Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Volume 21, No. 1, ISSN 1726-9679
ISBN 978-3-901509-73-5, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2010
Make Harmony Between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals of DAAAM International

SIMPLE OPERATING SYSTEM RTMON FOR HC08 MICROCONTROLLERS

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

Abstract: *This paper describes simple real-time operating system RTMON which was developed at our institute as a teaching aid for lessons of microcontroller programming. The system allows students to simply write applications in C language with several concurrently running processes. It works with Freescale HCS08 GB60 and QE128 and Atmel AVR Mega8 microcontrollers.*
Key words: *hc08, real-time, operating system, microcontroller*

1. INTRODUCTION

When programming MCUs a real time operating system (RTOS) can be used to help solve the usual problems related to this task, 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 quite often; on smaller 8-bit systems it is not usually so as these systems have limited memory and CPU power and it is more efficient to write the required program without RTOS. Nevertheless, if the RTOS is small enough to fit into such MCU, it can still bring the same advantages as on bigger MCUs. In our lessons 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. When we look at the available RTOS there are plenty of them. As mentioned above, most of them are focused on bigger 16 and 32-bit MCUs, but there are some which support also small 8 bit MCUs, for example, FreeRTOS (***, 2009) which is distributed under GPL license and currently officially ported to 23 architectures. Another example is MicroC/OS-II (Morton, 2001; ***, 2010) 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 a professional 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. and even though it is 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 the RTMON system for our lessons. Once the system was up and running for our HCS08 derivative (GB60) it became useful to port it to other derivatives also. As a result, RTMON currently supports two members of HCS08 MCU family: GB60 and QE128, and also Atmel AVR ATmega8. Adding new derivative is quite simple, so the list will possibly grow in the future. In the following text we describe the properties of the RTMON for HC08 (and AVR) microcontrollers, the original system for PC and HC11 has wider options.

2. IMPLEMENTATION OF THE 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.

The 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.

For the sake of simplicity of both the implementation and usage, 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 data structures 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 Kernel operation

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.

3. RTMON SERVICES

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.
The services for processes are as follows:

- Create a process
- Start a process
- Stop a process
- Delay a process
- Continue process execution
- Abort (delete) a process

For queues there are the following functions:

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

3.1 Example of usage

To create an application which takes advantage of RTMON th e uses needs to perform 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

```
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.
The code of each user process is contained in a C function. Example of a simple process can 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.

4. CONCLUSION

A simple real-time operating system for Freescale HCS08 microcontrollers has been created, intended as a teaching aid for lessons of microcontroller programming. Its interface is based on already existing version of the system for PC and older HC11 microcontroller, but the internals have been written completely for the scratch to allow it to work with limited data and code memory of small 8-bit microcontrollers. RTMON is a pre-emptive multitasking system which allows defining processes up to certain number (typically 10) and running these processes either in infinite loops or periodically with given period.

The available functions 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.

The main limitation of current implementation can be seen in the relatively wasteful use of RAM memory for process stacks and queues which are allocated at build-time of the RTMON library and must be therefore defined in close relation with intended application, number of processes and their contents. This implementation is advantageous for school exercises because it allows easier usage, but for practical use in the industry it is not comfortable as it requires rebuilding the library for each application. However, this rebuild is not a difficult task, so even in this implementation the system is usable.

Currently, RTMON is ported to Freescale HCS08 GB60 and QE128 derivatives and to Atmel AVR Mega8. The source is structured for easy modification and porting to other platforms. For the future it would obviously be desirable to port it to different MCUs, but also to extend the functionality by some I/O drivers, such us driver for general purpose inputs and outputs, serial communication etc.

5. ACKNOWLEDGEMENTS

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

6. REFERENCES

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

*** (2009) http://www.freertos.org/ - The FreeRTOS Project, Accessed on: 2010-02-27

*** (2010) http://micrium.com/page/products/rtos/os-ii - Micrium RTOS and Tools, Accessed on 2010-05-05

*** (2009) http://www.freescale.com – Freescale 8-bit Microcontrollers, Accessed on 2009-12-12

*** (2010) - http://www.atmel.com, Atmel 8 and 32-bit MCUs, Accessed on 2010-05-12