ECGR 4101/5101, Spring 2012: Lab 2

Learning by Example: You're First Embedded LCD Game

Learning Objectives:

This lab will introduce you to the exciting world of embedded game development (sort of) and introduce you to the necessary steps needed to successfully create a simplistic game on the RX62N development board along with teach you how to create your own RX62N project from scratch.

General Information:

1. Review the steps provided by the lab 1 supplemental information document
2. Create a new workspace YRDKRX62N
3. Copy the necessary files into your new workspace and import them
4. Write your code to create a very simplistic embedded game.
5. 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 an empty YRDKRX62N workspace and to do this you will need to follow the steps provided by the lab 1 supplemental information document up to but not including Step 8. **Be sure to name your workspace Lab 2 rather than Lab1!**

Once you have the dialog shown in step 8 on your screen rather than creating a "Tutorial" project you want to select the "**Application**" option that has the sub text "Empty application project." And press finish. At this point the lab 1 supplemental information document can be followed again as you will need to fix the linker complier error discussed in the document.

Once you have created your new Lab2 workspace you will need to open up windows explore and navigate to the **C:\Micrium\Software\EvalBoards\Renesas\YRDKRX62N\BSP\Glyph** folder. Note this folder is only created if you did a full install of the RX software, if you did not do a full install or these files are missing you can download them from the Renesas RX website or simply copy the required files from the Woodward 203 lab computer to a thumb drive.

You will need to copy the following files

1) ST7579_LCD.c
2) ST7579_LCD.h
3) types.h
4) font_bitmap.c
5) Glyph.h
6) Glyph_cfg.h

To your newly created Lab2 workspace which could be found at C:\WorkSpace\Lab2\Lab2 if you named your workspace Lab2 and your workspace folder is at C:\WorkSpace.

Note the correct folder should have a bunch of c and h files.

After you have copied the necessary files you will need to add them into your Lab2 workspace.

To do this, simply click the Project menu bar at the top of the screen and press the Add Files… sub item that appears.

Once you have added all of these files, you will need to open Glyph_cfg.h and comment out the following lines of code

```
#define USE_GLYPH_FONT_HELVR10
#define USE_GLYPH_FONT_5_BY_7
#define USE_GLYPH_FONT_6_BY_13
```

You will also need to uncomment out

```
//#define USE_GLYPH_FONT_BITMAP
//#define USE_DEFAULT_FONT Bitmaps_table
```

If you did this correctly your Glyph_cfg.h should look like this

```
#define USE_GLYPH_FONT_BITMAP
//#define USE_GLYPH_FONT_HELVR10
//#define USE_GLYPH_FONT_8_BY_16
#define USE_GLYPH_FONT_8_BY_8
//#define USE_GLYPH_FONT_WINFREE
//#define USE_GLYPH_FONT_5_BY_7
//#define USE_GLYPH_FONT_6_BY_13

#define USE_DEFAULT_FONT Bitmaps_table
//#define USE_DEFAULT_FONT FontHelvr10_table
//#define USE_DEFAULT_FONT Font8x16_table
#define USE_DEFAULT_FONT Font8x8_table
//#define USE_DEFAULT_FONT FontWinFreeSystem14x16_table
//#define USE_DEFAULT_FONT Fontx5x7_table
//#define USE_DEFAULT_FONT Fontx6x13_table
//#define USE_DEFAULT_FONT FontHelvr10_table
```

At this point you need to compile the code and ensure that you get no complier errors, if you do get complier errors then chances are you are missing a file or skipped a step.

Once your code complies, you also need to add an additional file to your project that is located in the RPDL folder which should be a subfolder in the directory where you copied all the Glyph files in the prior step. A typical location could be C:\WorkSpace\Lab2\Lab2\RPDL

The file you want to add to your project is

1) interrupt_adc_12.c

After you add this file to your project you will need to manually drag the file on the projects tab to the RPDL folder from the C source file folder.

