Real Time Operating Systems
EGC 451
Final Project

Real Time Oscilloscope/Heart Rate Monitor

Team Members:	**Contributions:**
Mitchell Wagner	Theory, Code, Implementation, Write Up
Paul Boston	Theory, Code, Implementation, Write Up

Professor: Mike Otis

Abstract:

Working with microprocessors can be tricky especially if you are trying to handle multiple functions at the same time. Unless you have multiprocessors to handle this, you need to be able to run multiple tasks at once. Real time operating systems (RTOS) fix this problem. RTOS can handle multiple tasks, interrupts and other systems functions all at once while it determines what the best thing to handle next is. Micrium is a great and freeRTOS (non-commercial) it gives you amazing expandability and advancement over normal c code by enabling the programming to write intense code. Sometimes this code can be used to create amazing devices such a an Oscilloscope, that we have created here.

Table of Contents

1. Introduction:

Real Time Operating Systems (RTOS) can enable a programmer to write some really amazing code; but before this we first had to write an oscilloscope using regular c code. This code would then be used to create our newer oscilloscope that was embedded into an RTOS. The RTOS would not only make the oscilloscope run better, but also faster enabling us to capture higher and lower frequencies.

The oscilloscope is made up of a couple of key components: the display on the LCD and the ADC conversion, and calculations. In order for this to work correctly our ADC had to read in the value convert it into a voltage and then display it as a waveform on our display. This had to be a moving waveform, therefore this was more complicated. These values were also sent to the RS232 port so that they could be sent to a computer for further analysis. Once implemented on an RTOS, the oscilloscope needed to be converted to be used as a Heart Rate Monitor. The HRM, had a couple of problems on its own making a heart beat digitally readable it a tough job. But was successful in simulating on an actual oscilloscope.

2. Theory:

The object of this system is accurately receive data from the analog port on the Renesas RX63N board, and use that data to create an oscilloscope. In order to do this we first had to figure out how to use micrium, we did this through learning and doing the six labs that lead up to this implementation. Micrium has a lot of advantages that we wanted to use to our advantage. In addition to micrium, the Analog port has to be configured to read analog data, but this has to happen at a certain rate. The display was also a tricky topic, the display is mainly configured to only write characters and needed to be reconfigured to write pixel by pixel.

2.1. RTOS:

The real time operating system (RTOS) is perfect for this type of project, because of the fact that the RTOS can handle so many different tasks at once. While learning how to use the RTOS, we learned how to use different tasks at the same time, while setting priorities, controlling ISRs, setting up semaphores and mutexes as well as sending messages. Learning all of these functions in the six labs aided us in designing our system. Because of the RTOS we wrote a very well written oscilloscope that can handle new information and work much quicker that our non-RTOS version.

2.2. ADC:

The analog to digital converter (ADC), is how we took in our data from the waveform generator. The ADC port is a 12 bit register therefore our data only had a range of 0-4095 in volts that is 0V-3.3V. The ADC values had to also follow the nyquist theory that states we

needed our sampling frequency has to be more than double the input frequency. If you sample too fast you will get double waves and too slow and you won't get a wave at all.

2.3. Display:

To display the waveform we needed to be able to display bit by bit, normally the lcd screen only lets you display characters; therefore we needed to edit some settings and files to get things to work.

The LCD screen is 8 characters high and 12 characters long this give us a character count of 96. In each character there is 64 pixels giving us a total pixel count of 6144. Since we also wanted to display other information on the display we decided to cut the waveform window down to a screen area for 4 characters high and 12 characters long, which can be seen in Table 1. This gives us a pixel area of 3072, but in our oscilloscope there will only be one dark pixel per vertical line, each vertical line will represent one ADC value. This means that there will only be 96 ADC values on a single screen or window. At the end of displaying the 96 values the screen should clear and a new 96 values should be displayed again.

Table 1: Display for the Oscillating Wave System

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Char 4	8x8	8x8	8x8	8x8	8x8	8x8	8x8	8x8	8x8	8x8	8x8	8x8
Char 3	8x8											
Char 2	8x8											
Char 1	8x8											

3. Design and Implementation:

In order to actually create our oscilloscope, we first needed to learn how to use the Renesas board, this was achieved in creating our train system. That project taught us how to use the ADC, switches, UART and LCD screen. Next we used our knowledge of microprocessors and c code to write a program that created a non-RTOS oscilloscope. Finally to help learn about RTOS systems we did the six RTOS labs with micrium. This helped us understand how micrium works and how to write programs in it. Using this knowledge we converted our non-RTOS oscilloscope into micrium using the micrium BSP Renesas functions. After that worked we added in all of the micrium advantages such as ISRs, multitasking, Semaphores and mutexes.

3.1. Non-RTOS:

To first start building this Oscilloscope we needed to first figure out how to make the system display a waveform in real time. In order to do that we needed to make the display work bit by bit this required editing a number of files to accept changes in the bitmap, so that we could send one pixel at a time to the screen. In addition to this we also needed to figure out how to make the ADC read from another pin other than the POT (potentiometer) which is by default enabled and the primary ADC input. Once we figured these two things out, we decided to add in some extras, such as an amplitude display and a frequency display. Once these add-ons were completed, we had a fully functioning oscilloscope which could display waves of any type from ~0Hz to 150Hz.

3.1.1. ADC:

In order to read in our values we needed to use the analog to digital signal converter (ADC). This ADC is a 12 bit converter which means our output could be a number anywhere from 0 to 4095. In order to use the ADC we needed to use some functions that the HEW (High-performance Embedded Workbench) uses. The function, S12ADC_init(); Sets up the ADC converter and initializes it; by default it is set up uses the POT as its input. The Function S12ADC_start(); starts the ADC conversion, in order to figure out when the conversion is over the command while(false==S12ADC_conversion_complete()){} is needed this makes sure that the program will not continue until the conversion is complete. This process can be repeated as long as necessary to get all the values the user needs. In order to make the ADC read from an external input, we needed to change the S12ADC_init() function. We needed to set the pin as an input pin, select the input channel on the ADC converter and set the read register accordingly.

```
PORT4.PODR.BIT.B1 = 0;    /* Clear I/O pin data register to low output. */
PORT4.PDR.BIT.B1  = 0;    /* Set I/O pin direction to input. */
PORT4.PMR.BIT.B1  = 0;    /* First set I/O pin mode register to GPIO mode. */
/* ADANS 0: A/D Channel Select Register 0
b15:b0  ANS0: Selects analog inputs of the channels AN 000 to AN015 that are subjected to A/D
conversion*/
S12AD.ADANS0.WORD = 0x0002; /* Read AN001, which is connected to the Pin 10 */
adc_result = S12AD.ADDR1;       /* Read the result register for AN1 */
```

3.1.2. Calculations:

For this project we needed to then take our ADC values and run some calculations on them so that we could find out the amplitude and the frequency. First we had to take our ADC value and convert it into a voltage. The voltage range is 0 to 3.3V, Therefore knowing this you can do this conversion:

```
adc_count=adc_count/4095;
float adccount=adc_count*3.3;
```

Now the value adccount is a float value that contains our voltage. In order to calculate our amplitude we need to figure out which is the highest voltage and the lowest voltage in a certain set of data. These high and low numbers would reset after 96 values have been read and the entire screen has been displayed. Once the max and min values have been found, the amplitude can be found by taking max- min then divided by two, as seen in the code below:

```
if(adccount>maxVolt)
{
        maxVolt=adccount;
}
if(adccount<minVolt)
{
        minVolt=adccount;
}
amplitude=(maxVolt-minVolt)/2;
```

To calculate frequency we needed to see when a single point of the wave passes over the same voltage again. This will happen 3 times in the wave, once in the beginning, on the second half of the wave and the end of the wave. This was accounted for by taking the time that passes from the first time it hits to the second time it hits and multiply the time by two. We read the time by running a watchdog timer. This timer will tell us how long the wave has taken. We then use that value to find the frequency f=1/T. This is the code that we used to find that time:

```
if((adccount>=(maxVolt-.01))&&(adccount<=maxVolt)) freqc=0;  else freqc=1;
               if(freqc==0) wdt_feed_watchdog();
               if(freqc==1){
                //watchdog start
                if(WDT.WDTSR.BIT.CNTVAL<mincnt)  mincnt=WDT.WDTSR.BIT.CNTVAL;

               }
```

3.1.3. Display:

The display was completed through a very complex and non-direct method. The main component that was desired from the display, is the creation of a waveform. In order to create an oscillating wave, each pixel would need to be controlled over the LCD screen. For the Renesas board each character displayed is has 8x8 pixel array, and there is potentially 12 characters able to be displayed in one row. To get the best looking signal there will be 4 of these rows on top of each other controlled and reserved for the oscillating waveforms.

There were many different paths that we attempted to control the LCD, but were unsuccessful in controlling one pixel. So the route that was used is complex. First we decided to follow the direction it was taken to be able to display a single character on the LCD. This lead us through the packages of the LCD to a C file “font_8x8.c”. This file created and controlled each pixel for the standard ASCII characters from 0-255 in an array format. The method we decided on was to be able to choose one of these arrays and continually update the values within the array to manipulate each pixel. We choose the character 126 (~), because this character was not seen as one that would be displayed at any time. Then to be able to change this character another method needed to be implemented and called from this file. We created a change(array){} method that would input an array, of the same data type and size, and the array would reset the entire array for the character 126 (~). This method of manipulating one character was called by the C file “lcd.c”.

The file “lcd.c” was the code that would perform the display functionality. Within this file there were four different arrays that were created to save the pixel location, so that the wave would be large enough to see the change in frequency. There are 2 main loops within the file to go through all 3072 different pixels. The first loop just loops between 0-11 to get each of the 12 sections of the 8x8 pixel. Then the four different 8x8 arrays are created and the second loop is called. This loop will last for 8 cycles to be able to associate each column of the arrays with a number that is retrieve from the ADC.

