RTP header is appended on the top of that buffer prior to its transmission (see figure 5.5). As mentioned in the previous chapter, the RTP header contains two parameters used for delivery monitoring (Sequence Number and Timestamp) which are incremented during the transmission of the voice packets. This task is performed by a synchronization unit that has two counters, which are incremented as the PCM samples are buffered and as the packets are sent.

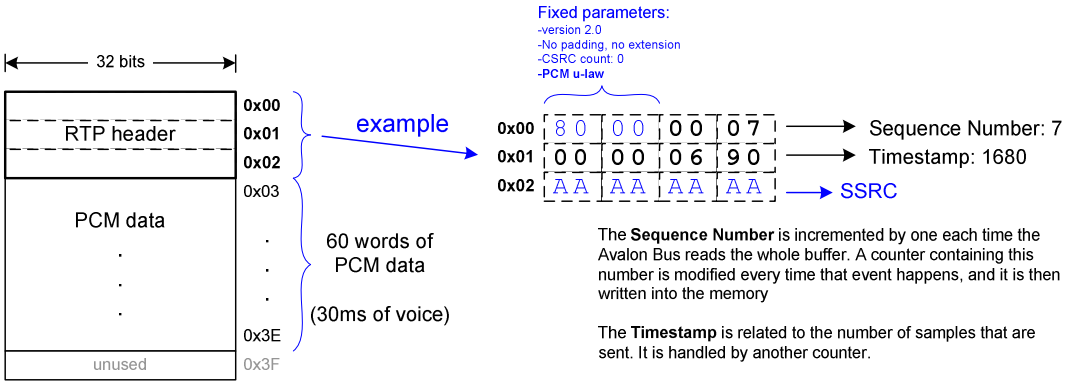

Figure 5.5 ***Organization of the on-chip memory block that implements the RTP Output Buffer***

Another counter takes care of the number of data written in the *LPM_RAM* block. It counts up to 60 words of 32 bits, which correspond to 30 ms of voice. When this counter reaches its maximum value, an interruption is triggered to indicate the soft-processor that the data is ready to be collected. After the soft-processor has gathered the data contained in the buffer, it also sends an acknowledge signal through the PCM control port (slave 0). Thereafter, the count re-starts and new PCM data is filled in the buffer.

5.3.4 RTP Jitter Buffer

RTP packets received from the network need to be queued and sorted in a jitter buffer according to their corresponding sequence numbers. Therefore, the jitter buffer has a storage unit that is made of a set of on-chip memory blocks (*LPM_RAM*), which hold the arriving RTP packets. The VHDL code includes a generic size (m) and number (n) of *LPM_RAM* blocks (see figure 5.6). Using generic coding style gives flexibility and scalability to future designs, which may need a jitter buffer that is able to hold more packets with different sizes. These parameters are seen at SOPC Builder when the custom component is integrated within the system.

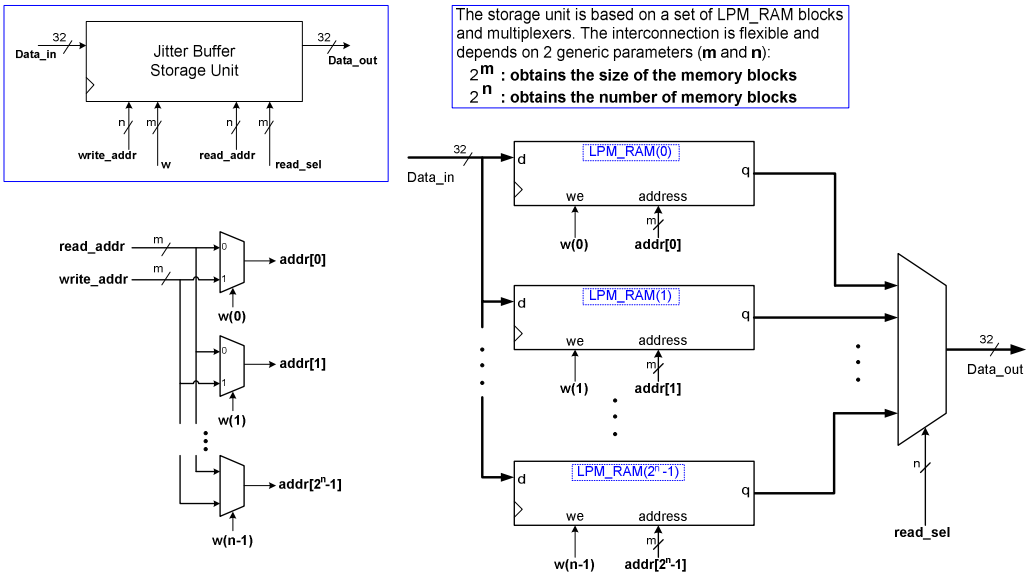

Figure 5.6 ***Flexible and scalable Storage Unit for the Jitter Buffer***

Besides, a dedicated control unit is used to synchronize the reception of those packets on any empty memory block. Simultaneously, this control unit permits that encoded voice samples, which are contained in other memory block, are sent towards the ProSLIC PCM interface in a sequential order. When the Avalon Bus writes on a specific memory block, the control unit makes that memory block not available for writing until its data has been sent towards the PCM interface; hence, it guarantees that any filled memory with relevant information is not overwritten. For that purpose, the control unit assigns consistent values to the signals *read_sel* and *w* (figure 5.6), which select the memories to read and write respectively.

5.4 Logic Synthesis and FPGA Utilization

SOPC Builder generates a system with encrypted HDL code that is later treated by the synthesis tools at Quartus II. The system architecture was synthesized on a Cyclone II EP2C35F672C6 FPGA that contains 33,216 logic elements (LE) and 483,840 bits of on-chip memory. A logic element is the smallest unit of logic located in the Altera devices. In the Cyclone II devices, each LE consists of a four-pin look-up-table (LUT), a programmable register and a carry chain. [14]

Normal physical synthesis effort and Standard Fitting effort were employed [46], and the synthesis report showed the following figures:

- Total LE: 5 355 / 33 216 (16%)
- Total I/O Pins: 232 / 475 (48%)
- Total memory bits: 64 512 / 483 849 (13%)
- Embedded Multipliers: 4 / 70 (5%)

The generic parameters of the RTP Jitter Buffer were configured in SOPC Builder, obtaining 4 memory blocks (n=2) holding 64 words each (m=6). Changing these parameters has a direct impact on the memory bits employed by the RTP hardware acceleration block (SOPC custom peripheral). Table 5.1 shows a detailed description of the resources consumed by the whole system architecture, the CPU (Nios II) and the custom peripheral; the *LE combination* and *LE Registers* columns list the number of LEs used by each design entity and the LEs (in parentheses) used only at that level of hierarchy. It is observed that the custom peripheral only consumes a small fraction of LEs and memory bits in comparison to the whole system.

Entity	***LE Combinational***	***LE Registers***	***Memory Bits***
System Architecture	4607 (14)	3301 (0)	64512
CPU (Nios II)	1552 (1275)	1133 (951)	45 952
RTP accelerator (custom peripheral)	406 (7)	323 (64)	10240
ProSLIC PCM interface	77 (55)	67 (38)	0
Output Buffer	132 (55)	141 (0)	2048
Jitter Buffer	184 (34)	41 (0)	8192

Table 5.1 ***Resources consumed by the system architecture, the CPU and the custom peripheral***

These results show that the designed system architecture only consumes few internal resources of the FPGA, but a quite high utilization of I/O pins. It means that further development can be performed on this platform in order to achieve more complex applications (for instance, adding a DSP unit with more vocoders or an echo canceller). Nevertheless, it is also good to keep a lower utilization of LE, since the cost of a final product increases with the number of LEs.

In addition, two phase-locked loops (PLL) (out of 4) contained in the Cyclone II FPGA supply clock frequencies of 60 Mhz and 85 MHz, using an input frequency of 50 MHz from an external clock. The design timing constraints – given by these two clock frequencies – were satisfactorily met. Theoretically, the designed system can achieve a maximum clock frequency of 91.58MHz, and the maximum critical path was found at the DDR SDRAM controller Intellectual Property hardware block that is given by Altera.

6 Embedded Software Design

6.1 Real-Time Design Environment

Altera provides the *Nios II Development kit*, which embraces a set of tools for developing real-time applications on Nios II with great simplicity. The Nios II compiler tool chain is based on the standard GNU GCC compiler, assembler, linker, and makefile facilities [47]. Also, this kit provides many example software designs that can be examined, modified, and re-used in new applications. As a result, applications can be developed and debugged without further knowledge of Altera technology beyond the Nios II software development tools. Nonetheless, familiarity with Altera hardware development tools gives a deeper understanding of the reasoning behind the Nios II software development environment.

The *Nios II integrated development environment* (IDE) [48] is the software development graphical user interface (GUI) for the Nios II processor, in which EMMA's embedded program was built up and debugged. All software development tasks can be accomplished within the Nios II IDE, including editing, building, and debugging programs. However, software development processes can be scripted and executed independently of the GUI, if necessary. Besides, the Nios II IDE provides a consistent debugging platform that works for all Nios II processor systems. The Nios II processor's JTAG debug module (see section 5.2) provides a single, consistent method of communication with Nios II IDE through a JTAG cable.

The development kit also supports the popular MicroC/OS-II real-time kernel [49] and a Lightweight TCP/IP stack (lwIP) [50]. MicroC/OS-II is a simple preemptive RTOS designed for embedded applications – such as the EMMA project – because its simplicity and well-documented structure make it easy to adapt functionality to embedded software or custom computing hardware. The TCP/IP stack is built on MicroC/OS-II, and it is well-suitable for implementing network applications.

This chapter describes the embedded software running on the CPU (Nios II) of the system hardware architecture described in the previous chapter. It completes the requirements of the system model specification for the VoIP gateway design (see section 3.4). The program is written in C/C++, and its development was considerably simplified due to the available development tools.

6.2 Hardware Abstraction Layer

One cornerstone component in software development is the hardware abstraction layer (HAL) system library. It provides a consistent, hosted C/C++ runtime environment based on the ANSI C standard libraries, which is independent of the underlying hardware features of the embedded system. For example, the HAL includes generic I/O software components, and they allow writing programs that access hardware using the C standard library routines, such as *fopen()* or *printf()* [47].

Furthermore, the HAL system library contains all information related to interfacing the system's program with the processor hardware. It minimizes the need to access

hardware registers directly to control and communicate with the system's peripherals. In general, the HAL system library isolates applications from changes to the target hardware. It is generated automatically for every Nios II processor system by the tools. Thus, programs based on a HAL system library are always synchronized with the target hardware. Figure 4.1 shows the hierarchy of the embedded software showing all the components that support the system's program.

