UUGear

Witty Pi 5 HAT+

Realtime Clock and Power Management for Raspberry Pi

User Manual (revision 1.00)

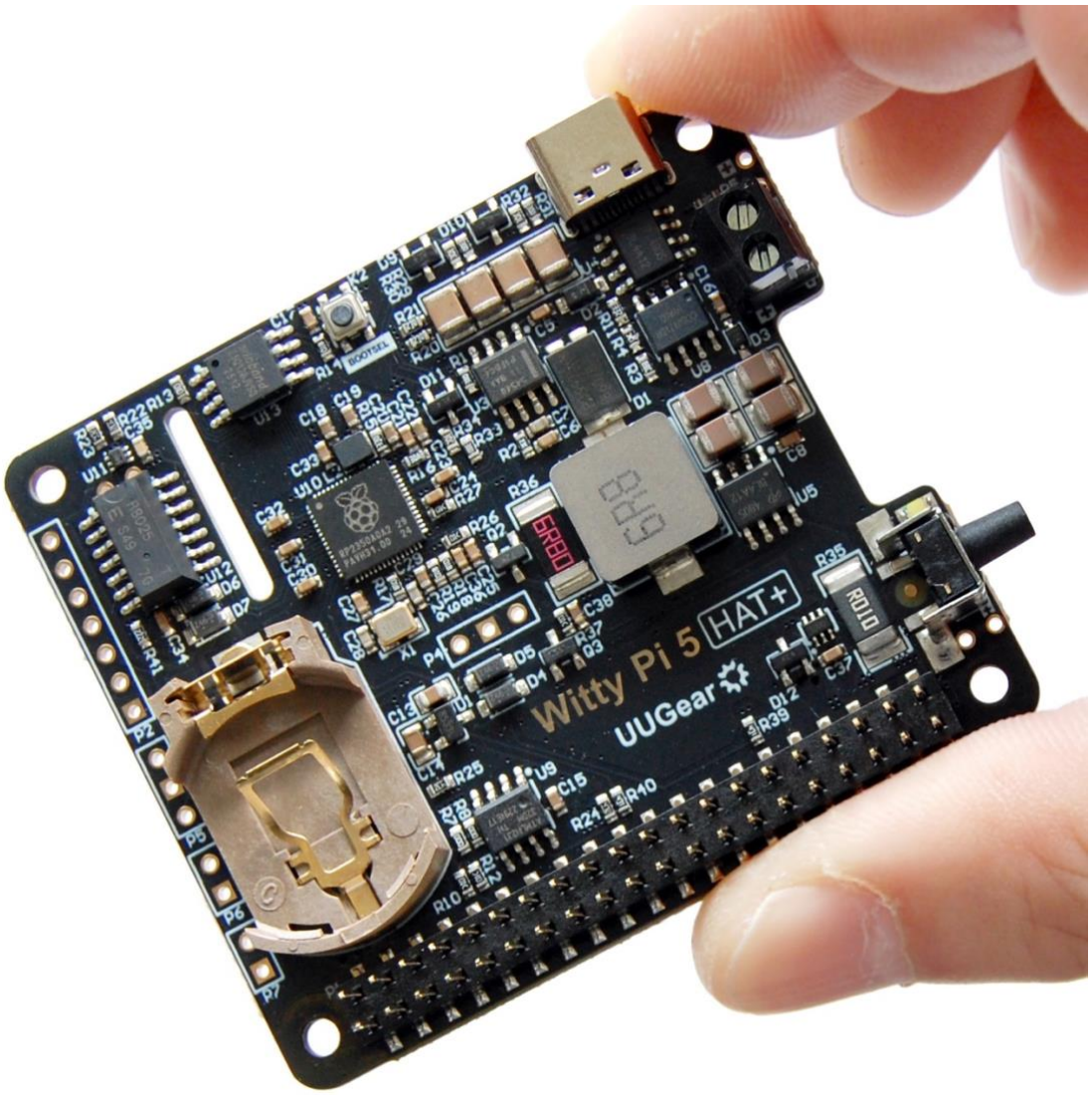
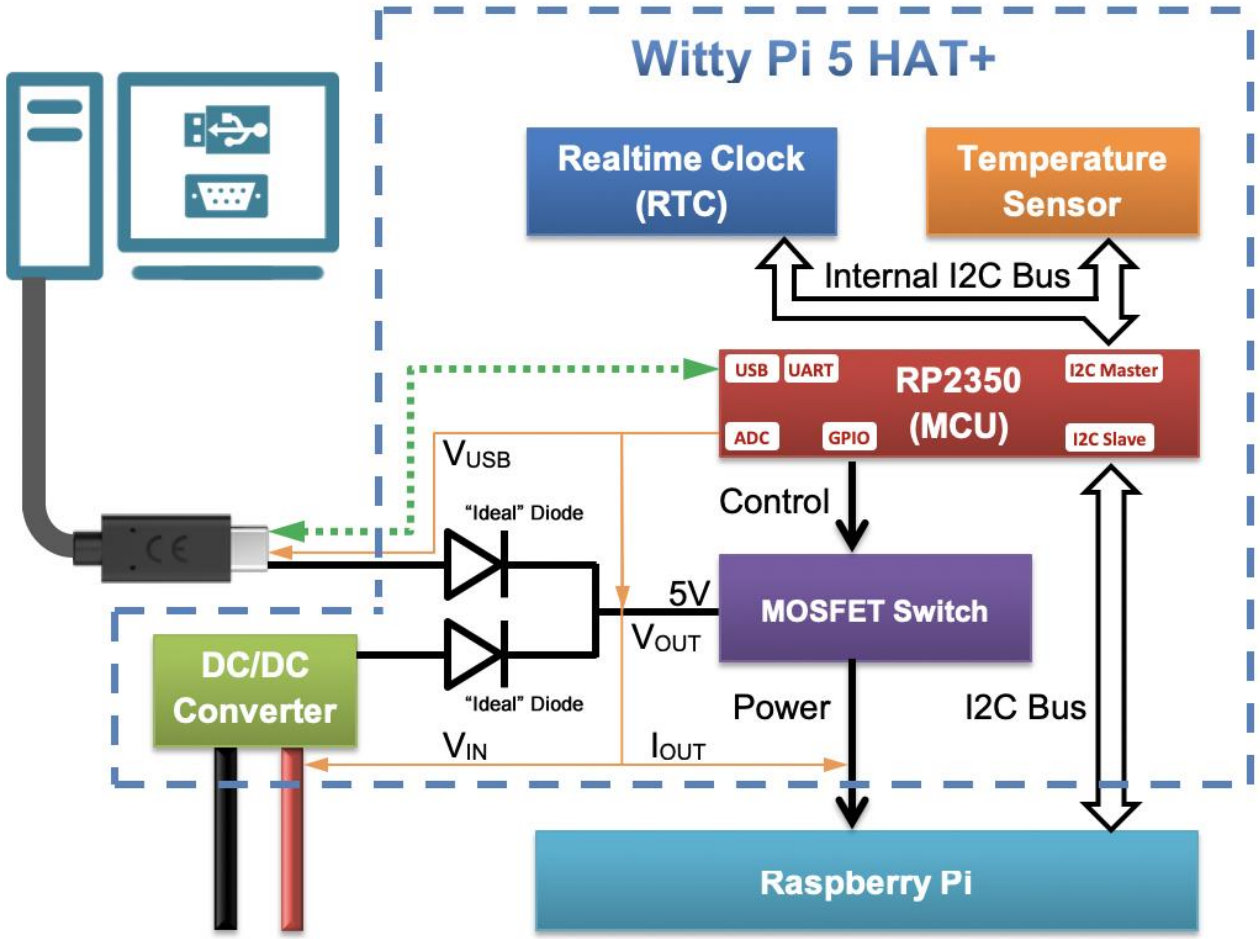


Table of Contents

1. Overview



1.1 Introduction

Witty Pi is an add-on board for Raspberry Pi that brings real-time clock (RTC) and power management functionalities to your Raspberry Pi. It can define the on/off sequence of your Raspberry Pi, perform power on/off actions based on temperature or voltage thresholds, and significantly reduce overall energy consumption.

Witty Pi 5 HAT+ is the fifth generation in the Witty Pi product line, and it is also the first Witty Pi board that complies with the Raspberry Pi HAT+ specification. For simplicity, we will refer to it as Witty Pi 5 later.

Powered by the **RP2350 microcontroller**, Witty Pi 5 can emulate both a USB flash drive and a USB serial device through its USB Type-C port. This allows the user to manage configuration files and scripts on Witty Pi directly from a computer, and also monitor real-time logs via the serial interface.

Communication between Witty Pi 5 and the Raspberry Pi is handled exclusively via the I²C interface, meaning that no additional GPIO pins on the Raspberry Pi are used.

Witty Pi 5 includes the following hardware features:

- A high-precision real-time clock, with an accuracy of ±3.8 to 5 ppm.
- A dedicated temperature sensor with 0.0625°C resolution.
- An onboard DC/DC converter that accepts up to 30V DC input.
- Two "ideal" diodes that isolate V_{IN} and V_{USB} channels.
- An RP2350 microcontroller (MCU) with an external 16MB flash memory.
- An ID EEPROM compatible with HAT+ specification.

1.2 What's New in Witty Pi 5?

Compared to Witty Pi 4, Witty Pi 5 introduces many significant new features and improvements.

Witty Pi 5 is implemented according to the Raspberry Pi HAT+ standard, and is categorized as a **Mode 1 Power HAT+**. It can supply up to **5A** current output, making it suitable for high-power applications.

One major architectural change is that the schedule script is now stored and processed entirely on the Witty Pi board, rather than on the Raspberry Pi as in previous generations. This allows Witty Pi 5 to independently handle scheduling logic, ensuring reliable operation even if the Raspberry Pi crashes or experiences issues during power on/off transitions.

