Witty Pi 4:

Realtime Clock and Power Management for Raspberry Pi

User Manual (revision 1.00)

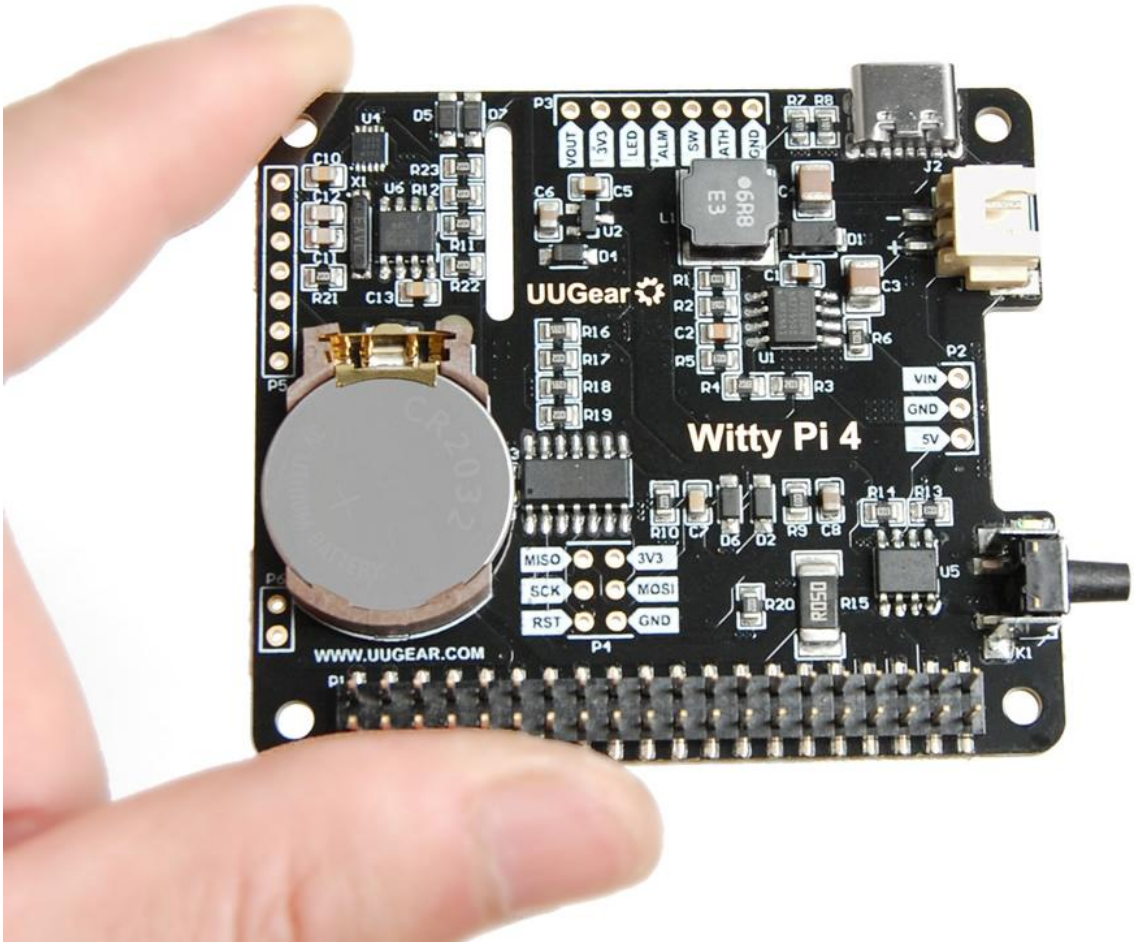
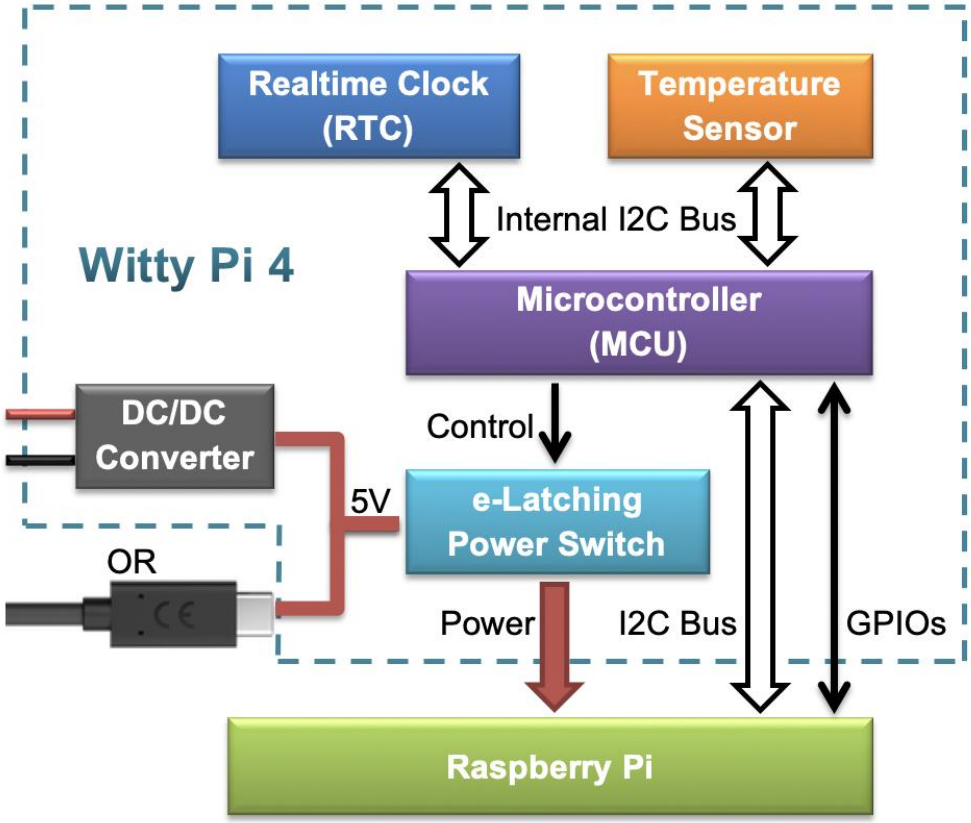


Table of Contents

1. Product Overview

Witty Pi is an add-on board that adds realtime clock and power management to your Raspberry Pi. It can define your Raspberry Pi's ON/OFF sequence, and significantly reduce the energy usage. Witty Pi 4 is the fourth generation of Witty Pi and it has these hardware resources onboard:

- Factory calibrated and temperature compensated realtime clock with ±2ppm accuracy.
- Dedicated temperature sensor with 0.125 °C resolution.
- On-board DC/DC converter that accepts up to 30V DC.
- AVR 8-bit microcontroller (MCU) with 8 KB programmable flash.

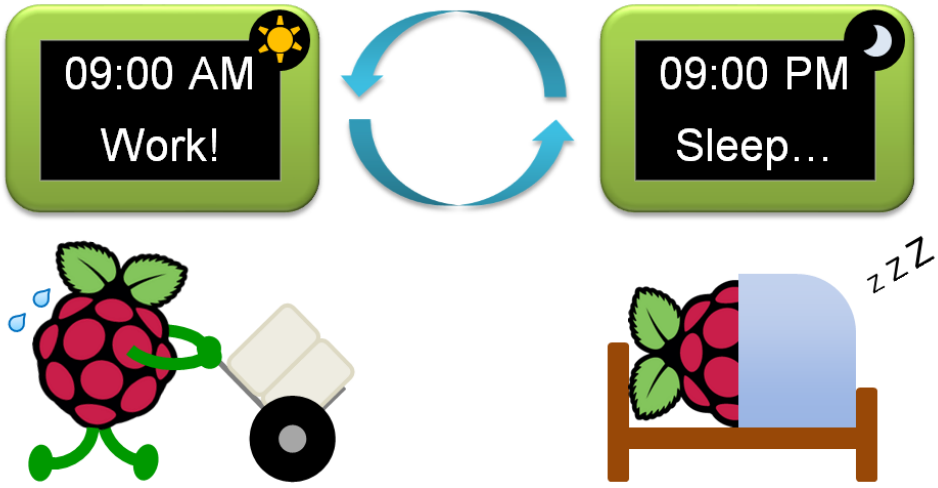


1.1 Accurate Realtime Clock and ON/OFF Scheduling

The realtime clock (RTC) on Witty Pi 4 has been calibrated in the factory and Witty Pi 4's firmware also makes temperature compensation for the crystal. This makes the RTC very accurate and the actual annual error is limited within ±2ppm (about 1 minute per year). When your Raspberry Pi boots up, the time stored in the RTC will overwrite the system time. As a result, your Raspberry Pi knows the correct time even without accessing the Internet.

You can schedule the startup and/or shutdown of your Raspberry Pi, and make it a time-controlled

device. You can even define a schedule script to schedule complicated ON/OFF sequence for your Raspberry Pi.

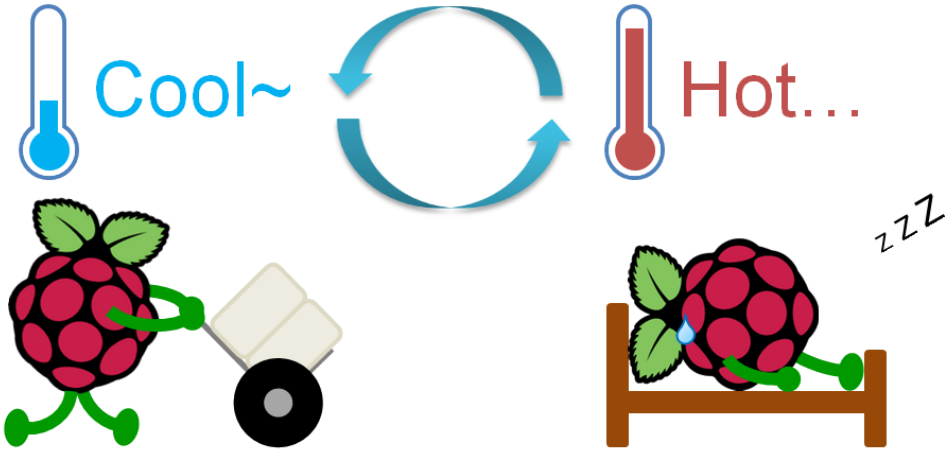


Scheduling the ON/OFF sequence for Raspberry Pi is the most popular feature of Witty Pi, and it is extremely useful for systems that powered by battery. By only turning on Raspberry Pi when necessary, the battery can work much longer with Witty Pi installed.

1.2 Temperature Controlled Device

The temperature sensor on Witty Pi 4 has 0.125 °C resolution. The temperature data is used for compensating the crystal and make the RTC more accurate.

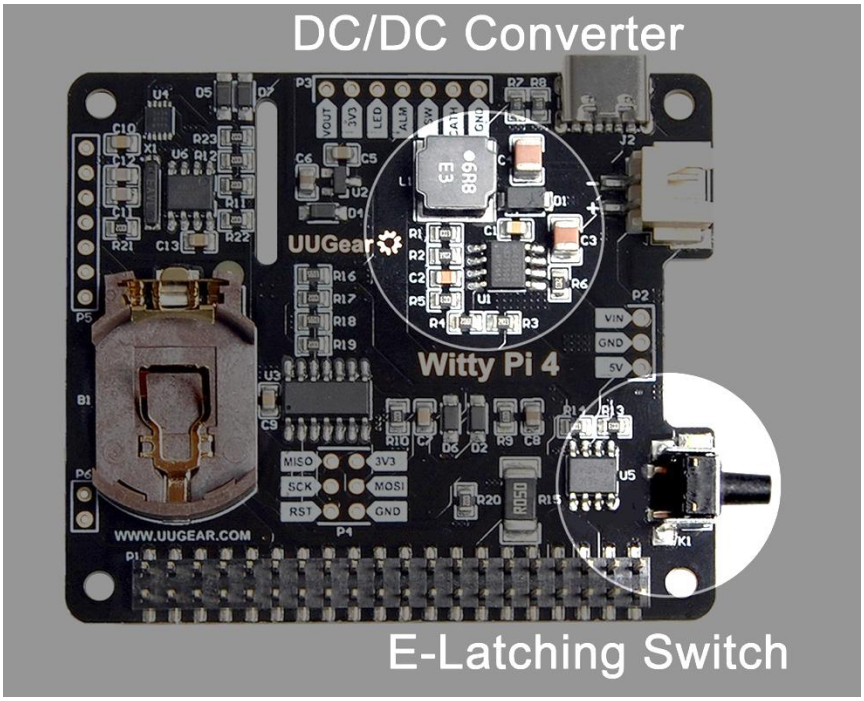
You can also specific the action (startup or shutdown) when temperature goes above or below the preset threshold. Which means you can also make your Raspberry Pi a temperature-controlled device.



1.3 DC/DC Converter and e-Latching Power Switch

Witty Pi 4 comes with a DC/DC converter on board, which allows you to power your device with 6~30V power supply. You can also power your device with 5V via the USB type C connector.

