
# *Scoreboard Controller*

*Microprocessors and Embedded Systems*

Cristescu Vlad-Andrei

# Contents

# 1.Purpose and Requirements Analysis

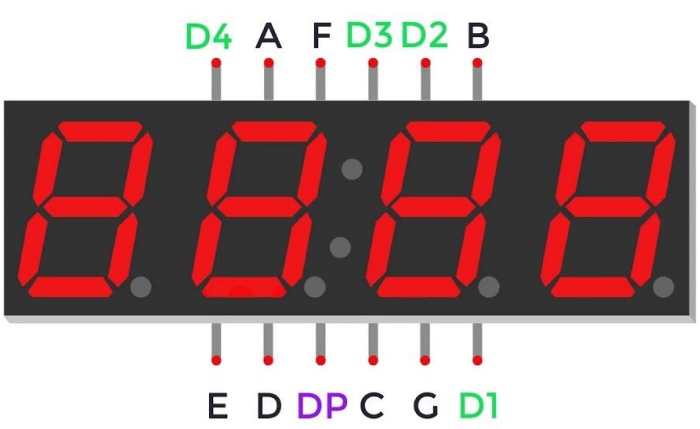
The purpose of this project is to design an embedded application that can control the score of a game and display it for the people attending the event. This can be done in many ways but I decided to interact with the user through pushbuttons and display the results on a 4 digit 7-segment display. The final device needs to be able to keep track of the score of 2 oposing teams, act as a clock and also include a timer. This makes it useful for a variety of sports, for which it can easily be modified through software. For the purpose of this project I will try to make it as universal as possible.

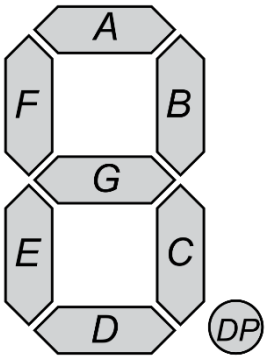
Instead of using the physical hardware I chose to use an online simulator, wokwi.com. This saved me the trouble of buying any components and wires etc. The only constraint is that I am limited to the components available in the simulator but that wasn't a problem.

# 2.Hardware

## 2.1. Display

The most important output of this application is the display. The one that I found suitable for this application is a 4 digit 7-segment display, as it can easily display the score, the current time and the value of a timer properly.




We can control each digit individually by setting the D1/D2/D3/D4 pins. DP stands for decimal point and A/B/C/D/E/F/G are the pins for each of the 7 segments of a particular digit.

## 2.2. Buzzer

Any piezoelectric buzzer can work here so I have simply chosen the one available in the simulator.

Its purpose is obviously to make a buzz sound whenever the score is reset, signalling the end of a game, or when the timer runs out, marking the end of an attack for example. I have programmed it to emit a short buzz when the timer reaches 24 seconds, which is the maximum duration of a basketball offensive play, allthough this can be effortlessly modified through code to suit another sport. The game ending buzz will be twice as long timewise but with the same frequency. This is also adjustable.

## 2.3. Pushbuttons

All functionalities of the project can be achieved with a total number of 6 pushbuttons. These act as the only user input of the system and allow the user to control what is displayed and to update the score.

We need 2 buttons for each team in order to increment and decrement their score, 1 button to be able to reset the score at the end of the game or in case of a mistake and finally one button that allows us to toggle between the scoreboard, the clock and the timer.

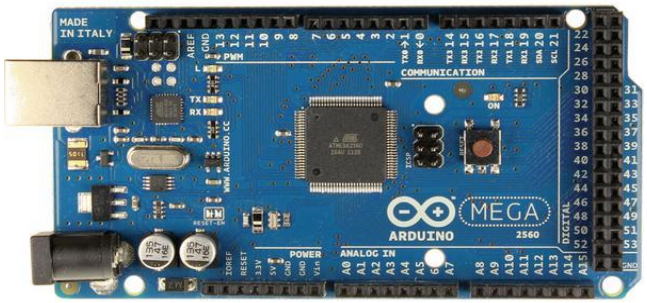
## 2.4. RTC Module

A real time clock module is necessary for the reading and displaying of the clock accurately. Its internal registers can keep track of the current year, month, day, hour, minute and even seconds.


