ECE 2036 Lab 6 – An IoT Alarm Clock - using threads and the mbed RTOS

Due Date: Section A Monday April 20
Section B, Tuesday, April 21

Name:_____

Check off Item	Lab Demo	Extra Credit
Ticker API blinking 4 LEDs at different rates		n/a
Mbed RTOS - LCD, LED, & Sound Effect Code		n/a
Color enhanced RGB LED effect		2%
IoT - clock time automatically set		5%
Settable Alarm Feature		5%
Early Check off - at Least two weekdays before due date for your section		2%

(**Part 1** *20%*) Blink mbed's four built-in LEDs at different rates of 1, 2, 3, and 4 seconds using the Ticker function (no *wait(),* RTOS or threads allowed!). See the handbook's Ticker wiki code example. Ticker uses a hardware timer on mbed to periodically generate an interrupt that triggers a different function to run. Ticker is similar to Timeout, but Timeout only works one time. It should look at bit like a binary counting sequence in the LEDs when it runs.

A periodic timer interrupt can also be used to time slice the processor among various tasks and such hardware is used in an OS scheduler for time slicing among processes or threads. The scheduler shares processor time among multiple threads. The mbed Real-Time Operating System (RTOS) uses the timer to provide its 1 ms. time slice interrupt. With an RTOS, the scheduler makes it possible to run multiple threads automatically. The RTOS also provides the needed synchronization primitives when threads share global variables or I/O devices. So once things get a bit more complex with several things happening in parallel at different asynchronous times, an RTOS may be needed and it makes coding easier. Read the LED lighting effects wiki & code examples for an introduction to using the mbed RTOS and threads. Pay particular attention to the examples near the end using the RTOS. An RTOS switches between threads an order of one or two magnitudes faster than the typical a desktop OS like Windows or Linux. Threads are also needed in modern applications to be able to utilize all of the processors on a multicore processor. They provide speedup by running on multiple cores. Without threads only one core is used by an application program. Mbed can only run about 8 threads given the limited 32K RAM memory (i.e., each thread needs its own stack and the RTOS uses some RAM). A modern PC is typically running one to two hundred threads.

(**Part 2** *80%*) Next, the mbed RTOS will be used to run four threads. A bare bones code template is provided which just sets up several threads that only blink the LEDs at a different rate, but by using the RTOS this time. To get a copy of the template program into your compiler, just use the import program link at http://developer.mbed.org/users/4180_1/code/2036lab7_template/ and then click the "Import this program" button and then selecting/confirming/changing the name of the project. Read the code and comments in this example to see how to setup and start running threads with the RTOS. Main always runs the first thread, so the code example only creates three more threads. The code template should already have all of the libraries need for all of the basic parts of the lab assignment. The template code uses the RTOS to blink the four LEDs at different rates and it also displays time on the LCD. Before proceeding, it would be a good idea to verify that your hardware setup for the color LCD will run the template code.

The threads in the code template will need to be modified in the lab assignment to do more complex things. Two threads will put something on the uLCD display. Since the display requires a complex sequence of commands, mutual exclusion access control on the display is required and a mutex synchronization lock must be used in each thread before it can access the display. This is required to avoid errors that could occur when the RTOS 1ms time slice interrupt forces a switch between threads when they are in the middle of writing a long serial command to the display. This could lock up the display with an invalid command. A single mutex synchronization lock must be setup that is always used in each thread using the LCD to lock and then unlock access to the uLCD. This must be done prior to sending a new LCD command such as a *locate* or *printf*. Minimize the scope of the mutex lock by putting as many calculations as possible outside of the lock/unlock. The same mutex problem can occur, if two threads share any global variables. Never use *wait()* with the RTOS, *thread::wait()* must be used instead! The argument of *thread::wait* is milliseconds, and *wait* is seconds. Most threads will keep running once started and will need a *while* loop.

The four threads should do the following:

1. Use the RGB LED or Shiftbrite to display a new lighting effect for a fire (orange) or welding (bluish-white) based on the code found in LED lighting effects wiki & code examples.
2. Play a wave file from an uSD card or USB flash drive on the speaker with sound effects appropriate for the LED lighting effect selected. There is a filesystem driver for the uSD card and also another driver for USB flash drives that can be used with the USB breakout board in the parts kit (only try this if you don't have a working SD card – it might run out of RAM). Sound effect web sites have sound clips to use, there is a basic sample at www.ece.gatech.edu/~hamblen/2036/handouts/lab7 . There is a mini USB to uSD card adapter in the mbed kit that can be used to write the sound file to the SD card using your PC. Some PCs also have an SD card slot and there is also an SD card to uSD card adapter in the parts kit. The mbed cookbook's waveplayer code will play a *.wav file. It will run in the RTOS without changes per the LED effects wiki page. Use Audacity to convert and downsample the *.wav file to a rate <16Khz mono, if you create your own wave file for the sound. Audacity can also convert *.mp3 format files to *.wav as described in the waveplayer section of the speaker wiki page. The speaker wiki page shows the setup and don't forget that it uses the D/A output (p18) to drive the speaker (i.e., not a PWM pin!). The waveplayer code does not return until it finishes playing the entire wave file and this is one reason it needs to be in it's own thread. When the sound file ends, just replay the sound file again in the thread.
3. Read the temperature using your TMP36 from the kit and display the temperature in degrees C (or F) on line 4 of the LCD every eight seconds with 2X size text. **Don't forget** that threads 3 and 4 will need the mutex lock/unlock to avoid crashing the LCD!
4. Start the RTC clock once, and display the time on line 2 of the LCD every second with 2X text in the format (HH:MM:SS). See time wiki page to see how C++ typically reads the system time. The time must be set with *set_time* before the clock starts running on mbed. The "%T" format string in *strftime()* is the easy option to use to generate the string needed to print the time.