Then the value inputted from the ADC will be split into first 4 different ranges, to be able to determine which array is this pixel number needs to be placed: #1(0-0.825), #2(0.825-1.65), #3(1.65-2.475), #4(2.475-3.3). This can be seen in Table 2.

Table 2: Arrays 1-4 ADC Value Separations

Array 4 Range	3.3
	2.475
Array 3 Range	2.475
	1.65
Array 2 Range	1.65
	0.825
Array 1 Range	0.825
	0

An example of the range that array 1 will be placed between in seen in the Table 3. To place in one column there are 8x4=32 different pixels located in one column. Each of the pixels will be on if the ADC number is within its range. Each pixel is incremented by a value of 0.103125. This number comes from the desire that our max value will be 3.3 and 3.3 / 4 = 0.825

for the 4 different arrays. Then 0.825 / 8 = 0.103125 for each 8 pixels in one column in one array. Table 3 shows the increments that a sample of one character array would go through.

Table 3: Array 1 ADC Number Ranges

Array 1 Number Range	0.825
	0.721875
	0.61875
	0.515625
	0.4125
	0.309375
	0.20625
	0.103125
	0

This was structured through splitting up each column of the display screen of the LCD. This was done to be able to control each of the pixels located within that column. After the second loop that was performed 8 times was complete all 8 character arrays would be sent to the LCD to be changed and displayed to get the location of the wave.

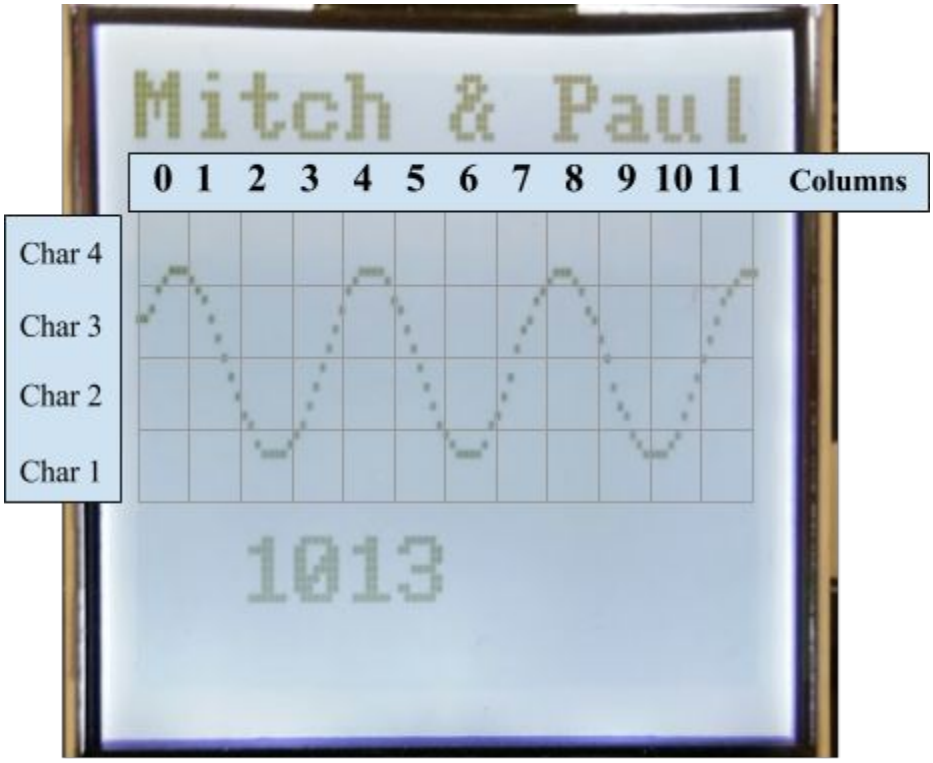


Figure 1: Display on LCD

Figure 1 helps to show how the 4 character arrays are listed with the 12 different columns to display the waveform for the actual oscilloscope. Each of the Columns have 8 inner columns located and once the ADC inputs the number it will decide which of the four Char blocks it is located and save the value there.

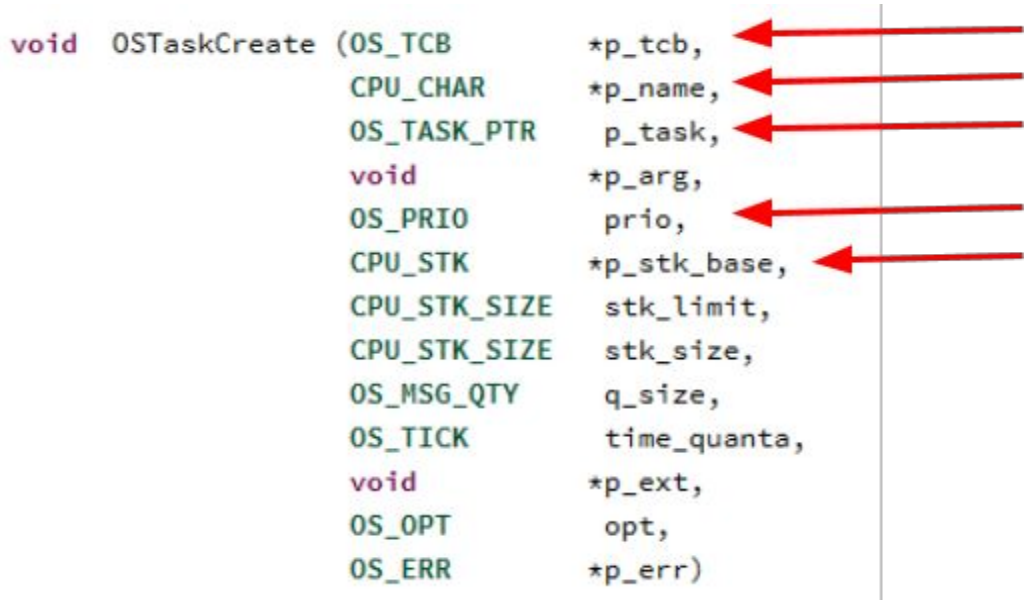
3.2. RTOS:

To be able to understand and create real time operating system (RTOS) component for our Oscilloscope we went through 6 tutorial labs. These labs went through the setup of multi task function, semaphores, and mutexes. All of which helped us in our final project design. These labs were performed on IAR Embedded Workbench which was also where the oscilloscope was added from HEW.

3.2.1. Labs 1-6:

Lab 1 and 2 both were simple getting used to the IAR platform and software. Lab 1 was performed to help understand how to build and download a program built in IAR to our Renesas board. Lab 2 then went into detail about how to create a task function.

Creating multiple tasks or multitasking is useful when developing code. This function is to be able to have multiple operation get preformed at the same time. In Order to make another task the function OSTaskCreate() is called. In order to create new tasks it is not advised to create these multiple tasks all in the main instead it is advised to create multiple tasks within the first task itself. There are several parameters within an OSTaskCreate() function as seen if the code below. When creating the new task the five main parts are: p_tcb, p_name, p_task, prio, and p_stk_base.



The ‘p_tcb’ argument is a pointer to the Task Control Block (TCB) which is of data type OS_TCB. This a global variable that is created above the main.

The argument ‘p_name’ is then a pointer to the ASCII string that will assign a name to the task, this is of data type CPU_CHAR.

The argument ‘p_task’ is then the pointer to the task itself, which is a function that is defined above the main and has a particular performed task.

The argument ‘prio’ is used to determine the priority of the task, this is build to determine the importance of the different tasks with the lower number of higher priority. The data type for this function of is of type OS_PRIO.

The final argument that may need some manipulation is the ‘p_stk_base’ with is a pointer to the task’s stack base address. This parameter is used to help determine the amount of space needed within memory for the task.

Lab 2 was then built to make another task that would operate at the same time as the base code that would cycle through to display all the LEDs on and then off. The second task that was

build was then able to simultaneously display and control the LCD by displaying 2 different groups of texts that would be delayed between each other.

Lab 3 and 4 both dealt with the creating an interrupt task function that was controlled by the ADC. Similar to Lab 2 we still needed to create a new task as well as an ISR function. A semaphore was then created and used in Lab 4 by calling the OSSemCreate() function. In order to use the ISR functions OSIntEnter() and OSIntExit() were needed to notify the beginning and end of an ISR. For Lab 3 S12AD.ADDR2 to be able to access the ADC information. Since Lab 4 was dealing with semaphore this meant the functions OSSemPost() and OSSemPend() were both used. OSSemPend() was used as a wait command to wait until the semaphore has been released with the call of OSSemPost() that is located with the ISR.

Lab 5 then deals with the creation of multiple tasks with the use of Mutex. This is useful because there is often multiple function that are desired to be created and working at the same time and in Lab 5 the buttons are being called to be able to perform different addition operations. The method for creating these multitasks are the same as in Lab 2 by creating two OSTaskCreate() calls in the first task. Also to create the Mutex function OSMutexCreate() is called within the main function. Then similar to semaphores there are two methods used in mutexes OSMutexPend() and OSMutexPost(). To be able to call the switches BSP_SwsRd(#) is used to determine which switch is being pressed at a time.

Lab 6 similar to lab 5 involved the switches by the location of the values were instead of displaying on the LCD would be stored within a queue. The methods to perform this was again OSQPost() and OSQPend(). To be able to access a queue you need to be able to create a queue by the method OSQCreate().

3.2.2. Micrium:

In order to convert the Oscilloscope to a RTOS we needed and operating system. Since we have already done the Micrium labs, micrium was the obvious choice. Micrium allows us to run multiple tasks at once, which allows us to multitask, performing operations quicker and better. In order to convert we first needed to convert to using IAR. So we first load our project into IAR and converted it from HEW using a tool in IAR used to convert project from HEW to IAR, this was very helpful. After that we switched all of our functions in our project with the micrium based functions, for the LCD, UART, ADC and timing.