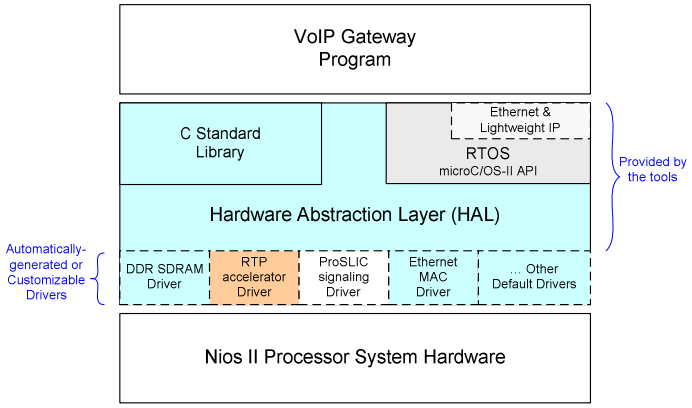

Figure 6.1 ***Architecture of the implemented embedded software [48]***

Software drivers abstract hardware details of the component so that software can access the component at a high level. The driver functions provide the software an application programming interface (API) to access the hardware. The software requirements vary according to the needs of the hardware peripheral. In particular, the driver of the RTP acceleration custom peripheral (see section 5.3) includes routines to initialize the hardware, read data, and write data. The driver defines a register map for each slave port that is accessible, and it relates them with a symbolic base address (see figure 6.2). The peripheral's driver consists of a C-header file that declares macros for reading and writing in each of the peripheral's registers.

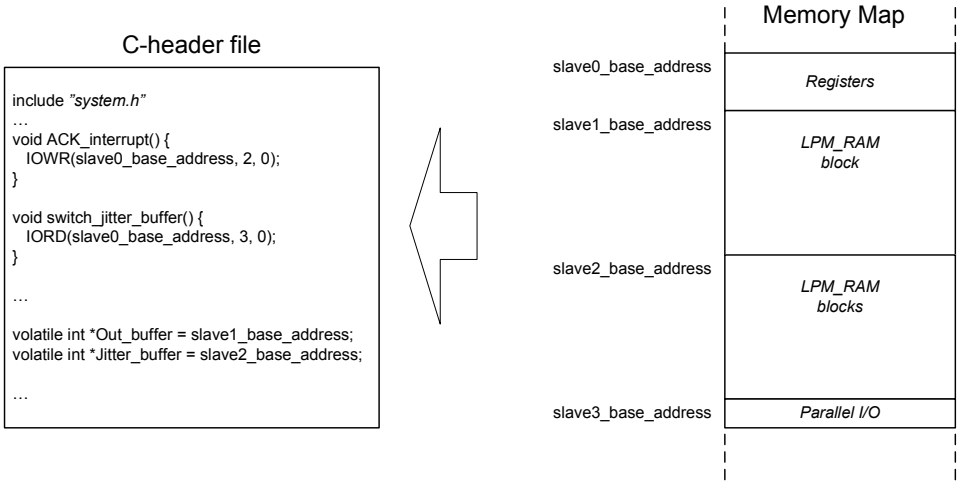

Figure 6.2 ***Simple Illustration of driver design based on the peripheral memory map***

The MicroC/OS-II kernel is another main component, since the system's program is designed to run on top of it. MicroC/OS-II is described as a portable, ROMable, scalable, preemptive, real-time, multitasking kernel; and it provides the following

services: Tasks (threads), Message passing, Memory management, Mailboxes, Semaphores and Time management. MicroC/OS-II operates over the HAL system library (see figure 6.1). Therefore, the system's program is also able to obtain all the underlying facilities provided by the HAL system library as well as it becomes portable and resistant to changes in the underlying hardware.

The Lightweight IP (lwIP) is a small-footprint implementation of the TCP/IP suite that was originally written at the Swedish Institute of Computer Science, but now is being actively developed by a team of developers distributed world-wide. It works on a RTOS and implements the standard UNIX Sockets API [51], which is very useful and comfortable for software developers. The lwIP represents the Packet Handling Unit according to the system model specification (see section 5.4), and it accomplishes all the requirements for the current version of the EMMA project:

- IP including packet forwarding over multiple network interfaces
- Internet control message protocol (ICMP) for network maintenance and debugging
- User datagram protocol (UDP)
- TCP with congestion control, RTT estimation and fast recovery and fast retransmit
- Dynamic host configuration protocol (DHCP)
- Address resolution protocol (ARP) for Ethernet
- Standard sockets for application programming interface (API)

Naturally, the Nios II processor system must also contain an Ethernet interface in order to use this lwIP. At present, the Altera-provided lwIP driver supports only the SMSC lan91c111 MAC/PHY device [52], which is the same device that is included on the *Nios II development board*.

6.3 Multitasking Program structure

The design process for real-time applications involves splitting the work into tasks, which are responsible for only a portion of the whole system's program. In embedded systems, a task – also called thread – is a simple program that thinks it has the CPU all to itself. Each task is assigned a priority, its own set of CPU registers, and its own stack area in memory. In essence, a task is a piece of code that can run or stop at any time; it stops in order to wait for an event to happen, or if it is preempted by another task with higher priority. The system's program is composed of five relevant tasks, which are described in table 6.1. Furthermore, these tasks are able to synchronize and intercommunicate.

Task Name	***Function***	***Priority Num.***(*)
SIP_task	Executes the SIP FSM as well as the message parsing and generation function. This task includes functions that fill the necessary fields in response SIP messages, set up a phone call and provide an interface towards the user. It is also responsible to send SIP messages by using the standard socket API, when it is required by the FSM.	10 (min.)
sip_receive_task	Helps *SIP_task* with the reception of SIP messages through the standard socket API. This task is blocked until a SIP message arrives. When a SIP message arrives, the task is reactivated (if a higher priority task is not ready to run), and it stores the SIP message in memory, which is then used by *SIP_task*.	9
ProSLIC_task	Access the registers inside the ProSLIC device to obtain their status or to modify them. Relevant information from those registers is conferred to *SIP_task*.	8
RTP_send_task	Interacts with the RTP output buffer block (see section 3.3.3) and sends the contained RTP packets through the standard socket API. The packet delivery is performed, when an interrupt is produced by the RTP acceleration peripheral.	7
RTP_receive_task	Receives the RTP packets coming from the network and queues them into the RTP Jitter Buffer block (see section 3.3.4). It is blocked until an RTP packet arrives. Its execution is reactivated due to any incoming RTP packet, and then it preempts any other lower-priority task.	6 (**)

(*) In MicroC/OS-II the maximum priority is assigned to the lowest number.
(**) There are other tasks with higher priority, but they are assigned by the RTOS for handling network operations related to the lwIP library.

Table 6.1 ***Description of the software tasks***

The first three tasks interact very closely, and together they compose the functionality of the system's main control unit. *SIP_task* carries out the heaviest processes, but they are not time-sensitive. On the other hand, *RTP_send_task* and *RTP_receive_task* are simpler, since they are designed only to interface with the RTP hardware acceleration block (see section 5.3), designed to do most of the work.

The main reason behind splitting this application into tasks is to execute different pieces of code at different priority levels. With a preemptive kernel as MicroC/OS-II, the execution of the tasks is deterministic because the kernel always executes the highest priority task that is ready to run. On the other hand, the hardware interruptions can achieve a similar behavior that the tasks but they are usually reserved to very time-sensitive instructions. A hardware interrupt preempts any task, and upon the completion of its interrupt service routine (ISR), the highest priority task ready to run (not necessarily the interrupted task) resumes its execution.

In this context, *SIP_task* is assigned the lowest priority because it contains algorithms that do not have real-time demands. Conversely, the highest priorities are given to *RTP_send_task* and *RTP_receive_task*, which execute time-sensitive operations. Since *ProSLIC_task* is responsible of gathering information from an external device that may change its status at any time, it gets a higher priority than *SIP_task*. *RTP_receive_task* and *sip_receive_task* obtain greater priorities than their respective counterparts because they are supposed to preempt upon the arrival of a new packet; otherwise they are inactive.

In MicroC/OS-II, each of the described tasks is written inside an infinite loop that can be in any of five states (see figure 6.3): dormant, ready, running, waiting (for an event), or ISR (interrupted). The dormant state corresponds to a task that resides in memory but has not been available to the kernel. A task is ready when it is able to execute but its priority is less than another task that has control of the CPU at that moment. Naturally, a task is running when it has control of the CPU. A task is waiting when it requires the occurrence of an event (for example, waiting for an I/O operation to complete). Finally,

a task is in the ISR state when an interrupt has occurred and the CPU is in the process of servicing the interrupt. [49, pp. 37]

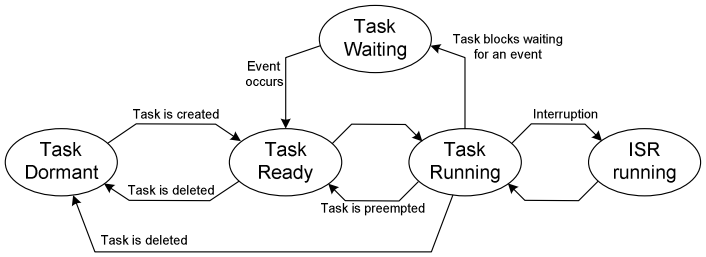

Figure 6.3 ***Task states in MicroC/OS-II*** [49, pp. 39]

Multitasking is the process of scheduling and switching the CPU between several sequential tasks (see figure 6.4). It maximizes the use of the CPU and also allows modular construction of applications. When a multitasking kernel decides to run a different task, it saves the current task's context (CPU registers) in memory. After this operation is performed, the new task's context is loaded from memory, and then the CPU resumes the execution of the new task's code. This process is called a context switch.

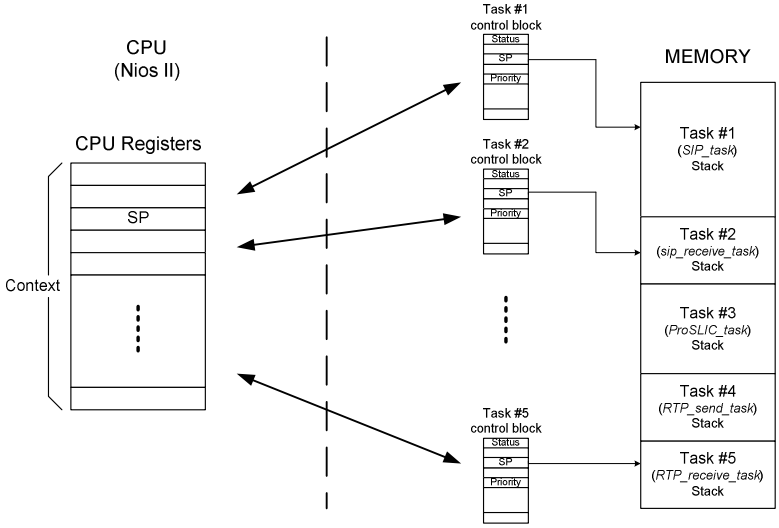

Figure 6.4 ***Context Switch***

The stack size for each task is determined according to the processes that each task executes and the variables that are involved. *SIP_task* allocates the larger stack in memory because it is the most complex task.

6.4 Intertask Communication

The described tasks were designed to share information continuously. Therefore, intertask communication is essential in the development of the embedded software. Information can be communicated between tasks in two ways: through global data or by sending messages.

There is a global variable that holds the information of the system's IP address, and it is fundamental for the other tasks. This IP address is obtained automatically during the system boot by a default DHCP task included in the lwIP. The program's tasks are able to access the global variable and thus to create network sockets, but the variable is never changed.

In that case, the global variable fulfills its purpose because it is allocated in memory before the other system's tasks start running, and it is never changed. However, during the normal execution of a multitasking application, the tasks are not aware when a global variable is changed, unless a *semaphore* [49, pp. 52] is employed to signal the occurrence of this change. Additionally, *message mailboxes* and *message queues* [49, pp. 60] can be utilized as a reliable mechanism to convey information among tasks. Therefore, the program's tasks make use all these communication mechanisms, exploiting the best of them for different purposes.

