RTOS kernel in portable electrocardiograph

C A Centeno, J A Voos, G G Riva, C Zerbini, E A Gonzalez
Grupo de Ingeniería Clínica, Facultad Regional Córdoba, Universidad Tecnológica Nacional, Maestro López esquina Cruz Roja, Ciudad Universitaria, X5016ZAA Córdoba, Argentina.
ccenteno@gmail.com

Abstract. This paper presents the use of a Real Time Operating System (RTOS) on a portable electrocardiograph based on a microcontroller platform. All medical device digital functions are performed by the microcontroller. The electrocardiograph CPU is based on the 18F4550 microcontroller, in which an uCOS-II RTOS can be embedded. The decision associated with the kernel use is based on its benefits, the license for educational use and its intrinsic time control and peripherals management. The feasibility of its use on the electrocardiograph is evaluated based on the minimum memory requirements due to the kernel structure. The kernel's own tools were used for time estimation and evaluation of resources used by each process. After this feasibility analysis, the migration from cyclic code to a structure based on separate processes or tasks able to synchronize events is used; resulting in an electrocardiograph running on one Central Processing Unit (CPU) based on RTOS.

1. Introduction
Due to the increasing integration of peripheral devices within a chip or integrated circuit, it is possible to implement more functionality in electronic designs based on these devices. As a result of this technological advancement, an extensive line of microcontrollers is available, which provides special features to platforms based on them.

For the our previously designed and implemented portable electrocardiograph [1] [2] the microcontroller PIC16F877A from the Microchip family was used.

At the current stage of development, the electrocardiograph CPU uses a microcontroller (uC) PIC18F4550. This PIC model includes more memory, which is a prerequisite for the possible implementation of RTOS kernel. An important feature, necessary for such a change, is the possibility of using the available RAM as stack space.

The availability of these capabilities allows to consider the use of an RTOS to manage all functions associated with control of the electrocardiograph stages, whether analog or digital. The migration from cyclic executive to multitasking mode requires a previous analysis of each process and its interaction with the other ones.

2. Materials and Methods

2.1. Analysis of a Cyclic Executive
The initial system is based on a cyclic executive, in which control processes execute one after another in an infinite loop. In the following, we make a description of each hardware stage and how they interact with the uC. Finally, a summary of this information is presented in Table 1.

2.1.1. Matrix Keyboard.
The keyboard is made up of eight keys in a linear arrangement connected to 8-to-3 BCD encoder lines. The encoder is connected to the uC through three programmable input/output lines. Debouncing is implemented by software and special keys are not defined. The keyboard function is to allow the user to select various modes and functions available to the electronic device, such as lead selection, gain, notch filter, etc.

2.1.2. Input Multiplexer.
In an electrocardiogram it is necessary to obtain different leads implemented by using a multiplexer that allows the combination of patient’s biomedical signals which must then be amplified. This block is controlled by four input / output port lines of the uC.

2.1.3. Conditioning module.
In an electrocardiograph a conditioning module is necessary for amplification of biopotential signals. After this process we obtain an appropriate signal level to be digitized. Because biopotential signals are different from one person to another respecting frequency and spectral composition, this electronic stage must provide comparable signals.

2.1.4. Filtering Stage.
The spectral composition of the conditioned signals allows us to determine certain diseases. If this signal is affected by noise, however, a correct determination is difficult to be reached. Among the spectral components that affect the obtained signal are the common mode signals that are induced on the device and the connections between it and the patient. In particular, the frequency at the common mode signal is 50Hz. For this reason, a Notch filter is introduced which is intended to mitigate the adverse effects of this frequency component. This stage is controlled through a 2-to-4 decoder connected to two input/output port lines of the uC.

2.1.5. Programmable gain amplifier.
Due to the mentioned variability of the signal amplitude after signal conditioning, it is necessary to use a variable-gain amplifier. For that purpose, a programmable gain amplifier can select between x1, x2, x4, x5 and x10 gain steps. For communication with this device, a synchronous serial communication via Serial Peripheral Interface (SPI) is used, which allows sending configuration commands from the uC.

