Department of Design Engineering and Mathematics

Middlesex University

DIGITAL ALARM CLOCK WITH TIMER AND STOPWATCH

Adam Sparkes (M00759855), Andrei Tocu (M00797763), Alaa Mubarak (M00780329), Yitzchak Meyer (M00780093)

Prof. Raja Nagarajan

PDE2101 Group project

BEng Computer System Engineering/ Electronic Engineering

April 2022

Table of Contents

1. Abstract

This project is centred around creating an Arduino device with set functions which are useable. This includes using various sensors connecting to a microcontroller to develop a certain output which can be displayed on a lcd display. There are many ways to proceed with Arduino based projects as there has been many different directions you are able to go towards. In the project we introduce further below, it will go through how we went about coming up with the idea of the project, the steps needed to work on for the final outcome to succeed and overall how we split the project into sections as it is a group work. Thus making everyone having to put in a certain level of input to ensure the correct amount of contribution was fair.

A clock can be in many different forms, analogue or digital. Alongside with all the different functions a clock can coincide with to allow it work in a more better way. This is the main part of what we work on with the project, how we have taken the standard function of a clock and what we can make it do and how it is possible to implement the ideas we had into the clock and its functions. But still ensuring it still keeps the key designs and structure of how a clock should act as.

2. Introduction

The aim of this project was to design a device which used a microcontroller and create a programme for it where it has multiple functions. We decided to use the microcontroller ItsyBitsy nRF52840 Express2 which is manufactured by Adafruit. The platform we used to do the coding for the programme was Arduino.

This project report will go through the design and implementation of an Arduino-based digital alarm clock with the additional features of a timer and stopwatch. The implementation includes a combination of hardware and software. The project aims to use an Arduino board, as well as various sensor and interface devices. The design will be validated by building a prototype. Going further into the report it will state the different sensors and components needed for our project and what we did as a group to put it all together to get a functioning digital alarm clock with the features of a timer and stopwatch.

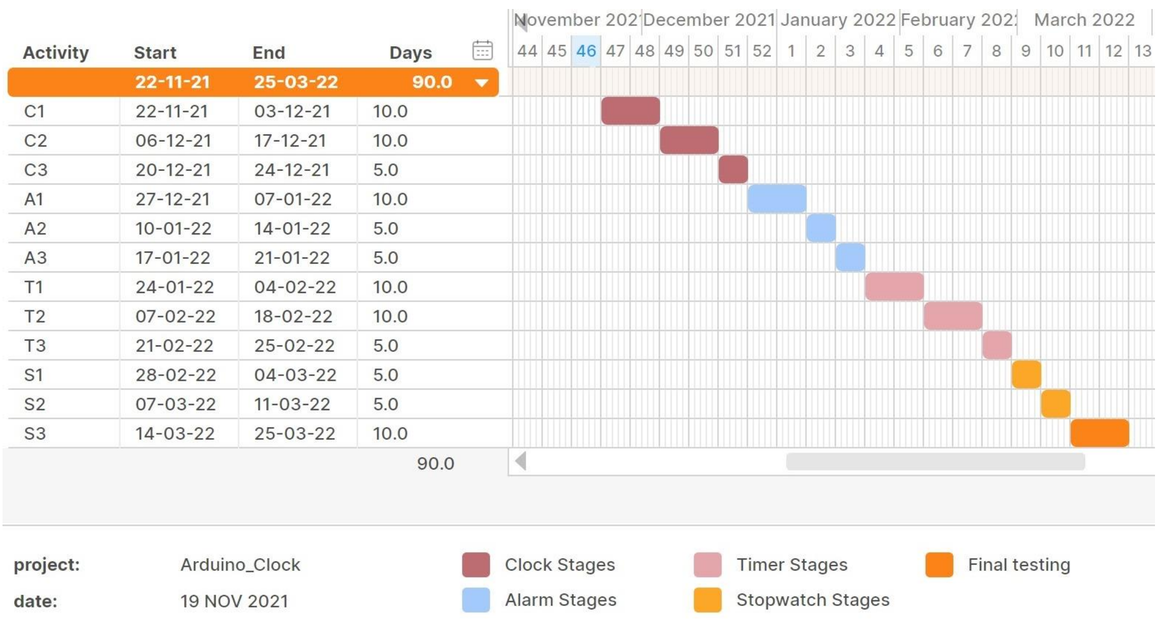
The reason why we have chosen to create an alarm clock with the following functions, is due to it being useful in day-to-day life. This allows the device to be very usable on multiple occasions whether you are needing it just to tell the time or if you need it for more of its functionality for example set a timer for a reminder of an event and many more reasons why you would want to use it. It is important creating a programme and device which is versatile towards many different reasons in using it, as it allows flexibility in the usage of what you are able to do with it, ultimately being more valuable to the user. Which overall makes the project we aim to create a success.

There is two parts to our project, the programming part and the devices hardware structure part. The programming side will all be done through the Arduino IDE (Integrated development environment), this gives us a good control declaring the sensors on the board and implementing them to work together to create the working device. This links into the devices hardware structure side which is mainly how we connect all the needed sensors with the correct jump wires. This allows the connection for the computer to get the reading which our programme will be able display what is needed on the LCD screen.

Splitting the project into two parts works out a lot better in many ways. One to make it more organised when working on the certain part of the project but also to do the research before starting to work on the clock project. Researching is very important for any project, as it gives an idea of where to start off and what is needed to do. Even when thinking of what we are trying to achieve, a clock with an alarm, timer and stopwatch displaying on a lcd screen. Also, when splitting it into two different parts it is easy to assign what work is needed to be done by each group member, as the strengths of one person may differ to someone else. This lets us focus on all of our strengths and putting in our best input towards what we will achieve, a fully working project.

3. Project planning

In any project it is critical to implement a clear plan for how we will all reach to the project's final goal in every endeavour. This is mainly achieved by creating and implementing a good plan where all the group members understand the objectives needing to be passed to reach the final goal of the project, a fully working device and programme. We developed a Gantt chart to allow all of us to have a clear view on what we individually need to ensure gets done but also where the project will hopefully be at, at any point of time. The importance of the Gantt chart in figure 1, allows you to see the rough layout of how the project is planned out within the group and what each group member is working on in the timeline.



Activity	Start	End	Days
	22-11-21	25-03-22	90.0
C1	22-11-21	03-12-21	10.0
C2	06-12-21	17-12-21	10.0
C3	20-12-21	24-12-21	5.0
A1	27-12-21	07-01-22	10.0
A2	10-01-22	14-01-22	5.0
A3	17-01-22	21-01-22	5.0
T1	24-01-22	04-02-22	10.0
T2	07-02-22	18-02-22	10.0
T3	21-02-22	25-02-22	5.0
S1	28-02-22	04-03-22	5.0
S2	07-03-22	11-03-22	5.0
S3	14-03-22	25-03-22	10.0
			90.0

project: Arduino_Clock
date: 19 NOV 2021

Figure 1 – Gantt chart

As shown in the Gantt chart, the project is split up into many different stages; clock, timer, alarm, stopwatch and the final testing stage. This is to allow progress on the project allowing more time to be worked on a certain stage if it is needed to. We will know that we have achieved the final goal of our project once we have the following functions, Displaying the current time, a repeating alarm at specific times of day, a one-time alarm after a specific amount of time has passed (a timer), a way to measure time as it is passing without a specified end (stopwatch).

4. Resources

It is important with any project to be able to understand how to use many different areas, hard and software to be used. This is all the different recourses of the project. In the hardware area it required us having to us a microcontroller, certain sensors, an LCD screen, a breadboard and various cables to connect the different sensors up with the microcontroller. The microcontroller which purpose will be for all the computing parts of the clocks features is the ItsyBitsy nRF52840 Express2, this features a Bluetooth LE 32-bit ARM Cortex M4 processor at 64MHz. It has 256kb of RAM (Random Access Memory) and 1024kb of flash (ROM) with another 2mb of QSPI flash chip for sorting files and circuit python code storage. It is capable of running Arduino code which makes it compatible with the device we are ultimately developing as it is primarily running through Arduino.

Regarding the software side of the project, it includes using Arduino to control and read the sensors and get the data into the computer where it is able to compute it and display the data. Therefore the programming language being used will be C++, this is the code used in the Arduino IDE. Alongside using Arduino it was all developed on Windows operating system, this allows us to install certain libraries for the LCD screen on Arduino for it to be able to understand the certain programming functions.

These were the two main resources used to create and work on with the clock project we had set out to achieve. It is important to learn and research the different resources we could have used instead of the ones we chose to; this gives a better understanding of how the overall clock and LCD screen would work alongside each other but also you may find a better, more efficient way of getting to the final outcome while still ticking off the main points of what we set out to do.