Witty Pi 5 supports a new type of schedule script: the .act script, in addition to the traditional .wpi script used in earlier versions. These two script types are designed for different use cases:

- The **.wpi** script defines a looped schedule, where the user specifies a few states within a repeating cycle. This allows for concise definitions of long-term schedules. However, some users found it less flexible—especially when trying to schedule the Raspberry Pi to power on at a specific time, which could require complex calculations.
- The **.act** script, on the other hand, is linear and event-based, essentially a timestamped list of actions. It allows the user to easily specify exact times for power-on/off events, making it much more intuitive for one-time or irregular schedules. However, .act scripts are typically longer and less compact when defining repetitive behaviors.

By supporting both .wpi and .act scripts, Witty Pi 5 provides more versatility and flexibility, allowing users to choose the scripting model that best fits their application. Both script types can be converted into **.skd** files, which are the actual schedule files executed by the firmware.

Witty Pi 5 is also more developer-friendly for firmware and software customization. Both its firmware (written in C using the pico-sdk) and its software (written in C) are fully open-source, allowing users to modify and adapt them as needed.

Just like other RP2350-based boards, firmware updates are very easy: by holding the BOOTSEL button while connecting the USB cable, the RP2350 enters USB boot mode and presents itself as a virtual USB drive. You can then drag-and-drop the compiled .uf2 firmware file to flash it.

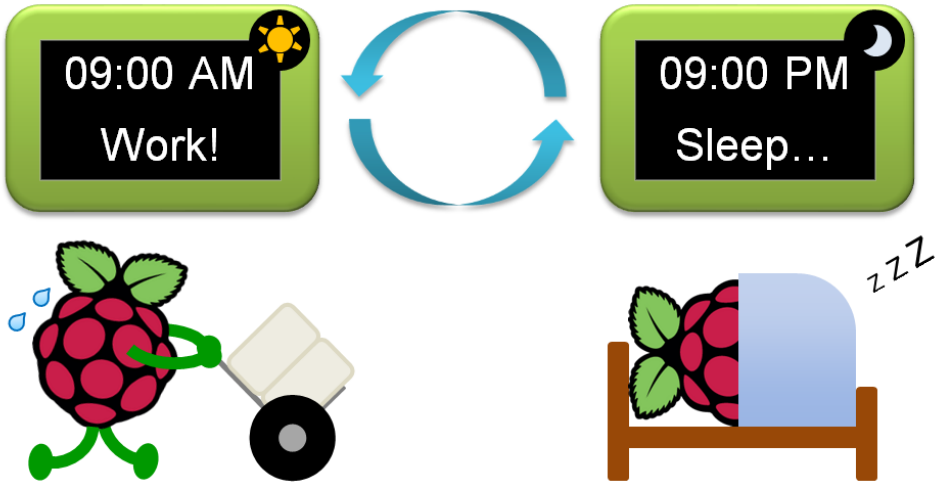
Witty Pi 5 also emulates both a **USB flash drive** and a **USB serial device**. This allows users to edit configuration files, view logs stored on the board, and monitor real-time logs via the serial interface—all of which lower the barrier for development and debugging.

1.3 Application Scenarios

1.3.1 Time Controlled Device

The high-precision RTC (real-time clock) on Witty Pi 5 has built-in temperature compensation and provides accurate timekeeping even without a network connection.

With Witty Pi 5, you can define the power-on and power-off times for your Raspberry Pi, or even create a complex schedule using scripting to fit your specific needs.

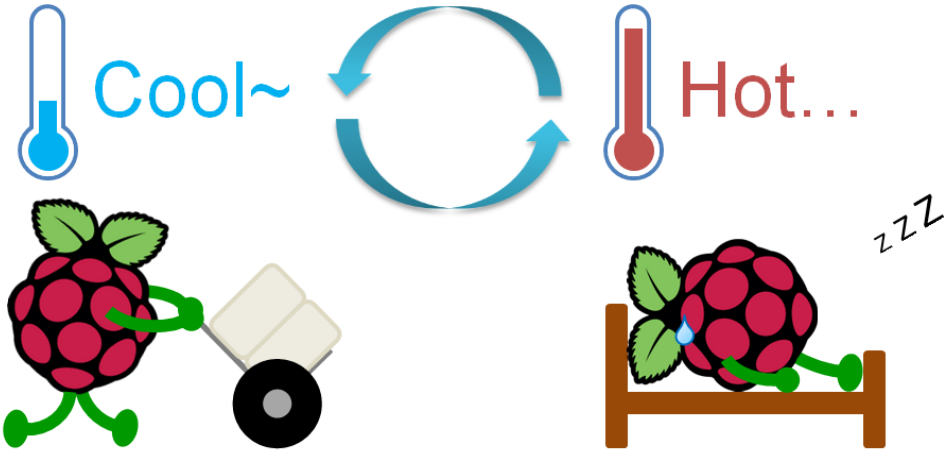


By powering down the Raspberry Pi when it's not needed and powering it up only at the right time, you can significantly reduce overall power consumption. This feature is especially valuable for battery-powered devices.

1.3.2 Temperature Controlled Device

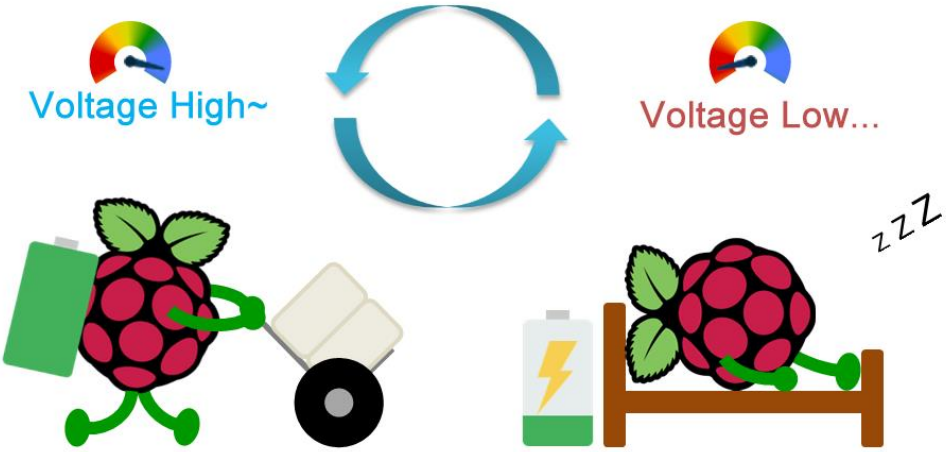
The temperature sensor on Witty Pi 5 offers a resolution of 0.0625°C and supports both high and low temperature threshold interrupts.

You can configure Witty Pi 5 to power on or off the Raspberry Pi based on temperature thresholds, effectively turning your Raspberry Pi into a temperature-controlled device.



1.3.3 Voltage Controlled Device

When powered via the VIN terminal block, Witty Pi 5 can monitor the input voltage and make power-on/off decisions accordingly.



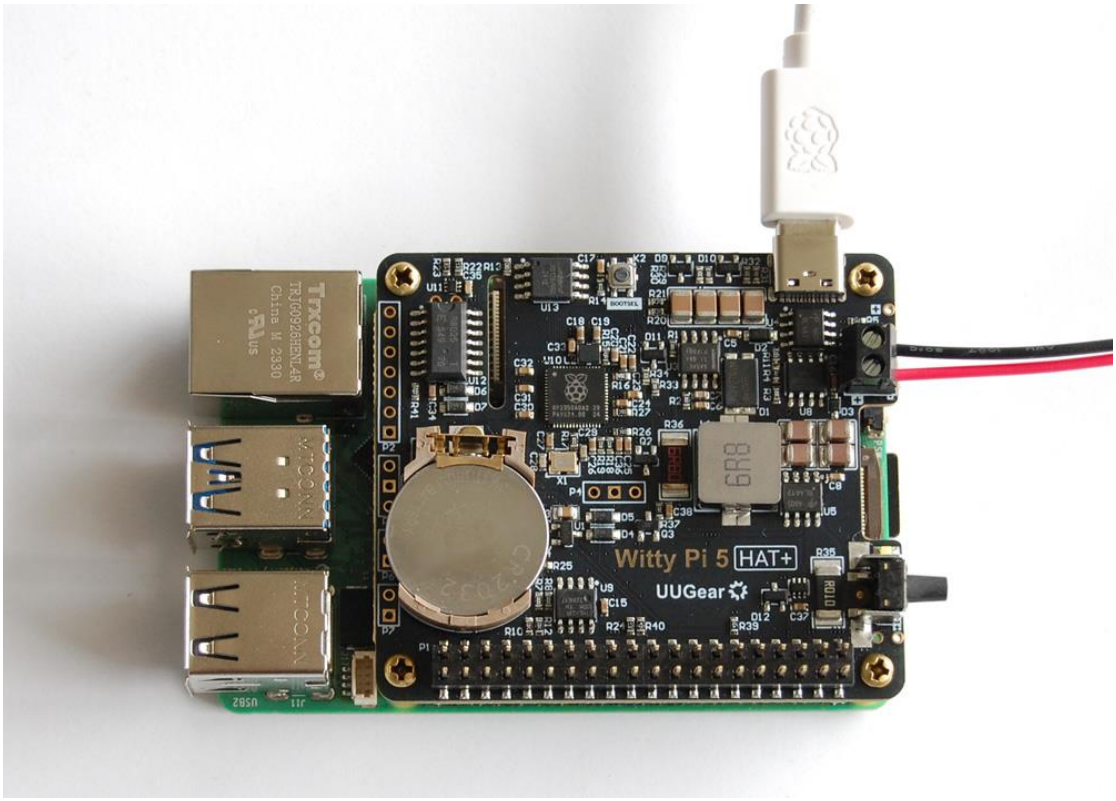
This feature is commonly used in battery-powered applications, where the Raspberry Pi can be shut down when the battery voltage drops too low and powered up again once the voltage recovers to a safe level.

1.3.4 Uninterruptible Power Supply (UPS)

Witty Pi 5 has two independent power input channels: V_{USB} and V_{IN}. These channels can operate simultaneously and back each other up.