It is very accurate because it uses an integrated temperature-compensated crystal oscillator along with a crystal. It uses I2C communication so it is enough to connect the power source, the ground, SDA and SCL to the corresponding pins on the board and the rest is done through software. For this project we only need to display the hours and minutes.

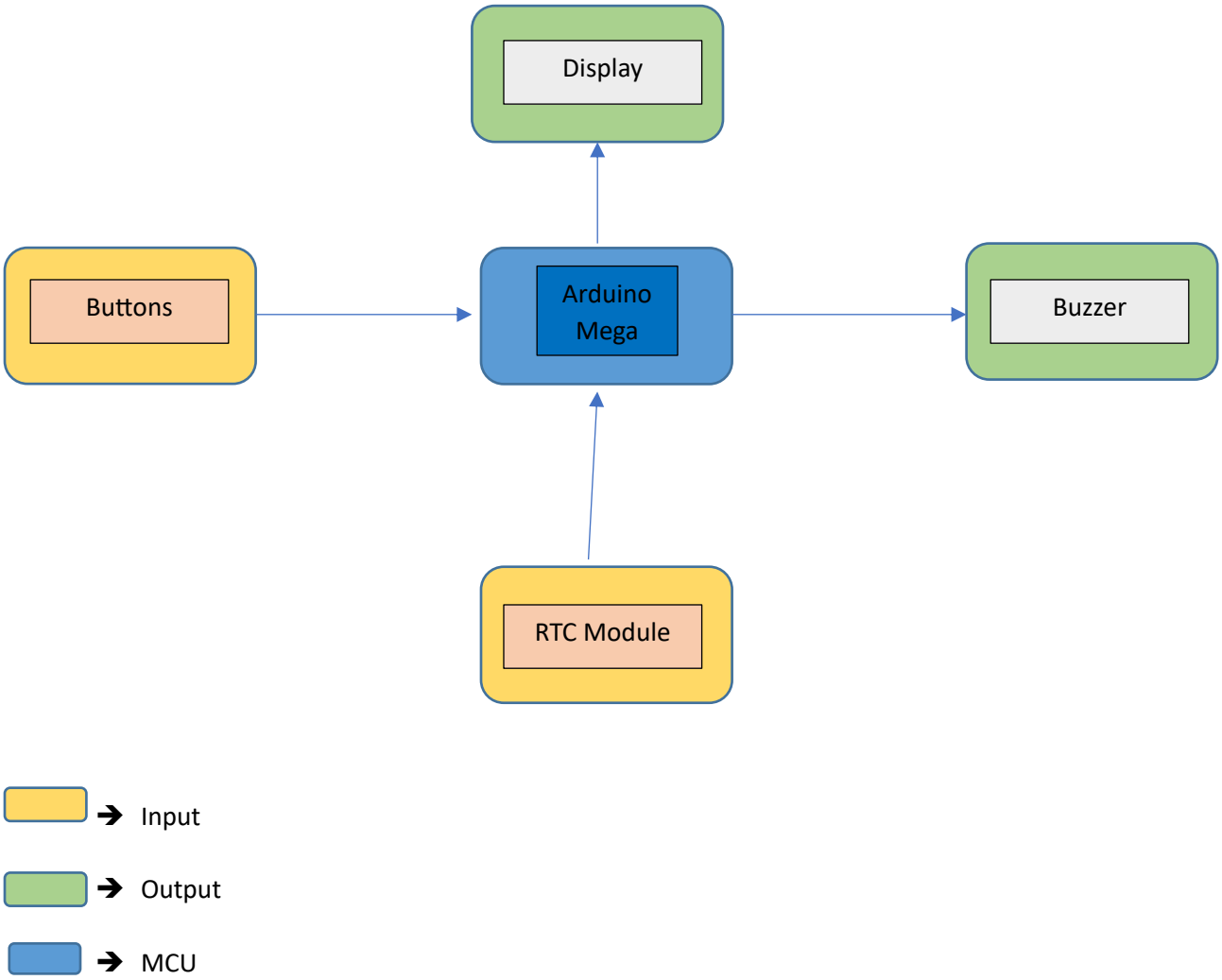
## 2.5. Arduino Mega

Finally, the “brains” of the entire system, the board that facilitates the communication between components and takes care of all the computations and processing. Many boards can be used for this application but I have chosen this one mostly because it has more pins than the Arduino Uno and is easier to use than an ESP32 or a Raspberry Pi.

