Lab 9. Displaying new characters on the LCD glass

Introduction

We have learned in the class that we can use the LCD glass on the STM32L476 Discovery Kit to display a string by calling the BSP (Board Support Package) API (Application Program Interface) provided by the manufacture of the Board. A demo project, lab09_lcd_uart_with_bsp_2018 (lab Lab09 of a previous semester), which illustrates the usage of the BSP API, was posted on Canvas.

This lab is an extension of the above demo project. You can still team up with another student in the lab to discuss and finish the work, but you **need to** submit your **own code and lab report** since the code and report contain results which are specific to each student.

Lab tasks

There are four Tasks for this lab.

Task 1. Building the base project of the lab

(20 points) We build this project based on the demo project. There are two approaches to building the base project of the lab to support the subsequent tasks. The first approach is to use the .ioc file of the demo project to regenerate the files for the lab project. The second is to copy the corresponding files from the demo project and do the necessary modifications.

Both approaches work fine. We have practiced a couple of times using the first approach. Here, we provide the instructions for the second one. First, we do the following preparations to reproduce the demo project in a new folder with changed folders and files.

1. Make a new folder named `lab09_new_LCD_chars` to host the project.
2. Copy all the following four subfolders from the demo project folders to the new one: `Drivers`, `Inc`, `MDK-ARM`, and `Src`.
3. CD (Change Directory) to the new `MDK-ARM` folder and do the following changes:
    a. Remove all the three subfolders.
    b. Change the name of the two project management files to-`lab09_new_LCD_chars.uvoptx` and `lab09_new_LCD_chars.uvprojx`, respectively.
4. CD to the `Drivers/BSP` folder and change the file names of `stm32l476g_discovery_glass_lcd.c` and `stm32l476g_discovery_glass_lcd.h` to `stm32l476g_discovery_glass_lcd_new_chars.c` and `stm32l476g_discovery_glass_lcd_new_chars.`, respectively. (We don't want to make changes to the files of the provided BSP driver directly.)
5. Open the project in Keil-MDK by double-clicking `lab09_new_LCD_chars.uvprojx` and do the following changes.
    a. Update the BSP driver file. Remove `stm32l476g_discovery_glass_lcd.c` from the project and add `stm32l476g_discovery_glass_lcd_new_chars.c` to the project.
    b. Update the include file in the above BSP driver file as well as the `main.c` file. Open `stm32l476g_discovery_glass_lcd_new_chars.c` and and `main.c` files and change `stm32l476g_discovery_glass_lcd.h` to `stm32l476g_discovery_glass_lcd_new_chars.`.

6. Update the output folder and file.
    a. Click the **Options for Target** icon to open the **Options for Target 'lab09_lcd_uart_with_bsp'** window. In this window, click the **Output** tab.
    b. In the new tab, Change the **Name of the Executable** to `lab09_new_LCD_chars`.
    c. Also in the tab, click **Select Folder for Objects…** and create a new `lab09_lcd_new_chars` folder under `MDK-ARM`. (Or you can rename the auto-created `lab09_lcd_uart_with_bsp` folder to `lab09_lcd_new_chars`.) Then, CD to the new folder so it will be used as the folder for the build files. Make sure the display is as the following figure.
7. Remove all the breakpoints in the code and build the project. There should be no error.
8. Load the code to the board to verify the code indeed works. Note that due to the programming of Scrolling Display of Sentences, the board is not very responsive to the keys. To verify is the build is correct, you need to connect the PuTTY terminal to the board to see the messages sent by the board. To do so, you need to use the Windows Device Manager to find the Port number of the board and use it in PuTTy. If you see "This is the start of the Lab 9 program…" then the porting of the project works fine.

Then, we reduce the number of states from four to two and remove the scrolling display which slows down the responses to build the base project following the steps below. All the changes below are in `main.c`.

1. Remove the definitions of `LEFT_KEY_IS_PRESSED` and `RIGHT_KEY_IS_PRESSED` since we don't need them anymore.
2. Change the definition of progState to include only two states: `State1` and `State2`.
3. Add the following two lines after `bool stateChanged = true;`

```
uint8_t myStr1[] = "State1";
uint8_t myStr2[] = "State2";
```

4. Change the user code in the `while` loop to the following

```
/* USER CODE BEGIN 3 */
  if (stateChanged)
      printf("The current state is State%1d.\n\r", currentState);

  switch (currentState) {
    case State1:
      if (stateChanged) {
          printf("Press the Down key to go to State 2.\n\r");
          stateChanged = false;
          BSP_LCD_GLASS_DisplayString(myStr1);
      }

      if (DOWN_KEY_IS_PRESSED) {
          currentState = State2;
          stateChanged = true;
          break;
      }

      break;
    case State2:
      if (stateChanged) {
          printf("Press the Up key to go to State 1.\n\r");
          stateChanged = false;
          BSP_LCD_GLASS_DisplayString(myStr2);
      }

      //BSP_LCD_GLASS_ScrollSentence(myStr2, 1, SCROLL_SPEED_MEDIUM);

      if (UP_KEY_IS_PRESSED) {
          currentState = State1;
          stateChanged = true;
          break;
      }
```

```
        break;
      default:
        printf("Something is deadly wrong if you see this state.\n\r");
        break;
    }

  }
  /* USER CODE END 3 */
```

After building the project, you will see the up and down keys should be very responsive. This is the base project.

