Implementation and Application of a Simple Real-time OS for 8-bit Microcontrollers

DOLINAY J., VAŠEK V., DOSTÁLEK P.
Department of Automation and Control Engineering
Tomas Bata University in Zlin, Faculty of Applied Informatics
Nad Stráněmi 4511, 760 05 Zlín
CZECH REPUBLIC
dolinay@fai.utb.cz   http://www.fai.utb.cz

Abstract: - This paper presents small real-time operating system which has been developed at our institute and also an application and practical verification of this system in a portable data acquisition unit. The system, named RTMON is intended mainly as a teaching aid for lessons of microcontroller programming where it allows the students to simply write applications in C language with several concurrently running processes. However, it can also be used in practical applications, as is demonstrated by the data acquisition unit described in this paper. The system works on 8-bit microcontrollers with the HC(S)08 core made by Freescale and also on Atmel AVR Mega8 microcontrollers.

Key-Words: - real-time, operating system, microcontroller, HC08, ATMega8, data acquisition.

1 Introduction

Real-time operating system (RTOS) can help to solve the usual problems related to programming microcontroller applications, such as need for executing multiple tasks concurrently, quick response to high priority events, managing hardware resources of the MCU, etc. In our lessons we include also RTOS based programming.

On 16-bit or 32-bit MCUs, RTOS are used often; on smaller 8-bit systems it is not so common because these systems have limited memory and CPU power and it is more efficient to write the required program without RTOS. However, if the RTOS is small enough to fit into such MCU, it can bring the same advantages as on bigger MCUs. At our department in lessons of Microcontroller programming we use 8-bit MCU from the HCS08 family made by Freescale. As we wanted to include a RTOS programming techniques into our lessons we needed a RTOS capable of running on this MCU. Such system would also be useful for our other projects, where we use Freescale HCS08 MCUs. If we look at the available real-time operating systems, there are many of them. But most of them are focused on bigger, 16 and 32-bit MCUs, although there are some which support also small 8 bit MCUs, for example, FreeRTOS [2] which is distributed under GPL license and currently officially ported to 23 architectures. Another example is MicroC/OS-II [1], [3] which is also free for educational, non-commercial use. It is suitable for use in safety critical embedded systems such as aviation or medical systems and is ported to great number of architectures including Freescale HC08 and Atmel AVR. One disadvantage of using such system is that it is often quite complex due to the wide options it offers; typical RTOS for 32 bit MCU contains drivers for USB, Ethernet etc. Despite the fact, that the systems are configurable to work in simple arrangements, the user can still have too many things to worry about. Moreover, we already had an RTOS system developed at our institute for PC based systems and also for HC11, and its interface is known to the students. So, even if it would be possible to choose from existing systems, we decided to implement a light-weight clone of our RTMON system to use on the HCS08 microcontroller. Once the system was up and running for the HCS08 derivative used in lessons (GB60) it became useful to port it to other derivatives also. As a result, RTMON currently supports not only several members of the HCS08 MCU family but also Atmel AVR ATmega8. Adding new derivative is quite simple, so the list of supported derivatives will possibly grow in the future. Besides using the system in the lessons RTMON was also successfully used in design of portable data acquisition unit – DAQ, which allows simple and cheap interface between personal computer and a technological process. In the following text we describe the properties and usage of the RTMON operating system and also the DAQ device which uses this system.

2 The RTMON system

RTMON is pre-emptive multitasking OS which is highly simplified for easy use by the students. It is written in C language except for a small, platform-specific part written in assembler. The system supports execution of two different types of tasks (processes): normal processes which execute only once (such processes typically contain infinite loop) and periodical processes which are started automatically by the OS at certain period. These periodical processes are useful for many applications, typically in discrete controllers which need to periodically sample the input signal and update the outputs.

RTMON is used as a precompiled library accompanied by a header file. This simplifies the organization of the project and the build process. User enables RTMON usage in his program by including the header file (rtmon.h) in his source and adding the library to his project. Currently the library and sample projects are available two development tools: Freescale CodeWarrior and Atmel AVR Studio with WinAVR suite.

