ESP32_MasterClock

Introduction

The project deals with the construction of a simple source of polarized pulses for sub-clock networks. The precise time is obtained from the internet using the NTP protocol. The heart of the device is the ESP32-C3 SuperMini module, which connects to the internet via WiFi. A prerequisite for the proper functioning of the device is, of course, an available WiFi signal at the deployment location. A possible failure of Internet connectivity only affects the accuracy of the operation, and after the connection is restored, the synchronization with the NTP server (and thus also the accuracy) is automatically restored.
The transition to DST and back is done automatically (thanks to synchronization with the NTP server).

The distribution of pulses is managed by a module equipped with two A4950 H-bridges. The design of the module was chosen primarily to ensure easy repair in case of failure. The advantage of the dual configuration is that in the event of a failure of one bridge, it should be sufficient to rotate the module and thus use the second, previously unused bridge.

The device does not contain any adjustment elements or a display. All settings are done through a web interface. The status of the device is indicated using two LEDs. The first LED is used to indicate the status of WiFi (via a steady light or blinking). The second LED is a multi-colored (WS2812) and indicates the status of the slave clock using different colors.

Building With PlatformIO

The build environment is based on VS Code/PlatformIO. The platformio.ini file contains two software variants (data a webh). The data variant is intended primarily for the development phase (but nothing prevents it from being used as a production one), while the webh variant is intended exclusively for production.

The following way of development follows from this:

1) Development takes place exclusively in the software data variant (in the platformio.ini file the line `default_envs = webh` is commented and the line `default_envs = data` is uncommented). After editing the source files, the compilation is done in the usual way and the device is upgraded via the serial port.

2) If you also need to edit the *.html or *.js files of the web server, you need to open the http://<device IP address>/edit page. For this to work, the PC must be connected to the Internet. This is because the online library "ace" (**A**jax.org **C**loud9 **E**ditor) is used to edit web pages. See https://ace.c9.io/ for more.

3) Once the required functionality is achieved using steps 1 and 2, the webh variant software can be created. This is done by running the `Download FS & Create WEBH` custom script, accessible through the `PROJECT TASKS/data/Custom` menu. This will transform the web server files (from the LittleFS file system) into webh files. Maxgerhardt's "pio-esp32-esp8266-filesystem-downloader" utility is used to download the filesystem image and extract it.

4) Switch to the webh variant (in the platformio.ini file, comment out the line `default_envs = data` and uncomment the line `default_envs = webh`) and proceed with the compilation. The resulting image „*firmware_webh_XXX.bin*" (created during the build process) can be found in the directory /bin/webh. The version „*XXX*" is set in the platformio.ini file before compilation.

Upgrade

The device running the webh variant of the firmware can be upgraded very easily through the web interface:

a) In the menu (on the left side of the screen), select the "Update" option.

b) A window will appear showing the current version at the bottom.

c) Use the "Select file" button to search for an update file (e.g. *firmware_webh_100.bin*) on the computer and start the process with the Update button.

If everything is fine, after restarting the device already contains the new software version.

First run

When the device is first started, it runs in AP mode by default. On the PC we connect to the "ESP32_MasterClock" network and in the browser to the address 192.168.4.1. A web page with a login dialog should appear. The default password is "admin" (can be changed on the **General Settings** web page).

Main menu

After logging in, a welcome screen will appear with basic information about the device. On the left side of the screen is the main menu. Description of main menu items:

Settings

If it is necessary to adjust the device settings (for example, at the first start), we select the Settings item. After clicking on it, the menu will expand and we will see five sub-menus:

Network Settings

On this page, you must first set the network (or multiple networks) to which the device should connect. Press Edit Networks and then New Network. In the dialog window that appears, set the required values. The Scan button can be used to search for available networks. Then you need to press Save Changes. Under the network table, it is possible to change the SSID, IP address and mask (all in AP mode). The changes must be confirmed by pressing the Save button and then also the yellow button with the text You have uncommited changes, please click here to commit and reboot …. However, it is advisable to use this yellow button only after going through all the other settings, because after using it, the device will restart.