If you did this correctly interrupt_adc_12.c should appear in the RPDL folder beneath the Interrupt_ADC_10.c file on the HEW project tab.

Once you have completed this step, compile your code again to ensure everything is in working order.

At this point you should be ready to begin the lab and work on creating your first embedded game!

Some Helpful Advice:

Note is recommended that you look over the datasheet for the ST7579 found at
http://www.tianma.com/web/uploads/controller/20080316012510_ST7579_V0.9a.pdf

And you should also look over the documentation from Renesas about Glyph found at
http://www.renesasrulz.com/servlet/JiveServlet/previewBody/1685-102-1-1614/Generic%20API%20for%20Graphics%20LCD%20v1.00.pdf

Prelab Questions:

1. Where should your “working directory” be located when using the lab computers?
2. Where is the first place to look for help with labs in this class?
3. What is the width and length in pixels of the LCD on the Renesas RX dev board?
4. What is Glyph and what is it used for?

Laboratory Assignment Overview:



Embedded Homage to Emergency!

"What is squad 51 and why does it keep calling Rampart"

- Radio Operator Spring 1972

Emergency!, is a American TV classic that came out in 1972 that depicts the adventures of two paramedics (John Gage and Roy DeSoto) at the birth of the Los Angeles paramedic program. While this show, for the most part, is forgotten by the masses, it helped to establish the idea that a paramedic program would be beneficial to a local community's wellbeing and also helped promote the idea that public training classes on CPR could be beneficial in saving lives. With this in mind there is a possibility that without this show's public influence we might not have a paramedic program in charlotte or the 911 emergency number today and thus this lab will pay homage to emergency! by having you create an emergency! themed game!

Highlights:

Custom graphics for a game title screen will be loaded and displayed at startup while all the Red LEDs on the board will blink at an approximately 100ms interval until any of the 3 switches are pressed.

After which the LEDs will blink in a clockwise pattern at a blink rate of approximately 200ms interval until any of the 3 switches are pressed. Additionally you will also render text to the LCD screen with your name somewhere on the screen and a custom LCD graphics either one that is your own design or an existing custom graph in the existing bitmap font.

When the game is un-paused the custom graphics for the game world will be loaded and displayed, the red LEDs will flash in an wig-wag pattern truck (left flash, left flash, right flash right flash) which is similar to those found on a fire truck and the road that will be rendered will move at a speed defined by the potentiometer. Pressing the SW1 button will make the rescues squad truck you control in the game go up while pressing the SW2 button will make the rescues squad truck you control in the game go down.

You are expected to expand upon this basic game framework once you get this working to make the game more complete.

Laboratory Assignment:

1. Do the prelab activity described above.
2. In the project editor, choose main.c as the file to edit. As you can see the new project is rather empty and no support headers have been provided. You will need to #include the following files in order to complete your project.
   i) r_pdl_tmr.h
   ii) r_pdl_cmt.h
   iii) r_pdl_spi.h
   iv) r_pdl_io_port.h
   v) r_pdl_adc_12.h
   vi) r_pdl_definitions.h
   vii) YRDKRX62N.h
   viii) switch.h
   ix) lcd.h

   In our last assignment we did not have such a massive number of includes in main.c because these files were included in other .c files. This point illustrates one of the many benefits of creating code classes. However, in this assignment we will avoid putting our code into other .c files and stick with main.c for the time.

   It should be noted that each one of the r_pdl prefix files is part of the board support package and these files can be rather fussy about the order they are included. The order given above is known to correctly compile, however other orders are possible and some of them could result in complier errors so I recommend that you play around with this notion and see what works and what doesn't work to help you gain a feeling for HEW complier RX complier.

   Note be sure your code compiles before going to the next step.

3. At this point we want to initialize the LCD screen in the main function prior to the main loop
   Tip: you should be able to figure this out by looking at the lab 1 assignment's main loop.
   Hint: (YRDKRX62N_RSPI_Init and InitialiseLCD) are required.