6.4.1 Synchronization between task and ISR

In RTOS, a semaphore is a mechanism used to: control access to shared resources; signal the occurrence of an event; or allow two tasks to synchronize their activities. A semaphore is the key that a piece of code acquires in order to continue its execution, so a task has to request the semaphore in order to execute that piece of code. If the semaphore is already in use – by another task, the requesting task is suspended until the semaphore is released.

MicroC/OS-II permits only three operations on semaphores: initialize (CREATE), wait (PEND) and signal (POST). A task trying to get a semaphore performs the PEND operation. Another task releases the semaphore by performing the POST operation. MicroC/OS-II allows the highest priority task waiting for the semaphore to get it. If a task that is ready to execute has a higher priority than the task that is running and releasing the semaphore, a context switch occurs and the higher priority task resumes execution.

A semaphore called *RTPSendIRQSem* is employed to synchrone the RTP acceleration hardware block's interrupt with *RTP_send_task*, because no data is exchanged. In this case, the respective ISR only occurs for a very short period in order to indicate *RTP_send_task* that an event has occurred: namely, a new RTP packet is ready at the output buffer. In fact, the semaphore is drawn as a flag to indicate that it is used to signal the occurrence of an event (see figure 4.5); using a semaphore for this type of synchronization is called *unilateral rendezvous* [49, pp. 57].

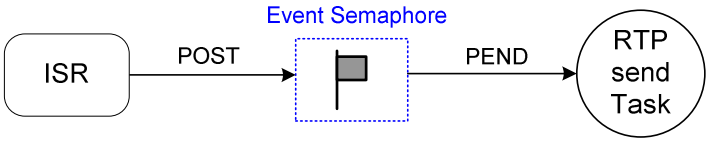

Figure 6.5 ***Synchronization between RTP_send_task and interrupt issued by hardware [49]***

As a consequence of the event signaling by the semaphore, *RTP_send_task* becomes ready to run. Therefore, it gets control of the CPU when the ISR completes its short execution, unless *RTP_receive_task* is running (it has higher priority). The task that was running before the ISR occurred is preempted, and *RTP_send_task* is resumed. Figure

6.6 shows an illustration of this scenario, in which *SIP_task* is the lower priority task that is running before the hardware interruption occurs.

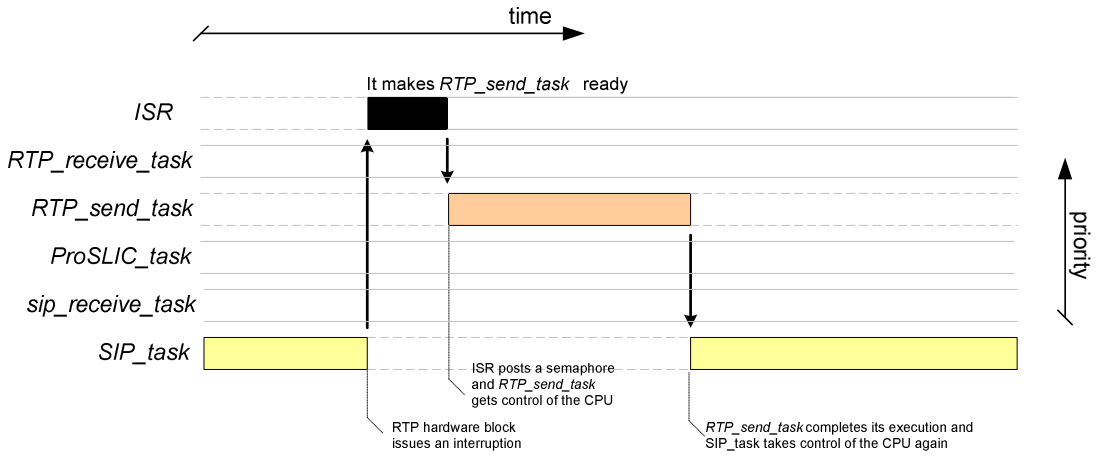

Figure 6.6 ***Execution profile for RTP_receive_task when SIP_task is running***

In this example (figure 6.5) *SIP_task* is interrupted, and then the CPU jumps to the ISR. The ISR handles an event (post a semaphore) that makes the *RTP_send_task* ready to run. At this point, the *RTP_send_task* executes the standard *sendto()* function [51] in order to transmit the content of the RTP output buffer (see section 5.3.3) towards the network. When *RTP_send_task* is done, it waits for the semaphore to be signaled again (which happens every 30ms). The kernel then sees that *SIP_task* needs to execute, and another context switch is done to resume its execution.

6.4.2 Intercommunication between two tasks

Through a service provided by the kernel, a task or an ISR can deposit a message (pointer) into a message mailbox, which is actually is a pointer-size variable. Consequently, one or more tasks can receive messages through that message mailbox. Both the sender and receiving task agree on the type of variable that is related to the pointer.

A task desiring a message from an empty mailbox is suspended and placed on the waiting state until a message is received. The kernel accepts that the task waiting for a message to specify timeout. If a message is not received before the timeout expires, the requesting task is made ready to run, and an error code (indicating that a timeout has occurred) is returned to it. A mailbox becomes empty (null pointer) when a task extracts the message from it.

MicroC/OS-II kernel provides the following mailbox services:

- Initialize the contents of a mailbox (CREATE). The mailbox initially might or might not contain a message.
- Deposit a message into the mailbox (POST).
- Wait for a message to be deposited into the mailbox (PEND).

- Get a message from a mailbox, but do not suspend the requesting task if the mailbox is empty (ACCEPT). If the mailbox contains a message, the message is extracted from the mailbox.

ProSLIC_task reads the registers inside the ProSLIC device, which provide status information of the analog telephone. Hence, *ProSLIC_task* gets to know whether the analog telephone's buttons have been pressed (DTMF detection) and if the phone is up or down (off-hook or on-hook respectively). Also, *ProSLIC_task* writes control information on some of those internal registers, so the ProSLIC device is configured to produce electrical signals [35] towards the analog phone in order to: make it ring; activate informational tones; and, activate the sampling and transmission of the voice. Besides, *SIP_task* has to be aware of that information, since it contains the system's main control algorithms. Thereafter, both *SIP_task* and *ProSLIC_task* exchange this control and status information; *SIP_task* provides control, while *ProSLIC_task* indicates the status. Then, an 8-bit variable (see figure 6.7) is employed to pass information between them by using a message mailbox mechanism.

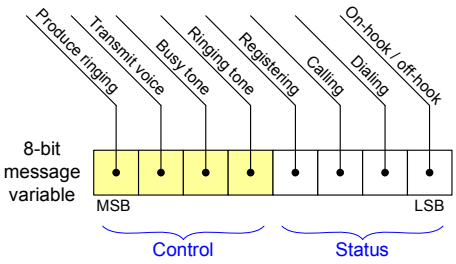

Figure 6.7 ***Control and status message variable***

Two message mailboxes are used to synchronize the activities of both tasks (see figure 6.8) besides conveying the mentioned informational variable; this mechanism is similar to the *bilateral rendezvous* with semaphores [49, pp. 58]. A message in the mailbox indicates that an event has occurred, and an empty mailbox indicates no event. *ProSLIC_task* always waits until *SIP_task* has posted data in the control mailbox; when it happens *ProSLIC_task* preempts because of its higher priority. Conversely, *SIP_task* uses the ACCEPT command, so it does not suspend its execution if *ProSLIC_task* has not posted the status mailbox.

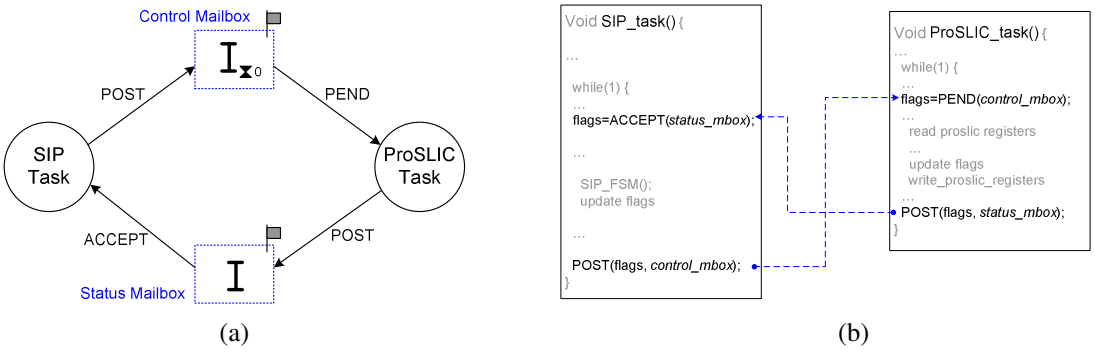

Figure 6.8 ***Intercommunication between SIP_task and ProSLIC_task by using two mailboxes; (a) Symbolic representation; (b) C code illustration***

In figure 6.8, the mailbox is represented by an I-beam and the timeout is represented by an hourglass; the “0” next to the hourglass indicates that the task waits forever for a message to arrive. The small grey banner indicates that the mailbox is also used to signal the occurrence of an event; these symbols are based on [49, pp. 57].

6.4.3 Queuing messages

A message queue is basically an array of mailboxes, and thus it is used to send one or more messages between tasks. Generally, the first message inserted in the queue is the first one extracted from the queue (FIFO). As with the mailbox, a task desiring a message from an empty queue is suspended until any message is received. Kernels typically provide the same mailbox services for queues.

A message queue, *sip_receiveQ*, is used to communicate *sip_receive_task* with *SIP_task*. In order to receive packets from the network, *sip_receive_task* uses the standard *recvfrom()* function [51], which blocks the task upon the occurrence of a network-related event: the arrival of a new SIP message. When a SIP message arrives, *sip_receive_task* preempts *SIP_task*, then it allocates a buffer in memory storing the SIP text message, and finally it puts the pointer – related to the buffer – in the message queue. The *sip_receiveQ* is requested by *SIP_task* (see figure 4.9) with the non-blocking command ACCEPT, so *SIP_task* continue running even if the queue is empty.

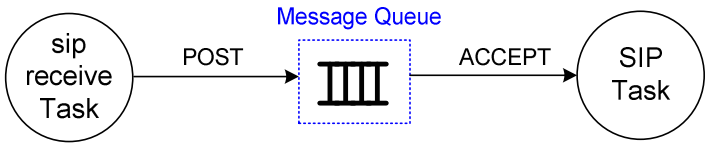

Figure 6.9 ***Communication between sip_receive_task and SIP_task with a message queue***

If there is a pointer in *sip_receiveQ*, it is extracted from the queue by *SIP_task*. Subsequently, the SIP message related to that pointer is analyzed by the parsing functions, which are based in the open source library OSIP [33]. The parsing functions operate over a complex string of characters, and they consist of algorithms that perform many string comparisons as well as memory allocation. They generate a logical data structure from the SIP message containing its relevant information. *SIP_task* is actually a very time consuming task due to the parsing functions. After the SIP message is analyzed by the parsing functions, the resulting logical data structure is fetched into the SIP FSM [18], which subsequently takes some actions. Either a response SIP message is generated, or certain control information is sent towards *ProSLIC_task,* or a change in the FSM is produced.