Task 2. Modifying the BSP_LCD_GLASS_DisplayString function to display a string with dots and colons

(30 points) Now that the base project is working, we will modify the BSP_LCD_GLASS_DisplayString function so that we can display a string with dots and colons. In order not to mess up the existing BSP driver, We need to change the name of the function to BSP_LCD_GLASS_DisplayString_Pnt_Cln.

The following steps are used for this task:

1. In the `main.c` file,
   a. Change the contents of `myStr1` to **"XYZ100"** where `XYZ` is your 3-letter initials.
   b. Add one array named `myStr1Point` as follows to represent the decimal points in the string:

```
Point_Typedef myStr1Point[] = {POINT_OFF, POINT_OFF, POINT_OFF,
    POINT_ON, POINT_OFF, POINT_OFF};
```

   c. Add one array named `myStr1Colon` as follows to represent the colons in the string:

```
DoublePoint_Typedef myString1Colon[] = {DOUBLEPOINT_OFF, DOUBLEPOINT_OFF,
    DOUBLEPOINT_ON, DOUBLEPOINT_OFF, DOUBLEPOINT_OFF, DOUBLEPOINT_OFF,};
```

   d. Change the string display function in State1 to

```
BSP_LCD_GLASS_DisplayString_Pnt_Cln(myStr1, myStr1Point,
    myString1Colon);
```

2. In the `stm32l476g_discovery_glass_lcd_new_chars.c` file, add the following after the definition of the original `BSP_LCD_GLASS_DisplayString` function:

```
/**
  * @brief  Write a character string in the LCD RAM buffer with points/colons.
  * @param  pstr: Pointer to string to display on the LCD Glass.
  * @param  ppnt: Pointer to point to display on the LCD Glass.
  * @param  pcol: Pointer to colon to display on the LCD Glass.
  * @retval None
  */
void BSP_LCD_GLASS_DisplayString_Pnt_Cln(uint8_t *pstr, Point_Typedef *ppnt,
    DoublePoint_Typedef *pcol)
{
    DigitPosition_Typedef position = LCD_DIGIT_POSITION_1;
    int i = 0;

    /* Send the string character by character on lCD */
    while ((*pstr != 0) & (position <= LCD_DIGIT_POSITION_6))
    {
        /* Write one character on LCD */
        WriteChar(pstr++, ppnt[i], pcol[i], position++);
        i++;
    }
    /* Update the LCD display */
    HAL_LCD_UpdateDisplayRequest(&LCDHandle);
}
```

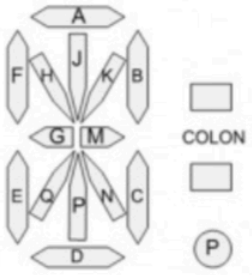
3. In the `stm32l476g_discovery_glass_lcd_new_chars.h` file, add the following after the prototype of the original `BSP_LCD_GLASS_DisplayString` function:

```
void BSP_LCD_GLASS_DisplayString_Pnt_Cln(uint8_t *pstr, Point_Typedef *ppnt, DoublePoint_Typedef *pcol);
```

4. Build the project and you will see that there are both point and colon displayed in State1.

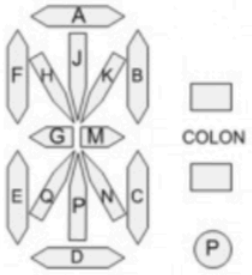
Task 3. Building three special characters

(18 points) As a preparation for the display of the special characters, you need to design three of them by printing out the following figure and tracing the segments you want to be on using a pen or pencil. Fill 1's and 0's to the corresponding segment tables. Write your final LCD codes for each character clearly as well. In the lab report, you need to include this work by taking a picture or scanning it.