   Note, if you look into what some of these functions are doing, you might become somewhat perplexed that SPI is used to communicate with the LCD display and that is perfectly ok for this lab since we will be using the powerful concept of abstraction to easily communicate with peripheral using encapsulated classes that are relatively straight forward to use.

   Note, not knowing what a function is doing and how it works is not always a good thing, but it is oftentimes easier to figure out how something works after seeing it work.

Be sure to test that your LCD initializing code is working by writing a simple Hello World to the screen before continuing, note be sure to comment the test Hello World afterwards before continuing.

4. Next, somewhere in the main.c you need to create a function called Reset_All_LEDs that will turn all of the ring LEDS off

   Note this function should be similar to the code you created in lab 1.

   Tip: be sure you put a function prototype below the #include at the top of the file to avoid complier errors.

5. Next, somewhere in the main.c you need to create a function called All_Red_LEDs_ON that will turn all of the red ring LEDs on

   At this point you should quickly compile and test that all your red ring LEDs turn on when this function is complied.

   Tip: be sure you put a function prototype below the #include at the top of the file to avoid complier errors.

6. Next, somewhere in the main.c you need to create a function called Block_Until_Switch_Press that will enter an endless while loop until one of the 3 switches on the board is pressed

   Note this function should be similar to the code you created in lab 1.

   Tip: be sure you put a function prototype below the #include at the top of the file to avoid complier errors.

7. Next, somewhere in the main.c you need to create a function called Blink_Red_LEDs that will use a global variable to keep track of the last LED state and either call the function All_Red_LEDs_ON or Reset_All_LEDs depending upon its state to toggle the Red LEDS on or off.

   Tip: be sure you put a function prototype below the #include at the top of the file to avoid complier errors.

8. Next we want to use the periodic timer code provided in Lab1 to flash all the red LEDs on and off at a set rate of 2 seconds until one of the three switches is pressed.

The first lab did not go into great detail about the mechanism behind how this can be accomplished and this lab will only discuses how to setup this mechanism, but not why this mechanism works. If you are really curious I recommend reading about interrupts.

In general the functions R_CMT_Create and R_CMT_Destroy can be used to registers and unregister periodic timer events that will cause a user defined function to run periodically every x number of seconds.

For example from lab1

R_CMT_Create (3, PDL_CMT_PERIOD, 50E-3, CB_CMTFlash,2);

will result in the function CB_CMTFlash being called every 50ms

likewise the function

R_CMT_Destroy(1);

Causes CMTFlash to stop being called every 50ms

For this lab, in the main function, after you have initialized your LCD use the R_CMT_Create to call the Blink_Red_LEDs function every 100ms seconds

Note you should change CB_CMTFlash to the function you want to call

Note functions called by R_CMT_Create are assumed to have no input parameters.

After you register your Blink_Red_LEDs function use the Block_Until_Switch_Press to prevent further code execution until a switch is pressed.

Note the Blink_Red_LEDs function will continue to be called every 100 ms regardless of the endless loop inside of Block_Until_Switch_Press. This is a concept that will be referred to as interrupts and the theory behind how it works will be disused in more detail near the end of the semester.

After the Block_Until_Switch_Press you will need to disable to the Blink_Red_LEDs function using the R_CMT_Destroy command, ensure all the LEDS get cut off, and the blink status variable gets to the off condition.

At this point you should compile your code and ensure that your board blinks its red LEDs every 100ms seconds until one of the buttons is pressed and after which all the LEDS turn off.

9. Next, somewhere in the main.c you need to create a function that will cause all of the red LEDs pairs to blink in a wig-wag pattern (left red, left red, right red, right red) Note all red led pairs are separated by green LEDS in the 12 segment LED ring.

