LED-Pixel-Clock

Release 1.0.0

Florian Laschober

Apr 17, 2021

CONTENTS

ATTENTION: You are looking at Version 1.0 of the project. To get documentation for this release take a look at this snapshot on read the docs The source code for this release can be found at this branch on github

This is the software of a variation of the LED clock from DIY machines. I have a separate branch with a ready to use configuration which works on the original version of this clock out of the box on this branch

I decided to code it completely from scratch at I wanted to use a ESP32 instead of the Arduino nano and RTC that is used in the original project. This enables some cool features like smartphone app control, OTA updates and fetching of the time using the internet.

Additionally this has full support for animations. By default I provide all needed animations for a 12h clock to morph one digit into another soothly. Further animations can be easily added and existing animation can be adjusted to your liking.

The whole codebase is highly modular and configurable and can be tweaked exactly to your preferences.

Detailed documentation is available on Read the docs.

If you are interested in my variation of the design which uses a lot of wood instead of the 3D prints and is a bit bigger than the original you can find it on thingiverse here

CHAPTER

ONE

DEVELOPMENT ENVIRONMENT:

I am using VScode with PlatformIO. VSCode can be downloaded from here. And PlatformIO is an extension that can easily be installed from inside of VSCode.

Via the PlatformIO home the following libraries have to be installed:

- “Blynk” by Volodymyr Shymanskyy
- “FastLED” by Daniel Garcia
- “LinkedList” by Ivan Seidel

CHAPTER

TWO

IMPORTANT FILES FOR CONFIGURATION:

Files that are important for configuration (sorted by importance):

1. At the top of /include/Configuration.h -> Contains almost all important settings, like WIFI config, pin configurations etc.
2. /src/DisplayManager/DisplayConfiguration.cpp -> Configuration of which leds belong to which segments and which segments belong to which display, the order of the segments and their animation directions *It is really important to get this one right!*
3. /src/SevenSegment/SegmentTransitions.cpp -> Configuring the animations between the digits
4. /src/DisplayManager/Animations.cpp -> configuration of animations like the loading animation

2.1 Where to go from here

2.1.1 Getting started

This is the software of a variation of the LED clock from here. You can find the code ready for download here You can also find a pre configured version that should work pretty much out of the box on the original version from DIY-Machines in this release. Thanks to DIY-Machines for testing my code on the one and only original.

I decided to code it completely from scratch as I wanted to use an ESP32 instead of the Arduino Nano and RTC that is used in the original project. This enables some cool features like smartphone app control, OTA updates and fetching of the time using the internet.

Additionally this has full support for animations. By default I provide all needed animations for a 12h clock to morph one digit into another smoothly. Further animations can be easily added and existing animation can be adjusted to your liking.

The whole codebase is highly modular and configurable and can be tweaked exactly to your liking.

1. *Wiring*
2. *Install the development environment and all needed libraries*

Starting points for Version 2.0:

Take a look at the latest version of this documentation

Starting points for Version 1.0 of this project:

1. *Configure the code to fit your needs*
2. *Understanding, modifying and crating Animations*

2.1.2 Wiring

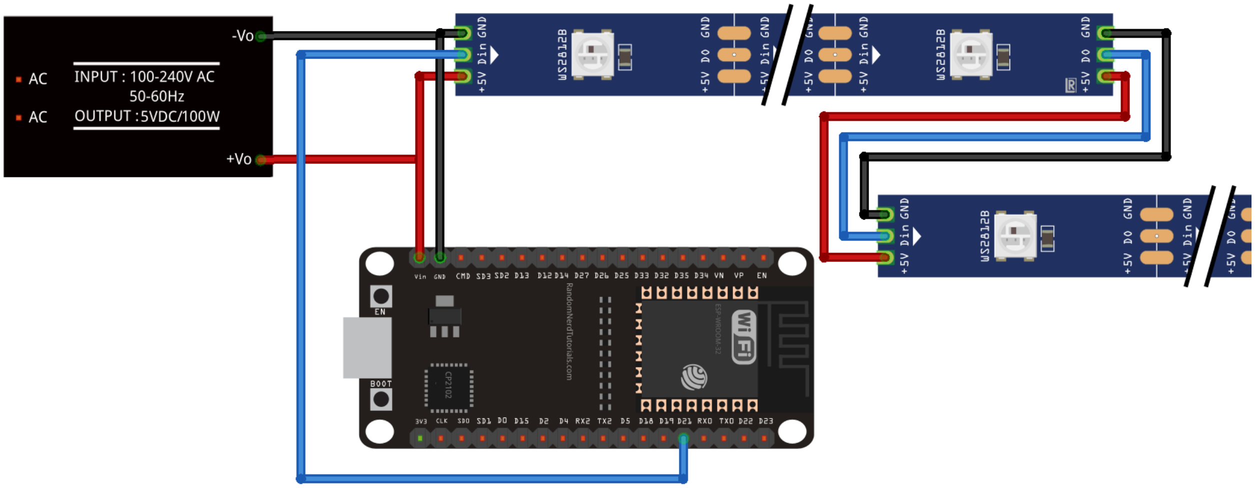
Wiring of the ESP32 is highly customizable. This can be changed and tweaked very easily by modifying the Configuration.h file.