Hardware Settings

On this page, it is possible to set the used GPIO pins, the brightness of the status RGB diode and the active level of the WiFi diode (LED, used to display the WiFi status). The set values must be confirmed again by pressing the Save button.

Note: The GPIO for connecting the status (multi-colored) LED must be known at the time of compilation and therefore cannot be changed by the user. It is hardwired to GPIO 6.

General Settings

On this page, it is possible to set the login password (default is "admin") and Host Name. It is confirmed again by pressing the Save button.

NTP (Time) Settings

On this page it is possible to set the NTP server, synchronization interval (in minutes) and time zone. The changes are confirmed again by pressing the Save button.
Sometimes it can happen that none of the known WiFi networks are available when the device is turned on (or restarted). Then, of course, it is not possible to synchronize the time with the NTP server. In such an extreme situation, the Sync Browser Time to Device button will help.

Slave Line Settings

Four parameters are selected on this page:

Line type. Pulse interval 60 or 30 seconds.

Clock cycle. For sub-clocks with an analogue 12-hour dial, but if there are also digital (flip-flop) clocks between the sub-clocks, a 24-hour cycle should normally be used.

Pulse length. It is set according to the type of slave clock.

Gap length. It is used during accelerated catch-up.

The set values must be confirmed again by pressing the Save button.

Slave Line Control

If we select the Slave Line Control item in the menu, we can control the slave line or check its status. After switching on or restarting the device, the slave line is always in the Stopped state. Two clocks are displayed on the page. The first ones show the time of the master clock (should be the correct exact time). The second clock is actually an image of the slave clock and serves to synchronize with its state. We synchronize by setting the time on the clock using the Hour+, Hour-, Min+ and Min- buttons. After execution, we can start the slave line using one of the Start, Wait or Accelerated Catch-Up buttons. Their name corresponds to the function. The Start button doesn't have much practical meaning, but it can come in handy sometimes. The Wait button puts the line in the Waiting state. As soon as the situation arises that the time of the main clock coincides with the time of the slave clock, the device automatically switches the slave line to the Running state. In this state, every full minute (or half-minute in the case of the corresponding type of clock) an impulse is sent to the slave line, which moves it by one step. The Accelerated Catch-Up button will put the line in the state of the same name. The slave clock advances at a rate that is given by the sum of the pulse length and the gap length. As soon as the agreed time with the main clock time is reached, the slave line is switched to the Running state. The meaning of the remaining two buttons is described in the Installation chapter.

Backup & Restore

In the main menu, we can also find the "Backup & Restore" item. The meaning is obvious from the name. It is possible to save a file with a backup of the settings to your PC and easily return to the same settings if necessary. In addition, another function is hidden under the mentioned item - device reboot.

Factory reset

A function of the same name is hidden under this main menu item. After its activation, the configuration file is deleted and after reboot the device is in the same state as when it was first started.

Logout

Použitím tohoto tlačítka se uživatel může odhlásit z webového rozhraní. Je doporučeno provést to vždy po dokončení nastavení či po kontrole stavu podružné linky. Dlouhodobé přihlášení do webového rozhraní zbytečně snižuje spolehlivost zařízení.

Installation

On the Slave Line Control page we can find two more buttons: Permanent voltage corresponding to an even impulse and Permanent voltage corresponding to an odd impulse. These two buttons are only relevant during system installation. It is used to set the correct polarity of the pulses on all slave clocks. Press one of the listed buttons and check all slave clocks. The corresponding (even or odd) minute (or half minute) should be set on all of them. If this is not the case, we reverse the polarity of the respective clocks. Then, to check, we press the second of the listed buttons and check that the clocks jumped one step to the correct (odd or even) minute (half minute). At the same time, we set all slave clocks to show the same time. We then cancel the permanent voltage by pressing the Stop button and synchronize using the Hour+, Hour-, Min+ and Min- buttons. We complete the installation by pressing the Accelerated Catch-Up or Wait button (at your discretion).

WiFi LED

The WiFi connection status is indicated by the "WiFi LED". We can distinguish 4 different states:

Connected to one of the set networks

The LED *lights up permanently*.

