ECGR 4101/5101, Fall 2011: Lab 5

Voltage Regulation with a Control System.

Learning Objectives:

This lab will test your ability to apply the knowledge you have obtained from the last four labs and expand that knowledge further by introducing you to queued serial communication and a glimpse into control system fundamentals.

Note: be warned, this lab expects that you are very comfortable with the Lab1 thru Lab4 procedures for creating a HEW project from scratch, implementing the custom LCD interface, and setting up the UART. Details on these steps will be skipped in this lab as it is expected that you already know how to do this.

Additionally, this lab will not provide as much step by step detail as lab four did so be prepared to spend additional time researching datasheets, reading over your class notes, and using the age old process of trial and error. I also recommend that you get acquainted with the HEW debugger as being able to add breakpoints and variable watches is very useful when trying to figuring out why your code isn't working correctly.

General Information:

1. Review the steps provided by the lab 1 supplemental information document
2. Review the steps provided by the lab 2/3/4 Activity
3. Copy the necessary files into your new workspace and import them
4. Import the necessary mathematical and string libraries (note string.h is under math.h)
5. Build and attach the circuitry required by the lab to the RX Development board.
6. Implement the lab requirements.
7. Demonstrate your working project to the TA, and turn in a lab report.

Prelab Activity:

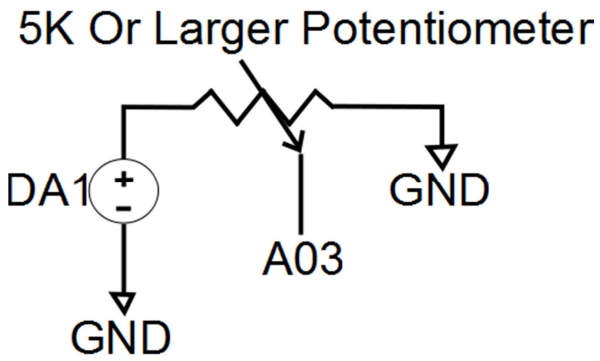
You may use the PCs in Woodward 203 or your own PC to do this lab experiment. If you want to work on lab assignments on your own PC, then you will need to load the necessary tools on your PC in order to perform this exercise.

In this lab you will need to create a new project called lab 5 that requires the custom LCD functions, CMT and TMR Timers, ADC and UART functions from your prior labs. You will also need to review your class notes, the Renesas API document, and possibly look at the available sample applications throughout this lab.

In this lab you will need to attach some external circuitry to your RX Development board. I recommend you obtain headers and solder them to your RX development board.

Before you try and solder the headers onto the board, I highly recommend that you watch a few soldering tutorial videos on YouTube as it is very important that you don't try to melt solder onto the

end of the iron tip and then try to solder the component since the flux will evaporate out of the solder and oxidization will occur which can make getting the solder to adhere very difficult.



You will also need to obtain a potentiometer that is somewhere between 5k to 20k ohm in size.

You will need to look at the RX development kit schematic and figure out where the DA1 and A03 pins are located on the external headers.

If you purchased the external module for the RX development kit you can solder the potentiometer to the module and simply plug your module into the RX board otherwise you will need to figure out a way to safely connect the Potentiometer to the RX development board.

Additionally, you will also need to download

- queue.c
- queue.h

from the Embedded website and import them into your project.

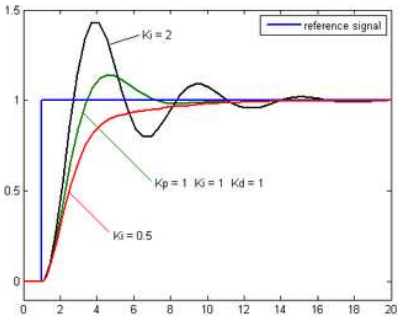
Some Helpful Advice:

Start on this ASAP!

