Mark Moerdyk

Etec 454

Multitap/Keypad Project

Introduction:

The main goal for this lab was to add a Multitap function to lab 4, as well as transmit and receive through a serial port. I learned how to transmit and receive data using an interrupt service routine, as well as low level driver routines. Instead of the LCD.c file, I had to use LayeredLcd.c which was a little more complex. The code also included transferring arrays, using double arrays, and case statements.

Program Description:

The first thing that needed to be done for this lab was to transfer over lab 4 code into lab 5, so we can build and add to the code. Once the code was transferred over, I decided to add two new modules so that the code could be easier to follow. Since there was the diagram of the three states, I added a TimeSet and a TextEntry module to make the code easier to track. The TimeSet module has the SetTheTime function, which is what was made in lab 4. The TextEntry module has the function TypeText, which allows the user to type a message, and the functions to calculate the checksum. The main module consists of the tasks StartTask, UITask, TimeDispTask, TransmitTask, and ReceiveTask.

Within the Multitap.c file, I start out with including the includes.h file. I also define the four ports for the UITask, so that it can read the A, B, C, and D buttons. Then I create each of the tasks that are all void because they do not pass anything. I also create tasks for being able to transmit and receive through the data port. Each of those four routines are listed in the SCIBIO9S12Dx.a12 file. Then I allocate task space for each of the tasks that need to run. I then create two flags that will be pending for the receive and transmit tasks. There are also global variable for using and storing data that is needed for both the ReceiveTask and the UITask.

From lines 55 to 70 in the Multitap.c file is my main(). It initializes the debug bits and the OS so that multiple tasks can run simultaneously. Then I create two flags that will help pend and post for the transmit and receive tasks. I create the start task, and have the tasks run.

Lines 78 to 112 in MultiTap.c is my StartTask. I initialize the key, timer, and LCD. The reason why the LCD has the (TRUE,TRUE,FALSE) is because of the LayeredLcd.c protocols. Then I go to the function SetTheTime() in order for the user to set the time. I place that function there because if I set the function after I create the other tasks, then the board needs to have a key press in order for you to be able to insert any sort of data into the SetTheTime() function. After the initial SetTheTime, function, the start task creates the UITask, TimeDispTask, TransmitTask, and ReceiveTask. Then I delete the StartTask, and the code runs forever to compensate for the other tasks running.

Lines 116 to 186 in the MultiTap.c file is my UITask. The main goal is that it looks for a key press. When it gets a key press, it looks to see if it is one of the three conditionals. If the key press is not one of the conditionals, it does nothing and waits for another key press. If the key press is the # sign, it goes into the function SetTheTime(). If the A button is pressed, it sets the InTran to TRUE so that if a message is sent to the board, it will alert the user while they are typing a message. It then hides the unwanted layers that would overlap the wanted layer. It then goes into the function TypeText() where the user can input characters for a transmitted message. Then, when it returns from that function, it sets the InTran to FALSE, and shows the display layer. Then it goes and checks if the A button was pressed within the function. If the A button was pressed, it will post the TransmitFlag, allowing the transmit task to start. Otherwise, it does nothing and goes back to wait for another key press. If the D button is pressed while in the UITask, it displays the dbutton layer, and shows the last received message address and when it was received. If the D button is pressed again, it hides the dbutton layer, and everything goes back to what it was.

The TimeDispTask is from lines 191 to 205 in the MultiTap.c file. It gets the clock time and displays the clocktime on the clock layer. It is exactly the same as lab 4, but uses LayeredLcd.c instead of LCD.c.

From lines 207 to 242 is the TransmitTask. It waits for the TransmitFlag to post, which is back at line 150 in the UITask. When the flag has been posted, it transmits the letter M followed by my unique three digit address. Then it gets the message that was created in the TypeText() function, and then transmits the array using sci\_write(). The last thing it sends out is the calculated checksum that it calculated in the MessageCheckSum() function. Then it transmits those last two bites and then waits for another post of the TransmissionFlag.