5. Methodology

The entirety of this project had to be done in a relative short time and need to have the entire team involved in the making of it. Having multiple modules, variables, designs that we needed to work with meant that we needed to have a rapid and incremental improvements in the delivered software. The communication between the team members needed to be continuous and efficient, so that we can all benefit from each other's work and the workload divided such that multiple members won't be working on the same functionality or having incompatible code. The approach we would take in making this project needed to be based on a development methodology that can accommodate well all of the above listed requirements. The development methodology that can be described best by these requirements is Agile methodology.

The Agile methodology can accommodate a big number of team members or workers that are taking on a project. In its description it is also defined that the clients, for which the projects are targeted, can very well be involved in the development process. Having such a close relationship between the team members themselves and the clients can only yield to a well understood and planned project, leaving no room for misinterpretation or misunderstandings. Having a self-organized team, an involved client and frequent feedback, allows for a constant improvement through incremental releases, rendering Agile methodology a fast, efficient and cost-effective approach for our project as well as for bigger and more complex ones.

6. Development and implementation

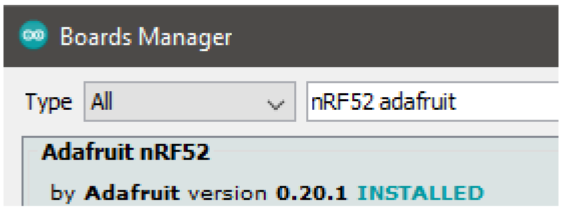
6.1. Setting up the IDE for nRF52

Before starting to work on the project we need to make sure we have the drivers and compatibility needed to access our board's features. We can do that by:

1. Opening the Arduino IDE
2. Go to top menu, File – Preferences
3. In the "Additional Boards Manager URLs" field we need to add the following link

```
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
```

4. Restart the IDE
5. On the top menu navigate to Tools – Board – Boards Manager
6. Search in the top field for the "nRF52 Adafruit"



nRF52 adafruit

Adafruit nRF52
by Adafruit version 0.20.1 INSTALLED

7. Select version 0.20.1 then click install
8. From top menu navigate to Tools – Board – Adafruit nRF52 Boards and select from the list *Adafruit ItsyBitsy nRF52840 Express*
9. Finish. Now we have the correct board installed so we can start working.

6.2. LCD

The Liquid Crystals Display (LCD) has a 2 row and 16 column block arrangements that can be used as an output device for anything: numbers, letters, sentences or even custom-made characters that can be displayed on the individual blocks. The LCD is also backlit, which makes it easy to read the characters even in the dark.

The LCD has a total of 16 pins that are available for use.

- Register Select (RS) pin that is connected to the LCD’s memory, responsible for what is displayed
- Read/Write (R/W) pin that dictates the flow of data – write or read
- Enable pin that allows for writing to registers
- (D0-D7) A total of 8 pins for data
- Display Contrast (Vo) pin
- Backlit cathode and anode pins
- Other Power Supplying pins
- In order to dim the backlit of the LCD we needed to use 2x 220Ohm resistors to prevent frying the backlit

We only needed to use 12 of them. 4 of these are used to connect to GND, 2 of them connect to 5V and the rest of 6 are used for data transfer, all of these pins are digital.

In the process of adding the LCD we used an online tutorial that is providing all the information listed above and is breaking down the process and inner workings of the LCD and how to use the library’s functions inside Arduino.

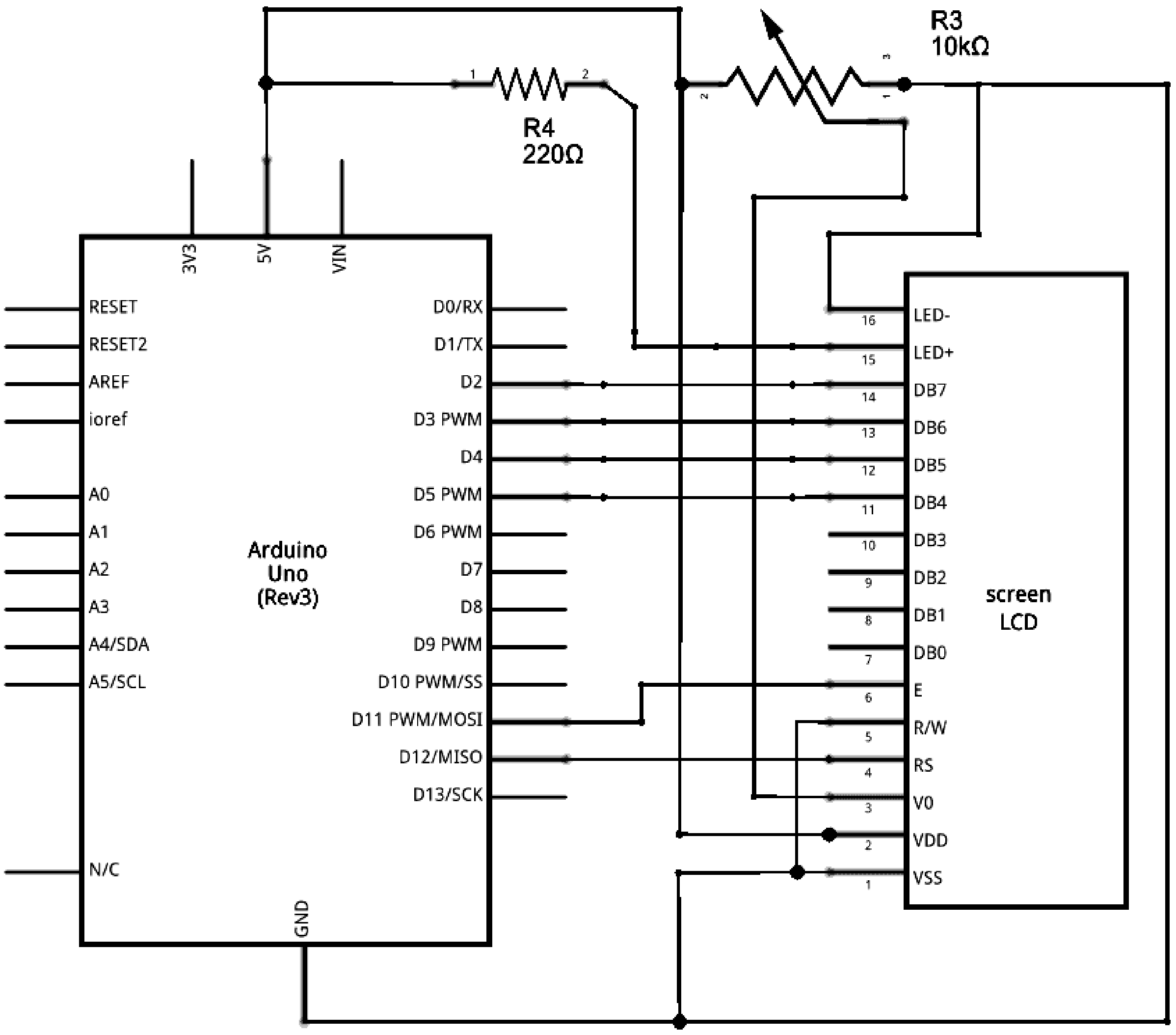


Figure 1 LCD Arduino module circuit [1]

The above diagram provided by the website is using a potentiometer that is connected at the V0 pin that we did not find useful in our case.

To make use of the display we used the available libraries of the hardware, LiquidCrystal.h [2]

```
#include <LiquidCrystal.h> // LCD library/driver

const int rs = 12, en = 11, d4 = 5, d5 = 7, d6 = 9, d7 = 10;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // assigning pins to LCD
void setup() {
    lcd.begin(16, 2); // start LCD of size 16 cols 2 rows

    lcd.setCursor(0,0); // set cursor on col 0 line 0
    lcd.write("HELLO00"); // cursor writes this
    lcd.setCursor(0,1); // set cursor on col 0 line 1
    lcd.write("WORLD!"); // cursor writes this
}
```

Since we need to print many things on the display, we created auxiliary functions that will take care of display printing.

As specified in the snipped above, we need to mention where the cursor is located on the display in order to print, otherwise we will just keep writing out of bounds, at the end of already existing text.

Due to the way the LCD library is constructed we can have multiple print lines to concatenate on the display. For example, if we will be printing once “MIDDLESEX” and then “HELLO”, the first 9 columns of the display will be populated by the first word, then when printing the second word, considering we reset the cursor to column 0, it will display “HELLOESEX” due to the fact that the previous print was larger and so there are residual characters. There are different approaches to this matter, like keeping track of what is already displaying and what is about to display, and so if the length of new value to print is smaller than the previous one, we can fill the rest with empty characters. But a better way of clearing the screen is by using the existing function *lcd.clear()* that takes care of the entire memory block of the LCD, even what is past the 16 columns, wiping everything.

