Didactic Embedded Platform Atmega128 for Developing Real Time Operating System

Adam Kaliszan
Poznan University of Technology
Faculty of Electronics and Telecommunications
Chair of Communication and Computer Networks
Email: adam.kaliszan@gmail.com
http://www.adam.kaliszan.yum.pl

Mariusz Głąbowski
Poznan University of Technology
Faculty of Electronics and Telecommunications
Chair of Communication and Computer Networks
Email: mariusz.glabowski@et.put.poznan.pl

***Abstract*—This paper presents a new didactic platform that is capable to run a real time operating system. The proposed platform consists of hardware, firmware and software tools. The project of the hardware part is distributed according to GPLv2 license. The software of the platform is based on FreeRtos ported by the authors on the platforms not originally supported, i.e. microcontrollers Atmega128 and Atmega168. All the software tools work on Linux operating system and are free of charge; most of them have open source code. The proposed platform is not expensive and most of the students can afford to build or modify it on their own. The main aim of the proposed platform is to familiarize the students with the basics of operating systems.**

I. Introduction

An important addendum of operating systems theory course is the practice. The practical part of the course is often conducted with the help of one of the existing operating systems – usually Linux operating system. The choice of the system as such is the right one because of its advantages and versatility ranging from small embedded devices to powerful supercomputers. However, the practical part is often limited to learning basis of Linux, i.e. learning fundamental commands like creating and removing files or directories, changing file attributes and launching programs. Such realization of practical part has no connection to the operating system theory since the laboratories do not cover topics like multitasking, interprocess communication and its synchronization or operations on filesystem. This is usually caused by the complicated Linux programming interface. The complication results mostly from a required compatibility with various standards, e.g. posix and sysV [2]. In order to provide the compatibility with each of these standards separate interfaces have been introduced. Consequently, it takes a lot of time to get familiar with the whole programming interface and, finally, the students getting prepared to their laboratories are generally focused on studying the documentation instead of understanding the sense of presented mechanisms of the operating systems. The mentioned difficulties encouraged the authors to elaborate a new didactic platform presented in Figure 1 including hardware, firmware and software tools. In the presented system a handling of mechanisms, such as files, multi-tasking, interprocess communication or process synchronization, was simplified. The software code, worked out for the platform's needs, is open and it allows students to get familiar with particular mechanisms of the operating systems.

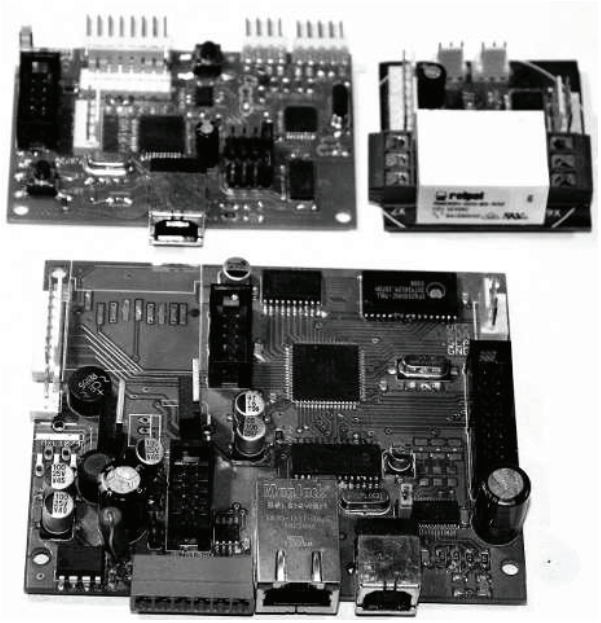

Fig. 1. Didactic Embedded Platform Atmega128

The remaining part of the paper is organized as follows. Section 2 present the main parts of the proposed platform. In section 3 exemplary exercises conducted with the help of the proposed platform are proposed. Section 4 concludes the paper.