The FIFO structure provided by a message queue is well suitable for a scenario, in which data transmission happens more rapidly than the processing of the task that requests the data. Although *sip_receive_task* is typically idled, sometimes two or more SIP messages arrive consecutively within a very short period of time. *SIP_task* would not able to parse and analyze them in a proper time, if a mailbox mechanism were used. Upon to the arrival of two consecutive SIP messages, the second one would overwrite the first one in the mailbox before *SIP_task* processes it.

6.5 Memory Utilization

Nios II IDE offers an option that calculates the amount of memory occupied by certain program after its compilation. The designed embedded program is compiled in Nios II IDE without optimizations in the code size [48]. The results show that the program occupies 861 Kbytes in total. This number comprises both the program code memory as well as initialized data. Table 4.2 shows detailed information of the memory consumed by the program, which is obtained from Nios II IDE.

Item	***Description***	***Bytes in Memory***
.exceptions	Exception handling code	432
.text	Program code	387972
.rodata	Static Variables	20416
.rwdata	Dynamic variables	9224
.bss	Non-initialized Variables	464316
	Total Memory	882360

Table 6.2 ***Memory consumed by the embedded software***

In addition to the memory employed by the program, all of the described tasks require some portion of the memory to implement their stack. The stack as well as dynamically allocated variables with the *malloc()* function [47] are not initialized, they are created and initialized during runtime. The stack size corresponding to each task is listed in Table 4.3. Those numbers are assigned deliberately according to the requirements of each task. The more complex is the task, the more memory that is assigned; *SIP_task* obtains the maximum stack size because it needs to allocate more functions and data in memory.

Task	***Stack size (Kbytes)***
SIP_task	32
sip_receive_task	8
ProSLIC	4
RTP_send_task	4
RTP_receive_task	4
Total Task stack memory	52

Table 6.3 ***Memory required for task stack and heap***

7 Prototyping and Verification

The designed system is prototyped with the *Altera Nios II Development Board* that has an Altera Cyclone II FPGA, and with the ProSLIC development board from Silicon Labs [46]. Both evaluation boards are interconnected with a shielded parallel cable, and thus they are able to work together; this cable is plugged into the parallel expansion port J15 of the FPGA board [41, pp. 30]. The ProSLIC development board provides an interface with an analog telephone, while the FPGA evaluation board contains the core of the system as well as it interfaces the system with an IP-based network.

The designed SOC hardware architecture (see section 3.2) is programmed into the FPGA through an USB Blaster download cable [53, pp. 16]. The FPGA pin assignment and configuration are performed at the Quartus II GUI. After the hardware architecture is settled, the embedded software is downloaded into the Nios II processor system that is inside the FPGA; it is done at Nios II IDE, which also uses USB Blaster download cable. Thereafter, the system is complete and able to work and be tested in a real lab environment.

7.1 Lab components

In order to verify the correct functioning of the EMMA prototype, a testing environment is implemented with some network appliances. Basically, this lab consists of a small LAN placed within the Ericsson Enterprise infrastructure. The LAN is composed of an Ethernet hub, a PC desktop, an IP-telephone, and the EMMA prototype with an analog telephone plugged to it (see figure 7.1). PoE were used to power up the EMMA prototype.

The IP-telephone is also based on the SIP standard, and it works as the EMMA counterpart. Both IP-clients, the SIP telephone and the EMMA prototype, are aimed to call each other. They register and interact via a proxy server according to the specifications of the SIP standard [18]. The proxy server is called Asterisk, because it runs the open source software that has the same name [30]. The server is located in other geographical area but inside the Ericsson Enterprise's network, so the EMMA prototype and the SIP-phone are able to access it through the hub (see figure 5.1). In fact, all the communication among SIP UAs is based on Asterisk; it serves as a point at which the user agents are globally reachable.

Actually, Asterisk is a complete PBX in software developed by Digium. It runs on Linux and provides all of the features from a PBX and more. Asterisk offers both classical PBX functionality and advanced features. Also, Asterisk interoperates with traditional standards-based telephony systems and Voice over IP systems. It performs PBX call switching, codec translations, and various applications. Furthermore, it is available for free in source code under the GNU Public Licence. Asterisk allows connecting calls between many users, which may arrive from various hardware and software interfaces. [30]

Asterisk comprises many VoIP protocols, and can interoperate with almost all standards-based telephony equipment using relatively inexpensive hardware; it has

support for SIP and H.323, as both client and gateway. Asterisk uses codec modules for the encoding and decoding of various audio compression formats used in the telephony industry. For testing purposes, the current version of the EMMA project only utilizes the SIP and G.711 codec features of Asterisk.

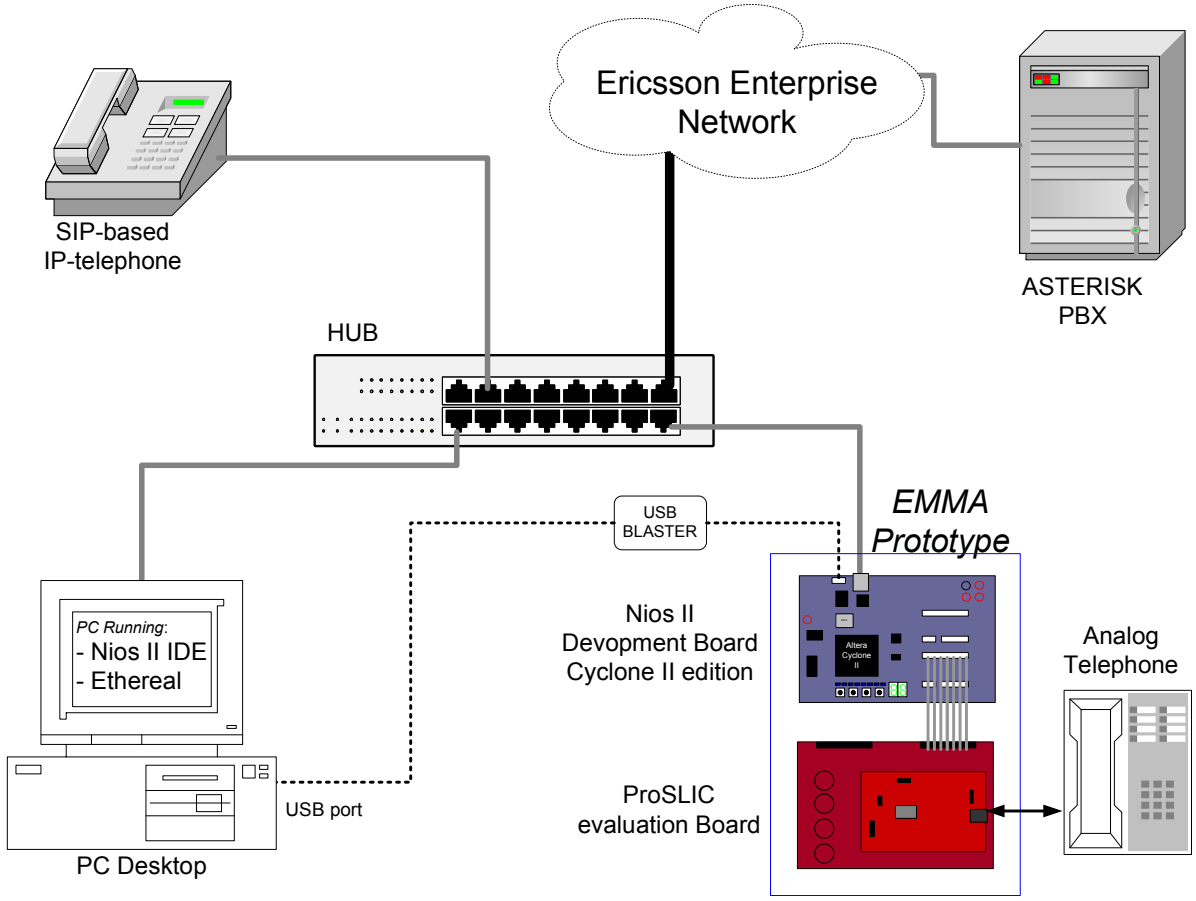

Figure 7.1 ***Components included in the Lab to test the EMMA Prototype***

7.2 Monitoring the SIP User Agents

The purpose of the PC desktop is twofold. First, it runs the Altera development tools, and thus it is responsible of programming the FPGA through the USB Blaster. Also, it makes possible to debug and analyze the Nios II processor behaviour in real time with Nios II IDE. On the other hand, the PC is used for observing – through its network card adapter – the packet flow produced by the IP-clients inside the small LAN as well as the packets coming from outside (see figure 7.1). For this purpose, a special program called Ethereal is employed. Ethereal watches all traffic going to and from the SIP phone and the EMMA prototype; however, it does not disrupt the network traffic as it shown in figure 7.2.

Ethereal is indeed a network analyzer, and it has a feature-rich GUI that eases the study of network and provides clear results. Ethereal is packed with features that are comparable to a commercial network analyzer. When a network analyzer reads data from the network, it needs to know how to interpret what it is seeing, and display the output in an easy to read format; this is known as protocol decoding. Like other network

analyzers, Ethereal supports over 480 protocols. Ethereal reads packets from the network, decodes them, and presents them in an easy to understand format [54]. Ethereal's default fields include: packet number, time, source address, destination address, and the name and information about the highest-layer protocol.

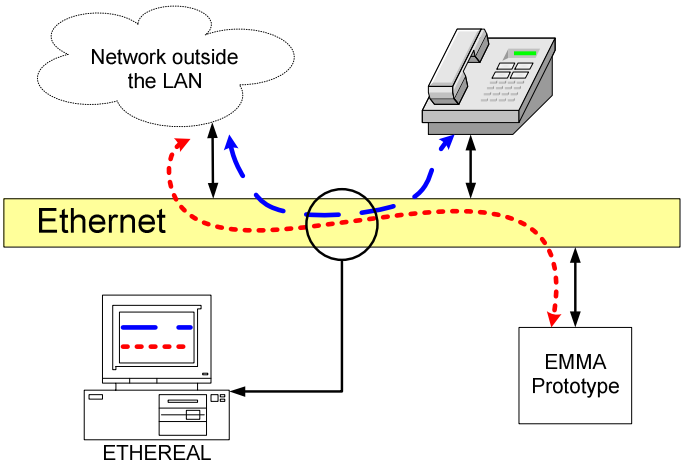

Figure 7.2 ***Analyzing the LAN with Ethereal***

In this context, the hub is very well exploited. Ethernet hubs permit that the IP-clients share an Ethernet LAN physically. A network of hubs is therefore called a "Shared Ethernet" or a "Collision Domain". This means that only one IP-client is allowed to proceed with a transmission of a frame at any one time. Also, a single hub is able to connect a group of equipments operating only at the same speed, and each of them has to share a proportion of the available network bandwidth.

The network analysis is then possible using a hub as showed in figure 7.1. The PC desktop running Ethereal is able to see the packets when the other IP-clients are transmitting. In fact, a hub does not recognize the addresses in the header of a frame, and therefore is unable to identify which port to send to. Thus, every frame is sent to every output port. This is in contrast to a bridge, switch or router, each of which only forwards a packet if the destination of the packet corresponds to a system reachable via the output interface. The testing of the EMMA prototype would not be able to be done with other network appliances, such as a switch or a router.

