**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. \_\_\_\_\_\_\_\_

The brightness of LED4 to LED15 is controlled 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 \_\_\_\_\_\_\_\_

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 \_\_\_\_\_\_\_\_

Commands 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 commands list ends with a semicolon \_\_\_\_\_\_\_\_

The text shown in req j is printed on the host every minute \_\_\_\_\_\_\_\_

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, and drives outputs via ISR’s.

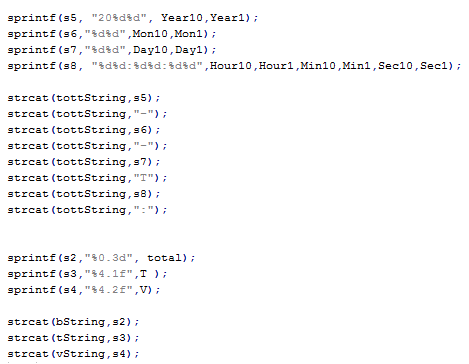
**General Steps:**

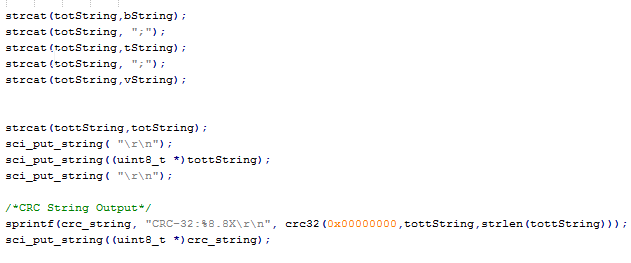
1. Go to the HEW IDE
2. Power up the Renesas RX63N by connecting it to the computer via mini-USB
3. Build upon previous lab and add UART/CRC-32 functionality and other requirements.
4. Demonstrate the working project.

**Detailed Steps:**

1. Building upon the work from last lab session (Lab 6). The ADC interrupts were created and UART communication with commands was used to communicate with a host terminal and display the values on the host terminal while controlling LED or PWM output on the board.
2. After verification of the working previous lab the ability to create a RTC, or real time clock was implemented such that the seconds, minutes, hours, day, and year.
3. Each BCD digit of the time is defined as a character so that it can be printed to a string and used later in CRC calculation.

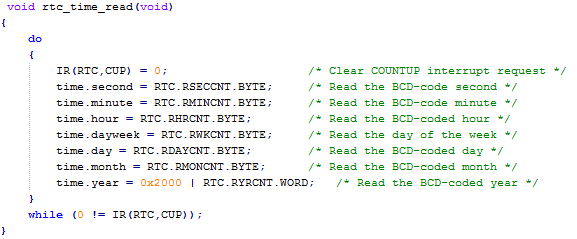


1. With the verified functionality of the RTC the next step is to calculate the correct CRC-32 value from the total string in the format of “YYYY-MM-DDThh:mm:ss:B Value=XXX;T Value=XX.X;V Value=X.XX”. 

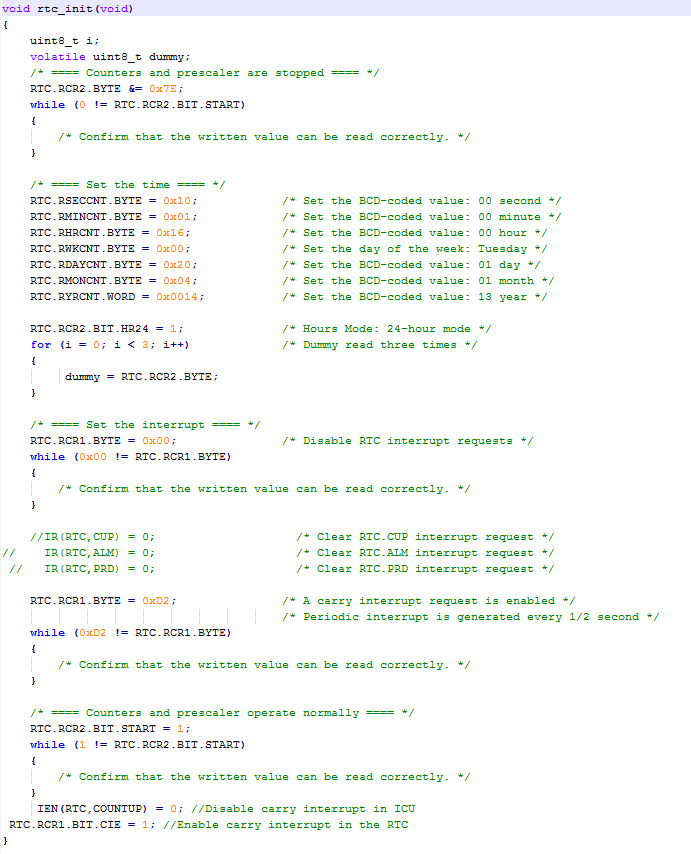


1. The output of the RTC string is checked by the CRC function and then is printed to the screen using the sci\_put\_string. This is only done after one minute via two alarms set by the interrupt initialization function. When the alarms go off the function knows to output the next output to the screen via the SW3 Interrupt.

1. The RTC is accessed by using the RTC\_Time\_Read function



1. RTC is initialized using the rtc\_init function.



**Observations:**

During the sixth 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, in our case every minute. 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 this lab the major issue is the RTC not counting in the proper time. Every “Minute” to the RTC is actually 15 seconds. When creating our code the output appeared to be incorrect as it output a new value every 4 minutes however it turns out that is correct and it is a conversion issue from BCD to decimal which causes the number to appear incorrect in the minutes place. Due to time constraints and a multitude of errors we were unable to completely finish interrupts. This issue stems from a problem in which the values can only be read from the potentiometer of the temperature sensor but not both. If more time was available a fix could be created in which the flag for one set off the conversion of the other.

**Summary:**

Overall, the lab experiment tested our abilities to create interrupt service routines and timers for components. 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. If better coding and debugging practices were followed in the previous labs we may have been able to overcome these issues.