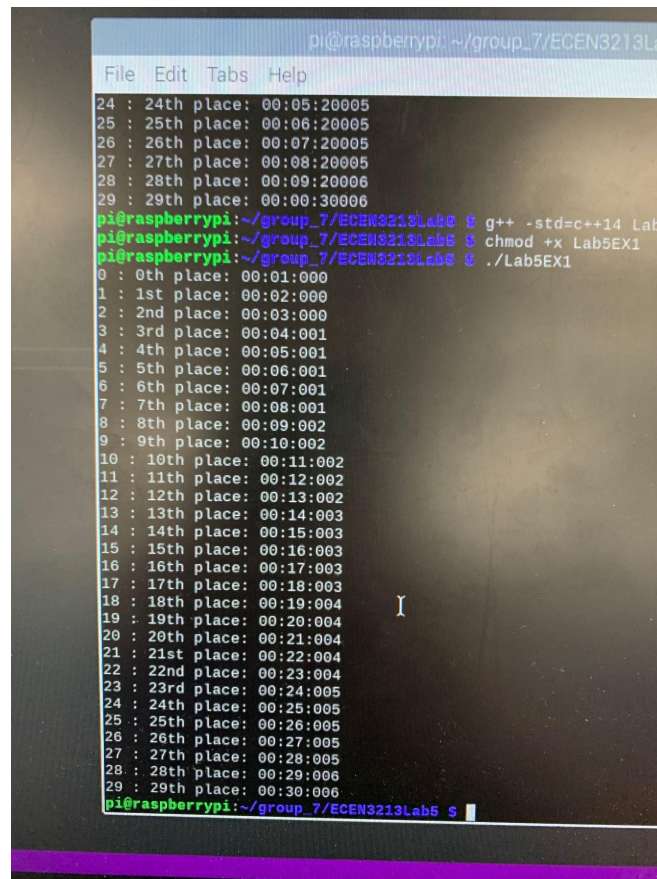


Lab 5: Implementing a Stopwatch

Luke McIntyre & Brendan Bovenschen

1 Exercise 1

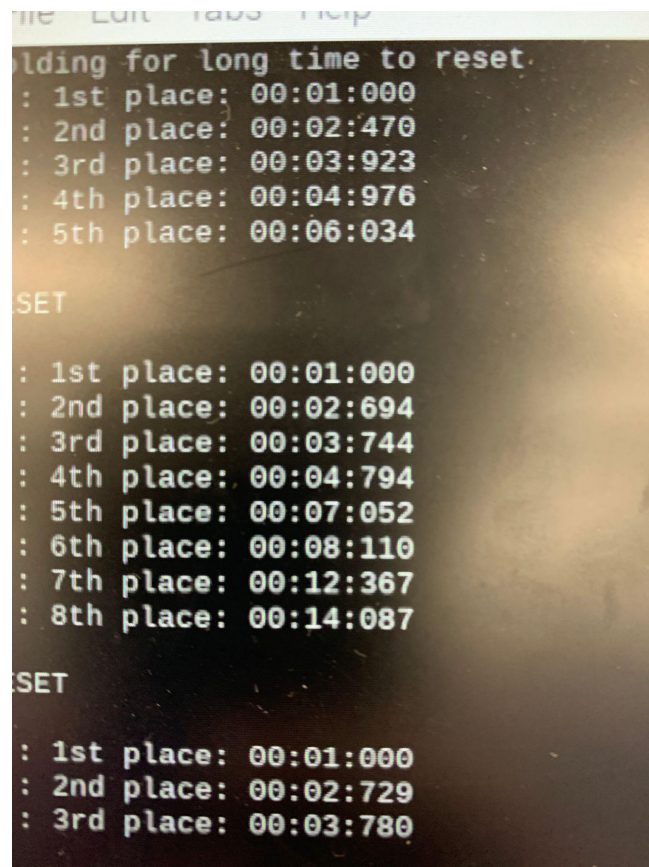
To begin tracking the times of each racer, we must first convert the `high_resolution_clock()` value to a formatted string. This is done through the manipulation of integer division and using the modulo operator to obtain the minutes, seconds, and milliseconds in the clock. The “`sprintf`” function will format these integer values into a 9-character static array (`cstring`). Using this value, we can append it with another string handling the place of the racer. The suffix of this can be calculated from the mod10 of the position (last digit) as well as the mod100 for the exceptions to these grammar rules (last 2 digits, exceptions: 11 12 13). The counter can be handled by incrementing a value by 1 after each place is printed. All this code will be called by an interrupt on the rising edge of our input.



```
pi@raspberrypi: ~/group_7/ECEN3213Labs
File Edit Tabs Help
24 : 24th place: 00:05:20005
25 : 25th place: 00:06:20005
