**Demonstration:** (NOTE: these change based on the lab - enter the demo tasks specified in each lab)

LCD shows names, brightness, temperature and internal voltage values \_\_\_\_\_\_\_\_

Pressing Switch 1 increases the brightness of the LEDs, pressing switch SW2

decreases the brightness of the LEDs, and pressing switch SW3 transmits the

text to the host PC. \_\_\_\_\_\_\_\_

Turning Analog In potentiometer controls the brightness and brightness values

are updated as the brightness of the LEDs changes \_\_\_\_\_\_\_\_

The date and time, brightness value, temperature value, internal reference

voltage value, and the CRC-32 values are shown on the host PC console \_\_\_\_\_\_\_\_

Commands are typed on the host PC console and the valid commands are executed \_\_\_\_\_\_\_\_

**Requirements:**

The code generated is written in C for the YRDKRX63N Evaluation Board. \_\_\_\_\_\_\_\_

Task 1 controls the brightness of LED4 to LED15 using a PWM signal generated

using a delay function \_\_\_\_\_\_\_\_

LCD should display your group member(s) first name on line 1 and line 2 \_\_\_\_\_\_\_\_

LCD should display the brightness value on line 3 as B Value= XXX, where XXX

is the duty ratio. [Example: ‘001’ corresponds to a value of ‘1’], the temperature

value on line 4 as T Value=XX.X, where XX.X is the temperature in degree Celsius,

the internal reference voltage on line 5 as V Value=X.XX, where X.XX is the internal

reference voltage. \_\_\_\_\_\_\_\_

Maximum value of brightness is 100 and the minimum is 0. Zero brightness value

means the LEDs are off \_\_\_\_\_\_\_

Switch 1 ISR is used to increase the brightness in steps of 10, switch 2 ISR is used

to decrease the brightness in steps of 10 and “Analog In” potentiometer is used to control

the brightness in steps of 1 [Range: 0 – 9], and switch 3 ISR is used to transmit the text

to the host PC \_\_\_\_\_\_\_\_

ADC interrupts are used for temperature reading and internal reference voltage,

but no timers \_\_\_\_\_\_\_\_

UART (SCI2) is used for serial communication at a baud rate of 9600 bps \_\_\_\_\_\_\_\_

Task 3 is used to output the date and time, brightness value, temperature value,

internal reference voltage value, and the CRC-32 value are shown on the host PC

console as YYYY-MM-DDThh:mm:ss:B Value=XXX;T Value=XX.X;V

Value=X.XX CRC-32:XXXXXXXX \_\_\_\_\_\_\_\_

Task 4 executes the commands that are typed on the host PC console and

transmitted using UART (SCI2) at a baud rate of 9600bps \_\_\_\_\_\_\_\_

The receive buffer is created dynamically and is incremented only

in chucks of ten bytes \_\_\_\_\_\_\_\_

Valid commands should be 8 characters ling and a comma is used to

separate the commands \_\_\_\_\_\_\_\_

A command should be of the form LEDXX=YY or PWM01=ZZ,

where XX is 04 to 15, YY either is 00 (off) or 01 (on), and ZZ is

00 (0%) to 10 (100%) \_\_\_\_\_\_\_\_

PWM duty ratio set from the console should add the duty ratio from the potentiometer. \_\_\_\_\_\_\_\_

Invalid commands are ignored and valid commands are executed from the

command list \_\_\_\_\_\_\_\_

The text shown in req i is printed on the host every minute (do not use real time

clock interrupts) \_\_\_\_\_\_\_\_

At least four tasks are created with task 1 having highest and task 4 having

lowest priority \_\_\_\_\_\_\_\_

Tasks are scheduled with RTC scheduler’s APIs. \_\_\_\_\_\_\_\_

The code is well documented and easy to follow. \_\_\_\_\_\_\_\_

**Learning Objectives:**

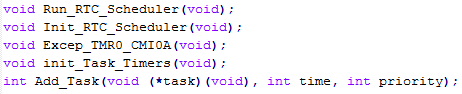
The experiment shows the conversion of analog inputs to digital, displays them on the LCD, reads ADC values for reference voltage and temperature, displays CRC-32 of the string created from the data, allows UART control of the board by serial commands, drives outputs via ISR’s, updates and keeps time using an RTC, as well as implementing a run to completion task scheduler.

**General Steps:**

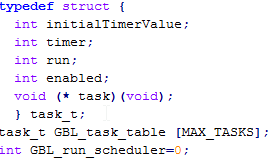
1. Go to the HEW IDE
2. Power up the Renesas RX63N by connecting it to the computer via mini-USB
3. Build upon previous lab and add UART/CRC-32 functionality and other requirements.
4. Demonstrate the working project.