II. Platform description

The presented platform consists of hardware, firmware and software tools. The main aim of our work was to create a platform the software and hardware part of which can be modified without license fees. Additionally, our aim was also to design a not expensive platform which can be built at home by students interested in embedded systems.

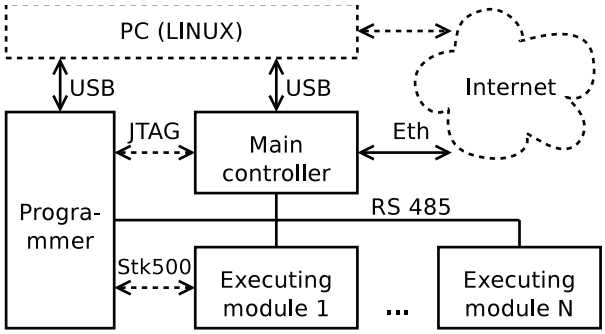

Fig. 2. Ideological schematic of platform

A. Hardware

The hardware part was designed with the help of freeware version of Eagle [3] CAD software. The dimension of PCB board was limited to 10 by 8 centimeters (freeware version of Eagle CAD software does not allow us to design larger PCB boards). The complete project of the hardware is available at http://akme.yum.pl/eagle. The limited dimension of the board allows students to modify the project using freeware version of Eagle CAD. The hardware was designed in user friendly manner: it uses a common interface and does not need any external power supply. The platform is connected to PC by USB since RS232 is not very common in modern personal computers.

In Figure 2 an ideological schematic of the platform is depicted. The objects drawn by continues lines belong to the platform's hardware. The main controller is connected with executing modules by RS485 bus. The programmer is also connected to the bus in order to facilitate debugging.

1) Main controller: The main controller is a system supervisor controlling the executing modules. The ideological schematic of the main controller is depicted in Figure 3. The main controller consists of: microcontroller Atmega128, 64 kB of data external memory, USB interface (Ft232Rl), RS485 interface (Max481), Ethernet interface (Enc28j60) and Secure Digital card reader. Ethernet controller, SD card reader and microcontroller share the same SPI bus. It is possible to connect 8 more devices to this bus using SPI connector placed on the main controller (the dotted box). The controller has also the optional pulse step down converter which can be useful if we want to use external power supply. In order to communicate with external devices, sensors and modules the controller use the following buses: SPI, I2C and RS485. All buses have their own connectors (SPI con, I2C Con, RS485). In order to reduce costs, user communicates with the controller using console (VTY100 protocol) connected to USB port. The main controller has no display nor keyboard. The CPU is programmed using JTAG interface (JTAG Con) that allows to debug the software. Additionally, there is connector (AC Con) with analog inputs and connector (Int Con) with inputs generating interruptions.

2) Executing module: The executing module is responsible for switching on/off various devices, for example lights or roller blinds in an intelligent home. The executing module consists of: microcontroller Atmega168, RS485 interface, 4 relays, 2 outputs for LED, two connectors with two inputs each, 5 jumpers for setting device address. The relays have independent power supply in order to avoid brownouts. The executing module can be programmed using SPI bus (STK 500 programmer) or RS485 bus (bootloader with xModem protocol).