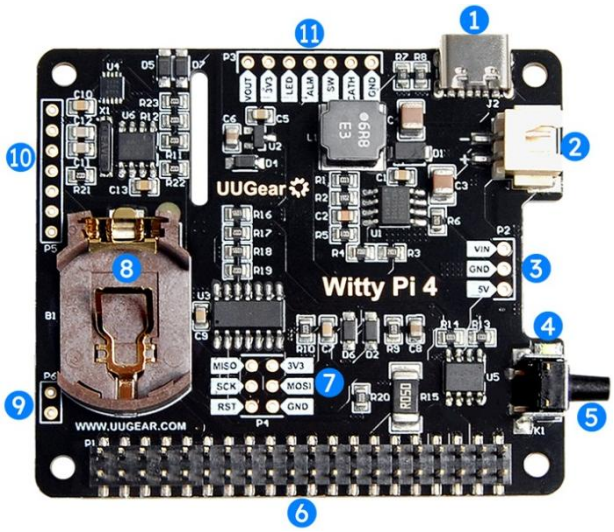
Witty Pi 4 also implements an e-Latching power switch, which is very similar to the power switch on your PC/Laptop computer. You can gracefully turn on/off your Raspberry Pi with a single tap on the button. The software running in background will execute the shutdown command before the power gets cut, and it avoids the data corruption caused by “hard” shutdown.



1.4 Interface Introduction

Witty Pi 4 supports all Raspberry Pi models that has the 40-pin GPIO header, including A+, B+, 2B, Zero, Zero W, Zero 2 W, 3B, 3B+, 3A+ and 4B. You will need to solder the 40-pin header to Zero/Zero W/Zero 2 W model beforehand, so they can make reliable connection with Witty Pi.

The picture below shows the available interfaces of Witty Pi 4.



1. USB type C connector for 5V power input (5V)
2. XH2.54 connector for higher voltage input (VIN)
3. Unpopulated power header with VIN, 5V and GND
4. White LED as status indicator
5. On/off switch
6. 2x20 pin stacking header for connecting to Raspberry Pi
7. ICSP header for uploading firmware
8. 3V (CR2032) battery holder
9. Unpopulated external 3V battery connector
10. Unpopulated 7-pin header for internal I2C bus and extra signals
11. Unpopulated 7-pin header for extension or integration

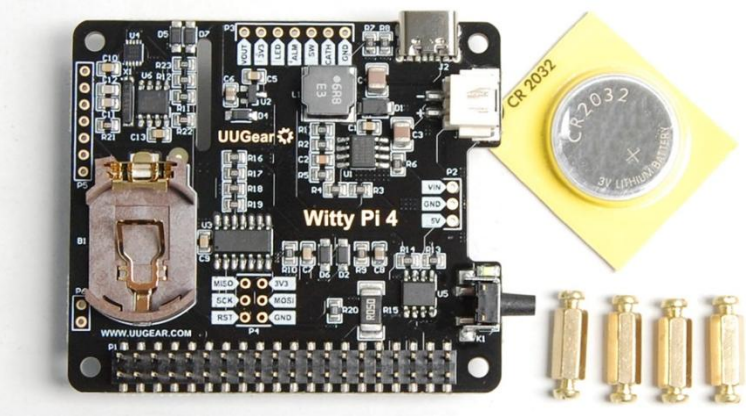
2. Specification / Technical Details

Dimension	65mm x 56mm x 19mm
Weight	23g (net weight without accessories)
Microcontroller	ATtiny841 (datasheet)
Realtime Clock	PCF85063A (datasheet), calibrated in factory.
Temperature Sensor	LM75B (datasheet)
DC/DC Converter	MP4462 (datasheet)
MOSFET Switch	AO4616 (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 XH2.54 connector)
Output Current	Up to 3A for Raspberry Pi and its peripherals
Standby Current	~0.5mA
Operating Environment	Temperature -30°C~80°C (-22°F~176°F) Humidity 0~80%RH, no condensing, no corrosive gas

3. Package Content

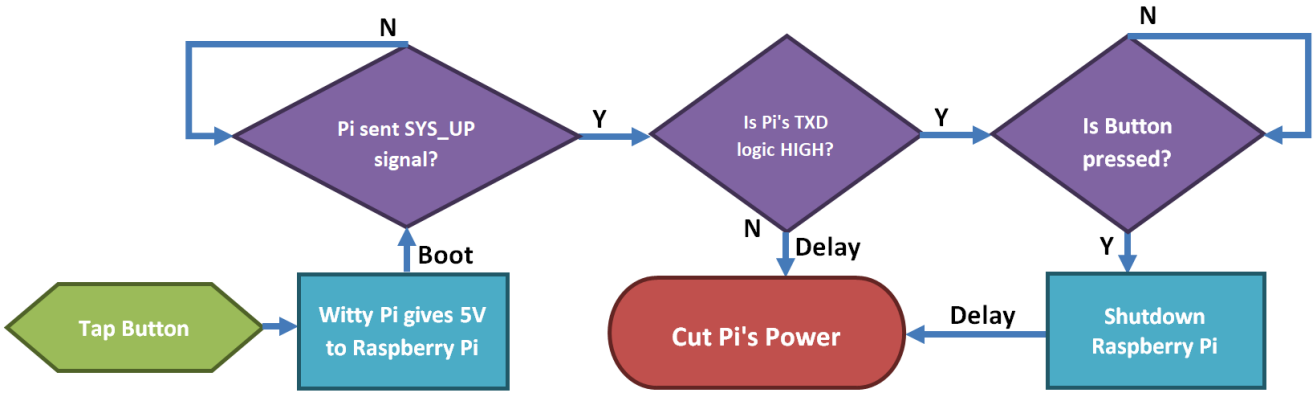
Each package of Witty Pi 4 contains:

- Witty Pi 4 board x 1
- CR2032 battery x 1
- M2.5x11 copper standoff x 4
- M2.5 screws x 8



4. How Does Witty Pi Work?

The diagram below shows the basic workflow of Witty Pi to turn on/off your Raspberry Pi.



After you tap the button on Witty Pi, it powers your Raspberry Pi via the GPIO header, and your Raspberry Pi will boot. Witty Pi's software will run automatically after boot, which will send the SYS_UP signal (via GPIO-17) to Witty Pi. Witty Pi's firmware receives the SYS_UP signal and starts to listen to Raspberry Pi's TXD pin (GPIO-14). If you shut down your Raspberry Pi (running shutdown command or choose "Shutdown" in GUI), the TXD pin will go LOW and Witty Pi will cut the power after some delay. If you tap the button again, that equals to shorting GPIO-4 to GND, and Witty Pi's software will run shutdown command for you.

Witty Pi has realtime clock (RTC) onboard and it always knows the time. You can schedule the shutdown and startup of your Raspberry Pi, which are implemented with two alarms. When the alarm is triggered, Witty Pi's firmware will emulate a "button clicking", which will turn on/off your Raspberry Pi accordingly. You can either configure single alarm for shutdown and/or startup, or you can define a simple script to plan a rather complex on/off sequence for your Raspberry Pi.

Witty Pi has Analogue to Digital Converter (ADC) in its microcontroller (MCU), which can measure the input voltage (via the white XH2.54 connector). If the voltage is dropped below the preset threshold, Witty Pi's firmware will emulate a "button clicking" to turn off your Raspberry Pi. If the voltage is raised above the preset threshold, Witty Pi's firmware will emulate another "button clicking" to turn on your Raspberry Pi again. This allows you to turn off/off your Raspberry Pi according to input voltage.

Witty Pi also has temperature sensor onboard. If the temperature rises over the preset threshold, Witty Pi's firmware can emulate a "button clicking" to turn off or turn on your Raspberry Pi. If the temperature drops below the preset threshold, Witty Pi can also turn on/off your Raspberry Pi. You can define the complete logic according to your needs, and turn your Raspberry Pi to a temperature-controlled device.

5. Software Installation, Updating and Uninstallation

5.1 Install Software

To install the software, please run this command to download the installation script:

```
pi@raspberrypi:~ $ wget https://www.uugear.com/repo/WittyPi4/install.sh
```

Then you can run the command below to install the software:

```
pi@raspberrypi:~ $ sudo sh install.sh
```

The reason that we use “sudo” here is install.sh needs to register the wittypi service (daemon.sh) to run automatically after boot, which needs root privilege.

The install.sh will also run UWI installation script. If you already have UWI installed before, it will compare the versions and update your UWI when needed.

5.2 Update Software

If there is a newer version of software and you want to install it, in most of the time you just need to repeat the installation process and let it overwrite your current version.

If you prefer, you can also fully uninstall the software (please see below) and then install it again.

5.3 Uninstall Software

To uninstall the software, you need to remove the “wittypi” directory (the one that includes the software files) and remove wittypi service from the auto-run list. Please use this command:

```
pi@raspberrypi:~ $ update-rc.d -f wittypi remove
```

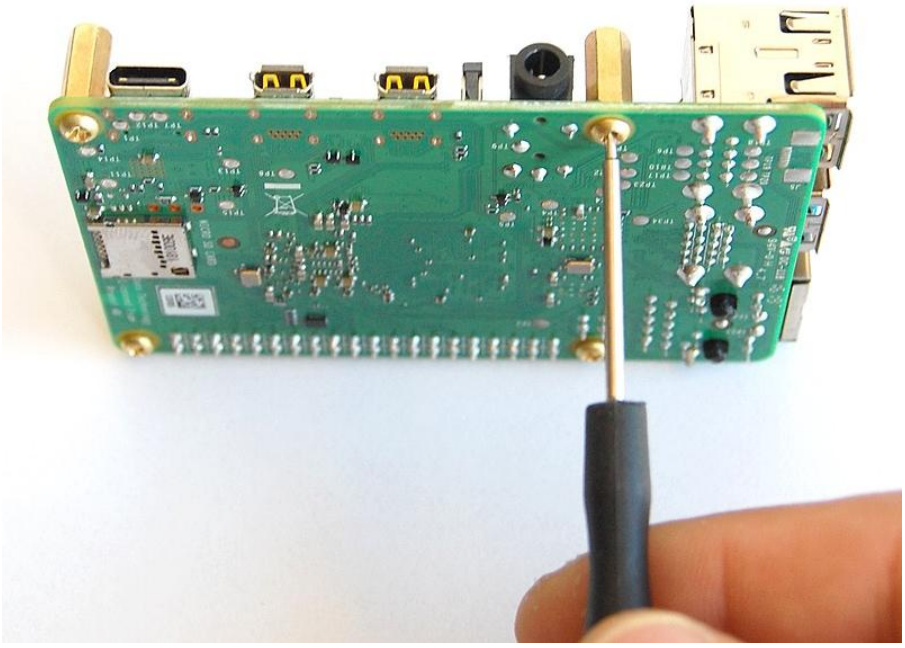
If you also want to uninstall UWI, you need to remove the “uwi” directory and remove uwi service from the auto-run list. Please use this command:

```
pi@raspberrypi:~ $ update-rc.d -f uwi remove
```

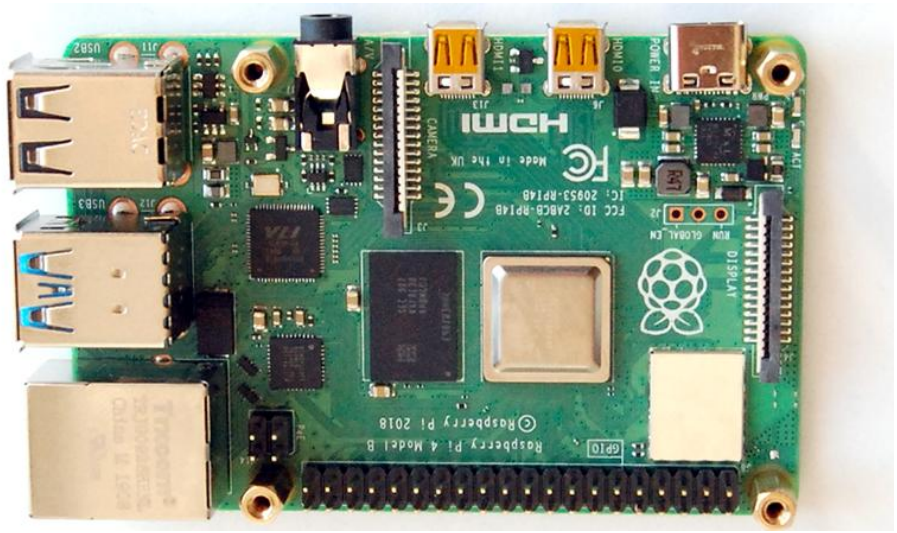
6. Mounting Witty Pi 4 on Raspberry Pi

You can directly mount Witty Pi 4 on your Raspberry Pi's 2x20pin header, and it will work. However, if you wish, you can use the copper standoffs and screws in the package to firmly mount Witty Pi 4 on your Raspberry Pi.

First, you can mount the four copper standoffs on your Raspberry Pi, using the screws.

