UUGear

Witty Pi 3

Realtime Clock and Power Management for Raspberry Pi

User Manual (revision 1.06)

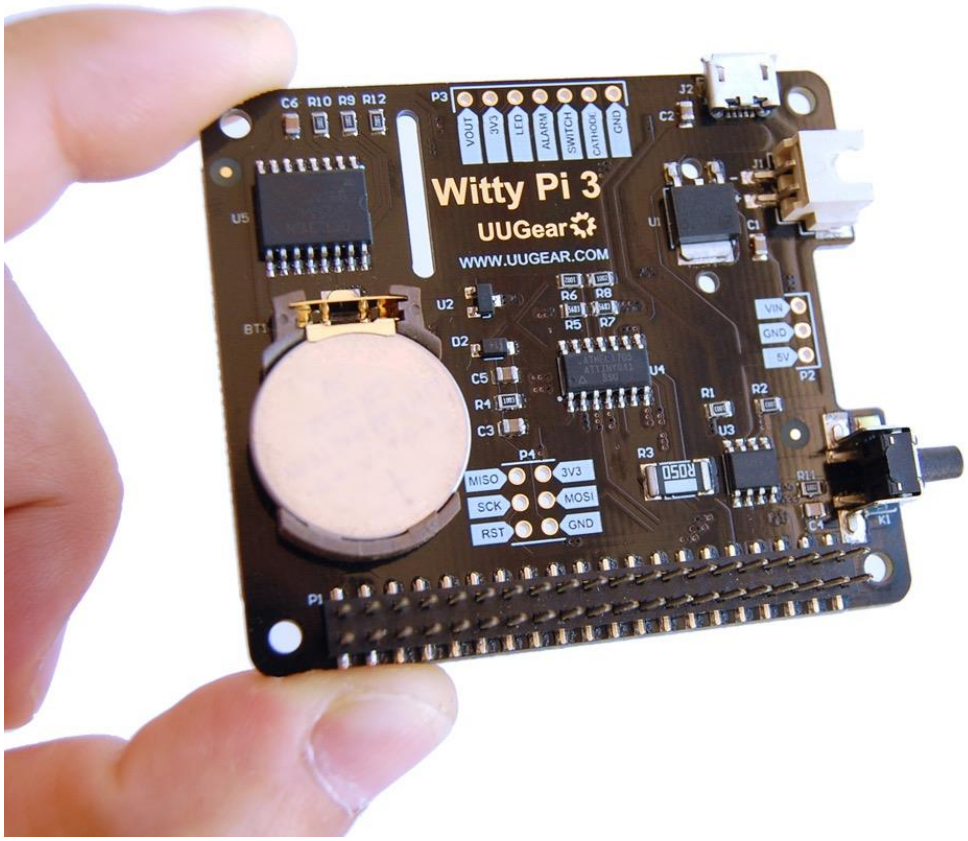


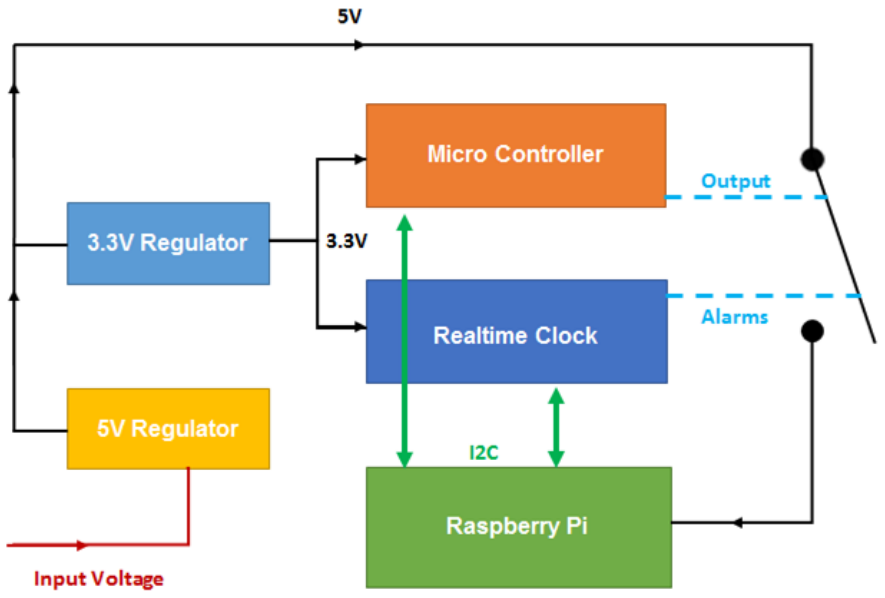
Table of Content

What is Witty Pi?