10. Note the procedure for this step is the very similar to the procedure for the last step. You should registers your new function with R_CMT_Create at a rate of 200ms, wait until a button is pressed with Block_Until_Switch_Press and stop the periodic function with R_CMT_Destroy and turn off all LEDS

    At this point you should compile your code and ensure that your board blinks its red LEDs every 100ms seconds until one of the buttons is pressed and after which all the LEDS turn off and your blinking LED pattern runs at a slower rate of 200ms until one of the buttons is pressed and after which all the LEDS turn off.

11. Next, somewhere in the main.c you need to create a function called Red_LEDs_Flash that will use a global variable to keep track of the last LED state and toggle between LED15,LED6,LED10 being on and LED7,LED11,LED14 being on.

12. Next change the while (1) ; to while(1) { } and add code inside the while loop to detect when switch 3 is pressed. Because switch 3 is going to toggle between game paused and game run you should create a local variable to keep track of the status of the switch (paused/ un-paused)

    When switch 3 is pressed a toggle event will occur, one state (un-paused) will use the R_CMT_Create to call the Red_LEDs_Flash function every 100ms while the other state (pause) will call R_CMT_Destroy and cut off all LEDs.

    Compile and test your code to ensure that the steps above work correctly, and that you are able to toggle between fast alternating red LEDs and no LEDs by pressing switch 3.

13. At this point we are ready to start modifying the LCD code. Open up lcd.c and in the function InitialiseLCD add

    ```
    GlyphWrite(G_lcd, GLYPH_FRAME_RATE, 137);
    GlyphWrite(G_lcd, GLYPH_CONTRAST, 255);
    ```

    Inside the if statement before the

    ```
    GlyphNormalScreen(G_lcd)
    ```

    Command.

14. In lcd.c create a new function called Set_Font_8_by_8() that has the following function call inside

    ```
    GlyphSetFont(G_lcd, GLYPH_FONT_8_BY_8)
    ```

    Note be sure to add a function prototype in lcd.h for Set_Font_8_by_8()

15. In lcd.c create a new function called Set_Font_Bitmap() that has the following function call inside

    GlyphSetFont(G_lcd, GLYPH_FONT_BITMAP)

    Note be sure to add a function prototype in lcd.h for Set_Font_Bitmap ()

16. In lcd.c create a new function called Set_LCD_Pos(int x, int y) that has the following function calls inside

    GlyphSetX(G_lcd,x)
    GlyphSetY(G_lcd,y)

    Note be sure to add a function prototype in lcd.h for Set_LCD_Pos(int x, int y)

17. In lcd.c create a new function called Set_LCD_Char(char value) that has the following function call inside

    GlyphChar(G_lcd,value)

    Note be sure to add a function prototype in lcd.h for Set_LCD_Char (int x, int y)

    At this point ensure your code complies and continue on to the next step.

18. At this point open font_bitmap.c

    This file contains all the binary information for custom LCD characters that are loaded onto the screen.

    The data is stored in an array in the following format

```
const uint8_t Custom_Grahpic_Name[] = {
        Width in pixels, Height in pixels, // width=???, height???
         data1,data2,data3,data4,data5,data6,…….
};
```

    Each data byte holds column information while each array element holds row information so for example if this grid was a LCD screen the above code would produce

Data1.b0	Data2.b0	Data3.b0	Data4.b0	Data5.b0	Data6.b0
Data1.b1	Data2.b1	Data3.b1	Data4.b1	Data5.b1	Data6.b1
Data1.b2	Data2.b2	Data3.b2	Data4.b2	Data5.b2	Data6.b2
Data1.b3	Data2.b3	Data3.b3	Data4.b3	Data5.b3	Data6.b3
Data1.b4	Data2.b4	Data3.b4	Data4.b4	Data5.b4	Data6.b4
Data1.b5	Data2.b5	Data3.b5	Data4.b5	Data5.b5	Data6.b5
Data1.b6	Data2.b6	Data3.b6	Data4.b6	Data5.b6	Data6.b6
Data1.b7	Data2.b7	Data3.b7	Data4.b7	Data5.b7	Data6.b7

If Data1=0xFF then a column of 8 pixels on the LCD screen would be dark

It is a relatively straightforward process to create small LCD graphics on a piece of paper and converting the binary numbers into hex and entering them into the custom font above.

however, Large custom LCD graphics can be made painstakingly by hand using this method, however a python script is available that will convert a bitmap image into LCD format and this script is available thru the TA, so if you are interested in making your own LCD graphics feel free to inquire about this.