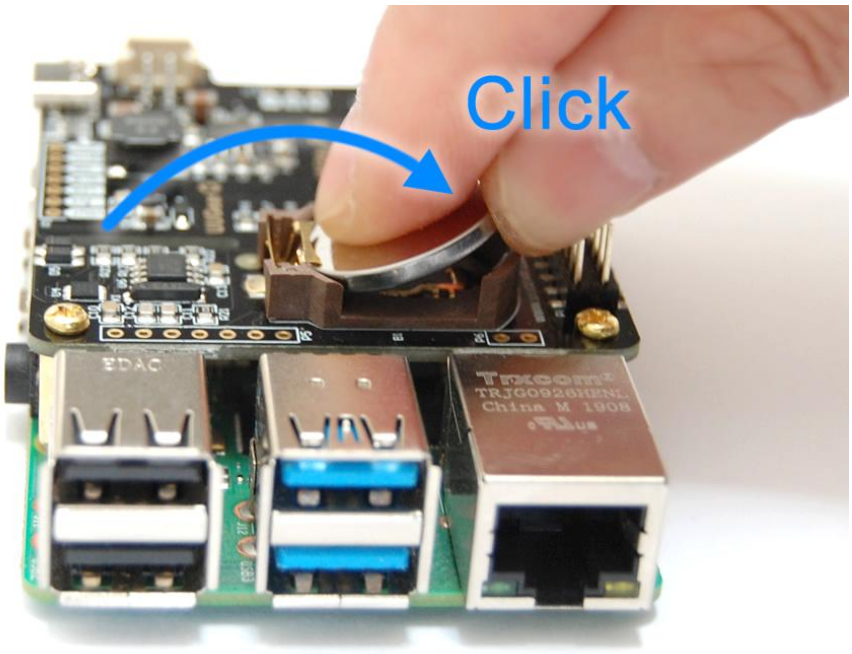


Your Raspberry Pi should look like this after mounting the four standoffs:

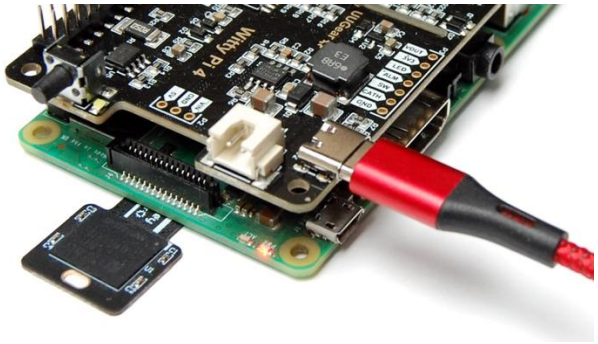


Then you can mount Witty 4's stacking header on Raspberry Pi's 2x20 pin male header, and then tighten the screws.

You can put the CR2032 button battery into the battery holder. With that battery, Witty Pi can remember the time even after you fully cut its power. The RTC only draws about 4uA current from the battery to keep the time, and the battery can last for years. **If your Witty Pi is always powered, you may omit this battery** and Witty Pi's functionalities will not get affected.



After mounting Witty Pi 4 on your Raspberry Pi, you can connect power supply to Witty Pi 4 (via the USB Type C connector or via the XH2.54 connector). **Please do not connect power supply to your Raspberry Pi directly**, because Witty Pi will power your Raspberry Pi via the GPIO header, and that is how it performs the power management.



When your Witty Pi is powered, you can see **the white LED blinks for every few seconds, which means it is standing by**.

Now your Witty Pi 4 is ready to go.

7. Software Usage

The main executable (wittyPi.sh) is a bash script, and you can run it like this:

```
pi@raspberrypi:~/wittypi $ ./wittyPi.sh
```

This interactive script displays the status of the device, and accepts your input to make configuration accordingly.

```
pi@raspberrypi ~/wittypi $ ./wittyPi.sh
================================================================================
|                                                                              |
|   Witty Pi - Realtime Clock + Power Management for Raspberry Pi              |
|                                                                              |
|            < Version 4.00 >     by Dun Cat B.V. (UUGear)                     |
|                                                                              |
================================================================================
>>> Current temperature: 35.625° C / 96.125° F
>>> Your system time is: 2022-05-15 11:14:18 CEST
>>> Your RTC time is:    2022-05-15 11:14:17 CEST
>>> Vout=5.00V, Iout=0.51A
Now you can:
  1. Write system time to RTC
  2. Write RTC time to system
  3. Synchronize with network time
  4. Schedule next shutdown
  5. Schedule next startup
  6. Choose schedule script
  7. Set low voltage threshold
  8. Set recovery voltage threshold
  9. Set over temperature action
 10. Set below temperature action
 11. View/change other settings...
 12. Reset data...
 13. Exit
What do you want to do? (1~13)
```

There are 13 options in the main menu, and you can input the number and press ENTER to confirm. Every time you press ENTER, the script will re-print the device status so you can see if the change has taken affect.

7.1 Write system time to RTC

This option will copy the time from your Raspberry Pi system to the Realtime Clock on Witty Pi. This

option should be used when you find the Raspberry Pi system time is correct (may be synchronized from the Internet) while the RTC time is not correct.

7.2 Write RTC time to system

This option will copy the time from the Realtime Clock on Witty Pi to your Raspberry Pi system. This option should be used when you find the RTC time is correct while the system time is not correct.

7.3 Synchronize time

If you choose this option, it will check if Internet is accessible, and apply network time to Raspberry Pi system and RTC when possible.

If your Raspberry Pi is not connected to Internet, choosing this option will do nothing.

7.4 Schedule next shutdown

This option allows you to specify when your Raspberry Pi should shutdown automatically.

Please notice the input format should be “DD HH:MM:SS”. DD means the day in the month, HH is the hour, MM is the minute and SS is the second. All these should be 2 digits and 24-hour system should be used.

Unlike the previous version of Witty Pi, **the “??” wildcard is no longer supported**. This is because the alarms are now implemented by the micro controller, and the MCU does not have enough resource to implement this wildcard. However, you can use schedule script to do what “??” wildcard does. Please refer the Migrating from Witty Pi 3 to Witty Pi 4 chapter.

Schedule script can actually make much more complex schedule. You can find more details in Schedule Script section.

7.5 Schedule next startup

This option allows you to specify when your Raspberry Pi should startup automatically.

Here you will need to input the startup time in the same “DD HH:MM:SS” format, and again the “??” wildcard is not supported here. You can use schedule script to do the same and even better.

7.6 Choose Schedule Script

If you want to define a complex ON/OFF sequence for your Raspberry Pi, you should schedule script.

A schedule script (.wpi file) defines a loop, with all states and their durations inside. After Raspberry Pi is boot, the “runScript.sh” will be run automatically, which will schedule the next shutdown and next startup for you, and hence a complex ON/OFF sequence could be achieved.

After you select the “Choose schedule script” option, it will list all schedule scripts in the “schedules” folder. You can choose one, and then Witty Pi will load and run it.

If you want to confirm what the script is doing, you can check the “schedule.log” file in the “~/wittypi” directory, when your Raspberry Pi is running.

If you want to create your own schedule script, please read the “Making Schedule Script” section.

7.7 Set low voltage threshold

If you are powering your Witty Pi via the XH2.54 connector, and the input voltage is lower than the low voltage threshold, Witty Pi will shut down your Raspberry Pi. Here you can specify the low voltage threshold from 2.0V to 25.0V. If you want to disable the low voltage threshold, just set it to 0, which is actually its default value.

7.8 Set recovery voltage threshold

If you are powering your Witty Pi via the XH2.54 connector, and the input voltage is higher than the recovery voltage threshold, Witty Pi will turn on your Raspberry Pi if it was previously shut down due to low input voltage. Here you can specify the recovery voltage threshold from 2.0V to 25.0V. If you want to disable the recovery voltage threshold, just set it to 0, which is actually its default value.

Low Voltage	Recovery Voltage	Result
Not Set	**Not Set**	Witty Pi doesn’t care about the input voltage changing.
Set	**Not Set**	Witty Pi will shut down your Raspberry Pi if input voltage is too low. It will not turn on Raspberry Pi unless you tap the button, or the scheduled startup comes (given input voltage is higher than low voltage threshold).
Set	**Set**	Witty Pi will shut down your Raspberry Pi if input voltage is too low. Once this happens, Witty Pi will monitor the input voltage during the sleep and will turn on your Raspberry Pi when input voltage gets higher than recovery voltage threshold.
Not Set	**Set**	Witty Pi will wake up your Raspberry Pi when input voltage is higher than recovery voltage threshold. Please mind that if input voltage is always higher than recovery voltage threshold, you will not be able to turn off your Raspberry Pi with these settings, because it always turns on Raspberry Pi again just after shutting it down.

The low voltage threshold and recovery voltage threshold can be set individually and their combinations will make Witty Pi works differently, please see the table below for details.

7.9 Set over temperature action

Here you can specify what to do if temperature exceeds a preset threshold. It does nothing by default, but you can choose to shut down or startup your Raspberry Pi when this event happens.

The action you chose will be triggered when the temperature is higher than the temperature you input. For example, if you input 55°C here, the action will be triggered when actual temperature is 56°C or higher.

Remarks: shutdown can be triggered by temperature only after 2 minutes since Raspberry Pi is on.

7.10 Set below temperature action

Here you can specify what to do if temperature drops under a preset threshold. It does nothing by default, but you can choose to shut down or startup your Raspberry Pi when this event happens.

The action you chose will be triggered when the temperature is lower than the temperature you input. For example, if you input 55°C here, the action will be triggered when actual temperature is 54°C or lower.

Remarks: shutdown can be triggered by temperature only after 2 minutes since Raspberry Pi is on.

7.11 View/change other settings...

Choosing this option will display a sub menu and allows you to set these parameters:

- **Default state when powered**
  It is “OFF” by default, which means your Raspberry Pi will not be turned on when power supply is connected to Witty Pi, and you will need to tap the button on Witty Pi to start it. Here you can set it to 1 (ON) or 0 (OFF).
- **Power cut delay after shutdown**
  Default value is 5.0 seconds, which means Witty Pi will fully cut the power after 5 seconds since Raspberry Pi has been shut down. Here you can input a number between 0.0 to 25.0.
- **Pulsing interval during sleep**
  This parameter will decide how long Witty Pi’s micro controller will wake up and drive the white LED and/or the dummy load. By default, this interval is 4 seconds. If you wish the white LED blinks slower, or the dummy load draws current less frequently, you can choose a bigger value. This value is in seconds.
- **White LED duration**
  This parameter will decide how long the white LED should stay on, when Witty Pi blinks it during the sleep. The default value is 100, and you can use bigger value if you wish the white LED to stay on for longer. Here you can input number from 0 to 255. If you input 0, the white LED will not blink at all.

- **Dummy load duration**
  This parameter will decide how long the dummy load should draw current from the power source.
  What is (pulsing) dummy load? It is a trick to keep power bank alive, when Raspberry Pi is off. If you are using power bank to power Witty Pi 4 + Raspberry Pi, after turning off Raspberry Pi and cut the power, Witty Pi 4 only draws about 1mA from the power bank, which is way too small to keep the power bank alive. If the power bank goes to sleep mode, it will not provide any power to Witty Pi 4 and hence your Raspberry Pi will not wake up when scheduled startup is due. Pulsing dummy load is such a trick, that it draws rather big current from the power bank for a short duration and it does so with a fixed interval. That way the power bank might be fooled and think the load is heavy enough, and avoid entering the sleep mode.
  The default value of dummy load duration is 0, which means disabled (no dummy load at all). You can set a value between 0 and 255 here. However, **we suggest using smallest value that could keep your power bank alive, usually 10 will do**.

- **Vin adjustment**
  The voltage is measured by the 10-bit ADC in micro controller. Due to the inaccuracy of internal voltage standard and error on divider resistors, the result could have up to 5% error. You can configure this parameter to adjust the result of voltage measurement.
  The default value is set to 0.20V, and you can input a value between -1.27 and 1.27 here.

- **Vout adjustment**
  You can configure this parameter to adjust the result of output voltage measurement.
  The default value is set to 0.20V, and you can input a value between -1.27 and 1.27 here.

- **Iout adjustment**
  The output current is calculated by measuring the voltage drop on the 0.05 Ohm sampling resistor. You can configure this parameter to adjust the result of output current calculation.
  The default value is set to 0.00A, and you can input a value between -1.27 and 1.27 here.

7.12 Reset Data…

If you want to erase some data you previously set (scheduled startup time, scheduled shutdown time, currently used schedule script, low voltage threshold, recovery voltage threshold), you can choose this option.

Once you select this option, the software will display a sub menu, which allows you to:

- **Clear auto startup time:**
  The auto-startup time will be erased and Witty Pi will not auto-start your Raspberry Pi.

- **Clear auto shutdown time:**
  The auto-shutdown time will be erased and Witty Pi will not auto-shutdown your Raspberry Pi.

- **Stop using schedule script:**
  The “schedule.wpi” file will be deleted.

