Practical Experience Teaching Embedded Systems

Carlos Almeida
Instituto Superior Técnico - Technical University of Lisbon
Avenida Rovisco Pais
1049-001 Lisboa, Portugal
carlos.r.almeida@ist.utl.pt

ABSTRACT

The area of embedded systems and their interconnection is of utmost importance nowadays. The technological evolution, and widespread use, of small electronic devices with processing and communication capabilities, creates the potential for the development of new applications in several different domains. Areas such as industrial control, automotive, home automation, surveillance systems, sensor networks applied to the monitoring of wild life, environment, or buildings, are examples of such potential and ubiquity.

The importance of this area can not be ignored in an Electrical and Computer Engineering Course. It is important to make students able to master the technologies related to embedded systems, including "hands-on" experience developing and testing real systems, and dealing with specific application environments with low resources and close hardware/software interaction.

This paper describes the author's experience teaching a course of embedded systems in an Electrical and Computer Engineering Course.

Keywords

Embedded Systems Education, Embedded Systems Development

1. INTRODUCTION

The area of embedded systems and their interconnection is of fundamental importance nowadays [1]. The technological evolution that one has seen in the last few years, in what concerns electronic devices of small size with processing and communication capabilities, creates the potential for the development of new applications in several different domains. Devices such as cellular phones, PDAs, control systems with "smart" sensors and actuators, are examples of those embedded systems.

The improvement of processing and communication capabilities, associated with the reduction of size and cost, allows the proliferation of an almost infinity of these small embedded systems, that can be interconnected, contributing to a common global goal. Areas such as industrial control, automotive, home automation, surveillance systems, sensor networks applied to the monitoring of wild life, environment, or buildings, are examples of such potential and ubiquity.

The importance of this area can not be ignored in an Electrical and Computer Engineering Course [5, 3]. It is important to make students able to master some of the technologies related to embedded systems. There are several aspects deserving consideration. In addition to an overall idea about the main characteristics and requirements of embedded systems, it is also important to know how to use the technologies and have "hands-on" experience developing and testing real systems. Learning how to use specific development and test (debug) tools, "feel" specific application environments with low resources and close hardware/software interaction, are issues of utmost importance.

This paper describes the author's experience teaching a course of embedded systems in an Electrical and Computer Engineering Course. It is organized as follows: in the next section we give an overview of the main areas of interest related to embedded systems and corresponding problems to address; in Section 3 we describe the development infrastructure used; in Section 4 we give an overview of the proposed laboratory project that addresses some of the previous referred issues; in Section 5 we discuss the results achieved so far. The paper ends with the conclusions and references to future work.

2. MAIN AREAS OF INTEREST AND PROBLEMS TO ADDRESS

2.1 Characteristics and limitations of embedded systems

An embedded computational system is characterized by being a system where the existing computational elements are integrated with the application, that is, they have a specific function related to the given application, and are not computers for generic use [15, 2]. Depending on the specific application, a given embedded system may present a set of specific characteristics related to that application. However, in a generic way, most embedded systems show a set of common characteristics/requirements. For example, they need to be concerned with:

- the right functionality and performance desired for the application, which may include a combination of real-time, reliability and fault-tolerance guarantees;
- power consumption management, mainly in situations where batteries are used, in order to increase autonomy;
- keeping, in many scenarios, system size small;
- ensure that the cost keeps low, mainly in situations where the number of units is very large.
- and all of this in scenarios of low resources, including CPU processing power and available memory.

Furthermore, when the embedded systems are interconnected, the aspects related to communications are also very important. For example, when the embedded system corresponds to a small node in a network of sensors, the way the communication is handled has a significant impact in the overall system. Depending on the used technology, we will have different characteristics in what concerns power consumption, bandwidth, covered distance, and cost, for example.