7.3 Packet Flow Analysis

The correctness of the SIP signaling between the user agents is verified. There is a close interaction between the SIP UAs and Asterisk, and thus all the attention was given to the packets sent to and from it. Besides, the SIP messages, which are sent and received by the EMMA, are also printed on the console window of Nios II IDE, while it runs in debug mode. The current state of the system is also seen on the LEDs and the seven-segment display provided by the evaluation board, and thus it is known when certain SIP messages produce changes on the system.

The first method that is checked is the SIP registration, which happens when the SIP UA powers up or re-starts. The EMMA prototype sends a REGISTER request to Asterisk, and immediately Asterisk replies an OK that grants its participation in the

network; it means that all the other SIP UAs are aware that the EMMA prototype is present, and the communication is accepted. The EMMA prototype is registered in Asterisk with the number 8104, whereas the SIP telephone with 8627. These numbers are actually the SIP user names employed in some fields of the SIP messages (see section 2.3.3).

A phone call starts when the user picks up the phone and dials the number 8627 and # is pressed. As a result, an INVITE request is sent to Asterisk. Then, Asterisk verifies the correctness of the message and sends it to the SIP telephone. The SIP telephone is able to send informational messages, and when its phone is picked up an OK response is sent to Asterisk. Naturally, Asterisk checks the message and forwards it to the EMMA. Afterwards, a similar process happens when the EMMA prototype produces an acknowledge message (ACK), as seen in figure 7.3, and finally the transmission of the media packets begins. When any of the phones is put down, a BYE request is produced and Asterisk replies with an OK message that terminates the communication.

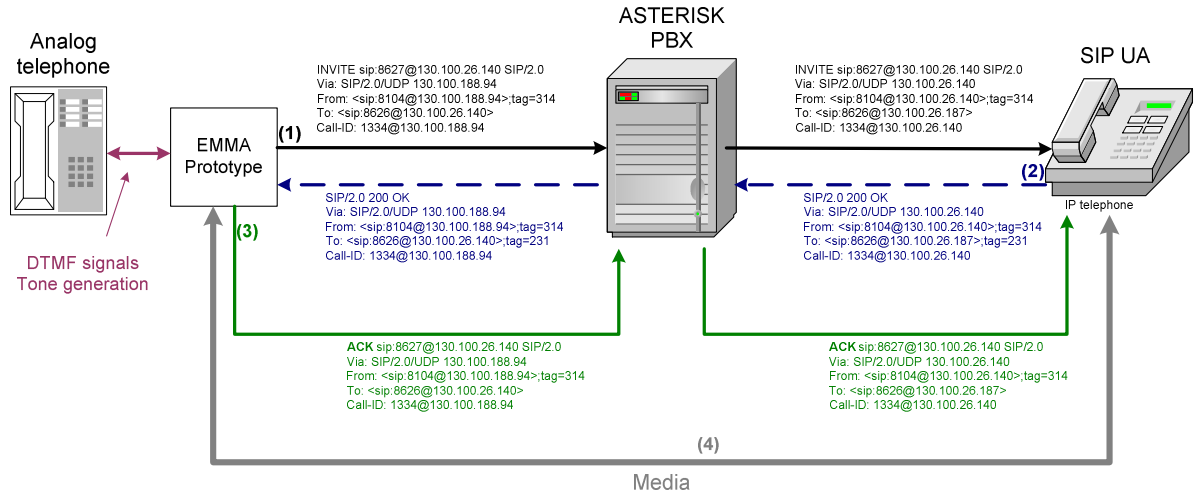

Figure 7.3 ***SIP messages involved when establishing a telephone call***

The transmission of the RTP packets is also checked and proven correct. The RTP audio packets produced the right sequence number and timestamp. The RTP packets coming from the SIP telephone are also received and played properly. Therefore, audio is successfully conveyed, and it is listened in the headsets of the SIP telephone and the adapted analog telephone. In addition, these RTP audio packets can be captured in Ethereal and saved in a file, which can be later decoded and plotted for further analysis in Matlab.

Ordinarily, SIP UAs send to the media to each other directly after the signaling has occurred. Nevertheless, Asterisk is able to receive media packets and retransmit them at the same or different pace. For instance, both of the UAs send RTP packets that contain 30 ms of voice; Asterisk receives those RTP packets and retransmits new packets with a length of 20 ms but with a higher speed. This feature provides flexibility and better control over the media that is transmitted.

8 Power-over-Ethernet

The presented work involved studied of different Power-over-Ethernet (PoE) designs that lies as ground for further development of the EMMA-design.

8.1 Definition

PoE enables enterprises to power network devices directly over the existing data connection. The PoE sourcing equipment detects the presence of a device that needs power and injects the applicable current into the data cable. No changes are required to the existing wiring or cable systems.

PoE emerged first with wireless LAN implementations in enterprises. It allowed IT managers to install wireless access points in locations were regular power connections were not available or were too difficult or too expensive to install.

The development of this technology was further advanced when VoIP solutions emerged on the enterprise market in the late nineties. To meet the strict availability requirement of VoIP it was necessary to have an alternative power source for IP-phones. In case of a power loss, alternative power for operation is provided to the phones over the existing data network infrastructure, which solves one of the major challenges for VoIP solutions.

But this technology is not limited to powering wireless LANs or providing back-up power capabilities to IP-phones. PoE can also be used to provide power to other devices, which cannot be powered by regular power sources for physical or financial reasons. Many enterprises are, for instance, interested in expanding the use of PoE by powering devices like security cameras and security card scanners remotely over the network.

Actually, the use of PoE provides financial and technical advantages to enterprise's networks. Delivering an electrical supply and data communications over a single standard LAN cable eliminates the need for connecting each Ethernet terminal to both an electrical socket and a data outlet, thereby reducing a corporation's installation and maintenance costs. It also enhances system reliability by maintaining the service during power interruptions.

8.2 The 802.3af Power-over-Ethernet Architecture

To enable enterprises to power different vendor's devices with PoE equipment, a standard was set within the IEEE. The IEEE 802.3af DTE Power via MDI working group released the standard for PoE in the spring 2003.

The PoE architecture consists of two elements: the power-sourcing equipment (PSE) and the powered device (PD). A PD is a device that is drawing power or is requesting power over the data link. Examples for PDs include wireless access points, security cameras and IP phones.

The standard allows for devices which may use up to 12.95 watts of power. The PSE is the equipment that provides energy to the PD. Its main functions are to probe the link segment for PDs, supply power to the link segment only if a PD is detected, monitor the power on the link, and remove power from the link when power limits are exceeded, a PD is disconnected or no longer requests power. PSE devices provide up to 15,4 watts of power to the cable plant at 48 volts. Some power is lost in the worst-case cable plant leaving just under 13 watts of usable power for attached devices.

To prevent harm to non-compatible Ethernet devices by PoE technology, the 802.3af standard defines a powered-device detection method. Power will only be applied by PSE if a PD has been successfully detected. This detection method is independent of the status of the data link and has been very effective at preventing power from being sent to devices inadvertently plugged into Ethernet outlets with PSE devices attached.

The PSE can also classify a PD based on information provided by the PD before power is applied. The classification defines the maximum power required by the PD during the operation and allows PSE devices to better manage limited power resources.

The 802.3af defines three ways to send the power thru the network, either can the spare pair be used (Fig 8.1 alt B) or the data pair (Fig 8.1 alt A) in the CAT5 cable. In the third way a feeder is put in the transmission path (Fig 8.1 alt C).

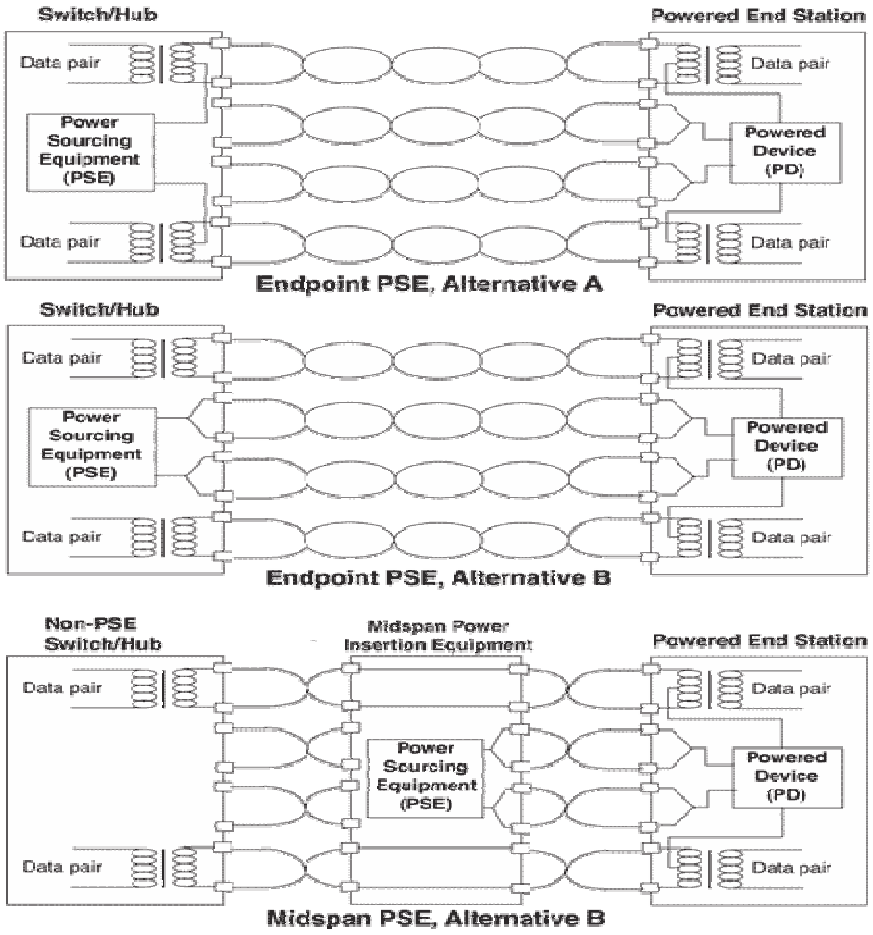

Figure 8.1 ***Power architecture***

8.3 The steps in detection method

8.3.1 Detection Phase

When a PoE-enabled Ethernet cable is plugged into a PD, the PSE interrogates the device to determine if it is PoE-enabled. This period is termed the detection phase. During the detection phase, the PSE applies a voltage ramp to the PD and looks for characteristic impedance from the load (25 kΩ). If the correct impedance is not detected, the PSE assumes that the load is not PoE-enabled and shuts down the PoE sending end. The system operates as a standard Ethernet connection. If the signature impedance is detected, the PSE moves on to the classification phase. The signature identification voltage is a ramp voltage between 2,5V and 10V. A 24,9k ohm resistor provides the correct signature impedance for detection [9].

8.3.2 Classification Phase

If the PSE verifies that the PD is PoE-enabled by detecting the correct classification impedance, it continues to increase the voltage applied to the PD. Between 14,5V and 20,5V, the PD draws a classification current. This classification current determines the amount of power required by the PD during normal operation. The PSE sends information to a controller that monitors the system's power budget. The controller permits the PSE to power up the PD if sufficient power is available from the system budget [9].

