Laboratory Project
Real-Time Systems
IST - 2025/2026

(Part 2 – FreeRTOS) Multi-Function Device

1 Introduction

Many real-time 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 real-time 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 FreeRTOS [1, 2, 3], working on a "Mbed platform" (ARM-CortexM3 based) [4, 5, 6].

2 Problem description

The second part of the project corresponds to a "Multi-Function Device" using a new environment with more resources, making it possible to have a concurrent application with several tasks supported by a multitasking kernel. Those tasks will interact among them, and will provide a more flexible user interface.

This part of the project is implemented using the development board **"mbed LPC1768"** [5], which includes a microcontroller **NXP LPC1768** (ARM-CortexM3 based), and can be connected to a "mbed Application Board" [6], which includes several devices. The application is programmed using the **C/C++ programming language**, and the mbed online development environment (Mbed Online Compiler / Keil Studio Cloud) [4]. Related documentation is available on the online site.

The main functions to be provided by this part of the project, corresponding to several different tasks, are:

- Temperature Sensor monitoring service (working in a periodic way and on demand, and with the capability of storing collected data and handling alarm situations).
- A **Bubble Level** based on an accelerometer.
- Management of RTC (real time clock) to keep date and time (allowing event timestamping and alarm clock management).
- Generic processing and alarm management.

- Device actuation (to be handled by specific resource manager tasks or in a distributed way).
- User interface (console).
- **Optional:** Hit Bit game.

A temperature sensor task will monitor, **both** in a periodic fashion (period `pmon`) and on demand, the value of temperature (sensor LM75B, available on the application board). If `pmon` is zero there will be no periodic information collected, but should still provide that service on demand.

The collected value, in addition to be represented both on the LCD and using the RGB LED (blue: low temperature; red: high temperature), is analyzed in order to verify if it corresponds to a maximum or a minimum, or to an alarm situation (if alarms are enabled). Maximum and minimum values are kept as records that, in addition to the value obtained from the sensor, also contains a timestamp with the current date/time (`time_t` – obtained from the RTC). If the collected value was obtained in response to a user request, in that case, it should also be sent back to the user task (console).

Temperature alarms occur if they are enabled and the collected temperature value is below or above specified thresholds (`tlow, thigh`).

The Bubble Level is implemented by a task that uses an accelerometer (MMA7660, available on the application board). It should use the information related to axes X and Y, and the user interface is done using the LCD.

In order to keep the current date/time, the RTC (real time clock) available on the mbed board should be used. The user interface (console) will provide commands to set and read the current date and time. Alarm clock functionality should also be provided using the RTC.

The notification of an alarm situation (both related to temperature and alarm clock) is done through the generation of a sound (using a buzzer available on the application board). The sound will have a duration of `tala`. A specific task should be in charge of this service. It should also be possible to change the configuration of the sound (period and duty cycle of PWM signal), using the potentiometers available on the application board (pot1 and pot2, respectively), if that functionality is enabled.

For user interaction, there will be a console, providing the ability to execute a set of commands, described bellow. A specific task should be in charge of this, with the help of a command interpreter that is made available by faculty.

Depending on the available time, it is also possible to add another task to implement a “Hit Bit” game, using the 4 mbed LEDs and the joystick (central button).

It should be possible for the user to activate/deactivate both the “Bubble Level” and the “Hit Bit” game, if desired.

3 User interface

The main user interface, in this part of the project, is performed using both an LCD, and through the use of a console. But there are some situations where the mbed LEDs, the RGB LED, the joystick, and the buzzer are also used.

3.1 LCD

The information presented on the LCD should be something similar to the following:

hh`:`*mm*`:`*ss*	
`A: C T`	`O`
`T(C)=` *tt.t*	

The current value of clock (“`hh:mm:ss`”) and temperature (“`tt.t`”) are presented. The right part of the LCD is used for the Bubble Level. In the middle line there will be the information related to alarm activity: letters **C** and **T**, will appear if the alarm clock or the temperature alarm are enabled, respectively.

