Software architecture for a multi-purpose real-time control unit for research purposes

S Epple¹, R Jung¹, K Jalba² and V Nasui³

¹University of Applied Sciences Kempten, Research Centre, Innovapark 20, 87600 Kaufbeuren, Germany
²University of Applied Sciences Ulm, Production Engineering and Production Economics, Prittwitzstraße 10, 89075 Ulm, Germany
³Technical University Cluj Napoca, Northern University Centre of Baia Mare, Faculty of Engineering, Strada Doctor Victor Babeş 62A, Baia Mare 430083, Romania

E-mail: steffen.epple@hs-kempten.de

Abstract. A new, freely programmable, scalable control system for academic research purposes was developed. The intention was, to have a control unit capable of handling multiple PT1000 temperature sensors at reasonable accuracy and temperature range, as well as digital input signals and providing powerful output signals. To take full advantage of the system, control-loops are run in real time. The whole eight bit system with very limited memory runs independently of a personal computer. The two on board RS232 connectors allow to connect further units or to connect other equipment, as required in real time. This paper describes the software architecture for the third prototype that now provides stable measurements and an improvement in accuracy compared to the previous designs. As test case a thermal solar system to produce hot tap water and assist heating in a single-family house was implemented. The solar fluid pump was power-controlled and several temperatures at different points in the hydraulic system were measured and used in the control algorithms. The software architecture proved suitable to test several different control strategies and their corresponding algorithms for the thermal solar system.

1. Introduction
The needs and approaches for a control unit suitable in research environments was analyzed in [1]. This paper gives an overview of the software structure for this control unit.

The hardware architecture is shown in Figure 1. The hardware consists of a heterogeneous internal communication. The digital multiplexers are connected via SPI to the Microcontroller. The analog multiplexer is connected to one ADC input pin of the Microcontroller and switching is performed with an external integrated circuit. This is controlled by five port pins of the microcontroller (four pins addressing and one pin for inhibit). The real time clock, not shown in Figure 1, is connected via Two Wire Interface (TWI) protocol with chip select signals to normal port pins. LCD display is connected in four bit mode to one of the four 8-bit digital multiplexing output ports. One free pin of this port is used to control the backlight. The push buttons and indication LEDs are each connected directly to an output or respectively input pin of the digital multiplexer.
2. Why another free programmable control unit?
Conventional programmable logic controllers (PLC) are usually built up by blocks of predefined software, like PID block, Logic blocks and others. A great advantage is the reduction of development time by using these predefined blocks of software. On the other hand a great disadvantage in the case of different control strategies and algorithms to be tested, conventional PLCs usually cannot be implemented appropriately.

Another great advantage of free programmability is timing control. Whenever faster processing is needed, it can be implemented. For instance in a detected dangerous state of the system under control, the free programmable PLC can be programmed to act upon detection of the situation within a few clock cycles of the Microcontroller. On standard PLCs hazard detection is part of the control blocks within the normal control cycle. No background test routine can be run at higher speed for instance.

3. Goals of the software design for the new control
In the hardware design the number of input and output signals needed for the experiments and the type of these signals were defined.

In the software design the routines to control the hardware functions were implemented.
- We implemented some functions in different ways and compared their performance or behavior in the test case application.
- As we used free software for the hardware design, we continued this approach with the software and used the open source WinAVR [2], a windows distribution of the AVR Libc. It comes with gcc compiler and other preconfigured tools [3].
- For static code analysis the open source tool Splint was used [4].
- To gain a deeper understanding of the Atmel mega microcontroller series and its peripherals implementation and to give our students in future a possibility to easily “dive into” the matter of microcontrollers, we kept the implementations close to the datasheet of the controller [5].
- Another goal was to separate the logic and control-logic functions from the implementation of the timing and IO-management. The idea was to change only one .c – file when the application changes.
4. Safety and reliability considerations
Unlike the hardware design considerations, the software design should fulfill one small but no easy to reach point: It should not “get stuck”. This is important, as already mentioned in the hardware part [1] in our test case with the solar collector: The sun does not care about our control. So the collector will heat up as long as the sun shines on it. The software must not switch the solar fluid pump on, when the collector is filled with steam. This might cause a steam explosion.

On the other hand, if the controller gets stuck and temperature rises above steam temperature, then the controller must wait until it is safe again to switch on. So the measured temperature should be as exact as possible. Otherwise “sunshine time” might get lost for energy harvest.

5. Conceptual approach

5.1. Microcontroller system
The applied Microcontroller natively offers 128kBytes of Flash, which can be used for program and constants storage, and four kilobytes of RAM.

These memory resources of the microcontroller are pushed with external RAM circuitry to 64 Kbytes of hardware addressable RAM. With manual memory management, 512Kbytes can be used.

64kbytes of RAM turned out to be sufficient for control purposes with the specified control concept. However, we used the rest of the free 64kbytes memory for data logging.

