AN2751

# Process Scheduling on an 8-bit Microcontroller

## Introduction

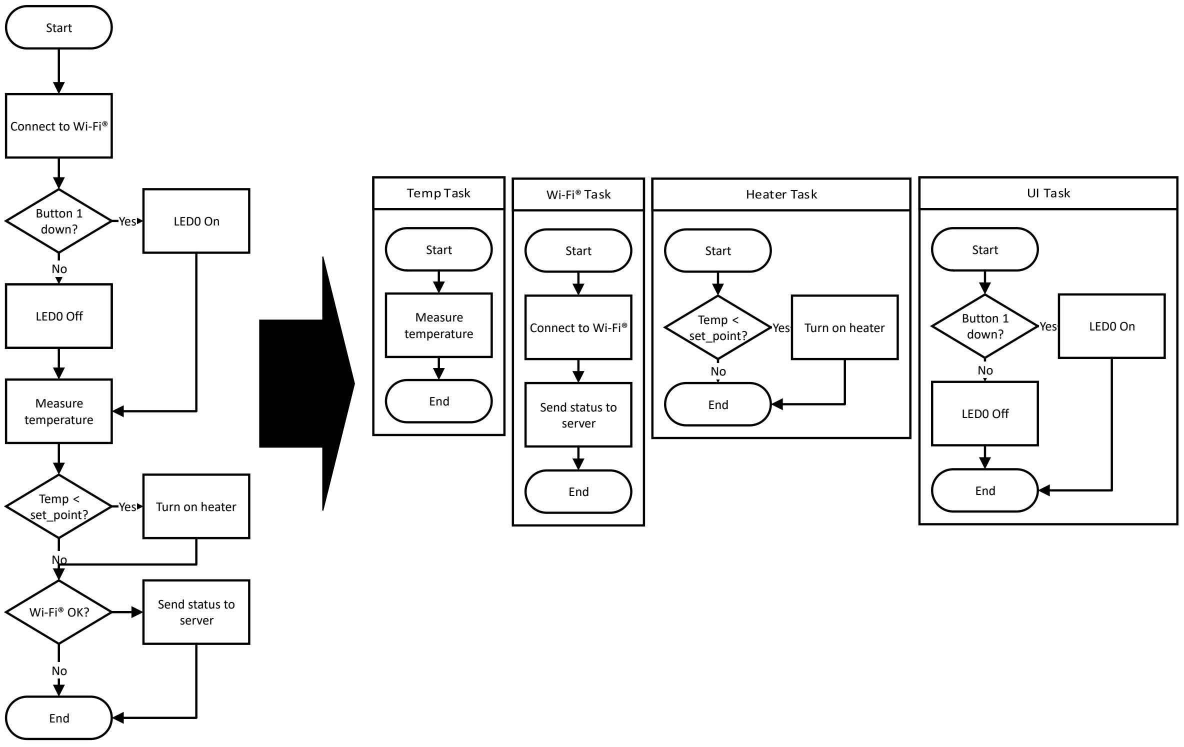
Author: Ivar Holand, M.Sc., Microchip Technology Inc.

In today's consumer market, products tend to do more and consumers generally expect more. Washing machines report remaining time in a smartphone app. Smoke detectors do not just blink and beep - they communicate with other smoke detectors to locate the fire, alert emergency personnel, and inform the consumer about the location of the fire through voice messages. Many embedded applications, previously relying on a button and an LED, now report their status and get their input over different channels. Features that are perceived as simple, and often commoditized, put a high demand on the underlying software and hardware. According to the AspenCore EE|Times/embedded.com *2017 Embedded Markets Study*, 19% think that managing increases in code size and complexity is their greatest technology challenge next year (2018), closely followed by integrating new technology or tools, 18%, and security concerns, 17%.