Prelab Questions:

1. What type of package does the RX62N processor have on the RX Development Board
2. What is DA1 and how would you set its DDR direction
3. What is AN3 and how would you set its DDR direction
4. What does the command IOPORT.PFFSCI.BIT.SCI2S Do, and after looking at the Board schematic why is it needed
5. In the file iodefine_rpdl.h find the macro MSTP() and determine what this command does
6. From 5, what does MSTP(SCI2) loosely evaluate to as prior to being complied and what is the purpose of this command

Laboratory Assignment Overview:



Your First Digital Control System

So far it's been a long, but hopefully very educational journey into embedded systems and now that you have the basics down it is time to refresh your knowledge thru repetition and introduce a few additional topics in the processes.

In the last lab we looked into how to implement serial communication without relying on the RX API and in this lab we will discover how to implement a data queue with serial to optimize our data transmission.

So let's begin!

Laboratory Assignment:

1. Do the pre-lab activity described above.
2. Once you have your new project up and running preform a quick test to ensure that your project is working as intended.
3. Initialize your LCD screen and test that it is working correctly by writing some text to the screen.
4. Initialize your Screen buffer LCD custom font code and test that it is working correctly by drawing a few pixels on the screen.
5. Create a 100ms timer that will set a LCD global draw flag and add code to your main loop to render the Screen buffer to the LCD screen when the draw flag is set. (note be sure to clear your flag after you render the screen in the main loop)
6. Create a global integer array called window that is the size of the number of horizontal pixels on the LCD screen.
7. Create a function that will iterate thru each window element from above and create an algorithm to translate the value of the integer into vertical pixel coordinates and then plot all the values on the LCD screen.

   Note: you only have to worry about vales from 0 to 4095

   Note: a value of 0 will appear at the bottom of the screen while a value of 4095 will appear at the top of the screen.

When your function is complete it should plot a line across the LCD screen since all of the values should be initialized to 0

8. Use the R_ADC_12_Create and R_TMR_CreatePeriodic commands to sample the ADC (A03) channel every 100ms.

   Note: you will need to modify your prior lab code to sample A03 rather than PDL_ADC_12_CHANNEL_4, you will need to figure out what channel A03 is on and simply change the number after PDL_ADC_12_CHANNEL_.

   Note: in your ADC function that gets defined in R_ADC_12_Create be sure you change the read ADC value from adc_results[4] to the correct index for ADC3

9. In your ADC sample done function create a global ADC Done flag and add code to your main loop to detect when a new ADC sample is available.

   Note: you might also need to create a global variable to hold the latest ADC A03 value.

10. In the main loop in the ADC Done code you created above, create a algorithm that will shift the window matrix to the right by 1 element (it should drop the last element in the index) and set the first index equal to the new ADC value.

    Note: be sure you reset the ADC done flag in the main loop.

11. Using the RX62N Datasheet or examples provided by class resources, write a function that will initialize DA0 as a 10 bit DAC and set the initial output voltage to 0.

    Note: it is recommended that you use a digital voltage meter to ensure that your DAC is working correctly prior to continuing.

    Note: before you try and measure the output voltage you should connect the potentiometer as shown in the pre-lab otherwise your output voltage could be wrong because of internal impedances.

12. At this point if everything is working correctly, you should be able to run the code and observe changes in potentiometer on the LCD screen like an oscilloscope.

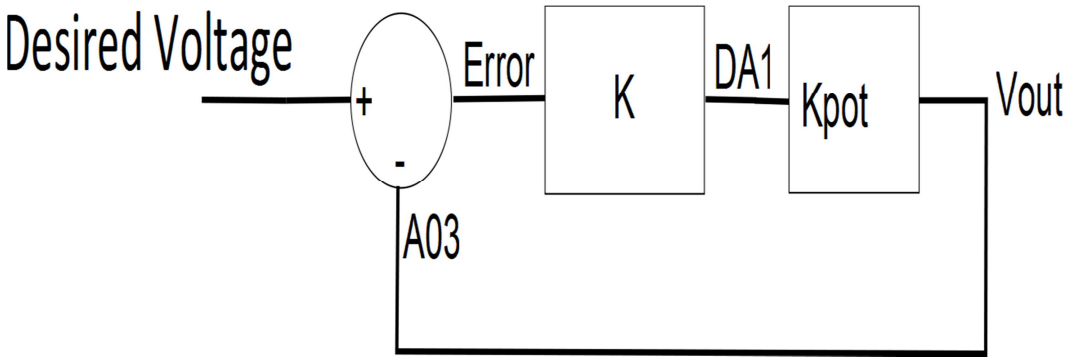
13. Create a global variable to hold the last DAC value, and write a function that will set the DAC to a passed value along with set the last DAC global variable.