3.2.3. ISR

An ISR is an Interrupt Service Routine. It is used to stop whatever is happening to do a certain task. In our case we are using an ISR to detect when our ADC is done converting. When it is done the ISR stop the current tasks and saves the ADC value to an array slot. This happens again and again until that array is filled. Once that array is filled it does not get cleared and overwritten until the display is done displaying all of them. The ADC receives the values much faster than we display the values.

3.2.4. Tasks

In our system we have two tasks, one is used to display our values and the other one is used to demonstrate the ability of micrium. The display uses the same code from the non-RTOS version of the display functions except now it is located in the top level main file and the LCD file only has the write functions that write that single changed character to the screen. The other task is an led ring that never stops, to just show how complex the system can get.

3.3. Heart Rate Monitor:

The monitor required a lot of hardware and more electronic knowledge. This was performed to be able to see the actual heart rate of a person. The total idea was to be able to see the heart rate on our own oscilloscope that we created, but it was determined that since our scaling factor was involved it was possible that we just couldn't see the small incremental pulses that a real heart rate would have.

Heart rate monitor was done to be able to see the different pulses that the patient's heart. Each of the pulses would be at a different frequency and wanted to be observed. Although the first the new digital oscilloscopes were attempted to be used it was determined that since there were simily problems involved the old school oscilloscope was taken to observe the rate the signal was operating at.

3.3.1. Hardware

The heart rate had many different components that were needed in order to create a view a actual heart rate. The first material that was needed were some pads. These pads are similar to those that are used in traditional sensor in depicting the electrical pulse.

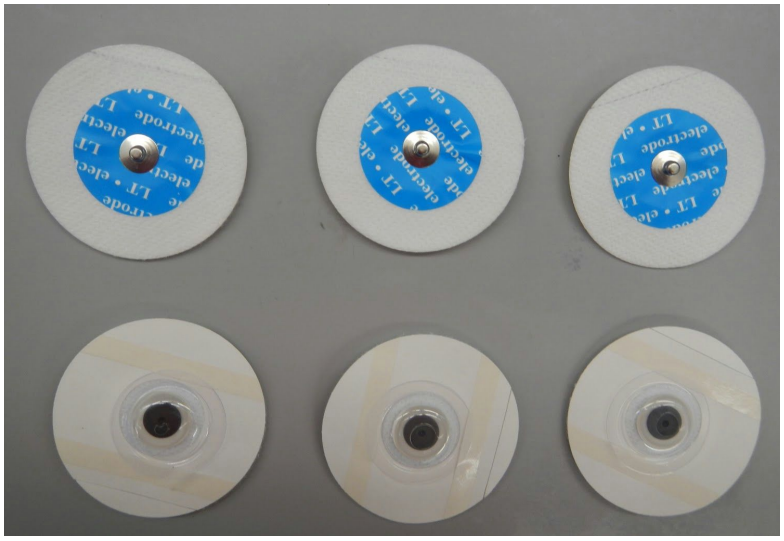


Figure 2: Heart Rate Patches

Also there is required some amplification of the signal from the person so that it could be picked up by the oscilloscope. To create this amplified signal an op amp was used. To create this op-amp 2 resistors were needed to be attached like the diagram below. The chip that was used to get the op-amp was a LM324N. The resistors that we made use of the create the op-amp were 1MOhms and 1KOhms were both used to get a gain of 1000. This was solved by the below equation and circuit of an inverting op-amp.

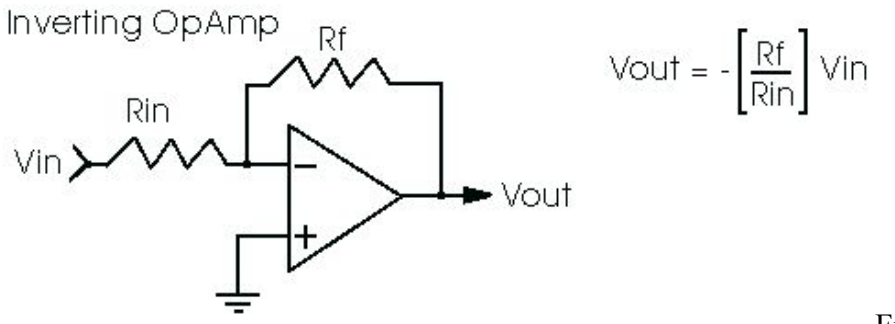


Figure 3: Inverting Op Amp

The chip LM324 was used to create the op-amp. This chip was used to create the amplification needed for the heart rate monitor and can be seen in Figure 4.

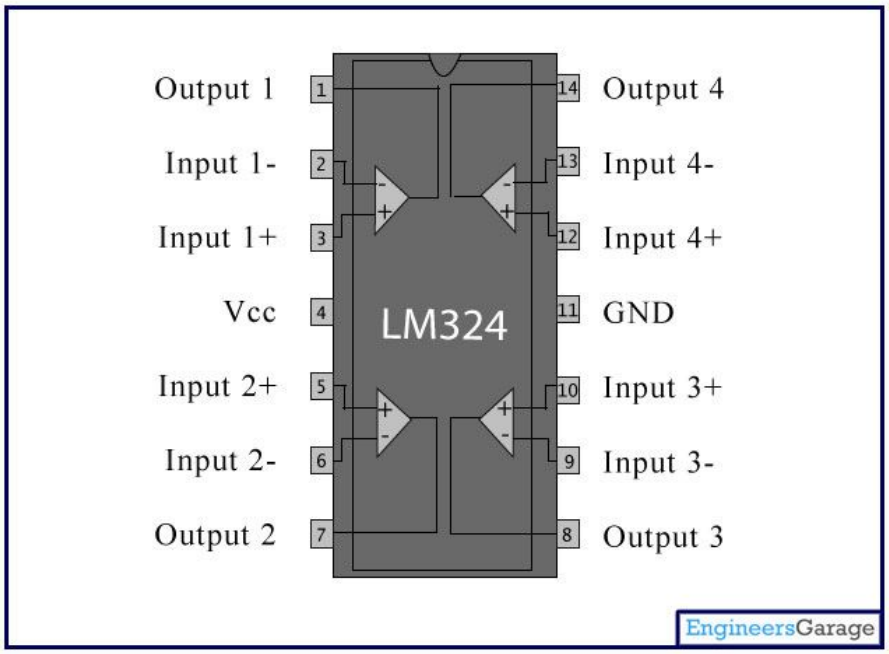


Figure 4: LM324 Chip

To further the implementation of the heart rate it was observed that there was noise that was being picked up through the circuit that was built. First there system was getting interference from the different devices around the system. It was determined that there would need to be some sort of voltage buffer to block out the 60Hz signal that was continually interfering with the signal.

The heart rate pads that were used as the sensors to pick up the heart rate was bought from China. The first issue involved with these was the location these pads needed to be placed these pads. We observed through many different online diagrams that had different placements for these pads..

The first test that was done was having 3 different leads as seen in the Figure 5 Try 1 has three circles. The blue represented the input signal with the black and red begin ground and our reference signal. This test was able done originally without any amplification and it seemed that we were able to see some signal.

The next try was just to test at the wrists just having a input and ground signal this result seemed to get some signal but not very reliable and was determined that there was just noise being picked up by nearby devices and signals. This test was on Figure 5, Try 2. We did try different variations of this test by having one on each wrist of just on one and the side of the body..

The final test that was proven to be the best success. We just had 2 pads directly over the heart and were able to see on the oscilloscope the actual heart rate. This is observed in the Final Try 3 located in Figure 5.

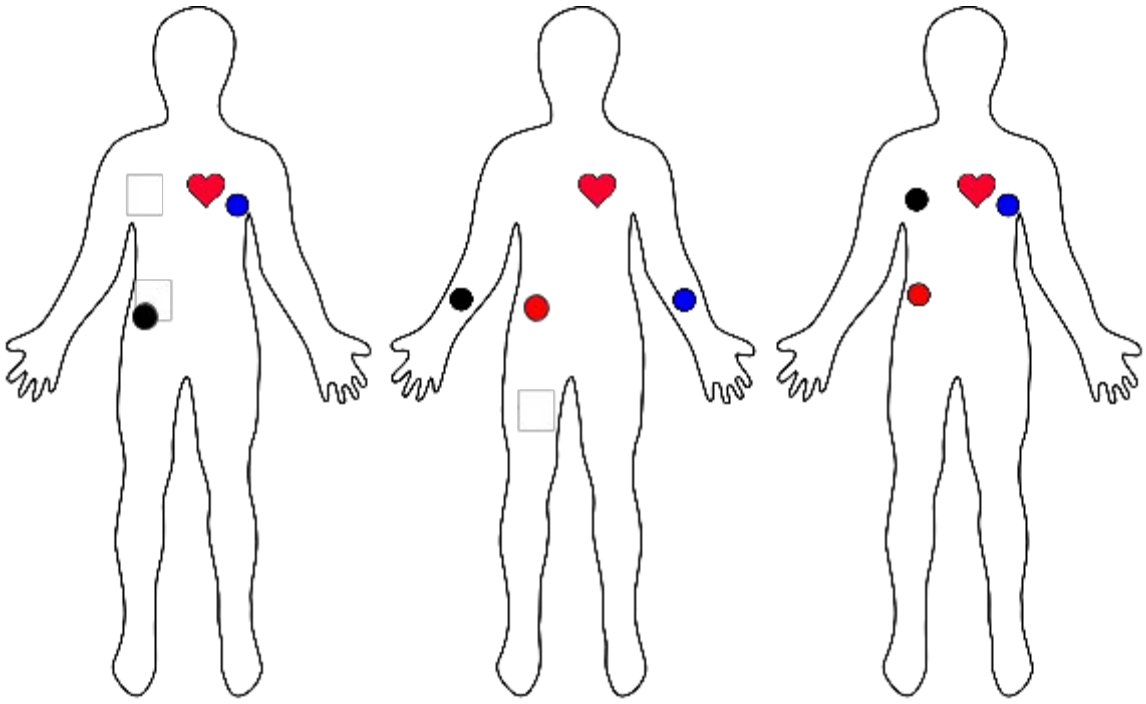


Figure 5: Heart Rate Patch Placement

3.3.2. Experimentation

The experimentation was done different methods with the best result when we had 2 pad directly over our patient's heart (And they died shortly after). In all seriousness we tried a multitude of methods, as shown on the previous page. Between all the noise we were having and outside sources of interference, it was very hard to try and get a good reading. A reading was finally achieved with the final method.