In conventional simple embedded systems, software is developed as linear code, with real-time requirements taken care of by dedicated on-board hardware, or as peripherals on a microcontroller (MCU). The moment all the peripherals, PCB real-estate, or budget is exhausted, the developer is forced to embed the real-time tasks inside a linear code, and fine-tune the system to get the expected behavior. Code re-usability is drastically reduced, and small changes in external hardware or application requirements make a big impact on software development and application testing.

A solution to this problem is to move to some sort of scheduling system as shown in Figure 1, allowing tasks to be assigned to a queue, prioritized, and run when needed. The AspenCore EE|Times/ embedded.com *2017 Embedded Markets Study* shows that 67% of all embedded projects run on an operating system, RTOS, kernel, software executive, scheduler, or similar. Of these, Embedded Linux is the biggest with 22%, closely followed by FreeRTOS at 20%, and in-house/custom at 19%.

**Figure 1. Moving from Linear Application Code to Task Driven Application Code**



## Table of Contents

# 1. Solutions

## 1.1 No Scheduling

A common approach to the problem is to carefully design the system in a way that ensures critical operations get enough CPU time, and that tasks with less timing constraints are interleaved. An example is that instead of the CPU busy-waiting for the 1-second period to then turn off an LED, it uses this time to check the interface buttons at a regular interval. However, this means that the LED task and the button-checking task will be highly dependent upon each other. Adding more tasks to the system requires the same careful integration to prevent altering the overall system performance.

Another approach is to identify all critical tasks and move them to an interrupt or hardware context, i.e., the function is directly supported by an MCU peripheral. Low priority tasks are moved to a `while(1)` loop where each task gets the necessary time to complete. If the implementation is done correctly, the system is not perceived as slow as all real-time critical tasks are handled immediately using a context switch in the form of interrupts to the CPU. For the developer this implementation is straightforward when the timing requirements are simple, or relatively simple; all low priority tasks will always run in a single pass of the `while(1)` loop. This scheme is moving towards what is called a cooperative multitasking system since each task cooperatively gives up its CPU time to the next task.

Complexity increases the moment tasks are expected to run at different time periods, or when time critical tasks must be moved from hardware/interrupt context to the `while(1)` loop. An example is a system in which an LED is blinking each 1 second, the user interface buttons are checked for status change each 250 ms, analog sensors are measured every 2 minutes, etc. Requirements like the ones listed above mean that one pass of the `while(1)` loop cannot be longer than the required latency in the system, in this example 250 ms for checking the user interface. Additional control software, like state machines, is required to control if the specific task should run at the given time around the loop.

## 1.2 Hybrid Hardware Driven Scheduling

MCUs generally offer hardware features (peripherals) for managing time bases, e.g., real-time counters/clocks, general timer counters, system ticks, watchdog timer, etc., which all can be used to drive a scheduling scheme. A common approach is to implement the `while(1)` application loop, as discussed above, where each task is associated with a global flag that controls whether or not the task is due to run in the next possible time slot. The associated flag is in turn controlled by a peripheral interrupt, which controls one or several flags depending on the application.

This approach somewhat simplifies the control of each task, and when it is due to run. It also makes code and tasks more reusable between applications. However, the same latency constraints exist, since the task with the longest execution time cannot run longer than the expected latency in the system. Also, the number of available peripherals limits the number of available control flags, which in turn limits the controllable number of tasks.

## 1.3 Homebrew Multitasking

Taking the scheduler a step further is to allow tasks to be suspended as needed in the middle of execution, without the task voluntarily giving up its CPU time. This scheme is known as a preemptive multitasking scheduling scheme. Preemptive multitasking involves an interrupt mechanism which can suspend the currently executing task and use the scheduler to determine which task should execute next.

Tasks are no longer dependent on each others timing, so this means increased flexibility for the developer.

