Mark Moerdyk

ETEC 454

Lab4

MicroC/OS Realtime Clock

**Introduction:**

The main goal for this lab was go get familiar with MicroC/OS commands, and using that for a clock. I learned how to use an OS timer, and how to get it to flag so that even if something else was running, the timer was still working in the background. Another part of the lab was to learn how to use a mutex key efficiently. There were also two different modules that we had to create.

**Program Description:**

The first thing needed to be done in the lab was to transfer all of the MicroC/OS data from previous labs, into a new lab. Then when this lab was created and running, I created a new main.c and a new clock.c. Attached in the main.c on lines 1 to 32, are the initialization of the three tasks, as well as another function that deals with setting the clock. For each of the three tasks in main.c, there is allocated space, which is defined in the app\_cfg.h. In lines 45 to 60 is the main function, which creates and runs the StartTask. It also sets up the debugging ports for each of the five tasks needed.

The StartTask, which is in the main.c code from lines 68 to 93, does several tasks. The first task is does is initialize the OSTick and move the cursor to the appropriate position for the clock printout. It then sets the debug port high and jumps to the TimeInit() function, located in Clock.h. Then it creates the UITask and TimeDispTask to run simultaneously. The debug port is set low and the StartTask is then deleted. It still runs forever so that other tasks can run.

Jumping to the Clock.c file attached in back, lines 1 to 42 are initializing all of the tasks, variables, and defines. There is only one task in this Clock.c, called ClockTask that increments every second. Also, there is a Clock.h which makes TimeInit, TimeSet, and TimeGet external so they can be used in main.c. There is a static global structure that is defined in includes.h as Time, which includes hr, min, and sec.

The TimeInit() function, which was called in the StartTask in main.c, is in Clock.c in lines 83 to 111. The first thing the function does is set the TimeOfDay structure equal to 12:00:00. This is done so that when the reset button is pressed, it starts back at 12:00:00. Then it displays the time on the LCD. I then set an OS\_TMR equal to a created timer that is in the ucos reference manual. I set the timer to do a 1 second cycle, and have a callback function ClockTimerFnct that will be used for posting the secflag. After I created the timer, I then started the timer, and create a mutex key for the other two functions TimeSet and TimeGet. Then I created the ClockTask to get the clock working and create a SecFlag that is used to count every second.

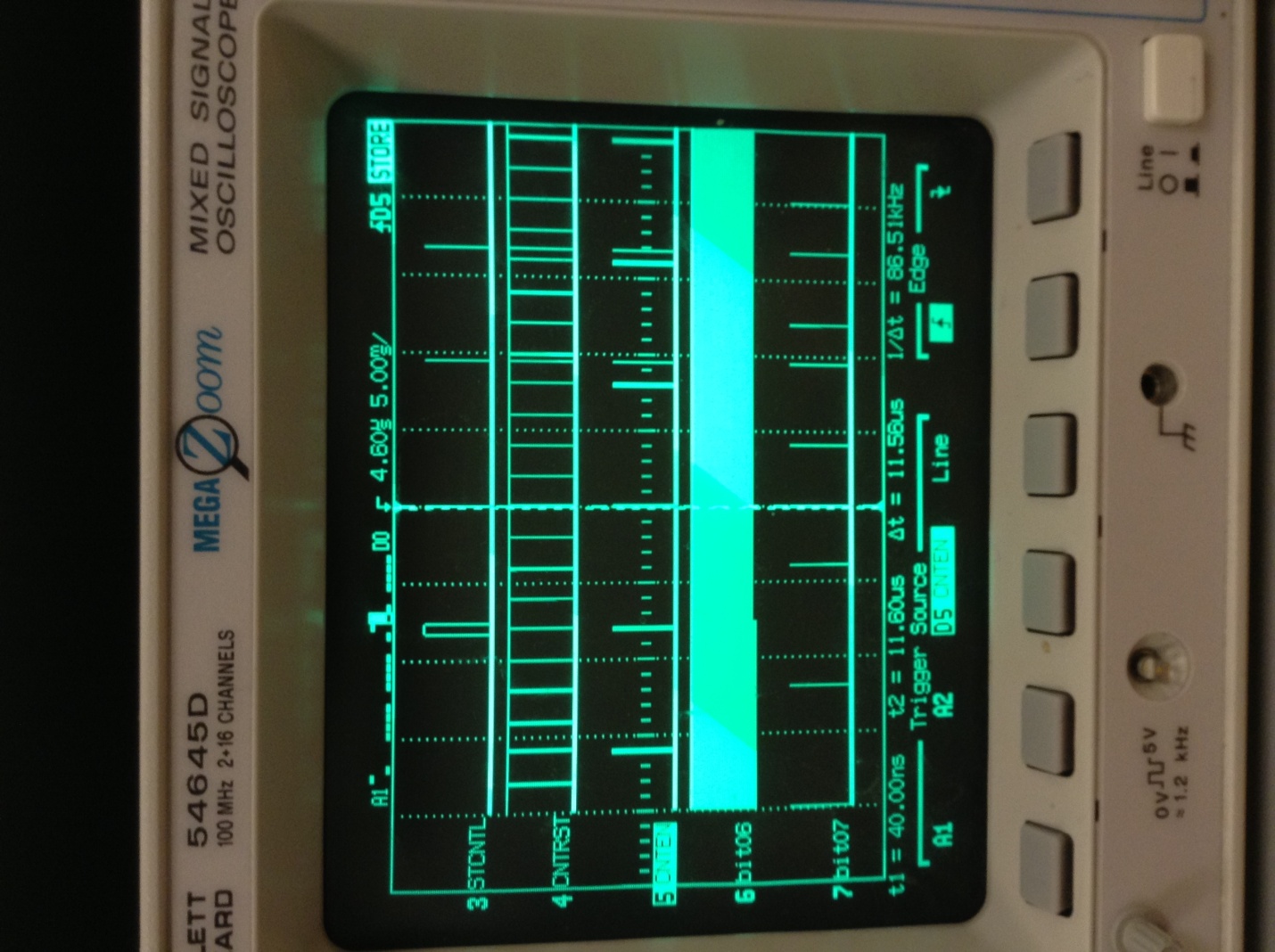
The ClockTask that was created in the TimeInit function is in Clock.c, in lines 42 to 77. Everything is in the forever loop because the clock should always be incrementing every second. The debug port is off while the SecFlag is pending. When the SecFlag hits the one second mark, it goes high, and sets the debug port high as well. Since the SecFlag goes every second, we can increment the TimeOfDay.sec by one. The if, else if statement is for implementing the rules of a standard 12 hour clock into this code.