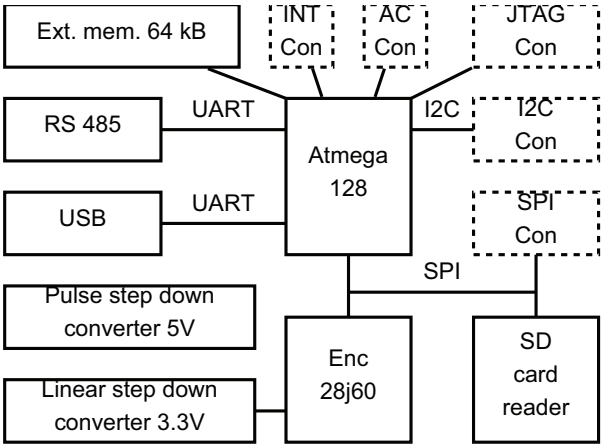

Fig. 3. Ideological schematic of main controller

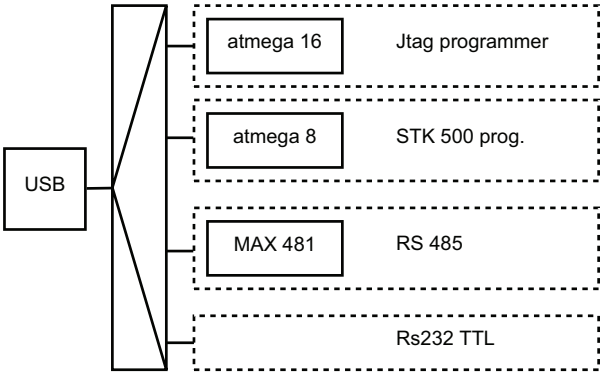

Fig. 4. Ideological schematic of programmer

3) Programmer: The firmware is flashed to main controller or executing module using programmer with USB interface. Both devices have different programming interfaces (JTAG and SPI). The constructed programmer is an universal device with JTAG, STK 500, RS485 and RS232 TTL interfaces. Its ideological schematic is presented in Figure 4. Two jumpers placed on device, depicted in Figure 4 as multiplexer, allow us to choose an appropriate function of the programmer. The programmer has two microcontrollers: Atmega8 and Atmega16 that are responsible for programming with STK 500 and JTAG protocol, respectively. In order to reduce costs, both microcontrollers share the same USB interface chip.

B. Firmware

Atmel – the producer of AVR microcontrollers – provides good Integrated Development Environment, called AvrStudio,

that works in Windows systems. The Windows operating system is not free and using it is against the philosophy of the proposed platform. For this reason, it was necessary to complete and configure all required and cost free tools on Linux. In Ubuntu distribution all required programs are available in its repositories and can be installed using apt-get install command. Thanks to this advantage it is very easy to write the students' instruction explaining how to prepare the system to work. Additionally, Linux enables symbolic linking – we are able to configure Linux in such a way that for example the main controller is always visible as /dev/atmega128eval. The prepared firmware code is written in C and includes Doxygen in code documentation. Consequently, in Kdevelop 4 all comments are visible, what helps to understand and develop the software. The prepared firmware takes the advantage of results of FreeRtos [4] project and AVRlib [5] project[1]. The main elements of the firmware part are: kernel, Command Line Interpreter, file system, communication protocol, bootloader and the TCP/IP protocol stack.

1) Kernel: The main controller and the executing module are controlled by embedded real time operating system FreeRtos. Its kernel needs 4 kB of program memory, so it is possible to use FreeRtos on microcontrollers with 8kB of program memory. Originally, FreeRtos was ported on Atmega32. In the case of the proposed platform it was necessary to make a port for Atmega168 and Atmega128 microcontrollers. FreeRtos uses two methods to provide multitasking: tasks and coroutines. The first method (tasks) requires more resources because each task has separate stack. Tasks has not so many limitations as coroutines that work on common stack.

Atmeta128 has 128 kB of program memory and 4 kB of internal data memory extended by external chip to 64 kB and allows us to use FreeRtos with tasks. It is recommended to place stacks of the tasks in internal memory, so there is 4 kB for stacks available. Buffers and other structures has been moved to two times slower external memory.

Executing module is not well equipped. Its microcontroller has 16 kB of program memory and 1 kB of data memory. In order to save data memory, the FreeRtos is using coroutines, that complicates its programming.