If one power source fails, the other can continue to power the system.

Witty Pi 5 also provides software-level access to the current power source status, allowing your software to decide whether to continue running or shut down the Raspberry Pi safely.

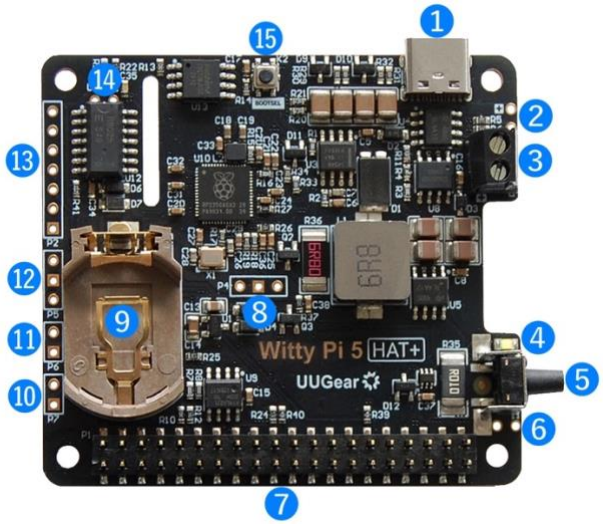


1.4 Board Layout

Witty Pi 5 supports all Raspberry Pi models with a 40-pin GPIO header, including A+, B+, 2B, Zero, Zero W, Zero 2 W, 3B, 3B+, 3A+, 4B, and 5B.
For models like Zero, Zero W, and Zero 2 W, you will need to solder a 40-pin header onto the board beforehand to ensure a reliable connection with Witty Pi.

The PCB shape of Witty Pi 5 is the same as that of Witty Pi 4, and the positions of the switch and the USB Type-C connector are also identical.
However, some other connectors are placed differently on Witty Pi 5, which makes it physically incompatible for a direct replacement of Witty Pi 4.

The diagram below shows all available interfaces on the Witty Pi 5 board.



1. USB type C connector for 5V power input (V_{USB})
2. Solder pads for V_{USB}.
3. KF350 screw terminal block connector for higher voltage input (V_{IN})
4. White LED as status indicator
5. ON/OFF button
6. Solder pads for V_{OUT}
7. 2x20 pin stacking header for connecting to Raspberry Pi
8. Serial Wire Debug (SWD) port for RP2350 (P4)
9. 3V (CR2032) battery holder
10. Unpopulated RP2350 reset header (P7: RUN and GND)
11. Unpopulated 3V RTC time keeping battery connector (P6: $V_{b\text{-}3V}$ and GND)

12. Unpopulated serial port connector (P5: RXD, GND and TXD)
13. Unpopulated 7-pin extension header (P2)
14. Solder pads for internal I2C bus (I-SDA and I-SCL)
15. BOOTSEL button (K2)

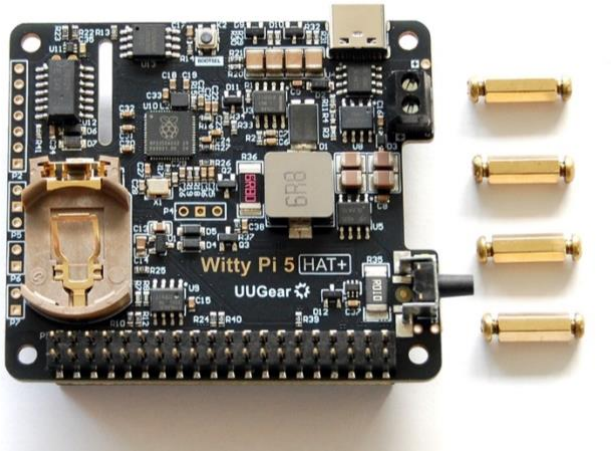
2. Specifications

Dimension	65mm x 56mm x 19mm
Weight	28g (net weight without accessories)
Microcontroller	RP2350 (datasheet)
Realtime Clock	RX8025T-UB (datasheet)
Temperature Sensor	TMP112 (datasheet)
DC/DC Converter	TPS54540 (datasheet)
MOSFET Switch	AO4805 (datasheet)
Battery	CR2032 (for time keeping only, when no power supply is connected)
Power In	DC 5V (via USB type-C connector) Or DC 6V~30V (via KF350-2P screw terminal block connector)
Output Current	Up to 5A for Raspberry Pi and its peripherals
Standby Current	~8mA (could be reduced to ~0.8mA, when this issue is resolved)
Operating Environment	Temperature -30°C~80°C (-22°F~176°F) Humidity 0~80%RH, no condensing, no corrosive gas

3. Package Content

Each package of Witty Pi 5 contains:

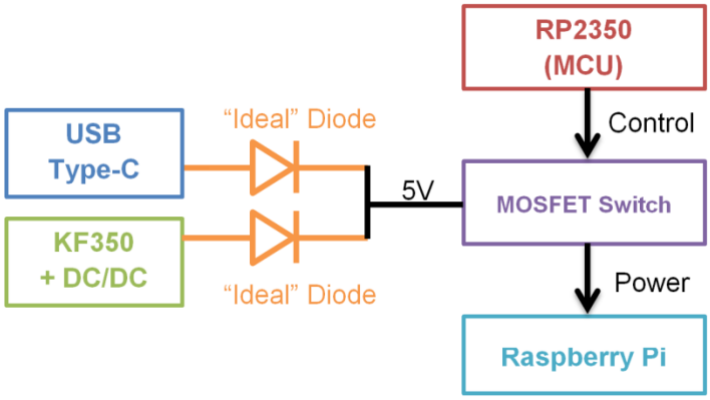
- Witty Pi 5 board x 1
- M2.5x11 copper standoff x 4
- M2.5 screws x 8



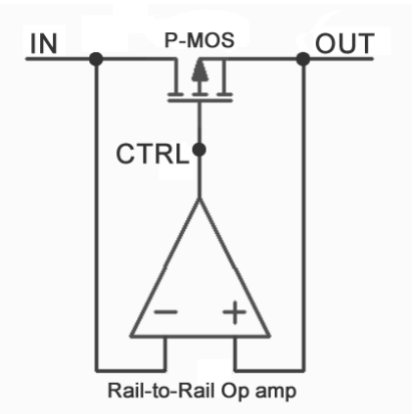
4. Working Principles

4.1 How Witty Pi 5 Powers Raspberry Pi

Witty Pi 5 provides two independent power input channels: one via the USB Type-C connector and the other via the KF350 screw terminal. These two channels are connected in parallel through two "ideal" diodes, which means they can serve as backups for each other. When one power source is cut off, Witty Pi 5's firmware detects the change and—depending on the configured strategy—either switches to the other power source to keep the system running or performs a safe shutdown of the Raspberry Pi.



The diagram on the right illustrates the "ideal" diode circuit implemented by Witty Pi 5. When V_{IN} is higher than V_{OUT}, the rail-to-rail op-amp outputs a low voltage ($V_{CTRL} \approx 0V$), turning on the P-MOSFET. When V_{IN} is lower than V_{OUT}, the op-amp outputs a high voltage ($V_{CTRL} \approx 5V$), turning off the P-MOSFET. Since the P-MOSFET (AO4805) has an extremely low $R_{DS(ON)}$ of about 20 milliohms, its voltage drop when turned on is negligible, making it behave almost like an ideal diode. This dual ideal diode configuration also prevents backfeeding between the two power sources.



The actual power path that Raspberry Pi actually uses is determined by the voltage level—whichever channel provides a higher voltage will dominate. When both channels are powered, the KF350 terminal typically wins because its DC/DC converter is designed to output 5.25V, which is usually higher than the voltage coming from the USB Type-C. However, by fine-tuning the input voltage, users can control which channel has higher priority.

A dedicated P-MOSFET is used to control whether the Raspberry Pi receives power, and this MOSFET is managed by firmware running on the RP2350. Witty Pi 5 decides when to power on or shut down the Raspberry Pi based on time, voltage, temperature, and user-defined rules.

4.2 The Three Key Factors: Time, Voltage, and Temperature

Time, voltage, and temperature jointly determine the ON/OFF state of the Raspberry Pi.

For shutdown, these three factors work in an OR logic: if any one of them triggers a shutdown condition, the Raspberry Pi will be shut down. This logic is intuitive—whether it's a scheduled shutdown, under-voltage, or over-temperature condition, each represents an urgent reason to shut the system down.

For startup, the logic is AND: all three conditions must be satisfied before the Raspberry Pi is powered on. This more conservative approach prevents the system from starting under unsafe or unstable conditions—for example, we don't want the Pi to boot on schedule only to be shut down again immediately due to low voltage.

4.3 The Schedule Scripts (.wpi, .act, and .skd Files)

The **.wpi** script format has been used in earlier versions of Witty Pi. It defines a start time, an end time, and a recurring ON/OFF sequence between them. The advantage of .wpi is its simplicity—it allows defining long-term schedules with just a few lines of code. However, some users found it too inflexible, especially when trying to set up a wake-up time precisely—they often had to perform manual calculations.

To address this, Witty Pi 5 introduces support for a new format: **.act** script. This format lists actions (power-on or power-off) with precise timestamps in a linear, human-readable format. This makes it easier to create and modify complex schedules, although long schedules can result in lengthy .act files.

Internally, Witty Pi 5 executes a **.skd** script, which is a compressed version of the .act script.
The .skd format trades readability for storage efficiency. When the firmware cannot find a .skd file, it tries to load the .act script and convert it into .skd. If that also fails, it will fall back to the .wpi script, convert it into .act, and then into .skd.

Unlike previous Witty Pi versions, Witty Pi 5 stores and processes the schedule scripts in its onboard flash memory. This makes scheduling fully independent of the Raspberry Pi. Even if the Raspberry Pi crashes during startup or shutdown, the schedule logic will continue to function properly.