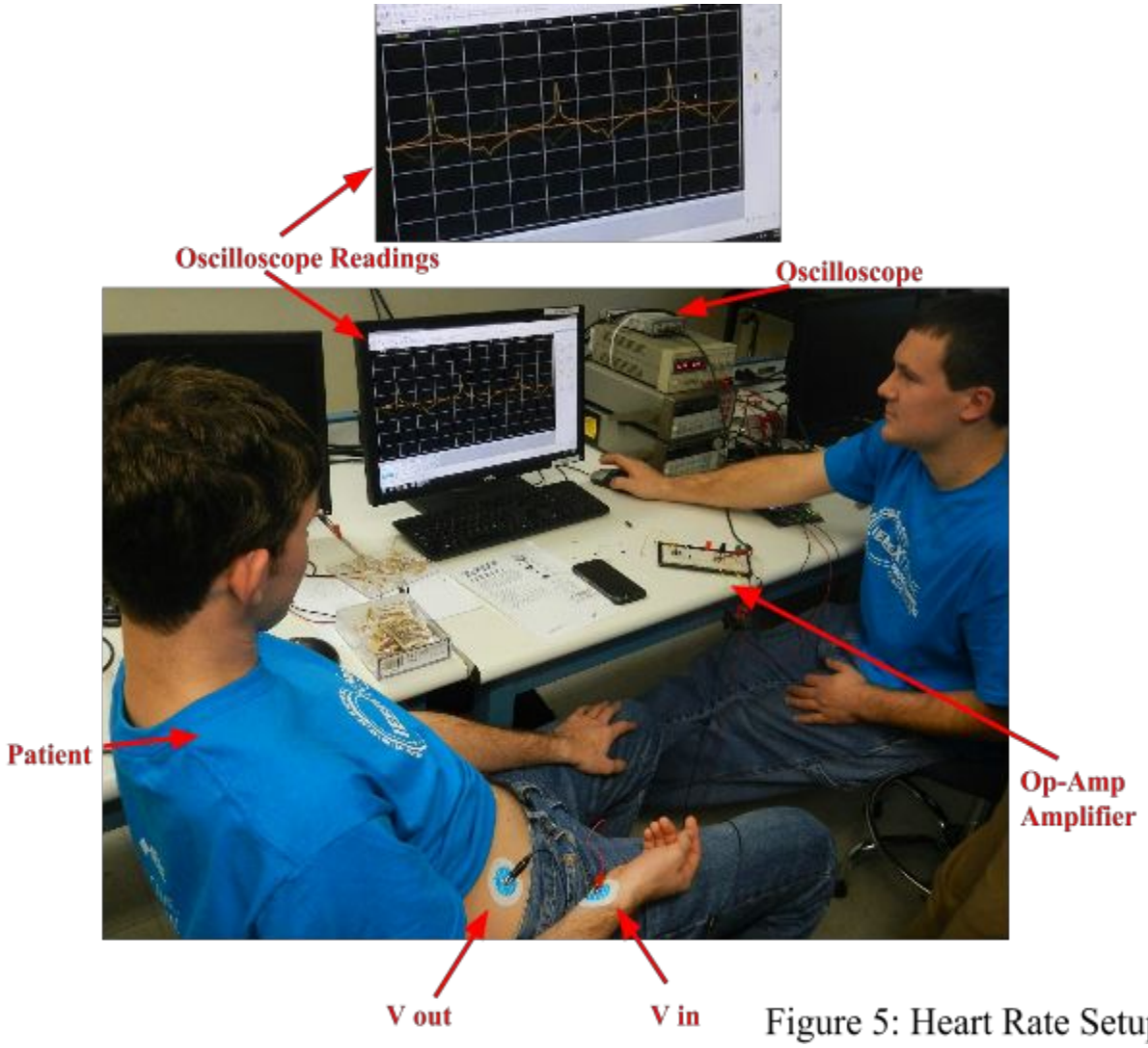


Figure 5: Heart Rate Setup

This diagram helps to show the locations of everything used to get a heart rate viewed on the oscilloscope. At this time the test that was being performed was from the wrist to the side.

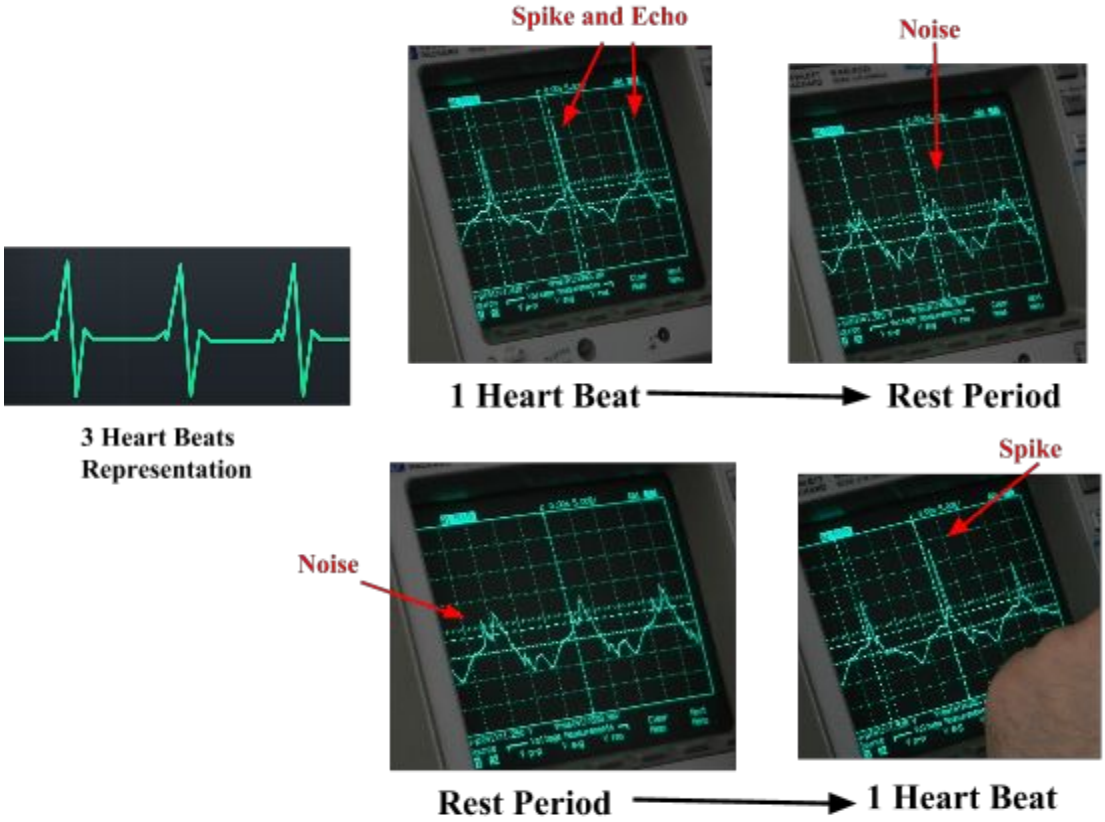


Figure 6: Heart Rate Readings

Figure 6 shows the heart rate pulses that we were able to view. It was finally determined and satisfied that these were heart rate pulses by use because there were not constant and you could see a pulse. Also the patient agreed that what we clearly sees his heart rate.

4. Problems and Results:

Throughout this project we encountered many problems**,** we have been able to overcome all of them. We have had three main problems, one with actually getting the lcd to display our waveform, another problem with how our ADC was working and NYQUIST as well as Hardware to read a heartbeat.

4.1. Oscilloscope

There were several problems to overcome with getting the oscilloscope to display. First was the fact that of the difficulty with displaying any sort of pixel control on the LCD. This was solved by being able to manipulate one of the character arrays and then use that character as the waveform. Then there was the moving from HEW to IAR. This wasn't as much a problem as it was just a stepping stone, this entitled finding which directories to put the code that was written in another software. The oscilloscope did cause problems when we needed to incorporate RTOS programming.

4.2. NYQUIST

When reading in an ADC value on our RTOS we were getting multiple waveforms, and we had no idea why. We also found out that the Oscilloscope only worked at low frequencies, extremely low frequencies. We thought it was a problem with our ADC, we thought we were reading our values too fast. It turns out we were reading our values too slow. When we did the math we found out the max frequency we could read accurately at the current ADC speed was 10 Hz, this was a problem since NYQUIST has to have double the reading points as the frequency. We found out that the ADC was waiting on a timer match. This ordinarily wouldn't be a problem, but the clock was already slowed down. So we took away the match timmer and now we are able to read waveforms up to 300 Hz.

4.3. Heart to Hardware Matchup

The hardware had problems because of the fact that this we needed to understand what the setup is. First was the fact that the signal from the patient needed to be amplified when we first viewed it on the oscilloscope. Then after the amplification there was a lot of noise that was seen often if the patient would do unwanted movement or touch the table there would be 60Hz stream that was seen. This was at first thought to be the heart rate until it was figured out that this was just from all the equipment that was attached to the table and the different vibrations that might transfer through the table. Then when the signal was moved to be located right across our patient's heart and an older oscilloscope was used, the best signal was seen on the oscilloscope.

5. Conclusion:

The complete projects of getting a Oscilloscope was implemented first through just a traditional non real time operating system then later implemented in micrium to be a RTOS so that multiple options could operate at the same time with the use of mutexes. This was successful by our method even though our method for the display was in a non ideal way.