- **Clear low voltage threshold:**
  The low voltage threshold will be unset.
- **Clear recovery voltage threshold**
  The recovery voltage threshold will be unset.
- **Clear over temperature action**
  The over temperature action will be cleared**.**
- **Clear below temperature action**
  The below temperature action will be cleared.
- **Perform all actions above:**
  Clear all scheduled times, remove the “schedule.wpi” file, unset the low voltage and recovery voltage thresholds, clear the over-temperature and below temperature actions. This is the fastest option to clear everything and prevent your Raspberry Pi being shut down by Witty Pi.

7.13 Exit

Selecting this option will exit the software and return to the console.

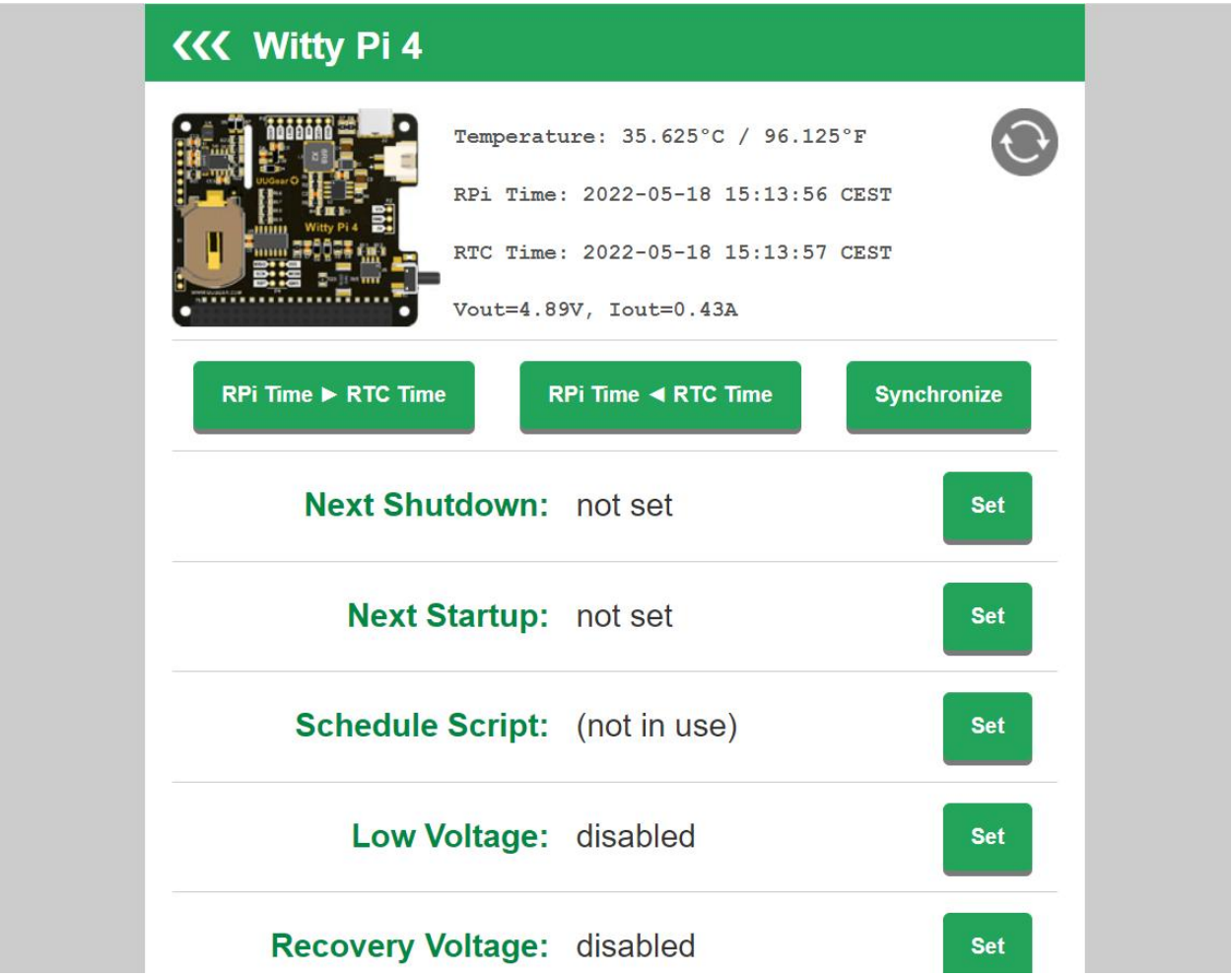
8. Using UWI (UUGear Web Interface)

UWI is a mini web server that uses very little system resource, while it allows you to monitor and configure your Witty Pi board via any device that can access the network, including mobile phones and tablets.

By default, UWI can be accessed via http://raspberrypi:8000/wittypi4/. If you cannot access it, most probably is because the host name "raspberrypi" is not resolved in your network environment. In such case, please configure your UWI to use actual IP address or accessible host name. You can also **run the "diagnose.sh" script** in "uwi" directory to do so automatically.

```
pi@raspberrypi:~/uwi $ ./diagnose.sh
```

In UWI, your Witty Pi 4 device is presented like this:



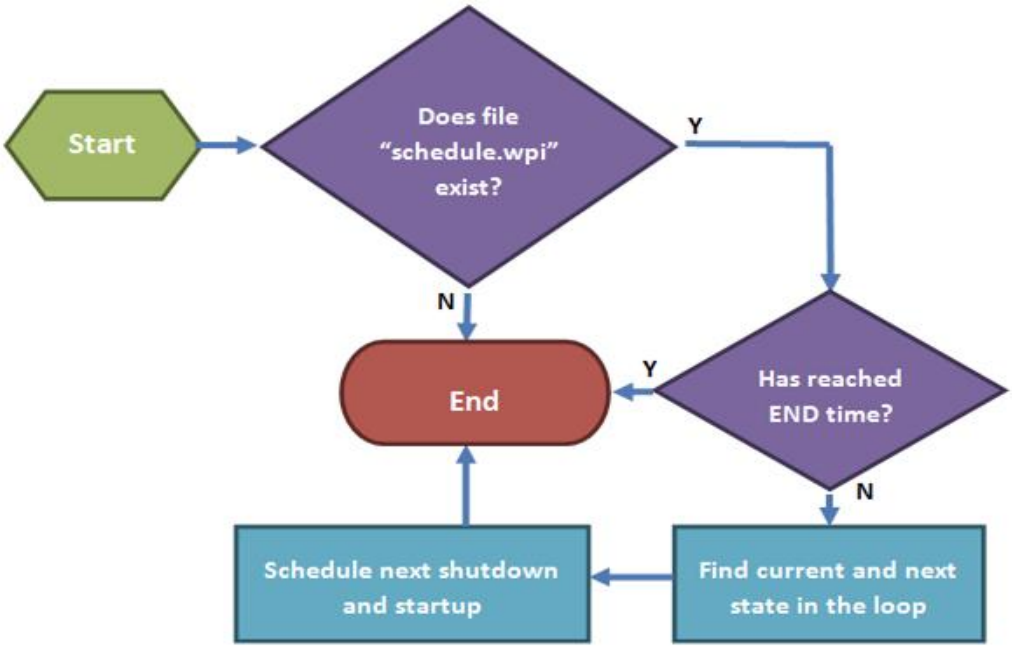
Here you can monitor and control your Witty Pi 4 device like using a website.

9. About Schedule Script

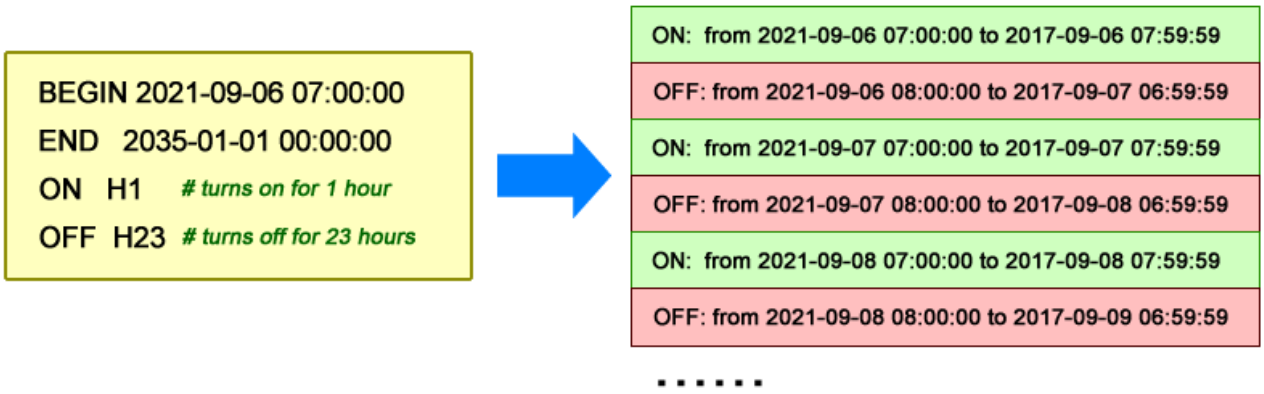
Besides manually schedule the next shutdown/startup time, you can also use a schedule script to plan the ON/OFF sequence for your Raspberry Pi. A schedule script is a text file with .wpi file extension. Witty Pi's software will load and execute it automatically after Raspberry Pi is boot up.

9.1 How does Schedule Script Work?

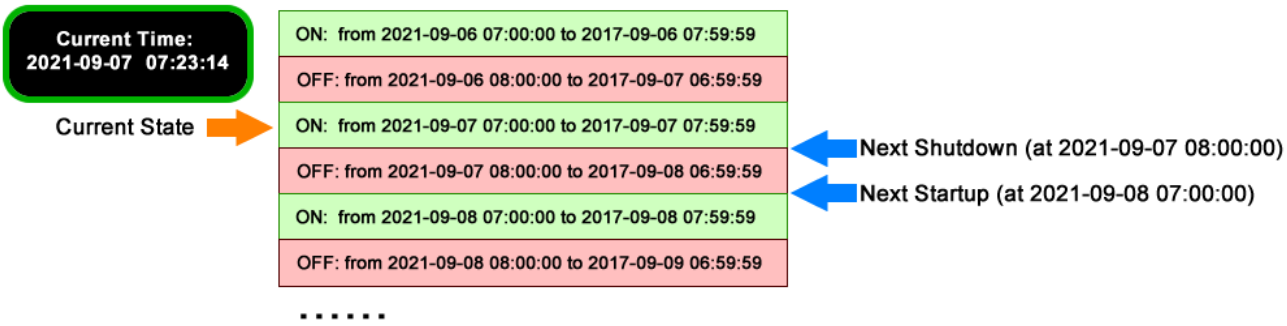
A schedule script defines a loop, and it put a serial of ON/OFF states into it. The duration of each state is also specified. At the end of ON state, there should be a scheduled shutdown, while at the end of OFF state, there should be a scheduled. All states in the schedule script will be executed in sequence and repeatedly, until the END time of the loop is reached.



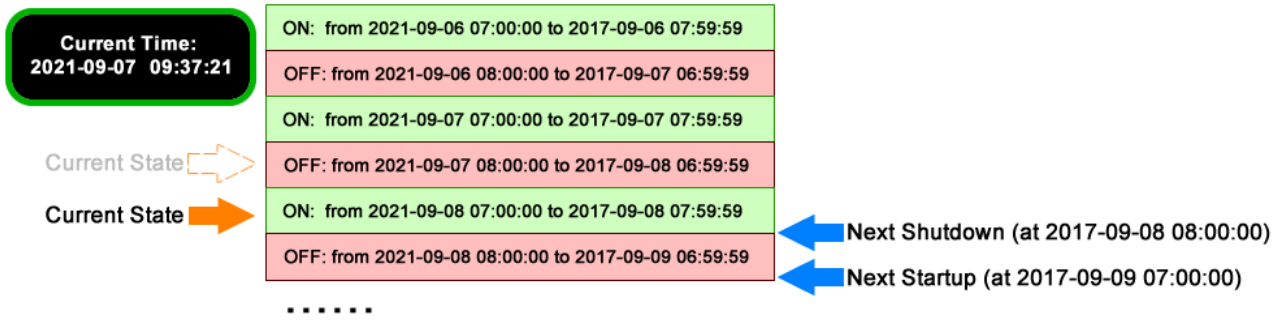
Every time your Raspberry Pi wakes up, it has a chance to run the runScript.sh file, which loads the schedule script ("schedule.wpi" file) and run it. If the current time doesn't reach the END time defined in the schedule script, the next shutdown and next startup will be scheduled automatically. When your Raspberry Pi is wakened at scheduled startup time, it will repeat this process and schedule the next shutdown and startup. Although a schedule script only defines a few ON/OFF states, they could become many ON/OFF states in reality.



Please keep in mind that, running the same schedule script at different moment may get different result, as the “runScript.sh” will search and find the proper state according to current time.



When the “runScript.sh” is executed, if the current time is located at an “OFF” state instead, it will take the next “ON” state as the current state, because it knows Raspberry Pi is currently ON.



This “shifting ON state” behavior confuses some users, because the actual scheduled shutdown/startup times are different than what they expected. In such case you can adjust the BEGIN time of the schedule script, and you will see the scheduled shutdown/startup times will move accordingly.

9.2 Make Your Own Schedule Script

A schedule script is actually a text file, and you can use any text editor to create and edit it.