4.4 Everything via I2C

Witty Pi 5 no longer uses any GPIO pins other than SDA and SCL, avoiding conflicts with other hardware and complying with the Raspberry Pi HAT+ specification.

All software interactions with Witty Pi 5 are performed over the I2C interface, including status polling, reading variables, issuing commands, and saving configuration.

Even the **watchdog** functionality is implemented via I2C. The software periodically polls a register over I2C, which will trigger the firmware to update the heartbeat value. If the firmware detects the heartbeat value hasn't been updated in time, it records a Heartbeat Missed event. After a predefined number of consecutive missed heartbeats, the firmware performs a forced power cycle to recover the system.

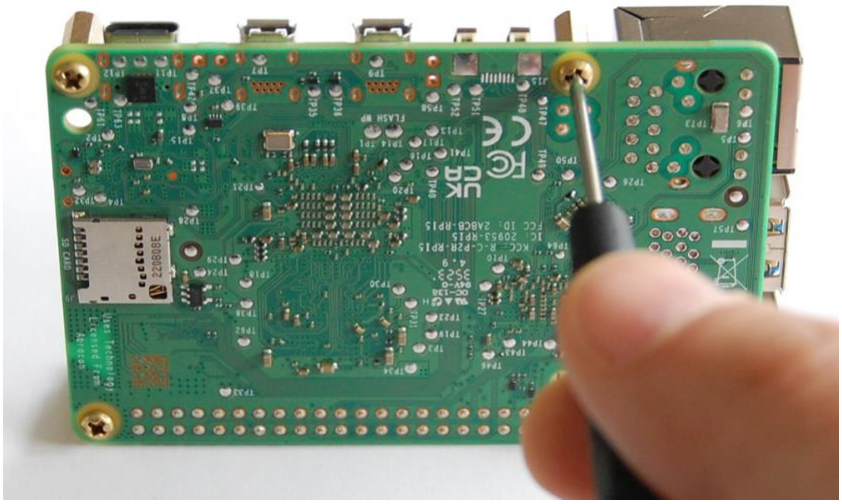
As required by the Raspberry Pi HAT+ specification, the **ID EEPROM** on Witty Pi 5 is connected to a separate I2C bus (using ID_SD and ID_SC pins), which is dedicated to board identification.

5. Hardware Installation and Software Setup

5.1 Hardware Installation

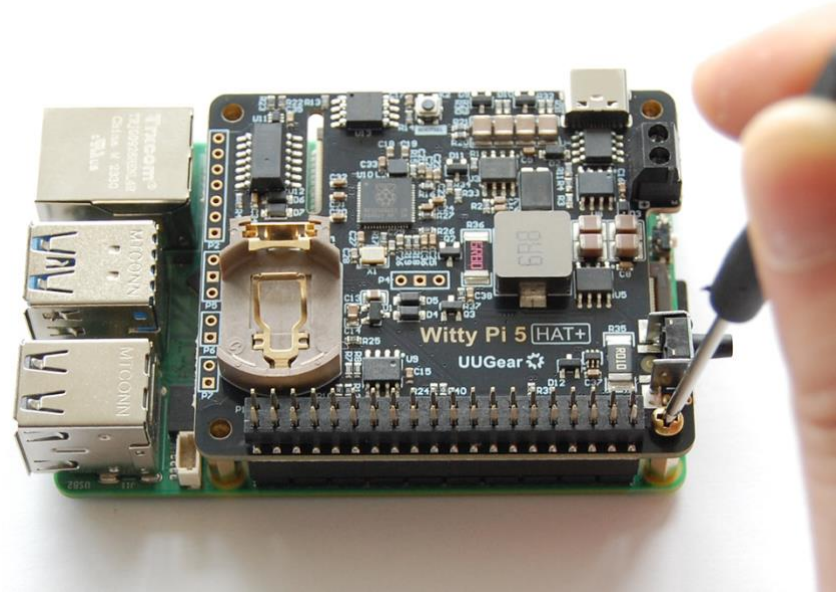
You can simply mount Witty Pi 5 onto the 2x20 pin GPIO header of your Raspberry Pi, and it will already start working. However, for a more secure and durable connection—especially in long-term deployments—you may wish to use the standoffs and screws included in the package.

First, fix the four standoffs onto your Raspberry Pi using the provided screws, as shown in the figure below:



Then your Raspberry Pi should look like this:

Next, mount Witty Pi 5's stacking header onto the 2x20 pin GPIO header of the Raspberry Pi and fasten it with the screws.

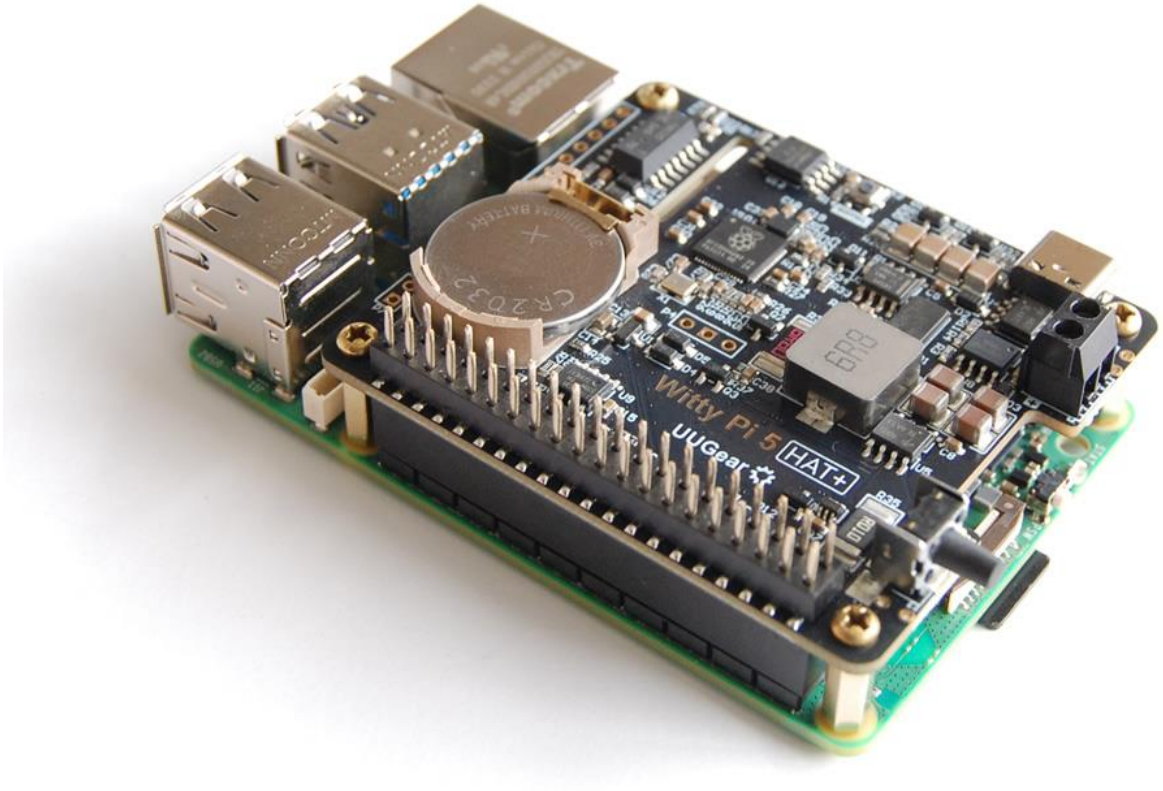


Optional: You may put a CR2032 battery into the battery holder. This battery allows Witty Pi 5 to remember the time even when no external power is connected. The RTC draws about 5μA from the battery, which means a typical battery can last for years. If your Witty Pi 5 is always connected to external power source, the battery is not required, and all functions will still work normally without it.

To allow Witty Pi 5 to fully control the power supply to your Raspberry Pi, make sure you only connect power to Witty Pi 5 (either via the USB Type-C port or the KF350 terminal block). If you supply power directly to the Raspberry Pi, Witty Pi 5 will be bypassed and will not be able to control the power state.

Once powered, Witty Pi 5 will blink its white LED every 10 seconds to indicate that it is ready. This is the default behavior and can be customized in the settings.

Now the hardware installation is complete.



5.2 Software Setup

Remarks: Witty Pi 5's software is developed and tested under **Raspberry Pi OS**.

Please make sure to enable I²C interface in your Raspberry Pi, because Witty Pi 5's software needs to talk to Witty Pi 5 hardware via I²C.

5.2.1 Installation with Internet Connection

If your Raspberry Pi has Internet connection, you may run the following command to download the latest .deb package:

```
pi@raspberrypi:~ $ wget https://www.uugear.com/repo/WittyPi5/wp5_latest.deb
```

Then install it with:

```
pi@raspberrypi:~ $ sudo apt install ./wp5_latest.deb
```

If a newer version of the software becomes available, you can repeat the above steps to update it.

5.2.2 Installation without Internet Connection

If your Raspberry Pi does not have Internet connection. You may still download the latest .deb package on another machine, and then copy the file to your Raspberry Pi to install it.

Alternatively, you may find a software .deb package located in the emulated USB flash drive, which can be used for installation. The emulated USB flash drive will be mounted to **/media/pi/Witty Pi 5** directory, when you connect Witty Pi 5 to Raspberry Pi's USB port with USB cable.

However, that software package is not likely to be the latest version. If you install that package, it is recommended to update the software when your Raspberry Pi has Internet connection.

