**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 2 decreases the brightness of the LEDs \_\_\_\_\_\_\_\_

Turning Analog In potentiometer controls the brightness \_\_\_\_\_\_\_\_

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

Pressing switch SW3 transmits the text to the host PC \_\_\_\_\_\_\_\_

The 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’] \_\_\_\_\_\_\_\_

LCD should display the temperature value on line 4 as T Value=XX.X, where

XX.X is the temperature in degree Celsius. \_\_\_\_\_\_\_\_

LCD should display 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\_ \_\_\_\_\_\_\_

LCD information should be centered\_ \_\_\_\_\_\_\_

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

“Analog In” potentiometer is used to control the brightness in steps of 1 [Range: 0

– 9] \_\_\_\_\_\_\_\_

No ADC interrupts or Timers are used \_\_\_\_\_\_\_\_

UART (SCI2) is used for serial communication. \_\_\_\_\_\_\_\_

Switch 3 ISR is used to transmit the text to the host PC \_\_\_\_\_\_\_\_

The brightness value, temperature value, internal reference voltage value, and

the CRC-32 value are shown as

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. \_\_\_\_\_\_\_\_

**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, and lastly allows UART control of the board by serial commands.

**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 4). The sample UART demonstration program was used to communicate with a host terminal and display the values on the host terminal.
2. After verification of the working previous lab the ability to send commands from the host PC console to the board was added using the sci\_put\_char() function
3. With the verified functionality of the character placement onto the board the next step is to parse a command string. When the comparison of the string and the known command is equivalent the Boolean value is set true for the corresponding LED and turned on in the PWM functions defined from an earlier lab.

//Parse Commands for LED

i[num+1] = '\0';//TERMINATING CHARACTER

if(strcmp(i,"LED04=01") == 0)

{

ledstate[0]= true;

}else if(strcmp(i,"LED04=00")== 0)

{

ledstate[0] = false;

}

if(strcmp(i,"LED05=01") == 0)

{

ledstate[1]= true;

}else if(strcmp(i,"LED05=00") == 0)

{

ledstate[1] = false;

}

if(strcmp(i,"LED06=01") == 0)

{

ledstate[2]= true;

}else if(strcmp(i,"LED06=00") == 0)

{

ledstate[2] = false;

}

if(strcmp(i,"LED07=01") == 0)

{

ledstate[3]= true;

}else if(strcmp(i,"LED07=00") == 0)

{

ledstate[3] = false;

}

if(strcmp(i,"LED08=01") == 0)

{

ledstate[4]= true;

}else if(strcmp(i,"LED08=00") == 0)

{

ledstate[4] = false;

}

if(strcmp(i,"LED09=01") == 0)

{

ledstate[5]= true;

}else if(strcmp(i,"LED09=00") == 0)

{

ledstate[5] = false;

}

if(strcmp(i,"LED10=01") == 0)

{

ledstate[6]= true;

}else if(strcmp(i,"LED10=00") == 0)

{

ledstate[6] = false;

}

if(strcmp(i,"LED11=01") == 0)

{

ledstate[7]= true;

}else if(strcmp(i,"LED11=00") == 0)

{

ledstate[7] = false;

}

if(strcmp(i,"LED12=01") == 0)

{

ledstate[8]= true;

}else if(strcmp(i,"LED12=00") == 0)

{

ledstate[8] = false;

}

if(strcmp(i,"LED13=01") == 0)

{

ledstate[9]= true;

}else if(strcmp(i,"LED13=00") == 0)

{

ledstate[9] = false;

}

if(strcmp(i,"LED14=01") == 0)

{

ledstate[10]= true;

}else if(strcmp(i,"LED14=00") == 0)

{

ledstate[10] = false;

}

if(strcmp(i,"LED15=01") == 0)

{

ledstate[11]= true;

}else if(strcmp(i,"LED15=00") == 0)

{

ledstate[11] = false;

}

1. Similarly the PWM commands are parsed in a similar fashion.

/Parse Commands for PWM

if(strcmp(i,"PWM01=00") == 0)