The heart rate section was done to be able see an actual heart rate on the oscilloscope. Even though the oscilloscope that was built by our processor to was not able to be viewed. It was seen after all the components added to see the actual heart rate. Even though this signal was not the clearest and could be cleaned up we were satisfied that our task was accomplished.

6. Works Cited:

"Welcome - Welcome - Doc," *Micrium documentation site*. [Online]. Available at: https://doc.micrium.com/display/welcome/welcome. [Accessed: Sep-2015].

"YRDKRX63N (For RX63N)," *Documentation*. [Online]. Available at: http://www.am.renesas.com/products/tools/introductory_evaluation_tools/renesas_demo_kits/yrdkrx63n/documentation.jsp. [Accessed: Sep-2015].

7. Appendix

7.1. Non-RTOS:

MAIN::

```
#include <stdint.h>
#include <stdio.h>
#include <machine.h>
#include "platform.h"
#include "s12adc.h"
//#include "..\lcd.h"
//#include "..\r_glyph\src\glyph\fonts\font_8x8.c"
void delayMS1(int delay_time);


void main(void)
{
    /* Initialize LCD */
    lcd_initialize();


    /* Clear LCD */
    lcd_clear();
       int freq=0;
       int count=0;

   int temp=lcd_display2(LCD_LINE2, "~");
   if((freq/count)-10>=temp  &&(freq/count)+10<=temp);
   else{
   freq=freq+temp;
   count++;
   }
   char buf[20];
   sprintf((char *)buf, "FREQ=%5d",freq/count);

   lcd_display1(LCD_LINE7, buf);
   delayMS1(150);
       }
      printf("This is the debug console\r\n");



} /* End function main() */

void delayMS1(int delay_time)
{
       /* Used to pace delay for the LED */
       uint32_t led_counter;
       int i;
       for(i=0;i<delay_time; i++)
       {
              for (led_counter = 0; led_counter < 40000; led_counter++){}
       }
}
```

LCD:

```
#include <machine.h>
/* Standard string manipulation & formatting functions */
#include <stdio.h>
#include <string.h>
/* Defines standard variable types used in this function */
#include <stdint.h>
/* Board includes. */
#include "platform.h"
/* Following header file provides function prototypes for LCD controlling functions & macro
defines */
#include "lcd.h"
/* Graphics library support */
#include "glyph.h"
/* RSPI package. */
#include "r_rspi_rx600.h"
#include "s12adc.h"
#include "wdt.h"

void delayMS(int delay_time);
void lcd_display1(uint8_t position, const uint8_t * string);
/***********************************************************************************************
*************************
Private global variables and functions
***********************************************************************************************
************************/

T_glyphHandle lcd_handle;



 void delayMS(int delay_time)
{
	/* Used to pace delay for the LED */
	uint32_t led_counter;
	int i;
	for(i=0;i<delay_time; i++)
	{
		for (led_counter = 0; led_counter < 14000; led_counter++){}//6000
	}
}




int lcd_display2(uint8_t position, const uint8_t * string)
{

		uint8_t char_61[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
};

              uint8_t char_60[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_62[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
};
for(int i=0; i<9;i++)
{
       if(i<7)
       {
              char_61[i+2]=0x80;
              char_60[i+2]=0x01;
              char_62[i+2]=0x00;
       }
       else
       {
              char_61[i+2]=0xFF;
              char_60[i+2]=0xFF;
              char_62[i+2]=0xFF;
       }
}

              change(char_61);


       uint8_t y = position - (position % 8);
    uint8_t xOffset = (position % 8)<<3;

  GlyphEraseBlock(lcd_handle, xOffset, y, (95 - xOffset), y+7);
float minVolt=0;
float maxVolt=0;
int mincnt=17000;
int freq=0;
int freqc=0;
float amplitude;
uint8_t buf[13];
       wdt_init();
       for(int i=0; i<12; i++)
       {
       int z=0;
       float adc_count;
       S12ADC_init();
              uint8_t char_1[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

              uint8_t char_2[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_3[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_4[] = {
    0x08, 0x08, // width=8, height=8
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

//variables


        while(z<8)
        {
                delayMS(1);

                S12ADC_start();
                while(false==S12ADC_conversion_complete())
                {}

                adc_count= S12ADC_read();


                adc_count=adc_count/4095;
                float adccount=adc_count*3.3;

                if(adccount>maxVolt)
                {
                        maxVolt=adccount;
                }

                if((adccount>=(maxVolt-.03))&&(adccount<=maxVolt)) freqc=0;  else freqc=1;
                if(freqc==0) wdt_feed_watchdog();
                if(freqc==1){
                 //watchdog start
                 if(WDT.WDTSR.BIT.CNTVAL<mincnt)  mincnt=WDT.WDTSR.BIT.CNTVAL;

                }
                freq=(1000/(((16383-mincnt)*2796)/16383));

                sprintf((char *)buf, "FREQ=%5d",freq);



                if(adccount<minVolt)
                {
                        minVolt=adccount;
                }



                if(adccount<=.825)
                {

                                if(adccount<=.103125)
                                {
                                        char_1[z+2]=0x80;
                                }
                                if((adccount>.103125)&&(adccount<=.20625))
                                {
                                        char_1[z+2]=0x40;
                                }
                                if((adccount>.20625)&&(adccount<=.309375))
                                {
                                        char_1[z+2]=0x20;
                                }
                                if((adccount>.309375)&&(adccount<=.4125))
```

```
                {
                        char_1[z+2]=0x10;
                }
                if((adccount>.4125)&&(adccount<=.515625))
                {
                        char_1[z+2]=0x08;
                }
                if((adccount>.515625)&&(adccount<=.61875))
                {
                        char_1[z+2]=0x04;
                }
                if((adccount>.61875)&&(adccount<=.721875))
                {
                        char_1[z+2]=0x02;
                }
                if((adccount>.721875)&&(adccount<=.825))
                {
                        char_1[z+2]=0x01;
                }
}
if((adccount>.825)&&(adccount<=1.65))
{

                if((adccount>.825)&&(adccount<=.928125))
                {
                        char_2[z+2]=0x80;
                }
                if((adccount>.928125)&&(adccount<=1.03125))
                {
                        char_2[z+2]=0x40;
                }
                if((adccount>1.03125)&&(adccount<=1.134375))
                {
                        char_2[z+2]=0x20;
                }
                if((adccount>1.134375)&&(adccount<=1.2375))
                {
                        char_2[z+2]=0x10;
                }
                if((adccount>1.2375)&&(adccount<=1.340625))
                {
                        char_2[z+2]=0x08;
                }
                if((adccount>1.340625)&&(adccount<=1.44375))
                {
                        char_2[z+2]=0x04;
                }
                if((adccount>1.44375)&&(adccount<=1.546875))
                {
                        char_2[z+2]=0x02;
                }
                if((adccount>1.546875)&&(adccount<=1.65))
                {
                        char_2[z+2]=0x01;
                }
}
if((adccount>1.65)&&(adccount<=2.475))
{

                if((adccount>1.65)&&(adccount<=1.753125))
                {
```

```
                        char_3[z+2]=0x80;
                }
                if((adccount>1.753125)&&(adccount<=1.85625))
                {
                        char_3[z+2]=0x40;
                }
                if((adccount>1.85625)&&(adccount<=1.959375))
                {
                        char_3[z+2]=0x20;
                }
                if((adccount>1.959375)&&(adccount<=2.0625))
                {
                        char_3[z+2]=0x10;
                }
                if((adccount>2.0625)&&(adccount<=2.165625))
                {
                        char_3[z+2]=0x08;
                }
                if((adccount>2.165625)&&(adccount<=2.26875))
                {
                        char_3[z+2]=0x04;
                }
                if((adccount>2.26875)&&(adccount<=2.371875))
                {
                        char_3[z+2]=0x02;
                }
                if((adccount>2.371875)&&(adccount<=2.475))
                {
                        char_3[z+2]=0x01;
                }
}
if((adccount>2.475)&&(adccount<=3.3))
{

                if((adccount>2.475)&&(adccount<=2.578125))
                {
                        char_4[z+2]=0x80;
                }
                if((adccount>2.578125)&&(adccount<=2.68125))
                {
                        char_4[z+2]=0x40;
                }
                if((adccount>2.68125)&&(adccount<=2.784375))
                {
                        char_4[z+2]=0x20;
                }
                if((adccount>2.784375)&&(adccount<=2.8875))
                {
                        char_4[z+2]=0x10;
                }
                if((adccount>2.8875)&&(adccount<=2.990625))
                {
                        char_4[z+2]=0x08;
                }
                if((adccount>2.990625)&&(adccount<=3.09375))
                {
                        char_4[z+2]=0x04;
                }
                if((adccount>3.09375)&&(adccount<=3.196875))
                {
                        char_4[z+2]=0x02;
```

```
                        }
                        if((adccount>3.196875)&&(adccount<=3.3))
                        {
                                char_4[z+2]=0x01;
                        }
                }
                z++;

        }
        int x1,y1;
        change(char_1);
        GlyphSetXY(lcd_handle, i*8, 32);
        GlyphChar(lcd_handle, '~');

                change(char_2);
        GlyphSetXY(lcd_handle, i*8, 24);
        GlyphChar(lcd_handle, '~');

                change(char_3);
        GlyphSetXY(lcd_handle, i*8, 18);
        GlyphChar(lcd_handle, '~');

                change(char_4);
        GlyphSetXY(lcd_handle, i*8, 9);
        GlyphChar(lcd_handle, '~');

        }
        char amp[20];
        char Freq[20];
        amplitude=(maxVolt-minVolt)/2;
        sprintf(amp,"Amp: %4f",amplitude);

        lcd_display1(LCD_LINE1,"Mitch & Paul");
        lcd_display1(LCD_LINE6,amp);
                return freq;

}
```