Witty Pi is small electronic circuit board that can add realtime clock and power management to your Raspberry Pi. Witty Pi 3 is the third generation of Witty Pi. After installing Witty Pi 3 on your Raspberry Pi, you get some amazing new features:

- You can power your Raspberry Pi with higher voltage.
- You can gracefully turn on/off Raspberry Pi with single tap on the switch.
- After shutdown, Raspberry Pi and all its USB peripherals' power are fully cut.
- Raspberry Pi knows the correct time, even without accessing the Internet.
- Raspberry Pi knows the temperature thanks to the sensor in RTC chip.
- You can schedule the startup/shutdown of your Raspberry Pi.
- You can even write a script to define complex ON/OFF sequence.
- Shutdown Raspberry Pi when input voltage is lower than pre-set value.
- Turn on Raspberry Pi when input voltage raise to pre-set value.
- When the OS loses response, you can long hold the switch to force power cut.

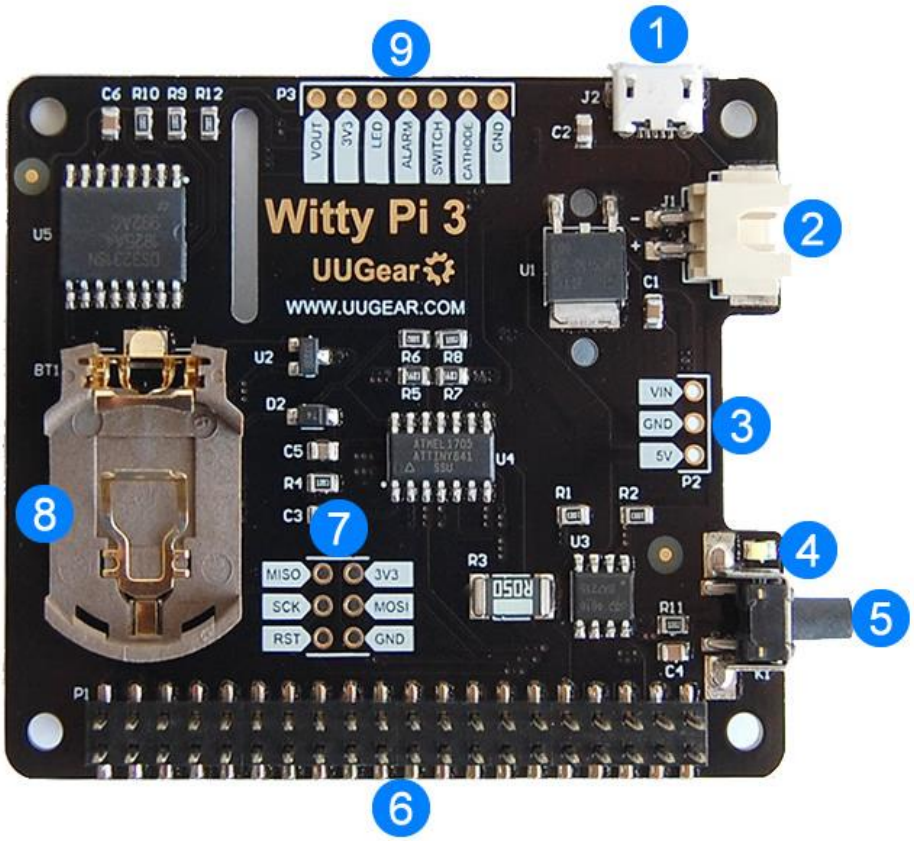
Witty Pi 3 supports all Raspberry Pi models that has the 40-pin GPIO header, including A+, B+, 2B, Zero, Zero W, 3B, 3B+, 3A+ and 4B.



Comparing to Witty Pi 2, Witty Pi 3 uses the same RTC chip (DS3231SN) and introduces a new micro-controller (ATtiny841), which allows it to behave smarter and handle more complex situations. Thanks to the new on-board LDO voltage regulator (LM29150), Witty Pi 3 can accept DC voltage higher than 5V (up to 26V, depends on the load and heat sink).

Witty Pi 3 doesn't have any jumper, and all configurations are done via I2C interface. Witty Pi's software can help you to make configuration very quickly.