Waiting for network after losing connection

Once the connection is lost, the device waits (for an unlimited amount of time) to be restored. The device can connect to any of the set networks. The one with the strongest signal is selected. In the standby state, the diode *flashes once*. The clock should continue to function during the loss of connection. However, their accuracy is reduced - there is no synchronization with the NTP server.

The device is in user-defined AP mode

If none of the set networks are found after the reset, the device switches to AP mode. If a configuration file exists (the device has been set up at least once), the parameters set on the **Network Settings** page will be used. In this state, the diode *flashes twice*.

The device is in default AP mode

At the first startup, after a factory reset, or if a valid configuration file is not found after some problem, the device switches to AP mode using the default parameters (SSID is ESP32_MasterClock and IP address is 192.168.4.1). In this state, the diode *flashes three times*.

A summary of this is in the following table:

WiFi status	LED is on/flashing	Graphic
Connected to the set network	Lights up permanently	———
Waits for network after connection is lost	Flashes once	•
AP mode in user settings	Flashes twice	••
AP mode by default	Flashes three times	•••

Status LED

The status of the slave line is indicated by a multi-colored LED. The assignment of colors to individual states is in the following table:

Slave line status	LED color
Stopped	Red
Waiting	Purple
Accelerated Catch-Up	Blue
Running	Green
Permanent voltage corresponding to an even impulse	Yellow
Permanent voltage corresponding to an odd impulse	Pink

The same colors indicate line status on the Slave Line Control web page as well.

Behavior after power failure

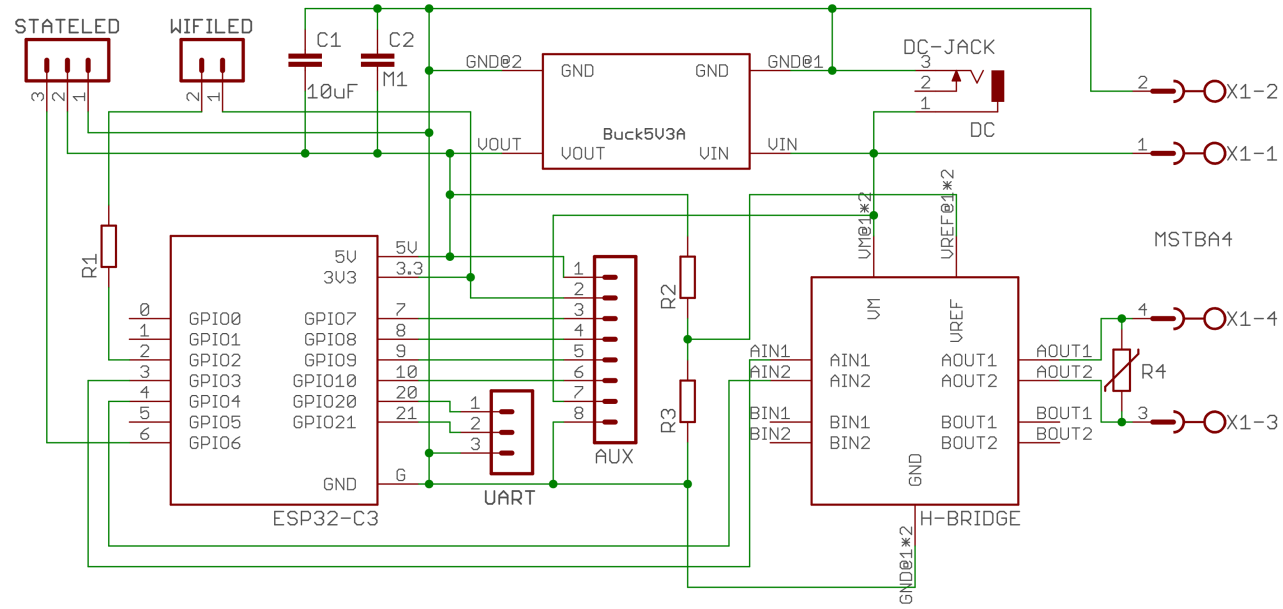
Several constructions similar to this can be found on the Internet, and of course there are also professional devices. I noticed that the authors usually deal with the software solution to the power failure problem. The principle of the solution usually consists in the fact that the time of each sent impulse is stored in the EEPROM or other similar memory. When the device starts up after power is restored, the number of missing pulses is calculated from the current time and the time of the last pulse sent. The missing pulses are then sent to the slave line in accelerated mode. The biggest problem with the mentioned solution is obvious – it is the known problem of memory wear. Another error can theoretically also occur if the supply voltage is lost at the time of writing to the memory. It follows that the reliability of the solution is not great. For that reason, I decided not to go down a similar path at all. I personally consider the use of a backed-up source to be a much better solution. It can be either a classic UPS, used in IT technology (especially suitable if there is already a backup source at the installation site) or a special source with a UPS function can be used to power the device. Examples of suitable source types are SCP-35-24 or SCP-50-24 from Mean Well. A huge advantage of this approach is that the clock runs normally even during power outages.