**Detailed Steps:**

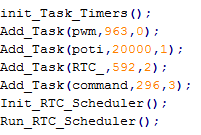
1. Building upon the work from last lab session (Lab 6). The RTC, real time clock, was implemented such that a time string was output to the host console and the CRC value. This value was checked using CRC and concatenated with other outputs to form a final string. The run to completion scheduler was implemented in such a way that a task can be added to the schedule and depending on its priority it can be run at the specific times designated.
2. After verification of the working previous lab the run to completion scheduler was implemented. The functions necessary for the task scheduler were added to the code.



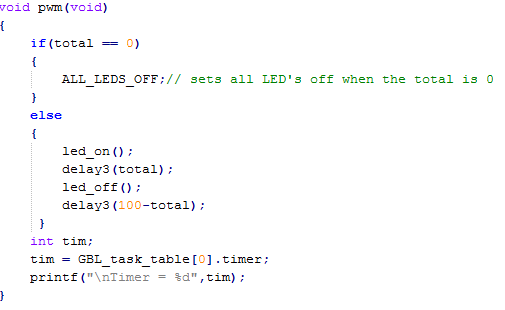
1. A typedef is defined to hold the structure for the task scheduler

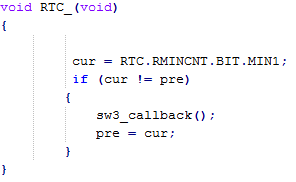


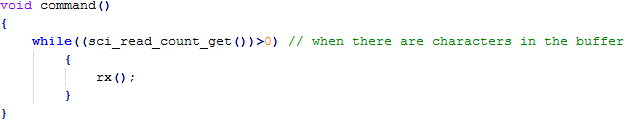
1. With the verified functionality of the RTC scheduler the next step is remove the while loop from the main and implement the task scheduler.



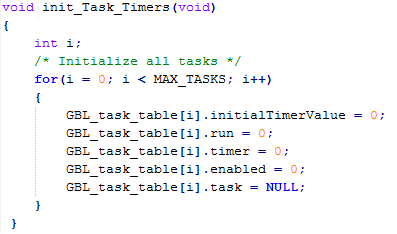
1. Add\_Task function calls the task then the time to execute the task and then the priority of the task.
2. In order to have the run to completion scheduler work properly we had to move most of the code that was in the main loop into separate functions that could be called. This freed up room in the main loop and made the code more modular. With this change the task timers functions as expected and did not have large variances in times as they did before moving everything to functions.

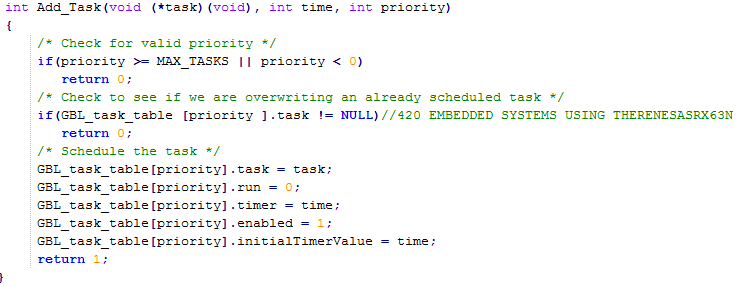




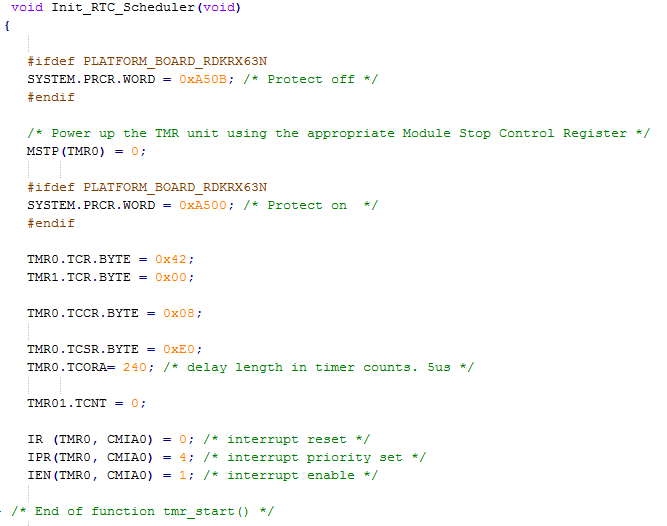


1. The functions defined in the prototype functions stage are created based on example code from the lecture slides and book.