2.2 Devices and technologies

Another issue related to small embedded systems concerns the type of devices that are usually used and the related technologies. In these type of systems there is a close hardware / software interaction and the need to perform input / output (IO) operations dealing with several different IO devices. In such scenario, microcontrollers are of extreme importance as they integrate in the same chip a set of different functional blocks such as: processor, memory (RAM, flash, EEPROM), IO ports, timers, interrupt controllers, analog / digital converters, network / interface controllers (CAN, I2C, SPI, RS232, ...), etc. This approach makes it possible to build small embedded systems in an easy way and with a low cost.

Having an environment with low resources also restricts the type of user interface that is usually available. Sometimes, it is simply a few switches, some LEDs, a small LCD and a buzzer, for example. The application programmer must deal with this type of interface.

Application developing process can also be a specific issue. There is the concept of *target*. The developing environment runs on a different machine using cross development tools. After the application is built, it must be transfered to the target, possibly programming a flash memory. Sometimes, if even possible, debugging will also be done remotely. Otherwise, one must deal with a very "poor" environment where a "printf message" may need to be replaced by "a blinking LED", for example.

Many times, the application itself must be concerned with specific issues like: the need to build a clock using available timers, in order to make it possible to timestamp events; collect information from sensors; register information in non volatile memory; use of low power modes to reduce power consumption.

2.3 System support

Many target applications in the area of embedded systems are inherently concurrent – they need to perform several different tasks at the same time. If there is no support for concurrency, the application programmer must handle directly that concurrency, by providing every time, for each application, the functionality usually available in operating systems, or by coding the application in such way that all tasks are addressed by the "main" program. These solutions imply a significant overhead and increase the complexity of the application. So, depending on the available resources, at least a minimal support to multiplex several different activities is desirable. The problem is that being constrained by the available resources (including memory size and CPU power), it is not just to pick an operating system. One must make sure that there is enough "room" for the application itself. We need a small memory footprint system.

Anyway, having or not always the support of an operating system, understanding and dealing with concurrency is an important issue. Interprocess communication and synchronization must be done with care, in order to achieve the desired program correction and safety. When there are real-time requirements, it is also important to understand the way the scheduler works and how priorities are handled.

3. OVERVIEW OF INFRASTRUCTURE

In order to address the main issues presented above, we put together an infrastructure composed of several different parts that can be interconnected. We have two main targets and the corresponding developing environments. One of these targets corresponds to the part were there is a more close hardware interaction, using a microcontroller and several devices in an environment with low resources. The other target is a slightly higher level system that will be used to run a concurrent application supported by a multitasking operating system. We also have the developing environments needed to build the applications that will run on each of those targets.

For the first target we are using the demonstration board PICDEM2 [9] from Microchip (see Figure 1).

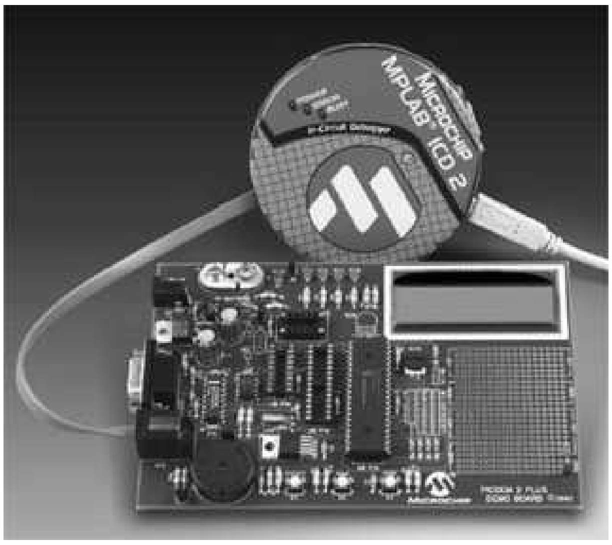

Figure 1: Aspect of PICDEM2 demonstration board, from Microchip, used in the experimental environment

This demonstration board, having a PIC18F452 microcontroller [7] that integrates flash program memory, RAM, EEPROM, several timers, an interrupt controller, an ADC converter, an USART, an I2C/SPI controller, also includes several IO devices such as LEDS, switches, a buzzer and a LCD. The board also has a temperature sensor and an external EEPROM connected by I2C. A prototype area is also available in case one wants to extend the board functionality. For this setting, as development environment, we use the integrated development environment MPLAB IDE [13] freely available from Microchip, together with a C compiler (C18 [12]) also from Microchip, that has a free student edition. The availability of free software is an important issue in academic environments, since it allows unrestricted use by students.

