Laboratory Project
Embedded Computational Systems
IST - 2020/2021

Monitoring and Alarm System (Part 2 – eCos)

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. However, in several other cases, the existence of support at operating system level (even if with reduced functionality) may significantly facilitate the development of applications.

This part of the laboratory project has as main goal the familiarization of students with the use of multitasking kernels for the development of concurrent applications in embedded systems. In particular, they should acquire some expertise in the utilization of communication and synchronization mechanisms between tasks, in the context of concurrent applications. The multitasking kernel (operating system) to use will be eCos (Embedded Configurable Operating System) [10], working on a PC platform.

2 Problem description

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 is an example of such a situation.

In this part of the project, 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 PIC board ("Curiosity HPC" [1] – 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 PIC board, and the processing of the information collected from the PIC board.

2.1 Monitoring and alarm system

In what concerns the first part of the laboratory project, the following functionality should now be added:

- RS232 serial communication to allow remote access.

- Implementation of operations related to reconfiguration and transference of collected information.
- Asynchronous notification of memory (almost) full (registers not transferred > NREG / 2) using RS232 serial line. The letter "`M`" should also appear in LCD (2nd line, 8th column). It will be kept as long as the duration of that situation.

3 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 PIC 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
rtl		- read temperature and luminosity
rp		- read parameters (PMON, TALA)
mmp	*p*	- modify monitoring period (seconds - 0 deactivate)
mta	*t*	- modify time alarm (seconds)
ra		- read alarms (clock, temperature, luminosity, active/inactive-1/0)
dac	*h m s*	- define alarm clock
dtl	*t l*	- define alarm temperature and luminosity
aa	*a*	- activate/deactivate alarms (1/0)
ir		- information about registers (NREG, nr, iread, iwrite)
trc	*n*	- transfer *n* registers from current iread position
tri	*n i*	- transfer *n* registers from index *i* (0 - oldest)
irl		- information about local registers (NRBUF, nr, iread, iwrite)
lr	*n i*	- list *n* registers (local memory) from index *i* (0 - oldest)
dr		- delete registers (local memory)
cpt		- check period of transference
mpt	*p*	- modify period of transference (minutes - 0 deactivate)
cttl		- check threshold temperature and luminosity for processing
dttl	*t l*	- define threshold temperature and luminosity for processing
pr	[*h*1 *m*1 *s*1 [*h*2 *m*2 *s*2]]	- 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 (`irl`, `lr`, and `dr`), corresponding, respectively, to information, listing and deleting of 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 between the user task 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 PIC 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, already transfered. (NOTE: if desired, the maximum number of registers in one transference can be enforced to 10.)

In the command to list registers from the local memory, `n` is the number of registers to list (if they exist) obtained starting at the position defined by index `i`. Index zero corresponds to the oldest register in the ring-buffer, independently of having been, or not, already read. If the index is omitted, the read index is used (first register not yet read).

In the interaction with the processing task, the user can check and modify the period used to start a new register transference, and also check and define temperature and luminosity thresholds to be used in the processing immediately after register reception. It is also possible to 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).

4 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 read and write indexes (iread, iwrite), associated with local memory, are used for this purpose. The periodic tranference of registers can be activated directly by the user or when a notification of memory (almost) full is received.

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 local memory region, and is available regardless register transference being active or not. The result of this processing is sent back to user interface task that is responsible to present it.

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 those memory regions by the several tasks is of extreme importance for the correct operation of the application.

5 Communication between PC and PIC board

The communication between the PC and the PIC 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 (already configured in eCos):

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
RTL	[T L]	read temperature and luminosity
RPAR	[pars]	read parameters (PMON, TALA)
MMP	p	modify monitoring period
MTA	t	modify time alarm
RALA	[h m s T L a]	read alarms (clock, temp., luminosity, active/inactive)
DAC	h m s	define alarm clock
DATL	T L	define alarm temperature and luminosity
AALA	a	activate/deactivate alarms
IREG	[N nr ri wi]	information about registers (NREG, nr, iread, iwrite)
TRGC	n [regs]	transfer *n* registers from current iread position
TRGI	n i [regs]	transfer *n* registers from index *i*
NMFL	[N nr ri wi]	notification memory (half) full

In the above table (Message Types), both the command codes and any of the several data fields, with the exception of `regs` and `pars`, have a size of 1 byte. Every register has a size of 5 bytes (timestamp: `h m s`; temperature value: `T`; luminosity: `L`).