There are also open source software and even web site applications that can also be used to generate LCD graphics, however the binary encoding LCD information they produce might not match the required formatted needed by the LCD font system our board supports 100% and you might have the tweak this format by hand to make it work.

In font_bitmap.c find Bitmaps_table[] near the end of the file

download the lcd.c file available from the embedded systems website and above the Bitmaps_table[] definition paste the contents of lcd.c into font_bitmap.c

19. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar0 with Logo
20. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar1 with sq1
21. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar2 with road

At this point make sure your project still complies

22. At this point open up main.c again, and before the first Block_Until_Switch_Press function call, call the functions

```
Set_Font_Bitmap();
Set_LCD_Pos(0,0);
Set_LCD_Char(0);
```

    Compile your code and observe what is displayed on the LCD

    Note that the Set_Font_Bitmap() was used to gain access to our custom LCD characters and that Set_LCD_Char was used to write character 0 which happens to be the first item in the Bitmaps_table[] which as we now know is the very large emergency! Logo.

    The Set_LCD_Pos allows us to draw our custom characters anywhere on the screen with two minor exceptions,

    I) You can't draw something outside the size of the screen
    II) The y axis will only move in increments of 8 numbers less than 8 get rounded down to the nearest 8 value (this is a LCD hardware issue we will address in future labs)

    Hopeful at this point you have a good idea about how to display custom LCD characters on the screen and will experiment with the rendering process.

23. Modify main.c again such that the LCD screen is cleared after the first Block_Until_Switch_Press function is called and change the default font back to 8by8 using the Set_Font_8_by_8 function created earlier. Once this is done, write your name to the screen and if you feel so inclined you may create your own custom LCD characters and render them to the screen here as well.

    Compile your code and observe its operation at this point you should see a LCD logo of Emergency and all the red leds blinking every 100ms, pressing any switch results in the LCD screen being cleared and your name appearing on the screen while the red leds blink every 200ms in a wig-wag pattern. Pressing any of the three switches again results in the pause or un-pause mode of operation where pressing switch 3 will result in a 3 red LED flashing pattern.

24. Modify main.c again such that the LCD screen is cleared after the second Block_Until_Switch_Press function is called and also reset your LCD font back to bitmap mode via Set_Font_Bitmap.

    Compile your code and observe the results

25. At the top of main.c create a new global int called draw, make sure it is defined as volatile
    Example volatile int draw=1;

We are going to use this variable to allow functions to tell our application to redraw the LCD screen as part of our game.

26. Inside the main function create 2 int variables to keep track of the current values of x and y for our game vehicle.
27. Inside the main loop create an if statement that will check for draw==1 and reset the draw flag inside this if statement.

    Inside this function you should check to see if the game is paused, if the game is paused then you should change your font to Set_Font_8_by_8 and print on LCD line 1 Paused and then change your font back to Set_Font_Bitmap

    If the game is not paused you should clear the LCD screen

    Compile your code and observe the operation of the code on the board.

28. When switch 3 is pressed the draw flag should be set to 1
    Compile your code and observe the operation of the code on the board. Your LCD should now toggle between pause and blank when you press the pause button

29. Inside the main loop in the draw un-paused routine, after you clear the LCD set the position of the LCD to the X and Y variable you defined to hold the game vehicle current X and Y location and then use

    Set_LCD_Char(1);

    To draw the vehicle graphic to the screen.