{

dly = 00;

}else if(strcmp(i,"PWM01=01" ) == 0)

{

dly= 10;

}else if(strcmp(i,"PWM01=02") == 0)

{

dly= 20;

}else if(strcmp(i,"PWM01=03") == 0)

{

dly= 30;

}else if(strcmp(i,"PWM01=04") == 0)

{

dly= 40;

}else if(strcmp(i,"PWM01=05") == 0)

{

dly= 50;

}else if(strcmp(i,"PWM01=06") == 0)

{

dly= 60;

}else if(strcmp(i,"PWM01=07") == 0)

{

dly= 70;

}else if(strcmp(i,"PWM01=08") == 0)

{

dly= 80;

}else if(strcmp(i,"PWM01=09") == 0)

{

dly= 90;

}else if(strcmp(i,"PWM01=10") == 0)

{

dly= 100;

}

1. The commands used by the PWM are seen by the client and then processed after the semi-colon in a way that sets the dly variable created in an earlier lab to the value seen by client. This will effectively replace the buttons on the board.
2. The command string is delimited by commas and then terminated by a semi-colon.When it reaches the max size for a command the memory needs to be reallocated. This is done as shown below.

free(i);//frees i to be reallocated

num=0;//resets count

i = (char\*)calloc(10,sizeof(char));//reallocates memory

1. After using the memory the string is freed and the counter is reset to 0 to begin again.
2. If the size of the count is at the edge of the array then the allocation must be extended. In order to do this the modulus of the string is taken by 10 and compared for a result of 0.

if(num%10 ==0)

{

realloc(i,10\*sizeof(char));/\* when the counter reaches the end of the buffer reallocate more memory\*/

}

1. The multiple commands are parsed using an if statement and the strcmp function to find the delimiting command and place a semi-colon in its place to execute at the correct time.

char new\_char= sci\_get\_char(); /\* Get the char waiting in buffer. \*/

if(new\_char == ';')// Finds the end of the command string

{

for(int ctr=0;ctr<(num+1); ctr++) // sets up a loop to check for ‘,’

{

if(strcmp(i,",")==0)// compares and looks for ‘,’

{

/\*If found write one back from the character buffer into new command string\*/

cmd[0]= i[num-8];

cmd[1]= i[num-7];

cmd[2]= i[num-6];

cmd[3]= i[num-5];

cmd[4]= i[num-4];

cmd[5]= i[num-3];

cmd[6]= i[num-2];

cmd[7]= i[num-1];

cmd[8] = '\0'; // null character necessary to terminate string

}

}

1. PWM control is also implemented to control the duty cycle of the PWM between 0 and 100%.
2. Lastly the ADC for the functions of temperature and reference voltage is implemented as interrupts instead of polling like the last labs.
3. These interrupts function by performing the required calculations inside of the ISR defined by the PRAGMA and then updating a Boolean value of true for when a new value is available. The Boolean value is polled by the ADC function defined in previous labs and then updated accordingly via the DISP() function.

#pragma interrupt (inrr (vect = VECT(S12AD0, S12ADI0))

static void inrr (void)

{

void voltage();

S12ADC\_temp ();

}

**Observations:**

During the fifth 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. 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. The commands varied from LED04-LED15 on and off commands to PWM01= 0-100% commands. These commands worked fine individually without any delimitation by commas. However when implemented in a command string the code to parse the delimiters was incorrect and would crash the board. This problem led to a major waste of time trying to debug the issue and ultimately caused the rest of the lab to be unfinished due to time constraints. Ideally if everything functioned as expected the ADC would also be implemented with interrupts so that the processor did not have to constantly poll for updates.

**Summary:**

Overall, the lab experiment tested our abilities to collect, collate, and communicate data effectively through a UART connection to a host. The communication of commands from the host to a client proved more difficult than expected and led to poor time management during the debugging phase. This lab would have been more effective given that our code functioned the first time and allowing us to finish.