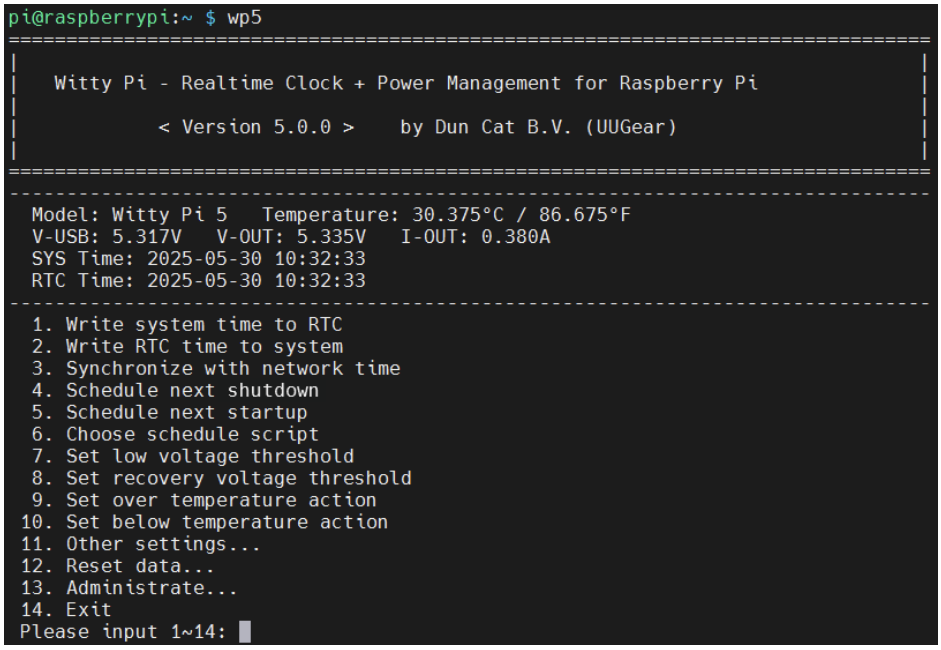
5.2.3 Uninstall the Software

To uninstall the software in the future, run:

```
pi@raspberrypi:~ $ sudo apt remove wp5
```

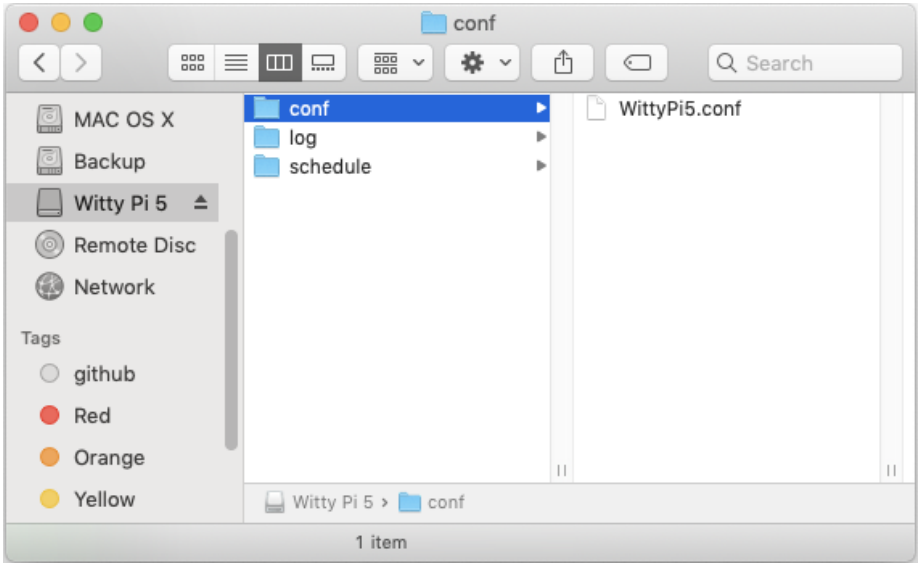
5.2.4 Run the software

To launch the software, simply type “wp5” in any directory and press ENTER.
The program will display the current device status along with a list of available options.
You can make a selection by typing the corresponding number and pressing ENTER.



6. The Emulated USB Flash Drive

When you connect your Witty Pi 5 to a computer using a USB cable, your system will detect a USB flash drive named “Witty Pi 5”. This virtual drive has a 14MB capacity and uses the FAT12 file system.



The drive contains the following three directories:

- **conf** – stores Witty Pi 5's configuration files.
- **log** – contains Witty Pi 5's log files.
- **schedule** – holds the available schedule script files.

Important Notes

Witty Pi 5's firmware may also access the FAT12 disk (e.g., to write logs or save configuration files). To avoid conflicts, the firmware will automatically eject the USB drive before performing write operations. This is why you may sometimes notice the drive is unexpectedly ejected from your computer.

Safe Removal Procedure

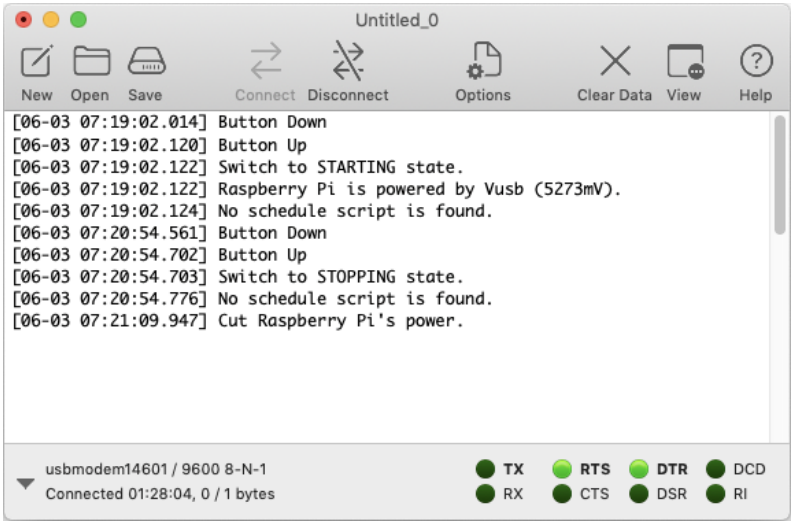
After you finish working with the files on the USB drive, it is strongly recommended to eject the drive from your operating system before physically disconnecting the cable.

If you notice the white LED on Witty Pi 5 flashing rapidly after ejecting the drive, please wait a few moments. The LED will turn off completely once all internal operations are done. Only after the LED is off, you can safely disconnect the USB cable.

7. The Emulated USB Serial Device

When you connect Witty Pi 5 to your computer via a USB cable, in addition to the USB flash drive, your system will also detect a USB serial device.

You can use your preferred serial terminal application (e.g., PuTTY, minicom, screen, CoolTerm etc.) to connect to this serial port and monitor the real-time logs from Witty Pi 5.



This is especially useful for debugging purposes.

Power Supply Consideration

Please note that most PC USB ports cannot supply enough current to allow Raspberry Pi to fully boot up.

If you plan to use the USB serial interface:

- It is recommended to power Witty Pi 5 via the KF350 terminal block,

or

- You may power the Raspberry Pi directly (e.g., via its USB-C port), but Witty Pi 5 will not be able to control the power to the Raspberry Pi in that case.

8. About Schedule Script

In addition to manually setting the next startup or shutdown time, you can define a more complex power schedule for your Raspberry Pi using Schedule Scripts.

These scripts are plain text files stored in the onboard flash memory of Witty Pi 5, and they are executed by the firmware directly.

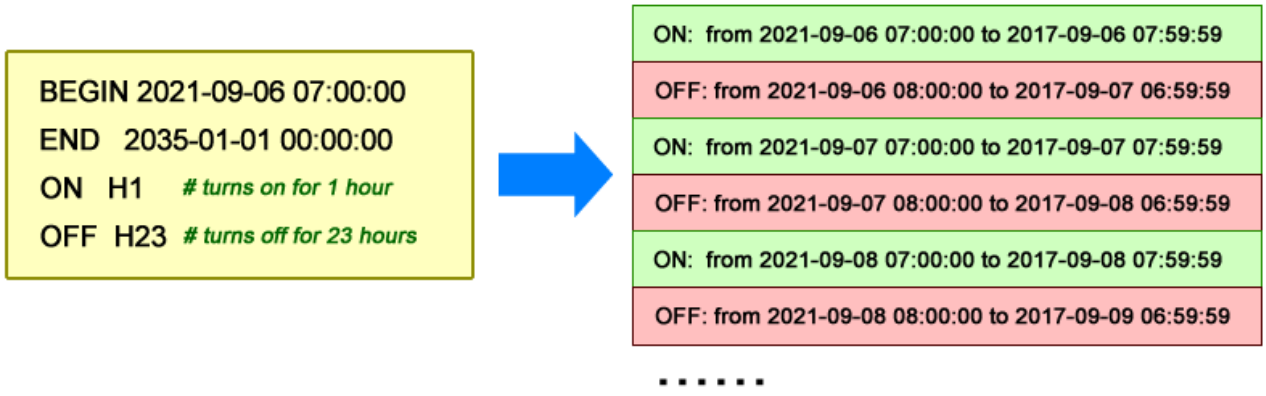
Witty Pi 5 supports three types of schedule script formats: **.wpi**, **.act**, and **.skd**.

8.1 .wpi File

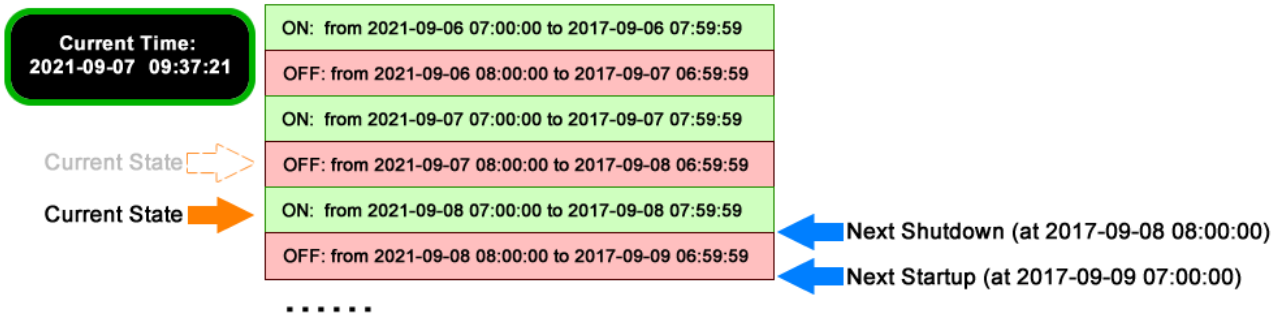
This is the schedule format used by earlier versions of Witty Pi.