The other target, used for the development of concurrent applications, is an old 486 PC (although considered out-of-date for generic use, is still a powerful platform for embedded systems). As multitasking operating system we use eCos (Embedded Configurable Operating System) [14], distributed by REDHAT/eCosCentric, complemented by GNU tools (C compiler, debugger, libraries).

Both settings previous described are target environments. Software development is done using a standard PC (Pentium based) with dual boot: Windows XP (to run MPLAB IDE) and Linux (to run GNU tools and build an eCos application). In both cases this host system uses cross compilers to produce the final application that is downloaded to the respective target.

4. PROPOSED PROJECT

4.1 Introduction

Many embedded systems are developed using relatively simple platforms, with low resources, where the utilization of operating systems to support the application may not be viable. In these type of systems there is essentially the need to do some elementary processing and access several input / output devices (e.g. access sensors to collect information). However, in several other cases, the existence of support at operating system level (even with reduced functionality) may significantly facilitate the development of applications.

The laboratory project to be implemented (composed of 2 parts) has as main goal the familiarization of students with the development of embedded systems, both with medium / low complexity, using microcontrollers and without the support of an operating system, and using multitasking kernels for the development of concurrent applications.

In particular, they should acquire some expertise in the utilization of communication and synchronization mechanisms between tasks, in the context of concurrent applications, and should familiarize with other embedded systems characteristics such as access devices using the network / bus I2C (Inter Integrated Circuit), save information in non-volatile memory devices (EEPROM), utilization of analog-digital conversion, and use of serial communication RS232.

They should also be aware of aspects related to energy consumption, resorting to operation modes that will allow as much as possible to increase system autonomy.

4.2 Problem description

The project is decomposed into two distinct parts, that will be interconnected later.

The first part corresponds to a system (multifunction device, monitoring system, or alarm system) that has a rudimentary user interface (switches, LEDs, buzzer, LCD) and no operating system support. The other part, with a more elaborate user interface and processing capability, has a multitasking kernel to support a concurrent application and will allow remote access (using a RS232 serial line) to the system developed in the first part (see Figure 2).

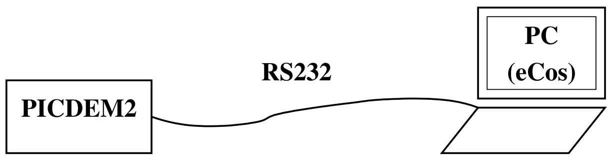

Figure 2: Part1 and Part2 are interconnected using RS232 serial communication

Part 1 - Monitoring and alarm system Overview

The first part of the project is implemented using the development board **"PICDEM 2 Plus Demonstration Board"** [9], which includes a microcontroller **PIC18F452** [7]. The application is programmed using the **C programming language** ("MPLAB C18 C Compiler" [10, 12, 11]) and the development environment "MPLAB Integrated Development Environment (IDE)" [13] from Microchip. Reading the related documentation is fundamental for the correct project implementation.

The main functions to provide are essentially:

- Clock (to keep current time and to allow timestamping of relevant events).
- Periodic sensor reading (with storing of data and the possibility of handling alarm situations).
- Basic user interface.
- Incorporation of power-saving mechanisms.
- Remote access support (using RS232 communication) to allow reconfiguration operations and data transfer (**to be implemented in conjunction with the second part of the project**).

The clock is built based on one of the timers available in the PIC, and must provide the current time on the form of hours, minutes and seconds.