However, developing a homebrewed system for handling a preemptive multitasking scheme is orders of magnitude more complex than the other schemes discussed above. Deep knowledge and insight into the CPU, registers, and RAM architecture is needed to be able to develop a good preemptive multitasking system. The AspenCore EE|Times/embedded.com *2017 Embedded Markets Study* shows that almost 1 in 5 applications that run a scheduling system run on an in-house/custom scheduler.

## 1.4 Commercial Preemptive Multitasking Systems

In a preemptive multitasking system tasks are preempted or halted before they complete. Either the scheduler can control this, or each task can agree on being preempted at certain points in their execution. Tasks can give up their CPU time while waiting for data to arrive on a queue, a signal to be ready, or a hardware resource becoming available to the task, etc. A combination of both forceful preemption and volunteer preemption is common. Tasks can be assigned priorities, and higher priority tasks can halt low priority tasks while the higher priority tasks run. In addition, tasks can give up their CPU time while waiting for something to happen, even though the task is not complete, allowing other lower priority tasks to run or the CPU and system to sleep to conserve power.

There are several commercially available multitasking systems for the embedded developer. Depending on the functionality they offer they are regarded as plain schedulers, operating systems (OS), or full real-time operating systems (RTOS). Some are available for purchase while others are free and open sources.

(RT)OSs usually offer a set of associated functions available for the user. These can be configurable scheduling schemes, configurable task priority, and many features associated with controlling task execution. Common control functions are semaphores, Mutexes, queues, software timers, etc. In addition, some systems also offer drivers to internal and external peripherals.

## 2. Software Development Strategy Pros. and Cons.

Many embedded systems are so simple, or the tasks so few, that they do not require scheduling. On-board peripherals can solve any real-time or scheduling requirements, and, e.g., timers can be used to blink LEDs. External interrupts can react to button inputs. Few external signals are monitored, and few actions are necessary to achieve the expected behavior. In these systems, a full-blown RTOS scheduler would be overkill and will overcomplicate the software. In addition, the total software size will probably be so small that the implementation is easy to follow. There is little to no benefit in splitting the application into several executable tasks. Code reuse in this scenario is probably kept on a driver level.

In slightly more advanced applications, global flags and state machines in combination with hardware interrupts can be used to control the application flow. Tasks become more flexible, and it should be possible to re-use tasks between applications. However, the total system needs to take all tasks into account, as the run-time of single tasks greatly affects the overall system performance. Depending on the complexity of the application and the number of hardware interrupts available in the selected MCU, this might still be the preferred solution for many developers. This implementation is straightforward and can be tailored to fit the exact application need, and in that way, be very code size effective.

A homebrewed multitasking system is a big development effort. Once developed, the solution should be used for several projects to justify the development cost. In-depth knowledge of the CPU, registers, and RAM architecture is needed to develop the system. The system must be ported and developed for all the intended devices and platforms. Also, extensive testing and verification are needed as it is difficult to develop this functionality without the scheduler constantly manipulating critical CPU registers. The benefit is that the system can be tailored to fit the exact application need, which, in turn, means that code size and code overhead should be at a minimum.

A commercially available (RT)OS requires the developer to be comfortable using large software stacks. There is some cost involved for the developer to learn the selected (RT)OS, but once learned, the (RT)OS can be re-used across projects, applications, and often platforms. The drawback with a commercially available (RT)OS is that the system is developed to be as general as possible to work on many different applications and platforms. For the developer, this means that some software overhead will be present in the final application. Most (RT)OS implementations try to counter this by offering as much configurability as possible. The biggest learning effort for the developer is often configuring the (RT)OS correctly. A big benefit is that the (RT)OS is tested and verified over several devices, architectures, and platforms.

## 3. Industry Standard RTOS/FreeRTOS™

According to the AspenCore EE|Times/embedded.com *2017 Embedded Markets Study*, 22% of applications running on an operating system, RTOS, kernel, software executive, scheduler, or similar use Embedded Linux. Since the focus of this paper is 8-bit microcontrollers, Embedded Linux is somewhat out of the scope due to the lack of support for 8-bit microcontrollers, and also the hardware required to run Embedded Linux is not generally available in 8-bit microcontrollers.