26 : 26th place: 00:07:20005
27 : 27th place: 00:08:20005
28 : 28th place: 00:09:20006
29 : 29th place: 00:00:30006
pi@raspberrypi:~/group_7/ECEN3213Labs $ g++ -std=c++14 Lab5
pi@raspberrypi:~/group_7/ECEN3213Labs $ chmod +x Lab5EX1
pi@raspberrypi:~/group_7/ECEN3213Labs $ ./Lab5EX1
0 : 0th place: 00:01:000
1 : 1st place: 00:02:000
2 : 2nd place: 00:03:000
3 : 3rd place: 00:04:001
4 : 4th place: 00:05:001
5 : 5th place: 00:06:001
6 : 6th place: 00:07:001
7 : 7th place: 00:08:001
8 : 8th place: 00:09:002
9 : 9th place: 00:10:002
10 : 10th place: 00:11:002
11 : 11th place: 00:12:002
12 : 12th place: 00:13:002
13 : 13th place: 00:14:003
14 : 14th place: 00:15:003
15 : 15th place: 00:16:003
16 : 16th place: 00:17:003
17 : 17th place: 00:18:003
18 : 18th place: 00:19:004
19 : 19th place: 00:20:004
20 : 20th place: 00:21:004
21 : 21st place: 00:22:004
22 : 22nd place: 00:23:004
23 : 23rd place: 00:24:005
24 : 24th place: 00:25:005
25 : 25th place: 00:26:005
26 : 26th place: 00:27:005
27 : 27th place: 00:28:005
28 : 28th place: 00:29:006
29 : 29th place: 00:30:006
pi@raspberrypi:~/group_7/ECEN3213Labs $
```