The system will monitor in a periodic fashion (period `PMON`) temperature (sensor TC74 [8]) and voltage (potentiometer available in PICDEM2 developing board) values. Depending on the values collected, they may be saved in non-volatile memory (external EEPROM 24LC256 [6]), and associated with alarm situations as well. If `PMON` is zero there will be no information collected. Saving the collected values in non-volatile memory is only performed if the new value is "significantly" different from the one previous saved, and is timestamped with the current time. It is possible to save up to `NREG` registers in a "ring-buffer" (when the buffer is full, new registers replace older registers). When active, alarms are generated when the new values (temperature or voltage) reach pre-defined thresholds.

The notification of an alarm situation is done through signalization on the LCD **and** generation of a sound signal (buzzer) of duration `TSOM`.

The relevant parameters for the correct system operation (including current time) should be saved in the PIC's internal EEPROM, so as to make it possible to recover them even in a situation of temporary power-down.

The project should be implemented is such a way that the following parameters are easily configurable. Whenever possible, they are recovered from the internal EEPROM. Initially they will have the following values:

NREG	30	maximum number of registers in buffer
PMON	5 sec	monitoring period
TSOM	3 sec	duration of sound signal
TINA	2 min	inactivity time
VART	1 ^{o}C	significant variation of temperature
VARV	0.5 V	significant variation of voltage

User interface

The user interface, in this part of the project, is performed through the use of 2 switches (S2, S3), a buzzer, and a LCD (2 lines of 16 characters) [4]. The switches are used for setting the correct time of the clock, to define alarms, and to select operation mode.

Switch S3 (RB0) is used to select the desired operation, and the switch S2 (RA4) is used to change the selected fields, or activate the desired operation. More precisely, starting from normal operation mode, pressing S3 will make the cursor blink over the first modifiable field (hours), making it possible to increment its value by pressing S2. Whenever S3 is pressed the cursor will move to the next field, until reaching normal operation mode again. When S2 is pressed, the value of the field that is being modified is incremented module the range of possible values for that field, or the given function is activated.

The information presented on the LCD will have the following aspect (some fields only appear in the modification mode):

hh:*mm*:*ss*	`TV a P`
tt `C`	*V,vv* `V`

The letter "`P`" (only in modification mode) allows the activation of the power saving mode. In that operation mode the LCD will be off (and the processor will be, whenever possible, in *sleep* mode). It will return to normal mode if the user presses S3, or if an alarm occurs. The LCD should also be turned off if there is no user activity during a time greater than `TINA` (if this parameter is 0, this option will be ignored).

Letters "`TV`" (that appear in modification mode, or in case of occurrence of the respective alarm) allow the definition of alarms corresponding to temperature and voltage, respectively. When activated, current alarm values can be modified (or not) using the respective fields (temperature or voltage).

The letter "`a`" (or "`A`" – it works in "toggle" mode) shows if the alarms are inactive (or active), and in modification mode makes it possible to activate / deactivate the alarms.

The characteristics of the IO devices that are used should be consulted on the manual of the board [9], or on the "Data Sheets" of the devices.

Part 2 - Remote access and processing

Overview

Intrinsic limitations of some embedded systems make one desire to have the possibility to remote interact with the embedded system for data transfer or reconfiguration operations. In the application developed in the first part of the project the limitations in what concerns user interface are an example of such a situation.

In the second part, the application to develop will run in an environment with more resources (PC) making it possible to offer to the user a more flexible interface to interact with the system developed in the first part.

This application will have several tasks/threads (concurrent application) that will interact among them, and with the PICDEM-2 board (first part of the project) using a RS232 serial line. More precisely, there should be **at-least** three threads responsible, respectively, for the user interface, the communication with the PICDEM-2 board, and the processing of the information collected from the PICDEM-2 board.