8.3.3 Turn on Phase

The final stage is the turn on phase. Now that the presence of a PD has been confirmed and the PSE is certain it is not about to destroy a non PoE-enabled device it ramps the voltage up to its full capability in order to feed the device's dc/dc converter. This final level can be anywhere between 36 and 57 Vdc (48V nominal), so a dc/dc converter is necessary to provide the actual voltage output the PD needs. It is important to note that the device is required to maintain a certain minimum level of power consumption to let the PSE know that it is still functioning properly [9]. Figure 8.2 describes the whole flow that related to the detection method.

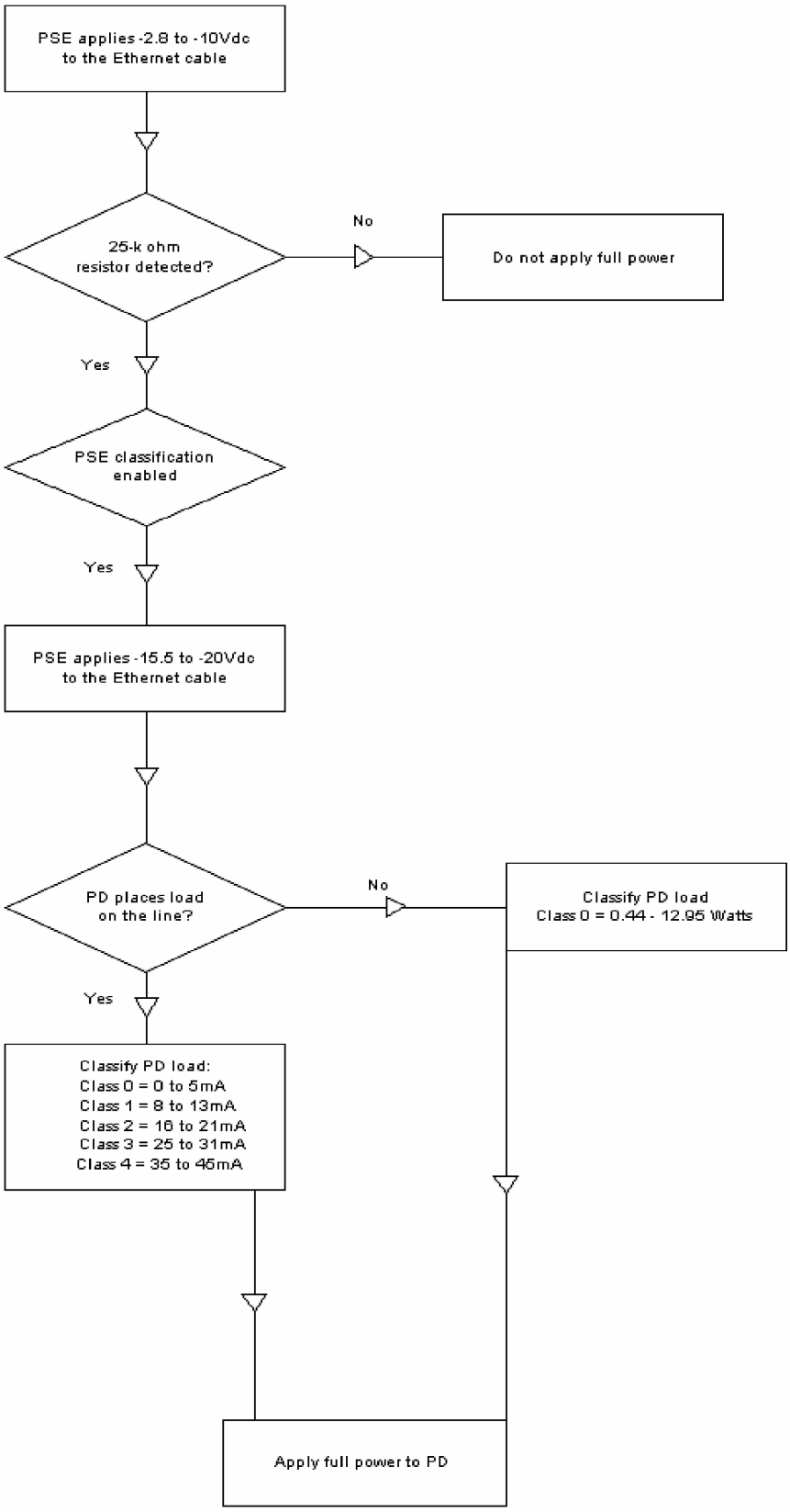

Figure 8.2 ***Power-over-Ethernet detection flow***

8.4 Design issues

The three most important desiderata for the EMMA PoE design are low cost, high efficiency and size. Low cost which is a must for any product that will see mass usage, implies simplicity of the design.

For the power levels under consideration, the flyback converter is the natural choice. Not only is it the simplest and cheapest of all isolated designs in its most basic form, but also it is best suited to generate multiple outputs. This design can be configured to generate voltages of either or both polarities of any magnitude.

The first requirement regarding to the EMMA PoE design was the total power consumption. It is not allowed to exceed 12.95 watts of power, because that is the limit for what the PSE can provide to each port. On the other hand, the circuit must also provide the voltages needed for the EMMA design; they are 3,3V for the ProSLIC and FPGA, and 1,5V core voltage for the FPGA. Additionally, the efficiency of the PoE solution is another important issue, because it is not wanted to waist any power unnecessarily. The total power consumption for the EMMA design is estimated 5W (see table 8.1).

Vendor	***Power***
Cyclone II	2W
Si3210 ProSLIC	3W

Table 8.1 ***Power consumption for EMMA-design***

There are new microchip solutions on the market, which represent a good way to obtain a good design in terms of size and efficiency. Two interesting companies in this field are Texas Instruments and Linear Technology, because they have the smallest components that are on the market right now. Particularly, Texas Instruments has released a really small microchip solution for PoE that is capable of decreasing the size of our EMMA design with half. Besides, Ericsson has very good prices and support from these companies.

8.4.1 Flyback architecture

The voltages coming from the PSE are too high for most electronic designs, and thus a transformer is needed to get the right voltage levels. The best design choice for a transformer is the flyback architecture (figure 8.3), because it requires only one magnetic component and it has very high efficient achieved.

Although flyback converters are usually operated in discontinuous conduction mode (DCM) at low power levels, the best efficiency is obtained in the continuous conduction mode (CCM) where, for a given out put power, the RMS current in the primary side FET is smaller.

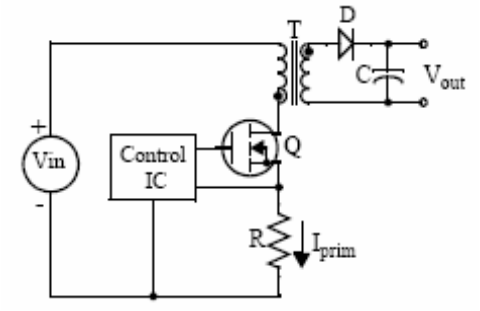

Figure 8.3 ***Flyback architecture***

There are two arguments generally used to justify the use of the DCM operation. A smaller transformer is employed, and the DCM design is simpler from the mathematical point of view. However, at PoE energy levels the transformer cores for CCM operation are small enough to be acceptable in most applications. In cases where the output voltage ripple needs to be low, an LC filter can be added to the output of the supply. Therefore, the CCM flyback design is preferred PoE power supply architecture.

The use of this design has been greatly facilitated by the fact that some of the major magnetic vendors are now offering families of PoE flyback transformers that are designed and optimized for the different PoE classes. For example when using a CCM flyback supply designed for an output power of 13W with a diode rectifier, the power supply will typically be about 84% efficient at input voltage of 48V and an output voltage of 3,3V. The efficiency will increase or decrease as the output voltage respectively increases or decreases.

8.5 Design proposals

The two designs that are proposed have flyback architecture with a high efficiency. They are also small designs with few external components as well as easy to modify to get exactly the right voltage levels for the EMMA's PoE design.

8.5.1 Texas Instruments design PMP717

This design uses the TPS2375 PoE PD Controller to get all the necessary detection, classification, inrush current limiting, UVLO and switch FET control necessary to be compliant with the IEEE802.3af PoE standard [56].

8.5.2 Linear design note 338

This design proposal has an efficient around 80-90% for the voltage levels that is used in the EMMA design. It is built around two main components and they are LTC 4257 and LT 1725 [57].

LTC 4257

The LTC 4257 provides with a complete signature and power interface for a device operating in an IEEE802.3af PoE system. It has the 25 kΩ signature resistor, classification, inrush current limiting and UVLO [57].

LT 1725

This is a monolithic switching regulator controller and it is specifically designed to isolate the flyback design [57].

9 Cost and size estimation

One part of this thesis work was to make an analysis of the product cost and size for the EMMA design. The cost analysis is based on the digital and analogue blocks of the EMMA design, and how many components each block contains. The cost that has been analyzed is the component cost except the box that is going to hold the EMMA-design. This cost analyze is only a rough estimation since the prices on memories and Power-over-Ethernet parts shift from day to day. The cost of a product, which is aimed for mass production, is always an important issue. To see the total prize and the total quantity of components for one complete EMMA-design look in Appendix A.

The size is also an important issue because, in the beginning of the thesis work, the size was one of the criteria. The size should be as small as possible so that the EMMA design should melt in to the surrounding, and not be considered as an extra box that occupies unnecessary space on the desk or wherever it is placed.

9.1 Size estimation

The size analysis is only an approximation, but it gives a good overview regarding the size of the EMMA design. The goal of this size estimation is to make a small appliance, which should be easy to produce in moderate amounts.

The design of the PCB layout has been done concerning the size of the components in each block and what function each block has. The best solution for the PCB is obtained with one two-sided PCB that includes the ProSLIC-channel and PoE block on one side, while the Cyclone II block is placed on the other side. The estimated size for the complete EMMA mechanical design is 60x100x30 mm.

9.1.1 PCB

The PCB that is suggested will give a total of 9000mm² available area; 4500mm² on each side. That will be more than enough to fit all the blocks on one double-sided PCB, as shown in table 8.1.

Component	***Width (mm)***	***Length (mm)***	***Area (mm²)***
ProSLIC channel	20	60	1200
Power-over-Ethernet	35	70	2450
Cyclone II	50	60	3000
Total			**6650**

Table 9.1 ***Total use of area on PCB***

9.1.2 Summary of Size estimation

The size is just a rough estimate, because the Power-over-Ethernet components are developing all the time, already now there is a circuit from Texas Instruments that is able to reduce the Power-over-Ethernet block with half the size. This is achieved by building the primary side inside an IC. *Håkan Karlsson* Analog Field Application Engineer at Texas Instruments provided this information.

9.2 Cost analysis

To make the cost analysis as good as possible the different design blocks in the EMMA-design were analyzed one at the time, in order to get a good overview over how many parts that each designs holds.

Prices are based on Ericsson's deals with the different vendors. And when some components that are used were not in that list, the best prices that could be found on the Internet were used. All the prices are based on quantities of 1K.

9.2.1 ProSLIC block

When this block was analyzed, the ProSLIC-channel design [2] that is used in the MX-ONE was used as a reference design, and the total amount of components in this block are 51. The total cost for this block is 68 SEK.

