Mark Moerdyk

Lab 3 write-up

2/12/13

Code: 2/6/13-2/12/13

ETEC 454

**Introduction:**

The main goal for this program was to create a MicroC/OS project that ran the same code as lab 2. I was supposed to take the functions in lab 2, and convert them to be separate tasks that would run simultaneously. There were 4 different tasks, which includes StartTask, KeyTask, DemoCntrlTask, and LCDDemoTask. The class was given an example code to refer to, so that we could understand how MicroC/OS works.

**Program Description:**

The first thing to do was to create a new main.c for the project. Then we included the files given for the µcos project, which was contained in the Y: drive. I then changed the Makefile to make it compatible for this project. You also had to insert the project .int file for the code to compile later. For calling the OSProj folder contents, you had to include a / so that it would go into the folder to add the required files. After modifying the Makefile, I could start writing the main.c file.

To start the main.c file programming, I included the includes.h and defines that I knew that I needed from lab 2. Since the KeyTask was already written in UcosKey.c, I only had to create three tasks. As you can see from the attached main.c file, each of the three tasks are global tasks because they are called with other tasks, and are made static for the same reason. Each of the three tasks has to have stack space allocated, which is defined under the OS\_STK, which is seen attached under main.c. Each of the stack space is defined under the app\_cfg.h file, which is also attached.

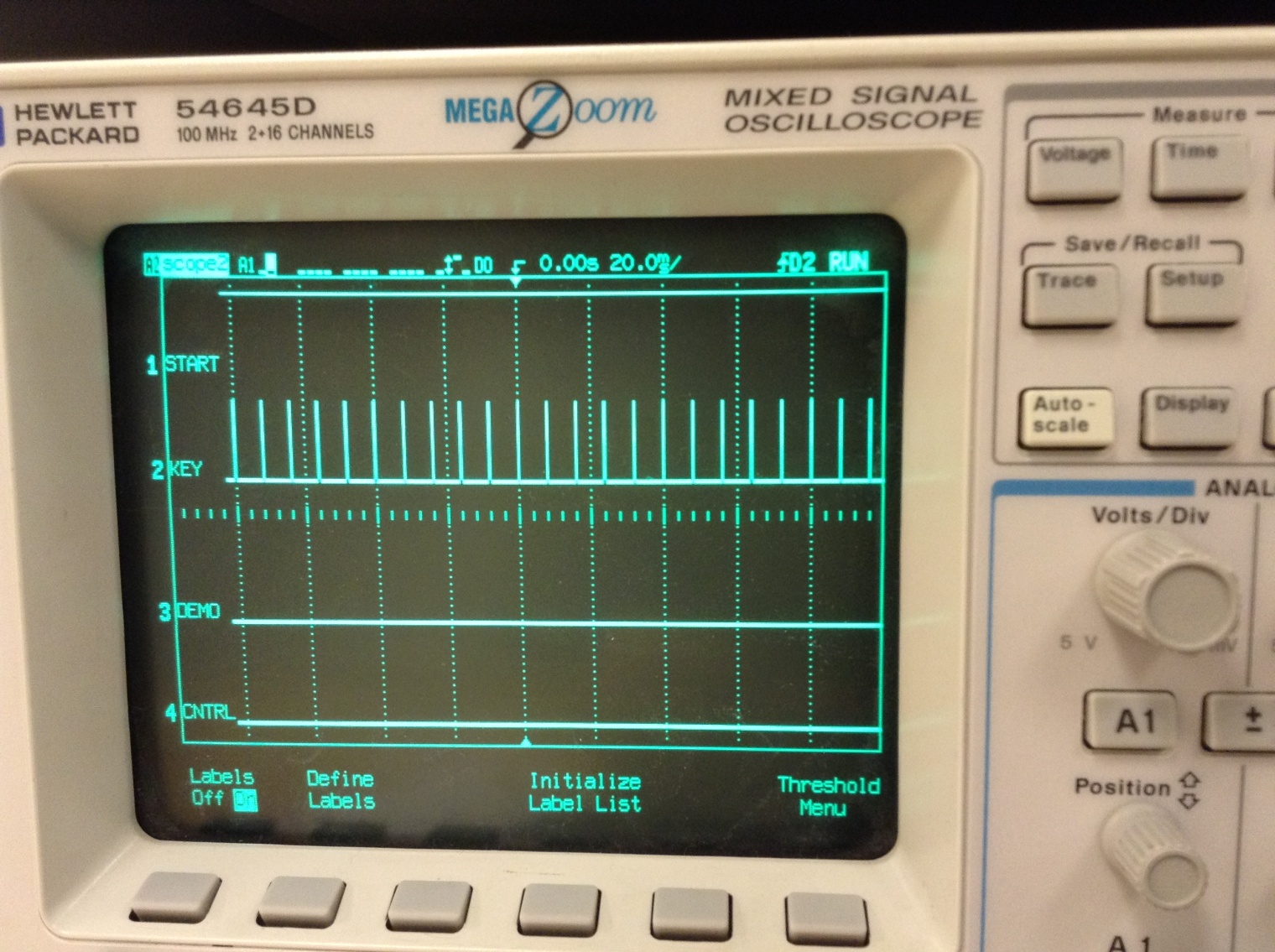
Within the main(), which is attached as a part of main.c, you can see that the first thing I do is set the debug ports to be able to see the tasks working from the port pins. Then I initialize the OS, Key, and LCD to be able to use them later in the code. After initializing, I create the first task, and include all of the parameters needed to create the task. Then I start the tasks.