2.1.6. Programmable offset voltage.
One of the design conditions poses the need to work with amplified signals with offset voltage to obtain the maximum excursion of the input voltage applied to the Analog/Digital Converter (ADC). This programmable voltage is obtained from a dedicated chip which is controlled by the SPI interface from the uC.

2.1.7. Analogic to Digital Converter ADC.
The conversion of analog signals to digital ones is made by a 12-bit ADC, which delivers the digital value by using the SPI interface. The maximum conversion rate is up to 100kbps.

2.1.8. Decoder selector
The use of multiple stages in the design of the electrocardiograph creates the need for more input/output control lines than available at our uC. In consequence, a 3-to-8 decoder for extension to five additional lines is included.

2.1.9. Digital stage.
The control of all above detailed stages is performed by a 8-bits PIC18F4550 uC, which includes 2048 bytes of RAM, 32768 bytes of ROM, 3 timers, SPI interface, USB interface and USART, among other parameters.

2.1.10. Liquid crystal display.
The digitized information is presented on a graphic 240X64 pixels LCD. On it, the biopotencial signal and data status of the electrocardiograph are displayed, such as gain, applied filters, reference levels, etc. The LCD uses 10 input/output lines for communication with the uC.

2.1.11. Communication interface.
Due to the future addition of digital post-processing, storage of studies in a database and increasing of the resolution in the graphical presentation, a communication interface must be provided to transmit the acquired information to a personal computer. For this purpose, we take advantage of the available USART interface. Through the use of an optocoupler, galvanic isolation required for this type of device is obtained at a RS232 port. An additional device can be included in order to interface to USB sockets.

| Table 1. Connection between the modules and microcontroller |
|-----------------|-----------------|
| Module          | Connection      |
| Matrix keyboard | 3 lines I/O     |
| Input multiplexer| 4 lines I/O     |
| Filtering       | 1 lines I/O     |
| Programable gain amplifier | SPI by software |
| Programable offset voltage | SPI by software |
| Analogic to digital converter | SPI by software |
| Decoder selector | 3 lines I/O     |
| Liquid cristal display | 10 lines I/O |
| Communication interface | Interfaz USART |

2.2. Module interactions
Each module has an associated specific function, which must be executed in a timely, synchronized manner in order to achieve the desired result. Table 2 presents the cyclic executive that controls each of the modules of the electrocardiograph.

Each of the steps/tasks must be properly synchronized in time. This way, a whole cycle can be completed within the set time. The conversion frequency is 500Hz, so this time is 2mSeg.

Additionally, it should be noted that the full cycle of on-screen display is 10mSec. Table 3 lists the minimum and maximum time needed to complete each phase/task.
Table 2. Cyclic Executive – Pseudocode – Flowchart

| Descripción | Flowchart |
|-------------|-----------|
| 1- Scan Keyboard | ![Flowchart Diagram] |
| a. Change acquisition parameters | |
| i. Filtering | |
| ii. Gain | |
| iii. Offset | |
| iv. Derivation | |
| 2- Clear Screen | |
| 3- Processing – Digital Filtering | |
| 4- Put pixel – Line Draw | |
| 5- Wait for a new cycle (including 5 conversions). End of the IDLE state by using a flag. | |
| 6- Repeat step 1- | |

Table 3. Time required for each process.

| Process                  | Minimum Time | Maximum Time |
|--------------------------|--------------|--------------|
| Leer_teclado             | 0.2 mSeg     | 0.2 mSeg     |
| Borrar_pantalla          | 8.6 mSeg     | 10.47 mSeg   |
| Dibujar línea            | 0.55 mSeg    | 3.55 mSeg    |
| Conversión_ADC           | 0.320 mSeg   | 0.320 mSeg   |
| Envío de datos vía USART | 0.100 mSeg   | 0.100 mSeg   |
| Total cycle time         | 9.77 mSeg    | 14.64 mSeg   |

The study of the runtime shows that the process Borrar_pantalla requires more than 10 mSec, so it is necessary to divide it in two stages, that is run completely every two cycles of screen printing. Additionally, due to the available screen resolution, it is only possible to print a dot for every five acquired samples by the ADC.