Meaning of data fields:

h - hours [0 .. 23]

m - minutes [0 .. 59]

s - seconds [0 .. 59]

T - temperature [0 .. 50]

L - luminosity [0 .. 7]

p - monitoring period (in seconds) [0 .. 99] (0 - inactive)

t - time alarm (in seconds) [0 .. 60]

a - alarms active or inactive [1/0]

N - maximum size of buffer (NREG)

nr - number of valid registers in buffer [0 .. NREG]

ri - buffer read index [0 .. NREG-1]

wi - buffer write index [0 .. NREG-1]

n - number of registers to transfer / transferred [0 .. NREG] (can be limited to 10)

i - buffer index [0 .. NREG-1] (0 - oldest)

pars - parameters (2 bytes – PMON, TALA)

regs - registers (size 5 bytes each – h, m, s, T, L)

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, with the exception of the asynchronous memory full notification message, are routed to the task that has made the respective request. 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. In the case of asynchronous notification, a message should be written to the screen, and the periodic transference activated (period = 1 minute).

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

6 Project development

This 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 advised.

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 **should** use a simple command interpreter that is made available by faculty.

The communication interface between Part 1 (PIC board) and Part 2 (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.

7 Project delivery

This part of the project should be delivered by $\boxed{\textbf{08/12/2020}}$. The delivery consists of a digital copy (ZIP file) of all developed programs. **1 or 2** pages with the specification of data structures, main blocks, or most relevant parts of algorithms (in pseudo-code) should also be added. All these elements should be correctly identified (group number and students).

Project demonstrations, based on the delivered material, and **oral exams** will be done on **14/12/2020** and **15/12/2020** (laboratory class) and, possibly, at other additional time slots.

References

[1] Microchip Technology Inc. *Curiosity High Pin Count Development Board User's Guide.* 2016-2018.

[2] Microchip Technology Inc. *PIC16(L)F18855/75 Data Sheet.* 2015-2018.

[3] Microchip Technology Inc. *MPLAB XC8 C Compiler User's Guide for PIC.* 2012-2018.

[4] Microchip Technology Inc. *MPLAB XC8 Getting Started Guide.* 2013.

[5] Microchip Technology Inc. *MPLAB XC8 C Compiler User's Guide.* 2011-2016.

[6] Microchip Technology Inc. *MPLABX IDE User's Guide.* 2011-2015.

[7] Microchip Technology Inc. *MPLAB Code Configurator v3.xx User's Guide.* 2018.

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

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

[10] Free Software Foundation, Inc. *eCos Reference Manual.* 2008.

[11] Free Software Foundation, Inc. *eCos User Guide.* 2009.

[12] Anthony J. Massa. *Embedded software development with eCos.* Prentice Hall. 2003.

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 RTL   0XC2  /* read temperature and luminosity */
#define RPAR  0XC3  /* read parameters */
#define MMP   0XC4  /* modify monitoring period */
#define MTA   0XC5  /* modify time alarm */
#define RALA  0XC6  /* read alarms (clock, temperature, luminosity, active/inactive) */
#define DAC   0XC7  /* define alarm clock */
#define DATL  0XC8  /* define alarm temperature and luminosity */
#define AALA  0XC9  /* activate/deactivate alarms */
#define IREG  0XCA  /* information about registers (NREG, nr, iread, iwrite)*/
#define TRGC  0XCB  /* transfer registers (curr. position)*/
#define TRGI  0XCC  /* transfer registers (index) */
#define NMFL  0XCD  /* notification memory (half) full */

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