Below is a very simple 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  2021-08-01 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, Witty Pi's 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.

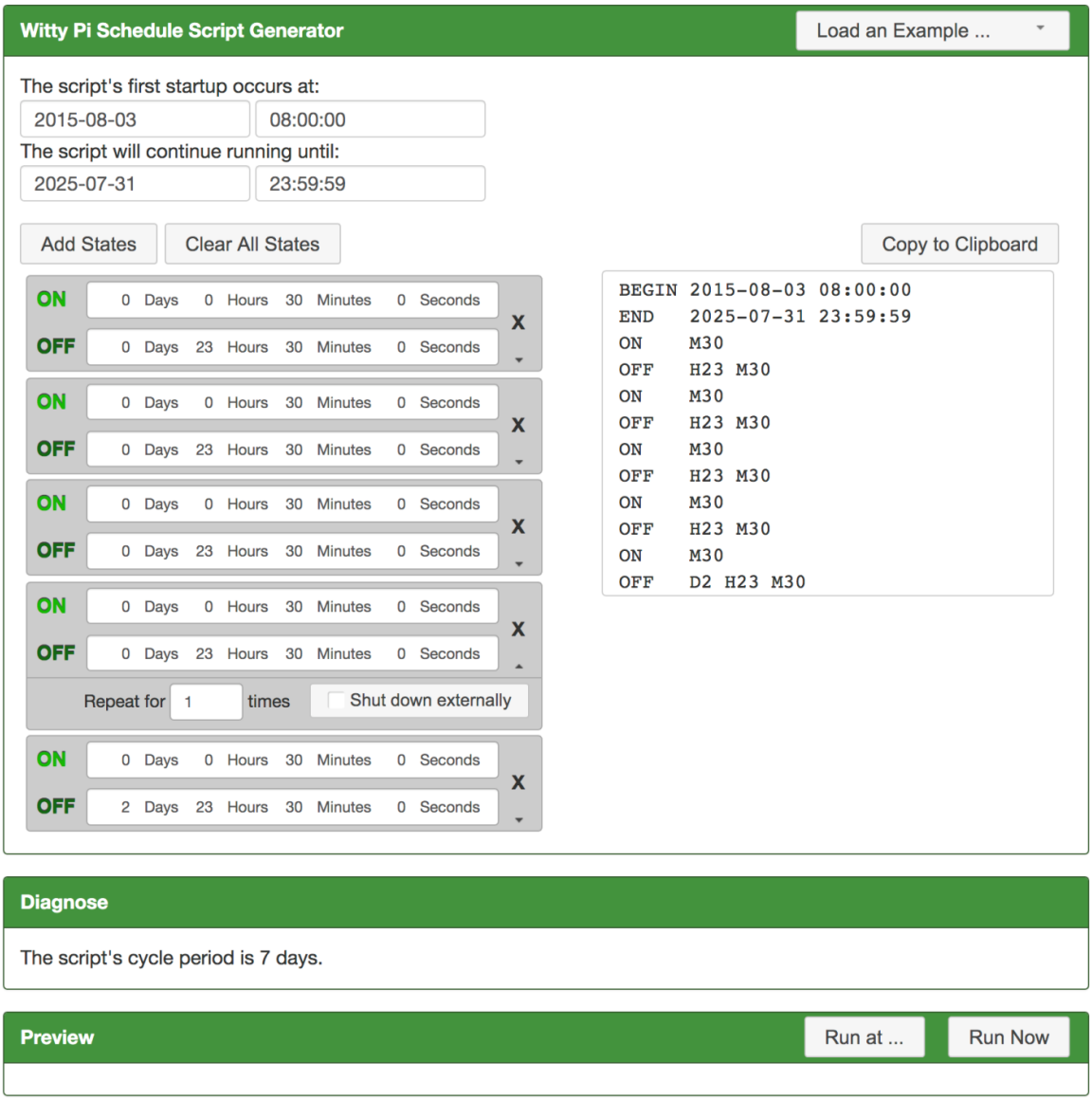
9.3 Using Schedule Script Generator

You can also use our web application to create your schedule script. Just simply open this URL in your web browser:

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

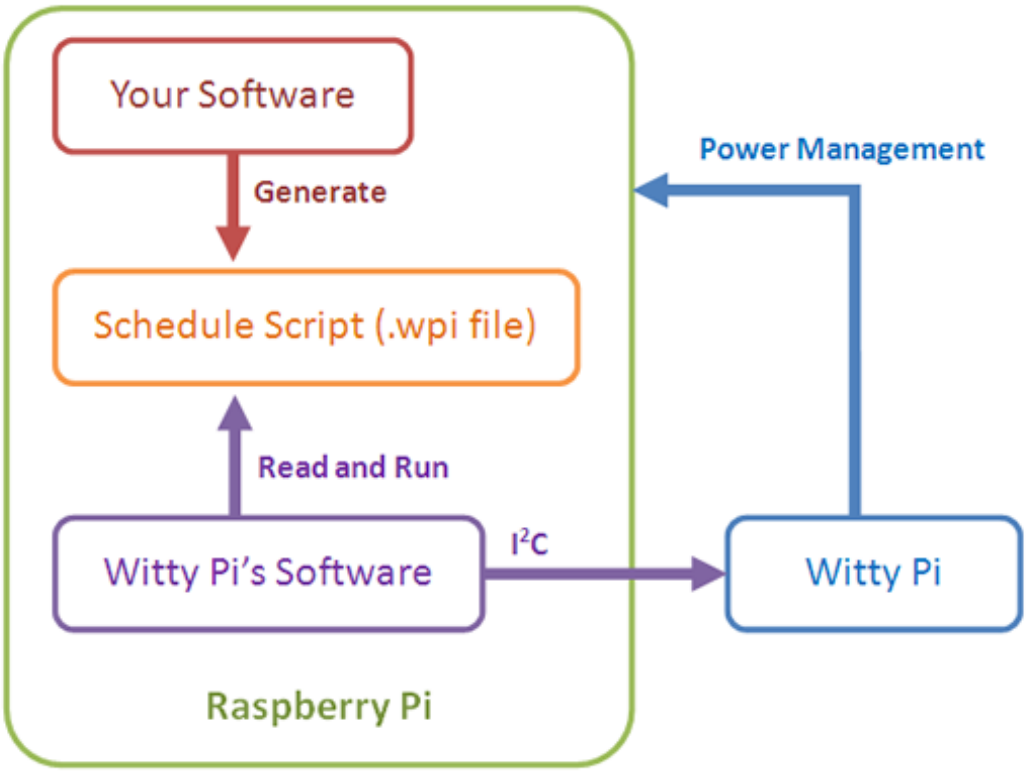
This web application allows you to visually create the schedule script, and it immediately generate the final code on the right.

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.



9.4 Advanced Usage of Schedule Script

Besides choosing the schedule script from wittyPi.sh, you can also use schedule script without the help from wittyPi.sh. Just copy the schedule script file to "~/wittypi/schedule.wpi" and then run "./runScript.sh" in the "~/wittypi" directory, the script will start to work. This allows you to use schedule script as an interface, to integrate other tools with Witty Pi together. For example, you can create your own tool to visually create a schedule script, or remotely generate the schedule script via a web interface.



10. Know More about the Realtime Clock

Witty Pi 4 uses PCF85063A realtime clock for time keeping. The realtime clock has been calibrated in factory, and can reach **±2ppm accuracy**. The workflow to calibrate the realtime clock can be found in chapter 8.2.3.3 in PCF85064A's datasheet.

10.1 CR2032 Battery and Time Keeping

Every package of Witty Pi 4 includes a CR2032 battery. You can put this battery into the battery holder on Witty Pi 4, so the realtime clock can still keep the correct time after you disconnect power supply from Witty Pi 4.

This CR2032 battery only powers the realtime clock chip (PCF85063), which only draws ~0.22μA current. Also, the realtime clock does not draw any current from the battery when external power supply is connected. As a result, you basically don't need to replace this battery because it can last for years.

If your Witty Pi 4 always have power supply connected, you can even omit this battery. The realtime clock will not lose the correct time when Raspberry Pi is off, because the realtime clock is still powered by the connected power supply. The CR2032 battery is helpful when you need to move the device to another site, or your device experiences power failure.

10.2 Alarms and Alarm Output

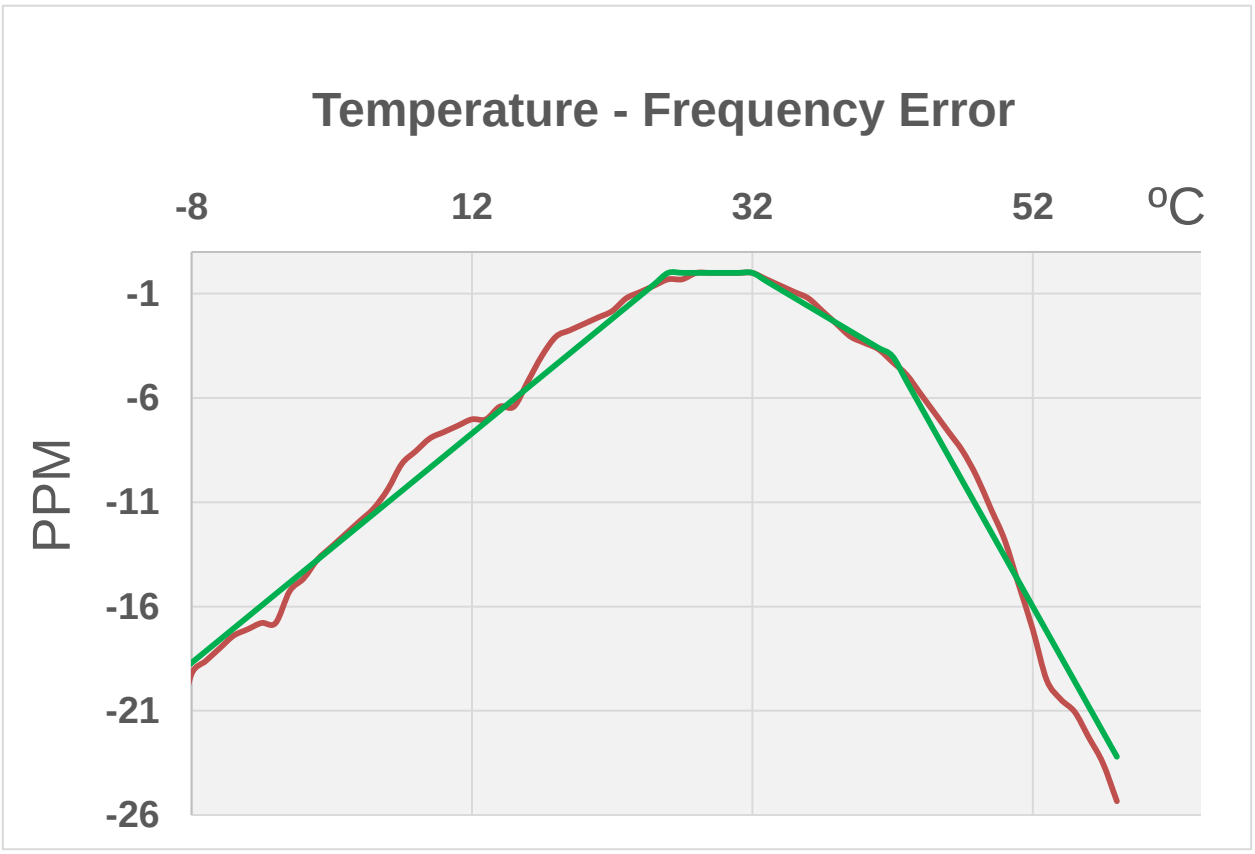
Witty Pi needs two alarms (one for scheduled startup and one for scheduled shutdown), but PCF85063A has only one. That's why we implemented two alarms in the firmware instead. This also brings an advantage that both alarms now can be accurate to seconds.

The ALM/ALARM in unpopulated header P3 is connected to the alarm output of realtime clock. This pin will stay in 3.3V level and will be pulled to 0V when an alarm occurs. This signal is for integration purpose, so other systems can know whether Witty Pi 4 has triggered an alarm.

Because the realtime clock itself only has one alarm, Witty Pi 4's firmware always copies the nearly overdue alarm data to the realtime clock, so both alarms can trigger the ALM/ALARM signal.

10.3 Temperature Compensation

In order to get even better accuracy, Witty Pi 4's firmware also makes temperature compensation for the crystal it uses. The temperature compensation bases on the fact that the actual frequency of the crystal changes according to the temperature. The frequency-temperature curve has been measured and the data are used for such compensation.



The orange curve is the actual measured temperature – frequency error curve. Witty Pi 4's firmware uses the green four-segment line to approximate the actual curve for temperature compensation.

11. Additional Interfaces

11.1 The Unpopulated 3-Pin Header (P2)

On the right edge of Witty Pi 4 board, there is an unpopulated 3-pin header, and you can solder a male or female connector here.