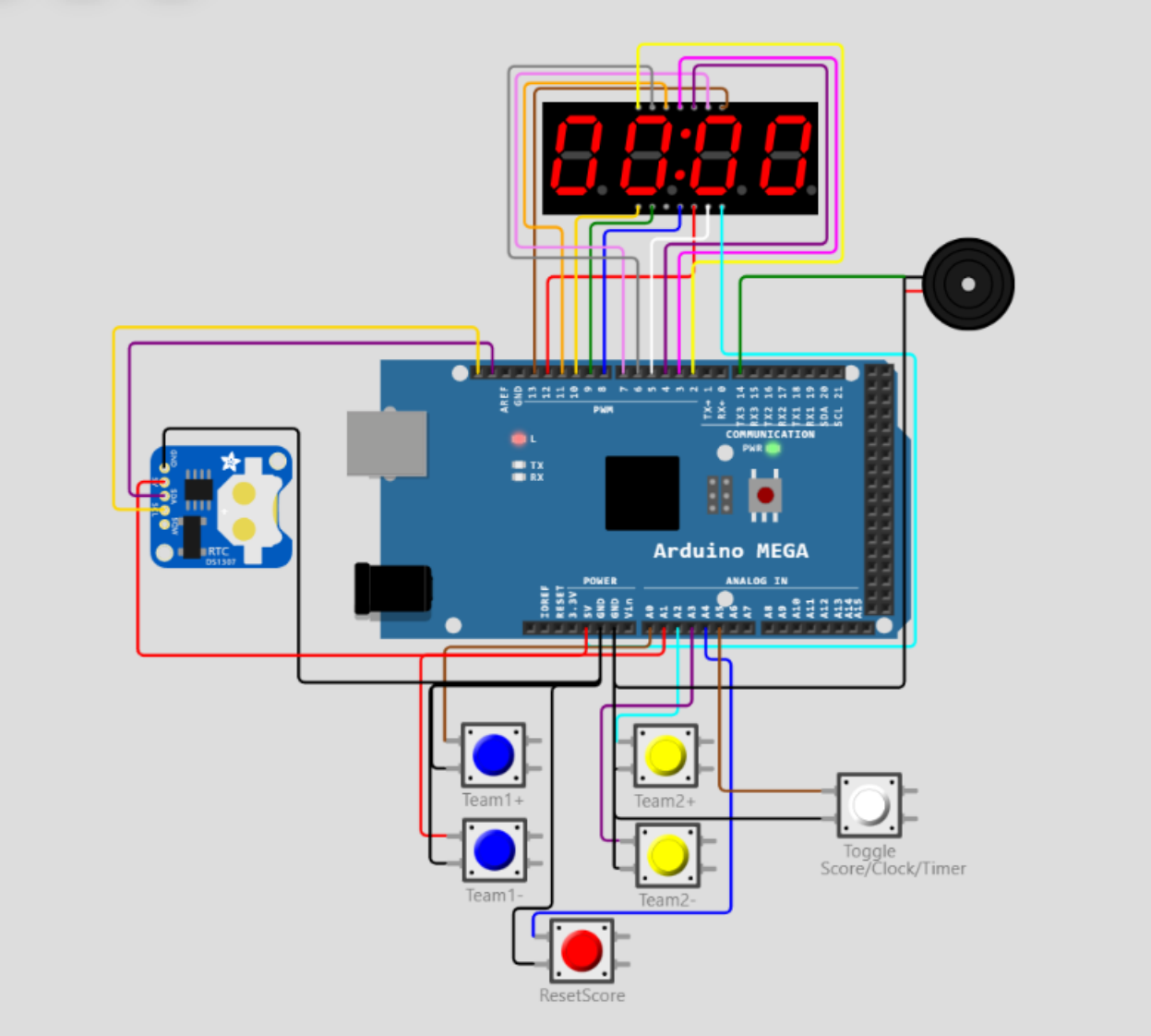


The most important part of this board is the microcontroller, which is an Atmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller. The board also comes with built-in DAC, ADC, voltage regulators and many more features and also its own development environment, Arduino IDE.

## 2.6. Block Scheme



## 2.7. Electrical connections




Given the nature of the display, it is controlled using the digital pins of the Arduino in output mode, the same as the buzzer. The buttons have 1 pin connected to ground and the other one to one of the analog input pins of the board. The RTC is connected to 5V, ground, SCL and SDA pins.

# 3.Software

## 3.1. Display

For a single digit, you'll need 8 microcontroller GPIO pins. Each pin should be connected to a single segment through a resistor, and the common pin should be connected to 5V (or GND if you are using the common cathode variant). You can spare one pin (DP) if you don't use the dot LED. Turn a segment on by driving the corresponding segment on (or HIGH for the common cathode variant).

For multiple digits, you'll need 8 microcontroller pins for the segments and the dot plus one extra microcontroller pin for each digit. So if you have 4 digits, you'll need 12 microcontroller pins in total. Controlling the display in this mode is a bit tricky, as you'll need to continuously alternate between the different digits.

Luckily, there are libraries that can help, so we will use SevSeg.h. Now, all we have to write is:

```
#include <SevSeg.h>
SevSeg sevseg;