Witty Pi 5 still supports this format, but unlike previous models, the script is now interpreted by the firmware instead of the software running on the Raspberry Pi.

A .wpi file defines a looped schedule with a begin time, end time, and repeating ON/OFF states within that range. This effectively divides the time axis into alternating ON and OFF states.



With .wpi file you may define a long schedule with just a few lines. However, such simplicity sometimes sacrifices a bit flexibility. For example, if you request the next shutdown time (as the end of ON state) at a moment when the Pi is in an OFF state, you will get the end time of the **next ON state**.



This behavior, known as “shifting ON state”, can be confusing to some users—it may require extra effort to schedule a shutdown at your exact desired moment.

8.2 .act File

This is a new format supported by Witty Pi 5.

Compared to .wpi, the .act file has a simpler structure: it is a linear list, where each line specifies a startup or shutdown time.

Its advantage is flexibility: you can schedule each operation at any specific time without needing to calculate durations or loop timings.

The downside is that for a long schedule, you may need to write many lines to cover all actions.

8.3 .skd File

The .skd file can be seen as a compressed version of the .act file.

It trades off human readability for compactness, reducing the overall size of the script.

In fact, .skd is the format that Witty Pi 5 actually executes. Both .wpi and .act scripts will be automatically converted into .skd format by the firmware before execution.

When a .wpi file is converted to .skd, unnecessary loop iterations are skipped, and only future ON/OFF transitions are translated into actual operations with timestamps.

8.4 Run a Schedule Script

Once a schedule script is selected to run, the firmware will parse it and determine the next startup and shutdown times (if any) from the current time.

These times will be saved into the configuration, and based on the current state of the Raspberry Pi (ON or OFF), the firmware will configure the RTC with the corresponding alarm.

When the RTC alarm is triggered, the firmware will perform the scheduled operation to power ON or OFF the Raspberry Pi.

8.5 Write Your Own Schedule Script

A schedule script is actually a plain text file, and you can use any text editor to create and edit it. You can then place your own .wpi file into the “schedule” directory in the USB drive emulated by Witty Pi 5, so it can be listed and chose via the software (wp5).

8.5.1 .wpi Format

Below is a very simple .wpi schedule script and it will keep your Raspberry Pi on for 5 minutes in every 20 minutes.

```
# Turn on Raspberry Pi for 5 minutes, in every 20 minutes
BEGIN  2025-06-01 00:00:00
END    2025-07-31 23:59:59
ON     M5      # keep ON state for 5 minutes
OFF    M15     # keep OFF state for 15 minutes
```

Like many other scripting languages, .wpi schedule script also uses "#" to make single line comment.

The first two lines define the time range for executing the script. Please make sure to use the correct time format (YYYY-mm-DD HH:MM:SS). You can use one or more white characters (space or tab) between BEGIN/END and the time string.

The rest of the script defines the states in the loop. It could be "ON" or "OFF", and you should define at least one "ON" and one "OFF" states in the loop. The ON and OFF states are used in pair.

You should also specify the duration of each state. You can do so by putting one or more parameters after ON/OFF text, separated by space or tab. Each parameter starts with a capital letter and follows by a number, where the capital letter is the unit of time:

- D = Days (D2 means 2 days)
- H = Hours (H3 means 3 hours)
- M = Minutes (M15 means 15 minutes)
- S = Seconds (S30 means 30 seconds)

For example, if you wish to define an ON state for one and a half hours, you can write:

```
ON    H1    M30
```

When the script engine executes this line, it will schedule a shutdown at the end of the ON state.

If you wish to define an OFF state for two days, you can write:

```
OFF   D2
```

When this line gets executed, a startup will be scheduled at the end of the OFF state.

Sometimes you may want to skip certain scheduling of shutdown/startup, and let your own program to do the job. This can be achieved by using the WAIT syntax. For example:

```
ON    M15   WAIT
```

This will keep your Raspberry Pi ON and no shutdown will be scheduled after 15 minutes, because there is a WAIT at the end of the line. The parameter M15 is here only to make sure the next OFF state can be calculated correctly and next shutdown can be scheduled properly.

Once you use WAIT in the ON state, you (or your program) are responsible for the shutdown of your Raspberry Pi. If you use WAIT in the OFF state, you will need to turn on your Raspberry Pi (manually or via external electronic switch).

After installing the software, there are some schedule scripts in the "schedules" directory, and they all have comments inside to explain themselves. You can take them as example to learn how to create the Witty Pi schedule script.

8.5.2 .act Format

Below is an example of .act file. It is a linear list, and each line is an action (UP or DN) with its associated time.

```
# Example of .act schedule script
UP 2025-06-01 00:00:00
DN 2025-06-01 00:05:00
UP 2025-06-01 00:20:00
DN 2025-06-01 00:25:00
UP 2025-06-01 00:40:00
DN 2025-06-01 00:45:00
```

You can also use # to start a comment in .act file. This format is so simple, that the only thing to take care is the correct time format (YYYY-mm-DD HH:MM:SS).

8.5.3 .skd Format

Although .skd file is the actual script format that used by Witty Pi 5, usually you don't need to write it by yourself, because the firmware will automatically convert .wpi or .act file to .skd file.

However, knowing the format of .skd file may still be useful. For example, you may want to develop a software to directly generate .skd file and let Witty Pi 5 to run it.

Below is an example. The .skd file is also a linear list, and each line is an action with its associated time. Different than .act file, the action type uses only one letter: "U" for "UP" and "D" for "Down", also the time is converted to a timestamp value (total seconds since the beginning of year 2000).

```
# Example of .skd schedule script
U707356800
D707357100
U707358000
D707358300
U707359200
D707359500
```

8.6 Using Schedule Script Generator

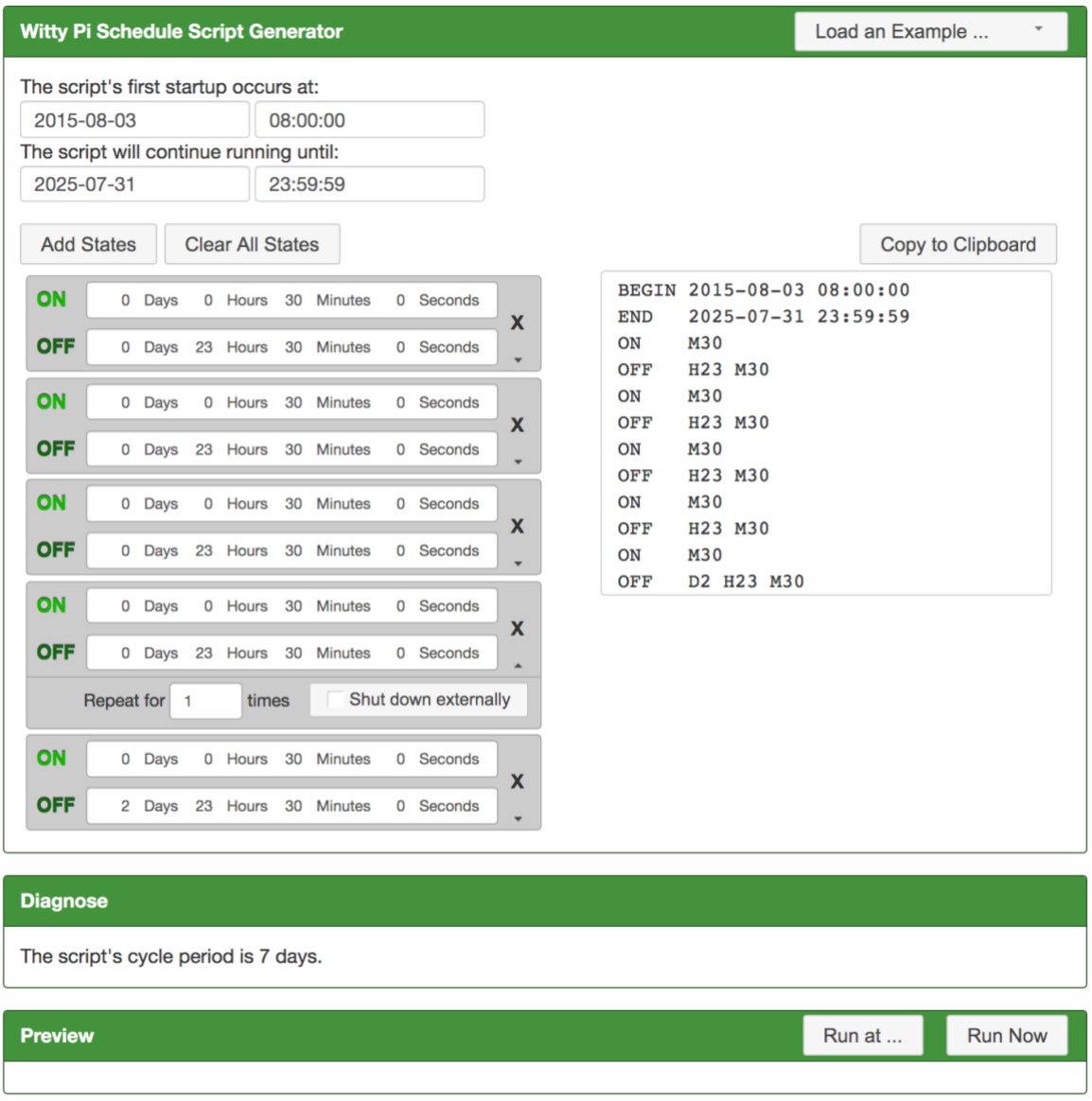
You may also use our web application to create your schedule script. At this moment only .wpi format is supported.

Just simply open this URL in your web browser:

https://www.uugear.com/app/wittypi-scriptgen/

This web application allows you to visually create the schedule script, and it immediately generate the final code on the right.

