Project Final Report - Pomodoro Study Timer

Adolfo Sanpedro Gante and Justinas Janovskis

CS 435: Real-Time Concepts of Embedded Systems

07 December 2024

Abstract

This project is a Pomodoro timer, which decreases mental fatigue while studying by splitting up study sessions with smaller breaks. It takes a dedicated study session time, usually around 25 minutes, and rewards the user with a small 5 minute break after each session. Many students are starting to develop a shorter attention span and have a harder time studying, due to the rise of short form content across all social media, therefore this timer will adapt to that and provide small sessions, which the user can focus on more. After a total of 4 study sessions are completed, then a longer 15 minute break is given to the user. In this project, the user may manually adjust any of the values for the timer such as the interval length to have a longer break between sessions, and the timer is fully customizable. For user comfort, there are a total of 5 Oled displays, which output each timer as well as the total number of sessions studied, and an extra menu for the user to change the timer values through 4 different push buttons. After each session or break, a noise will be played and a corresponding LED will turn on to let the user know that it is time to take a break. The pomodoro timer is a very effective method, and in future versions of this project, it could expand directly onto some social media apps as a setting, which may encourage a student to start studying, instead of spending it all on social media.

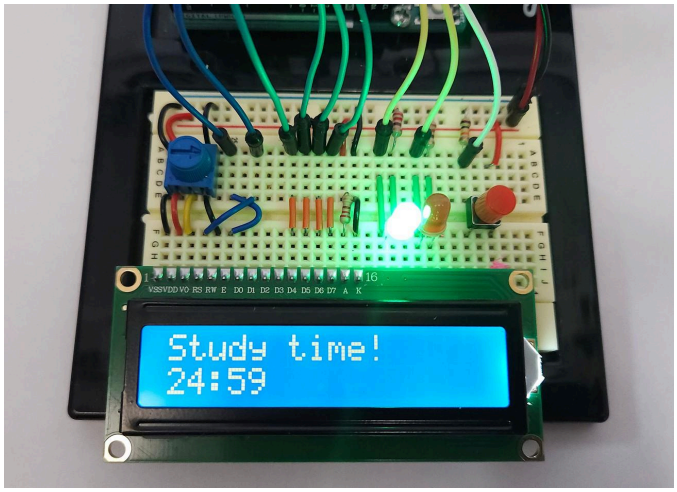
Introduction

Aim, scope, and relevance of the project

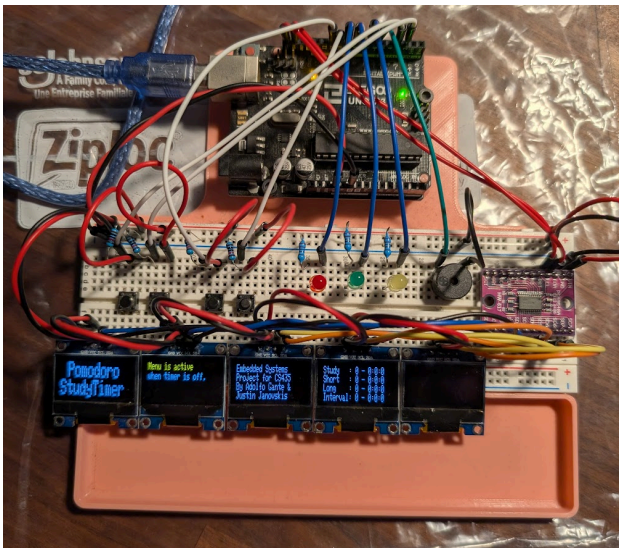
The purpose of this project is to allow a user to have a more interactive studying session where they can manually adjust a timer to their studying needs. Students tend to avoid studying, but with this timer that the user can pick as a default timer, or set themselves, they will be more inclined to study for the time decided. Thus by encouraging them to study for a certain period of time, they will earn a well rewarded break and will allow them to relax for a bit and make sure that the knowledge obtained was fully absorbed.