Other processes, and their additional times, must be completed within the established 10mSec as well.

If sending the converted information to a PC is considered, a transmission rate must be accordingly defined. On the other hand, it should be mentioned that 12 bits AD conversion takes at least 2bytes for each acquired value. Using asynchronous transfer without flow control, we introduce a marker byte in each packet transmission to ensure proper reception on the PC. In short, for every transmitted data 3 bytes must be sent, so the baud rate must be greater than or equal to 19200bps.

The necessary synchronization between each process in order to avoid data loss is shown in Figure 1. It must be kept in mind that start control of each repetition is performed using one of the available timers.
2.3. Using a Real Time Operating System (RTOS)
The uCOS-II is a Real Time Operating System [4] (RTOS kernel) that allows preemptive multitasking processing (i.e., the execution of a process is interrupted when a higher priority process is ready to run). It also offers elements called "events", such as mailbox, semaphores, queues, and others for synchronization and/or data transfer between processes. In addition, it has functions for time control and it may suspend or restart processes, as well as remove any of them.

We will highlight as part of the following analysis the hardware requirements for the RTOS port to PIC18 to be successfully embedded in the development platform [5] [6].

The RTOS, through the priority assigned to each process, determines which process runs first. This priority is set between zero and a maximum configurable value. That capability would allow the process "conversión_ADC" to have the highest priority to be executed as frequently as required, during time remaining from the other defined processes.

One of the advantages of using an RTOS is to exploit time previously wasted to consume clock cycles (e.g., to introduce delays), for the execution of a process / task.

2.4. System requirements
An RTOS typically requires RAM and ROM memories to implement its functionality, so it is very important to know the minimum required resources.

In the following, we analyze software modules that compose the RTOS and our specific hardware.

For each task that integrates the system, the RTOS requires a Task Control Block (TCB); and each element needs an Event Control Block (ECB) for synchronization. The total amount of TCBs and ECBs depends on how the final system is structured.

One of the features of the adopted RTOS is the ability to modify its configuration, and therefore its intrinsic capabilities, namely to enable and / or disable functions to obtain the smallest size of the resulting executable. A proper configuration allows us to save memory, both RAM and ROM. On the other hand, it is also possible to define how to manage the available stack for different tasks. This can be either Minimum or Extended mode. In the last one, the user is able to know the stack required by the tested process.
In Table 4, we present details of the amount of RAM and ROM associated with the different possible configurations of our system.

| Process                  | Mode                      | ROM   | RAM   | bytes |
|--------------------------|---------------------------|-------|-------|-------|
| uCOS II – PIC18          | Minimum mode              | 4712  | Bytes |
|                          | Extended mode             | 5363  | Bytes |
| TCB                      | Minimum mode              | 14    | Bytes |
|                          | Extended mode             | 26    | Bytes |
| ECB                      | Both modes                | 8     | Bytes |
| Statistics Task          | Minimum mode              | 322   | 124   | Bytes |
|                          | Extended mode             | 973   | 220   | Bytes |
| Use of SEMAPHORES        | Both modes                | 1870  | 24    | Bytes |
| Use of MAILBOX           | Both modes                | 2108  | 30    | Bytes |
| MAILBOX + SEMAPHORES     | Both modes                | 3139  | 30    | Bytes |

Each task needs, in order to interact with the other ones, a priority and an associated stack. The definition of the amount of stack is very important, and depends entirely on factors such as the CPU overhead due to the kernel (i.e., how many times per second the scheduler is called), the number of tasks that make up the entire system, and their duration times, among others.

Each time the scheduler puts a task in Waiting state, it saves its context into its associated stack, so that when that task has to return the Run state, it does it from the point where it left. An Incorrect stack allocation of each process can cause the system to become unstable. Figure 2 shows the possible states that each process can adopt [5].

Priority and stack size are saved in the associated TCB in the Minimum mode; while in the Extended mode, the points where the stack starts and ends are also saved. Thus it is possible, by using functions defined for this purpose, to know at any instant the percentage of stack used. This feature defines the optimal stack value for each task, with the premise of full use of RAM resources. The TCB of each task also changes its configuration depending on the mode used, either Minimal or Extended.