sevseg.setNumber((number to be displayed), (number of digits));
```

## 3.2. Buzzer

For the buzzer we do not need any library as it is rather easy to set off:

```
tone((pin number), (frequency), (duration in ms);
```

```
tone(8, 262, 250); // Plays 262Hz tone for 0.250 seconds
```

## 3.3. Pushbuttons

The buttons seem very simple but are slightly more complicated in nature and will also require an additional library. When you press physical pushbutton, the circuit opens and closes tens or hundreds of times. This phenomenon is called bouncing. This happens because of the mechanical nature of pushbuttons: when the metal contacts come together, there's a brief period when the contact isn't perfect, which causes a series of rapid open/close transitions. So Button.h will help us with the debouncing and also some additional features such as input pullup resistors and more.

```
#include "Button.h"
Button Team1P(A0);    // initialize buttons
Team1P.begin();
if (Team1P.pressed()) {     // increment score
        Team1 = Team1 + 1;
  }
// just an example of how to initialize and monitor a button
```

## 3.4. RTC Module

The simulated DS1307 is automatically initialized to the current system time when starting the simulation. It then keeps counting the time.The code running in the simulation can update the date/time of the DS1307, and the DS1307 will keep track of the update time.

```
#include "RTClib.h"      // this is how to access the inner registers of the RTC
RTC_DS1307 rtc;
DateTime now = rtc.now();
sevseg.setNumber(now.hour() * 100 + now.minute(), 4);
```

