ECGR 4101/5101, Fall 2013: Lab 3

Learning by Example: Your 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 RX63N development board along with teach you how to create your own RX63N project from scratch.

General Information:

1. Review the steps provided by the lab 3 supplemental information document
2. Create a new workspace YRDKRX63N
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:

To begin this lab, follow the steps of the "Lab Supplemental Information" document provided on Dr. Conrad's website, if you are unfamiliar as to how you create a project for the RX63N or you want to confirm functionality of your board and compiler. We will modify the tutorial application for this lab. Be sure to name your workspace "Lab3."

Once you have created your new Lab3 workspace you will need to open up windows explore and navigate to the **C:\workspace/lab3/lab3/r_glyph/src/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.

You will need to ensure the following files exist:

1) drivers/ST7579_LCD.c
2) drivers/ST7579_LCD.h
3) fonts/bitmap_font.c
4) fonts/font_8x8.c
5) Glyph.c
6) Glyph.h
7) config.h

If any of these files are missing, they can be found in the **C:\Program Files (x86)\Renesas** folder.

You will also have to add these files to your project. You can do that by right clicking any file in the project and selecting "add files". Choose the missing files from the **C:\workspace/lab3/lab3/r_glyph/src/glyph** folder to add to the project. Make sure that they are placed in appropriate sections in the project.

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

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

If you did this correctly your config.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
```

You will also need to edit glyph.h by uncommenting and adding “extern”:

```
extern const uint8_t * Bitmaps_table[256];
```

After uncommenting, you will need to add bitmap_font.c to your project. Do this by right clicking on one of the files in your project and selecting “Add Files…”. Then, navigate to r_glyph/src/glyph/fonts/ , select bitmap_font.c, and click “Add.”

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.

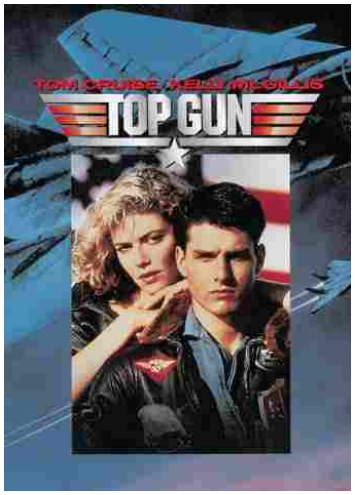
At this point you should be ready to begin the lab and work on creating your first embedded game!

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

Laboratory Assignment Overview:



Embedded Top Gun Game!

Top Gun has had many terrible, low resolution video games created in its image, and today, we shall add to that number by creating a very crude dogfighting game in which you will guide a jet through the air while dodging enemy jets!

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 approximately 400ms interval until any of the 3 switches are pressed.

The LEDs on the board will blink in any pattern you want at a blink rate of approximately 200ms interval until any of the 3 switches are pressed. Additionally you will also render LCD text with your name somewhere on the screen or your own custom graphics if so desired.

The game will now be in the paused mode and all the RED LEDs are turned on until SW3 is pressed.

When the game is un-paused the custom graphics for the game world will be loaded and displayed, the all LEDs are turned off. A jet image will be rendered and enemy jets images will be rendered that fly towards your jet at a speed determined by the board's potentiometer value. Pressing the SW1 button will make the jet you control in the game go up while pressing the SW2 button will make the jet you control in the game go down.

Laboratory Assignment:

1. Do the prelab activity described above.

2. In the project editor, choose tutorial_main.c as the file to edit.
   Be sure to test that your LCD by editing the lcd_display(LCD_LINE1, " RENESAS   "); to display "Hello World."

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

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

4. Next, somewhere in the tutorial_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.

5. Next, somewhere in the tutorial_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: Add a software debouncing code in this function.

   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 tutorial_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.

7. 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 400 ms until one of the three switches is pressed.

   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 cmt_callback_set cmt_start and cmt_stop 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. You can find these functions in cmt_periodic_multi.c file. Currently these functions are being used in the flash_led.c file.

   For this lab, in the main function, after you have initialized your LCD, use the cmt_callback_set and cmt_start functions to call the Blink_Red_LEDs function every 400ms seconds

   Observe how these functions are being used in flash_led.c file and use a modified version of these functions in the main.c file to blink the RED LEDs every 400ms.

   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 400 ms regardless of the endless loop inside of Block_Until_Switch_Press.

   After the Block_Until_Switch_Press you will need to disable to the Blink_Red_LEDs function using the cmt_stop function, 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 400ms seconds until one of the buttons is pressed and after which all the LEDS turn off.

8. Next, somewhere in the tutorial_main.c you need to create a function that will cause the LEDs to blink at in any pattern you desire (you could even use the train moving pattern from lab 2 other than blinking or alternating Red).

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

   At this point you should compile your code and ensure that your board blinks its red LEDs every 400ms seconds until one of the buttons is pressed and after which your blinking LED pattern runs at a faster rate of 200ms until one of the buttons is pressed and after which all the LEDS turn off.

10. Next in the while(1) { } loop, add code to detect when switch 3 is pressed. Because switch 3 is going to toggle between "game paused" and "game run" you should create a global 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 (paused) will use the All_Red_LEDs_On function while the other state (un-paused) will call Reset_All_LEDs 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.

    Tip: Do not forget to add software debouncing to SW3.

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

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

    Inside the if statement before the

    ```
    GlyphNormalScreen(Lcd_handle)
    ```

    Command.

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

    ```
    GlyphSetFont(Lcd_handle, GLYPH_FONT_8_BY_8)
    ```

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

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

    ```
    GlyphSetFont(Lcd_handle, GLYPH_FONT_BITMAP)
    ```

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

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

    ```
    GlyphSetX(Lcd_handle,x)
    GlyphSetY(Lcd_handle,y)
    ```

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

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

    GlyphChar(Lcd_handle,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.

16. At this point open bitmap_font.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.

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

Above Bitmaps_table[] definition, copy and paste the LCD information found in the file lcd.c on the course lab webpage.

17. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar0 with Logo
18. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar1 with explode
19. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar2 with jet0
20. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar3 with jet1
21. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar4 with jet2
22. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar5 with jet3
23. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar6 with jet4
24. Next modify Bitmaps_table[] and replace Bitmaps_VolumeBar7 with jet5

    At this point make sure your project still complies

25. 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 Top Gun 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

    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.

26. 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 Top Gun and all the red LEDs blinking every 400ms, pressing any switch results in the LCD screen being cleared and your name appearing on the screen while LEDS of your choosing are blinking every 200ms. Pressing any of the three switches again results in the pause or un-pause mode of operation where pressing switch 3 will result in 3 red LED flashing pattern.

27. Modify main.c again such that the LCD screen is cleared after the second Block_Until_Switch_Press function is called.

    Modify the part of the while loop where you are checking if SW3 is pressed. You are checking if the game is paused. When switch 3 is pressed the pause flag should toggle.

    If the game is paused then you should change your font to Set_Font_8_by_8 and print on LCD line 1 Paused. You will clear the LCD screen and change your font back to Set_Font_Bitmap when the game is unpaused.

    Compile your code and observe the results

28. Create 2 more global int variables to keep track of the current values of x and y for our game obstacle.

29. Create another 2 global int variable to keep track of our jet animation and initialize the value to 0.

30. Inside the main loop in the 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 jet's current X and Y location and then use Set_LCD_Char(3);

    To draw the jet graphic to the screen.

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

31. Create a new function called Create_Obstacle() somewhere in main.c file.

32. Note be sure to add a function prototype in timer_Adc.h file.

33. In this function write a routine that will detect when switch 1 Is pressed, if switch 1 is pressed then the vehicle Y counter should be decremented by 8 and value of Y less than 0 should force the Y value to equal 0

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

34. In this function write another routine 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 0x32 should force the Y value to equal 0x32

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

    Call this function inside the while(1) loop after checking if SW3 is pressed.

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

    To get a clearer observation add a software debouncing routine for both the switches. These debouncing routines will have to be removed for the next part of the project.

35. If you followed all these steps carefully and observed all the changes as expected, then comment out the Create_Obstacle function call in the while(1) loop and comment out the software debouncing code for SW1 and SW2 inside the Create_Obstacle function.
    Note: Do not remove software debouncing code from any other part of the project except for the ones called in the Create_Obstacle function.

36. Inside the Create_Obstacle() function we will need to add game obstacles, which will be enemy jets spawning. To do this we will need to add several integer variables to the top of our main function:

    bool enemy_flag = 0; //variable for creating enemy plane
    bool hit_flag = 0; //variable for tracking hits and hit animation

    When "enemy_flag" is set to 0, it will indicate that a new enemy jet is to be created. The "hit_flag" variable will indicate when your jet has collided with the enemy jet. Add the following code to your main loop to evaluate jet collisions and spawning.