The last three functions in Clock.c are TimeSet, TimeGet, and ClockTimerFnct, which are from lines 118 to 145. TimeSet passes in the TIME structure from main.c and sets the time of the passes structure to TimeOfDay. There is also the mutex that is pended and posted here so that there is no interruption of other tasks. The TimeGet function does the opposite of TimeSet. It takes the TIME structure given from main.c, and gives it the value of TimeOfDay, so it can be displayed in TimeDispTask, back in the main.c code. The mutex is also applied here. The function ClockTimerFnct does one thing, which is to post the SecFlag for the timer, so it can be read for the ClockTask.

Going back to main.c, the UITask is from lines 97 to 120. Within the forever loop, it waits for the button press of the keypad. If the keypad press is equal to ‘#’, then it goes into the SetTheTime() function. Otherwise, it waits for the key press. The TimeDispTask in main.c from lines 125 to 139 is very simple. It send in a new structure into the TimeGet function, and retrieves the value of TimeOfDay. Then, it displays the time of TimeOfDay on the LCD.

The largest function is SetTheTime, which is from lines 145 to the end of main.c. The main purpose for this function is to take any input that the user will put in, and then print out the result print. It also takes into account the values of the button press, and assigns them when either the user presses A, or is done in putting all six values. The first thing the task does is suspend the TimeDispTask because then there will not be any overlap on displaying two different things. Then it goes into a long while loop, in which it continues until a specific value is assigned to finishset. The process is set up as a case statement, in which it goes through 8 different cases. Each of the first 6 cases deals with the cursor placement for hours, minutes, and seconds. Each of the three hours, minutes, and seconds are split up into two positions of ten and one. The one with the most conditionals to consider is ten hours place. It is because it can only be a 1 or a 0, but it depends on the ONEHRPLACE. This is why there are all of those conditionals under the TENHRPLACE case. The ONEHRPLACE case is the only other case with conditionals in it. However, this depends on the value that is placed from the ONEHRPLACE. If the TENHRPLACE value is equal to a one, then only value of zero, one, and two can be placed. If a zero is in the TENHRPLACE, then a one through nine can be placed. For the other four values, they have no constraints. The ten values for both minutes and seconds can be valued from 0 to 5, while the ones place for the minutes and seconds can be from 0 to 9. There are also conditions where the A or C button is pressed.

If all six of the values are set, or the A button is pressed, it goes into the finishset part of the code. If B is pressed, then it skips that code and returns to the tasks. When finishset is equal to one, it then goes through the flag values in reverse order to make sure something is not skipped. If the condition is true, it sets the needed TIME structure values to appropriate values, and sends those values to the SetTime() function. Then it exits out of the function, returning to the tasks.



Figure

Figure 1 above shows the debug pin output. Port 3 is the StartTask pin, which goes high in the beginning, but low when it ends. Port 4 in figure 1 is the UITask. It goes high when the ‘#’ button is pressed, and returns low when after going through the SetTheTime function. Port 5 is the KeyTask, where it waits for a key press. Port 6 is the TimeDispTask. It is the busiest task in the code because it is constantly writing to the LED, which takes the most time. Port 7 is the ClockTask, which toggles on the addition of the clock.

**Comments and Conclusions:**

This lab was interesting, but at times for me, very long and boring. The timer was the hardest to figure out, but most of the work went into the programming of the clockset. If I had more time and done more research, I would have liked to reduce the SetTheTimeFunction, but I feel that at least the way I set it up, it is very easy to follow. I do understand SecFlags more. The lab itself was hard for me to understand what you wanted as a product of the code.