The following table are the default values that the code comes pre configured with:

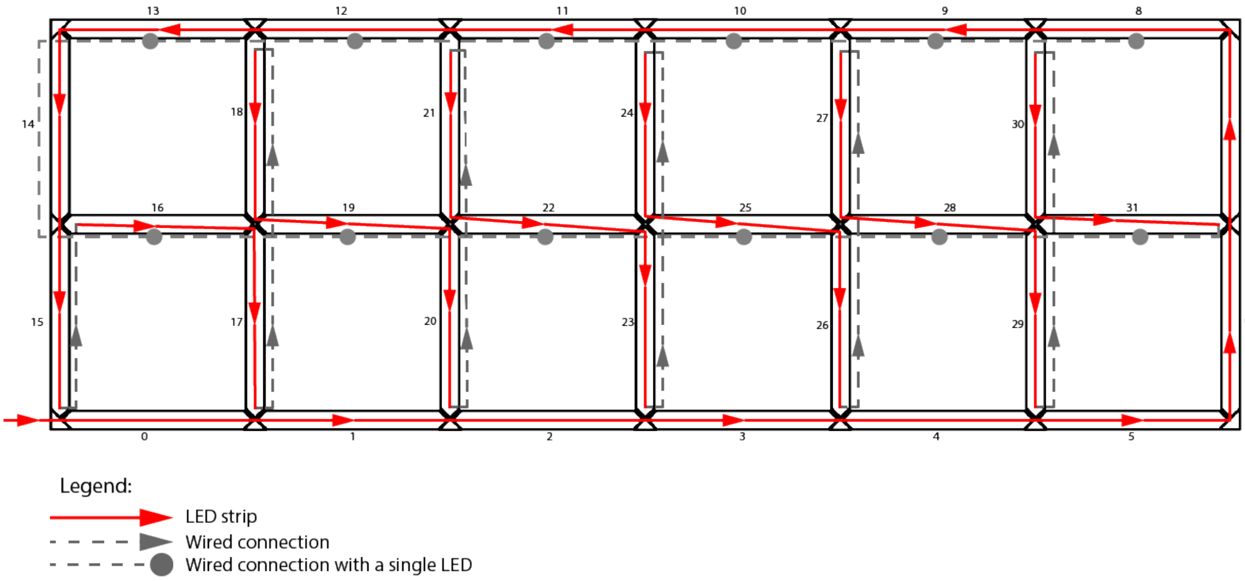
Connection	Pin	Location in config	Comment
LED strip data	21	LED_DATA_PIN	(MANDATORY) This is the pin to which the LED strip is connected to.
Internal LED strip data	22	DOWNLIGHT_LED_DATA_PIN	(OPTIONAL) This is the pin to which the LED strip of the internal dowlighter LED's is connected to in case they are separated. This only takes effect if APPEND_DOWN_LIGHTERS Is set to `false`.
Light sensor	34	LIGHT_SENSOR_PIN	(OPTIONAL) This is the pin to which the light sensor is connected in case it is enabled by setting ENABLE_LIGHT_SENSOR to `true`.

Default wiring:

This is the minimal wiring diagram according to the default configuration:


-Vo
AC
AC
INPUT : 100-240V AC
50-60Hz
OUTPUT :5VDC/100W
+Vo
+5V Din GND
WS2812B
+5V DO GND
EN
BOOT
Vin GND CMD SD3 SD2 D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 CLK SDO SD1 D0 D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
WiFi

The WS2812B LED Strips should be wired together by connecting the pads like shown on the diagram above. The Connections of the LED strips in the default config is done like this:



Power Supply specs

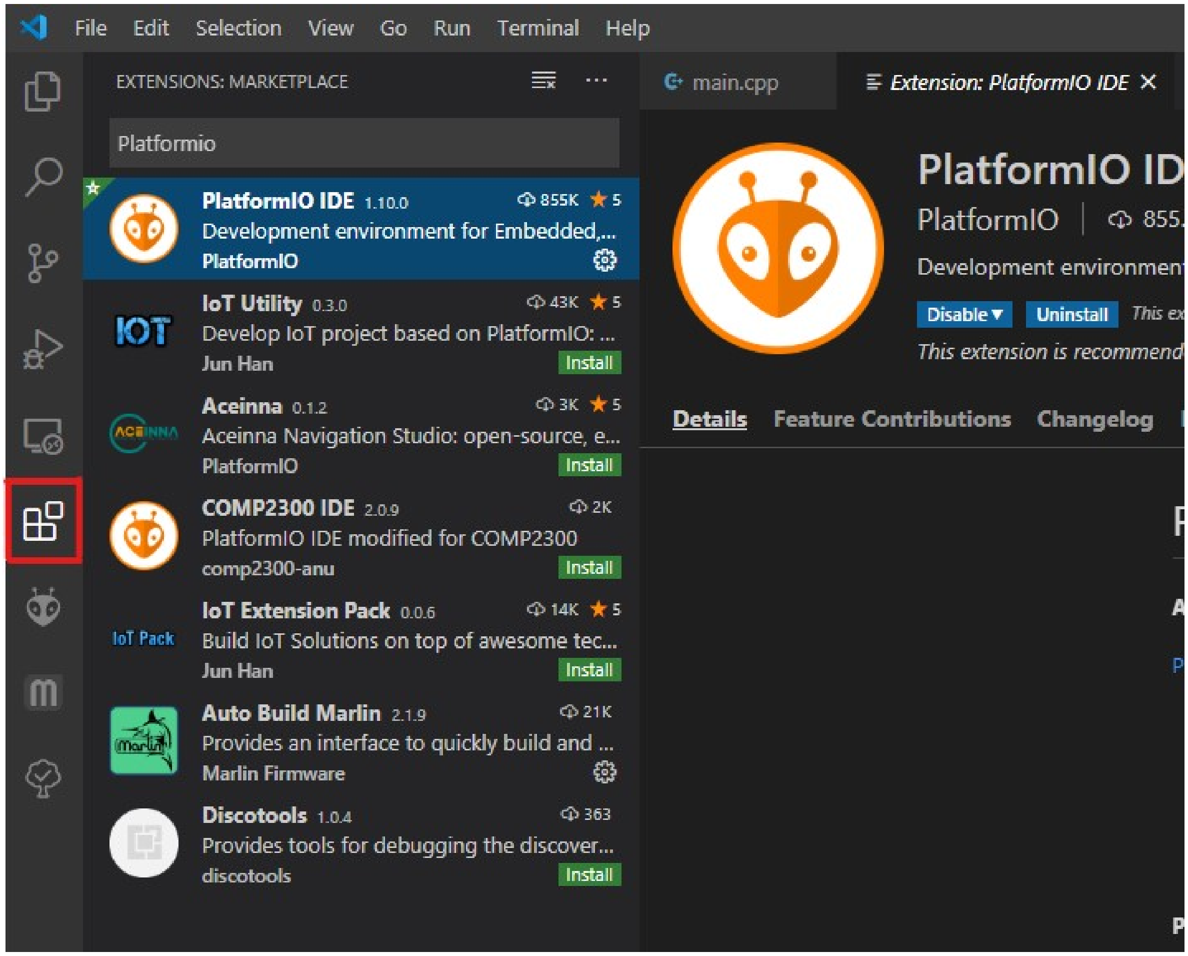
The +5V and GND connections of the LED strip should be connected straight to a suitable 5V power supply. The ESP32 can also be connected straight to the power supply, just make sure to connect the +5V to the VIN pin and NOT the 3V pin!! The required wattage/Max current rating needed can be easily calculated: According to the manufacturer of the WS2813B LED's each LED consumes a maximum of 0.24W per piece. (figure taken form here). This means for the default configuration (32 Segments with 12 LED's each and an additional 12 downlighter LED's The power supply should at least be able to handle (32 * 12 + 12)*0.24 = 95W which is around 20A at 5V to have a little bit of headroom.

2.1.3 The Development environment

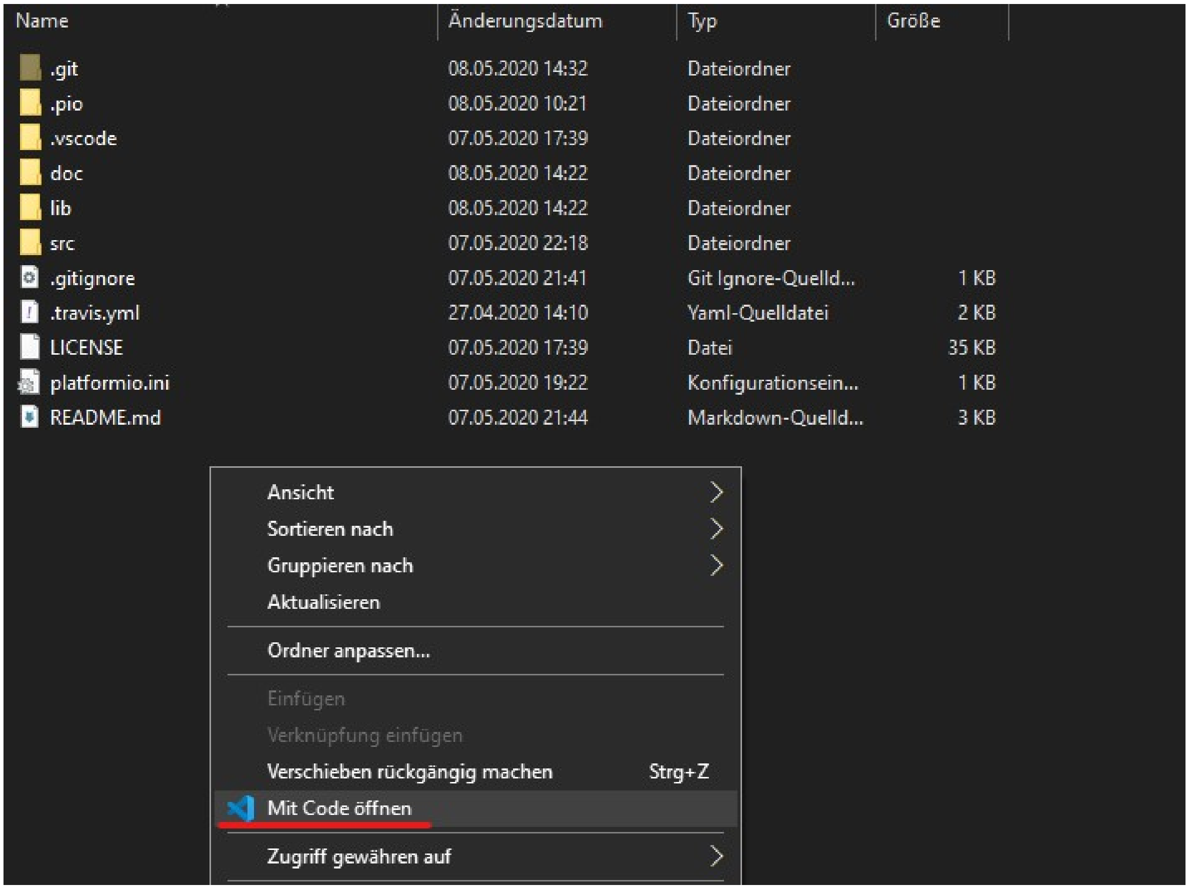
I find the default Arduino IDE very hard to use if programming more than just basic functionality. This is why I decided to code all of this using Visual Studio code with PlatformIO. It is free, easy to install, really powerful and almost as easy to use as the Arduino IDE.