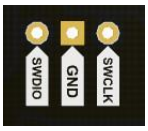
You can also click the "Run Now" button to preview how the schedule script will work. Alternatively, you can click the "Run at…" button and specify the moment to run the script.



9. Additional Interfaces

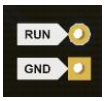
Witty Pi 5 provides a number of unpopulated interfaces in the form of solder pads. These are usually labelled on the back side of the board.

9.1 Serial Wire Debug (SWD) Port for RP2350 (P4)



This 3-pin header is the Serial Wire Debug (SWD) interface for RP2350. You can solder a connector here to connect a Raspberry Pi Debug Probe for firmware debugging purposes.

9.2 Unpopulated RP2350 Reset Header (P7)

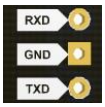


This 2-pin header (RUN and GND) allows you to connect an external reset button to the RP2350 microcontroller.

9.3 Unpopulated 3V RTC Battery Connector (P6)

This 2-pin header (Vb-3V and GND) lets you connect a 3V battery or other power source for the RTC, enabling timekeeping when no external power is present.

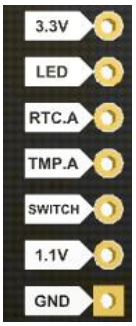
9.4 Unpopulated Serial Port Connector (P5)



This 3-pin header (RXD, GND, TXD) allows you to connect a USB-to-Serial adapter, or link it to a Raspberry Pi Debug Probe for serial communication with RP2350.

9.5 Unpopulated 7-Pin Extension Header (P2)

This header exposes several internal signals:



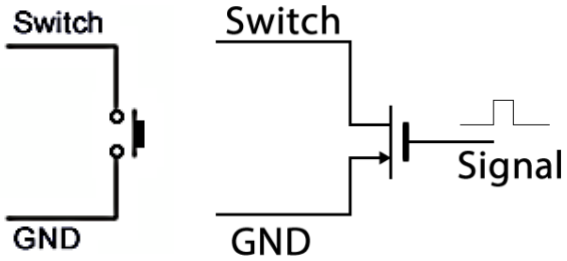
3.3V – Power supply used by RP2350, RTC, and temperature sensor (not connected to Raspberry Pi's 3.3V rail).

LED – Connected to RP2350 GPIO2 through a 2.2kΩ resistor. Intended for driving an external LED (not yet used in firmware).

RTC.A – RTC alarm output pin, connected to RP2350 GPIO8. Default is HIGH (3.3V), goes LOW when an alarm occurs.

TMP.A – Temperature sensor alert output, connected to RP2350 GPIO9. Default is HIGH, goes LOW on alert.

SWITCH – Signal from the onboard button, connected to RP2350 GPIO3. Pulled HIGH, goes LOW when pressed.



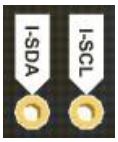
If you want to connect your own switch, you may wire the two leads to SWITCH and GND pins.

Alternatively, if you wish to trigger Witty Pi 5 with external signal, you can use an N-MOSFET to achieve this. The signal should be a positive pulse. Processing a pulse will be equal to taping the switch once. This will turn on your Raspberry Pi when it is off, or turn off your Raspberry Pi when it is on.

1.1V – The 1.1V power rail used by RP2350.

GND – Ground (connected to other GND points).

9.6 Solder Pads for Internal I²C Bus (I-SDA and I-SCL)

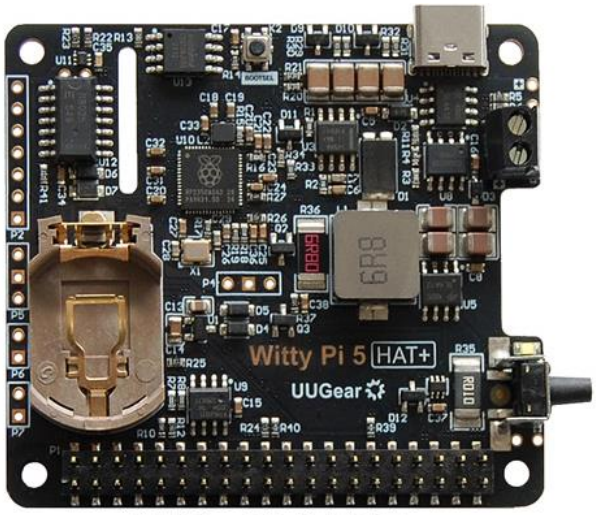


This 2-pin header exposes the internal I²C bus used by RP2350. It is not the same as the I²C bus on the Raspberry Pi. Devices connected to this bus cannot be directly accessed by the Raspberry Pi; instead, they must be accessed through register mapping implemented in Witty Pi 5 firmware.

10. Migrating from Witty Pi 4 to Witty Pi 5

Although Witty Pi 5 shares the same PCB size with Witty Pi 4, and has matching locations for the 2x20 pin stacking header, push button, and USB Type-C connector, other interfaces differ. If your project makes use of additional interfaces on Witty Pi 4, you may not be able to directly switch to Witty Pi 5.

Witty Pi 4



Witty Pi 5 HAT+

The software used for Witty Pi 4 is not compatible with Witty Pi 5. If you are upgrading to Witty Pi 5, you need to install the dedicated software package for Witty Pi 5, named **wp5**.

The .wpi schedule scripts used on Witty Pi 4 are still compatible with Witty Pi 5. To use them, copy them into the schedule directory inside the emulated USB drive of Witty Pi 5. Just like with Witty Pi 4, Witty Pi 5 will look for a file named **schedule.wpi** and execute it (after converting it to a .skd file). You may also use the software interface to select and execute a different script.

If your project communicates with Witty Pi via I²C, please be aware the I²C address has changed:

- Witty Pi 4 used address **0x08**
- Witty Pi 5 uses address **0x51**

In addition, the set of I²C registers provided by Witty Pi 5 is different from that of Witty Pi 4. Please refer to the "What I²C Registers are Provided by Witty Pi 5" section for details.

Unlike Witty Pi 4, Witty Pi 5 no longer uses any GPIO pins other than the I²C interface.

On Witty Pi 4, the push button was connected to Raspberry Pi's GPIO-4, but that is no longer the case with Witty Pi 5. If you wish to connect an external push button to Witty Pi 5, you must connect it between the **SWITCH** and **GND** pads on the Witty Pi 5 board, not between Raspberry Pi's GPIO-4 and GND.

11. Log Files for Witty Pi 5

In previous versions of Witty Pi, if the Raspberry Pi failed to start for any reason, no log would be available—because logging relied entirely on software running on the Pi itself.

Witty Pi 5 introduces a much more comprehensive logging system, with logs generated both by its software and its firmware. This is a major improvement for debugging and monitoring.

11.1 Software Log (on Raspberry Pi)

Witty Pi 5's software daemon wp5d automatically starts after the Raspberry Pi boots. It maintains its own log file at:

/var/log/wp5d.log

This file records events such as system startup, shutdown requests, and power management decisions made by the daemon etc.

11.2 Firmware Log (on Witty Pi 5)

Witty Pi 5's firmware also maintains a dedicated log file:

WittyPi5.log

This file is stored inside the **log** directory on the emulated USB flash drive. It contains the most complete and low-level debug information, which is identical to the messages you would observe via the USB serial interface.

With this dual-layer logging system, Witty Pi 5 can provide diagnostic insights even when the Raspberry Pi fails to start, making it much easier to trace hardware or timing issues.

12. Frequently Asked Questions (FAQ)

12.1 What I²C address is used by Witty Pi 5? Can I change it?

Witty Pi 5 communicates with realtime clock and temperature sensor in its internal I²C bus, and only exposes one I²C slave device to Raspberry Pi, which has an I²C address: **0x51**. This address is configurable and you can change it if you want.

If you have Witty Pi 5 connected to Raspberry Pi and run "i2cdetect –y 1" command in the console, you will see this:

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- 51 -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

If you want to change the I²C address used by Witty Pi 5, you can change the value of I²C register at position #16. For example, if you want to change the I²C address to 0x35, you can run:

```
pi@raspberrypi:~ $ i2cset -y 1 0x51 16 0x35
```

After that you also need to modify and recompile the software (wp5), so it will use the new address.

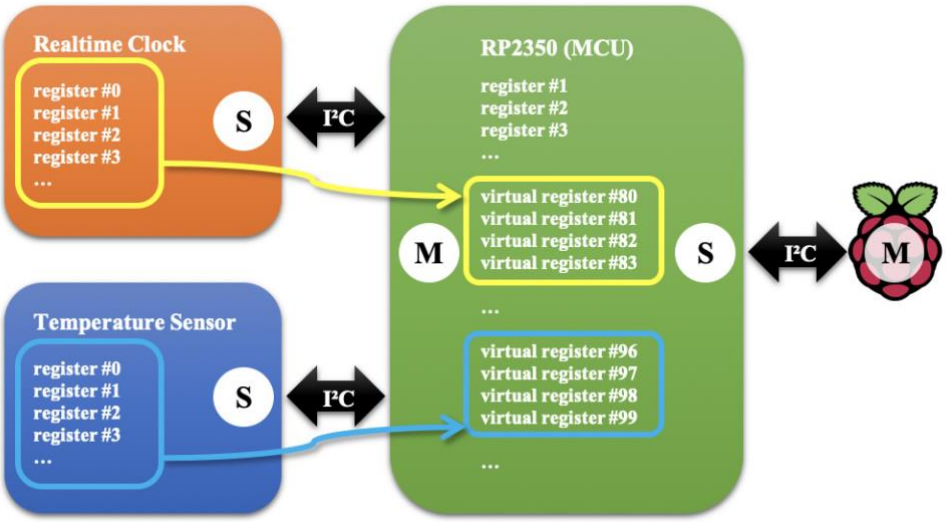
In the file "**wp5lib.h**", modify the value of **I2C_SLAVE_ADDR** from 0x51 to the value you want.

The final step is to shut down your Raspberry Pi, fully disconnect the power supply of your Witty Pi 5, and then reconnect the power supply. Then the MCU will start working with new address.

