# **Task Scheduling on the AVR**

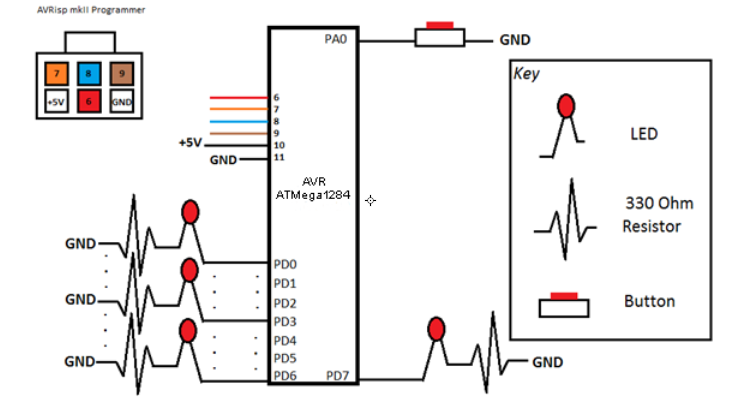
(1 day lab)

In this lab you will use a task scheduler to schedule concurrent tasks on an AVR. Scheduling concurrent tasks is necessary when each task has different periods.

## PRELAB

Read Programming Embedded Systems, Chapter 8 - Task Scheduler (it's short). The code template for the task scheduler can be copy/pasted into your lab assignment, making necessary adjustments for the code to work with Atmel Studio.

Build the following hardware circuit. If you don't have materials yet, you can group with someone who does (this week only).



**PART 1:**

RIMS uses simple function calls like TimerSet(), TimerOn(), and TimerISR(). The AVR has no such functions, thus the functions need to be reimplemented for the AVR platform. Implement the three functions below using AVR Studio. Target the 16-bit 'Timer1' peripheral on the AVR (Timer0 is only 8-bits and Timer1 is 16-bits).

For reference on the ATMega1284, go to Student Materials -> labs -> datasheets.

TimerOn should set up the timer and begin counting. We need to set the timer to the appropriate mode and enable the appropriate interrupt. We should also determine the appropriate prescaler value to use for determining the interrupt rate of the timer in TimerSet().

***void TimerOn() {***

TCCR1B = (1<<WGM12)|(1<<?); // (ClearTimerOnCompare mode). // Prescaler=?

TIMSK1 = (1<<??); // Enables compare match interrupt

SREG |= 0x80; // Enable global interrupts

***}***

TimerSet should set the rate that the timer will call the TimerISR function. We will use the compare output match interrupt vector, which means that whenever the timer counter in TCNT1 counts up to the value contained in OCR1A, the TimerISR function will be called. Figure out the correct value of OCR1A to use, given that your ATMega1284 has an internal frequency oscillator of 8MHz, and you have set a prescalar value in TCCR1B. Hint: Use unit cancellation and algebra to figure out how many ticks it takes to wait a millisecond. To get you started, the number of timer ticks/second = ((8e6 ticks/second) / (prescaler value)).

***void TimerSet(int ms)*** {

TCNT1 = 0; // Clear the timer counter

OCR1A = ??; // Set the match compare value

}

TimerISR is the function that contains the task scheduler, and should be called when the timer compare match interrupt is fired. In RIMS, TimerISR is the name of the interrupt vector, but it differs on the AVR. On the AVR, either use the following function to replace TimerISR, or simply insert a call to TimerISR inside.

**ISR(TIMER1\_COMPA\_vect) {**

//task scheduler

**}**

Once you have set up all of the above functions, implement the following tasks using the task scheduler.

1) Blink the LED at D0 at a rate of 500 ms.

2) Blink the LED at D2 at a rate of 1000 ms.

3) Blink the LED at D4 at a rate of 2500 ms.

When you are done, show your TA.

**Video Demonstration:** [**http://youtu.be/wjoOTpu494s**](http://youtu.be/wjoOTpu494s)

## PART 2:

Using the task scheduler, implement a simple morse code to ASCII translator.

For those unfamiliar with morse code, more information can be found on wikipedia at<http://en.wikipedia.org/wiki/Morse_code>

A button on A0 is used to send morse code signals to your translator. Use a single syncSM task to decode signals and determine if they are dots or dashes.

One method for doing this might be to say that a dot is any '1' on A0 that lasts less than ~300 milliseconds, and a dash lasts greater than 300 milliseconds (choose the task period appropriately). A signal could be considered complete if no '1' on A0 is detected for 1 second. You can ignore the actual morse code timing specifications - use your own timing intervals as desired.

Once you have decoded a complete signal, display the ASCII translation on a bank of 8 LEDs for 5 seconds.

If a complete morse signal is invalid, display 0xFF on the LED bank.

Hint: There are 36 alphanumeric morse codes (A-Z, 0-9). When receiving the morse signal, create a char array of '0s'(dots) and '1s'(dashes). Then write a function that creates a unique hash value for the char array. Use the hash value to index into a lookup table that contains the corresponding ASCII code for the given code.

When you are finished, show your TA how your system decodes S-O-S.

**Video Demonstration:** [**http://youtu.be/ygidHesNfnU**](http://youtu.be/ygidHesNfnU)

## Part 3 (Challenge)

Expand upon Part 2 by implementing a queue that will allow multiple morse signals to be received while still displaying one morse signal every 5 seconds. A queue.h file can be found in **Student Materials -> labs -> .h files**.

Instead of writing a complete morse signal to the LED bank, push the signal onto a queue.

Write a concurrent task to output the ASCII value of the queue's top element on the bank of LEDs. Output each received character in ASCII code for 5 seconds. If the morse code signal that you received is invalid, output 0xFF.

When you are finished, show your TA how your system decodes S-O-S.

**Video Demonstration:** [**http://youtu.be/jXyZmvpokwo**](http://youtu.be/jXyZmvpokwo)

Zip up all of your source code (only .c, .h files) and submit it on iLearn according to the lab submission guidelines found on the GDrive.