Step by step installation guide

1. Install VSCode from the official microsoft page here. To make things a bit easier you can enable the right click menut entry option during the VSCode installation.
2. Start VSCode and install PlatformIO from the extension menu in the sidebar.

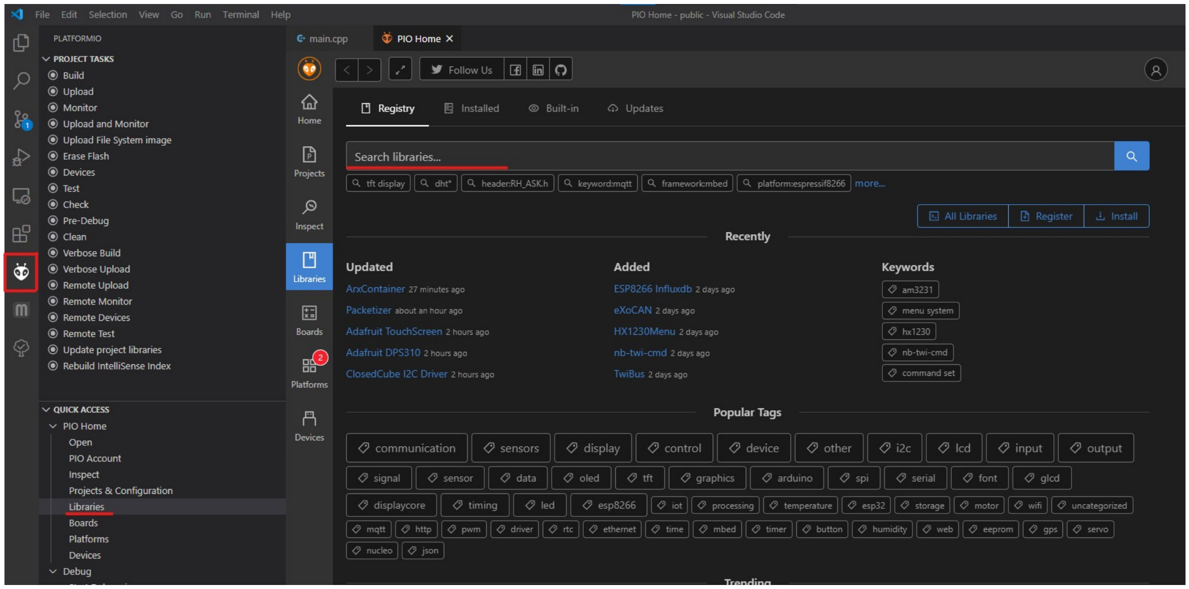

File
Edit
Selection
View
Go
Run
Terminal
Help
EXTENSIONS: MARKETPLACE
Platformio
PlatformIO IDE 1.10.0 855K ★ 5
Development environment for Embedded,...
PlatformIO
IoT Utility 0.3.0 43K ★ 5
Develop IoT project based on PlatformIO: ...
Jun Han
Install
Aceinna 0.1.2 3K ★ 5
Aceinna Navigation Studio: open-source, e...
PlatformIO
Install
COMP2300 IDE 2.0.9 2K
PlatformIO IDE modified for COMP2300
comp2300-anu
Install
IoT Extension Pack 0.0.6 14K ★ 5
Build IoT Solutions on top of awesome tec...
Jun Han
Install
Auto Build Marlin 2.1.9 21K
Provides an interface to quickly build and ...
Marlin Firmware
Discotools 1.0.4 363
Provides tools for debugging the discover...
discotools
Install
main.cpp
Extension: PlatformIO IDE
PlatformIO ID
PlatformIO 855
Development environmen
Disable ▾
Uninstall
This ex
This extension is recommend
Details
Feature Contributions
Changelog

3. Now download or clone the code from github, and put it to your hard drive. (Unzip it if you downloaded it)
4. navigate to the folder where you put your code and right click to open the Context menu and choose open VVSCode here:



Name	Änderungsdatum	Typ	Größe
.git	08.05.2020 14:32	Dateiordner	
.pio	08.05.2020 10:21	Dateiordner	
.vscode	07.05.2020 17:39	Dateiordner	
doc	08.05.2020 14:22	Dateiordner	
lib	08.05.2020 14:22	Dateiordner	
src	07.05.2020 22:18	Dateiordner	
.gitignore	07.05.2020 21:41	Git Ignore-Quelld...	1 KB
.travis.yml	27.04.2020 14:10	Yaml-Quelldatei	2 KB
LICENSE	07.05.2020 17:39	Datei	35 KB
platformio.ini	07.05.2020 19:22	Konfigurationsein...	1 KB
README.md	07.05.2020 21:44	Markdown-Quelld...	3 KB

5. After waiting for all the plugins to load you should see a PlatformIO button in your sidebar. Click on it and choose Libraries:



6. From here install the following libraries:

- "Blynk" by Volodymyr Shymanskyy
- "FastLED" by Daniel Garcia

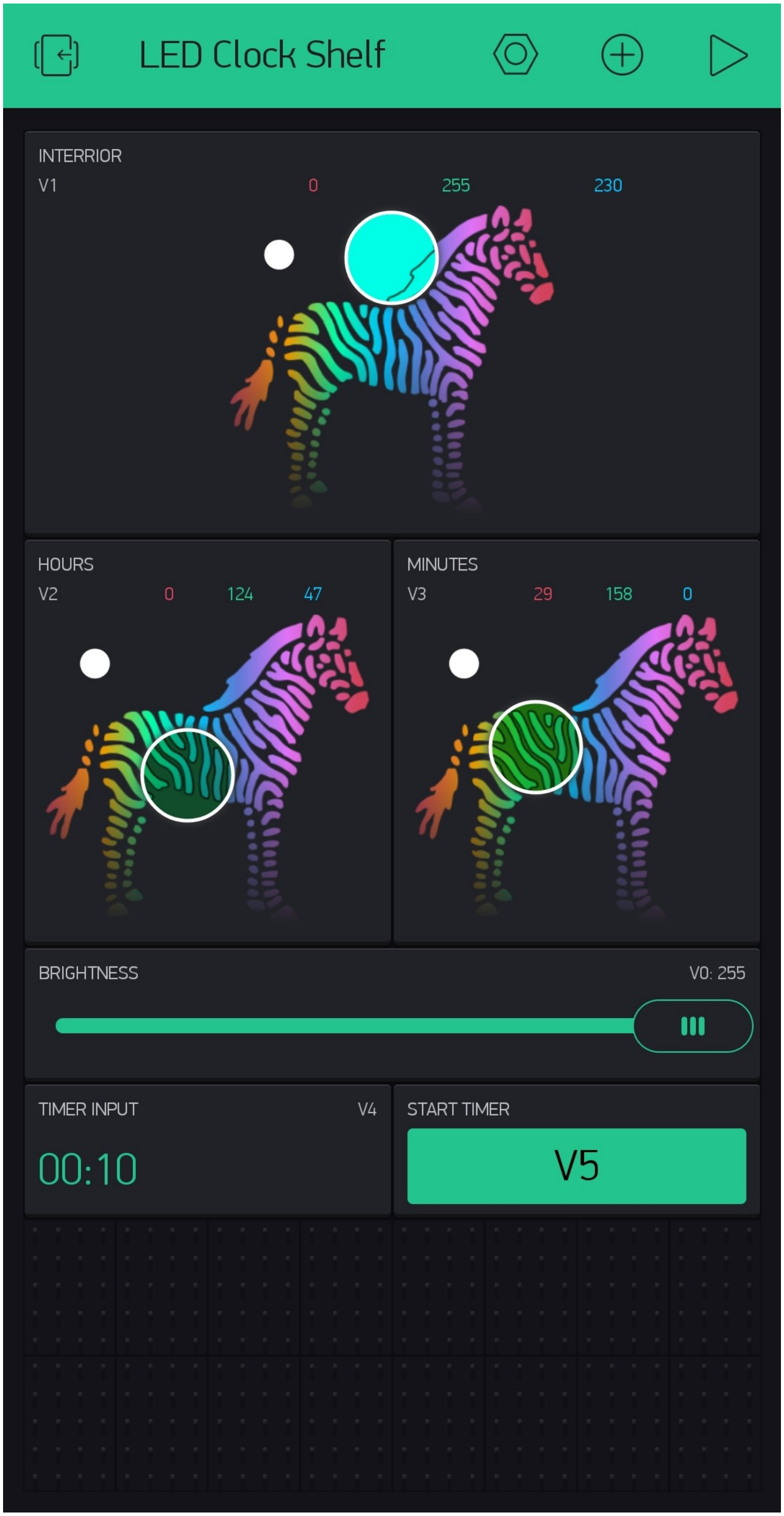
   - “LinkedList” by Ivan Seidel

7. Now you can test if the build is working by hitting the little checkmark icon in the bottom bar of VScode. Uploading to the target is done by clicking the arrow button Just like in the Arduino IDE. You don’t need to worry about your COM port as PlatformIO is smart enough to figure it out automatically.