Similar work and reference material (and how your work is different)

Initial idea was found online [1] and then modified to give the user more input and feedback. The original idea used a LCD screen which we found to be too large, illegible at times and required the adjusting of a potentiometer. Instead we decided to use multiple OLED displays using a multiplexer [2]. A buzzer to give the user an auditory queue that the timer has either started or ended [3] and LEDs to distinguish which timer is currently being displayed. The original idea also only allowed the adjusting of each timer and the long break interval within the code only instead we will allow the user to adjust timers and the long break interval using pushbuttons. One push button to select which timer or the long break interval to adjust and 2 push buttons to increment and decrement the selected timer. We will also have a start/reset push button. The following pictures show a clear difference from the original idea and our project.



Original Idea Study Timer [1]



Conceptual design of the system

- Components needed
  - Arduino UNO (1 total)
    - https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf
  - SD1306 I2C OLED Display 128x64 (5 total)
    - https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
  - TCA9548A I2C Multiplexer (1 total)
    - https://cdn-shop.adafruit.com/datasheets/tca9548a.pdf
  - LEDs (3 total)
    - https://www.farnell.com/datasheets/1498852.pdf
  - Push button (4 total)
    - https://components101.com/sites/default/files/component_datasheet/Push-Button.pdf
  - Buzzer (1 total)
    - https://components101.com/sites/default/files/component_datasheet/Buzzer%20Datasheet.pdf

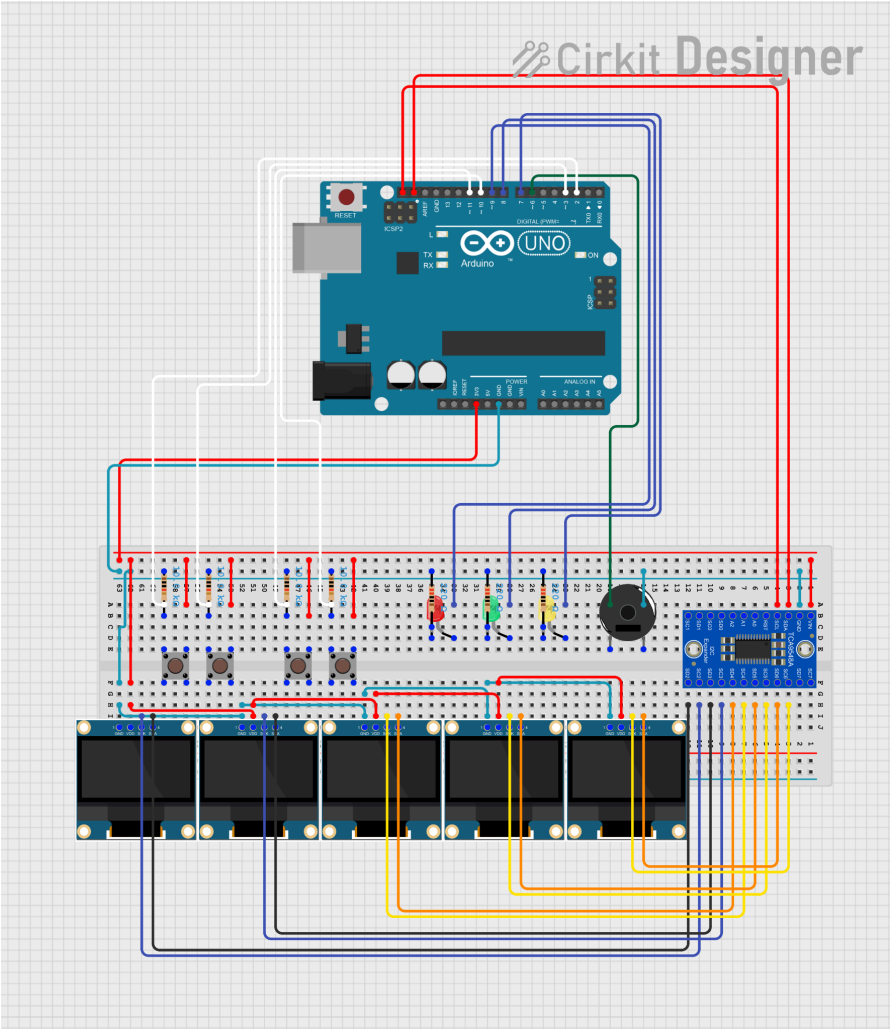
- top level flow-charts or PDL

```
//Initialize variables
pushbutton mode;
pushbutton up;
pushbutton down;
pushbutton start;
led modeAuto;
led modeUser;
led infinite;
buzzer audio;
digitalOut display;
int minutes;

if(autoMode)
    Turn led modeAuto on;
    for(4 iterations)
            Activate buzzer; //To make noise that it started
            Start timer;
```

```
            while(timer < 15 mins)
                    Update display;
            Stop timer
            Activate buzzer twice; //signal end of timer
            for(5 mins)
                    Blink led; //Every second and then repeat
if(userMode)
    userLED = true;
    while(true)
        if(start pressed)
            startUser()
        if(up pressed)
            minutes++;
        //Minimum time for the timer intervals is 1 minute
        if(down pressed AND timer > 1)
            minutes--;
if(infiniteMode)
    infiniteLED = true;
    while(true)
        Activate buzzer; //To make noise that it started
        Start timer;
        while(timer < 25)
                update display
        Stop timer
        Activate buzzer twice; //signal end of timer
        for(5 mins)
                Blink infiniteled; //Every second and then repeat

startUser()
{
    for(4 iterations)
            Activate buzzer; //To make noise that it started
            Start timer;
            while(timer < minutes)
                    update display
            Stop timer
            Activate buzzer twice; //signal end of timer
            for(5 mins)
                    Blink led; //Every second and then repeat
}
```