14. Create another timer (be careful to select a unused timer) that runs at a rate of 2s.

Note: you cannot create a timer that runs at a rate of 2s using the API call, you have to define a new time at a rate of 500ms and then inside the function that is called, increment a global counter that after 4 function calls sets a global Update Control Flag before resetting the counter.

In the main loop add code that will check for the Update Control Flag and clear the flag afterwards.

15. At this point we are ready to implement the following digital control system



Without exploring the wonders of control theory, which is a worthwhile subject that I recommend you study.

Conceptually, a desired voltage value which by the end of the lab will be provided by the user is compared to the current voltage value observed by the ADC A03 pin.

The difference between the desired and the current voltage value is called the error signal and it represents the amount of change that is required to make the current voltage of A03 equal to the desired voltage.

K represents a gain linear gain, and in this controlled system there are two gains. One gain is an internal gain that the user can change while Kpot represents an external fractional gain that is changed by the potentiometer

Both of these gains scale the error signal and the scaled error signal becomes the new current voltage seen by the microcontroller when A03 takes another sample.

This process repeats over and over until the error becomes zero.

Mathematical theory can be used to predict how a control systems will respond and the optimal values of K to make the system reach the desired voltage in the quickest amount of time.

Mathematics can also be used to determine if the control system is stable, and when it will become unstable.
(Note this model is simplified digital controls adds extra terms that are not shown)

16. In order to implement the control system above in the main loop under the Update Control Flag you created you will need to perform the following steps

    a. Convert the newest ADC value located in window element 1 into a floating point 2^12 voltage value assuming a maximum voltage of 3.3V . Note your homework has examples of how to do this.
    b. Create a global floating point desired voltage variable that will hold the desired voltage (.8 is a good initial value for this variable)
    c. Subtract the desired voltage from the converted ADC value from step a (this is your analog error signal)
    d. Create a global floating point variable called K to hold the internal gain
    e. Multiply the analog error signal found above by the variable K (note K should never be 0, make this 1 to begin with)
    f. Convert your analog error to a 2^10 DAC value to a digital error value assuming a maximum voltage of 3.3V Note your homework has examples of how to do this
    g. Because the error signal is a positive or negative number and our dac only takes positive numbers we need to adjust our last DAC value by the error amount to do this simply add the last DAC value you recorded earlier by the error and set the result equal to the DAC error
    h. Set the DAC to the DAC error value
    i. Reset your Update Control Flag and this code will run again after 2 seconds have passed

17. Compile and test your code, you should observe the waveform on the LCD screen changing and settling on the desired voltage value after a period of time. if you measure the voltage at A03 you should see the voltage is very close to the desired value.

    Note: you might have to adjust your potentiometer to somewhere in the middle to avoid rail conditions since the DAC can produce more than 2.3 volts under 5k load.

18. At this point you are ready to add UART support. Create a function that will initialize the UART to the following specifications.

Baud	19200
Data Bits	8
Parity	Even
Stop Bits	1
Flow Control	None (only worry about this when you use HyperTerminal on the PC)

Note: there is a RX API for the UART, However, I would like you to explore how to do this without using functions from "r_pdl_sci.h"

Note: if you have PORT1 anywhere in your UART code you are wrong! Answer the prelab questions and come back to this step.