2.1.4 Blynk configuration

The blynk configuration I am using is pretty basic but it still provides a way to select the color and brightness for the hours and minutes displays and also the interior lights.

My setup looks like this and if you didn’t change the code you will have to make sure that you are also outputting to the same virtual pins:



LED Clock Shelf

INTERRIOR
V1 0 255 230

HOURS
V2 0 124 47

MINUTES
V3 29 158 0

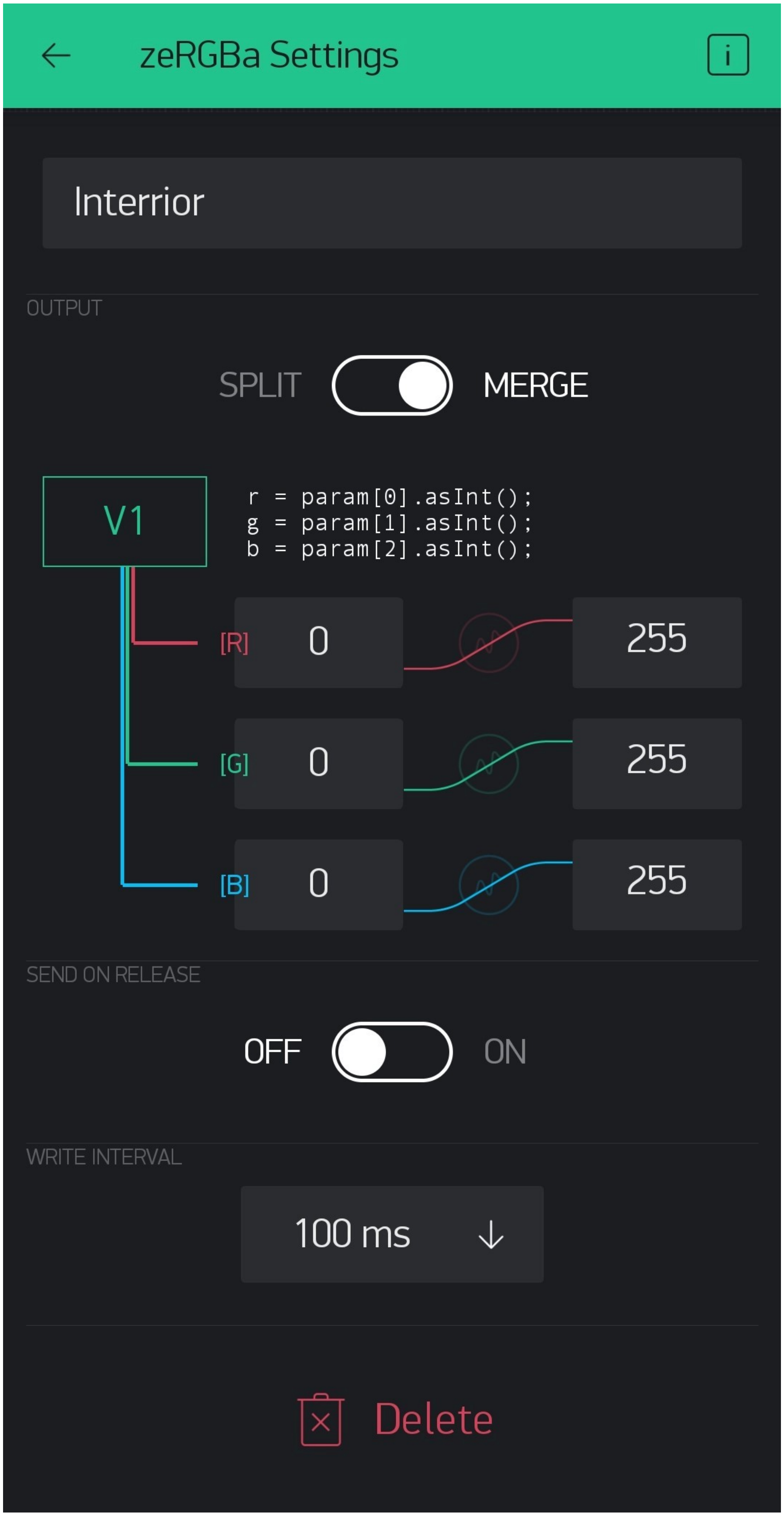
BRIGHTNESS V0: 255

TIMER INPUT V4
00:10

START TIMER
V5

If you are not interested in the Timer Functionality just simply remove the buttons for it. If you want to have an easy and quick setup exactly like me you can scan this QR code from inside the Blynk app to get up and running quickly:

If yu choose to set it up manually be aware that I have set my controls to merge the colors, so you will have to do the same. An additional nice feature to have is to update the clock every 100ms while you are changing the color to see it changing in real time on the shelf itself:



zeRGBa Settings

Interrior

OUTPUT

SPLIT MERGE

V1

```
r = param[0].asInt();
g = param[1].asInt();
b = param[2].asInt();
```

[R] 0 255

[G] 0 255

[B] 0 255

SEND ON RELEASE

OFF ON

WRITE INTERVAL

100 ms

2.1.5 Configuring the Code

I designed the code for maximal flexibility. So it is split into multiple parts which I will call units that have one very distinct purpose to them.

