Fall 2015

ECGR 4101/5101, Fall 2015: Lab 7

Digital Oscilloscope

Objective:

In this lab, we will create a digital oscilloscope using the RX63N's timers and the ADC. Analog voltage values ranging from 0 to 3.3 will be displayed in a scrolling fashion as found on oscilloscopes, which shows the most recent reading on the leftmost side and previous readings scroll across the screen to the right.

Assignment:

1. Begin by creating the tutorial project workspace under the YRDKRX63N Simple Demos option. Consult the supplemental instruction from lab 2 if you need help setting up the workspace.
2. Once we have created a tutorial workspace, we will begin by removing most of the functions in tutorial_main.c.
    a. Remove flash_led(); from main()
    b. Remove timer_adc() from main()
    c. Remove statics_test() from main()
3. We are going to need to write custom graphics to the screen, so we need to prepare functions that allow us to set individual pixels on the LCD. We will do this by setting the LCD front to bitmap and writing one large, square character to the entire screen and setting the individual bits within the character array.
    a. Begin by adding bitmap_font.c to your workspace. When you create the tutorial workspace, this file is created for you, but not included in the project. It is located in "your workspace root dirctory"/r_glyph/src/glyph/fonts/.

    b. Then, 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
```

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

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

d. 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.

e. 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()

f. 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 ()

g. 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)

h. 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.

i. 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.

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

Above Bitmaps_table[] definition, copy and paste the following array:

```
uint8_t Screen[] = {

        0x60,0x44, // width = 96, height = 68

        0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0
x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
```

0,0x0,0x0,0x

```
0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0
x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,

0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0
x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0

};
```

Once you have added Screen[] to your bitmap_font.c file, edit the Bitmaps_table[256] array to include Screen. Replace Bitmaps_VolumeBar0 with Screen. This way we can call Set_LCD_Char(0) to pull up our Screen character.

j. Screen[] provides us a blank array to write pixels to. We will also create 2 more functions for setting individual pixels and clearing the screen. Add the following functions to your lcd.c file and don't forget to add function prototypes to your lcd.h file.

```
void Clear_Screen(void)
{
        int i = 2;
        for(i = 2; i <= (2+(96*8)); i++){
                Screen[i] = Screen[i] & 0x00; }
}

void LCD_Set_Pixel(int x, int y)
{
        int xIndex = 0, yIndex =0, shiftby = 0;

        if(x >= 96 || y >= 64 || x < 0 || y < 0) return ;
        else{
                yIndex = y/8;
                shiftby = y - (yIndex*8);
                xIndex = 2 + (yIndex*96) + x;

                Screen[xIndex] = Screen[xIndex] | (1 << shiftby);
        }
}
```

Clear_Screen() writes all values of the Screen[] array to 0, except for the first 2 positions, because they hold the character size. LCD_Set_Pixel will set a pixel in the specified X,Y position on the screen.

k. Inside of lcd.c, we will need to add

```
extern uint8_t Screen[];
```

at the top of the file so that Clear_Screen() and LCD_Set_Pixel() can access the external variable.

4. Now, we need to set up one of the timers to refresh the display. The period configured on the timer will determine how often the signal is updated on the screen. To do this, we can use the following functions:

```
cmt_init()
cmt_callback_set();
cmt_start();
```

These functions are described in the cmt_periodic_multi.c file, so look in that file to find more details. Our end goal is to use the timer's interrupt to refresh the screen and display an image of the updated signal.

5. Once you have configured your timer, inside the timer's interrupt service routine, we need to read the signal from an analog input pin and display it on the LCD. Feel free to use your customADC library from the previous lab to read the voltage. Once you have read the voltage, you will need to store it somewhere and display it on the screen. When displaying the voltage on the LCD, allow the very top pixel row to represent the maximum voltage (in our case, 3.3 volts), and the very bottom row to represent the minimum voltage (0 volts). Remember, the top left corner of the LCD is position (0,0), and the LCD is 68x96 pixels, so row 68 will represent the minimum value. All voltages in between should be properly scaled. Because the screen is 96 pixels wide, we will be storing the last 96 values read. Once we have 96 values stored, as new values are read, all of the past values are shifted to the right by 1 pixel, and the very last value will be dropped.

6. Now that the signal can be displayed on the screen, add functionality such that pressing switch 1 will cause the current voltage to be displayed on the screen. Pressing switch 1 again will toggle it off. The signal should continue to be printed simultaneously while the voltage is being written to the screen.

7. Add functionality such that pressing switch 2 will toggle transmission of the latest voltage through the RS-232 port on the board on and off. The printing of screen and/or voltage value should still be functional while streaming data from the serial port to a PC. Feel free to use your code from the UART lab to transmit the data.

8. Add functionality such that pressing switch 3 will toggle the analog channel being read from channel 2 to another channel of your choosing which can be interfaced with an

external voltage source. When you change channels, clear the previous voltage values and erase the previously displayed signal on the LCD.

To Submit:

- A zip file containing the workspace of your project
- Your lab check-off sheet at the demonstration

Fall 2015

Embedded Systems Lab Demonstration Validation Sheet

This sheet should be modified by the student to reflect the current lab assignment being demonstrated

Lab Number:	Lab 7 – Digital Oscilloscope
Team Members	Team Member 1: Team Member 2:
Date:	

Lab Requirements

Obtain a list of the Lab requirements from the end of the lab handout and type them here, perform a self-review and indicate with an X if you met each requirement or not.

REQ Number	Objective	Self-Review	TA Review
1	The LCD correctly displays the scrolling analog voltage.		
2	Switch 1 toggles the display of the analog voltage value. The scrolling analog voltage should continue while the number is being displayed.		
3	Switch 2 toggles UART streaming of the current analog value through the RS-232 port.		
4	Switch 3 toggles between analog channel 2 and some external channel. Channel 2 and the external channel are properly read.		

Code Requirements (will not be graded during lab demo)

REQ Number	Objective	Self-Review	TA Review
5	All code must be commented and indented properly.		