2) Command Line Interpreter: The main module provides interactive communication with a user thanks to Command Line Interpreter. On the very beginning, the CLI was taken from AvrLib project [5] that is now in stagnation. Original CLI was not designed to multitask environment: only one instance of CLI was possible and furthermore it was working on global variables. The original CLI has also some bugs – some of them caused deadlock. Consequently, for the purpose of the proposed platform most of the codes of the original CLI has been rewritten. Now it is possible to use many independent instances of CLI. Each CLI has history of 4 last commands and works on a new engine. The memory usage is optimized. Additionally, all strings are placed in program memory, not in data memory, what is very precious. The proposed CLI is compatible with avrlibc and it is possible to use fprintf functions in order to make a print.

The new CLI API is friendly (it allows users to add new commands easily) and communication with the main controller is simple. The command *help* displays all available commands and its description. The proposed CLI, similarly to Cisco routers, works in three modes: normal, privileged and configuration. In each mode different commands are available. The CLI taken from AvrLib project was working in one mode only.

3) File system: An important part of operating system theory is devoted to file systems. For the proposed didactic platform a simple file system, the so-called Fat8, has been written. It can address up to 256 clusters (each cluster has 256 bytes). The whole implementation takes about 500 lines of code and is compatible with avrlibc API. The file is visible as a stream. Writing to file is possible using fprintf function. CLI provides commands for creating, deleting, editing and viewing files. The command *dir* allows to list all files. The file system does not support directories. All files are placed in main directory. In the presented platform the memory available for file system is limited to 32 kB.

4) Communication protocol: The main controller and the executing modules are connected to a common medium – RS485 bus. The communication model is the following. The main controller (master) starts the transmission on the bus. Each frame sent by the master main controller has an address of a slave device (an executing module) – the receiver of the message. The slave device can answer to the message. The frame format is Type Length Value. The frame fields are the following: synchronization sequence, address, type of message, message length and message data. Two bytes with CRC sum end the frame. Most of the messages with non matching address are ignored. There is only one exception of the rule, presented in the next paragraph.

5) Bootloader: The bootloader is mainly used when STK 500 programmer is not available or when is not connected. The main module can flash firmware to the executing module. With the help of xModem protocol the firmware image is uploaded first to the main controller and stored in a file. Next, the main controller sends restart command to the executing module. If the address is matched, the device restarts. Otherwise the device disconnects from RS485 bus for 60 seconds – this is enough to write firmware to executing module. After restart of the executing module the bootloader code is executed. The bootloader waits 30 seconds for flash command. After receiving it, the executing module is trying to download firmware using xModem protocol. The main controller sends firmware according to xModem protocol.

6) TCP/IP stack: The TCP/IP stack implemented in the presented didactic platform is based on the stack proposed within project *HTTP/TCP with an Atmega88 microcontroller*

[1]The name "AVRlib" bears an unfortunate resemblance to "AVR libc" even though the two libraries are different and should not be confused. AVR libc is the Standard C Library for AVR-Series Processors and provides functions we typically think of as being standard C like printf, stdio calls, math functions, plus some AVR-specific functions. Procyon AVRlib provides additional higher-level functions designed to help designers accomplish typical embedded systems tasks.

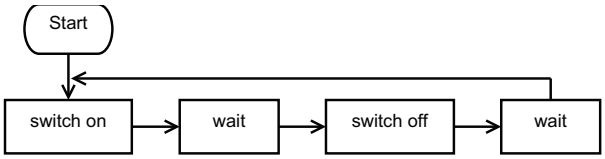

Fig. 5. Blinking diode algorithm

(AVR web server) [6]. For the purpose of our project, the TCP/IP working on Atmega88 with 8 kB of program memory was adopted for multitasking system. The TCP/IP stack is supported in the presented didactic platform only partially. At current stage only ICMP protocol and simple WWW server is working. The next releases of the didactic platform will include, among others, an implementation of servicing many TCP connections.

C. Software tools

The software toolset consists of editor (Integrated Development Environment – IDE), compiler, repository and programmer software.

