LCD Controller

Chris Ross, Brandy VanLoo
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: ckross@oakland.edu, bmvanloo@oakland.edu

***Abstract*—For our project, we made an LCD Controller. The user would be able to put text on the LCD using only command line, making it so they don't have to have extensive knowledge of microcontrollers. After the text was completed and the user was satisfied, they would submit it, and then be able to text effects, such as blinking and scrolling, using the buttons on the board. Overall this worked well, although we would have preferred if the user would have been able to do everything at once via the command line.**

I. INTRODUCTION

This project covers the basic LCD and serial communication interface (SCI) functionality, as well as our special commands and text effects for our system. The user enters a specific command to enter text, and the SCI connection delivers it to the chip (HCS12DG256). Then the chip relays this communication to a command for the LCD, and translates that command to the LCD to do display what the user wants. After the user is satisfied with the text, they would submit their text to the system. Then they would be able to add any text effects they wished by pressing the buttons on the dragon board.

The purpose for this is to create a system that would provide the user a simple way to display text on the LCD screen. This allows the user to display text and add effects without the knowledge of assembly or C languages.

Our project covers the basic LCD functionality, such as initialization, displaying characters, and clearing the display. It also covers the basics of the Serial Communications Interface including sending, receiving, and timing of the interface.

Some applications of this project include displaying text on an LED sign. With some modifications, this project could be used to take in some input and automatically display relevant information. For example, the air pressure inside of an airplane could be measured using a sensor, and then displayed as part of the pilots' cockpit display for them to monitor.

This project also made use of the mixasm.h, mixasm.asm, sci.h, and timer.h files that were used in lab.

II. METHODOLOGY

A. *LCD Commands*

We decided there would be five commands available, and two different text effects.

Setup - This is something that is done automatically when the program is started. Since the user will always have to setup the LCD, it is one less command for them to type in before they start to use the program.

"FirstLine"-This will prompt the user to enter in a string. The program will then take this *string*, up to the first 16 characters, and display it on the first line of the LCD. Because this command specifies that is uses the first line, no overflow to the second line will be done. This command will also leave whatever is in the second line as it was before calling the command.

"SecondLine "-Similar to the "Firstline" command, this will prompt the user to enter in a strong. The program will then take whatever is in the string, up to the first 16 characters, and display it on the second line of the LCD screen. Also similarly, it will only write to the second line of the LCD, and leave the first line as it was before the command was called.

"Write "- Similar to the last two commands, the user will be prompted to enter in a string. However, this command will utilize both the first and second lines of the LCD display by taking the first 32 characters of the string displaying the first 16 characters on the first line, and the second 16 characters on the second line. This will allow the user to have more flexibility and ease of programming by programming both lines at the same time.

"Clear"- This command will clear everything on the LCD, specifically, the text.

"Exit"- Upon this command, the program will exit the text portion and go to the text effects part. This is what finalizes and submits the text.

B. *Serial Communication Interface (SCI)*

We used the serial communication interface for the communication between the user and the board. We used the RS-232 standard that is described in our textbook. We used the SCI serial connection (SCI1 port) on the Dragonboard to connect to the computer to the program for the user.

The program on the computer is the interface for the user for the text entering portion. It sends the communication to the dragonboard via the serial connection, and then the dragonboard responds after executing the instruction given to it. The user is able to see the execution of the command immediately on the LCD.

C. *Buttons*

This project also used the buttons wired to Port H. These were utilized to control the text effects. Pressing the first

button returned the text to normal. Pushing the second button caused the text to blink until the first button is pressed again. The third button causes the text to scroll across the screen. For these to work, all of the DIP switches were placed in the high position. Pushing each buttons would cause a corresponding bit on port H to go low. Using a logical bitwise OR, they were compared to a char value with the respective bit brought low.

D. Programming

After deciding the commands for the LCD, we could then begin programming the LCD using the functions we obtained from Lab 5. To do a simple write to the console, for communicating the commands to the user for example, we were able to use the SCI1_putline function. Similarly, to gather the commands from the user, we used the SCI1_getline function. This helps us determine what has been entered into the program.

In addition to these functions, some private functions were created. For example, we have an "OutputLines" function which prints out the lines to the LCD. The first and second lines are stored as global variables, in order to be accessible from any function. This outputlines function then writes those to their respective places using the Lab 5 commands.

Another group of functions we created have to do directly with writing to the LCD. These include the "write", "firstline" and "secondline" commands. All of these commands update the global variables, and then call the "outputlines" function discussed above.

We also have the "clear" and "exit" functions that directly relate to the serial commands. In "clear" we set both of the global variables to null characters so they won't display anything. The "exit" command is used to exit the program, and as such, is the defining characteristic to break out of our while loop for putting the text on the LCD.

For the text effects, we also have functions that correspond to both of them. We have a "blink" that utilizes the Timer Module to be able to provide the user a blink, as well as a "noblink" that returns the text to a normal, non-blinking state. We also have the "scroll" function that scrolls once on the LCD.