Since we need to print many things on the display, we created auxiliary functions that will take care of display printing and will take care of all the behaviours explained above. The best approach we thought would be to have a function in which we can have a separation between 1st and 2nd line.

```
void printToLCD(String str0, String str1){ // line1, line2
  if(millis()-lastframe>250){ // remove screen flicker
    unsigned int len, len0 = str0.length(), len1 = str1.length(), index=0;
    len0>len1?len=len0:len=len1; // check which line is longer
    lcd.clear(); // clear LCD
    lcd.setCursor(0,0); lcd.print(str0); // print first line
    lcd.setCursor(0,1); lcd.print(str1); // print 2nd line
    lastframe = millis(); // update last frame time - avoid flicker
    while(len-index>15) // check if print is bigger than 16 chars
      if(millis()-lastframe>250){
        lcd.scrollDisplayLeft(); // scroll text to left
        lastframe=millis(); // update last frame time
        index++; // update position of first char on display
      }
  }
}
```

The way Arduino works is looping the code forever and depending on the code this can happen hundreds of times per seconds. Because of this our LCD can flicker as the refresh rate of the display and our eyes differ, so we implemented the variable that keeps track of when the last frame was posted, *lastframe*, and we will not be updating the display earlier than 250ms, making a refresh rate of around 4 frames per second and now the display appears as it is not flickering anymore.

[3]In order to keep track of the time even when the device is powered off, we will need a device that has its own power source and that we can input such data like date and time, and we can request this data anytime we want. This device is called RTC [4] – Real Time Clock and it uses just 4 pins:

- GND – GND, ground
- VCC – USB pin, 5v connection
- SDA – SDA, serial clock sync
- SCL – SCL, serial data

```
#include "RTClib.h" // RTC library/driver
RTC_DS1307 rtc; // RTC variable
DateTime now = rtc.now(); // getting values from RTC
```

The SDA pin is used for data transfer, input and output, using the i2c interface, while the SCL is used for synchronisation with the microcontroller.

Before using the module, we need to make sure it uses the correct date and time. The time can be set using function *rtc.adjust(DateTime(year, month, day, hour, min, sec))*. This can be run just once and the RTC will be set and will keep track of time from the set value.

To get any output from the RTC we can use the functions made available by the library. For example as code nippet as *var1 = now.day(), var2 = now.hour()* will get the values of day and hour from the RTC module. Now having these values, we can use them however we want. In the case of the clock, we made two strings, one for time format and one for date format.

```
String getDateStamp(){ // collect Date value
    DateTime now = rtc.now();
    String datestr;
    datestr += now.day(); datestr += '/';
    datestr += now.month(); datestr += '/';
    datestr += now.year();
    return datestr;
}

String getTimeStamp(){ // collect Time value
    DateTime now = rtc.now();
    String timestr;
    timestr += now.hour(); timestr += ':';
    timestr += now.minute(); timestr += ':';
    timestr += now.second();
    return timestr;
}
```

These strings can then very easily be submitted to print on the LCD such as:

```
printToLCD(getTimeStamp(), getDateStamp());
```

6.3. Joystick [5]

Due to the high number of digital pins used by the other modules we will be having difficulties later on connecting other modules that require digital pins, so we had to find a way of making navigation within the menus possible whilst using analogue pins.

Initially, in the proposal we specified that we will be using the numpad membrane switches in order to input values and navigate the menus. This was not possible, as we only had one more digital pin available and a couple analogue ones. After a few hours of brainstorming, we decided to go with a joystick that requires just one digital pin and two analogues, plus ground pin.

Making use of the joystick for our project proved to be a bit more difficult than we would have expected, as moving the stick up/down/left/right would not output predictable values: 0 to n or -n to n. It would just have a range of values that we needed to monitor and map for each of the forementioned movements, including mapping values for the centre position of it.

```
  valX = analogRead(joyX);
  valY = analogRead(joyY);
  posX = map(valX, 0, 1023, -9, 9);
  posY = map(valY, 0, 1023, -9, 9);
  if(posX == 4 && posY == -9) pos = 1; // up
  else if((posX == 0 || posX == 1) && (posY == 6 || posY == 5)) pos = 2;
//right
  else if(posX == 5 && (posY == 0 || posY == 1)) pos = 3; // down
  else if(posX == -9 && posY == 4) pos = 4; // left
  else if(posX == 4 && posY == 4) pos = 0; // center
```

In the code above we are getting the raw output of the joystick and then mapping it to a range from -9 to 9 for both axes in order to make the ranges smaller and easier to determine their positions. Having now tested the joystick with the mapped values we can hardcode the values to that respective position of the joystick. Possible positions are centre, up, right, down and left; and these positions have assigned to them these values: 0, 1, 2, 3 and 4 respectively.

6.4. Navigation

Now that we know in which position, we have our joystick set, we have to come up with a way of navigating through the menu. We decided that in order to not overcomplicate things, we can have the up/down functions to navigate us through the main menu options, while the left/right movement can navigate us through the submenus of these. Once we have the right option displayed on screen, we can select it by pressing the joystick's button which will act as a selector or enter/return button of a keyboard.

```
  if(millis()-lastpress > 500){ // avoid spam
    if(pos!=lastpos){ // actioning only when current pos is different from
last pos (avoid spam)
      switch(pos){ // current joystick pos
        case 0: // center
        break;
        case 1: // up
          menuVal++;
        break;
        case 2: // right
          submenuVal++;
        break;
        case 3: // down
          menuVal--;
        break;
        case 4: // left
          submenuVal--;
        break;
        default: // any other - unknown
        break;
      }
      lastpos=pos; lastpress=millis(); // update last pos to current pos
    }
  }
```

In the code above we can observe that we have a variable lastpress. What this function does is taking a timestamp of the milliseconds since the board was powered on, lastpress=millis(). This is a way of avoiding SPAM input or false inputs that can happen due to the fact that the microcontroller is able to take hundreds of inputs per second. For example, when moving the joystick up just once, due to the high input rate, it will not show the next input as we want, but a random one, depending on where the module stopped taking inputs when we let go of the joystick. Therefore, the lastpress is used as a measurement for last input occurrence and will take a new input only when the time between last input and current one is over 500ms, so it can take two inputs per second, thus avoiding cases of input flooding and inaccurate selections. This can be found in multiple areas of the code where input timing is important and is similar to the workings of the frame refresh rate of the LCD.

6.5. Menu

We have 4 main menu options and up to 3 submenus for each. We will list those as follows:

- Menu, [menuVal][0]
    - Submenu, [menuVal][submenuVal]

- Clock, [0][0] – it has no submenu, once selected on the display we can see the date and time in real-time. Time format H:M:S; Date format D/M/Y

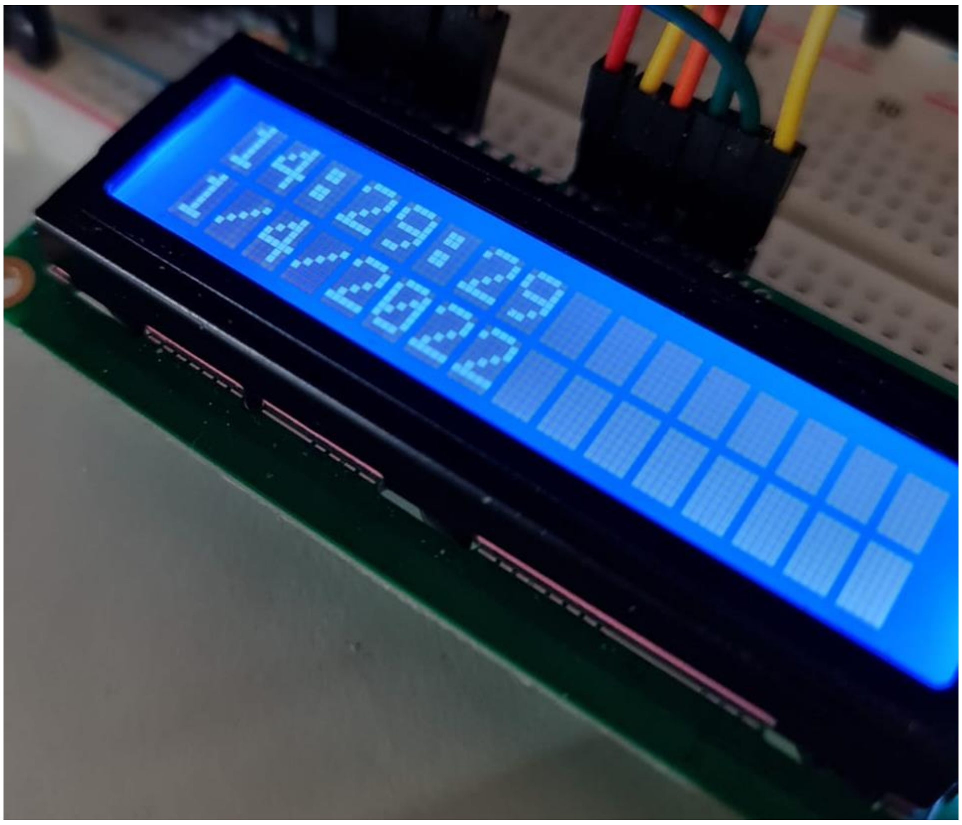


Figure 2 The digital clock displays time and date

The data such as date and time is being pulled off from the RTC module for each frame refresh.