User interface

In this part of the project that runs on the PC, there is a task responsible for the user interface that makes it possible to execute a set of commands to interact both with the PICDEM-2 board (through the communication task) and the task in charge of information processing. The commands that should be made available are the following:

Available Commands		
cmd	args	description
rc		- read clock
sc	*h m s*	- set clock
rtv		- read temperature and voltage
rp		- read parameters (NREG, PMON, TSOM, TINA, VART, VARV)
mpm	*p*	- modify monitoring period (seconds - 0 deactivate)
mti	*t*	- modify inactivity time (minutes - 0 deactivate)
ra		- read alarms (temperature, voltage)
dt	*t*	- define alarm temperature
dv	*V v*	- define alarm voltage (integer part, decimal part)
aa		- activate/deactivate alarms
trc	*n*	- transfer registers (curr. position)
tri	*n i*	- transfer registers (index)
lr		- list registers (local memory)
dr		- delete registers (local memory)
cpt		- check period of transference
mpt	*p*	- modify period of transference (minutes - 0 deactivate)
dtt	*t*	- define threshold temperature
dtv	*V v*	- define threshold voltage
pr	"t1" "t2"	- process registers (max, min, mean) between instants t1 and t2 (h,m,s)

In the first group of commands the interaction is done with the communication task, and in the last group of commands the interaction is done with the processing task. The commands in the central part (`lr` and `dr`), corresponding, respectively, to listing and deleting registers that are in the local memory, are executed accessing directly that memory region (shared by the several tasks). This memory region is organized in the form of a ring-buffer, with capacity to `NRBUF=100` registers. Register update in this memory region is done by the communication task, when registers are received. The request for register transference can be done both by the processing task and by the user interface task, that transmit those requests to the communication task.

All the commands specified above that imply communication with other tasks have a reply message (synchronous interface). In the case when the command execution is not successful, the reply message will have an error code.

Register transference commands (between PICDEM-2 board and PC) have two variants. In the first one (`trc`), only the number (`n`) of wanted registers is specified, being them (if they exist) obtained starting on the first register not yet transfered. In the second variant (`tri`), in addition to the number `n`, it is also specified the index (`i`) where the transference should start. Index zero corresponds to the oldest register in the board's ring-buffer, independently of having been, or not, transfered.

In the interaction with the processing task, the user can check and modify the period used to start a new register transference, and also define temperature and voltage thresholds to be used in the processing immediately after register reception. It can also make requests to process a set of registers belonging to a time interval defined by `t1` and `t2` (specified in the form `h m s`). If these time instants are missing, it corresponds to consider the totality of registers (or from `t1` to the end, in the case where only `t2` is missing).

Processing of collected information

The processing task is responsible both for starting the transference of registers (if this option is active) and for specific register processing. Register transference requests are performed through the communication task that will notify when the transference is done. At that time the processing task will analyze the received registers and will print on the screen the ones exceeding the defined thresholds.

The processing task also accepts requests to process a set of registers that belong to the time interval defined by **t1** and **t2** (h1:m1:s1 - h2:m2:s2). It will determine maximum, minimum and mean values. This processing is done using the registers that are in the shared memory region, and is available regardless register transference being active or not.

As stated above, the collected information (registers) should be kept in a shared memory region accessible by the communication task and by the processing and user interface tasks. The consistent access to that memory region by the several tasks is of extreme importance for the correct operation of the application.

Communication between PC and PICDEM-2

The communication between the PC and the PICDEM-2 board is done through one dedicated task (or two – reception and transmission) that will make the interface with the serial port device driver provided by eCos ("/dev/ser0"). The physical support for the communication is a **RS232** serial line with the following characteristics:

9600 baud, 8 bits, no parity, 1 stop bit.