From the StartTask task, I copy code from lab 2 that is needed for the checksum functions, as well as create new tasks for when the checksum is complete, and the C button is pressed. As you can see from the attached main.c under the StartTask, I first set up the integers needed for the code, as well as void the pointer p\_arg, since we don’t need it for this code. I then initialize the OSTick for counting, and clear the LCD display from previous use. Then I set the cursor to the proper spot for printing. Then I call the function CalcChkSum, which is seen below the StartTask in main.c. This adds the contents between the start and end address, and returns the value of the total content back to a 16 bit integer. It then prints out the “CS: “ followed by going to function called DisplayCheckSum, which you can see attached in main.c. DisplayCheckSum takes the 16 bit integer, breaks it into upper and lower 8 bits, so that it can be displayed on the LCD. Both DisplayCheckSum and CalcChkSum are from lab 2. After all of the printing is done on the LCD, it waits for the user to press the C button. If the C button is not pressed, then it keeps waiting for the C button to be pressed. Once the C button is pressed, then it continues and creates the LCDDemoTask and DemoCntrlTask. It then pulls the StartTask pin low, and deletes the task. There is the FOREVER() loop because it is a task.

The DemoCntrlTask is a very simple task, that is attached under main.c. The whole task looks for a keypress, and if that key press equals the B button, then it deletes the LCDDemoTask, and creates a new LCDDemoTask, thus restarting the LCDDemoTask. If any other button is pressed, or no button is pressed, then it does nothing.