The ReceiveTask is from lines 243 to 318. The receive task waits for 22 characters once the first character is the letter ‘M’. Any other input is ignored until the ‘M’ is seen. Once the ‘M’ is seen, if fills the rest of the array with the remainder of the message. Then it clears the display so that the new message can be written in its place. If the user is within the TypeText() function, it will display the special character to tell the user that there was a received message. Since the display layer is a lower priority, it will not overlap the TypeText() function. When the array has all of the received data, it breaks up the string and stores the appropriate data into the variables associated. Then I perform a checksum and then check my calculated value with the message’s calculated checksum. If they match, then the message is displayed in the second row. Otherwise it displays the CS Error.

The ISR is from lines 346 to 352. It waits for a character from the data port, and when it gets a character, it posts the ReceiveFlag and waits for another character to be read.

The other main module that was written was the TextEntry.c file. It includes the includes.h file and defined the four buttons of A, B, C, and D, as well as TEXT\_CHAR that is used for arrays. Then I defined each of the functions that are used in the module. I also included an enumerator type that is used for the case statement in TypeText(). I created a double array in order to make it easy to do the multiple presses needed in the TypeText() function. I also created some global arrays that will take the typed message and put it into the transmission task.

The function TypeText was the longest function that I had to create for this lab. This was due to all of the different case statements and conditions that needed to be met. The function goes from lines 49 to 328 in the TextEntry.c file. Before I go into the case statements, I set the initial array with null characters so that no matter where the message stops, it will be filled with something. The case WAITFORPRESS waits for a key press. Depending on the key press, it will go into different cases. The next case WAITFORONESEC, waits for one second for a key press. If there is no key press within that one second it goes PLACEVALUE. If another key is pressed within that one second, it goes to ANOTHERBUTTONPRESSED. If the same key is pressed, then it cycles though the double array of values. For the case ANOTHERBUTTONPRESSED, it stores the value of previous button, and displays the next button press, and then it goes back to the WAITFORONESEC case. Depending on what button was pressed, it deals with each of those cases. For the PLACEVALUE case, it stores the button press into the array, and goes to the WAITFORPRESS case. The CLEARROW case is for when the character is the representation of the delete row. The case clears the MESSAGE\_LAYER, and sets all of the array values back to null. The IGNORECASE case is if the LCD is full. If it is full, it waits for either a backspace, or for the message to be transmitted. Otherwise, all other button presses are ignored. The BACKSPACE case moves the cursor back one value, and sets the previous cursor value spot to null. If C is pressed at the beginning, it exits the case statement, and leaves without setting the Message\_Status to TRUE. The last case ENDTASK, is when the A button is pressed. It takes the initial 16 bit array, and stores it into the global array so it can be used for transmission. Then it sets the Message\_Status to true so that the message can be sent. Then it clears the MESSAGE\_LAYER.

The other functions in the TextEntry.c file are simple tasks that do simple things. For the TransmitCheck(), it just checks if the Message\_Status is TRUE, which will tell the UITask to transmit the message. Both the ReceivedCheckSum and TransmitCheckSum functions do the same thing. The main difference is that the ReceivedCheckSum function also has to input the received array. Both tasks add the arrays and the address given. Then it calculates the total sum and breaks up the eight bit charter into four bit parts so that it can be sent or checked properly. The GetReceiveTime() stores the time stamp, while the DispTimeStamp function displays that time stamp.

In the LayeredLcd.c file, I had to add a few functions that were not given. Most of them were direct imports from LCD.c, but I had to add LcdHideLayer and LcdShowLayer. From looking at the layer functions, there was something called hidden, so I used that to tell which layers to hide or to show. Also within the LcdWriteBuffer I had to call the LcdMoveCursor function again, otherwise the cursor would not show up.

Looking at the map file for my lab 5, I am noticing that I am using up a lot of ram space. Part of the reason why I am using a lot of ram space is that I have a lot of tasks, and each of those tasks has variables. It could also be because I have a lot of arrays for the messages being sent and received. In ways of improving my memory map, I can see how much my tasks fill, and then eliminate unused space to save memory. I could also go back and see where in my code I can reduce variables, so that it will save space as well.

Conclusion:

This was a very long lab that had very difficult parts in them. The hardest part of this lab has to be the data transfer and receiving. I had a tough time figuring out how to insert commands with only the SCIBIO9S12Dx.a12 file. I did learn a lot about the LayeredLcd file though and how it functions. I also learned how to use interrupt service routines more efficiently. I wish I had more time to go back and reduce my code, seeing as the extra time was used for debugging my program.