Implementation details

Hardware design/setup

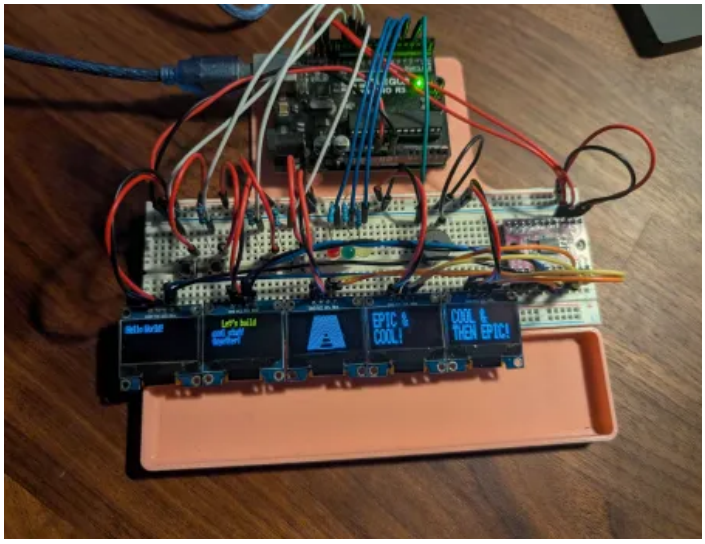


Implementation of the code

We decided to house the timer code within a switch statement because we wanted a default state where the menu would be active whenever the timer was not. This also allowed the start/stop interrupt button to break from the individual timer loops and go straight to the menu. We also decided to house the menu in a switch statement where every case was used to increment/decrement the timer values and then the start/stop button could interrupt and go straight back to the timer's loops. We found that interrupt buttons and switch statements would work well together. We also quickly noticed that we had a lot of repeating code and therefore it would benefit to place that code in functions we could call. That helped with the readability and logic of the program. We also decided to implement our own debounce handling as a learning opportunity instead of relying on libraries. We used the `millis()` function to calculate the needed values to handle debounce. We also decided to use a library to handle the multiple oled display because otherwise this would have added a lot of complexity to the project.

Experimental results

The following picture is initial testing of the oled screens and multiplexer. The screen would constantly blink which we found distracting. After some experimentation we found that instead of updating the screens every single loop iteration we should instead only update the screens when text was updated.



After implementing the timers we found that the time did not match real time. For example, a 30 seconds timer would actually be 34 seconds. We tested every timer using the `millis()` function which returns the time since the program started to when it was called. From that value we could calculate the exact time a timer would take in milliseconds. Unfortunately, we were unable to achieve high precision with our timers; we instead achieved an average of 1.2 seconds. That is to say one tick of the timer is equal to 1.04 real time seconds. (30 + 1.2) / 30 = 1.04 seconds.