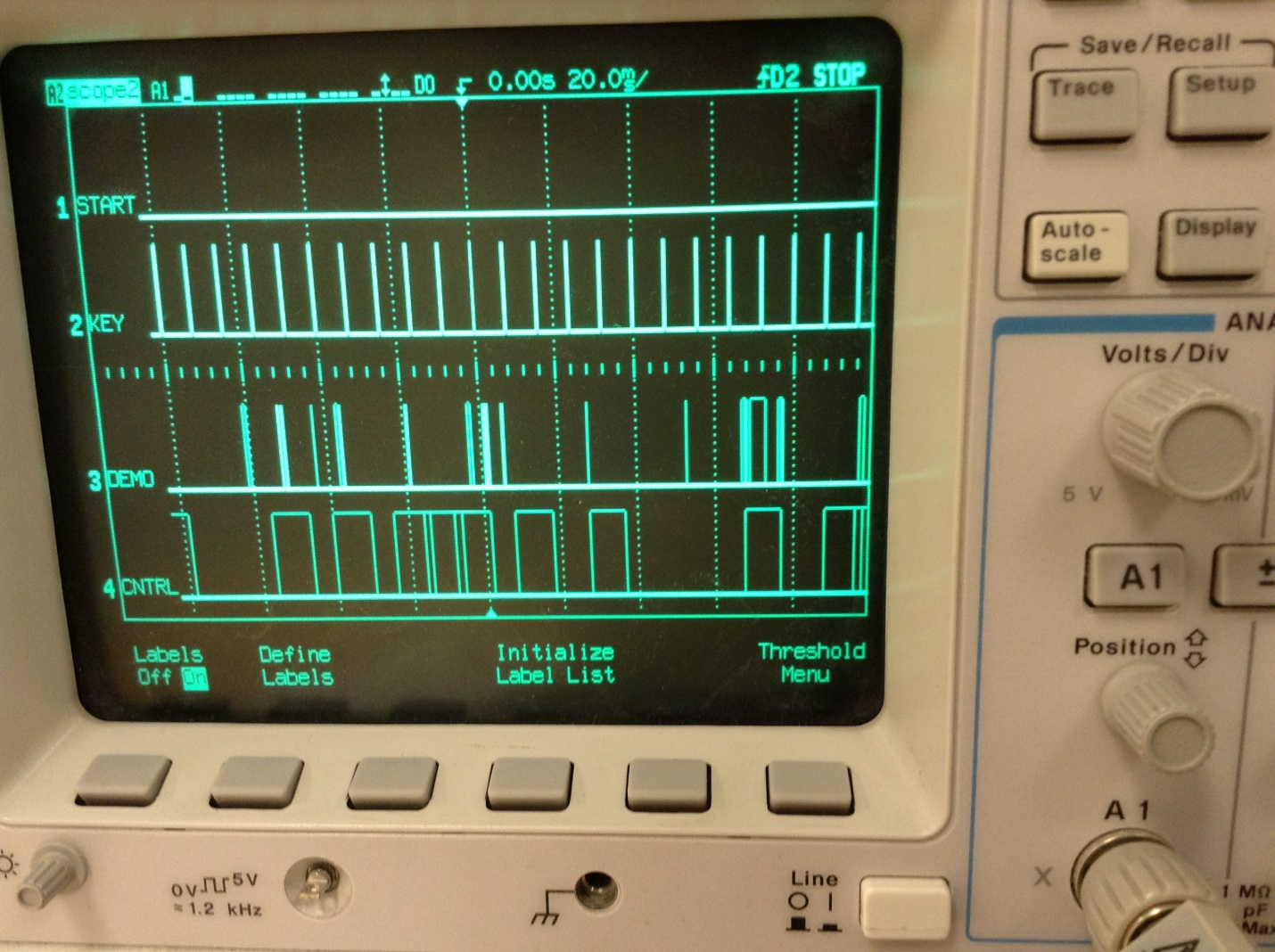
The LCDDemoTask is a copy of the LCDDemoTask in lab 2. There are some differences between the two codes. The LCDDemoTask code is attached under main.c. The first thing was to clear the LCD display, so that the contents of the checksum would not interfere with the demo. All of the contents are within the forever loop so that the demo could run forever if needed. The main difference between this demo task and lab 2’s demo task is there is no need for a case statement in this version. Instead, for inserting time delays, we just use the OSTimeDelay function. I have the debug ports toggle high when a task is running, and low when it is waiting.

There were a few changes in the includes.h file, which is attached. The main change is that under the project Constant and Macro Definitions, I put the pin define contents there. I did this because the contents are needed for more than one file, and it was easier to define them once, rather than twice. Also included under the module header files are the UcosKey and LCD header files.