8X8:

```
#include "Config.h"
void change(uint8_t array[10]);
//void change2(uint8_t array2[10]);


 uint8_t Font8x8_char_126[] = {
    0x08, 0x08, // width=8, height=8
    0x02, 0x03, 0x01, 0x03, 0x02, 0x03, 0x01, 0x00,
};


const uint8_t * Font8x8_table[256] = {
    Font8x8_char_0,
    Font8x8_char_1,
    Font8x8_char_2,
    Font8x8_char_3,
    Font8x8_char_4,
    Font8x8_char_5,
    Font8x8_char_6,
```

```
Font8x8_char_7,
Font8x8_char_8,
Font8x8_char_9,
Font8x8_char_10,
Font8x8_char_11,
Font8x8_char_12,
Font8x8_char_13,
Font8x8_char_14,
Font8x8_char_15,
Font8x8_char_16,
Font8x8_char_17,
Font8x8_char_18,
Font8x8_char_19,
Font8x8_char_20,
Font8x8_char_21,
Font8x8_char_22,
Font8x8_char_23,
Font8x8_char_24,
Font8x8_char_25,
Font8x8_char_26,
Font8x8_char_27,
Font8x8_char_28,
Font8x8_char_29,
Font8x8_char_30,
Font8x8_char_31,
Font8x8_char_32,
Font8x8_char_33,
Font8x8_char_34,
Font8x8_char_35,
Font8x8_char_36,
Font8x8_char_37,
Font8x8_char_38,
Font8x8_char_39,
Font8x8_char_40,
Font8x8_char_41,
Font8x8_char_42,
Font8x8_char_43,
Font8x8_char_44,
Font8x8_char_45,
Font8x8_char_46,
Font8x8_char_47,
Font8x8_char_48,
Font8x8_char_49,
Font8x8_char_50,
Font8x8_char_51,
Font8x8_char_52,
Font8x8_char_53,
Font8x8_char_54,
Font8x8_char_55,
Font8x8_char_56,
Font8x8_char_57,
Font8x8_char_58,
Font8x8_char_59,
Font8x8_char_60,
Font8x8_char_61,
Font8x8_char_62,
Font8x8_char_63,
Font8x8_char_64,
Font8x8_char_65,
Font8x8_char_66,
Font8x8_char_67,
```

```
Font8x8_char_68,
Font8x8_char_69,
Font8x8_char_70,
Font8x8_char_71,
Font8x8_char_72,
Font8x8_char_73,
Font8x8_char_74,
Font8x8_char_75,
Font8x8_char_76,
Font8x8_char_77,
Font8x8_char_78,
Font8x8_char_79,
Font8x8_char_80,
Font8x8_char_81,
Font8x8_char_82,
Font8x8_char_83,
Font8x8_char_84,
Font8x8_char_85,
Font8x8_char_86,
Font8x8_char_87,
Font8x8_char_88,
Font8x8_char_89,
Font8x8_char_90,
Font8x8_char_91,
Font8x8_char_92,
Font8x8_char_93,
Font8x8_char_94,
Font8x8_char_95,
Font8x8_char_96,
Font8x8_char_97,
Font8x8_char_98,
Font8x8_char_99,
Font8x8_char_100,
Font8x8_char_101,
Font8x8_char_102,
Font8x8_char_103,
Font8x8_char_104,
Font8x8_char_105,
Font8x8_char_106,
Font8x8_char_107,
Font8x8_char_108,
Font8x8_char_109,
Font8x8_char_110,
Font8x8_char_111,
Font8x8_char_112,
Font8x8_char_113,
Font8x8_char_114,
Font8x8_char_115,
Font8x8_char_116,
Font8x8_char_117,
Font8x8_char_118,
Font8x8_char_119,
Font8x8_char_120,
Font8x8_char_121,
Font8x8_char_122,
Font8x8_char_123,
Font8x8_char_124,
Font8x8_char_125,
Font8x8_char_126,
Font8x8_char_127,
Font8x8_char_128,
```

```
Font8x8_char_129,
Font8x8_char_130,
Font8x8_char_131,
Font8x8_char_132,
Font8x8_char_133,
Font8x8_char_134,
Font8x8_char_135,
Font8x8_char_136,
Font8x8_char_137,
Font8x8_char_138,
Font8x8_char_139,
Font8x8_char_140,
Font8x8_char_141,
Font8x8_char_142,
Font8x8_char_143,
Font8x8_char_144,
Font8x8_char_145,
Font8x8_char_146,
Font8x8_char_147,
Font8x8_char_148,
Font8x8_char_149,
Font8x8_char_150,
Font8x8_char_151,
Font8x8_char_152,
Font8x8_char_153,
Font8x8_char_154,
Font8x8_char_155,
Font8x8_char_156,
Font8x8_char_157,
Font8x8_char_158,
Font8x8_char_159,
Font8x8_char_160,
Font8x8_char_161,
Font8x8_char_162,
Font8x8_char_163,
Font8x8_char_164,
Font8x8_char_165,
Font8x8_char_166,
Font8x8_char_167,
Font8x8_char_168,
Font8x8_char_169,
Font8x8_char_170,
Font8x8_char_171,
Font8x8_char_172,
Font8x8_char_173,
Font8x8_char_174,
Font8x8_char_175,
Font8x8_char_176,
Font8x8_char_177,
Font8x8_char_178,
Font8x8_char_179,
Font8x8_char_180,
Font8x8_char_181,
Font8x8_char_182,
Font8x8_char_183,
Font8x8_char_184,
Font8x8_char_185,
Font8x8_char_186,
Font8x8_char_187,
Font8x8_char_188,
Font8x8_char_189,
```

```
Font8x8_char_190,
Font8x8_char_191,
Font8x8_char_192,
Font8x8_char_193,
Font8x8_char_194,
Font8x8_char_195,
Font8x8_char_196,
Font8x8_char_197,
Font8x8_char_198,
Font8x8_char_199,
Font8x8_char_200,
Font8x8_char_201,
Font8x8_char_202,
Font8x8_char_203,
Font8x8_char_204,
Font8x8_char_205,
Font8x8_char_206,
Font8x8_char_207,
Font8x8_char_208,
Font8x8_char_209,
Font8x8_char_210,
Font8x8_char_211,
Font8x8_char_212,
Font8x8_char_213,
Font8x8_char_214,
Font8x8_char_215,
Font8x8_char_216,
Font8x8_char_217,
Font8x8_char_218,
Font8x8_char_219,
Font8x8_char_220,
Font8x8_char_221,
Font8x8_char_222,
Font8x8_char_223,
Font8x8_char_224,
Font8x8_char_225,
Font8x8_char_226,
Font8x8_char_227,
Font8x8_char_228,
Font8x8_char_229,
Font8x8_char_230,
Font8x8_char_231,
Font8x8_char_232,
Font8x8_char_233,
Font8x8_char_234,
Font8x8_char_235,
Font8x8_char_236,
Font8x8_char_237,
Font8x8_char_238,
Font8x8_char_239,
Font8x8_char_240,
Font8x8_char_241,
Font8x8_char_242,
Font8x8_char_243,
Font8x8_char_244,
Font8x8_char_245,
Font8x8_char_246,
Font8x8_char_247,
Font8x8_char_248,
Font8x8_char_249,
Font8x8_char_250,
```

```
    Font8x8_char_251,
    Font8x8_char_252,
    Font8x8_char_253,
    Font8x8_char_254,
    Font8x8_char_255,
};



void change(uint8_t array[10])
{
        Font8x8_char_126[0] = array[0];
        Font8x8_char_126[1] = array[1];
        Font8x8_char_126[2] = array[2];
        Font8x8_char_126[3] = array[3];
        Font8x8_char_126[4] = array[4];
        Font8x8_char_126[5] = array[5];
        Font8x8_char_126[6] = array[6];
        Font8x8_char_126[7] = array[7];
        Font8x8_char_126[8] = array[8];
        Font8x8_char_126[9] = array[9];
}
```

S12ADC:

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include "platform.h"
#include "s12adc.h"

void S12ADC_init (void)
{
#ifdef PLATFORM_BOARD_RDKRX63N
        SYSTEM.PRCR.WORD = 0xA50B; /* Protect off */
#endif

    /* Power up the S12ADC */
    MSTP(S12AD) = 0;

    /* Set up the I/O pin that will be used as analog input source. In this
       demo the potentiometer will be used and is connected to port 42. */
    PORT4.PODR.BIT.B1 = 0;    /* Clear I/O pin data register to low output. */
    PORT4.PDR.BIT.B1  = 0;    /* Set I/O pin direction to input. */
    PORT4.PMR.BIT.B1  = 0;    /* First set I/O pin mode register to GPIO mode. */
    //MPC.P42PFS.BYTE = 0x80;   /* Set port function register to analog input, no interrupt.
*/

#ifdef PLATFORM_BOARD_RDKRX63N
        SYSTEM.PRCR.WORD = 0xA500; /* Protect on  */
#endif

    /* ADCSR: A/D Control Register
    b7    ADST     0 a/d conversion start, Stop a scan conversion process
    b6    ADCS     0 Scan mode select, Single-scan mode
    b5    Reserved 0 This bit is always read as 0. The write value should always be 0.
```

```
    b4    ADIE     0 Disables conversion complete IRQ to ICU
    b3:b2 CKS      0 A/D conversion clock select = PCLK/8
    b1    TRGE     0 Disables conversion to start w/ trigger
    b0    EXTRG    0 Trigger select, Scan conversion start by a timer source or software
       */
    S12AD.ADCSR.BYTE = 0x00;

    /* ADANS0: A/D Channel Select Register 0
    b15:b0  ANS0: Selects analog inputs of the channels AN000 to AN015 that are subjected to
A/D conversion
    */
    S12AD.ADANS0.WORD = 0x0002; /* Read AN002, which is connected to the potentiometer */

    /* ADANS1: A/D Channel Select Register 1
       b15:b5  Reserved: These bits are always read as 0. The write value should always be 0.
    b4:b0   ANS1:     Selects analog inputs of the channels AN016 to AN020 that are subjected
to A/D conversion
    */
    S12AD.ADANS1.WORD = 0x0000;