Lessons learned

Project management, task allocation, and team communication

We had a causal approach to project management. We would discuss the project in class or over Discord and while we did communicate effectively; we could have had better time management with a more rigid approach.

External learning resources used to complete project

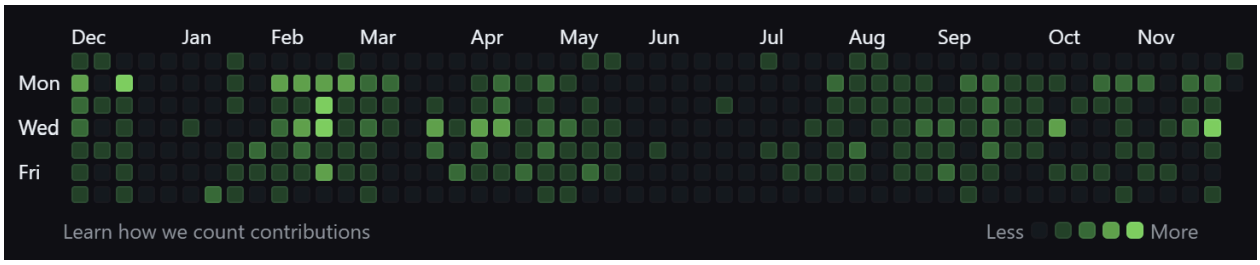
We learned about Arudino, specifically the Arduino UNO, which is extremely popular in the embedded systems world. We learned how to code using Arudino's flavor or C++ and upload code to the Arduino UNO using the Arduino IDE. We utilized the Arduino documentation, youtube videos and example projects/code from guides.

Challenges faced by each team member

We both faced quite a few challenges in system design and implementation. Namely the button debounce, we did not use a library to fix this but instead decided to implement our own debounce handling. Testing was also a bit tedious because we would need to constantly start the timer and then wait to get results, analysis, fix and retest. This process was a bit frustrating. Adolfo handled the hardware which was probably the easiest portion of the project, he did not encounter challenges except one. The Arduino UNO only supports one I2C connection but we had five. This was quickly fixed with a google search which led us to using an I2C multiplexer giving us eight I2C connections, more than we needed. Justin worked on the timers and quickly discovered the accuracy of the timer which added to the tedious process of testing to find a suitable delay time for accurate timers.

Future work and desirable changes if you had more time to work on the project

If we had more time to work on the project we would definitely add a RTC module [6] to properly calculate time and give exact results. We would also give the user the option to have their Break times is automatically adjusted to be ⅕ of study session time so they don't have to manually adjust timers every study session. We would also like to add session tracking with weekly reports sent by email. We could achieve this by adding an ESP32 [7] and have a similar graph to Github's contributions chart. In our case the color would deepen based on the number of hours studied that day.



Summary

The Pomodoro Study Timer project is a customizable embedded system designed to improve study efficiency by alternating study sessions with breaks. It uses an Arduino UNO, multiple OLED displays, LEDs, a buzzer, and push buttons to create an interactive and user-friendly experience. The project is different from similar designs by improving display readability, incorporating button-based timer adjustments, and adding auditory and visual feedback. Challenges like button debounce and timer accuracy were resolved through custom implementations. The project had successful functionality with minor timing deviations. We learned valuable lessons about project management, Arduino programming, and hardware design. Future improvements include adding real-time clock module, dynamic timer adjustments and session tracking with reports via an ESP32 module.

Reference/Bibliography

[1]: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Elec_p099/electricity-electronics/pomodoro-study-clock

[2]: https://www.youtube.com/watch?v=MO6hbQcX8fE

[3]: https://www.youtube.com/watch?v=gj-H_agfd6U&t=51s

[4]: https://docs.arduino.cc/built-in-examples/digital/Debounce/

[5]: https://reference.arduino.cc/reference/en/language/functions/time/millis/

[6]: https://randomnerdtutorials.com/guide-for-real-time-clock-rtc-module-with-arduino-ds1307-and-ds3231/

[7]: https://www.espressif.com/en/products/socs/esp32

Code listings

https://github.com/adoante/CS-435-Final-PomodoroStudyTimer

A link to the video recording of your demonstration

https://youtu.be/8dqkHUR8TN0