- Alarm, [1][0]
    - Enable, [1][1] – If no alarm is ongoing user is able to input a time to which they want to set the alarm to go off at. After the time was input, it will send the user in the “check” submenu. If there is an ongoing alarm, then it will send the user in the “check” submenu. Once it reaches the set time it will ring the buzzer and will set user into the “check” submenu.
    - Check, [1][2] – From here the user is able to see the time to which the alarm was set to. If no alarm was set it will display “not set”.
    - Disable, [1][3] – Once selected it will reset the current alarm and send the user in the “check” submenu.

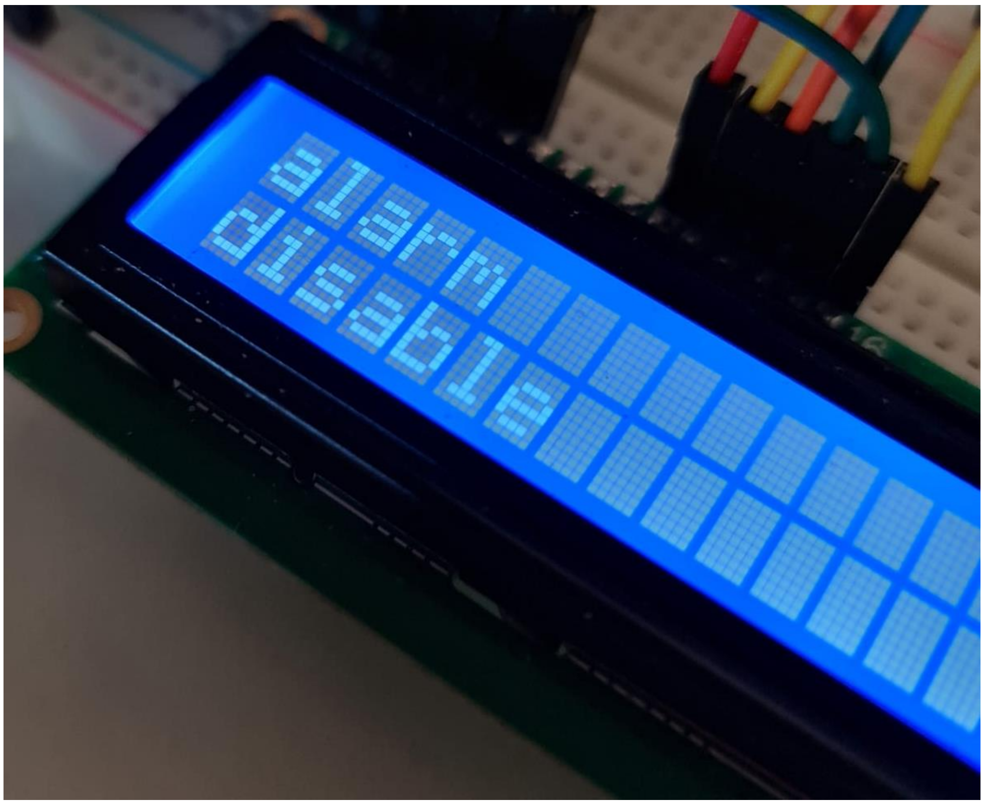


Figure 3 The alarm

- Stopwatch, [2][0]
    - Start, [2][1] – If the stopwatch was not started yet, when actioned it will start the stopwatch and send user inside “check” submenu. If it was actioned already, it will send the user inside “check” submenu.
    - Check, [2][2] – Here the user can see the time passed since the stopwatch was actioned.
    - Reset, [2][3] – When actioned it will reset the stopwatch and send the user in the “check” submenu that will display “not started”.

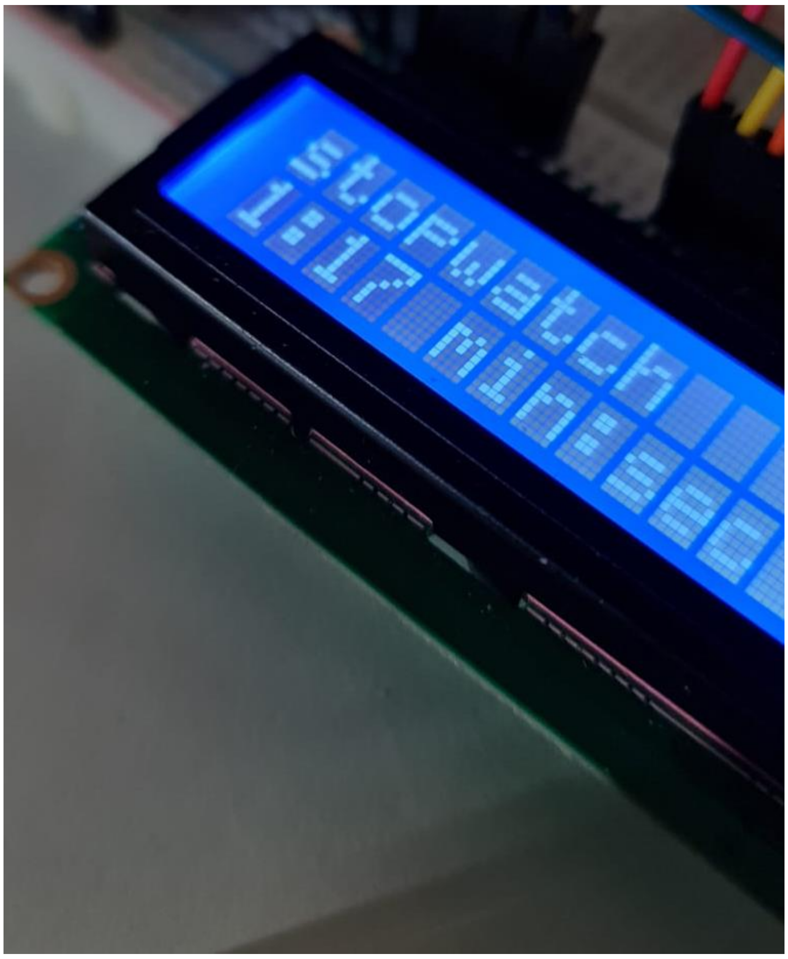


Figure 4 The stopwatch

- Timer, [3][0]
    - Set, [3][1] – If no timer is ongoing the user can set a minute and second type of value from which, when set, the device will countdown. Once it reaches 0 it will sound the buzzer and will send the user to the "check" submenu.
    - Check, [3][2] – Here the user can see the time left till the timer runs out or if there is no timer ongoing it will display "not set".
    - Reset, [3][3] – When actioned the timer will be reset to 0 and will send the user in the "check" submenu.

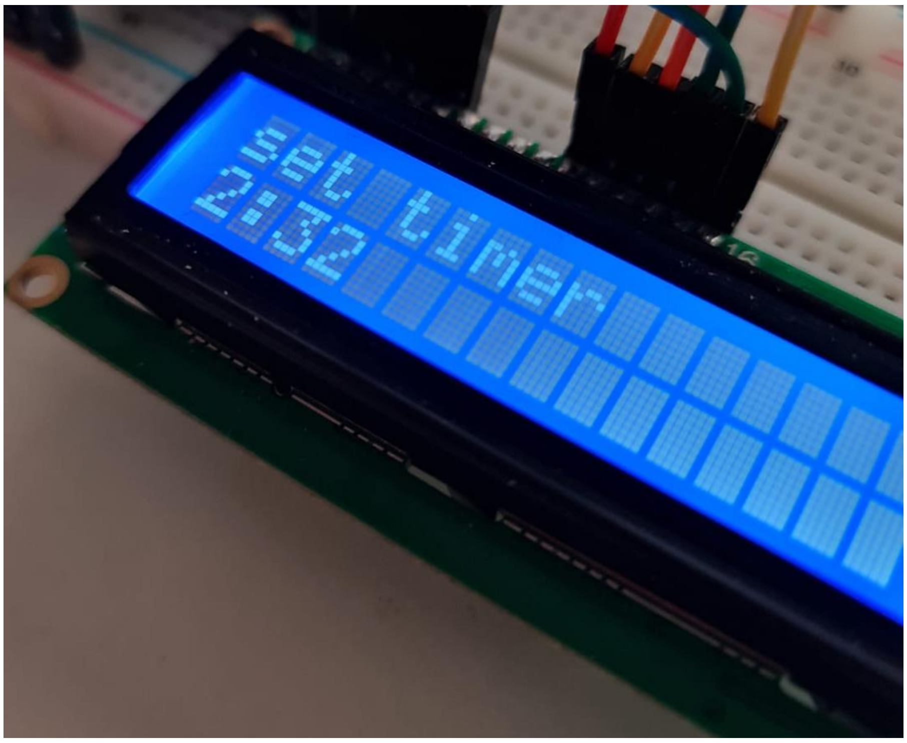


Figure 5 The timer

Each one of the menu's values are stored in a two-dimensional array.

```
char* menuTitle[4][4] = {
    {"clock"},
    {"alarm","enable","check","disable"},
    {"stopwatch","start","check","reset"},
    {"timer","set","check","reset"}};
```

