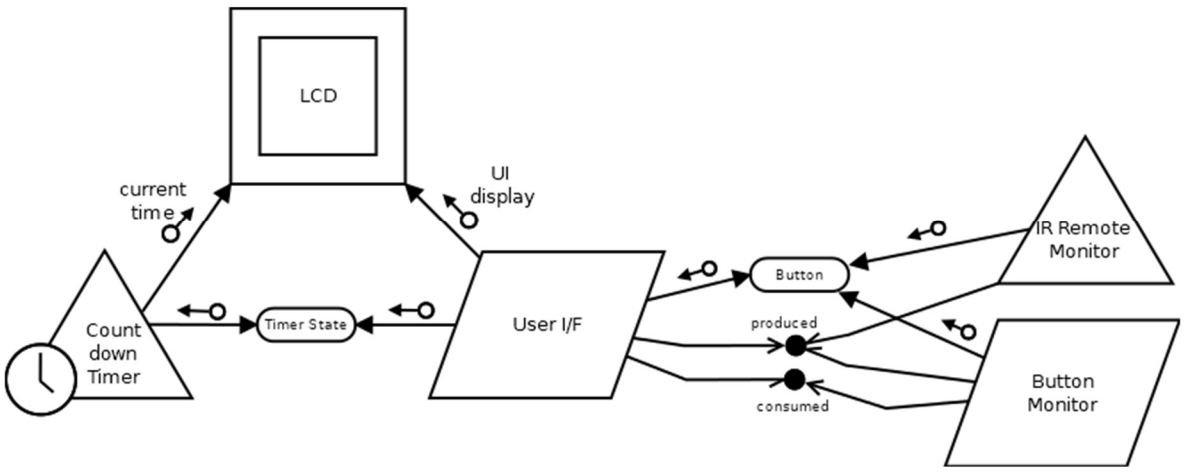
CSCD 462: Embedded Real-Time Systems
Final Project: Enhanced Kitchen Timer

For your project you will modify the midterm kitchen timer such that when it reaches a terminal count it flashes the display until the user presses any button (I would have had you also sound a buzzer if I had remembered to include that in your kit). You will also add auto-increments and decrements when holding a button down and support the use of the IR remote as an alternative to the keypad.

There are a lot of ways to do this, and I have implemented several designs: (i) 1 main task and 2 ISRs; (ii) 4 tasks and no ISRs; (iii) 3 tasks and 1 ISR; (iv) 2 tasks and 2 ISRs. No one approach is necessarily better than another, but a case can be made that (i) and (iv) are the best alternatives. Approach (i) would require no RTOS. The approach would be the same as your midterm with an added ISR to process input from the IR remote. Of the possible designs that require an RTOS, a case can be made for (iv) as follows:

1. While a 0.1 sec periodic task could be used to drive the countdown, that requires sleeping until the next update. Making that as accurate as possible requires a computation of the remaining time after updating the display, as discussed in class. At best the accuracy is driven by the RTOS sleep resolution. In contrast, a timer ISR is very accurate and requires no computation of the next execution. It also eliminates the need for producer/consumer semaphores to communicate with the U/I task.
2. It makes sense to separate the main task (from your midterm) into two tasks: (i) a task responsible for maintaining the user interface (other than the countdown), and (ii) a task that polls the buttons. That cleans up the encapsulation of different activities and makes it easier to change the input method, or add an additional input method, which is what we're doing here. Dealing with the button interface must be done in a polling task (as opposed to an ISR) because the keypad provides no digital signal indicating that any one button has been pressed. Note that you've used a task to poll a button in lab.
3. The infrared remote input could be polled and decoded from a task, but we like to avoid polling whenever possible because it wastes processor cycles (for non-bouncing inputs at least). An ISR is a good choice here, with the interrupt being generated from the leading edge of the start pulse.

Since a major objective of this course is to gain some experience with RTOS, that is the approach you will take. A Buhr/Bailey task interaction diagram of the design looks like this:

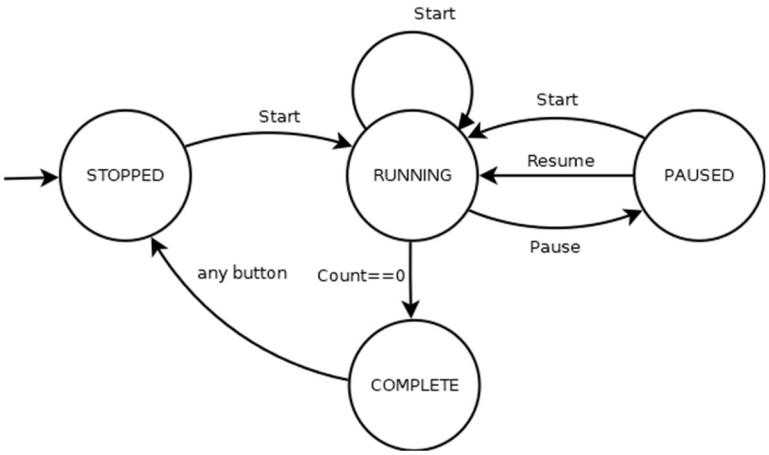


Connect the IR sensor to pins 19, 20, and 21, with the signal on pin 19. You can then arrange for pin 19 to fire an interrupt on the leading-edge of the IR start pulse. You'll need to use a different hardware timer than the midterm because Timer1 is used by ARTK. I didn't move that to Timer2 for a couple of reasons: the associated library (MSTimer2) works differently than TimerOne which would have meant code changes other than replacing a 1 with a 3 or 4. My solution also used the tone() function to drive a buzzer and that uses Timer2 (you are not required to do that). I didn't move it to Timer3 either, because I use that for IR pulse timing (as in the Lab). That left Timer4 for the countdown timer.

Note from the diagram above that either the button monitor OR the remote monitor can send commands to the user interface task (via a producer/consumer buffer). The equivalencies between the keypad and IR remote buttons should be: SELECT=5, LEFT=4, UP=2, DOWN=8, RIGHT=6. Holding a button down on either device should result in repeated actions. That means you should make use of the remote's REPEAT code. For the keypad you'll have to detect when a button is being held down. The pace should make auto-increment and auto-decrement usable when setting the minutes and seconds fields. In other words, it should not blow through a lot of numbers before the user can react near their target setting.

When a running timer decrements to 0, it must enter an alarm state, which will flash the LCD off and on (without changing what it currently displays). It should cycle between being on for 500 msec and off for 500 msec, until the alarm is canceled by the user by pressing any button (on either the keypad or the remote. When used to cancel the alarm, that keypress should not perform any other function. For example, canceling an alarm with the Select button should not force a return from the Set screen to the Main screen. Note that the LiquidCrystal library has functions to turn the display off and on without modifying its content.

Compared to the midterm, this means that the state transition diagram for the timer now has an additional COMPLETE state:



All other behaviors must be identical to your midterm. If your solution uses multiple source files, zip up your project directory and use the usual naming convention for the zipfile.