If needed, user can also rebuild the RTMON library. Typically this is useful to change the configuration such as maximum number of tasks, length of the OS time period (tick), etc. There is documentation which describes the procedure and also projects for the two supported IDEs, which can be simply opened and rebuild.

To make both the implementation of the system itself and its usage as simple as possible, several restrictions are applied. First, the RAM memory for processes and their stacks is statically allocated for the maximal number of processes as defined in configuration file. In the user program, it is not possible to use this memory even if there are fewer processes defined. In case more RAM is needed for the user program, the maximum number of tasks and/or stack-pool size can be changed in configuration file and the RTMON library must be rebuild.

The priority of each task must be unique, so that in each moment one task (the one with highest priority) can be selected and executed on the CPU. Processes can be created on the fly, but it is not possible to free and reuse memory of a process. No more than the maximal number of processes can be created, even if some processes were previously deleted.

However, these restrictions do not present any problem for most applications and allow for small kernel code size and ease of use.

2.1 Kernel objects

There are only two objects which RTMON contains: a process and a queue. The queues are buffers for transferring data between processes. Several queues can be created, each containing a “message” (data buffer) of certain size. The size can be specified when creating the queue and is limited by the total size of RAM reserved for all buffers of all the queues (queue pool size). Processes can read and write data to the queue and wait for the queue to become empty or to become full. This allows using a queue also for process synchronization.

2.2 Implementation of the system

The OS uses timer interrupt which occurs at certain period (e.g. 10 ms) to periodically execute the scheduler, which decides which process will run in next time slice. The timer interrupt routine is implemented in assembler for HCS08 MCUs and in C for AVR MCUs. It first stores CPU registers onto the stack and then calls RTMON kernel, which is a C function. The kernel then finds the process with highest priority which is in ready-to-run state and switches the context, so that the code of this process is executed after return from the interrupt service routine. If no process is ready to run, then a special dummy process is executed. This dummy process is contained within RTMON code and does nothing.

Task descriptor in RTMON is a C-language structure (IDPROC) which occupies 18 bytes of memory (given that char is 8-bit and int is 16-bit). The size of RAM required, for example, for 10 user-defined processes is then 12 x 18 = 216 bytes - there are two extra structures reserved for the init and dummy processes. The memory consumption may be reduced if we limit some of the values (e.g. stack size and time intervals) to 8 bits. This is enabled by RTMON_SMALL directive and it reduces the size of RAM required for one process to 14 bytes. There is an array of these structures with the number of items defined by RTMON_MAXPROCESSES constant in RTMON configuration file.

The structure for a queue (IDQUEUE) requires 10 bytes of RAM and similarly as for processes, RTMON allocates array of IDQUEUE structures with the number of items defined by RTMON_MAXQUEUES constant.

2.3 Services provided by the system

The OS provides set of services to user applications to manipulate processes and queues. Each service corresponds to a function in the RTMON library which user program can call. There are services for processes which allow to:

- Create a process
- Start a process
- Stop a process
- Delay (sleep) a process
- Continue (wake up) a process
- Abort (delete) a process

And for the queues there are the following services:

- Create a queue (specify size)
- Write to a queue with or without waiting
- Read from a queue with or without waiting

2.4 Usage of the system

To create an application which takes advantage of RTMON the user needs to perform just several simple steps:

Step 1:
Define variables for process identificators, e.g.:

```
IDPROC* init, *p1;
```

Step 2:
Initialize RTMON (typically in the main function):

```
rtm_init(&init);
```

Step 3:
Create user processes:

```
rtm_create_p("proc1", 10, proc1, 64, &p1);
```

This call creates process with priority 10 and stack size of 64 bytes. The body of the process is in function proc1 which should have the following prototype: void proc1(void). The variable p1 receives the ID of the newly created process and is used in all further calls to RTMON services to manipulate this process.