The easy way to set all of this this up in the RTOS is by using *Thread::wait(ms);* in each thread at the end. It would probably be a good idea to leave in the code in each thread's while loop that toggles one of the builtin LEDs, so that it is apparent that each thread is still running correctly just by watching when the LEDs toggle. Don't forget that main is the first thread, so only three more threads need to be created.

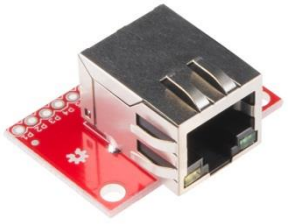
There is a similar RTOS thread idea for a different LCD hardware setup used on the mbed application board at http://mbed.org/users/4180_1/notebook/mbed-application-board-hands-on-demos/ if you need to see more RTOS code examples with threads, waits, and mutexes. There are similar graphics member functions for the uLCD display as in this example, but they do not have exactly the same names and it uses a different LCD.

Extra Credit: (2%) In addition to just changing the brightness level on the RGB lighting effect, it should also change the color just a bit randomly as suggested in the LED effects wiki page (i.e. fire orange and reddish orange color changes, welding white to bluish white). This also adds a bit of interesting spatial movement to the effect since each LED color is in a slightly different position inside the RGB LED.

Extra Credit: (5%) Make your clock a state-of-the art Internet of Things (IoT) device that automatically sets itself to the correct time whenever it powers up. In main, prior to starting the other threads, use the mbed's networking software and the internet to automatically set the current time on mbed's RTC. The Ethernet library sends and receives network packets. The NTP client library uses Ethernet network packets to talk to a server on the Internet to get the time. There are Network Time Protocol (NTP) client code examples in the cookbook that can do this. First, the mbed will need a wired network connection to the Internet using the Ethernet magjack breakout adapter, four jumper wires, and the network cable from your parts kits. The board might look a bit different in older kits.

Wiring Table

Mbed	**adapter**
TD+	P1
TD-	P2
RD+	P7
RD-	P8



Two new include files are needed for the networking and NTP client software (the *.h files are already in the template project):

```
#include "EthernetInterface.h"
#include "NTPClient.h"
```

Here is some sample code to initialize the network and set the mbed's clock using NTP:

```
{
    EthernetInterface eth;
    NTPClient ntp;
    eth.init(); //Use DHCP
    wait(2);
    uLCD.cls();
    uLCD.printf("\nGetting IP Address\r\n");
    if(eth.connect(60000)!=0) {
        uLCD.printf("DHCP error - No IP\n");
        set_time(1360000000); //starts RTC
        wait(10);
    } else {
        uLCD.printf("IP is %s\n", eth.getIPAddress());
        wait(2);
    }
    uLCD.printf("\nTrying to update time via NTP...\r\n");
    if (ntp.setTime("0.pool.ntp.org") == 0)
        uLCD.printf("Set time successfully\r\n");
    wait(2);
    uLCD.cls();
}
```

Run this code first in main, prior to starting any other threads and make sure that it is in a code block (i.e.,{}). This will force it to recover the memory used by the networking code. The networking code requires the RTOS and runs its own threads, so it uses a big chunk of the RAM. If your other code has another *set_time*, don't forget to remove it. Before using the Internet, mbed needs to have an IP address. It takes several seconds even up to a minute to get an IP address for mbed from the DHCP server, and then a few more seconds just to get the time – so be patient! A network NTP time server returns the current UTC time without an adjustment for the local time zone. It would be something like (hour +8)%12 for the local time here. The time can just be displayed in UTC format on the LCD. Be very careful, if you don't use the two networking libraries already provided in the template program, there are several versions with the same name and if you somehow mix old and new versions a lot of strange compile errors will pop up.

Getting an IP address from a DHCP server can require some additional steps on secure networks like those at GT (some require a password logon or a registered MAC address before they will provide a computer with an IP address). Chances are you don't have this DHCP security setup on a home wired network. One way around this problem at GT, is to setup a shared network connection on a laptop with Wi-Fi and a network jack. Instructions are on this wiki page at http://developer.mbed.org/users/4180_1/notebook/setting-up-a-network-bridge-connection-for-mbed-on/. On the campus housing network connections, a device's MAC address must be registered at start.gatech.edu to enable DHCP to provide a valid internet IP address. If it is needed, a program to print out the MAC address for your mbed can be found at http://developer.mbed.org/users/simon/code/mac/file/8b0b6c9343c2/main.cpp. This issue can turn into more work than adding the code needed. If it says "DHCP error - no IP" and the cable is hooked up correctly to a working network jack, then this issue is the problem.

Extra Credit: (5%) Make your clock an alarm clock. Using two or more pushbuttons, add code to set an alarm time to HH:MM (no seconds). Display the Alarm time on the uLCD on another line. When the time matches the alarm setting time, turn on the LED effect and sound effects until a button is hit to turn off the alarm. There are time structures setup in C++ that can be used to check the alarm setting with the RTC time. Something like *time.tm_min* or *time.tm_hour* can be used to access just the hours and minutes in the current time. A signal or a *volatile* global variable can be used to tell the existing RGB and waveplayer threads to start the sound and led flashing whenever the alarm should sound. The exact way the pushbuttons set the alarm is up to you, but it should be relatively easy to use and reliable. Just like a real clock, a user (or TA) might need to read some instructions to understand how to do it!