It is of course possible that (even though a backup source is used) a power failure will occur. In this case, after power is restored, the slave line will remain in the Stopped state and must be started "manually", i.e. via the Slave Line Control web page.

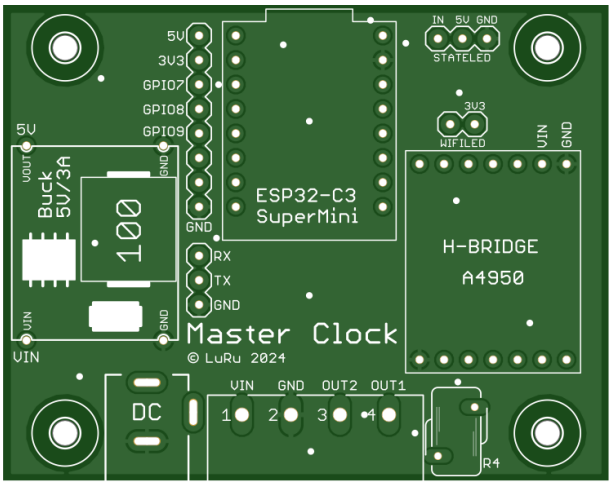
Design of the device

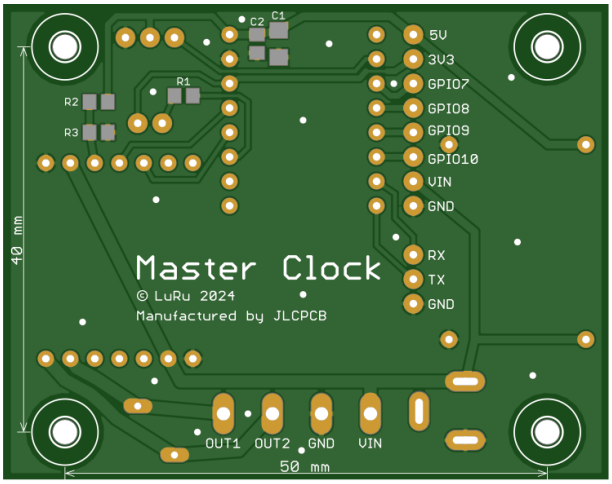
The wiring diagram and circuit board were created using the Eagle design system.