E. Timer Module

We chose to use the output capture in the Timer Module to be able to make the text on the LCD blink. We first figured out the maximum delay we could have at one time using the output capture. This used the pre-scale value of 128, and is shown below:

$$MaxDelay = 65535 * \frac{128}{24MHz} = 349.52\ ms$$

We said this was approximately 350 ms. We wanted our LCD to be on about twice the time it was off, because it's easier to see that way. We then said our off-time would be 175 ms. From the above calculation, we knew that we would need 65535 cycles for 350 ms. Since 175 is exactly half of 350, we knew the number of cycles would be exactly half (and since it is an odd number we rounded down one clock cycle) to obtain 32767 clock cycles.

Similar to what we saw in class, we used a boolean variable to flip between the 350ms high and 175ms low time. To ensure that we always started with the low time, we set the timer low for 10 cycles, and then waited until it went high. This way we ensured that we would always get the correct high and low signals.

F. Interrupts

Our project made use of interrupts through the timer module. When the channel 4 counter on the output capture would change from either low to high or high to low, we received an interrupt. If the signal was high and is switching to low, we clear the LCD, add the amount of low cycles, and flip the boolean flag value. If the signal was low and is switching to high, we output the lines of the LCD, add the amount of high cycles, and flipped the boolean value once more.

In order to have an interrupt on both the rising and falling edges, we had to set the TCTL1 register to toggle, or 0x01. We used this particular register and value, since we were using the fourth channel of the timer module, and the toggle value is 0x1.

To turn off the blinking, we simply turned off the timer interrupts. This was done by writing 0x00 to the TSCR1 register. If we wanted to enable blinking again, we would re-enable the timer module interrupts by setting it up again just as we did the first time.

G. Assembly and C Code

Our code made use of assembly functions in the setup of the timer module. We called these function in the main c code, which then connected the functions to those in assembly (mixasm.asm) through the header file (mixasm.h) that connected them.

There was one function that set up the basic stuff for the first time running the timer. This set a bunch of registers to the correct values to set up the timer module. This was called "timerinit". We then had another function that reset the interrupts so that they fire on a rising or falling edge ("toggle4"). Our last assembly function was a "finalinit" function which finished up our initialization. Since we wanted to ensure that we always started on a low signal, we had to break up the initializations to wait for the signal to go the way we want.

III. Experimental Setup

To test our project, we used the Freescale Codewarrior IDE, as well as both the Dragon12-Plus Board and the Dragon12-Lite. Our final project was presented on the Dragon12-Plus board. For that board, we used an actual serial connection to USB to hook into the computer. Also unique to the Dragon12-Plus board, there was a jumper that had to be placed on the RS232 pin.

The expected results were that we would be able to use the serial communication to communicate with the LCD in the first stage, by using PuTTY to receive and transmit the messages to/from the DragonBoard, which would then change what was on the LCD as appropriate. In the second

stage, there was no specific hardware or software, as this used just the buttons and LCD that was on the board.

RESULTS

Originally, we were hoping to be able to do all of the text effects as well as writing the text via serial command line. While we could write the text and scroll the text with no problem, we ran into issues once we implemented the timer module code for the blink. Once it started blinking, we lost all communication with the board and had to restart.

Since we couldn't solve this problem, we went with the current implementation, which was to use the buttons to be able to control the text effects. The rest of the original plan had stayed the same.

After we split the project up into the two stages, we were able to test just the serial communication to the LCD in regards to writing text. This part worked fine, and we were able to perform this up to expectations.

For the text effects, after splitting up the project, we were able to test using the buttons on the board. The blink seemed good, as did the non-blink. In trying out the scroll, we discovered we could both blink and scroll at the same time and even that worked fine. This was by design, and sometimes you wanted to be able to do both.

For the most part, everything worked as it was expected to, with the exception of the bug that we weren't able to blink and keep control of the serial communication. From trying to debug the issue, we believe it was either an interrupt error, or a timing error. The serial communication uses interrupts to be able to time the receiving and sending of the data. When we globally enabled interrupts, this feature seemed to go away, which could explain the interruption in serial communication. It could have also been a timing issue, related to enabling the timer module clock, and that frequency getting mixed up with the one that the serial communication was using.

The LCD as well as serial communication were talked about extensively in class. These concepts, as well as knowledge of the buttons and mixed assembly and c programming were all implemented in this project. Except for the bug, all of these worked as explained in class without many issues.

CONCLUSIONS

This project taught the difficulty of working with the serial communications. Utilizing interrupts at the same time cause several issues. While these were addressed by using the serial port at a different portion in the code than the interrupt, the conflict was avoided. Finding the reason for the conflict could give this project more real-world implications. Moving forward, there are several more improvements that can be added. The duration of blinking could be varied. Also, longer strings could be stored for the scrolling module, or scrolling could be line-by-line.

REFERENCES

[1] Huang, H. (2010). *The HCS12 / 9S12: An Introduction to Software and Hardware Interfacing*. Clifton Park, NY: Delmar, Cengage Learning.