We can see here there on each row we have a menu on position 0 and their submenus after that. This way we can have an easy way of navigating the entirety of the menu by just incrementing and decrementing values and using the remainder from dividing that value to the number or entries that menu has.

```
void showMenu(unsigned int &menuIndex, unsigned int &submenuIndex){
  menuIndex=menuIndex%4; // 4 = number of total menus
  switch(menuIndex){ // check selected menu index
    case 0: // clock
      submenuIndex=1; // display nothing - clock has no submenu
    break;
    case 1: case 2: case 3: // alarm, timer, sw
      submenuIndex=submenuIndex%4; // 4 = number of entries per menu
      if(submenuIndex<1) submenuIndex=1; // don't go below 1
      if(submenuIndex>3) submenuIndex=3; // don't go above 3
    break;
  }
  printToLCD(menuTitle[menuIndex][0],menuTitle[menuIndex][submenuIndex]);
}
```

The variables menuSel and submenuSel represent the current menu that the user is in, and these will have a negative value when the user is not in any menu at that time otherwise these will have the v

```
  while(!digitalRead(sw) && pos==0){ // when the joystick is centered and
button actioned
      switch(menuSel){ // check in which menu we are currently
        case -1: // we are in no menu
          menuSel = menuVal; // enter displayed menu
          submenuSel = submenuVal; // and submenu
        break;
        default: // we are in a menu
          menuSel = -1; // exit current menu
          submenuSel = -1; // and submenu
        break;
    }
    lastpress=millis(); // avoid spam
  }
  while(digitalRead(sw)){ // while joystick button is not actioned
    if(menuSel!=-1 && submenuSel!=-1) // we are currently in a menu/submenu
      enterMenu(menuSel, submenuSel); // stay entered in that menu/submenu
    else{ // we are not in a menu
      navigate(); // keep listening for joystick input
      showMenu(menuVal,submenuVal); // display menu/submenu on display
    }
  }
```

The two functions, showMenu and enterMenu have two different purposes. The function showMenu, as we can see, it will only happen when we are not in a menu like alarm, clock, timer etc. Its solely purpose is to check what are the values of menuVal and submenuVal and just print on the display the correct menu for these values.

6.6. Buzzer

For the alarm sounds we need to use a piezo speaker / buzzer that only has a ground and signal pin that connects to the mainboard on a digital pin. When the alarm is actioned, the buzzer will play a 1KHz tone that will only stop when the user will access the "reset" submenu of alarm/timer.

```
void soundAlarm(){
  if(millis()-lastpress>500){
    buzzing=!buzzing;
    lastpress=millis();
  }
  if(buzzing)
    noTone(buzzer); // stop buzzing
  else
    tone(buzzer,1000);
}
```

7. Testing

This process was divided into two steps, the testing of the hardware which included the physical components used, and the software which included the code. Then the testing of the software was split into two stages. The first stage is testing each module individually by running each piece of code separately. This is a good practice when designing a device that has concurrent functionalities and makes the process of debugging and troubleshooting easier because the errors would be immediately pointed out. The second stage is testing the final product to check if the four modules work fast, easy, and concurrently. We started by making sure that all the components function (including the pins of the LCD). Because the digital clock has four modules: the digital clock, the Stopwatch, the alarm, and the timer, a separate menu for each module has been implemented. Each menu of which included its own submenus that suit the roles of each module (exp: the alarm requires Enable and disable, however, the stopwatch requires a start and reset). These submenus are the settings where the module status can be changed and modified. This will make it easy for the user to access each function individually, check whether there has been something previously set (exp: alarm), and change it based on his/her need.

The first part of this section, to begin with, is the hardware. It was decided in the proposal that we will use the keypad to adjust the time and shift between the menus and their submenus. However, the LCD connection requires 16 pins 10 of which are digital. Hence why there was no space to connect the keypad which requires digital pins as well, so we had to replace it with a joystick that connects to analogue pins. The joystick values of the X-axis and the Y-axis were mapped from 0 to 1023. This made the movement between the menus happens so fast that we could not select an option accurately, so the only solution was to shrink the scale to values from –9 to 9. To test the LCD, we had to connect it to the board and run a simple program of hello world which indicated that the screen with its all pins works fine.

The first menu for testing is the clock. The clock was implemented to display time and date which required only one menu. The clock displays time in 24 hours format. So, if the time is 12:30:20 A.M, it will be displayed on the top row of the LCD as 00:30:20 by which 00 refers to hours, 30 refers to minutes, and 20 refers to seconds. As shown in figure (2), the time is shown as 14:29:29, which is the time by which that picture has been taken. For the date functionality, it is displayed in the second row of the LCD as represented in figure (2) and it indicates the day, the month, and the year (1 /4 /2022). The implementation of this module was straightforward and did not require any troubleshooting or debugging because it just gets the update of the time and date and displays it on the user interface.

The Stopwatch was the second menu to be implemented. This menu has three submenus to control it as mentioned in the previous section: the start submenu, the check submenu, and the reset submenu. The stopwatch starts counting from 0 when we click on the start submenu. It counts the time passing and it will not stop until either the reset submenu is triggered by the user, or the time passes by what the function can count, then it will adjust the time back to 0. Each of these submenus was tested separately as follows:

- The start submenu: by clicking on start submenu, the stopwatch is activated and starts counting from 0 up to whatever time passes. It starts by counting the seconds passed first, so the digit of seconds will appear on the screen. After 60 seconds have passed, the value of the seconds resets to 0 indicating that a full minute has passed, then the value of minutes will show up next to it. Similarly, after 60 minutes pass the value of min is reset to 0 indicating that a full hour has passed, and the digit of the hours will be displayed directly next to minutes. The stopwatch has been tested up to 00:04:32 and worked fine. This module can theoretically count up to 35,000 minutes (approximately 583.33 hours of continuous counting). Accessing the start submenu once activates the stopwatch. However, accessing it again while the stopwatch is already active will act as a navigator to the check submenu. Hence the user will make sure that the stopwatch is accurate, and it can be deactivated only by clicking on reset where the count will stop and reset back to 0, keeping up a space for a new stopwatch to be set.
- The check submenu: The check submenu was tested twice. First when the stopwatch was active, which would navigate the user to the time that has already passed without starting a new count just in case the user has set the stopwatch previously and forgot about it. Second, when the stopwatch is not active, the check submenu will display “Not Active”.
- The reset Submenu: by clicking on this submenu, the stopwatch readjusts the time back to 0 waiting for next activation.

The third menu is the alarm. The main function of the alarm is to alter the user at a specified time. The implementation consists of three submenus:

- The *enable* submenu: to set the time of the alteration.
- The *check* submenu: To check if there is a previously activated alarm.
- The *disable* submenu: To deactivate the alarm.

To test this module, the alarm was set through the *enable* submenu to go off at 8 P.M. This was done by accessing *enable* and setting the time manually using the joystick. If the user wants to set the alarm for 8 A.M., they set it as 08:00. If the time is 8 P.M., it should be set as 20:00 because the time format in the digital clock is in 24h. After setting the alarm we accessed the *check* submenu to make sure that everything works correctly. Because there is an alarm that is already active, the *check* submenu displayed 20:00 which is the time that we set the alarm for. At 8 P.M., the buzzer went off as expected. Another alarm was set at 8:30 P.M. (20:00) just to test if the *disable* submenu works. By accessing disable, the alarm stopped, and the *check* submenu displayed “not_set”.

The last menu is the timer. This function is quite similar to the stopwatch by which both are used to measure a specific and accurate period of time. However, the stopwatch starts counting from 0 up to whatever time passes and it counts in order of seconds. It is mostly used in races like horseraces...etc. The timer is usually set to whatever time the user wants to measure and it will count down until it reaches 0, then it will go off. It is mostly useful if the user is cooking or baking or doing any task that requires a known amount of time to be finished. The user can set the timer to a specific period of time, and it will alter them after this period is over.

This module has also three submenus which operate as follows:

- The *set* submenu: To set the timer.
- The *check* submenu: To check if there is another previous time operating. If there is any, it will display the remaining time.
- The *reset* submenu: To reset the timer to 0.