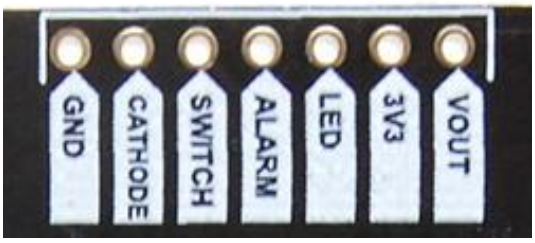
This header is for inputting voltage to power Witty Pi. It will be useful if you neither want to input 5V via the USB Type C connector, nor input higher voltage via the XH2.54 connector.

You can either input 5V between the “5V” and GND pins, or input higher voltage between the “VIN” and “GND” pins.

11.2 The Unpopulated 7-Pin Header on the Top (P3)

On the upper edge of Witty Pi 4 board, there is an unpopulated 7-pin header, and you can solder a male or female connector here.

(Front View)



(Back View)

This header breaks out some useful signals and is very helpful for extension and integration. The pins from left to right (at top view) are VOUT, 3V3, LED, ALARM, SWITCH, CATHODE and GND.

VOUT

This pin is actually connected to the +5V pin in Raspberry Pi’s GPIO header, which is also the output voltage of Witty Pi 4 board. You can also know whether your Raspberry Pi is in ON state by measuring this voltage.

Please notice that your **Raspberry Pi is powered by the voltage between VOUT and CATHODE**. The GND in this header is the ground for Witty Pi 4, but not the ground for Raspberry Pi.

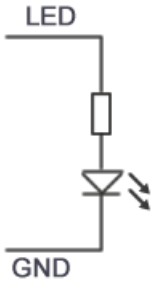
If there is another device need to get powered (by 5V) with Raspberry Pi together, you can connect it to this VOUT and CATHODE pins.

3V3

It is 3.3V voltage on Witty Pi 4 board that powers the micro controller, RTC and temperature sensor. It has nothing to do with the 3.3V pin in Raspberry Pi's GPIO header.

LED

It is connected to the anode of the white LED. You can use this pin to connect your own LED, but don't forget to put a 1K resistor in serial to limit the current.



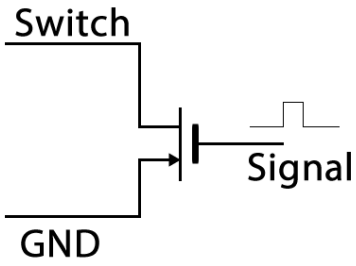
ALM / ALARM

It is the interrupt signal that generated by the RTC alarm. It is in 3.3V level and has HIGH state (3.3V) by default. If any alarm occurs (scheduled startup or shutdown), it goes to LOW state (0V), and this state will be cleared once Witty Pi 4's software detects and processes it.

SW / SWITCH

It is the signal line that connects to the switch (button) on Witty Pi 4. It is also directly connected to GPIO-4 in Raspberry Pi's GPIO header. If you want to connect your own (2-lead) switch, you may wire the two leads to SWITCH/GPIO-4 and GND pins.

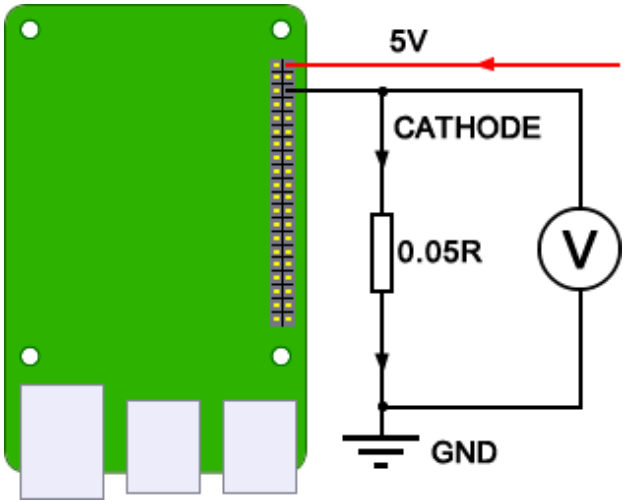
Alternatively, if you wish to trigger Witty Pi 4 with external signal, you can use a N-channel MOSFET to achieve this:



The signal should be a positive pulse. Processing a pulse will be equal to taping the switch once, so it will turn on your Raspberry Pi when it is off, or turn off your Raspberry Pi when it is on.

CATH / CATHOD

It is the cathode (ground) for Raspberry Pi. There is a 0.05 Ohm sampling resistor between CATHOD and GND, to measure the actual output current.



If you measure the voltage between cathode and ground as V_k, you can calculate the actual current with this formula:

$$I_{out} = V_k / 0.05 = 20 * V_k$$

GND

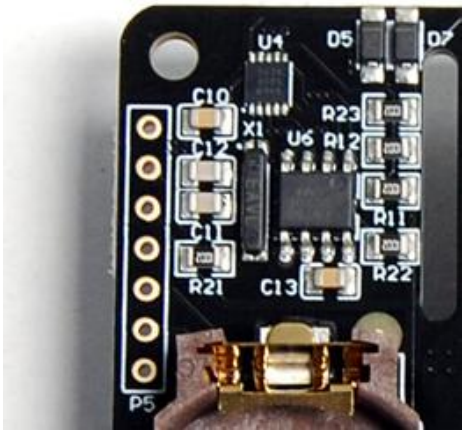
It is the ground of Witty Pi 4 board, and it directly connects to the GND wire of power supply.

11.3 The Unpopulated 2 x 3 Pin Headers (P4)

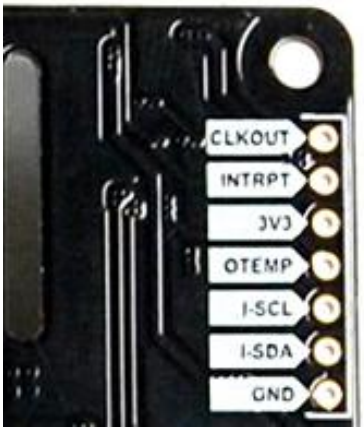
This is the In-Circuit Serial Programming (ICSP) header for uploading the firmware to the micro controller. You can find the same ICSP header on many Arduino boards. The details about how to use this header can be found in this tutorial.

11.4 The Unpopulated 7-Pin Header on the Left (P5)

On the left edge of Witty Pi 4 board, there is another unpopulated 7-pin header, where you can also solder a male or female connector.



(Front View)



(Back View)

This header breaks out the internal I2C bus and also some useful signals.

CLKOUT

This is the 32768Hz clock output signal from the RTC. This signal can be used to calibrate the RTC.

INTRPT

This is the interrupt signal from the RTC, and it is actually the same as the ALM/ALARM pin in another unpopulated header P3. It goes LOW when alarm is occurred.

3V3

It is 3.3V voltage on Witty Pi 4 board that powers the micro controller, RTC and temperature sensor. It has nothing to do with the 3.3V pin in Raspberry Pi's GPIO header. It is the same as 3V3 in P3.

OTEMP

This pin is at HIGH (3.3V) level by default. It goes to LOW (~0V) level when the current temperature exceeds the preset over-temperature threshold. It will go back to HIGH level when the current temperature drops under the below-temperature threshold.

I-SCL and I-SDA

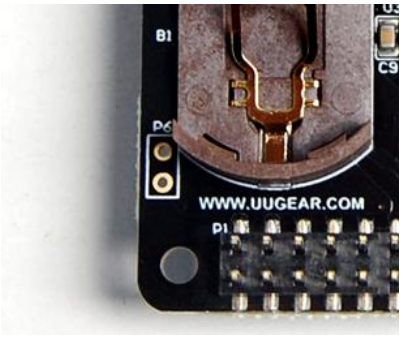
These are the SCL and SDA for internal I2C bus. The RTC and temperature sensor are directly connected to this internal I2C bus.

GND

It is the ground of Witty Pi 4 board, and it directly connects to the GND wire of power supply.

11.5 The Unpopulated 2-Pin Header (P6)

Usually you don’t need to worry about the CR2032 battery because it can last for years. However, if you want to use a different 3V power source, you can make use of this header. The two pins of this header directly connect to the two poles of CR2032 battery. If you are using a rechargeable CR2032 battery, you may also use this header to charge it.

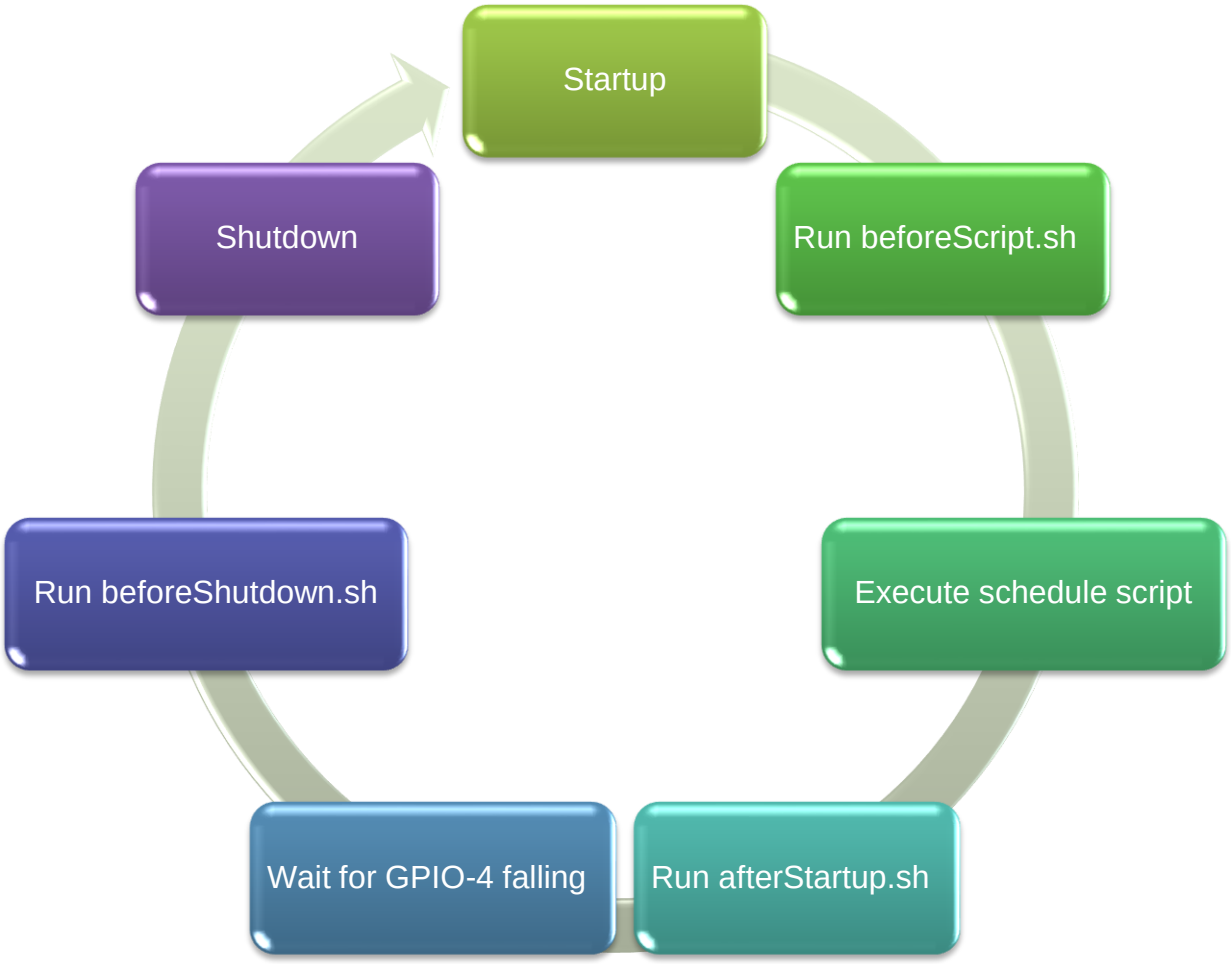


(Front View)

(Back View)

12. Integrate with Other Programs

Witty Pi's software is written in BASH, and it is very easy to modify and integrate with other programs. The daemon.sh script is registered to be executed after boot, and this registration was done by the installation script (using update-rc.d command). In the daemon.sh script, it will run beforeScript.sh before executing the schedule script (schedule.wpi), and run afterStartup.sh file before waiting for GPIO-4 falling, and also run the beforeShutdown.sh before actually shutdown Raspberry Pi.



If you want to run your own program before executing the schedule script (e.g. your program will generate a new schedule script file), you can put your program command(s) in **beforeScript.sh**.

If you want to run your own program after executing the schedule script (e.g. your program will use the scheduled date time for next shutdown), you can put your program command(s) in **afterStartup.sh**.

If you want to run your own program before shutting down your Raspberry Pi (e.g. your program will send out an email before shutdown), you can put your program command(s) in **beforeShutdown.sh**.