Figure

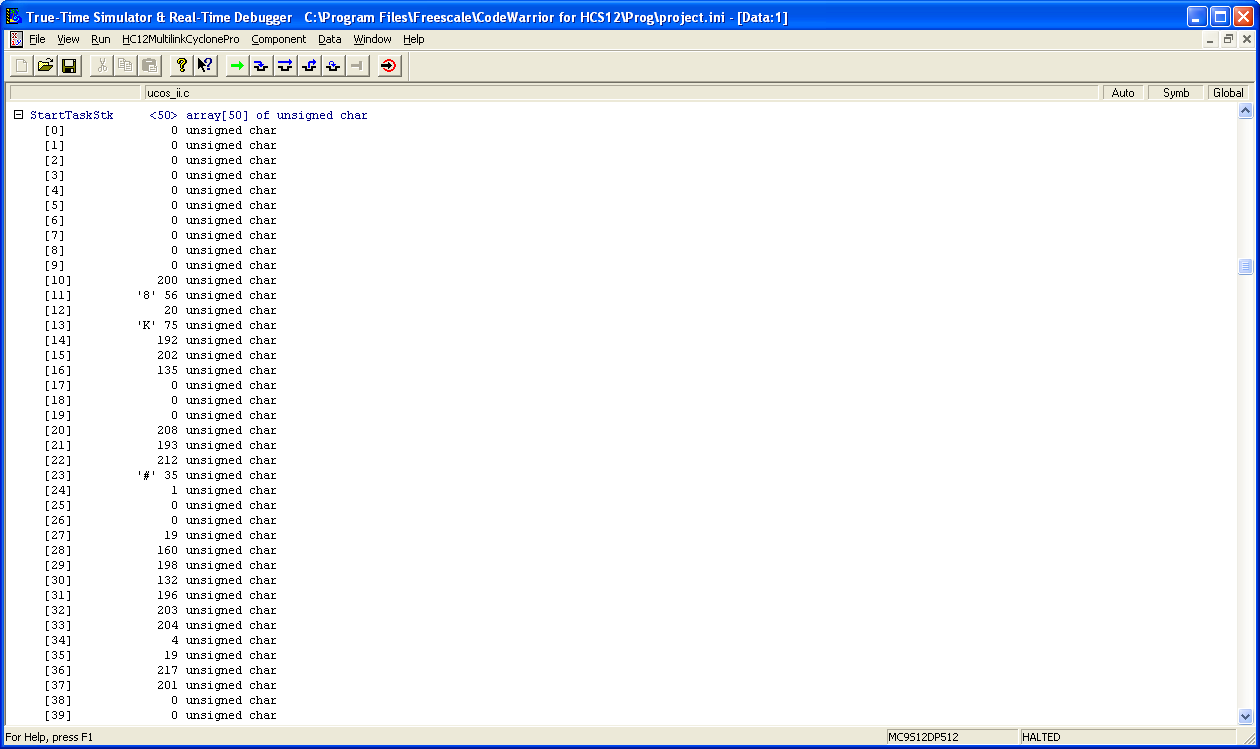
Figure 1 above shows the debug bits for each of the four tasks when the code starts up. As you can see, the StartTask is running because it is high, and the Demo and Cntrl are low since they have not been created yet. The KeyTask is getting the input from the Keypad, and seeing if any key has been pressed.



Figure

Figure 2 above shows the debug bits when the C button is pressed, and the Demo and Cntrl tasks are running simultaneously. Since the start task is has stopped running, it is now low. The demo line shows each of the demos running, and the Cntrl working when the B button is pressed. The KeyTask is unaffected by any of the three other tasks.

The other two files that needed to be changed were the os\_cfg.h file, and the app\_cfg.h file, both attached to the document. The changes to the os\_cfg.h file were to the OS\_LOWEST\_PRIO and OS\_MAX\_TASKS. I changed the lowest priority to 11, and max tasks to 4, which is due to the four tasks listed in the introduction. For the app\_cfg.h file, the task priorities were changed, as well as the stack sizes. The StartTask always has the highest priority, and the KeyTask came second because the task was always running. I made the LCD demo third priority because it runs more than the control task. The control is last because it only does something if a button is pressed. For the Task Stack size, I looked at each of the stacks, and saw where they started having continuous zeroes in the stack. An example of looking at the stack can be seen in Figure 3 below. Then I would reduce the stack size accordingly.



Figure

**Conclusion:**

This lab was not as bad as I thought it would be. It was nice to have a demo to run, to make sure that we could get the Ucos code to work. Transferring the code over from lab 2 was simple, and once I got the Tasks to work, it was easy to understand how the code worked. However, I don’t get some of the OS tasks in the os\_cfg.h file, and how that could optimize our code better.