To test the timer, we had to make sure that no previous timer was active. This was done by accessing the *check* menu first where it displayed “not set”. On contrary, if there was any active timer, the *check* submenu will act as a navigator to show the time left for the buzzer to go off. After setting the time to 2:32 (two mins and 32 sec), the alarm started counting down until it reached 0:0, then the buzzer went off.

After testing each menu individually along with its submenus, the last step in this stage was to debug the whole code together and test whether the modules work concurrently. First, the alarm was set followed by the stopwatch, and after two minutes we set the timer. All functions worked fine without any lagging or time delays in their functionalities.

8. Results

This project has two testing stages. The first stage is to test each module separately to confirm the accuracy. The second stage is to test the project as a whole to confirm the achievement of the other goals like the ease of use, and the concurrency. As mentioned before, some hardware components were replaced by others like the keypad that was originally assign as an input way and was replaced by the joystick. This could have made the shift between the menus and the submenus happens faster. Moreover, adding the check menu to every module to check whether there are any previous active functions facilitated the testing process for many reasons. For example, if the stopwatch has to be tested to more than 12 hours, the check menu makes sure that we indicate exactly how much time passed which unnecessary mistakes like forgetting about the active stopwatch and setting another one in the middle of the testing process.

To confirm the results, each module was tested twice at least under different conditions. Some of which were setting the alarm, the timer, and the stopwatch to different times throughout the day, another was accessing different modules together. The clock itself only required checking up on time and date to make sure that it is up to date and the time zone can be changed and adapted from the code itself.

The other components of the clock required more testing to compare the results. Starting with the stopwatch, we had to set it many times, set the stopwatch of the smartphone, and compare the count between both to check if there is a gap or some lag in the debugging process which could be justified by the limited memory of the board (256Kb of RAM). However, the stopwatch was fairly accurate, and the time count was similar to that of the smartphone.

To test the alarm no smartphone was required, we set the alarm and had to check when the buzzer goes off and whether it notifies at the correct time. There was an issue with this module because it worked for a couple of tries However, it would not for many other. We assumed that the lag is because we accessed more than one functionality at a time. To solve this, we disabled all other modules and tested the alarm several times individually, the result was similar.

Finally, the same approach of testing the stopwatch was used to test the timer. The timer was many times for different alerts and was compared with a timer of a smartphone to check the accuracy. As described in the previous section of the report, the timer counts down until it reaches 0, then the buzzer goes off. There was no issue with implementing and testing this module and the test was made several times to confirm the functionality.

The overall results were satisfying, easy, and straightforward. The only issue was that the group members had to spend the whole time during the first test for each module to make sure that the clock does not disconnect or lag in the middle of the process. Most tests were done for a short period of time (order of minutes). Others required more time (up to more than two hours), which have not been completed because the group members were all committed to other responsibilities including their jobs as well as the submissions of another assignment. It is good to mention that some testing was done during the period of study and this project was helpful to measure how many hours of study had been spent.

9. Evaluation

Overall to evaluate how the whole project went is hard to put in just one area, as the project was over a long amount of time this meant we all (as a group) had various personal and physical issues with the project. In other words, getting the project completed at the set time frame stated at the beginning in our Gantt chart we all created and confirmed. At the start, we had four group members working on the project giving ideas and researching what we will need to o, and each do on their behalf of each other's contribution. But as time went on different group members had different contributions. This meant nearing the end of the project we only had three group members, but we have stated in the individual contribution part of the report who and what we all did to make it fair and show what understanding we all have towards the clock project we all worked on.

The project went well while considering everything else we had on throughout this project. We finished what we set out to do and completed every stage in the Gantt chart. This is a successful group project and we all worked well together, with good communication throughout the whole of the time we worked on this, and we understand each other's strengths and weaknesses. Taking this into consideration the tasks we were all set out to do matched what we were good at. This allowed us to show what we understand and making sure the work was done to the best of each of our ability.

The different stages – clock, alarm, timer and stopwatch were completed to the right timeframe with our Gantt chart. This is important as it meant nothing was delayed when going through the different stages. Keeping on time is very important with anything especially when working with other members in a project as if there is a delay with one part that could mean other testing or working on the later stages will have to be delayed too.

We all worked well together as a group letting everyone have their own input and what they want to get done with the project. It is important it being easy to work with the group members as this allows it easy to communicate with everyone and no disagreements going on when trying to work on the same work.

It is good to mention that there were some difficulties that slowed down the process. First, the testing and implementation of hardware were planned to start earlier than it had, and the code was supposed to undergo more testing. The group members were waiting for the components at the beginning which introduced more time delay. Moreover, some components were replaced to be able to commit to what we have given in the starter kit, and this required more time of research and introduced a bit of confusion at first because we were thinking about ordering another microprocessor that has more digital pins. But this decision was a bit late, and we were not sure how much time it will take to arrive. Another problem was that not all group members are familiar with C++ syntax. As a result, this has put more pressure on some group members who already have some knowledge and some other members who had to read and understand more about the language. Finally, as mentioned before, the plan was to divide the workload evenly among the four members where each decides their part and starts working on it. However, some unexpected situations happened, and we had to cover up the missing work at the last minute.

To sum up, the project was a pleasant experience to increase our knowledge. We learnt how to adapt to different conditions and how to develop our communication and teamwork skills.

10. Ethical issues

The ethical issues the clock project brought to the surface were making sure it was suitable for all cultures and was easy to alter the overall time for different time zones for all countries. Another issue was the effects of the alarm on the heart rate. Research suggests that waking up to a jolting noise (which is considered as a shock to our bodies), would cause serious heart problems. It increases the blood pressure and hence the heart rate. Moreover, this abrupt condition causes a sudden elevation in the adrenaline levels which increases the stress and may lead to anxiety in the long term. Other than this the clock is a very basic device that does not require many ethical issues that need to be thought about or overcome as it requires the user to do most of the functions it has onboard, thus making sure the user is already compliant towards what they are inputting the clock to do. Overall this means our project is ethically right.

Common ethical issues all group projects need to take in regard with is security and personal issues to make sure it does not breech any of those guidelines as it will affect how the device works if we need to alter the functionality of it due to any problems it comes up with those areas. Going through the project we had this in mind making sure it was always taking in account of the user’s security and personal area making sure it was not in breech on any of them while still maintaining the functions we set out to achieve.

11. Conclusion

In this paper, a preliminary design of an Arduino-based digital clock with an alarm, stopwatch, and timer features is presented. The goal of this project is to design a fully functioning digital clock, with the mentioned modules, that works concurrently and accurately. The process included the implementation of the hardware and the software based on the Agile methodology where the project has been broken into simpler phases. Each phase represented the implementation of each one of the functionalities separately.

The development of this project required a sound knowledge of the hardware and the programming language (C++) that has been used as well as continuous communication and planning between the team members which eventually made the troubleshooting and the testing easy and straightforward. The results indicate that the project was successful and met the expected requirements which are the accuracy, the ease of use, the low power consumption, and most importantly the concurrency. This project could be developed and used in the future either on its own as a separate wearable device, or as a starter for an advanced smart device that requires digital clock features. Overall it has been a good project to work on as it has given us all a chance to show the different set of skills we have learnt and researched over the time while working on this clock project. It has also given us a good insight on what it is like working with everyone and how to make it efficient working together while making sure the work is getting completed too.

12. Individual contribution

Adam: Abstract, Introduction, Project planning, resources, Evaluation, Ethical issues, Conclusion.

Andrei: Development and implementation, Methodology, Evaluation.

Alaa: Testing, Results, Evaluation, Ethical issues, Conclusion.

13. References

[1] Arduino.cc, “Liquid Crystal Displays (LCD) with Arduino,” Arduino, 22 04 2022. [Online]. Available: https://docs.arduino.cc/learn/electronics/lcd-displays. [Accessed 22 04 2022].

[2] A. LLC, “Liquid Crystal Library for Arduino,” Arduino LLC, 10 08 2017. [Online]. Available: https://github.com/arduino-libraries/LiquidCrystal. [Accessed 15 04 2022].

[3] A. News, “True or False: Can an Alarm Clock Affect Your Health,” ABC News Internet Ventures, 18 10 2011. [Online]. Available: https://abcnews.go.com/GMA/true-false-alarm-clock-affect-health/story?id=14730219. [Accessed 15 04 2022].

[4] MisterBotBreak, “How to Use a Real-Time Clock Module (DS3231),” Arduino LLC, 24 02 2019. [Online]. Available: https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-a-real-time-clock-module-ds3231-bc90fe. [Accessed 15 04 2022].

[5] MisterBotBreak, “How to Use a Joystick with Serial Monitor,” 18 12 2018. [Online]. Available: https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-a-joystick-with-serial-monitor-1f04f0. [Accessed 15 04 2022].