    Compile your code and observe the operation of the code on the board.

30. In the main loop write an if statement that will detect when switch 1 Is pressed, if switch 1 is pressed then the vehicle Y counter should me decremented by 8 and value of Y less than 0 should force the Y value to equal 0

    Note You should update the draw=1 flag in this function

    Note this function should not change the Y counter if the game is paused

31. In the main loop write an if statement that will detect when switch 2 Is pressed, if switch 2 is pressed then the vehicle Y counter should me incremented by 8 and value of Y greater than 0x28 should force the Y value to equal 0x28

    Note You should update the draw=1 flag in this function
    Note this function should not change the Y counter if the game is paused

Compile your code and observe the operation of the code on the board. Your truck should move up and down on the screen now when you press a button.

32. At the top of main.c add a global volatile int variable called shift to keep track of the roadway movement

33. Inside the main loop in the draw un-paused routine, before the truck is drawn add the following code

```
for(lp=1;lp<95;lp+=10)
{
        if(lp-shift<1)
        {
                continue;
        }
        Set_LCD_Pos(lp-shift,0);
        Set_LCD_Char(2);

        Set_LCD_Pos(lp-shift,0x20);
        Set_LCD_Char(2);
}
```

You will have to add a local variable to the main function to make this function work.
Compile your code and observe the operation of the code on the board. You should now have roadway lines.

34. Similar to Lab1 we are now going to use the ADC to control a timer which will make the roadway move at an adjustable rate to do this we need to register the ADC and timer events in the main function.

Before the while (1) loop you will need to add

```
R_ADC_12_Create(0,PDL_ADC_12_SCAN_SINGLE|PDL_ADC_12_CHANNEL_4|PDL_ADC_12_DATA_ALIGNMENT_RIGHT|PDL_ADC_12_DIV_1,
PDL_ADC_12_TRIGGER_TMR0,PDL_NO_DATA,ADC_Conversion_Done,6);

R_TMR_CreatePeriodic(PDL_TMR_UNIT0,PDL_TMR_PERIOD|PDL_TMR_ADC_TRIGGER_ON,GameSpeed,5E-3,PDL_NO_FUNC,PDL_NO_FUNC,0);
```

35. You will also need the function ADC_Conversion_Done and a global float variable GameSpeed to make this code compile, the function ADC_Conversion_Done is defined as the following

```
void ADC_Conversion_Done()
{
        uint16_t adc_results[8];
        R_ADC_12_Read(0,adc_results);
        GameSpeed = ((adc_results[4]>>2) * 2E-4) + 35E-3;
        shift+=5;
        if(shift>15)
        {
                shift=0;
        }
        draw=1;
   R_TMR_ControlPeriodic(PDL_TMR_UNIT0,PDL_TMR_PERIOD,GameSpeed,5E-3);
}
```

Compile the code, and run. At this point you should have a speeding truck that can change lanes.

36. To add a unique spin onto this project, change the roadway const uint8_t road[] to another custom LCD character of your own design. Additionally, you may freely modify and edit this example to complete the game as long as the base function remains intact.
37. Complete your lab report.
38. Bring the 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	All Red LEDs blink at 100ms rate and EMERGENCY! LCD graphic loads up on boot	
2	Pressing any Switch makes the red LEDs blink at a 200ms rate wig-wag pattern and the LCD screen says the names of people in the group	
3	Pressing any switch again makes the LEDs cut off and the word paused displayed on the LCD screen	
4	Pressing switch 3 un-pauses the game, the correct red LEDs toggle (standard light bar style), squad 51 is correctly drawn on the screen, and the roadway moves with the ADC value	
5	Pressing switch 1 makes squad 51 move up	
6	Pressing switch 2 makes squad 51 move down	
7	Group added their own unique spin to the project by changing the roadway LCD graphic or adding more functionally to the game.	

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 . . . .