Short overview of the units

- The main unit: brings together all other units and combines them to make the clock work. Also responsible for Blynk and WIFI init and handling all cyclic tasks. The in app blynk config can be found *here*
- SevenSegment: This is the unit that is responsible for managing one seven segment display. It contains code for displaying numbers on the leds.
- DisplayManager: Contains code to bring together multiple SevenSegment units in order to manage all of them at once through one object. This also contains the initialization functions for the segments.
- Animator: Responsible for handling animation stages and playing, pausing resetting animations that can be added to any class that inherits the AnimatableObject class.
- TimeManager: This is a work in progress, but once done it shall be responsible to keep track of the time even if the internet goes down for a few hours

Step by Step configuration guide:

In order to get the SW up and running for your particular use case you will have to configure a few things. I will go over them in an order that makes sense to keep it as clear as possible.

Step 1

Configure the general feature set that you would like. For that open the `/include/Configuration.h` file and in it you will find a bunch of defines complete with an explanation of what they do. You can choose whether you would like to use Blynk to control your lights from the smartphone, enable or disable OTA updates to make the update process a little easier after the first flash.

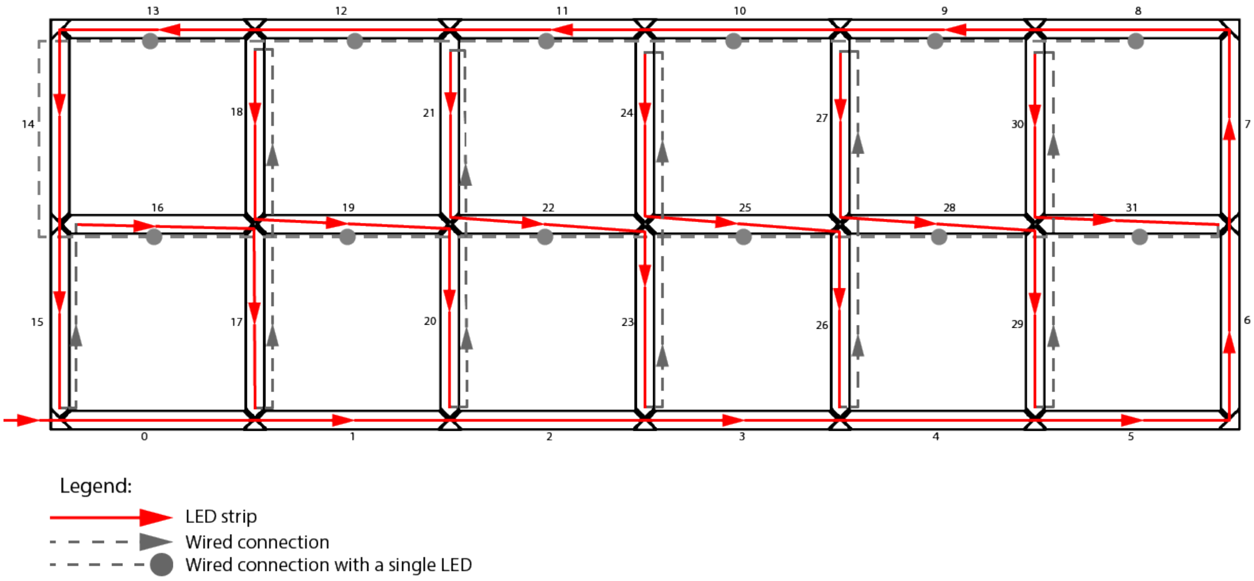
IMPORTANT: The OTA update feature does not provide a fool proof method of updating the controller. If you upload a wrong configuration and the micro controller crashes you will need to resort to flashing the controller using a cable.

Step 2

Also in the same file you have to specify how many leds you use per segment, how many segments there are in total and so on. Everything is explained there in code.