14. Appendix

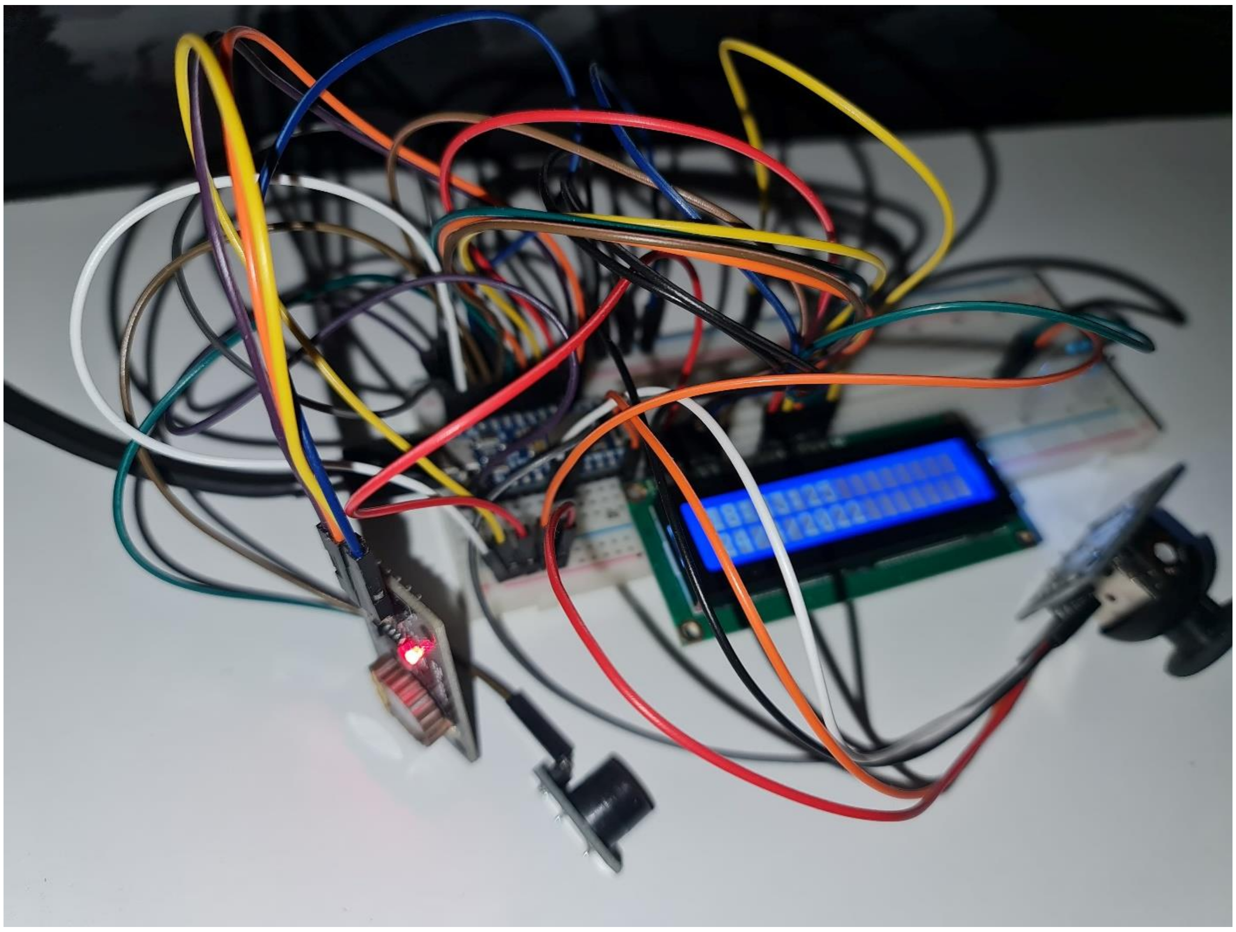


Figure 6 The final project

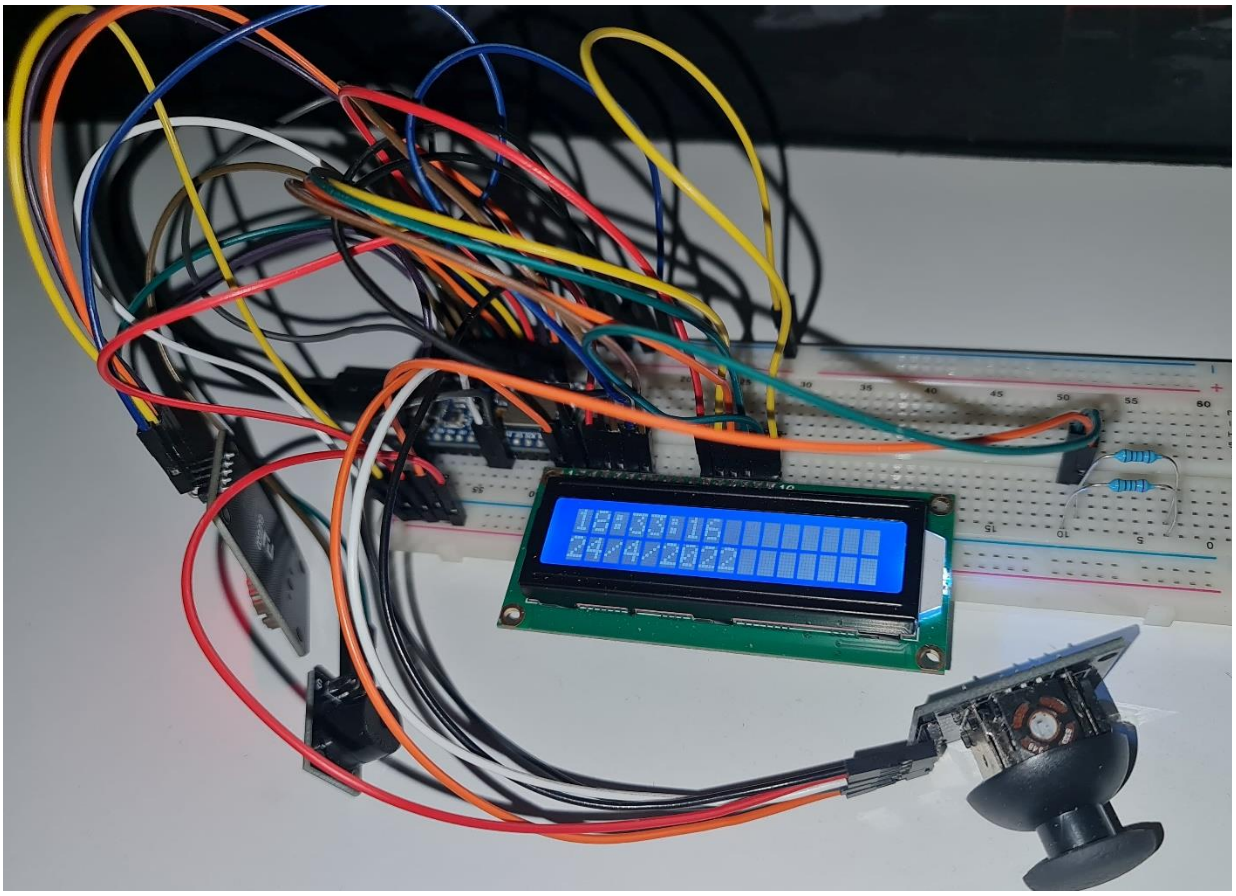


Figure 7 The final project

Below is the entire source code of the project.

```
#include "RTClib.h" // RTC library/driver
#include <LiquidCrystal.h> // LCD library/driver
#include <Adafruit_LittleFS.h>
#include <InternalFileSystem.h>
using namespace Adafruit_LittleFS_Namespace;

#define joyX A3
#define joyY A4
const int sw = 2, buzzer=13;
unsigned int lastpress = millis(), lastpos, pos, menuVal=0, submenuVal=0,
lastframe=0, swTime=0, timerM, timerS, alarmH, alarmM;
bool buzzing=0;
int valX, valY, posX, posY, menuSel=-1, submenuSel=-1, stopwatchVal=-1,
timerVal=-1, alarmVal=-1;
char* menuTitle[4][4] = {{"clock"}, {"alarm","enable","check","disable"},
{"stopwatch","start","check","reset"}, {"timer","set","check","reset"}};
RTC_DS1307 rtc;
const int rs = 12, en = 11, d4 = 5, d5 = 7, d6 = 9, d7 = 10;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // assigning pins for the LCD
File file(InternalFS);

void setup() {
  Serial.begin(115200);
  lcd.begin(16, 2); // start LCD of size 16 cols 2 rows
  InternalFS.begin();
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    while (1) delay(10);
  }
  if (! rtc.isrunning()) {
    Serial.println("RTC is NOT running, let's set the time!");
    // When time needs to be set on a new device, or after a power loss,
the
    // following line sets the RTC to the date & time this sketch was
compiled
    //rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to
set
    // January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
    rtc.adjust(DateTime(2022, 4, 1, 11, 41, 0));
  }
  lcd.setCursor(0,0); // set cursor on col 0 line 0
  lcd.write("HELLOOO"); // cursor writes this
  lcd.setCursor(0,1); // set cursor on col 0 line 1
  lcd.write("WORLD!"); // cursor writes this
  pinMode(buzzer, OUTPUT);
  noTone(buzzer); digitalWrite(buzzer,LOW);
  delay(10000);
  pinMode(sw, INPUT_PULLUP);
  pinMode(joyX, INPUT);
  pinMode(joyY, INPUT);
}
```

```
String getDateStamp(){ // collect Date value
    DateTime now = rtc.now();
    String datestr;
    datestr += now.day(); datestr += '/';
    datestr += now.month(); datestr += '/';
    datestr += now.year();
    return datestr;
}

String getTimeStamp(){ // collect Time value
    DateTime now = rtc.now();
    String timestr;
    timestr += now.hour(); timestr += ':';
    timestr += now.minute(); timestr += ':';
    timestr += now.second();
    return timestr;
}

void clearLine(bool line){
  lcd.setCursor(0,line);
  lcd.print("                ");
}

void printToLCD(String str0, String str1){ // line1, line2
  if(millis()-lastframe>250){ // remove screen flicker
    unsigned int len, len0 = str0.length(), len1 = str1.length(), index=0;
    len0>len1?len=len0:len=len1; // check which line is longer
    lcd.clear(); // clear LCD
    lcd.setCursor(0,0); lcd.print(str0); // print first line
    lcd.setCursor(0,1); lcd.print(str1); // print 2nd line
    lastframe = millis(); // update last frame time - avoid flicker
    while(len-index>15) // check if print is bigger than 16 chars
      if(millis()-lastframe>250){
        lcd.scrollDisplayLeft(); // scroll text to left
        lastframe=millis(); // update last frame time
        index++; // update position of first char on display
      }
  }
}

void showMenu(unsigned int &menuIndex, unsigned int &submenuIndex){
  menuIndex=menuIndex%4; // 4 = number of total menus
  switch(menuIndex){ // check selected menu index
    case 0: // clock
      submenuIndex=1; // display nothing - clock has no submenu
    break;
    case 1: case 2: case 3: // alarm, timer, sw
      submenuIndex=submenuIndex%4; // 4 = number of entries per menu
      if(submenuIndex<1) submenuIndex=1; // don't go below 1
      if(submenuIndex>3) submenuIndex=3; // don't go above 3
    break;
  }
  printToLCD(menuTitle[menuIndex][0],menuTitle[menuIndex][submenuIndex]);
}
```

```
void enterMenu(unsigned int menuIndex, unsigned int submenuIndex){
  switch(menuIndex){
    case 0: // clock
      switch(submenuIndex){
        default: // no submenu
          printToLCD(getTimeStamp(), getDateStamp());
        break;
      }
    break;
    case 1: // alarm
      switch(submenuIndex){
        case 1: // enable
          if(alarmVal == 0){
            DateTime now = rtc.now();
            alarmH = now.hour(); alarmM = now.minute();
            setAlarm();
          }
          submenuSel = 2;
        break;
        case 2: // check
          if(alarmVal == -1)
            printToLCD(menuTitle[menuIndex][0],"not set");
          else if(alarmVal == 0)
            printToLCD(menuTitle[menuIndex][0],"ALARM");
          else printToLCD((String)menuTitle[menuIndex][0]+" set to",
