Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
*Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird*
*Annals & Proceedings of DAAAM International 2011*

# GRAPHICAL USER INTERFACE SIMULATORS FOR LESSONS OF REAL-TIME PROGRAMMING

**DOLINAY, J[an]; DOSTALEK, P[etr] & VASEK, V[ladimir]**

***Abstract:** This paper describes graphical user interface simulators we developed and use in lessons of programming real-time applications with real-time operating systems. The simulators represent attractive way of displaying status of the program and improve student's motivation in the lessons. The paper describes both the implementation of the GUI simulators in C# programming language and their usage in student's programs written in C language.*

***Key words:** teaching, real-time programming, GUI, simulator*

## 1. INTRODUCTION

Embedded computers are found in large number of devices around us and their number increases rapidly. In many applications implemented with discrete parts few years ago are nowadays used microcontrollers (MCU). This is due to their very low price and advantages in terms of reliability, flexibility of the functions of such device (which can be changed by changing the program in the MCU without change in the hardware) and also the lower price of the device due to smaller number of parts. (Morton, 2001). Real-time operating systems (RTOS) are necessary part of many embedded computers, especially those which perform more complex tasks**.** Therefore it is not surprising that a university graduate specialized on informatics should have understanding of embedded computers hardware and software design, as well as real-time operating systems functionality and programming.

Schools attempt to give their students this knowledge in various ways, ranging from "classical" lessons focused on computer or microcontroller parts and their handling in program, to more interesting lessons, such as programming some real world applications or models, robots, etc. It seems that the lessons have better impact on students if they show how certain goals can be achieved by program using practical tasks, rather than describing how certain MCU peripheral is programmed (Hamrita & McClendon, 1997; Klassner, 2002).

But not always it is necessary or even desirable to use a real-world system in the student's program. For example, in a simple hello-world type of program, including communication with hardware can make the program more complicated (unnecessarily) and thus make it harder for the students to understand it. This way the good intention of making the lesson more attractive by including some real-world object can lead to unexpected and undesired outcome of worse performance of the students. On the other hand, the ability to see outputs of their programs is very important for learners. If for no other reasons, it can help diagnose problems in the program by printing out debug messages and so on. For typical lessons, where the students use standard personals computers (PC) to develop and run their programs, the output is typically represented by console window into which the program can write text messages (using e.g. printf function in C). This is sufficient as far as the information value is concerned, but not very interesting for the students. Moreover, it does not resemble the real problems students will be facing in their professional career, since we can nowadays hardly expect they will be writing programs which have console interface.

One solution to the above problems is offered by use of simulators, which simulate the user interface of a real object, e.g. a clock, temperature controller, etc. It has the advantage of real interactivity, almost identical to using real device, while being much cheaper and less prone to damage by improper use. Such kind of tools is commonly used, for example, in developing application for mobile phones and we decided to adopt it for our purpose as well. In the following text we will describe the simulator interface and their usage in the lessons.

## 2. SIMULATOR DESIGN

At our faculty we teach programming microcontrollers both at the low level, creating programs from scratch and also using real-time operating systems. We try to make the lessons more attractive for students by using models of real-world systems and various expansion modules. (Dolinay et al., 2007). This article is focused on lessons of programming with RTOS, which can be taught mostly on PC. The program running on PC needs to communicate with the user in some way, e.g. to allow him/her to adjust the parameters. As mentioned above, console interface is not the best option for this and we wanted to improve the ways student's programs can communicate with the user. After evaluating the options we decided to use software simulators, which will substitute real user interface, such as buttons or displays.

The simulator is in fact a program with graphical user interface (GUI), which communicates with the program written by the student and displays the information student's program want to output or sends information to the student's program about user inputs, e.g. button press.

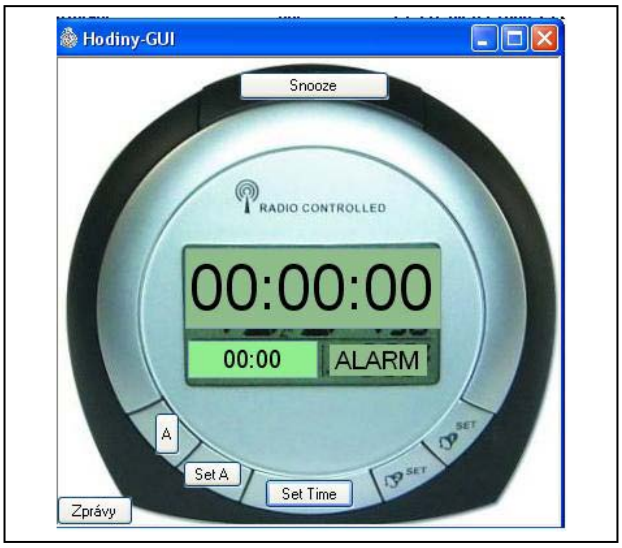

Fig. 1. Alarm clock simulator GUI

For explanation of the simulator implementation it will be best to start with practical examples. In our lessons we currently have 3 simulators:

- Stop watch
- Alarm clock
- Temperature controller.