Wiring diagram



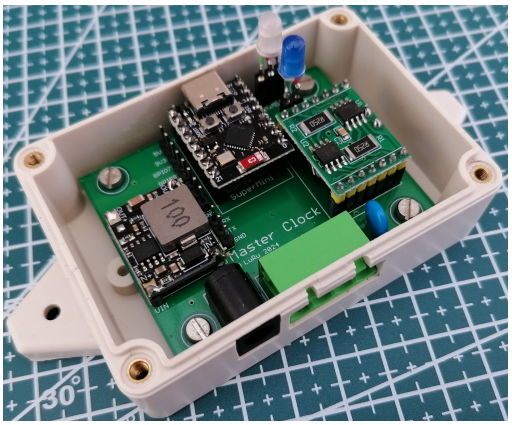
Printed circuit board





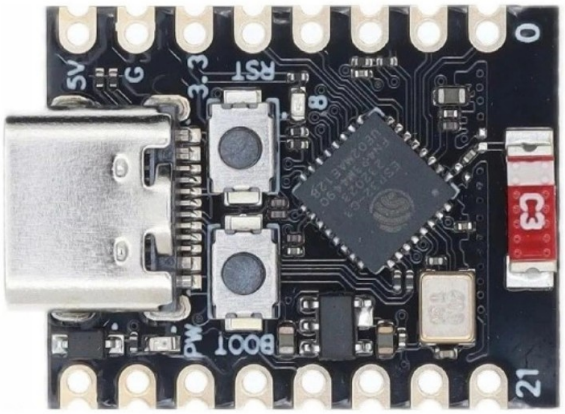
A box

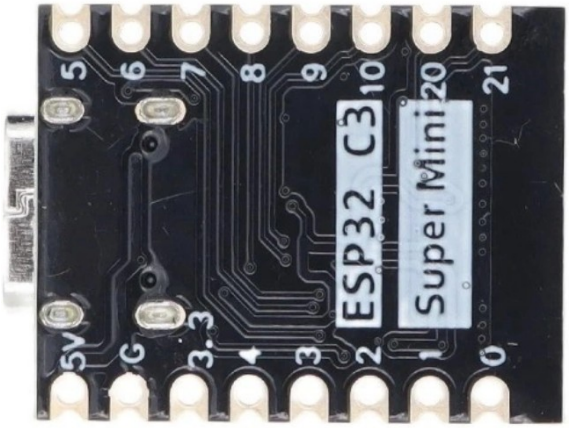
The size of the printed circuit board and the spacing of the mounting holes were adjusted after the design so that the board fits into some - existing on the market - box. A box with dimensions of 83 x 58 x 33 mm fits perfectly.



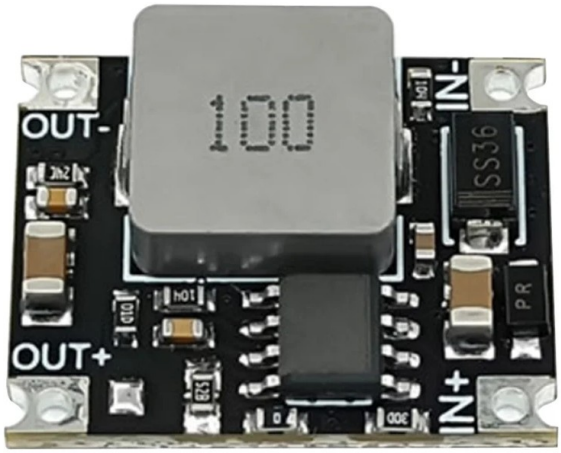
Choice of components

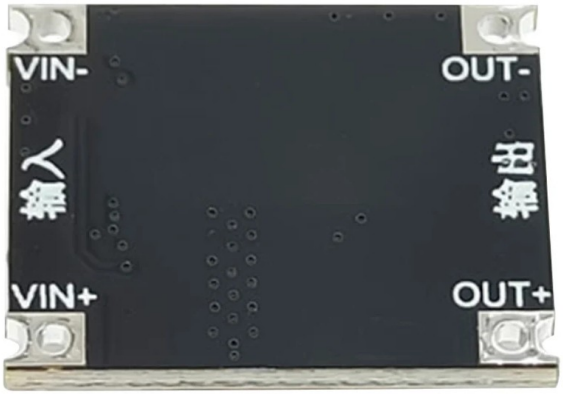
Even before the design started, I was clear that one of the ESP32 chips would be used in the device, and it would be one of the many existing modules to ensure easy replacement in case of failure. In the end, I chose one of the smallest modules – **ESP32-C3 SuperMini**. It is small, cheap and more than adequately equipped for the given use.





