Objective

Building on prior work, this project will see you create a stopwatch.

You are tasked with creating a "proof-of-concept" prototype of the circuit and writing the firmware to demonstrate the idea.

Constraints and Requirements

As with most projects, your design and its function are going to be constrained in a variety of fashions and has some requirements to make the proof-of-concept useful. Many of these will naturally lead to specifications!

1. please make use the ATmega328P or ATtiny2313A;
2. quickly pressing a pushbutton is to start and stop the stopwatch;
3. when stopped, the string "`Stopped`" is displayed centred on the top line of the LCD, and when running, the string "`Running`" is to be displayed;
4. the time in "`HH:MM:SS.hh`" format (where hh is the number of hundredths of seconds) is to be centred in the bottom line;
5. ensure that the display does not flicker;
6. pressing the same Run/Stop pushbutton for an extended time when the stopwatch is stopped will reset the recorded time to 00:00:00:00; and
7. using a PS1420P02CT piezo buzzer (the same as used in Project 2) a 1kHz tone is to be issued for 100ms at the start of each second that passes when the stopwatch is running;
8. please **make use of an interrupt service routine** to track the time;
9. the stopwatch's time should be accurate to +/- 1 second over a 1 hour period.

Deliverables

- A working prototype of the circuit.
- A cost estimate of your final prototype.
- A (computer-drawn) schematic of the final circuit.
- C and header file(s) submitted via eClass.
- A specification sheet for your product (1-2 pages)
  - basic description of the function
  - operating conditions (input voltage range, description of output levels, temperature range …)
  - result of your testing
- Live presentation of your product to the lab instructor
  - Circuit function
  - Code walk-through
  - Scope tweak: a hands-on task for each group member

Background

Although this lab could conceivably be completed without the use of interrupts, there are several reasons interrupts are required:

1. it is a learning objective of the course;
2. code should end up being much more organized; and
3. time tracking should be more accurate.

As an example of the last, if you were asked to display the time on a slower peripheral (like via a serial port), you would find that the time taken to send out the data would take so much time that your program would easily miss a full period of the stopwatch's 1/100th of second. In fact, you would miss several. This would also be a problem with display on the LCD, though less evident because the time the LCD takes to receive the data is much smaller.

Depending upon how you display information on the LCD, you may note "glitches" appear in the displayed time arising from non-atomic access to your time variable(s). For instance, if the time you are about to display is '00:13:59:99' and it takes more than 1/100th of a second to format the string, you may see something like '00:13:00:13' displayed (this would mean that it took 14/100ths of a second). This is obviously wrong, but will be tolerated for this project, in part because the next refresh will occur quite quickly, but also because the risk associated with failure of our prototype is low. As a workaround, be sure that the time is refreshed at least once when the stopwatch is stopped.

We will learn how to fully address this problem when we talk about the Real Time Loop!

To keep accurate time, you will likely want to use a crystal as the system timing element. For this purpose, each group will be issued one TXC Corporation 9B-10.000MAAJ-B 10MHz +/- 30ppm crystal.

Before coming to the lab

Reorient yourself with the timer peripherals available on the candidate MCUs, noting the interrupt capabilities that these peripherals provide.

Read AVR LibC Documentation section 23.17 to familiarize yourself with how to write AVR interrupt service routines.

Plan your software before coming into the lab. Write and test as much of the software as you can using the simulator that comes as part of Atmel Studio. You may find conditional compilation to prove convenient so that the LCD library does not block simulation when the LCD hardware is not available to respond.