9.2.2 Power-over-Ethernet block

For the Power-over-Ethernet part the PMP 717 [55] design from Texas Instrument is used. Texas Instrument has provided us with document that gives exactly all the components that are needed in the PMP 717 design. The total amounts of components in this block are 68. The total cost for the block is 144 SEK.

9.2.3 Cyclone II design

In this cost analysis the logical elements, the flash memory and the SDRAM are included. Prizes regarding the logical elements have been received from *Daniel Stackenäs* at Altera because Ericsson has special prices and they are not meant to go public. The total price for this block is 103,75 SEK.

9.2.4 PCB-design

To get a good estimation of the price of the PCB design, the different sizes of the blocks were measured, and estimation over the PCB design was made. The criteria's for the PCB design were: size 90 x 50mm, double sided with 35/35µm copper thickness. The company that provided with the cheapest price for the PCB was found in the Internet, and it was *Multi-Teknik Mönsterkort AB*. They sent a quotation with the prize for the PCB design. The prize for the PCB design were 8.8 SEK/card.

9.2.5 Summary Cost analysis

The total cost for one EMMA design, by just looking at the cost for the components, is going to be slightly higher than 325 SEK. Since the cost is just a rough estimation, and the prices for PoE components and FPGA's are becoming cheaper by time which will make this price decrease.

9.3 Comparison between EMMA and existing products

The comparison gives a good overview at how the EMMA design is standing against similar products that are out on the market nowadays. The adaptors that are chosen as examples for the comparison are *Grandstream Handytone* ATA-286[1] and Sipura Phone Adapter SPA-1001 [2]. These two adaptors have similar features as EMMA except from Power-over-Ethernet witch they do not support.

	EMMA	***SPA-1001***	***HandyTone ATA-286***
Size (w*l*h) (mm)	60*100*30 (estimated)	63*94*28	56*90*28
Codec's	G.711 (A-law)	G.711 (A-law and mu-law), G.726, G.729 A, G.723.1	G.723.1, G.729 A/B, G.711 (A-law and mu-law), G.726, G.728
Power supply	Power-over-Ethernet	Power Adaptor	Power Adaptor
Protocols	SIP, RTP, DHCP, UDP and IP	SIPv2, IPv4, ARP, DNS, DHCP, ICMP, TCP, UDP, RTP, RTCP and SNTP	SIPv2, TCP, UDP, IP, RTP, RTCP, HTTP, ARP, RARP, ICMP, DNS, DHCP, NTP and TFTP
DTMF	Yes	Yes	Yes
Prize	325,47SEK (components)	471,93 SEK	428,95SEK

Table 9.2 ***Comparison***

As the table shows the EMMA is a potential competitive solution with further development.

10 Conclusions and Future Work

10.1 Conclusions

Ericsson MD110 is a technology that enables the evolution of enterprise communications towards all-IP as well as the integration of different services. The EMMA project becomes one component of MD110's evolution. It is also aimed to provide more flexibility and scalability within the MD110 distributed architecture that is based on a set of LIMs

The EMMA project achieves the implementation of the SIP standard. It consists on a simple telephony application, which requires only a reduced set of the fields on the SIP messages. This implementation basically requires two elements: a parsing unit for processing long strings of characters; and an FSM for handling the sequence of events involved on a typical telephone call.

A typical SOC contains multiple cores that are usually selected for specific applications such as VoIP. Nowadays, there are many SOC for VoIP applications on the market, and they are mostly developed in ASICs. These VoIP SOC differ from each other by the type of functions and voice-processing algorithms they support. Alternatively, the EMMA project proposes a different VoIP solution with the SOPC design methodology. It integrates a fully reconfigurable digital architecture, which comprises a CPU and other intellectual property blocks, inside an FPGA.

A proper design flow was followed, which includes system model specification, hardware and software co-design, and finally prototyping and verification. The system modeling methodology that was employed enables modularity and scalability as well as good understanding of the system requirements. It also helped to preserve consistency through all blocks, and thus it avoided unnecessary iteration making the design process more efficient and faster.

A clear hardware/software partition was performed, which aimed to accelerate the real-time processing of voice packets with dedicated hardware, while it implemented VoIP protocols with embedded software running on a soft-processor core.

The system architecture was assembled at SOPC Builder obtaining a high degree of scalability and easy adaptability for changes in the components within the system architecture. It utilizes advanced features of the Avalon switch fabric. Avalon is a high performance on-chip bus that offers a standard interconnection for all the system's components as well as it makes the adding and removing of modules a very straightforward task. Therefore, the designed system architecture can be easily improved later.

The Altera Nios II soft-processor core was chosen as the CPU of the system, since it is a powerful and flexible computation core that is also compatible with many existing peripherals. The standard Nios II – configured at SOPC Builder – showed good performance and low resource utilization. Moreover, the Nios II IDE permits that engineers develop complex programs, including a powerful RTOS with a TCP/IP stack, without having a deep understanding of the underlying technology.

The Power-over-Ethernet circuit that was chosen to be tested in the design was Texas Instruments PMP717. The voltages that it provided the design with were 1.5 volts and 3.3 volts.

The system was prototyped with a Cyclone II FPGA in the *Nios II Development Board, Si32xPPT-EVB and Texas Instruments PMP717 circuit.* A test environment was created with a small LAN inside Ericsson Enterprise. The prototype was tested with an SIP-based telephone and a SIP proxy server. The test proved the correctness of the system design and verified all the involved VoIP protocols and that the entire design can be powered thru Power-over-Ethernet. Thus, it is concluded that the EMMA works as a basic VoIP gateway properly.

An industrial application was developed in short time with the SOPC methodology, and it proves the feasibility of developing low-cost products with fully programmable devices. All the knowledge gathered and the experience acquired using the design tools will enrich the Design Department at Ericsson Enterprise. Besides, the obtained results are a strong basis for future versions of the EMMA project. With further development, the EMMA project can become an actual network test instrument within the MD110 technology.

10.2 Future Work

The current version of the EMMA project has a limited set of VoIP protocols. Therefore, one major field of effort to be addressed in the future is the integration of more protocols (for example, RTCP, H.323) as well as support for more vocoders. The multitasking environment provided by MicroC/OS-II will allow the addition of these protocols as new tasks. However, the vocoders should be implemented as dedicated hardware blocks inside the custom peripheral for hardware acceleration. Keeping the existing hardware blocks, the vocoders or any other voice processing unit must be placed between the ProSLIC PCM interface and RTP buffers.

As a consequence of adding more complex vocoders, the control unit of the jitter buffer must be improved. The current support for G.711 is simple, and it only required a lightweight implementation of a jitter buffer. This improvement entails the inclusion of more memory units in order to store more incoming RTP packets in the system, but it is straightforward due to the generic VHDL coding employed (see section 3.3.4).

The addition of more media processing units is possible in the handy Cyclone II that was employed. The results of the compilations on the FPGA showed that there is a considerable amount of LEs and available embedded multipliers to implement some DSP cores. For instance, an echo canceller could be implemented as a dedicated hardware block. It will also allow using the EMMA in voice conversations that have long delays.

The SIP message-parsing engine that was employed is not considered the best option for an embedded application. OSIP's basic operations over string characters are based on dynamic memory allocations. When used iteratively, dynamic memory allocation may cause memory partition, unless special care is taken on the underlying program. Hence,

it is recommended to try other libraries that implement the basic operations of the SIP standard more reliably.

The maximum slack was found at the DDR SDRAM controller, which is provided by Altera. In case that the system's clock frequency is increased, the synthesis tools may not meet the timing constraints. Hence, another controller for the SDRAM or different synthesis tools could be tried with the purpose of achieving better performance.

The lwIP works very well for simple network applications, especially if the applications do not demand high-speed packet transmission. Since the lwIP is implemented as a software component, it shares the processor resources with other tasks. Then, it might not respond properly when there are many tasks using the standard socket API. It might produce some delays in the transmission and reception of packets, which is not good for a real-time application. Therefore, it is recommended to use a dedicated hardware block that implements the TCP/IP stack.

The chosen Power-over-Ethernet circuit works well. But there can be much more improvements to the size of the Power-over-Ethernet circuit.

Last but not least, it is essential to put special care in the usability when building an actual product. The EMMA should support all the services provided by the traditional telephones. In fact, this is a general recommendation for any future product of Ericsson Enterprise. Eventually, it is the customer or user who will qualify the product, and he or she will expect to feel comfortable when using it.

11 References

[1] Ericsson AB, "*Evolution towards converged services and networks*", White paper, February 2005, ref. 284 23-3017 Uen Rev A

[2] Ericsson AB, "*Enterprise communications trends, needs and opportunities. Delivering business advantage through multidimensional convergence*", White paper, June 2004, ref. 284 23-2978 Uen Rev A

[3] Ericsson AB, "*MD110. The best of both worlds in one dynamic workforce*", White paper, March 2004, ref. EN/LZT 102 3666 RB

[4] Ericsson AB, "MD110. Convergence Communication System", Statement of Direction, September 2003

[5] Gedal S., "System ASB 501 04. Description", Ericsson AB, September 2005, Ref. 1551-ASB 501 04 Uen

[6] Bernice B., "ASBU 501 – A digital SPC Voice / Data Switching System", IEEE Journal on selected Areas in Communications, Vol. SAC-3, NO. 4, July 1985

[7] Robert C., Lars R., "Communications Protocol for Switching Systems", Assignee: L M Ericsson, Stockholm Sweden, July 1994

[8] Ericsson AB, "Evolution to All-IP: the Service Layer", White paper, February 2005, ref. 284 23-3015 Uen Rev A

[9] IEEE, "Standard 802.3af™", June 2003

[10] Telco Systems, "Access 241. Analog telephone adaptor", Datasheet, June 2005

[11] Cisco Systems, "Cisco ATA 186 Analog Telephone Adaptor", Datasheet, 2004

[12] Nekoogar F. and Farnak N., "From ASICs to SOCs: A Practical Approach", Prentice Hall PTR, first edition, May 2003, ISBN: 0130338575

[13] Texas Instruments, "IP phone solutions TNETV1050/1055", Product bulletin, 2004

[14] Altera Corp., "Cyclone2 Device family datasheet", November 2004

[15] Feit S., "TCP/IP: Architecture Protocols & Implementation with IPv6 & IP security", McGraw-Hill Professional Book Group, 1998, ISBN: 0-07-022069-7

[16] Zhang Y., "SIP-based VoIP network and its interworking with the PSTN", IEEE, Electronics & Communication Engineering Journal, December 2002

[17] Mehta P., Prof. Udani S., "Overview of Voice over IP", University of Pennsylvania, February 2001, Technical Report MS-CIS-01-311

[18] Schulzrinne H. et al., "SIP: Session Initiation Protocol", IETF draft, 2003

[19] Vlaovic B. and Brezocnik Z., "Packet Based Telephony", IEEE, EUROCON'2001, International Conference on Trends in Communications

[20] Chan T. et al., "Building Residential VoIP Gateways: A tutorial", Texas Instruments, VoIP Business Unit.