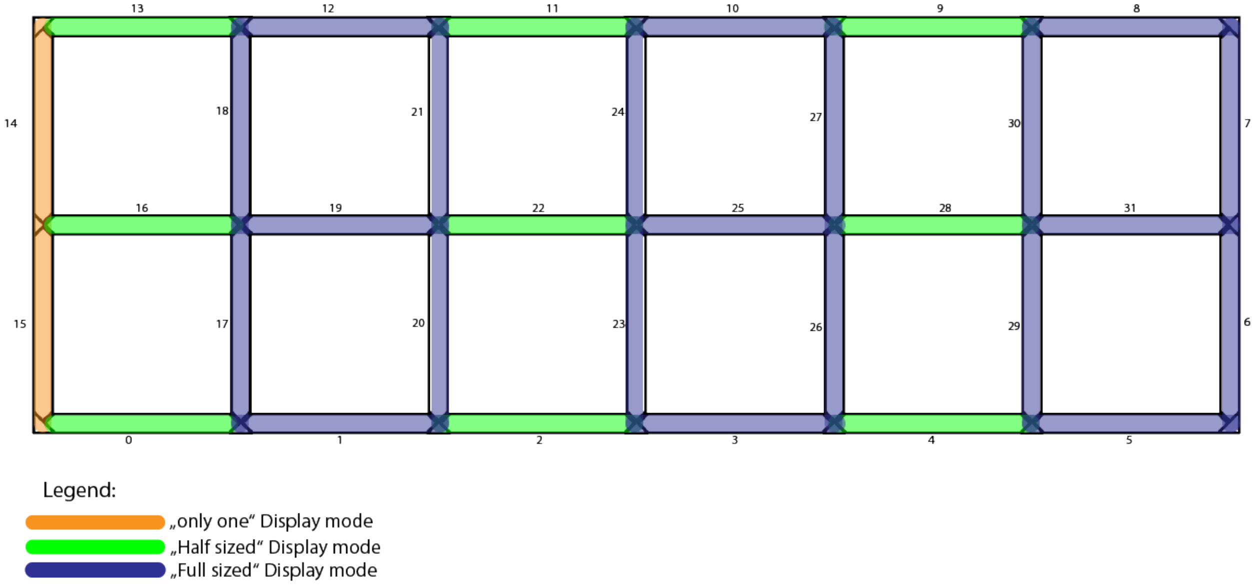
Step 3

This is the most critical and hardest to understand. You will need to configure the configuration arrays in the file `/src/DisplayManager/DisplayConfiguration.cpp` These arrays tell the system how you wired up your LEDs. Open up the file that is provided with the code and take a look at this schema:



Here you can see how I wired up my LEDs. The order of the segments is important and always has to be the same as they are wired.

First it makes sense to look at the arrays called `SegmentPositions` and `SegmentDisplayModes`. You will have to think about to which seven segment display each single segment belongs. Imagine this:



As you can see, in my example there are 3 full sized displays, 3 half sized displays (the ones with just the horizontal segments) and 1 segment which can only display a 1. If you take into account the way I wired the LEDs on the graphic above you can see that the first segment in my strip is the middle bottom segment of the second display from the left which is a "Half display" This is why the first entry in the `SegmentPositions` array is `SevenSegment::MiddleBottomSegment`. Just go through them one by one, think about which display they belong to and what their position is within their display.

The `SegmentDisplayModes` array is easy, just put in the display types you have from left to right. You can see the three modes you can choose from in the provided file.

Next up is the `SegmentDirections` array. This one is important for animations. You have to tell the program in which direction you wired the LEDs. Just take a look at the first diagram and follow the arrow directions. They will tell you what to put into the array.

The final array to configure is the `displayIndex` array. It is used to know which segment belongs to which display. So each position in the array corresponds to the same position in the `SegmentPositions` array. The number you have to put there corresponds to the index number of the entries in the `SegmentDisplayModes` array. So for example, the first segment in my code is the middle bottom segment of the second display which is a half mode display. So you have to put a 1 there to point the code to the second position of the `SegmentDisplayModes` array.

Step 4 (optional)

Now you should be at a point where your system should at least compile and display something. Everything onward from here are additional steps and are not needed if you are not concerned with altering the looks of Digits or animations.

If you want to change how a certain number is displayed you will have to go to the file `/src/SevenSegment/SevenSegment.cpp`. On the top here you will find an array which is used to configure which segments to turn on for displaying a certain number. For example, some people prefer to have the bottom segment turn off when displaying a "9" so if you wanted to do that (My code turns the bottom segment on for 9 by default) you just simply remove the "MiddleBottomSegment" value from the array at it's 9th position.

Be aware that if you alter the looks of the digits you will have to also alter the animation that is used to morph the digits. You can find how to configure animations *on this page*

if you just want to change the speed of how long it takes to morph one digit into another you can do that in the file `/src/DisplayManager/Animations.cpp` by changing the define there.

2.1.6 Tweaking and adding animations

All animations are highly adjustable. First let's talk about the animations that come reconfigured in the code.

The most important animations in the system are the ones that are responsible for morphing one digit into another to provide a smooth transition. These animations are defined in the file `/src/SevenSegment/SegmentTransitions.cpp` But there is one additional animation that I provide with my code, and that is the startup loading animation. This animation can be found in the file `/src/DisplayManager/Animations.cpp` but it basically works exactly the same as the others.

Changing animations

An animation consists of the following essentials:

- A global struct variable that holds all the animation information.
- An initialization function that is responsible for populating the values to the above mentioned variable.

Each animation has a number of steps, it must at least be one and it can be as many as you desire. With each step multiple animations can be started, and as soon as they are finished all animations of the next step will be started. Each step has two arrays. One array defines all the indices for which the animations shall be started. These indices are used to address the objects inside the AnimationObject array that has to be passed to the start animation function. the second array defines the actual animation effect for each of the objects. The size of the arrays for each animation step has to be the same for all steps within one animation! And the `animationComplexity` attribute is set to the same value! Otherwise the system WILL CRASH! If you need let's say 5 animations at once in the first step but only one in the second one you have to fill all unused array positions with `-1` for the indices. For the animations effects it does not matter but I fill them with 0 to make it clearer that there is nothing happening there.

Adding new animations

If you want to create new transitions between two digits that currently do not have a animation just add a `ComplexAnimation` variable and a corresponding init function prototype to the `SevenSegmentTransitions.cpp` file. Simply following the example of the already defined animations. You should also add a definition to the new local variable to the `SevenSegmentTransitions.h` file and declare it as extern. To register your new Animation to be called when the digits change simply add it to the TransformationLookupTable array at the right spot. There are comments there making it clear where you have to put the new animation.