## 3.5. The Code

```
#include <SevSeg.h>
#include "Button.h"
#include "RTClib.h"

Button Team1P(A0);    // initialize buttons
Button Team1M(A1);
Button Team2P(A2);
Button Team2M(A3);
Button reset(A4);
Button changeMode(A5);

const int COLON_PIN = 13;

SevSeg sevseg;        //instantiate objects
RTC_DS1307 rtc;

enum States {       // define states to be displayed
  DisplayScore,
  DisplayClock,
  DisplayTimer
};

States currentState = DisplayScore;  // initial state

void changeState(States newState){   // function for updating state
  currentState = newState;
}

void Clock() {
  DateTime now = rtc.now();
  sevseg.setNumber(now.hour() * 100 + now.minute(), 4);
```

```
35.
36.  if (changeMode.pressed()) {
37.        changeState(DisplayTimer);
38.        return;
39.  }
40.}
41.
42.void Timer(){
43.  static unsigned long timer = millis();  // create a milliseconds counter
44.  static int centiSeconds = 0;
45.
46.  if (millis() - timer >= 10) {
47.    timer += 10;
48.    centiSeconds++; // 10 milliSeconds is equal to 1 centiSecond
49.    if (centiSeconds == 2400) { // Reset to 0 after counting for 24
   seconds(for basketball)
50.      centiSeconds=0;
51.      tone(14, 300, 250);  // short beep to mark end of offence
52.      delay(1000);
53.      changeState(DisplayScore);
54.    }
55.    sevseg.setNumber(centiSeconds, 1);
56.  }
57.  if (changeMode.pressed()) {
58.        changeState(DisplayScore);
59.  }
60.}
61.
62.void Scoreboard(){
63.  static int Score ;
64.  int Team1 = 0, Team2 = 0;
65.
66.  if (Team1P.pressed()) {     // increment score
67.        Team1 = Team1 + 1;
68.  }
69.  if (Team1M.pressed()) {    // decrement score
70.      Team1 = Team1 - 1;
71.  }
72.  if (Team2P.pressed()) {
73.        Team2 = Team2 + 1;
74.  }
75.  if (Team2M.pressed()) {
76.      Team2 = Team2 - 1;
77.  }
78.  if (reset.pressed()) {
```

```
79.         Score = 0;
80.         tone(14, 200, 500); // long beep to mark end of a match
81.   }
82.   if (changeMode.pressed()) {
83.     changeState(DisplayClock);
84.   }
85.
86.   Score = Score + Team1*100 + Team2;
87.   sevseg.setNumber(Score, 4);
88.}
89.
90.void setup() {
91.   Serial.begin(115200);     // initiate serial communication and buttons
92.   digitalWrite(COLON_PIN, HIGH);
93.   Team1P.begin();
94.   Team1M.begin();
95.   Team2P.begin();
96.   Team2M.begin();
97.   reset.begin();
98.   changeMode.begin();
99.
100.        if (! rtc.begin()) {      // check for RTC availability
101.          Serial.println("Couldn't find RTC");
102.          Serial.flush();
103.          abort();
104.        }
105.
106.        byte numDigits = 4;         // display settings
107.        byte digitPins[] = {2, 3, 4, 5};
108.        byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13};
109.        bool resistorsOnSegments = false;
110.        byte hardwareConfig = COMMON_ANODE;
111.        bool updateWithDelays = false;
112.        bool leadingZeros = false;
113.        bool disableDecPoint = false;
114.
115.        sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins,
    resistorsOnSegments,
116.        updateWithDelays, leadingZeros, disableDecPoint);
117.        sevseg.setBrightness(90);
118.      }
119.
120.      void loop() {
121.        sevseg.refreshDisplay(); // Must run repeatedly
122.
```

```
123.       switch (currentState) {     // handling the display state
124.         case DisplayScore:
125.           Scoreboard();
126.           break;
127.
128.         case DisplayClock:
129.           Clock();
130.           break;
131.
132.         case DisplayTimer:
133.           Timer();
134.           break;
135.       }
136.     }
```

# 4. Link to the Project

https://wokwi.com/projects/363083213780176897

# 5. Video Demonstration of Functionality

https://komododecks.com/recordings/1MGuBeMVPaxgvvg2Hpco

# 6. References


https://docs.wokwi.com/parts/wokwi-7segment

https://wokwi.com/projects/344891439152366164

https://wokwi.com/projects/297787059514376717

https://docs.wokwi.com/parts/wokwi-pushbutton

https://docs.wokwi.com/parts/wokwi-ds1307

https://wokwi.com/projects/305979285237137984

https://docs.wokwi.com/parts/wokwi-arduino-mega

https://store.arduino.cc/products/arduino-mega-2560-rev3

https://www.circuitbasics.com/how-to-use-a-real-time-clock-module-with-the-arduino/

https://docs.wokwi.com/parts/wokwi-buzzer