In the same survey, 20% stated that they use FreeRTOS™. FreeRTOS supports the 8-bit AVR® MCU and has done so for many years. FreeRTOS offers preemptive multitasking out of the box, together with many other features associated with RTOS functionality, e.g., semaphores, Mutexes, queues, software timers, etc. All of these features greatly simplify development of software components. The RTOS takes care of inter-task timing requirements, which means the tasks will be more or less independent of each other. This, in turn, implies that software built for one application can be, with little or no alteration, re-used in other applications.

Using FreeRTOS is as simple as adding the software library/source code to the project, as one would add any driver or middleware to any application. Once the required source files have been added to the project, tasks are created as separate never returning functions:

```
void my_task(void *pvParams)
{
    /* Task init code, if any, here */
    for ( ;; ) {
        /* Task application code here */
    }
}
```

The task is added to the scheduler task queue by:

```
xTaskCreate(my_task, "My Task", [stack size for task], NULL, [priority], NULL);
```

When all tasks are added to the queue the RTOS is started by calling:

```
vTaskStartScheduler();
```

The scheduler now takes care of all task priorities. Each task can be viewed as single executables.

## 4. Benefits of Using FreeRTOS™ on AVR® Microcontrollers

The AVR® microcontrollers support FreeRTOS™. An example made for the ATmega4809 microcontroller is available in Atmel | START (http://start.atmel.com/#examples/atmega4809/freertos). Some older examples are available from the FreeRTOS web page: http://www.freertos.org. The example in Atmel | START is ready to be used out of the box in Atmel Studio.

Using FreeRTOS with the AVR microcontroller typically consumes 2400 bytes of flash and 51 bytes of RAM plus a configurable amount of heap (stack) allocated for the tasks. The above numbers are from the START example with all but one task stripped from the application. Adding one additional task consumes additionally 22 bytes of flash, and no additional RAM as each task gets its stack allocated on the heap. Adding features like stream support, Mutex, queues, etc. will as expected consume extra flash.

There is some overhead in using FreeRTOS, but for larger software projects the benefits easily overcome this code overhead. Tasks can be re-used between applications and platforms. By working on different tasks, developers can work independently on different parts of the application. The developer does not need to worry about creating timing issues for other tasks in the same application. Also, applications can easily be expanded by simply adding a new task. In addition, FreeRTOS also offers many useful features for the developer which one normally would spend a lot of time creating for each application. E.g., buffer, stream, and queue functionality are trivial to implement in FreeRTOS.

FreeRTOS offers debugging functions like stack painting and stack overflow hooks. These capabilities are easily enabled in the *FreeRTOSConfig.h* file by setting `configCHECK_FOR_STACK_OVERFLOW` to either "1", or "2". With a setting of "1" to enable the stack overflow hook, FreeRTOS will then call the function,

```
void vApplicationStackOverflowHook( TaskHandle_t xTask, signed char *pcTaskName );
```

defined by the user, if the task's stack pointer goes outside the allocated area. A setting of "2" will fill the stack with a known value at the start. These tools together with the extensive debugging features offered by the AVR microcontrollers make it easy to fine-tune the tasks' memory consumption, e.g., by reading out the content of SRAM during run-time debugging. That way, it is easy to pinpoint stack overflow issues in the application that would otherwise be difficult to detect and debug.

## 5. Revision History

| Doc. Rev. | Date | Comments |
|---|---|---|
| A | 07/2018 | Initial document release. |

## The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under “Support”, click on “Customer Change Notification” and follow the registration instructions.

## Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

## Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

## Legal Notice

## Trademarks

ISBN: 978-1-5224-3304-0

## Quality Management System Certified by DNV

**ISO/TS 16949**
Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

# Worldwide Sales and Service

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/
support
Web Address:
www.microchip.com

**Atlanta**
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

**Austin, TX**
Tel: 512-257-3370

**Boston**
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

**Chicago**
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

**Dallas**
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

**Detroit**
Novi, MI
Tel: 248-848-4000

**Houston, TX**
Tel: 281-894-5983

**Indianapolis**
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

**Raleigh, NC**
Tel: 919-844-7510

**New York, NY**
Tel: 631-435-6000

**San Jose, CA**
Tel: 408-735-9110
Tel: 408-436-4270

**Canada - Toronto**
Tel: 905-695-1980
Fax: 905-695-2078

## ASIA/PACIFIC

**Australia - Sydney**
Tel: 61-2-9868-6733

**China - Beijing**
Tel: 86-10-8569-7000

**China - Chengdu**
Tel: 86-28-8665-5511

**China - Chongqing**
Tel: 86-23-8980-9588

**China - Dongguan**
Tel: 86-769-8702-9880

**China - Guangzhou**
Tel: 86-20-8755-8029

**China - Hangzhou**
Tel: 86-571-8792-8115

**China - Hong Kong SAR**
Tel: 852-2943-5100

**China - Nanjing**
Tel: 86-25-8473-2460

**China - Qingdao**
Tel: 86-532-8502-7355

**China - Shanghai**
Tel: 86-21-3326-8000

**China - Shenyang**
Tel: 86-24-2334-2829

**China - Shenzhen**
Tel: 86-755-8864-2200

**China - Suzhou**
Tel: 86-186-6233-1526

**China - Wuhan**
Tel: 86-27-5980-5300

**China - Xian**
Tel: 86-29-8833-7252

**China - Xiamen**
Tel: 86-592-2388138

**China - Zhuhai**
Tel: 86-756-3210040

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-3090-4444

**India - New Delhi**
Tel: 91-11-4160-8631

**India - Pune**
Tel: 91-20-4121-0141

**Japan - Osaka**
Tel: 81-6-6152-7160

**Japan - Tokyo**
Tel: 81-3-6880- 3770

**Korea - Daegu**
Tel: 82-53-744-4301

**Korea - Seoul**
Tel: 82-2-554-7200

**Malaysia - Kuala Lumpur**
Tel: 60-3-7651-7906

**Malaysia - Penang**
Tel: 60-4-227-8870

**Philippines - Manila**
Tel: 63-2-634-9065

**Singapore**
Tel: 65-6334-8870

**Taiwan - Hsin Chu**
Tel: 886-3-577-8366

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830

**Taiwan - Taipei**
Tel: 886-2-2508-8600

**Thailand - Bangkok**
Tel: 66-2-694-1351

**Vietnam - Ho Chi Minh**
Tel: 84-28-5448-2100

## EUROPE

**Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

**Denmark - Copenhagen**
Tel: 45-4450-2828
Fax: 45-4485-2829

**Finland - Espoo**
Tel: 358-9-4520-820

**France - Paris**
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

**Germany - Garching**
Tel: 49-8931-9700

**Germany - Haan**
Tel: 49-2129-3766400

**Germany - Heilbronn**
Tel: 49-7131-67-3636

**Germany - Karlsruhe**
Tel: 49-721-625370

**Germany - Munich**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Germany - Rosenheim**
Tel: 49-8031-354-560

**Israel - Ra'anana**
Tel: 972-9-744-7705

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Italy - Padova**
Tel: 39-049-7625286

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**Norway - Trondheim**
Tel: 47-7289-7561

**Poland - Warsaw**
Tel: 48-22-3325737

**Romania - Bucharest**
Tel: 40-21-407-87-50

**Spain - Madrid**
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

**Sweden - Gothenberg**
Tel: 46-31-704-60-40

**Sweden - Stockholm**
Tel: 46-8-5090-4654

**UK - Wokingham**
Tel: 44-118-921-5800
Fax: 44-118-921-5820