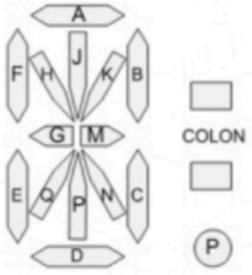
Segments	G	B	M	E
Encoding				
Segments	F	A	C	D
Encoding				
Segments	Q	K	Colon	P
Encoding				
Segments	H	J	DP	N
Encoding				

LCD code 1:



Segments	G	B	M	E
Encoding				
Segments	F	A	C	D
Encoding				
Segments	Q	K	Colon	P
Encoding				
Segments	H	J	DP	N
Encoding				

LCD code 2:



Segments	G	B	M	E
Encoding				
Segments	F	A	C	D
Encoding				
Segments	Q	K	Colon	P
Encoding				
Segments	H	J	DP	N
Encoding				

LCD code 3:

Task 4. Modifying the Convert function to display the special characters

(32 points) Now, we can display the above three special characters in State2 of the program. We can add our coding of the new special characters in the `stm32l476g_discovery_glass_lcd_new_chars.h` file, but this is a bit messy. Instead, we create a new `my_special_lcd_code.h` file and put it in the BSP folder of the project. In the `stm32l476g_discovery_glass_lcd_new_chars.h` file, we just need to include this new header file at the beginning. Below is the structural code for the new header file:

```
#ifndef __MY_NEW_SPECIAL_LCD_CODE_H
#define __MY_NEW_SPECIAL_LCD_CODE_H

#ifdef __cplusplus
extern "C" {
#endif

// My own new LCD characters and codes
```

```
#define MY_CODE1              ((uint8_t) 251)
#define C_MY_CODE1            ((uint16_t) to be modified)

#define MY_CODE2              ((uint8_t) 252)
#define C_MY_CODE2            ((uint16_t) to be modified)

#define MY_CODE3              ((uint8_t) 253)
#define C_MY_CODE3            ((uint16_t) to be modified)

#ifdef __cplusplus
}
#endif

#endif /* __MY_NEW_SPECIAL_LCD_CODE_H */
```

You need to fill in the 16-bit coding for your special characters in the above header file. Again, you need to include the above header file in the `stm32l476g_discovery_glass_lcd_new_chars.h` file.

At this stage, you should rebuild your project to make sure there is no error introduced. The point is that we don't want to accumulate errors in the code.

Now, we modify the driver function `Convert` to include the newly created special characters in the `stm32l476g_discovery_glass_lcd_new_chars.c` file. The same as before, we don't want to mess up the original function. We don't want to duplicate code either. Since the *new* `Convert` function will include all the functionalities of the original one with only a couple of characters added. We can, as we indicated in the class, use a kind of poor man's "inheritance" to program this. The approach is as follows. We change the name of the original `Convert` function to `Convert_bsp`. We rewrite the new `Convert` function immediately after `Convret_bsp` to just handle the three cases of special characters we just built. If the input character is not one of the special ones, we just call `Convert_bsp`. Of course, we have to perform the needed handling of the decimal point and colon as well as the preparation of the data for writing to the LCD_RAM if `Convert_bsp` is not called. To this end, we need to have a boolean variable `isDefault`. The first part of the `Convert` function is as follows.

```
static void Convert(uint8_t *Char, Point_Typedef Point,
            DoublePoint_Typedef Colon)
{
    uint16_t ch = 0 ;
    uint8_t loop = 0, index = 0;
    bool isDefault = false;

    switch (*Char)
    {
      case MY_CODE1:
        ch = C_MY_CODE1;
        break ;

      case MY_CODE2:
        ch = C_MY_CODE2;
        break ;

      case MY_CODE3:
        ch = C_MY_CODE3;
        break ;

      default:
        isDefault = true;
        Convert_bsp(Char, (Point_Typedef)Point, (DoublePoint_Typedef)Colon);
      break;
    }

    // Set the decimal point and the colon and prepare the data in Digit
    // See the comments in Convert_bsp for functions of the code below.
    if (!isDefault) {
```

```
    ...
}
```

Note that you need to add the remaining code to handle the decimal point and colon as well as prepare the data to be saved to `Digit` which is used to transfer data to the LCD_RAMs.

There are two more other changes needed in the `stm32l476g_discovery_glass_lcd_new_chars.c` file:

1. We need to add #include <stdbool.h> in the file so that we can use the boolean variable and `true` and `false` values.
2. We need to add the **private** prototype of `Convert_bsp` before that of `Convert`. Searching for `STM32L476G_DISCOVERY_LCD_Private_Functions Private Functions` you will find the location in the file to add.

Now, we can come back to `main.c` to add code to display the special characters in State2. Here, you can replace the original `myStr2` string with

```
uint8_t myStr2[] = "LGOabc";

myStr2[3] = MY_CODE1;
myStr2[4] = MY_CODE2;
myStr2[5] = MY_CODE3;
```

Here `LGO` stands for `Logo` and `abc` are just place holders.

We also want to display a colon after `LGO` without any points. As such, we need to build the corresponding point and colon arrays, using the same approach as we discussed in Task 2.

Submission

No need to include code in the report for this lab. Just include the following images:

- Two photos showing Task 1 works fine. One photo of the LCD display result for each state.
- One photo showing Task 2 works fine. You need to show the LCD display result of State1 where both a point and a colon should appear.
- One photo/scanning for Task 3.
- One photo for Task 4. You need to show the LCD display result for State2 where the special characters should be consistent with those given in Task 3.

Submit the code for the project as usual.