2 Exercise 2

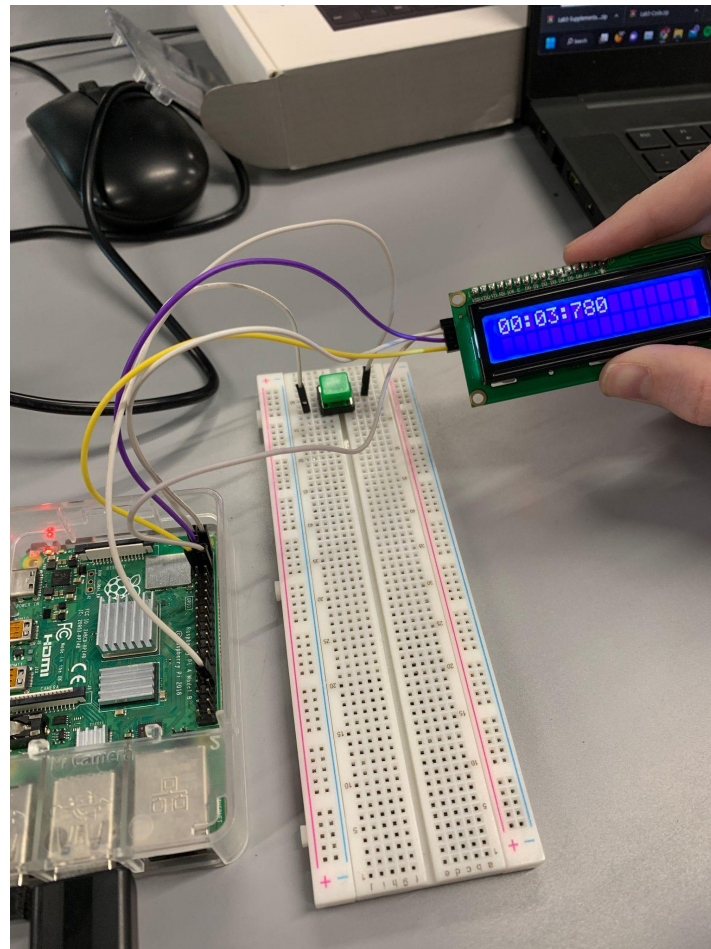
Implementing the same code for producing an output string, handing the racer position, and incrementing the counter, we can add more functionality to our input button with another loop inside our empty “while(true)” loop that handles reset functionality. Through polling, while the button input experiences a high voltage, a block of code to handle the reset function will be run. This code starts by setting the reset timer value to zero, and then counting the timer up until it reaches the reset time value. If the button is released before this is true, the variables will be set to zero and the loop will be broken out of. If the reset is reached, the state will be set to stop, and the global timer will be reset once the button is pressed again. Now, using a state system, we can change the button's behavior based on the current state to perform different functions based on the state.



```
lding for long time to reset.  
: 1st place: 00:01:000  
: 2nd place: 00:02:470  
: 3rd place: 00:03:923  
: 4th place: 00:04:976  
: 5th place: 00:06:034  
  
SET  
  
: 1st place: 00:01:000  
: 2nd place: 00:02:694  
: 3rd place: 00:03:744  
: 4th place: 00:04:794  
: 5th place: 00:07:052  
: 6th place: 00:08:110  
: 7th place: 00:12:367  
: 8th place: 00:14:087  
  
SET  
  
: 1st place: 00:01:000  
: 2nd place: 00:02:729  
: 3rd place: 00:03:780
```

3 Exercise 3

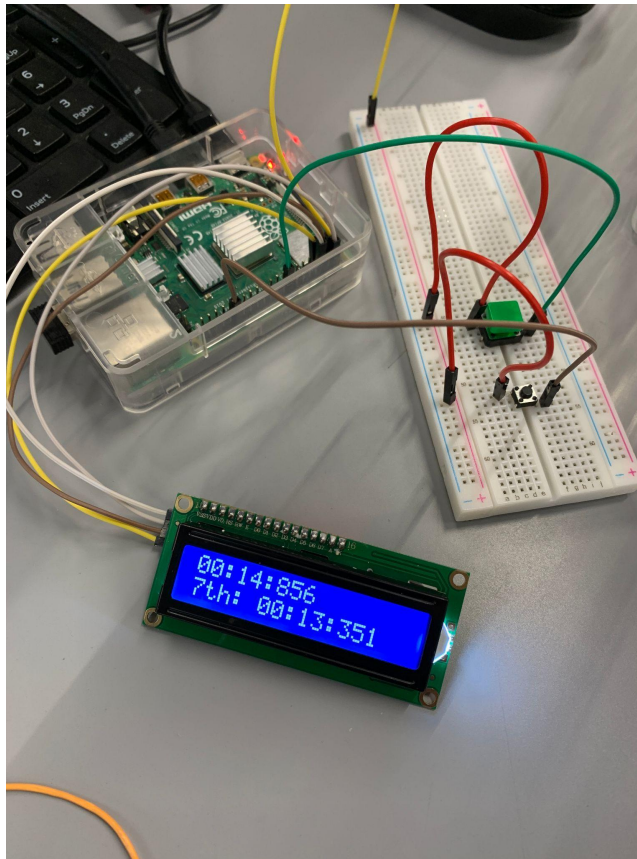
The same system from the last exercise can be utilized with the LCD display with little change to the system. The display will update every iteration of the while loop to the current time, and will be stopped once the system is moved to the stop state. The timer will be reset back to zero after one press, and begin updating again after a second.