Please keep in mind the daemon.sh script will be executed by root user, which means it may not

have required path information for frequently used commands. As a result, you will need to use full path for all commands in beforeScript.sh, afterStartup.sh and beforeShutdown.sh. Below is an example:

```
while [ "$(/usr/bin/hostname -I)" = "" ]; do
  /usr/bin/sleep 1
done
/usr/bin/wget "https://www.uugear.com/" >/home/pi/wittypi/wget.log 2>&1
```

This code snippet waits for network and then query the webpage from UUGear's website. Please notice the full path for commands and files.

hostname → /usr/bin/hostname

sleep → /usr/bin/sleep

wget → /usr/bin/wget

wget.log → /home/pi/wget.log

If you are not sure about the full path of a command, you can use whereis command to find out.

```
pi@raspberrypi:~ $ whereis sleep
sleep: /usr/bin/sleep /usr/share/man/man1/sleep.1.gz /usr/share/man/man3/sleep.3.gz
```

All commands you put in beforeScript.sh, afterStartup.sh or beforeShutdown.sh will be executed in sequence. If one of the commands takes rather long time, the commands after it will be postponed accordingly.

If you don't want a command to block other commands, you can run it in the background by appending "&" at the end of that command, like this:

```
/usr/bin/wget "https://www.uugear.com/" >/home/pi/wittypi/wget.log 2>&1 &
```

13. Migrating from Witty Pi 3 to Witty Pi 4

If you have been using Witty Pi 3 before, migrating to Witty Pi 4 will be quite easy to you. Witty Pi 4 has the same shape factor like Witty Pi 3, and the locations of button and connectors are the same. **They are physically exchangeable**.

In Witty Pi 3, you cannot specify the "seconds" in scheduled shutdown time, and that was due to the limitation from RTC hardware. In Witty Pi 4, the alarms for startup and shutdown are implemented by the firmware, and **you can specify the "seconds" in both scheduled times**.

The "??" wildcards are no longer supported in Witty Pi 4. This is because implementing "??" wildcards need more resources than the MCU has. However, you can still use schedule script to achieve the same result. The table below shows how to use schedule to replace the "??" wildcards.

Wildcard Use Case	Explanation	Equivalent with Schedule Script
Startup: ?? 09:45:00 Shutdown: ?? 22:30:00	Turn on at 9:45, and turn off at 22:30 every day.	**BEGIN 2022-05-10 09:45:00** **END 2025-07-31 23:59:59** **ON H12 M45** **OFF H11 M15**
Startup: ?? 09:45:00 Shutdown: no plan	Turn on at 9:45 every day, and do not schedule the shutdown.	**BEGIN 2022-05-10 09:45:00** **END 2025-07-31 23:59:59** **ON M5 WAIT** **OFF H23 M55**
Startup: no plan Shutdown: ?? 22:30:00	Turn off at 22:30 every day, and do not schedule the startup.	**BEGIN 2022-05-10 22:25:00** **END 2025-07-31 23:59:59** **ON M5** **OFF H23 M55 WAIT**
Startup: ?? ??:40:00 Shutdown: ?? ??:55:00	Turn on at 40 minutes and turn off at 55 minutes every hour.	**BEGIN 2022-05-10 00:40:00** **END 2025-07-31 23:59:59** **ON M15** **OFF M45**
Startup: ?? ??:40:00 Shutdown: no plan	Turn on at 40 minutes every hour, and do not schedule the shutdown.	**BEGIN 2022-05-10 00:40:00** **END 2025-07-31 23:59:59** **ON M5 WAIT** **OFF M55**
Startup: no plan Shutdown: ?? ??:30:00	Turn off at 30 minutes every hour, and do not schedule the startup.	**BEGIN 2022-05-10 00:25:00** **END 2025-07-31 23:59:59** **ON M5** **OFF M55 WAIT**

If your system has integrated Witty Pi 3's software and you have called the functions in utilities.sh file. You will need to **compare the two versions of utilities.sh files** and make sure all functions you use are still there. We try not to change the function names, but there are some functions get added/removed.

If your system also directly accesses Witty Pi 3 via I2C interface, you will need to **change the I2C address and register indexes accordingly**. Witty Pi 3 exposes two I2C devices to the bus (address 0x68 and 0x69), while Witty Pi 4 exposes only one I2C devices to the bus (address 0x08). In Witty Pi 4, the I2C registers in RTC and temperature sensor are mapped to virtual I2C registers in single device. Please refer the "What I²C Registers are provided by Witty Pi 4" section for more information.

14. Witty Pi Log Files

In the directory that you install your Witty Pi software, you can find two log files: **schedule.log** and **wittyPi.log**. If you need our help for solving a problem, please kindly put the log files in email attachment too. This will help us to help you better.

The “schedule.log” file contains the history of schedule script executions. In this log file you can see how the next shutdown and startup were scheduled. If you saw unexpected schedule script behavior, this log should be the first file to check.

The “wittyPi.log” file records the history of all Witty Pi activities. If you think your Witty Pi doesn’t behave correctly, this log file might provide more information for troubleshooting.

The time-stamp in the log file is always the system time, instead of the RTC time. During the booting, the first log written by Witty Pi’s software is “Witty Pi daemon (vX.XX) is started.”, but its time-stamp is not shown. You will see the correct time after the RTC time has been written to the system. For example:

```
[xxxx-xx-xx xx:xx:xx] Witty Pi daemon (v4.00) is started.
……
[xxxx-xx-xx xx:xx:xx] Synchronizing time between system and Witty Pi...
[xxxx-xx-xx xx:xx:xx] Writing RTC time to system...
[2022-05-09 14:05:13] Done :-)
```

The correct time-stamp firstly appears in the line with text “Done :-)”, so you know this startup happened at 14:05.

When the booting is done, the last log written by Witty Pi software is “Pending for incoming shutdown command…”.

Again, if you need our help for solving a problem, please kindly put the log files in email attachment too. In the majority of cases, they will help us a lot to help you better.

15. Frequently Asked Questions (FAQ)

15.1 What I2C address is used by Witty Pi 4? Can I change it?

Witty Pi 4 implements its own I2C master to communicate with realtime clock and temperature sensor, and only exposes one I2C slave device to Raspberry Pi, which has an I^{2}C address: **0x08**. This address is configurable and you can change it if you want.

If you have Witty Pi 4 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:          -- -- -- -- -- 08 -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

If you want to change the I^{2}C address used by Witty Pi 4, you can change the value of I^{2}C register at position #16. For example, if you want to change the I^{2}C address to 0x35, you can run:

```
pi@raspberrypi:~ $ i2cset -y 1 8 16 0x35
```

After that you also need to modify the "utilities.sh" file (in wittyPi directory). Find the I2C_MC_ADDRESS variable and change its value from 0x08 to 0x35.

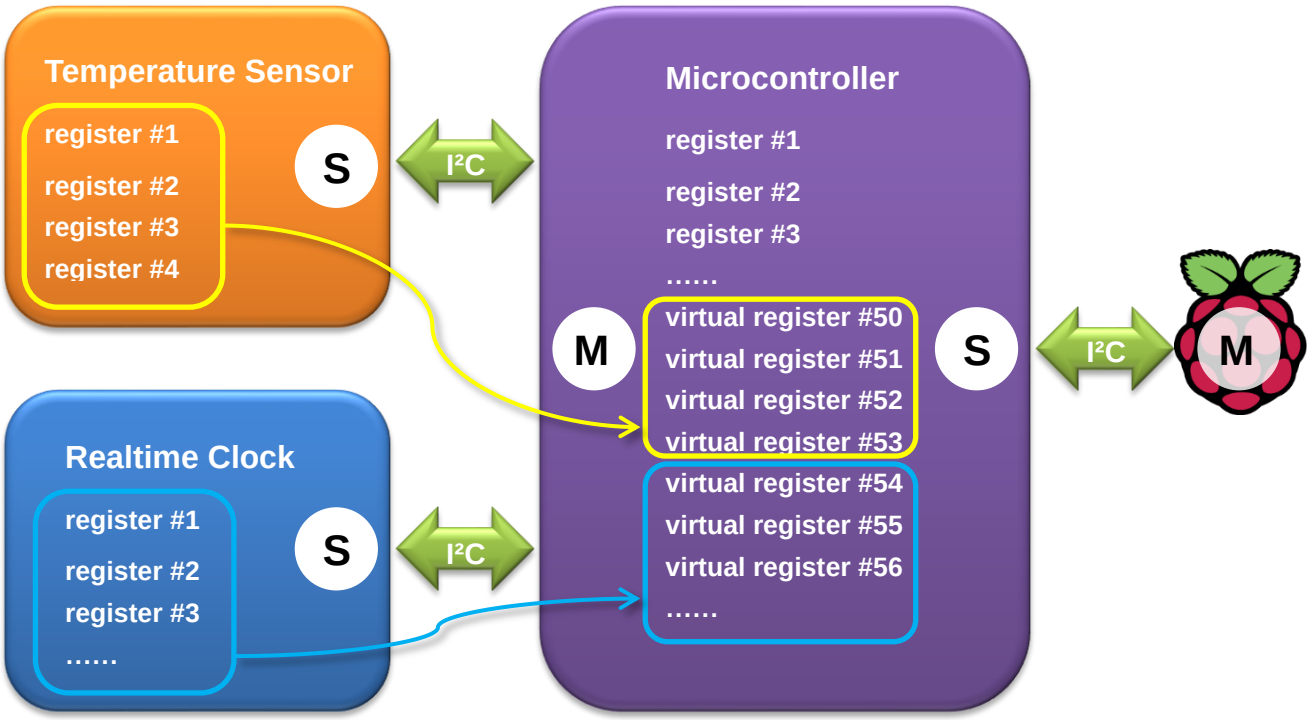
In order to access Witty Pi 4 in UWI, you also need to modify the "uwi/wittypi4/api.sh" file. Find "if ["$value" == "08"]; then" and replace the "08" to "35".

The final step is to shut down your Raspberry Pi, fully disconnect the power supply of your Witty Pi 4, and then reconnect the power supply. Then the MCU will start working with new address.

15.2 What I²C Registers are provided by Witty Pi 4?

The micro controller on Witty Pi 4 works as an I²C slave and Raspberry Pi can read/write its registers via I²C interface. Witty Pi's software configures Witty Pi 4 by setting the I²C register accordingly.

The micro controller also implements an I²C master to access the realtime clock and temperature sensor via internal I²C bus. The I²C registers in realtime clock and temperature sensor are all mapped as virtual I²C registers in Witty Pi 4's I²C slave device, so Raspberry Pi can also access them (not always necessary though).



The temperature sensor contains 4 I²C registers and they are mapped to virtual registers #50~#53.

The realtime clock contains 18 I²C registers and they are mapped to virtual registers #54~#71.

In Raspberry Pi's view, Witty Pi 4 provides 3 kinds of I²C registers:

- Read-only I²C registers
- Normal I²C registers
- Virtual I²C registers

The table below shows the registers provided by Witty Pi 4. 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	Defa	Accessible

			ult	
0	Firmware ID	--	0x26	Read-only
1	Integer part for input voltage	0~255	0	Read-only
2	Decimal part (multiple 100 times) for input voltage	0~99	0	Read-only
3	Integer part for output voltage	0~255	0	Read-only
4	Decimal part (multiple 100 times) for output voltage	0~99	0	Read-only
5	Integer part for output current	0~255	0	Read-only
6	Decimal part (multiple 100 times) for output current	0~99	0	Read-only
7	Power mode: • Power via LDO regulator = 1 • Input 5V via USB Type C = 0	1 or 0	1	Read-only
8	A flag indicates whether the previous shutdown was due to low voltage: • Yes = 1 • No = 0	1 or 0	0	Read-only
9	A flag indicates whether ALARM1 (startup) has been triggered. • Yes = 1 • No = 0	1 or 0	0	Read-only
10	A flag indicates whether ALARM2 (shutdown) has been triggered. • Yes = 1 • No = 0	1 or 0	0	Read-only
11	The reason code for latest action. • 0 = N/A	0~8	0	Read-only

	• 1 = ALARM1 • 2 = ALARM2 • 3 = Button is clicked • 4 = Input voltage too low • 5 = Input voltage is restored • 6 = over temperature • 7 = below temperature • 8 = ALARM1 delayed			
12	The firmware revision	1~255	1	Read-only
13	Reserved for future usage #1	0~255	0	Read-only
14	Reserved for future usage #2	0~255	0	Read-only
15	Reserved for future usage #3	0~255	0	Read-only
16	I2C slave address: default=0x08	0x08 ~0x77	0x08	Read & Write
17	State when power connected: • Default On = 1 • Default Off = 0	1 or 0	0	Read & Write
18	Pulse interval in seconds, when Raspberry Pi is off. This parameter affects the white LED blinking and dummy load.	0~255	4	Read & Write
19	Low voltage threshold (multiple 10 times, 255=disabled)	0~255	255	Read & Write
20	LED lights up duration. 0 if white LED should not blink. The bigger value, the longer time to light up the white LED	0~255	100	Read & Write
21	The delay (multiple 10) before power cut: default=50	0~255	50	Read & Write