Figure 2. States defined in the system
It is also necessary to know when the CPU is overloaded. Overloading can cause the CPU not being able to complete a process before it must be executed again. This may be because more tasks are required than can be processed, or because their times are shorter than the minimum CPU time. In order to minimize the overhead, we need to know what the percentage use of the CPU is; if this value reaches 100%, the system may become unstable. In order to know this value, the system has a statistical task, which also consumes valuable resources. This task should only be compiled at the time of evaluation, and not in the final product.

After weighting each of the capabilities of the system, we conclude that 7600 bytes of ROM and 925 bytes of RAM are necessary for use of the RTOS uCOS-II on a PIC18. This amount of resources is about 50% of available RAM and ROM on the selected microcontroller.

3. Results

3.1. Definition of Processes/Tasks

After analyzing each of the modules of the electrocardiograph, tasks (also called processes) that should compose the general system were determined. From this, four tasks were defined, which are described below.

3.1.1. Tarea_Conversion process

This process makes use of an ADC to convert the analog signal into a digital value, and an SPI interface to link the uC with the ADC. It should be noted that the SPI is implemented by software, due to pinout constraints. This task runs with a period of 2 mSeg. The conversion result is sent to Tarea_LCD and Tarea_COM tasks by means of events.

3.1.2. Tarea_Leer_Teclado process

This process runs with a period of 200 mSeg, reading the value from the 10-to-1 encoder of the key pressed. This process must be run after the ADC completes a conversion, because a result of keystrokes, changes settings of the programmable gain amplifier (PGA) and programmable voltage reference. Both modules make use of the SPI interface.

3.1.3. Tarea_LCD process

This process is executed after the Tarea_Conversion process, and takes as input the value sent by that process through a mailbox event.

3.1.4. Tarea_COM process

This process is executed after the Tarea_LCD process, with the value received through a mailbox from the Tarea_Conversion process. This value is transmitted to a PC through the available USART hardware interface.

3.2. Process interaction

In Figure 3 the interactions of each of the described processes are presented.
We can summarize that at least 40 bytes are needed by the scheduler for the control for this configuration using 5 processes. At the same time, the use of events, i.e., mailbox, flags, etc, requires reserving RAM memory for each of them.

The system makes use of events for synchronization of processes or tasks for optimal use of available physical hardware resources. In Table 4, the Conversión_Task and Task_LCD processes are presented as an example.

The most critical module for work in multitasking mode is the SPI serial communication interface. Due to the impossibility of using specific included SPI hardware, which was used for other purposes, this interface has to be managed by software. The sharing of the same channel by two different and independent processes, to synchronize these tasks required the use of semaphores. The Teclado_Task process and the Task_Conversion process can access to the same resource, so was used two independent events for the synchronization between them.
Table 4. Process Task_LCD and process Conversión_TASK

| void Task_LCD(void *pdata) |
|---------------------------|
| void Task_Conversion(void *pdata) |
| #if OS_CRITICAL_METHOD == 3 |
| OS_CPU_SR cpu_sr; |
| #endif |
| char *rxmsg; |
| INT8U err; |
| INT16U valor_LCD; |
| x=0;inicio = 0;fin = 25; |
| for(;;) |
| { |
| rxmsg = (char *)OSMboxPend(COM_Mbox, 0, &err); |
| // Espera mensaje de Conversion_TASK |
| if(err == OS_ERR_PEND_ISR) |
| valor_LCD = 0; |
| else if(err == OS_ERR_PEVENT_NULL) |
| valor_LCD = 0; |
| else if(err == OS_ERR_EVENT_TYPE) |
| valor_LCD = 0; |
| else if(err == OS_ERR_PEVENT_NULL) |
| valor_LCD = 0; |
| else |
| valor_LCD = *rxmsg; |
| cuenta++; |
| if(cuenta>=7) |
| { |
| bloquex++; |
| cuenta=0; |
| if(bloquex>=29) |
| { |
| bloquex = 0; |
| x=0; |
| } |
| } |
| if(Bits1.bit1_velocidad_25_50mm){ |
| if(cant_datos>=10) |
| Bits1.bit1_interrupcion=1; |
| // salgo del loop infinito e imprimi punto |
| cant_datos=0; |
| } |
| cant_datos++; |
| dato_ADC = Leer_Adc(); |
| if(dato_ADC >= 4096) |
| dato_ADC = 4096; |
| if(dato_ADC >= maximo_ADC) |
| maximo_ADC = dato_ADC; |
| if(Bits1.bit1_interrupcion) |
| Bits1.bit1_interrupcion = 0; |
| if( OSSemAccept( ADC_Sem ) ) |
| { |
| OSSemPost( TECLADO_Sem ); |
| } |
| OSMboxPost(LCD_Mbox, (void *)&maximo_ADC); |
| OSMboxPost(COM_Mbox, (void *)&maximo_ADC); |
| maximo_ADC = 0; |
| } |