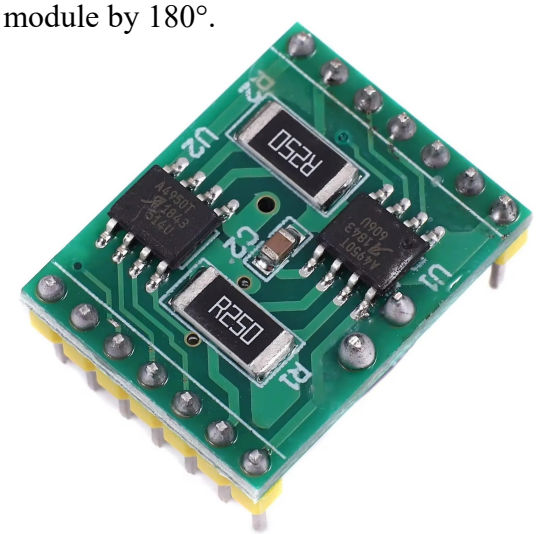
The device is powered by a voltage of 24V, which is required for the correct operation of the slave clock. However, the ESP32-C3 SuperMini module needs a voltage of 5V and therefore a suitable step-down module is used as a power source.

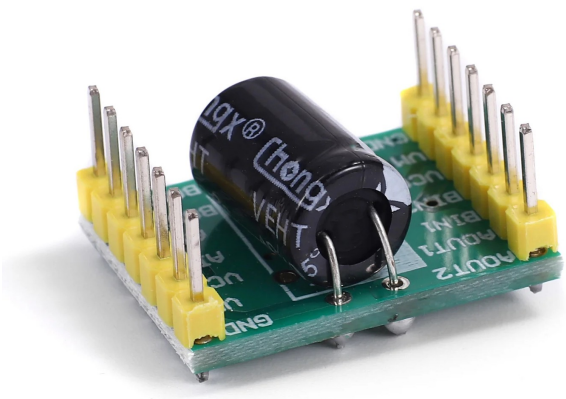




The selected converter is sufficiently dimensioned both in terms of input voltage (up to 40V) and output current (up to 3A).

One of the most important components of the entire device is the H-bridge, with which polarized pulses are sent to the slave clocks. For this task, I chose the A4950 module equipped with two bridges, but only one of them is used. The second one can (due to the symmetrical design of the module) be considered as a "spare". In case of failure of the first bridge, it is enough to turn the module by 180°.





The rather awkwardly labeled "VCC" inputs are connected to the V_{REF} inputs of both A4950 chips. The V_{REF} input is used to set the current limit. According to the datasheet for the A4950 chip, $I_{MAX} = V_{REF}/(10*R_s)$, where R_s is the sense resistor. Sensing resistors with a value of 0.25Ω are used on the module. After substitution, the simplified relationship $I_{MAX} = V_{REF} * 0.4$ comes out. The maximum value of the voltage at the V_{REF} input is 5V, and at this voltage the output current of the bridge is limited to 2A. This is too much for my needs, so I reduced the voltage at the V_{REF} input using a resistor divider, consisting of resistors R2 and R3. It can be deduced that under the given conditions (voltage on the divider 5V and sensing resistor 0.25Ω) the following relationship applies between resistors R2 and R3:

$$R2 = R3 * \left(\frac{2}{I_{MAX}} - 1\right)$$

I chose I_{MAX} = 0.5A, so between the resistors R2 and R3, the ratio R2 = R3 * 3 should apply in my case. In the actual design, I used resistors with a resistance of 6.8 kΩ and 2.2 kΩ, which corresponds to I_{MAX} = 0.49A.
Resistor R1 limits the current of the WiFi LED. The choice of its value depends very much on the luminosity of the LED used and on the required light intensity. In any case, resistor R1 should not be less than about 330Ω. Resistor R4 is a 07D390K varistor that serves to protect the A4950 chip.
A WS2812 multi-color diode is used as a status diode.
An AUX connector is fitted on the printed circuit board. With its use, it will be possible to expand the device with additional functions in the future. For example, programmable channels for controlling periodic events (like the school bell).
Using the UART connector, it is possible to connect a serial console or, for example, a logger. There is no firmware support for this yet.
The supply voltage can be connected either using the usual "barrel jack" type connector (5.5/2.1mm) or using a screw terminal block, which also has an output of polarized pulses.