12.2 What I²C Registers are provided by Witty Pi 5?

The micro controller (RP2350) on Witty Pi 5 works as an I²C slave and Raspberry Pi can read/write its registers via I²C interface. Witty Pi 5's software configures Witty Pi 5 by setting the I²C registers accordingly.

The micro controller also implements an I²C master to access the realtime clock and temperature sensor via an internal I²C bus. The I²C registers in realtime clock and temperature sensor are all mapped as virtual I²C registers in Witty Pi 5's I²C slave device, so Raspberry Pi can also access them.



The realtime clock's registers are mapped to virtual registers #80~#95.

The temperature sensor's registers are mapped to virtual registers #96~#103.

In Raspberry Pi's view, Witty Pi 5 provides 4 kinds of I²C registers:

- Read-only I²C registers
- Normal I²C registers (configuration and administration registers)
- Virtual I²C registers

The table below shows the registers provided by Witty Pi 5. As you can see, some of them are read-only (cannot be changed, or can only be updated by the firmware itself):

Index	Description	Range	Default	Accessible
0	Firmware ID	--	0x51	Read-only
1	Firmware major version	1~99	1	Read-only
2	Firmware minor version (x100)	0~99	0	Read-only
3	The MSB of V_{USB} (in mV)	0~255	0	Read-only
4	The LSB of V_{USB} (in mV)	0~255	0	Read-only
5	The MSB of V_{IN} (in mV)	0~255	0	Read-only
6	The LSB of V_{IN} (in mV)	0~255	0	Read-only
7	The MSB of V_{OUT} (in mV)	0~255	0	Read-only
8	The LSB of V_{OUT} (in mV)	0~255	0	Read-only
9	The MSB of I_{OUT} (in mV)	0~255	0	Read-only
10	The LSB of I_{OUT} (in mV)	0~255	0	Read-only
11	Power mode: 0=via V_{USB}, 1=via V_{IN}	0 or 1	0	Read-only
12	Number of missed heartbeats	0~255	0	Read-only
13	Raspberry Pi state: 0=OFF, 1=STARTING, 2=ON, 3=STOPPING	0~3	0	Read-only
14	The latest action reason (higher 4 bits for startup and lower 4 bits for shutdown): 0=unknown, 1=startup alarm, 2=shutdown alarm 3=button click, 4=V_{IN} drops, 5=V_{IN} recovers, 6=over temperature, 7=below temperature, 8=newly powered, 9=reboot, 10=missed heartbeat, 11=external shutdown, 12=external reboot	--	0	Read-only
15	Miscellaneous state: bit0=schedule script in use	--	0	Read-only
16	I2C slave address	--	0x51	Read & Write

17	The delay (in second) between power connection and turning on Pi: 0=immediately turn on, 255=stay off	0~255	255	Read & Write
18	The delay (in second) between Pi shutdown and power cut	0~255	15	Read & Write
19	Pulse interval in seconds, for LED and dummy load	0~255	10	Read & Write
20	How long the white LED should stay on (in ms), 0=no blink	0~255	100	Read & Write
21	How long the dummy load should be applied (in ms), 0=off.	0~255	0	Read & Write
22	Low voltage threshold (x10), 0=disabled	0~255	0	Read & Write
23	Recovery voltage threshold (x10), 0=disabled	0~255	0	Read & Write
24	Power source priority, 0=V_{USB} first, 1=V_{IN} first	0 or 1	0	Read &Write
25	Adjustment for measured V_{USB} (x100)	-128~127	0	Read & Write
26	Adjustment for measured V_{IN} (x100)	-128~127	0	Read & Write
27	Adjustment for measured V_{OUT} (x100)	-128~127	0	Read & Write
28	Adjustment for measured I_{OUT} (x1000)	-128~127	0	Read & Write
29	Allowed missed heartbeats before power cycle by watchdog, 0=disable	0~255	0	Read & Write
30	Whether to write log into file: 1=allowed, 0=not allowed	0 or 1	1	Read & Write
31	Whether to allow long press BOOTSEL and then click button for factory reset: 1=allowed, 0=not allowed	0 or 1	1	Read & Write
32	Second register for startup alarm (BCD format)	--	0	Read & Write
33	Minute register for startup alarm (BCD format)	--	0	Read & Write
34	Hour register for startup alarm (BCD format)	--	0	Read & Write
35	Day register for startup alarm (BCD format)	--	0	Read & Write
36	Second register for shutdown alarm (BCD format)	--	0	Read & Write

37	Minute register for shutdown alarm (BCD format)	--	0	Read & Write
38	Hour register for shutdown alarm (BCD format)	--	0	Read & Write
39	Day register for shutdown alarm (BCD format)	--	0	Read & Write
40	Action for below temperature: 0-do nothing; 1-startup; 2-shutdown	0~2	0	Read & Write
41	Below temperature threshold (signed degrees of Celsius)	-128~127	0	Read & Write
42	Action for over temperature: 0-do nothing; 1-startup; 2-shutdown	0~2	0	Read & Write
43	Over temperature threshold (signed degrees of Celsius)	-128~127	0	Read & Write
44	bit7=mode; bit6~0: DST offset in minute, 0=disable DST	--	0	Read & Write
45	DST begin month in BCD format	--	0	Read & Write
46	mode=0: bit7~4=week in BCD, bit3~0=day in BCD mode=1: bit7~0=date in BCD	--	0	Read & Write
47	DST begin hour in BCD format	--	0	Read & Write
48	DST begin minute in BCD format	--	0	Read & Write
49	DST end month in BCD format	--	0	Read & Write
50	mode=0: bit7~4=week in BCD, bit3~0=day in BCD mode=1: bit7~0=date in BCD	--	0	Read & Write
51	DST end hour in BCD format	--	0	Read & Write
52	DST end minute in BCD format	--	0	Read & Write
53	Whether DST has been applied	--	0	Read & Write
54	System clock (in MHz) for RP2350	--	48	Read & Write
55~63	Reserved for future usage	--	--	Read & Write

64	Register to specify directory	0~255	0	Admin
65	Register to provide extra context	0~255	0	Admin
66	Register to provide download stream	0~255	0	Admin
67	Register to provide upload stream	0~255	0	Admin
68	Password for administrative command	0~255	0	Admin
69	Administrative command to run	0~255	0	Admin
70	Heartbeat register for watchdog	0~255	0	Admin
71	Shutdown request: 0=none, 1=turn RPi off, 2=RPi is shutting down, 3=RPi is rebooting	0~3	0	Admin
72~79	Reserved for future usage	--	--	Admin
80	RX8025T: Second in RTC time	--	--	Virtual
81	RX8025T: Minute in RTC time	--	--	Virtual
82	RX8025T: Hour in RTC time	--	--	Virtual
83	RX8025T: Weekday in RTC time	--	--	Virtual
84	RX8025T: Date in RTC time	--	--	Virtual
85	RX8025T: Month in RTC time	--	--	Virtual
86	RX8025T: Year in RTC time	--	--	Virtual
87	RX8025T: RTC RAM register	--	--	Virtual
88	RX8025T: RTC minute alarm register	--	--	Virtual
89	RX8025T: RTC hour alarm register	--	--	Virtual
90	RX8025T: RTC day alarm register	--	--	Virtual

91	RX8025T: RTC timer Counter 0	--	--	Virtual
92	RX8025T: RTC timer Counter 1	--	--	Virtual
93	RX8025T: RTC extension register	--	--	Virtual
94	RX8025T: RTC flag register	--	--	Virtual
95	RX8025T: RTC control register	--	--	Virtual
96	TMP112: MSB of temperature	--	--	Virtual
97	TMP112: LSB of temperature	--	--	Virtual
98	TMP112: MSB of configuration	--	--	Virtual
99	TMP112: LSB of configuration	--	--	Virtual
100	TMP112: MSB of low-temperature threshold	--	--	Virtual
101	TMP112: LSB of low-temperature threshold	--	--	Virtual
102	TMP112: MSB of high-temperature threshold	--	--	Virtual
103	TMP112: LSB of high-temperature threshold	--	--	Virtual

Below is an example to read the register with index 13, to know the current state of Raspberry Pi (0x02 means Raspberry Pi is ON):

```
pi@raspberrypi:~ $ i2cget -y 1 0x51 13

pi@raspberrypi:~ $ 0x02
```

And below is an example to write the register with index 19, to set the pulsing interval to 5 second:

```
pi@raspberrypi:~ $ i2cset -y 1 0x51 19 5
```

12.3 What GPIO Pins Are Used by Witty Pi5?

The GPIO pins used by Witty Pi 5 are marked with **green** color in the table below.

GPIO (BCM)	Name	Physical		Name	GPIO (BCM)
	3.3V	1	2	5V	
GPIO 2	SDA 1	3	4	5V	
GPIO 3	SCL 1	5	6	GND	
GPIO 4		7	8	TXD	GPIO 14
	GND	9	10	RXD	GPIO 15
GPIO 17		11	12	PCM CLK	GPIO 18
GPIO 27		13	14	GND	
GPIO 22		15	16		GPIO 23
	3.3V	17	18		GPIO 24
GPIO 10	MOSI	19	20	GND	
GPIO 9	MISO	21	22		GPIO 25
GPIO 11	SCLK	23	24	CE0	GPIO 8
	GND	25	26	CE1	GPIO 7
GPIO 0	ID_SD	27	28	ID_SC	GPIO 1
GPIO 5		29	30	GND	
GPIO 6		31	32	PWM0	GPIO 12
GPIO 13	PWM1	33	34	GND	
GPIO 19	PCM FS	35	36		GPIO 16
GPIO 26		37	38	PCM DIN	GPIO 20
	GND	39	40	PCM DOUT	GPIO 21

As you can see, Witty Pi 5 only uses I2C buses and do not use any other GPIO. This maximize the chance to be compatible with other hardware.

13. Revision History

Revision	Date	Description
1.00	2025.05.28	Initial revision