[21] Lucent Technologies, "Voice over Internet Protocol. Voice Quality of Service", Whitepaper, September 2004

[22] International Telecommunication Union, "ITU-T G.711: Pulse Code Modulation (PCM) of Voice Frequencies", November 1988

[23] International Telecommunication Union, “ITU-T G.723.1: Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s”, March 1996

[24] International Telecommunication Union, “ITU-T G.729: Coding of Speech at 8kbit/s using Conjugate-Structure Algebraic-code-Excited Linear-Prediction (CS-ACELP)”, November 1996

[25] Schulzrine H., Casner S., Frederick R. and Jacobson V., “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, January 1996

[26] Gorry Fairhurst , “Communications Engineering Course: Medium Access Control”. Available: www.erg.abdn.ac.uk/users/gorry/course/lan-pages/mac.html,

[27] Granström P., Olson S. and Peck M., “The future of communication using SIP”, Ericsson AB, check review No. 1, 2002

[28] Bengtsson J., “SIP – H.323 Signaling Gateway”, Thesis Project, Ericsson Enterprise AB, 2003

[29] Schulzrinne H. and Rosenberg J., “Signaling for Internet Telephony”, IEEE. Proceedings. Sixth International Conference on Network Protocols, 1998

[30] Asteriks, The Open Source PBX, Website: www.asterisk.org

[31] Nurmela T., “Session Initiation Protocol”, Seminar on Transport of multimedia streams, University of Helsinki

[32] Handley M., Jacobson V., “SDP: Session Description Protocol”, RFC 2327, April 1998

[33] Foeckel S., Kranz M., Kuthan J. and Sisalem D., ”OSIP: An Open Source SIP Architecture”, GMD-Fokus, Germany

[34] Zou H. et al., “Prototyping SIP-based VoIP Services in Java”, IEEE paper. International Conference on Communication Technology Proceedings, ref. WCC - ICCT 2000

[35] Silicon Laboratories, “PROSLIC™ Programmable CMOS SLIC/CODEC with ringing/battery voltage generation”, Datasheet, September 2001, Rev. 1.21

[36] Xilinx, “Microblaze Reference Guide”, Embedded Development Kit 7.1, April 2005

[37] Altera Corp., “Creating Multiprocessor Nios II. System Tutorial”, April 2005

[38] Lysecky R. and Vahid R., “A study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning”, IEEE, Department of Computer Science and Engineering University of California, 2005, ref. 1530-1591/05

[39] De Micheli G., Gupta R., “Hardware/Software Co-Design”, Proceedings of the IEEE, Vol. 85, No. 3, March 1997

[40] New Product Development Solutions (NPD) Consulting, Software/Hardware Codesign. Website: http://www.npd-solutions.com/swcodesign.html

[41] Altera Corp., “Nios II Development Board. Cyclone II Edition Reference Manual”, May 2005

[42] Altera Corp., “Avalon Interface Specification”, May 2005

[43] Altera Corp., “Quartus II version 5.0 Handbook. Volumen 4: SOPC builder”, May 2005

[44] Altera Corp., “Nios II Processor Reference Handbook”, May 2005

[45] Altera Corp., “Quartus II version 5.0 Handbook. Volumen 1: Design and Synthesis”, October 2005

[46] Altera Corp., “Quartus II version 5.0 Handbook. Volumen 2: Design Implementation & Utilization”, October 2005

[47] The GNU Project web server, www.gnu.org

[48] Altera Corp., "Nios II Software Developer's Handbook", May 2005

[49] Labrosse J., "MicroC/OS-II. The Real Time Kernel", Second edition, CMP Books, 2002, ISBN: 1-57820-103-9

[50] Dunkels A., "Full TCP/IP for 8-bit Architectures", Swedish Institute of Computer Science,

[51] Stevens R., "Unix Network Programming. Vol. 1: Networking APIs: Sockets and XTI ", Second Edition, Prentice Hall, 1998, ISBN 0-13-490012-X

[52] SMSC, "LAN911C111. 10/100 non-PCI Ethernet single chip MAC/PHY", Datasheet, July 2005, Rev. 1.8

[53] Altera Corporation, "Nios II Development kit. Getting started user guide", May 2005

[54] Ethereal, network protocol analyzer. Website: www.ethereal.com

[55] Altera Corporation,

www.altera.com/products/devices/cyclone2/features/power/cy2-power-compare.html

[56] DStrasser, "POE PowerSupply",

focus.ti.com/lit/ml/slvr239a/slvr239a.pdf, 2004

[57] Jesus Rosales, "Power over Ethernet Isolated Power Supply Delivers 11,5Wat 90% EfficiencyDesignNote338", www.linear.com/pc/downloadDocument.do?navId=H0,C2,C1161,C1188,D4580, 2004

[58] Grandstream, http://www.grandstream.com/HandyTone_Spec.pdf

[59] Sipura Technologies, http://www.sipura.com/Documents/SPA-1001.pdf

Appendix A

Qty	Refdes	Part Number	Value	Description	Vendor	Size	Prize (1k) sKr
1	C18	Std	68pF	Capacitor, Ceramic, 68pF, 50V, C0G, 5%	Farnell	603	0,33
2	C10, C26	Std	270pF	Capacitor, Ceramic, 270pF, 50V, C0G, 5%	Farnell	603	0,66
1	C23	Std	330pF	Capacitor, Ceramic, 330pF, 50V, C0G, 5%	Farnell	603	0,33
1	C21	Std	1000pF	Capacitor, Ceramic, 1000pF, 50V, X7R, 10%	Farnell	603	0,46
2	C3, C22	Std	0.01uF	Capacitor, Ceramic, 0.01uF, 50V, X7R, 10%	Farnell	603	0,66
2	C25, C27	Std	0.033uF	Capacitor, Ceramic, 0.033uF, 25V, X7R, 10%	Farnell	603	0,66
1	C19	Std	0.1uF	Capacitor, Ceramic, 0.1uF, 16V, X7R, 10%	Farnell	603	0,27
2	C20, C24	C1608X5R0J105K	1uF	Capacitor, Ceramic, 1uF, 6.3V, X5R, 10%	Farnell	603	0,44
2	C2, C4	C2012X5R0J106M	10uF	Capacitor, Ceramic, 10uF, 6.3V, X5R, 20%	Mouser	805	6,02
3	C12, C13, C14	C2012X5R0J226M	22uF	Capacitor, Ceramic, 22uF, 6.3V, X5R, 20%	Mouser	805	17
2	C9, C11	Std	0.1uF	Capacitor, Ceramic, 0.1uF, 100V, X7R, 10%	Elfa	1206	0,6
1	C17	C3216X5R1C106M	10uF	Capacitor, Ceramic, 10uF, 16V, X5R, 20%	Mouser	1206	4,55
2	C7, C8	C3225X7R2A105K	1uF	Capacitor, Ceramic, 1uF, 100V, X7R, 10%	Mouser	1210	4
1	C5	1812CG222MAT	2200pF	Capacitor, Ceramic, 2200pF, 2KV, X7R, 20%	Farnell	1812	11,5
2	C15, C16	EEVFK0J331P	330uF	Capacitor, Aluminum, 330uF, 6.3V, 20%	Farnell	8*6.2mm	12,4
1	C6	EEVFK1J220P	22uF	Capacitor, Aluminum, 22uF, 63V, 20%	Farnell	8*6.2mm	4,77
2	D6, D7	BAS16LT1		Diode, Switching, 200mA, 75V, 225mW	Farnell	SOT-23	0,86
2	D1, D2	HD01		Bridge Rectifier, 100V, 0.8A, Glass Passivated, SMD	Diodes	MINI DIP4	1,62
1	D3	MBRS340T3		Diode, Schottky, 3A, 40V	Farnell	SMC	2,14
2	D4, D8	MURA120T3		Diode, Rectifier, 1A, 200V	Mouser	SMA	1
1	D5	1SMA58AT3	58V	Diode, TVS, 58V, 400W	Mouser	SMA	0,77
2	J1, J2	520252		Connector, Jack, Modular, 8 POS	Mouser	0.705*0.82	7
1	L1	DO1608C-472	4.7uH	Inductor, SMT, 4.7uH, 1.5A, 90milliohm	Mouser	0.26*0.09	4,86
2	R10, R11	Std	0	Resistor, chip 0 Ohms	Elfa	603	0,68
1	R12	Std	10	Resistor, chip 10 Ohms, 1/16W, 5%	Elfa	603	0,02
1	R9	Std	20	Resistor, Chip, 22 Ohms, 1/16W, 5%	Elfa	603	0,02
1	R13	Std	249	Resistor, Chip, 220 Ohms, 1/16W, 1%	Elfa	603	0,03
2	R17, R24	Std	499	Resistor, Chip, 470 Ohms, 1/16W, 1%	Elfa	603	0,06
1	R14	Std	1K	Resistor, Chip, 1.00K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R22	Std	3.01K	Resistor, Chip, 3.01K Ohms, 1/16W, 1%	Elfa	603	0,03
2	R18, R19	Std	9.09K	Resistor, Chip, 9.09K Ohms, 1/16W, 1%	Elfa	603	0,06
1	R20	Std	10K	Resistor, Chip, 10.0K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R23	Std	20K	Resistro, Chip, 22.0K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R21	Std	23.7K	Resistor , Chip, 22.0K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R3	Std	24.9K	Resistor, Chip, 27.0K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R16	Std	40.2K	Resistor, Chip, 39.0K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R6	Std	178K	Resistor, Chip, 180K Ohms, 1/16W, 1%	Elfa	603	0,03
1	R5	Std	10	Resistor, Chip, 10 Ohms, 1/10W, 5%	Elfa	805	0,03
1	R7	Std	549	Resistor, Chip, 560 Ohms, 1/10W, 1%	Elfa	805	0,02
1	R8	Std	57.6K	Resistor, Chip, 56K Ohms, 1/10W, 1%	Elfa	805	0,02
1	R4	Std	100K	Resistor, Chip, 100K Ohms, 1/10W, 1%	Elfa	805	0,02
1	R15	Std	0.82	Resistor, Chip, 0.82 Ohms, 1/4W, 5%	Elfa	1206	0,38
1	Q2	MMBT3904TT1		Bipolar, NPN, 40V, 200mA, 200mW	Farnell	SC-75	0,35
1	Q1	Si3440DV		MOSFET, N-ch, 150V, 1.5A, 375 milliohms	Mouser	TSOP-6	5,4
1	T1	H2019		Xfmr, Center-tapped, Voice Over IP	Mouser	0.50*0.37	19,9
1	T2	PA1032		Transformer, Flyback	Mouser	0.49*0.45	6,24
1	U4	H11A817AS		IC, Optocoupler, 5300V, 80-160% CTR	Mouser	0.435*0.21	1
1	U5	TLV431ACDBVR		IC, Shunt Regulator, 1.24V ref, 6V, 10mA, 1%	Texas Instruments	SOT23-5	1,9
1	U1	TPS79518DCQ		IC, LDO, 500mA, 1.8V	Texas Instruments	SOT223-6	8
1	U2	TPS2375PW		IC, IEEE 802.3af POE PD Controller	Texas Instruments	TSSOP8	9,63
1	U3	UCC3809P-2		IC, Economy Primary-Side Controller	Texas Instruments	MSOP-8	7,32
1				Si3210 complete channel (Document nr 1911-ROJ2080031/1)			68,7
1				Mönsterkort 90*50 (mm)	MULTEK		8,8
1				Cyclone II (5,531 LE)			26
1		73-851-07	16Mbit	Flashmemory	ELFA		67,6
1			64Mbit	SDRAM	More Electronics	TSOP	10,15
							325,47