Using the RS232 serial line a simple message exchange protocol is built. A message starts with a specific start of message code `SOM` (codes provided as appendix) and finishes with a specific end of message code `EOM`:

```
SOM <MSG> EOM
```

The "real" message (`<MSG>`) starts with the command identifier (also provided as appendix) followed by the given data associated with that command:

```
<MSG> := <CMD> <DATA>
```

Message Types		
cmd	data	description
RCLK	[h m s]	read clock
SCLK	h m s	set clock
RTEV	[T V v]	read temperature and voltage
RPAR	[pars]	read parameters
MPMN	p	modify monitoring period
MTIN	t	modify inactivity time
RALA	[T V v]	read alarms (temp., voltage)
DALT	t	define alarm temperature
DALV	V v	define alarm voltage
AALA		activate/deactivate alarms
TRGC	n [regs]	transfer registers (curr. position)
TRGI	n i [regs]	transfer registers (index)

In the above table (Message Types), both the command codes and any of the several data fields, with the exception of `regs`, have a size of 1 byte (the parameter VARV is decomposed into 2 bytes: integer part and decimal part). Every register has a size of 6 bytes (timestamp: `h m s`; temperature value: `T`; integer part of voltage: `V`; decimal part of voltage: `v`).

Meaning of data fields:

h	hours [0 .. 23]
m	minutes [0 .. 59]
s	seconds [0 .. 59]
T	temperature [0 .. 50]
V	integer part of voltage [0 .. 5]
v	decimal part of voltage [0 .. 99]
p	monitoring period (in seconds) [0 .. 99] (0 - inactive)
t	inactivity time (in minutes) [0 .. 99] (0 - inactive)
pars	parameters (NREG, PMON, TSOM, TINA, VART, VARV-int, VARV-dec)
regs	registers (size 6 bytes each - h, m, s, T, V, v)

In the above table, data fields represented in square brackets ([]) only exist in the reply message and not in the request (even though the command code is the same). In the case of commands TRGC and TRGI, the fields `n` and `i` are also part of the reply message but their values may need to be adjusted to the specific reply.

Messages whose reply has no data, or where an error has occurred in the remote execution of the command, will have the following format:

```
<MSG> := <CMD> <ERROR>
```

where `<ERROR>` can take the values `CMD_OK` or `CMD_ERROR` (see appendix).

The specified communication interface must be strictly followed so as to make it possible to interconnect components developed in an independent way, if desired.

All messages received by the communication task are routed to the task that has made the respective request, that will be in charge of printing it, if wanted. In the case of register transference commands, the task responsible for the starting of that operation will only receive the notification of operation conclusion with success or not. The registers are directly placed by the communication task in the shared memory region, as said before.

Being the screen a resource shared by more than one task, its use must be done ensuring consistency.

4.3 Project development

In the development of the project, the utilization of a modular structure and a phased testing is advised.

In the target board PICDEM-2 there will be no operating system to support the execution of the application. It is students responsibility to structure the program in a modular fashion accordingly to the several tasks that must be performed. The execution support to those tasks should be organized in the form of a "cyclic executive", resorting to the use of interrupts in the situations where that is justifiable.

Underlying all aspects of project implementation, there should be the concerning of, without jeopardizing the desired functionality, trying to optimize energy consumption, resorting to the instruction "`sleep`" (power saving mode) whenever possible.

In terms of project development phases, students can/should take advantage of the availability of a PIC controller simulator in the integrated development environment (MPLAB-IDE). The direct utilization of the development board can be postponed until after an initial phase of simulator testing.

The second part of the project is also programmed using the C programming language (in this case, gcc from GNU). The development environment is a standard PC running Linux as operating system. However, the operating system that will support the application is eCos (Embedded Configurable Operating System), being Linux only used as development system. The final application, linked with eCos, will later run in a native way in the target PC that will be reinitialized with this application/operating system.

As was suggested for the first part of the project, in the development of the second part the utilization of a modular structure with phased tests is also advisable. In that way, one can take advantage of the existence of a "Linux Synthetic Target" in eCos for the PC platform, that runs in the context of the Linux operating system. This way, it is possible to test several parts of the implementation without the need to transfer the application to the target PC. It also allows to locally run the application with the GNU debugger (gdb).