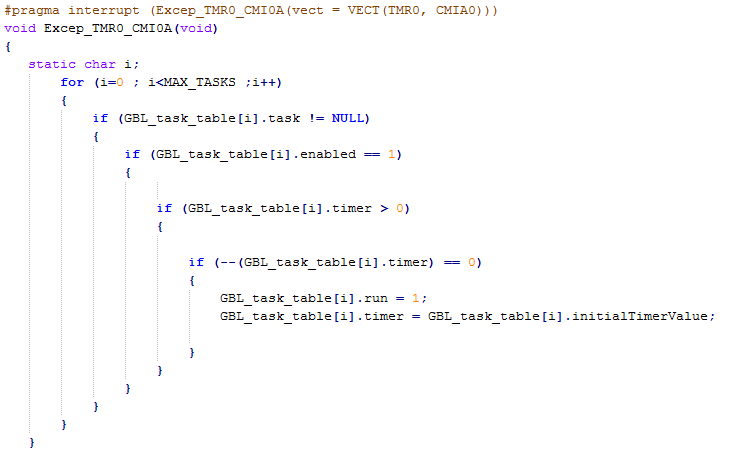


****

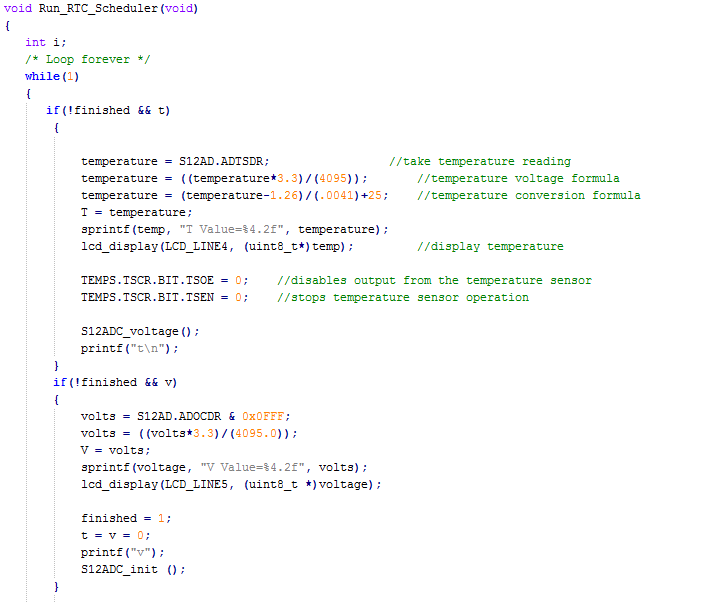
1. The RTC is initialized by using the Init\_RTC\_Scheduler function

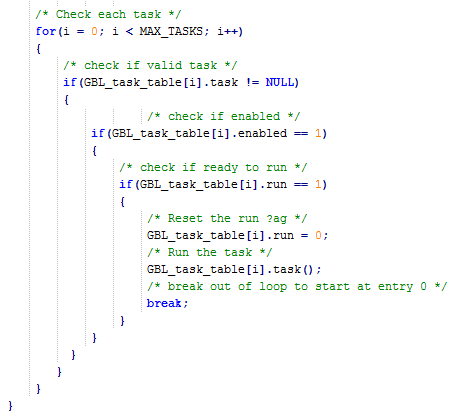


1. The interrupt for the timer is set up



1. Lastly the Run\_RTC\_Scheduler function is implemented to run the tasks added.





**Observations:**

During the seventh lab experiment the main purpose was to perform communication from a host terminal (pc) and control the client by serial commands captured by the microcontroller from the host while using interrupts and RTC to output data to the host only during prescribed times and have this all operate outside of a while loop by using a run to completion scheduler API. Building off what was already created in earlier labs the CRC value of the string sent to the host was calculated and then displayed as well on the host system with the addition of the time string created. During the previous lab the major issue was the RTC not counting in the proper time, this was fixed and the functions for everything in the main while loop were moved into separate functions to avoid timing issues. The scheduling of tasks makes the addition of new functions much easier than in previous labs. The ability to add something with its priority and timing to a function makes additions to the code much easier.

**Summary:**

Overall, the lab experiment tested our abilities to create a task scheduler and implement the proper timing and functions. The lack of working interrupts quickly became a time sink and took away precious time from getting other portions of the code to function without flaws. However we were finally able to overcome the issue and finish all of the code. This allowed us to add new tasks to the scheduler as expected.