The alarm clock simulator can be seen in fig. 1.

Students write their programs in C language and use school operating system RTMON (Dolinay et. al. 2010), and also Windows API. One of the first tasks they solve in the lessons is creating a simple stop-watch program with two threads (processes). One thread increments the time and the other thread handles user commands, such us stop or reset the time. Such program can interact with the outside world in these ways:

(1) Through console window – printing time and responding to keyboard commands.
(2) Through GUI implemented directly in the program.
(3) Through simulator program which provides the GUI.
(4) Through a real user interface represented by real display and buttons, somehow connected to the PC.

As described earlier, option (1) has several disadvantages and is also less attractive to students. Option (2) requires that the RTOS used has a GUI interface support and that the user knows how to incorporate this GUI in his/her program – which is typically not an easy task. Option (3) is the one we deal with in this article and will be explained in details later. Option (4) is probably the best as it is both attractive to students (there is a real device to play with) and also it represents the typical real-world configuration of an embedded system. The reason why we do not use this approach for all tasks in lessons is its complexity – it is hard to prepare such tasks and it is hard to program them as well. However, we use this approach at least partially in some tasks, where students control real model of a heating plant.

As mentioned, our approach is the option (3) from the list above; that is a simulator program which provides GUI to the student's programs. The simulator is intended for Windows platform, so it is assumed that the development PC (host PC) is running Windows OS. But it would be possible to port the simulators to run also on Linux or other systems. The simulator is independent program (process) which communicates with the student's program (also independent process) via messages. However, the messaging between the simulator and the user program is hidden from the students. In their programs they use C-language library functions provided with the simulator, as will be described in the next chapter.

## 3. IMPLEMENTATION

As mentioned earlier, the simulator is independent Windows process which communicates with the student's program using Windows messages. This mechanism is hidden inside the simulator library, which the students use in their programs. This library represents the features provided by the simulator packed into easy-to-use functions such as DisplayTime or GetKeyPress. This has additional advantage of similarity to the real application with physical hardware instead of simulator. The programmer can expect to have similar functions offered by the driver software which he obtains together with the hardware. So from the developer's point of view the program is written in the same way for simulator as it would be for the real hardware. There are no extra things to learn which would be needed only for simulation but had no practical use in real life.

The simulator interface can be divided into 2 main parts:

- The GUI part (simulator itself, running as an independent process on the host PC).
- The "client" part, which is included in the student's program.

The simulator itself is a program written in C#. This program is able to process and send Windows messages from/to virtually any other windows program.

The client part of the interface is library written in C, which the user adds to his/her program and calls the library functions. For maximum flexibility this part has 2-layer design.

The low-level layer (implemented in gui.h and gui.lib) are common functions which allow sending and receiving Windows messages, but do not interpret the meaning or data in these messages.

The high-level layer is then library specialized for given simulator program, e.g. for the alarm clock. This layer utilizes the functions from the lower layer and provides functions to the user's programs. These functions are simulator-dependent, so for the alarm clock there may be functions for displaying time, starting or stopping the buzzer, etc.

## 4. CONCLUSION

In this article we described our approach to teaching RTOS programming using simulators which provide graphical user interface for student's programs. These simulators were developed based on our experience with teaching such courses. They represent compromise between simple console interface and full hardware interface with real display, buttons, etc. Using such simulator help motivate students for writing programs and at the same time it brings the programs written in lessons closer to real applications, which students may face in their future career. The simulator is implemented as a program written in C# language and accompanied by a library for C language which students include in their programs. As of now we developed and use three such simulators: stop-watch, alarm clock and temperature controller. In future more simulators could be developed and also the interface could be ported to other operating systems besides Windows.

## 5. ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the Research Plan No. MSM 7088352102 and by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

## 6. REFERENCES

Dolinay, J.; Dostalek, P. & Vasek, V. (2007). Educational models for lessons of microcontroller programming, *Proceedings of 11th International research/expert conference TMT 2007*, Tunisia, ISBN 978-9958-617-34-8, pp. 1447-1450, Hammamet

Dolinay, J.; Dostalek, P. & Vasek, V. (2010). Simple operating system RTMON for HC08 microcontrollers, *Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium*, ISBN 978-3-901509-73-5, ISSN 1726-9679, pp 0258, Katalinic, B. (Ed.), pp. 0515-0516, DAAAM International, Vienna

Hamrita, T. K., McClendon, R. W., A New Approach for Teaching Microcontroller Courses, *International Journal of Engineering Education*, Vol.13, No.4, 1997, pp. 269-274

Klassner, F. (2002). A Case Study of LEGO Mindstorms' Suitability for Articifial Intelligence and Robotics Courses at the College Level, *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, Kentucky, ISBN 1-58113-473-8, pp 8-12, Cincinnati

Morton, T. D. (2001). *Embedded Microcontrollers*, Prentice Hall, ISBN 0-13-907577, Upper Saddle River