5.2. Operating system
To realize many fast control functions with very little resources, we thought that it is necessary to reduce the operating system functions to an absolute minimum. So we had the idea to leave away any operating system completely and replace it with an architecture that enables real time capability on one hand and allows events on the other hand.

![Figure 2. Conceptual structure](image)

Legend: green: comfort functions, blue: real time or time critical functions

The approach is, to use a single large timer interrupt, as shown in Figure 2, which triggers all the time sensitive actions and calls their respective functions. The main timer interrupt was described as
suitable for very small systems only in [6]. The timer interrupt was split up to trigger the sampling of only one channel every 10 milliseconds. This behavior was described in [7, chapter 5] and named “to be in time”.

5.3. Communication
To allow event based communication, a standard interrupt to receive RS232 bytes was implemented. The RS232 Data is processed within the main routine. This way, the processing of serial data can be interrupted by the timer and does not disturb the time critical functions. This construct allows real time communication with other controls or RS232-sensors as well as PC communication. Under RS232-sensors we understand laboratory measurement equipment like mass spectrometers, optical spectrometers and so on.

Although we tested only PC communication yet, the second serial port can also be used for daisy chaining. Care must be taken that the controller is not fed with too much serial data within short time. This would lead to data loss of some extent as soon as the receiving ring buffer overflows. This effect could be prevented by a hardware-handshake. Unfortunately the hardware-implementation of the mega128 does not offer hardware-handshake (RTS/CTS) for the serial lines. An implementation in software is a future option. To be able to use the four line LCD display, the display communication algorithms were completely rewritten, since the used display behaved slightly different to several two line variants used before.

5.4. Timing and performance
Much care had to be taken to estimate the runtime of each function to avoid that the last function within the timer interrupt has finished when it is triggered again.

Such overrun situations were prevented by implementing a detection and clearing mechanism. For debugging, occurrences of overrun situations are reported to RS232 and can be logged by a PC. When properly designed, overruns should not occur.

All programming was done with “programmers’ notepad” which came along with the “WinAVR” package.

Due to low bit rates in the communication protocols of display and real time clock, communication between the microcontroller and its external peripherals is very time consuming. Many small delays must be implemented in the protocols. Without a dispatcher, this time can hardly be used for other calculations. A not negligible part of processing time is wasted for very small “waiting” purposes. This is particularly the case when protocols are implemented in software and run over normal IO-pins, - like the two wire interface (TWI) communication with the real time clock (RTC).

5.5. Display state machine
To be able to use different manipulation options in each menu, we did not use a list-based implementation for the menu system. Instead we implemented a rather large state machine. Each “menu” (=state) with user input options contains another state machine to process the input actions. In this way we realize modification of values with up/down and +/- buttons operating different in various menus.

6. Conclusion
The chosen software design proved very well in the tested configurations. We could implement and test several different control strategies for a solar system and could see great differences in solar performance, behavior and electric power consumption of the solar system.

Power control in the millisecond timing range proved to work well, with little restrictions due to jitter issues caused by the necessary slow communication with the LCD controller.

The tested power control strategies worked very well with the used solar fluid pumps. One pump was used for desktop-testing, a different brand and power was used in the test system.
Splint proved well to find systematic programming failures that gcc warnings did not complain. At least with the compiler options we used [4].

Whenever we had obscure behavior of the software during the implementation phase, a run through the static code analyzer Splint gave a clear answer where to take a closer look.

The control unit we tested controlled the solar system for several months without a watchdog-reset and without malfunction, proving its stability.

7. Outlook
Further improvements of the software may include hardware handshake on RS232 lines (RTS/CTS) as well as improvements of execution speed, primarily the display update-rate. This may depend on major changes in hardware design. Eventually the PC-RS232 will be replaced by an Ethernet connection with some web content capabilities. The free ram already allowed data logging for a period of about one week at one complete and uncompressed dataset per minute.

References
[1] Epple S, Jung R, Jalba K and Nasui V 2016 Real time capable control design with increased life expectancy for research purposes, Innovative Ideas in Science IIS2016, Baia Mare, Romania, November 11-12
[2] ***WinAVR: https://sourceforge.net/projects/winavr/files/
[3] ***http://www.nongnu.org/avr-libc/
[4] Evans D and Larochelle D 2002 IEEE Software, Jan/Feb http://www.splint.org/ and http://www.cs.virginia.edu/~evans/pubs/ieeesoftware-abstract.html
[5] ***Atmel Corporation, 2011 8-bit Atmel Microcontroller with 128Kbytes, In System Programmable Flash ATmega128 ATmega128L
[6] Rob W 2006 Real-Time Systems Development, Oxford, Butterworth-Heinemann
[7] Heinz W and Uwe B 2005 Echtzeitsysteme, Funktionsweisen, Anwendungen, Springer