Step 4:
Start one or more processes:

```
rtm_start_p(p1,0,5);
```

This call starts process p1. The number 0 means that the process is started immediately (with delay of 0 ticks) and the number 5 means the process is started with period 5 ticks (it will be automatically started by RTMON each 5 ticks).

Step 5:
Delay the init function (the main process):

```
rtm_delay_p(init,0);
```

By this call the init process (main function) puts itself into infinite sleep and thus allows other processes to run. At this line the execution of main stops and it moves to the process with highest priority.

Code of each user process is contained in a C function. Example of a simple process could be:

```
void proc1(void)
{
  rtm_stop_p(p1);
}
```

This process does nothing, it just calls rtm_stop_p(p1) informing the system that it stopped execution.

3 Data acquisition device with RTMON

As already mentioned, RTMON is used in our lessons of microcontroller programming. But besides this usage the operation of the system was also verified by using it in one of our devices – a multi-channel portable data acquisition device DAQ. This device was developed in our department mainly for controlling and monitoring of educational laboratory models. It offers cheap alternative to professional I/O cards and modules when a technological process needs to be controlled or monitored from a computer.

3.1 Hardware of the DAQ

Hardware design of the DAQ device offers 16 analog inputs with 12-bit resolution, 8 digital inputs and outputs and one analog output with 12-bit resolution. The design focuses on low power consumption which allows long operation when battery supply is used. The core of the DAQ device is 8-bit general purpose Motorola microcontroller 68HC908GP32 with Von-Neumann architecture which is fully up-ward compatible with the 68HC05 family. On the chip are integrated many useful peripherals including: timer interface with input capture and output compare functions, 8-channel analog-to-digital converter with 8-bit resolution, up to 33 general-purpose I/O pins, clock generator module with PLL, serial communication interface and serial peripheral interface. The MCU has implemented several protective and security functions such as low-voltage inhibit which monitors power supply voltage, computer operates properly (COP) counter and FLASH memory protection mechanism preventing unauthorized reading of the user’s program. Internal RAM memory has capacity of 512B and FLASH memory 32 KB. Internal clock frequency can be 8 MHz at 5 V operating voltage or 4 MHz at 3 V operating voltage. The MCU also supports wait and stop low-power modes [4].

Analog-to-digital conversion is performed by Linear Technology A/D converter LTC1298. It is micro power, 2-channel, 12-bit switched-capacitor successive approximation sampling A/D converter which can operate on 5 V to 9 V power supplies. Communication with microcontrollers is handled by 3-wire synchronous serial interface. It typically draws only 250μA of supply current during conversion and only 1nA in power down mode in which enters after each conversion [6].

Digital-to-analog circuit uses 12-bit D/A converter Burr-Brown DAC7611 with internal reference and high speed rail-to-rail amplifier. It requires a single 5 V supply. Power consumption is only 2.5 mW at 5 V. Build-in synchronous serial interface is compatible with variety of digital signal processors and microcontrollers [5].

3.2 Software of the DAQ

The software in the DAQ device is based on the RTMON operating system described above. The software is formed of RTMON core and individual processes which perform all necessary tasks. Each process activity is controlled by operating system core on the basis of process priority and other information stored in the task descriptor. Structure of the DAQ device firmware is depicted in the Fig. 1. As can be seen in the figure, there are 4 main processes and 1 interrupt handling routine.