In what concerns the user interface, we do **not** want to have anything too much complex (that is not the main goal). In order to simplify the implementation, students can/should use a simple command interpreter that is made available by faculty.

The communication interface between Part 1 (PICDEM-2 board) and Part 2 (486 PC with eCos), using a RS232 serial line, is specified in more detail, and must be strictly followed, in order to make it possible to interconnect components from different students if desired.

5. DISCUSSION OF RESULTS

There are several issues arising from the design and implementation of these type of projects. One important aspect is to acquire a global idea about the technologies and environments (both development and execution environments) associated with embedded systems. The direct contact with the hardware (even if no new hardware is developed) is an enriching experience that is not usually provided in a more generic programming course.

On one hand it implies to access specific and detailed information about the devices used. This information must be obtained from "Data Sheets" and manuals, that sometimes are very large and/or are presented in very specific ways. Learn how to read this type of documentation is fundamental.

On the other hand, the debug process constitutes an important experience where the right methodology with a systematic and modular approach must be followed if one wants to achieve the desired results and overcome the inherent difficulties. Otherwise, it might be a very time consuming task, due to the fact that it is performed in a harsh environment. A "brute-force" trial-and-error approach should be avoided, and, when necessary, system components should be isolated. The use of debug tools, when available, is also recommended.

Another aspect concerns concurrency and related problems. As previous said, most embedded systems are inherently concurrent. Being able to develop correct concurrent applications using multitasking kernels is mandatory. Many students were mainly used to program sequential applications, and were not aware of potential problems arising in concurrent environments. Understanding and dealing with critical sections, synchronization, and interprocess communication is a major contribution to be prepared to develop applications for embedded systems. In situations where there are real-time requirements it is also important to be aware of how scheduling works, being able to choose the right priorities to assign to each task.

The importance of strictly following an interface specification was highlighted with the interconnection of *Part1* and *Part2* of the project. Being able to interconnect works developed by different student groups showed its importance. There were cases where this approach helped in demonstrating the full functionality of one part when the other part developed by the same group had some limitations.

In what concerns project development phases, it was also important to resort to simulation environments using the simulator in the MPLAB-IDE, and the *Linux Synthetic Target* in eCos. This approach allowed students to do some work outside the laboratory environment without the targets. This flexibility was very helpful as debugging is a very time consuming activity and there were some restrictions on laboratory hours.

Another issue concerns project size and corresponding work hours needed to develop it. The functionality / complexity that can be demanded must take into account the time available for development. This implies to address smaller problems, or provide already some components in order to make it possible to have bigger projects. Besides access to library functions available in the Microchip C18 environment (some of them adapted by faculty to the target used), other functions, built from scratch or adapted from existing ones, were made available by faculty to help in this process. The basic idea is to give more to ask more, having into consideration which are the aspects that are more important for students to address, and which are the aspects that are mainly "standard" programming.

During project development, there were also several other specific issues contributing to the enrichment of students personal experience. Some examples are:

- being aware of compiler optimizations and special compiler directives (e.g. use of *volatile* and *interrupt* attributes to deal with low level issues and hardware interaction).
- use of interrupts to improve response time in situations where there are tight timing requirements (e.g. timer interrupt to keep global clock accurate, USART interrupt to ensure that characters are not lost in the RS232 communication).
- dealing with power consumption management and associated restrictions (use of *sleep* mode; timer with external oscillator to keep it working in sleep mode; functionality versus consumption – USART interrupt associated with RS232 communication does not wakeup the microcontroller used).
- being aware of scheduling and correct use of task priorities in a concurrent application. Choosing wrong values could block the full application.