	(5 seconds)			
22	Recovery voltage threshold (multiple 10, 255=disabled)	0~255	255	Read & Write
23	Dummy load duration. 0 if dummy load is off. The bigger value, the longer time for dummy load to draw current	0~255	0	Read & Write
24	Adjustment for measured input voltage (multiple 100)	-127~127	20	Read &Write
25	Adjustment for measured output voltage (multiple 100)	-127 ~127	20	Read & Write
26	Adjustment for measured output current (multiple 100)	-127 ~127	0	Read & Write
27	Second of ALARM1 (startup), with BCD format	0~59	0	Read & Write
28	Minute of ALARM1 (startup), with BCD format	0~59	0	Read & Write
29	Hour of ALARM1 (startup), with BCD format	0~23	0	Read & Write
30	Day of ALARM1 (startup), with BCD format	1~31	1	Read & Write
31	Weekday of ALARM1 (startup), with BCD format	0~6	0	Read & Write
32	Second of ALARM2 (shutdown), with BCD format	0~59	0	Read & Write
33	Minute of ALARM2 (shutdown), with BCD format	0~59	0	Read & Write
34	Hour of ALARM2 (shutdown), with BCD format	0~23	0	Read & Write
35	Day of ALARM2 (shutdown), with BCD format	1~31	1	Read & Write
36	Weekday of ALARM2 (shutdown), with BCD format	0~6	0	Read & Write
37	Standard value for RTC offset	-127~127	0	Read & Write
38	Turn on/off temperature compensation.	0 or 1	1	Read & Write

	• ON = 1 • OFF = 0			
39	A flag that indicates ALARM1 (startup) is triggered and not processed.	0 or 1	0	Read & Write
40	A flag that indicates ALARM2 (shutdown) is triggered and not processed.	0 or 1	0	Read & Write
41	Set 1 to ignore I2C_POWER_MODE for low voltage shutdown and recovery.	0 or 1	0	Read & Write
42	Set 1 to ignore I2C_LV_SHUTDOWN for low voltage shutdown and recovery.	0 or 1	0	Read & Write
43	Below-temperature action: • 0 = do nothing • 1 = shutdown • 2 = startup	0~2	0	Read & Write
44	Below-temperature threshold in °C	-30~80	75	Read & Write
45	Over-temperature action: • 0 = do nothing • 1 = shutdown • 2 = startup	0~2	0	Read & Write
46	Over-temperature threshold in °C	-30~80	80	Read & Write
47	Reserved for future usage #4	0~255	0	Read & Write
48	Reserved for future usage #5	0~255	0	Read & Write
49	Reserved for future usage #6	0~255	0	Read & Write
50	LM75B: temperature register	(2 bytes)	?	Read-only
51	LM75B: configuration register	0~255	0	Read & Write

52	LM75B: hysteresis temperature register	(2 bytes)	75	Read & Write
53	LM75B: overtemperature register	(2 bytes)	80	Read & Write
54	PCF85063: Control_1 register	0~255	0	Read & Write
55	PCF85063: Control_2 register	0~255	0	Read & Write
56	PCF85063: Offset register	0~255	0	Read & Write
57	PCF85063: RAM_byte register	0~255	0	Read & Write
58	PCF85063: Seconds register	0~59	0	Read & Write
59	PCF85063: Minutes register	0~59	0	Read & Write
60	PCF85063: Hours register	0~23	0	Read & Write
61	PCF85063: Days register	1~31	1	Read & Write
62	PCF85063: Weekdays register	0~6	0	Read & Write
63	PCF85063: Months register	1~12	1	Read & Write
64	PCF85063: Years register	0~99	0	Read & Write
65	PCF85063: Second_alarm register	0~59	0	Read & Write
66	PCF85063: Minute_alarm register	0~59	0	Read & Write
67	PCF85063: Hour_alarm register	0~23	0	Read & Write
68	PCF85063: Day_alarm register	1~31	1	Read & Write
69	PCF85063: Weekday_alarm register	0~6	0	Read & Write
70	PCF85063: Timer_value register	0~255	0	Read & Write
71	PCF85063: Timer_mode register	0~255	0	Read & Write

Below is an example to read the register with index 7, to know the current power mode (0x00 means input 5V via the USB Type C connector):

```
pi@raspberrypi:~ $ i2cget -y 1 0x08 7
0x00
```

And below is an example to write the register with index 18, to set the pulsing interval to 1 second:

```
pi@raspberrypi:~ $ i2cset -y 1 0x08 18 1
```

Remarks: although the register with index 16 is writable, **changing the I2C address is more than writing an I²C register**. You will need to modify the software accordingly, and reconnect the power supply to make it work. You can find more details in the "What I2C address is used by Witty Pi 4? Can I change it?" section.

15.3 What GPIO Pins Are Used by Witty Pi4?

The GPIO pins used by Witty Pi 4 are marked with **green** color in the table below.

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

As you can see, Witty Pi 4 uses **GPIO-4**, **GPIO-17**, **GPIO-2(SDA 1)** and **GPIO-3(SCL 1).**

Witty Pi 4 doesn't actually use the TXD pin, but it will monitor its voltage. The TXD pin is supposed to be HIGH when the system is on, and should go LOW after system has been shut down. If you connect some other devices that also use the TXD pin, please make sure they don't change this default behavior, otherwise Witty Pi 4 doesn't know when the system is off, and cannot fully cut the power.

GPIO-2 and GPIO-3 are for I^{2}C communication between Raspberry Pi and the MCU (ATtiny841). I^{2}C devices are identified by I^{2}C address, and they can share the I^{2}C pins as long as they have different I^{2}C addresses.

15.4 Is Witty Pi 4 Compatible with "Other Hardware"?

We have got a lot of emails asking a question like this, with the "Other Hardware" replaced by different kinds of hardware.

Please understand that we might not have the hardware you refer, and even if we have, it is difficult for us to make tests on all those hardware with Witty Pi 4. Fortunately, it is not that difficult to figure out the answer by yourself. Basically, you just need to consider the I^{2}C address and GPIO pins used by the "Other Hardware".

If the "Other Hardware" also uses 0x08 I^{2}C address, you will need to change the I^{2}C address used by Witty Pi's micro controller, and change the software accordingly (details here).

If the "Other Hardware" doesn't use any GPIO pin that used by Witty Pi, then it should be compatible with Witty Pi.

If you have no idea which I^{2}C address (if applicable) or GPIO pins are used by the "Other Hardware", please contact its developer, as they certainly know their hardware and can provide you accurate information about it.

15.5 Witty Pi 4 does not boot?

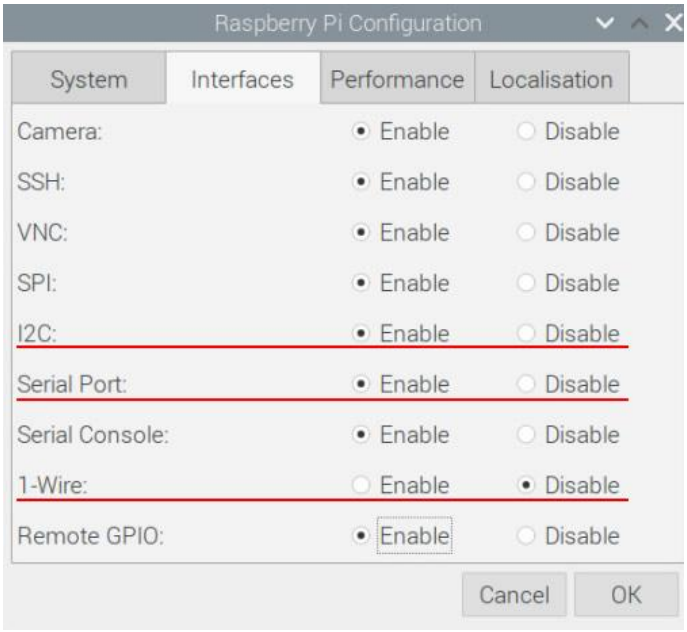
Some customers meet the situation that Witty Pi 4 only boot Raspberry Pi for a few seconds, and then shutdown Raspberry Pi or cut the power directly.

Remarks: you may need to disconnect Witty Pi 4 from your Raspberry Pi, and power on Raspberry Pi only for troubleshooting.

Before making further troubleshooting, please first check these interfaces are properly configured:

- I2C interface should be enabled
- Serial Port should be enabled
- 1-Wire interface should be disabled if you don't need it, or you need to assign a GPIO pin other than GPIO-4 to it.

If your Raspbian (Raspberry Pi OS) has GUI installed, you can review/configure these interfaces via Raspberry Pi Configuration:



Also, you can check all GPIO pin states with the command below. You need to run it in the directory that has software installed (usually “wittypi”).

```
pi@raspberrypi:~/wittypi $ bash -c '. gpio-util.sh; gpio readall'
```

The good GPIO states are shown below (please mind those highlighted in green). If you see something different, then it might be the culprit that causes the problem.

```
pi@raspberrypi ~/wittyPi $ gpio readall
 +-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 1 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
```

There are some possible reasons that can cause this kind of problem:

1. You are using NOOBS

NOOBS is not always causing this kind of problem, but the chance does exist. If you installed multiple operating systems into your Pi, the chance that NOOBS causes this problem will increase. If possible, please try not to use NOOBS. If you already use NOOBS and still try to make it work, you may set a big power cut delay value (e.g. 25 seconds). If it doesn't help, you will need to install the OS without using NOOBS.

2. 1-Wire interface is enabled on GPIO-4, which conflicts with Witty Pi

If you have 1-Wire interface enabled, by default it will use GPIO-4. Because Witty Pi uses GPIO-4 pin to receive shutdown command, this will bring confliction and Witty Pi will shut down your Raspberry Pi after every boot. You won't even have chance to login the system.

In order avoid this problem, 1-Wire interface should be disabled, or be assigned to a different GPIO pin before installation of Witty Pi's software. If the problem already happens, you need to take out the micro SD card from Raspberry Pi and access its file system via an SD card reader.

You need to edit the config.txt file in the "boot" volume to change the GPIO pin used by 1-Wire interface, or you can disable 1-Wire interface if you don't need it for now. You need to find something like "dtoverlay=w1-gpio" in the file.

If you want 1-Wire to use GPIO-18, just replace "dtoverlay=w1-gpio" with:

```
dtoverlay=w1-gpio,gpiopin=18
```

If you want to disable 1-Wire interface, just comment it out:

```
#dtoverlay=w1-gpio
```

Save the file and eject your micro SD card, and put it back to your Raspberry Pi. Now your Raspberry Pi should be able to boot normally.

3. Serial Port is not properly configured

The serial port (UART0) should be enabled, and it should be on GPIO-14 and GPIO-15 (BCM naming). The TXD pin in serial port will be monitored by Witty Pi and it goes to LOW when the system has been shut down. If the serial port is not properly configured, the TXD pin doesn't have proper default state and Witty Pi may mistakenly think the system is off and then cut the power after some delay.

4. GPIO-4 pin doesn't reach a stable status in given time

After the system is on, GPIO-4 pin should be in input mode and gets internally pulled up. However, during the startup the GPIO-4 pin could be unstable. In the **daemon.sh** script, the GPIO-4 listener will be started once the pin state hasn't changed for 5 seconds. Once the GPIO-4 listener is started, any action that pulls down GPIO-4 will be regarded as a shutdown command. So if GPIO-4 pin doesn't really get stable after that given 5 seconds, Witty Pi will take it as a shutdown command, **lights up the white LED** and then shutdown the system.

There are many factors that might cause the GPIO pin unstable, and **the most common one is the power supply**. If your power supply is not strong enough, during the booting its voltage may drop from time to time, which may also make the GPIO pin voltage drop, and trigger Witty Pi's software to shut down your Raspberry Pi.

If it is the case, you can try to delay the starting of GPIO-4 listener, and in the majority of cases it will help. You can modify the "**daemon.sh**" script, in line 94:

```
while [ $counter –lt 5 ]; do
```

Try to change the number 5 to 25. The bigger number you use, the later the GPIO-4 listener will be started.

This modification may workaround the problem. If it doesn't, it means your GPIO-4 is really pulled down (by software or hardware), and you can confirm that by measuring the voltage on GPIO-4 with a multimeter. By default, the GPIO-4 should be internally pulled up. If it gets pulled down, try to find out who does this and don't let it do this again.

15.6 Why Raspberry Pi Immediately Turns On after Shutdown?

When ALARM1 (startup) occurs, Witty Pi's firmware will check Raspberry Pi's status. If Raspberry Pi is off, Witty Pi will turn it on; If Raspberry Pi is still powered, Witty Pi's firmware will cache this startup request and will turn on Raspberry Pi just after it gets power off.

This scenario may happen when your Raspberry Pi's TXD pin does not go LOW after system has been shut down. The reason could be some other hardware strongly pulls up TXD pin, or the system crashed during shutdown. When ALARM1 (startup) occurs, your Raspberry Pi is still powered because the turning off process is still not finished.

When this happens, you can find this message in the wittyPi.log file:

System starts up because of the scheduled startup got delayed. Maybe the scheduled startup was due when Pi was running, or Pi had been shut down but TXD stayed HIGH to prevent the power cut.

This will not be a problem as long as the TXD pin can eventually go LOW after shutdown and the device can continue the ON/OFF sequence.

16. Revision History

Revision	Date	Description
1.00	2022.06.08	Initial revision