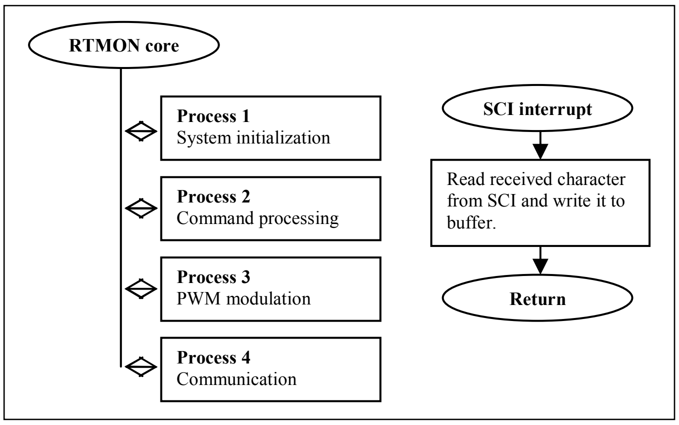

Fig.1. Internal software structure of the DAQ

Process 1 is highest priority process which performs DAQ device initialization after power up or reset. It sets all digital outputs to low state (logic 0), setups serial communications interface to communication speed of 57600 Bd, 8-bit data frame, 1 start bit and 1 stop bit, sets analog output to 0 V and finally initializes all necessary data structures. Because of its highest priority no other processes can be switched by OS core into the “run” state. After all initializations process suspends itself.

Process 2 performs all tasks related to command interpretation and execution. It waits for complete command string in the receiver buffer which is handled by serial communication interface (SCI) interrupt routine. This interrupt routine is automatically called when SCI receive one character from the higher-level control system. When command is completely received in the buffer, process will decode it and executes required action.

Process 3 is periodically activated process performing pulse-width modulation (PWM) on all 8 digital output channels when it is demanded. Its priority is set to higher level than process 2 and process 4 because the PWM is time critical function sensitive to accurate timing. Its 8-bit resolution allows setting of 256 different duty cycles at output. Period of the PWM signal is set to 1000 ms which is optimal value for many controlled systems with higher time constants.

Process 4 provides communication via RS232 serial interface with supervisory system. It generates responses to all commands regarding to defined communication protocol include error processing. It has the lowest priority from all the processes.

4 Conclusion

This paper presented our simple real-time operating system for Freescale HCS08 microcontrollers and an application of this system in portable data acquisition unit. The system is also used as a teaching aid for lessons of microcontroller programming. The interface is based on older version of RTMON operating system for PC and HC11 microcontroller (predecessor of HCS08). However, the internals of the system were written completely from the scratch to allow it to work with limited data and code memory of small 8-bit microcontrollers. RTMON is pre-emptive multitasking system which allows defining processes up to certain number (default is 10) and running these processes either in infinite loops or periodically with a given period.

The system services are very simple due to the limited memory of the target microcontrollers and intended use of the system, but still the system provides the advantage of easy implementation of embedded system as a set of independent, concurrently running tasks.

For the future it would be useful to port it to different MCUs, but also to extend the functionality by some I/O drivers, such us driver for GPIO, serial line etc.

RTMON proved to be functional during the lessons at our department, where it is used to demonstrate to students the basics of programming applications with operating systems, and it was also used in design of portable data acquisition unit DAQ. This device was also developed at our department and is used for control and monitoring related tasks. It is designed with respect to possible battery operation enabling measurement in areas where power source is not available. It provides sixteen analog inputs with 12-bit resolution, eight TTL compatible digital inputs and outputs protected against electrostatic discharge and overloading and one analog output channel equipped with 12-bit D/A converter. Communication with supervision system is realized with RS232 serial interface. It uses universal ASCII-based communication protocol which can be easily implemented in many software environments.

This work was supported by research project MSM 7088352102. This support is gratefully acknowledged.

References:

[1] Morton, T. D., *Embedded Microcontrollers*, Prentice Hall, 2001.
[2] FreeRTOS, *The FreeRTOS Project,* Available from: < http://www.freertos.org/>
[3] Micrium, *Micrium RTOS and Tools*, Available from: <http://micrium.com/page/products/rtos/os-ii>.
[4] Freescale, *M68HC08 Microcontrollers: MC68HC908GP32 Data Sheet.* Available from: <http://www.freescale.com/>.
[5] Burr-Brown, *DAC7611 : 12-Bit Serial Input Digital-to-Analog Converter.* Available from WWW: <http:// www.burr-brown.com/>.
[6] Linear Technology, *LTC1286/LTC1298 Micropower Sampling 12-Bit A/D Converters*, 1994. Available from WWW: <www.linear.com>