(String)(alarmH)+":"+(String)(alarmM));
        break;
        case 3: // disable
          alarmVal = 0;
          submenuSel = 2;
        break;
      }
    break;
    case 2: // stopwatch
      switch(submenuIndex){
        case 1: // start
          if(stopwatchVal==-1) stopwatchVal=millis();
          submenuSel=2;
        break;
        case 2: // check
          if(stopwatchVal==-1)
            printToLCD(menuTitle[menuIndex][0],"not started");
          else {
            swTime = (millis()-stopwatchVal)/1000; // number of seconds
            if(swTime>59)

printToLCD(menuTitle[menuIndex][0],(String)(swTime/60)+":"+(String)(swTime%
60)+" min:sec");
            else
              printToLCD(menuTitle[menuIndex][0],(String)swTime+" sec");
          }
        break;
        case 3: // reset
          stopwatchVal=-1;
          submenuSel=2;
        break;
      }
    break;
    case 3: // timer
      switch(submenuIndex){
        case 1: // set
```

```
          if(timerVal==-1) setTimer();
          submenuSel=2;
        break;
        case 2: // check
          if(timerVal==-1)
            printToLCD(menuTitle[menuIndex][0],"not set");
          else if(timerVal==0)
            printToLCD(menuTitle[menuIndex][0],"ALARM");
          else {
            timerM = (timerVal - millis())/60000;
            timerS = ((timerVal - millis())%60000)/1000;
            if(timerS == 0 && timerM == 0)
              timerVal=0;
            else
              printToLCD((String)menuTitle[menuIndex][0]+" set",
(String)timerM+":"+(String)timerS);
          }
        break;
        case 3: // reset
          timerVal=-1;
          submenuSel=2;
        break;
      }
    break;
  }
}

void setAlarm(){
  menuVal = alarmH;
  submenuVal = alarmM;
  while(digitalRead(sw)){
    navigate();
    alarmH = menuVal;
    alarmM = submenuVal;
    printToLCD("set
"+(String)menuTitle[menuSel][0],(String)(alarmH)+":"+(String)(alarmM));
  }
  menuVal = 1; submenuVal = 2;
  alarmVal = 1;
}

void setTimer(){
  timerM = 1; timerS = 30;
  menuVal = timerM; submenuVal = timerS;
  while(digitalRead(sw)){
    navigate();
    timerM = menuVal; timerS = submenuVal;
    printToLCD("set
"+(String)menuTitle[menuSel][0],(String)timerM+":"+(String)timerS);
    if(!digitalRead(sw)) break;
  }
  timerVal = millis() + timerM*60000 + timerS*1000;
}

void navigate(){
  valX = analogRead(joyX);
  valY = analogRead(joyY);
  posX = map(valX, 0, 1023, -9, 9);
  posY = map(valY, 0, 1023, -9, 9);
  if(posX == 4 && posY == -9) pos = 1; // up
```

```
else if((posX == 0 || posX == 1) && (posY == 6 || posY == 5)) pos = 2;
//right
  else if(posX == 5 && (posY == 0 || posY == 1)) pos = 3; // down
  else if(posX == -9 && posY == 4) pos = 4; // left
  else if(posX == 4 && posY == 4) pos = 0; // center
  if(millis()-lastpress > 500){
    if(pos!=lastpos){ // actioning only when current pos is different from
last pos (no spam)
      switch(pos){ // current joystick pos
        case 0: // center
        break;
        case 1: // up
          menuVal++;
        break;
        case 2: // right
          submenuVal++;
        break;
        case 3: // down
          menuVal--;
        break;
        case 4: // left
          submenuVal--;
        break;
        default: // any other - unknown
        break;
      }
      lastpos=pos; lastpress=millis(); // update last pos to current pos
    }
  }
}

void soundAlarm(){
  if(millis()-lastpress>500){
    buzzing=!buzzing;
    lastpress=millis();
  }
  if(buzzing)
    noTone(buzzer); // stop buzzing
  else
    tone(buzzer,1000);
}

void loop() {
  if(timerVal!=-1 && timerVal<millis()){ timerVal=0; menuSel=3;
submenuSel=2; }
  if(alarmVal!=-1){
    DateTime now = rtc.now();
    if(alarmH>=now.hour() && alarmM>=now.minute()){ alarmVal=0; menuSel=1;
submenuSel=2; }
  }
  if(alarmVal==0 || timerVal==0) soundAlarm();
  navigate();
  while(!digitalRead(sw) && pos==0){ // when the joystick is centered and
button actioned
      switch(menuSel){ // check in which menu we are currently
        case -1: // we are in no menu
          menuSel = menuVal; // enter displayed menu
          submenuSel = submenuVal; // and submenu
        break;
        default: // we are in a menu
          menuSel = -1; // exit current menu
          submenuSel = -1; // and submenu
        break;
    }
    lastpress=millis(); // avoid spam
  }
  while(digitalRead(sw)){ // while joystick button is not actioned
    if(menuSel!=-1 && submenuSel!=-1) // we are currently in a menu/submenu
      enterMenu(menuSel, submenuSel); // stay entered in that menu/submenu
    else{ // we are not in a menu
      navigate(); // keep listening for joystick input
      showMenu(menuVal,submenuVal); // display menu/submenu on display
    }
  }
}
```