3.2 Console

There is a task responsible for user interface using a console, that makes it possible to execute a set of commands to interact with the system. The commands that should be made available are the following:

Available Commands		
cmd	args	description
rdt		- read date/time (dd/MM/YYYY hh:mm:ss)
sd	*d M Y*	- set date (day, month, year)
rc		- read clock
sc	*h m s*	- set clock (hours, minutes, seconds)
rt		- read temperature
rmm		- read maximum and minimum of temperature
cmm		- clear maximum and minimum of temperature
rp		- read parameters (pmon, tala)
mmp	*p*	- modify monitoring period (seconds - 0 deactivate)
mta	*t*	- modify time alarm (seconds)
rai		- read alarm info (alarm clock, tlow, thigh, active/inactive)
sac	*h m s*	- set alarm clock (hours, minutes, seconds)
sat	*tl th*	- set alarm temperature thresholds (tlow, thigh)
adac	1/0	- activate/deactivate alarm clock
adat	1/0	- activate/deactivate alarm temperature
rts		- read task state (Bubble Level, Hit Bit, Config Sound)
adbl	1/0	- activate/deactivate Bubble Level task
adhb	1/0	- activate/deactivate Hit Bit task
adcs	1/0	- activate/deactivate Config Sound operation

In this list of commands, some may be executed directly by the console task, and others through the interaction with specific tasks.

Meaning, and assumed range, of data fields:

d - day [1 .. 31]

M - month [1 .. 12]

Y - year [1970 .. 2037]

h - hours [0 .. 23]

m - minutes [0 .. 59]

s - seconds [0 .. 59]

tl - temperature threshold low [0 .. 50]

th - temperature threshold high [0 .. 50]

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

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

Consider the following initial values for some of the referred parameters and task states:

pmon	5 sec	monitoring period
tala	3 sec	duration of alarm signal (PWM)
tlow	10 °C	temperature threshold low
thigh	25 °C	temperature threshold high
ALAT	0	alarm temperature initially disabled
ALAC	0	alarm clock initially disabled
ALAH	12	alarm clock hours
ALAM	0	alarm clock minutes
ALAS	0	alarm clock seconds
BL	1	Bubble Level active
HB	0	Hit Bit inactive
CS	0	Config Sound inactive

4 Considerations about some activities

In the representation of temperature using the RGB LED (blue: low temperature; red: high temperature), as the temperature range is small in a normal situation, the alarm thresholds (`tlow, thigh`) can be used as "saturation points": any temperature below `tlow` is represented as only blue; any temperature above `thigh` is represented as only red. Color gradation, using the 3 colors, is done only for the interval `[tlow, thigh]`.

In the "Bubble Level" implementation, graphical facilities of the LCD library can be exploited (e.g. rectangle, circle, filled circle).

In the "Hit Bit" implementation, the 4 mbed LEDs should be used to "rotate the bits". The central button of joystick is used "to hit the bit" when it is on the right position (LED4). If the LED was ON when the button is pressed, that bit is cleared, otherwise a new bit is added to the sequence of bits. When all bits are cleared there is a WIN situation. That situation is signaled by blinking the 4 LEDs 3 times. You may consider the use of interrupts associated with the button to notify the task implementing the game.

As said above, device actuation and other shared resource management might in some cases be done by specific tasks or using distributed control. In any case, consistency must be ensured for correct operation.

5 Project development

This second part of the project is also programmed using the C programming language (and possibly some C++). The development environment is the "Mbed Online Compiler". However, the operating system that will support the application is FreeRTOS, and the final application, containing FreeRTOS, and possibly some mbed libraries, is transferred to the target board (mbed) 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 (console related), 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 characteristics of the IO devices that are used should be consulted on the manual of the board, and/or on the "Data Sheets" of the devices (this information can be obtained from the online site).

6 Project delivery

This part of the project should be delivered by $\boxed{\textbf{11/01/2026}}$. The delivery consists of a digital copy (ZIP file) containing all developed programs and a small report (3 or 4 pages) explaining the main requirements and solutions related to real-time aspects and communication/synchronization aspects. In addition to source files, the ZIP file should include the final executable file (file "frtos.bin"). 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 the week starting on **12/01/2026**.

References

[1] *FreeRTOS: Real-time operating system for microcontrollers.* 2022. URL: http://www.freertos.org.

[2] Amazon Web Services. *The FreeRTOS Reference Manual.* 2017.

[3] Richard Barry. *Mastering the FreeRTOS Real Time Kernel – A Hands-On Tutorial Guide.* Real Time Engineers Ltd. 2016.

[4] Mbed. *Mbed Online Compiler / Keil Studio Cloud.* 2022. URL: https://mbed.org.

[5] Mbed. *mbed LPC1768.* 2022. URL: https://os.mbed.com/platforms/mbed-LPC1768.

[6] Mbed. *mbed Application Board. Application board for mbed NXP LPC1768.* 2022. URL: https://os.mbed.com/components/mbed-Application-Board.