Note: you will need to also add

```
IPR(SCI2,RXI2) = 1; // Set priority level of interrupt
IEN(SCI2,RXI2) = 1; // Enable RX ISR

IPR(SCI2,TXI2) = 1; // Set priority level of interrupt
IEN(SCI2,TXI2) = 1; // Enable TX ISR
```

To enable RX and TX interrupts at the end of the setup code

19. Open up interrupt_sci.c and find the following functions
    void Interrupt_SCI2_RXI2(void)
    void Interrupt_SCI2_TXI2(void)

    you will need to gut both functions, this is where you will be writing your own UART code

20. Include the queue class provided and read over the queue example in the embedded book.

21. To provide an illustration of how queues work consider the following examples

```
void Interrupt_SCI2_TXI2(void){
    if(!Queue_Empty(&TX)){
            if(SCI2.SSR.BIT.TDRE==1){
                    SCI2.TDR=Queue_Dequeue(&TX);
            }
    }
    else{
            TxFlag=0;
    }
}
```

And

```
void Interrupt_SCI2_RXI2(void){
        char data=0;
        data=SCI2.RDR;
        Queue_Enqueue(&RX, data);
        if(data=='\n')    {
                RxFlag++;
        }
}
```

22. From the examples provided above create the required code for your Interrupt_SCI2_TXI2 and Interrupt_SCI2_RXI2 function in interrupt_sci.c
23. Consider the following function (found in main)

```
void UART_TX_START(){
        if(TxFlag==0)    {
                SCI2.TDR=Queue_Dequeue(&TX);
                TxFlag=1;
        }
}
```

Note the idea behind this function is because Interrupt_SCI2_TXI2 will only run after a byte of data is sent over the UART, the user must write 1 byte of data to start the interrupt process which will loop automatically until all data has been removed from the Queue.
Note: the TxFlag is determines if the UART is in the middle of a transmission, and if it is then Interrupt_SCI2_TXI2 is already running.

24. In the main loop write a condition that only runs when the value of RxFlag is greater than zero.

Note: this implies that the user has received data that was terminated with a \n character
Note: RxFlag contains the number of \n character separated strings in the queue
Note: if hyperterm adds \r to the string you will need to modify Interrupt_SCI2_RXI2 to ignore \n

25. Consider the following Code

```
while(RxFlag>0){
        index=0;
        while(!Queue_Empty(&RX)){
                check=Queue_Dequeue(&RX);
                if(check=='\n')   {
                        RxFlag--;
                        break;
                }
                CKSTR[index]=check;
                index++;
        }
        CKSTR[index]=0;
        if(strcmp(CKSTR,"Hello")==0) {
                // The PC Sent Hello over RS232 and you found it
                // Reply back to the User
                Queue_Enqueue_String(&TX,"CMD Found\n");
                UART_TX_START();
        }
}
```

26. Based on the example code provided above create a serial data processor that will perform a desired action based on the users input from hyper term.

Command	Action
Set D:#.##	Sets the desired value to a floating point value specified by the user (Note # is a ascii value between '0' to '9') (Note you will need to write a function to convert ASCII to DEC)
Set K:###	Sets the internal gain K to a integer value specified by the user

(Note) strcmp returns 0 if a match was found, and requires string.h to be included like math.h was

27. Bring the new board to the lab TA and demonstrate the new code (without the HEW application running). When the TA checks your board, he will also take your lab report. You will not need to include a printout or soft copy all of the code – just "snippets".

Lab Report:

1	LCD Plots Window correctly	
2	Window Reflects A03 Value	
3	DA1 works and can change voltage	
4	Circuit was constructed and connected correctly	
5	Control Loop was implemented correctly	
6	UART works using queues and interrupts	
7	System can change parameters from UART input from the user	

Include in your lab report observations and procedure like the following:

The general learning objectives of this lab were . . .

The general steps needed to complete this lab were . . .

Some detailed steps to complete this lab were . . . .

1. *Step one*
2. *Step Two*
3. *. . . .*

Code generated or modified to complete this lab...

No need to include all the files for the lab. Just include the modified code.

Some important observations while completing/testing this lab were . . .

In this lab we learned . . . .