4 Exercise 4

The three states will serve a slightly different functionality in this exercise and another system will have to be created to handle the storing and data management of the racers. Once the first button is pressed in the start state, a racer will be saved to an array (with a reasonable size for each racer) at the position of "counter - 1." Later, when in the stopped state, each button will act to create a digital cursor that will shift up and down this array of strings (without traveling outside the bounds of the list.)

Now that the time must be displayed as well as racers in the bottom line, we will have to handle delaying the printing of each text to prevent them from writing to the LCD at the same time. This can be done with a flag, that will, when active, print the current racer after a delay.



5 Supplemental Questions

1. Briefly summarize what you learned from this lab.

Within this lab we were able to implement the `high_resolution_clock()` function in many different applications in order to record specific times that racers finish a race. These times are both recorded and displayed in different ways, depending on the exercise. Within exercise 1, the `high_resolution_clock()` function is converted into a character array at 1 second intervals and then that array is concatenated with the placement of the racer. These values are then displayed directly into the console for the user to see. Exercise 2 implements the same clock function, but instead of printing the racers' positions at 1 second intervals, a button system is constructed on the breadboard that allows for the recording of the time at the specific moment when the button is pressed. The additional functionality of a reset system is included within the code as well, which uses a polling method to count how long the button is being held down for. By doing this a specific time can be set to where when the button is held down for that period of time, the clock will be reset along with the racers, allowing the whole process to start over again. Exercise 3 takes all of the features used within exercise 2, and implements the use of the LCD screen along with the button. This LCD screen is utilized to show the constant increase of the clock, allowing

the user to press the button at very specific time intervals. This is accomplished by updating the characters on the screen for every iteration of the while loop, producing the effect of a constant timer on the LCD screen. The bonus exercise 4 tasked us with not only displaying the running time on the LCD screen, but also displaying the racers and their recorded times on the line below the running time. The addition of a second button within this exercise allows even more functionality to the system, where the first button starts the timer and successive button presses record the times of the racers. The second button will pause the running time and allow the user to scroll through all the different racers and their times, all on the LCD screen alone. Problems arose when attempting to update the screen from these two different functions at the same time, so delays were added so that both functions were not trying to update the information on the LCD screen at the same time. Throughout this lab the `high_resolution_clock()` function has proven to be an extremely useful function that has many different practical applications whether it be for recording the specific times of button presses, or displaying a running stopwatch on an LCD screen.

2. In Exercise 1, when you check the time recorded after each one-second delay, is the time increment exactly one second. If not, please explain the reason.

Although the time recorded for each racer is extremely close to one second intervals each time, there appears to be an extremely small discrepancy of around 1 millisecond every couple of racers. The Raspberry Pi (as well as many computers/microcontrollers) will attempt to run a looped block of programming as many times as possible within a period of time but will be limited by the processing speed of the CPU as well as the number of operations to be performed in this code block. In the case of our timer, the class handling timing functions will attempt to produce the most accurate time it is capable of achieving, but, the Pi cannot perform a poll on the timer every millisecond due to the required operations. This causes the printed time to become delayed as the program enters more iterations.

3. Explain how you use an external function defined outside of the file of the `main()` Function.

The `#import` function allows a programmer to access methods from their own files outside of their main program as well as other libraries holding abstract capabilities. The most common of these utilized in C++ would be “`iostream`” which handles the input and output of information through the command prompt. For this lab, we utilize many functions of the “`wiringPi`” library to communicate with the hardware in the Raspberry Pi. The ability to import from other files and libraries is essential to object-oriented programming as large data structures and objects can be abstracted away from the main developer space.

6 Ideas & Suggestions (Optional)

Ideas: N/A

Suggestions: N/A

ACKNOWLEDGMENTS

I certify that this report is my/our own work, based on my/our personal study and/or research and that I/we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I/We also certify that this assignment/report has not previously been submitted for assessment anywhere, except where specific permission has been granted from the coordinators involved.

Author-1 Signature **Luke McIntyre**

Author-2 Signature **Brendan Bovenschen**

REFERENCES: N/A