    /* ADADS0: A/D-converted Value Addition Mode Select Register 0
    b15:b0  ADS0: A/D-Converted Value Addition Channel Select for AN000 to AN015.
    */
    S12AD.ADADS0.WORD = 0x0000;

    /* ADADS1: A/D-converted Value Addition Mode Select Register 1
       b15:b5  Reserved: These bits are always read as 0. The write value should always be 0.
    b4:b0   ADS1: A/D-Converted Value Addition Channel Select for AN016 to AN020.
    */
    S12AD.ADADS1.WORD = 0x0000;

    /* ADADC: A/D-Converted Value Addition Count Select Register
    b1:b0   ADC: 00 = 1 time conversion (same as normal conversion)
    */
    S12AD.ADADC.BYTE = 0x00;   /* 1-time conversion */

    /* ADCER: A/D Control Extended Register
    b15     ADRFMT:0  Right align the data in the result registers
    b5      ACE:0 Disables automatic clearing of ADDRn after it is read
    */
    S12AD.ADCER.WORD = 0x0000;   /* Right align data, automatic clearing off. */

    /* ADSTRGR: A/D Start Triggger Select Register
    b7:b4   Reserved. Always read/write 0.
    b3:b0   ADSTRS: 0, Software trigger or ADTRG0#
    */
    S12AD.ADSTRGR.BYTE = 0x00;
} /* End of function S12ADC_init() */


/***********************************************************************
* Function name: S12ADC_read
* Description  : Reads the results register for channel AN002 of the S12ADC
* Arguments    : none
* Return value : uint16_t -
*                  The ADC conversion value.
***********************************************************************/
uint16_t S12ADC_read (void)
{
    uint16_t adc_result;
```

```
    adc_result = S12AD.ADDR1;        /* Read the result register for AN2 */

    return adc_result;
} /* End of function S12ADC_read() */

/***********************************************************************
* End of file s12adc.c
***********************************************************************/
```

WATCHDOG:

```
#include <stdint.h>
#include <platform.h>


void wdt_init(void)
{
       /* Note that OFS0.WDTSTRT is 1 at power up, so the WDT is in register start mode */

    /* WDT Reset Control Register (WDTRCR)
       Writing to this register is only possible between processor reset and the
          very first watchdog refresh operation.
          This makes it hard for a runaway rogue program to disable the watchdog.
       b7      RSTIRQS:  Reset Interrupt Request Selection
       b6:b0   Reserved: These bits are read as 0 and cannot be modified.
    */
       WDT.WDTRCR.BYTE = 0x00; /* 0x80 = Reset (and not NMI) output is enabled. */

       /* If you want to use the Non-Maskable Interrupt NMI with the watchdog, you must
          unmask it by using the Non-Maskable Interrupt Enable Register NMIER. */
       ICU.NMIER.BIT.WDTEN = 1;

    /* WDT Control Register (WDTCR)
       Writing to this register is only possible between processor reset and the
          very first watchdog refresh operation.
          This makes it hard for a runaway rogue program to disable the watchdog.
       b15:b14 Reserved: These bits are read as 0 and cannot be modified.
       b13:b12 RPSS:     Window Start Position Selection
       b11:b10 Reserved: These bits are read as 0 and cannot be modified.
       b9:b8   RPES:     Window End Position Selection
       b7:b4   CKS:      Clock Division Ratio Selection
       b3:b2   Reserved: These bits are read as 0 and cannot be modified.
    b1:b0   TOPS:     Time-Out Period Selection
    */
       WDT.WDTCR.WORD = 0x3383; /* Timeout period without feedings is 134,217,728 PCLKs */
                                /* PCLK = 48 MHz */
                                                /* Timeout period = 134,217,728 / 48,000,000
= 2.796 seconds */
                                                /* The watchdog may be fed at any time. */

    /* WDT Status Register (WDTSR)
       b15     REFEF:    Refresh Error Flag
```

```
        b14     UNDFF:    Underflow Flag
        b13:b0  CNTVAL:   Down-Counter Value.
    */
        WDT.WDTSR.WORD = 0x0000; /* Clear the refresh error and underflow flags */

        wdt_feed_watchdog();     /* The very first watchdog reset operation */
} /* End of function wdt_init(). */

/***********************************************************************************
End of file: wdt.c
***********************************************************************************/
```

7.2. RTOS

```
#include  "app_cfg.h"
#include  "cpu_cfg.h"
#include  "bsp_cfg.h"


#include  <cpu_core.h>
#include  <os.h>
#include  <iodefine.h>

#include  "..\bsp\bsp.h"
#include  "..\bsp\bsp_misc.h"
#include  "..\bsp\bsp_int_vect_tbl.h"

#if       BSP_CFG_SER_EN       > 0u
#include  "..\bsp\bsp_ser.h"
#endif

#if       BSP_CFG_LED_EN       > 0u
#include  "..\bsp\bsp_led.h"
#endif

#if       BSP_CFG_GRAPH_LCD_EN > 0u
#include  "..\bsp\bsp_glcd.h"
#endif

#if       SPIN_LEDS_APP_EN     > 0u
#include  "spin_led.h"
#endif



static  OS_SEM App_SemADC;
static  OS_TCB AppTaskLCDTCB;
static  CPU_STK AppTaskLCDStk[APP_CFG_TASK_LCD_STK_SIZE];

static  CPU_STK     AppTaskStartStk[APP_CFG_TASK_START_STK_SIZE];

static  OS_TCB      AppTaskStartTCB;



static  void     AppTaskStart (void  *p_arg);
static  void     AppTaskLCD    (void *p_arg);
CPU_ISR  App_ISRADC    (void);
void delayMS(int delay_time);
```

```
int data[96];
int pos=0;
int current=0;

void  main (void)
{
    OS_ERR  err;


    CPU_IntDis();                                               /* Disable all interrupts.
*/

    BSP_IntVectSet(27, (CPU_FNCT_VOID)OSCtxSwISR);              /* Setup kernel context switch
*/

    OSInit(&err);                                               /* Init uC/OS-III.
*/


    OSTaskCreate((OS_TCB     *)&AppTaskStartTCB,                /* Create the start task
*/
                 (CPU_CHAR   *)"Startup Task",
                 (OS_TASK_PTR ) AppTaskStart,
                 (void       *) 0,
                 (OS_PRIO     ) APP_CFG_TASK_START_PRIO,
                 (CPU_STK    *)&AppTaskStartStk[0],
                 (CPU_STK_SIZE) APP_CFG_TASK_START_STK_SIZE / 10u,
                 (CPU_STK_SIZE) APP_CFG_TASK_START_STK_SIZE,
                 (OS_MSG_QTY  ) 0u,
                 (OS_TICK     ) 0u,
                 (void       *) 0,
                 (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                 (OS_ERR     *)&err);

    OSStart(&err);                                              /* Start multitasking (i.e.
give control to uC/OS-III). */

    while (1) {
        ;
    }
}

static  void  AppTaskLCD (void *p_arg)
{
  CPU_INT16U  adc_val;
  CPU_CHAR    adc_str[7];
  OS_ERR      err;
  BSP_Ser_Init(115200);
  int freqF;
  int count;
//BSP_GraphLCD_String(3, "ADC Value:");
BSP_IntVectSet(VECT_S12AD0_S12ADI0, (CPU_FNCT_VOID)App_ISRADC);
BSP_ADC_Init();

while (DEF_ON) {


  int8_t char_61[] = {
    0x08, 0x08, // width=8, height=8
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

              uint8_t char_60[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_62[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
};
for(int i=0; i<9;i++)
{
       if(i<7)
       {
              char_61[i+2]=0x80;
              char_60[i+2]=0x01;
              char_62[i+2]=0x00;
       }
       else
       {
              char_61[i+2]=0xFF;
              char_60[i+2]=0xFF;
              char_62[i+2]=0xFF;
       }
}
BSP_GraphLCD_Clear ();
float minVolt=0;
float maxVolt=0;
int mincnt=17000;
int freq=0;
int freqc=0;
float amplitude;
uint8_t buf[13];
for(int i=0; i<12; i++)
       {
         int z=0;
        float adc_count;

        uint8_t char_1[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

              uint8_t char_2[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_3[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
              uint8_t char_4[] = {
    0x08, 0x08, // width=8, height=8
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
               };

while(z<8)
       {

      CPU_CHAR    adc_str[7];
```

```
    int adcNEW=data[current];

              adc_count=data[current];
              current++;
              if(current==96) {current=0;}
              adc_count=adc_count/4095;
             float adccount=adc_count*3.3;

             if(adccount>maxVolt)
             {
                    maxVolt=adccount;
             }

             if((adccount>=(maxVolt-.03))&&(adccount<=maxVolt)) freqc=0;  else freqc=1;
             if(freqc==0) ;//wdt_feed_watchdog();
             if(freqc==1){
              //watchdog start
              //if(WDT.WDTSR.BIT.CNTVAL<mincnt)  mincnt=WDT.WDTSR.BIT.CNTVAL;

             }
             if(freqc==2){
             freqc=0;
                    //if(WDT.WDTSR.BIT.CNTVAL<mincnt)  mincnt=WDT.WDTSR.BIT.CNTVAL;
                     //wdt_feed_watchdog();

             }
              freq=(1000/(((16383-mincnt)*2796)/16383));
              sprintf((char *)buf, "FREQ=%5d",freq);

              if(adccount<minVolt)
             {
                    minVolt=adccount;
             }

if(adccount<=.825)
             {

                           if(adccount<=.103125)
                           {
                                  char_1[z+2]=0x80;
                           }
                           if((adccount>.103125)&&(adccount<=.20625))
                           {
                                  char_1[z+2]=0x40;
                           }
                           if((adccount>.20625)&&(adccount<=.309375))
                           {
                                  char_1[z+2]=0x20;
                           }
                           if((adccount>.309375)&&(adccount<=.4125))
                           {
                                  char_1[z+2]=0x10;
                           }
                           if((adccount>.4125)&&(adccount<=.515625))
                           {
                                  char_1[z+2]=0x08;
                           }
                           if((adccount>.515625)&&(adccount<=.61875))
                           {
                                  char_1[z+2]=0x04;
```