1) IDE: Efficient programming can be supported by good selection of a editor. We have considered many candidates, among other eclipse, netbeams and kdevelop. We have selected kdevelop as the main editor of the proposed platform due to its best efficiency: it recognizes structures, completes the code and displays documentation in code.

2) Compiler: Besides many commercial compilers like ICCAVR and IAR, one of the most popular avr compiler is the avr-gcc. Avr-gcc compiler fulfils the main assumption of the proposed didactic platform: it is free of charge. Consequently, avr-gcc has been selected and the platform's compiles and, consequently, the firmware is optimized for avr-gcc compiler.

3) Repository: In order to synchronize our work on the platform's project, the code has been placed on SVN repository. The project is managed by trac system. The current project state is available at http://akme.yum.pl/trac, login and password are student.

4) Programmer software: As the programmer software for the didactic embedded platform avrdude software has been selected. The software work only in text mode what sometimes can be confused, especially during determining the correct fuse settings. In such a case it may be helpful to use WWW AVR fuse calculator http://www.engbedded.com/fusecalc/ determining avrdude parameters. The parameters are saved to makefile. The makefile created for the platform's project purposes includes the special command that flashes microcontroller. It is enough to write 'make program' command in the shell. All parameters are saved to the makefile. In our opinion the usage of programmer softwares with graphical interface can be dangerous because it is easy to set wrong fuse that cause bricking the microcontroller.

III. Laboratory exercises

The presented platform allows for preparing many exercises. Students can modify the Command Line Interpreter by adding new commands. It is also possible to add a new task (thread or coroutine) that can communicate with other tasks using global variables or queues. A new firmware can be flashed by sending its content by RS485 bus using xModem protocol. Students can also write programs that use file system API. In final work they can customize platform for a specific applications, for example temperature monitor where the temperature can be read by WWW interface or by CLI.

As an exemplary exercise for students it can be considered a program causing bliniking four diodes. Each diode has different time of being off and on and the diodes are independent from each other. The algorithm of blinking one diode is depicted in Figure 5. The diode is switching on and next system is waiting for a specified period of time. Next the diode is switched off and analogically the system is waiting for a specified period of time. Such task can be written in an infinite loop. The work is getting complicated if we want to write program with two blinking independent diodes. It can be done using timers and interruptions. So we may have as many blinking diodes as many timers are in microcontroller. If we wish to blink more diodes we have to modify the diode loop or use tasks.

The solution based on using tasks is easier and more readable. Each task blinks one diode. The tasks are executed parallely and they are independent. Adding a new task we can improve the program. The new task can scan the inputs. If the input key is pressed the task is sending information to the rest of the tasks about this event. Tasks that are responsible for blinking diode may change frequency of the diode blinking. The best method of communication between tasks is using queues or mutexes and global variables.

IV. Conclusion

The presented didactic system is a valuable addition to theory of operating and embedded systems. It enables students to get familiarized with aspects like multitasking, interprocess communication, process synchronization and networking. The presented solution is cheap and most of students can afford to build this device and use it for didactic or practical purposes limited only by their imagination.

Acknowledgment

The authors would like to thank to all developers taking part in open source projects cited in the article.

References

[1] Free Software Foundation, *GNU General Public Licence v2*. Boston, USA, 1991.
[2] W. Richard Stevens, Stephen A. Rago *Advanced Programming in the UNIX Environment*, 2nd ed. Addison-Wesley, 2005.
[3] Lewin Edwards, *Embedded System Design on a Shoestring. Achieving High Performance with a Limited Budget*, , USA: Elsevier Sience, 2003.
[4] Richard Barry *Using the FreeRTOS Real Time Kernel - a Practical Guide*
[5] Pascal Stang, *Procyon AVRlib*, Harlow, England: Addison-Wesley, 2006.
[6] Guido Socher *HTTP/TCP with an atmega88 microcontroller (AVR web server)*, http://www.tuxgraphics.org/electronics/200611/embedded-webserver.shtml 2006