6. CONCLUSIONS AND FUTURE WORK

Embedded systems and their interconnection is an extremely important area nowadays, both from the point-of-view of engineering and from the point-of-view of research. We are surrounded by an almost infinity of quasi-invisible electronic devices that ensure a multiplicity of functions. The relevance of these pervasive systems is ever increasing. This is an area of knowledge that can not be ignored by an Electrical and Computer engineer. Direct dealing with embedded systems, discovering their main characteristics and challenges, and familiarizing with the related technologies, can make a big difference in acquired experience that will help in problem solving later on. This has been observed in cases of students doing their final graduation projects or master thesis: similar problems were seen as easy or very difficult, depending on having had, or not, this previous experience.

As future work we plan to also address other subjects and technologies. For example, we plan to introduce in the project aspects related to wireless communications. The communication between Part 1 and Part 2 can be done using Bluetooth or ZigBee as an alternative to RS232. The use of RFID is also planned. Furthermore, we also plan to improve consumption management by switching to another microcontroller with more power-saving modes and address that subject in a more integrated fashion. Building an HTTP server to provide more global remote access capabilities, is also an issue that might deserve some consideration.

7. ACKNOWLEDGMENTS

This work was partially supported by EU and FCT, through Project POSC/EIA/56041/2004 (DARIO).

8. REFERENCES

[1] C. Almeida and J. Rufino. Interconnected embedded systems: Challenges and main problems to solve. In *In Proceedings of the 6th IEEE International Workshop on Factory Communication Systems (Work in Progress Sessions)*, Torino, Italy, June 2006.

[2] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages (third edition)*. Addison-Wesley, 2001.

[3] H.-G. Gross and A. van Gemund. The delft MS curriculum on embedded systems. *ACM SIGBED Review, Special Issue on The Second Workshop on Embedded System Education (WESE 2006)*, 4(1), January 2007.

[4] Hitachi. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver).*

[5] W. A. S. Kenneth G. Ricks, David J. Jackson. Incorporating embedded programming skills into an ECE curriculum. *ACM SIGBED Review, Special Issue on The Second Workshop on Embedded System Education (WESE 2006)*, 4(1), January 2007.

[6] Microchip Technology Inc. *24AA256/24LC256/24FC256 - 256K I2C CMOS Serial EEPROM*, 2002.

[7] Microchip Technology Inc. *PIC18FXX2 Data Sheet*, 2002.

[8] Microchip Technology Inc. *TC74 - Tiny Serial Digital Thermal Sensor*, 2002.

[9] Microchip Technology Inc. *PICDEM 2 Plus User's Guide*, 2004.

[10] Microchip Technology Inc. *MPLAB C18 C Compiler Getting Started*, 2005.

[11] Microchip Technology Inc. *MPLAB C18 C Compiler Libraries*, 2005.

[12] Microchip Technology Inc. *MPLAB C18 C Compiler User's Guide*, 2005.

[13] Microchip Technology Inc. *MPLAB IDE User's Guide*, 2006.

[14] Red Hat, Inc. *eCos Reference Manual*, 2003.

[15] W. Wolf. *Computers as Components: Principles of Embedded Computing Systems Design.* Morgan Kaufmann Publishers, 2000.

APPENDIX

A. COMMAND CODES USED IN THE SERIAL COMMUNICATION

```
/* It is assumed that SOM and EOM values
   do not occur in the message */

#define SOM   0xFD  /* start of message */
#define EOM   0xFE  /* end of message */

#define RCLK  0xC0  /* read clock */
#define SCLK  0XC1  /* set clock */
#define RTEV  0XC2  /* read temperature and voltage */
#define RPAR  0XC3  /* read parameters */
#define MPMN  0XC4  /* modify monitoring period */
#define MTIN  0XC5  /* modify inactivity time */
#define RALA  0XC6  /* read alarms (temp., voltage) */
#define DALT  0XC7  /* define temperature alarm */
#define DALV  0XC8  /* define voltage alarm */
#define AALA  0XC9  /* activate/deactivate alarms */
#define TRGC  0XCA  /* transfer registers (curr. position)*/
#define TRGI  0XCB  /* transfer registers (index) */

#define CMD_OK    0     /* command successful */
#define CMD_ERROR 0xFF  /* error in command */
```