A summary of the resources after the migration complete cyclic executive to RTOS platform is presented in Table 5. It shows a comparison of the resources cost both in the executive cyclic version and the multitasking version.

Table 5. Summary of resources

| Resource          | Cyclic executive | %   | RTOS             | %   |
|-------------------|------------------|-----|------------------|-----|
| RAM               | 614 bytes        | 29.98% | 1101 bytes     | 53.76% | |
| ROM               | 4268 bytes       | 26.05% | 11544 bytes     | 70.46%  |

| System composition | * Process interrupt-driven conversion |
|--------------------|----------------------------------------|
|                    | * Polling of other processes |

* 5 TASK
* 4 EVENT
* Mail Box Enabled
* Minimum MODE
4. Conclusions

The use of an RTOS for CPU tasks on an electrocardiograph requires changes in the definition of operations to be performed by modules in the system. On the other hand, the inclusion of multitasking allows for much better exploitation of CPU capacity to process tasks in the system. Factors to be considered while deciding about using an RTOS on the developed platform are: requirement of minimum amount of RAM and ROM, eventual migration to another microcontroller, and possibility of implementation on a license-free RTOS platform or adding license costs to the product.

For stability analysis of the software version based on RTOS versus the cyclic executive version, two electronic equipments would be necessary. In this work, the analysis of a possible migration was performed on an existing platform; therefore it was necessary to adapt existing software modules in order to minimize the use of RAM and ROM. As part of these operations, we suppress variables which were previously used and are not further necessary in the new platform in order to reduce memory consumption.

Even if the electrocardiograph is able to run all the previously mentioned tasks with an RTOS, in the present it is not possible to add more functionalities. This is due to the lack of remaining ROM after the implementation of the RTOS system on the PIC18F4550. However, it may be considered using the PIC18F4675 uC in order to increase functionality without modifying the physical hardware.

The final application interacts properly with the associated hardware, and allows for further enhancements to the system. We therefore conclude that it is suitable and convenient to use an RTOS on the developed electrocardiograph.

References

[1] Monitor de pulsos cardíacos con interfaz a PC. SABI 2003. Córdoba. Argentina. 2003
[2] Procesamiento digital sobre monitor cardiaco. CLAIB 2007. IV Congreso Latinoamericano de Ingeniería Biomédica. Caracas. Venezuela. 2007
[3] Sistema para el análisis de la morfología de la señal de ECG de ratones de laboratorio infectados con mal de chagas. CBEB 2010. Congreso Brasileiro de Ingeniería Biomédica.
[4] Medical Data Transmission System for Remore Healthcare Centres, Gonzalez E, Cagnolo F, Olmos C, Centeno A, Riva G and Zerbini, J. Phys.: Conf. Ser., Vol. 90, No. 1. (2007), 012029. doi:10.1088/1742-6596/90/1/012029 Key: citeulike:2069632
[5] uCOSII, The Real-Time Kernel , Jean J. Labrosse, R&D Technical Books, 1998,ISBN 0-87930-543-6.
[6] MPLAB-uCOS-II-PIC18 Version 2.51 www.micrium.com