```
if(!enemy_flag)
{       //create enemy plane
        valx_1 = 80;  //spawn on the other side of the screen
        valy_1 = 8*(rand() % 8);

        enemy_flag = 1;

}
else
{
        if(valx_1 <= 23)  //check for collision
```

```
                {
                        if((valy_1 <= (valy + 16)) && (valy_1 >= valy))
                        {
                                enemy_flag = 0;//respawn enemy
                                hit_flag = 1;
                        }
                }
                if(valx_1 == 0)
                        enemy_flag = 0;
                else
                {
                        valx_1 = valx_1 - 5;
                        /******************
                        Include the code provided on next page here
                        ******************/
                }
        }
```

You will add the hit animation in place of the commented lines on the previous page, which will be represented by an explosion graphic, we will need to update our section of the code which draws the jet. The player's jet should be drawn after the lines of code presented above, and should look something like this:

```
                        if(hit_flag == 0)
                        {
                                Set_LCD_Pos(valx_1, valy_1);
                                Set_LCD_Char(7);        //draw jet
                        }
                        else
                        {
                                Set_LCD_Pos(valx_1,valy_1);
                                Set_LCD_Char(7);
                                Set_LCD_Pos(valx, valy);
                                Set_LCD_Char(1);
                                hit_flag = 1;
                        }
```

Where the variables valx_1, valy_1, valx and valy should be replaced by the your obstacle x, y variable; and player's jet x,y variables, respectively.

37. 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

Timer_adc() function.

38. You will also have to change the toggle_led() function to Create_Obstacle() function in the cmt_callback_readADC() function in timer_adc.c function.

    Compile the code, and run. At this point you should have a jet flying through the air, which you can dodge oncoming enemy jets, and explode when you run into them!

39. Complete your lab report.
40. Bring the new board to the lab TA and demonstrate the new code (without the HEW application running). When the TA checks your board, he will also take your lab checkout sheet.
41. Upload your lab report to Moodle. 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 400ms rate and Top Gun LCD graphic loads up on boot	
2	Pressing any Switch makes the LEDs blink at a 200ms rate at user defined pattern and 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, jet is correctly drawn on the screen, and the roadway moves with the ADC value	
5	Pressing switch 1 makes jet move up	
6	Pressing switch 2 makes jet move down	
7	Enemy jets spawn randomly on the screen.	