```
                }
                if((adccount>.61875)&&(adccount<=.721875))
                {
                        char_1[z+2]=0x02;
                }
                if((adccount>.721875)&&(adccount<=.825))
                {
                        char_1[z+2]=0x01;
                }
}
if((adccount>.825)&&(adccount<=1.65))
{

                if((adccount>.825)&&(adccount<=.928125))
                {
                        char_2[z+2]=0x80;
                }
                if((adccount>.928125)&&(adccount<=1.03125))
                {
                        char_2[z+2]=0x40;
                }
                if((adccount>1.03125)&&(adccount<=1.134375))
                {
                        char_2[z+2]=0x20;
                }
                if((adccount>1.134375)&&(adccount<=1.2375))
                {
                        char_2[z+2]=0x10;
                }
                if((adccount>1.2375)&&(adccount<=1.340625))
                {
                        char_2[z+2]=0x08;
                }
                if((adccount>1.340625)&&(adccount<=1.44375))
                {
                        char_2[z+2]=0x04;
                }
                if((adccount>1.44375)&&(adccount<=1.546875))
                {
                        char_2[z+2]=0x02;
                }
                if((adccount>1.546875)&&(adccount<=1.65))
                {
                        char_2[z+2]=0x01;
                }
}
if((adccount>1.65)&&(adccount<=2.475))
{

                if((adccount>1.65)&&(adccount<=1.753125))
                {
                        char_3[z+2]=0x80;
                }
                if((adccount>1.753125)&&(adccount<=1.85625))
                {
                        char_3[z+2]=0x40;
                }
                if((adccount>1.85625)&&(adccount<=1.959375))
                {
                        char_3[z+2]=0x20;
                }
```

```
                if((adccount>1.959375)&&(adccount<=2.0625))
                {
                        char_3[z+2]=0x10;
                }
                if((adccount>2.0625)&&(adccount<=2.165625))
                {
                        char_3[z+2]=0x08;
                }
                if((adccount>2.165625)&&(adccount<=2.26875))
                {
                        char_3[z+2]=0x04;
                }
                if((adccount>2.26875)&&(adccount<=2.371875))
                {
                        char_3[z+2]=0x02;
                }
                if((adccount>2.371875)&&(adccount<=2.475))
                {
                        char_3[z+2]=0x01;
                }
}
if((adccount>2.475)&&(adccount<=3.3))
{

                if((adccount>2.475)&&(adccount<=2.578125))
                {
                        char_4[z+2]=0x80;
                }
                if((adccount>2.578125)&&(adccount<=2.68125))
                {
                        char_4[z+2]=0x40;
                }
                if((adccount>2.68125)&&(adccount<=2.784375))
                {
                        char_4[z+2]=0x20;
                }
                if((adccount>2.784375)&&(adccount<=2.8875))
                {
                        char_4[z+2]=0x10;
                }
                if((adccount>2.8875)&&(adccount<=2.990625))
                {
                        char_4[z+2]=0x08;
                }
                if((adccount>2.990625)&&(adccount<=3.09375))
                {
                        char_4[z+2]=0x04;
                }
                if((adccount>3.09375)&&(adccount<=3.196875))
                {
                        char_4[z+2]=0x02;
                }
                if((adccount>3.196875)&&(adccount<=3.3))
                {
                        char_4[z+2]=0x01;
                }
}
  char adcnum[12];

int ADCNEW1=adcNEW & 0xFF;
 int ADCNEW2=adcNEW & 0xFF00;
```

```
                ADCNEW2=ADCNEW2/256;


                 BSP_Ser_WrByte(ADCNEW2);
                z++;
                 delayMS(15);
                 BSP_Ser_WrByte(ADCNEW1);

        }
       int x1,y1;
BSP_GraphLCD_String1(char_1, i*8, 32);
BSP_GraphLCD_String1(char_2, i*8, 24);
BSP_GraphLCD_String1(char_3, i*8, 18);
BSP_GraphLCD_String1(char_4, i*8, 9);

        char str1[10];
       char Freq[12];
       amplitude=(maxVolt-minVolt)/2;
        Str_FmtNbr_Int32U(amplitude*1000, 6, 10, ' ', DEF_NO, DEF_YES, str1);

       BSP_GraphLCD_String(0,"Mitch & Paul");
       BSP_GraphLCD_String(6,str1);

        if((freqF/count)-10>=freq  &&(freqF/count)+10<=freq);
   else{
   freqF=freqF+freq;
   count++;
   }
   char buff[12];


        }



  delayMS(2000);
}


}




static  void  AppTaskStart (void *p_arg)
{
    CPU_INT08U  i;
    OS_ERR      err;


   (void)&p_arg;

    BSP_Init();                                                 /* Initialize BSP functions
*/

    CPU_Init();                                                 /* Initialize the uC/CPU
services                         */

    OSSemCreate(&App_SemADC, "ADC Sem", 0, &err);
OSTaskCreate((OS_TCB     *)&AppTaskLCDTCB,
```

```
                (CPU_CHAR   *)"App Task LCD",
                (OS_TASK_PTR ) AppTaskLCD,
                (void       *) 0,
                (OS_PRIO     ) APP_CFG_TASK_LCD_PRIO,
                (CPU_STK    *)&AppTaskLCDStk[0],
                (CPU_STK_SIZE) APP_CFG_TASK_LCD_STK_SIZE/10u,
                (CPU_STK_SIZE) APP_CFG_TASK_LCD_STK_SIZE,
                (OS_MSG_QTY  ) 0u,
                (OS_TICK     ) 0u,
                (void       *) 0,
                (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                (OS_ERR     *)&err);


#if (OS_CFG_STAT_TASK_EN > 0u)
    OSStatTaskCPUUsageInit(&err);                               /* Compute CPU capacity with
no task running            */
#endif

    Mem_Init();

#ifdef CPU_CFG_INT_DIS_MEAS_EN
    CPU_IntDisMeasMaxCurReset();
#endif

#if BSP_CFG_SER_EN > 0u
    BSP_Ser_Init(115200);                                       /* Initialize serial
communications                     */
#endif

#if (BSP_CFG_GRAPH_LCD_EN > 0u)
    BSP_GraphLCD_Clear();
#endif



    while (DEF_ON) {
        for (i = 4; i <= 15; i++) {
            BSP_LED_On(i);
            OSTimeDlyHMSM(0, 0, 0, 100,
                          OS_OPT_TIME_HMSM_STRICT,
                          &err);
        }
        for (i = 15; i >= 4; i--) {
            BSP_LED_Off(i);
            OSTimeDlyHMSM(0, 0, 0, 100,
                          OS_OPT_TIME_HMSM_STRICT,
                          &err);
        }
    }

}

int first=0;

void delayMS(int delay_time)
{
       /* Used to pace delay for the LED */
       uint32_t led_counter;
       int i;
```

```
        for(i=0;i<delay_time; i++)
        {
                for (led_counter = 0; led_counter < 14000; led_counter++){}
        }
}

#if      __RENESAS__
#pragma  interrupt  OS_BSP_TickISR
#endif
CPU_ISR  App_ISRADC (void)
{
  OS_ERR  err;
OSIntEnter();

if((pos==96) && (current==0)) {

  first=0;
  pos=0;
  data[pos]=adc_val;
  pos++;
}

if(pos!=96){
data[pos]=adc_val;
pos++;
}
OSIntExit();
}
```

LCD:

```
void  BSP_GraphLCD_String1(uint8_t char_1[], uint32_t aX ,uint32_t aY)
 {
   change(char_1);
        GlyphSetXY(BSP_GraphLCD, aX, aY);
       GlyphChar(BSP_GraphLCD, '~');
 }
```

BSP_MISC:

```
void  BSP_ADC_Init (void)
{
```

```
    PORT4.PMR.BIT.B1     = 1;                    /* Enable the input buffer for
the pot's pin            */

    SYSTEM.PRCR.WORD     = 0xA50B;               /* Protect off
*/
    MSTP(S12AD) = 0;                             /* Enable the 12-bit ADC
*/
    SYSTEM.PRCR.WORD     = 0xA500;               /* Protect on
*/


match                     */                    /* Set up conversions on TMR0
    S12AD.ADSTRGR.BYTE   = 0x09;

conversion                   */                  /* Clear value after each
    S12AD.ADCER.WORD     = 0x0020;

*/                                               /* Select ADC channel 4
    S12AD.ADANS0.WORD    = 0x0002;
    S12AD.ADADS0.WORD    = 0x0000;

*/                                               /* Set up clock and trigger
    S12AD.ADCSR.BYTE     = 0x12;
#ifdef OS3_ADC_LAB
    IR(S12AD0, S12ADI0)  = 0;                    /* Clear the ADC's interrupt
flag                       */
    IPR(S12AD0, S12ADI0) = 5;                    /* Set the priority for ADC
interrupts                  */
    IEN(S12AD0, S12ADI0) = 1;                    /* Enable ADC interrupts
*/
#endif

    SYSTEM.PRCR.WORD     = 0xA50B;               /* Protect off
*/
    MSTP(TMR0)           = 0;
    SYSTEM.PRCR.WORD     = 0xA500;               /* Protect on
*/


match                     */                    /* Set timer to be cleared on
    TMR0.TCR.BYTE        = 0x08;

conversions on match           */                /* Set timer to trigger
    TMR0.TCSR.BYTE       = 0x10;

*/                                               /* Specify match value
    TMR0.TCORA           = 0x01;

PCLK/8192                   */                   /* Specify clock speed of
    TMR0.TCCR.BYTE       = 0x0E;
}
```