The picture below shows how is Witty Pi 3 look like:



1) Micro USB connector as DC 5V power input
2) XH2.54 connector as higher voltage DC power input
3) Unpopulated 3-pin header/connector for VIN, GND and 5V
4) White LED as indicator
5) On/off switch
6) 2x20 pin stacking header (connects to Raspberry Pi)
7) Unpopulated 2x3 pin ICSP header for uploading firmware

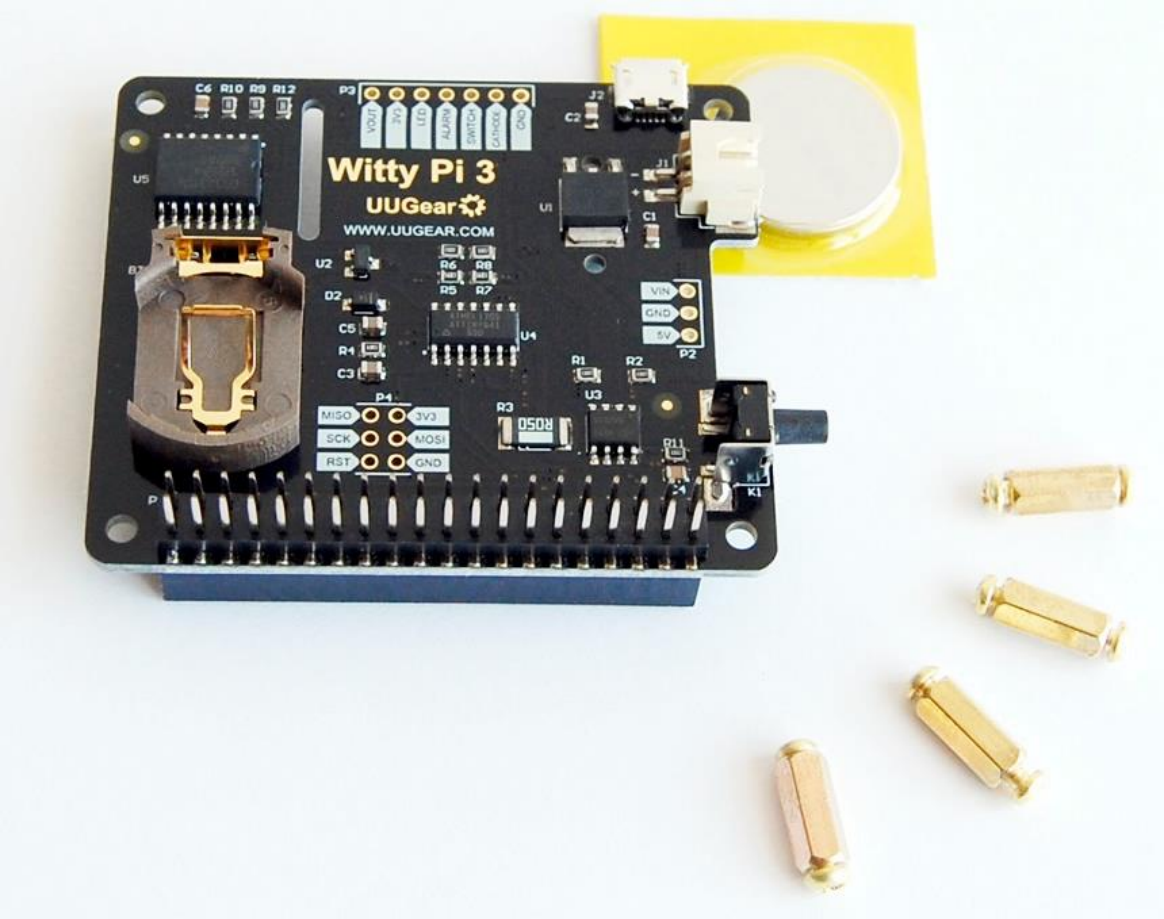
8) Battery holder for CR2032 button cell lithium battery

9) Unpopulated 7-pin header for extension or integration

What is in the Package?

Each Witty Pi 3 package contains:

- Witty Pi 3 board x 1
- CR2032 battery x 1
- M2.5 x 11mm Copper Standoff x 4
- M2.5 screws x 8



Witty Pi 3 Specifications

Dimension:	65mm x 56mm x 19mm
Weight	20g (net weight without battery)
Micro Controller	ATtiny841(datasheet)
Realtime Clock	DS3231SN (datasheet)
LDO Voltage Regulator	LM29150 (datasheet)
Battery	CR2032 (for time keeping)
Power In	DC 5V (via micro USB) or DC5.3V~26V (via XH2.54 connector, better no higher than 8V without additional heat sink)
Output Current	Up to 3A for Raspberry Pi and its peripherals if input via micro USB port. Up to 2A if input via XH2.54 connector.
Standby Current	~ 1mA
Operating Temperature	-30℃~80℃ (-22°F~176°F)
Storage Temperature	-40℃~85℃ (-40°F~185°F)
Humidity	0~80%RH, no condensing

Software Installation

It is recommended to install the software for Witty Pi 3 before physically mount Witty Pi 3 on your Raspberry Pi.

Before installing the software, please check whether you have 1-Wire interface enabled. This is important because 1-Wire interface uses GPIO-4 by default, which will conflict with Witty Pi. If you need to use 1-Wire interface, please assign it to a different GPIO pin.

You can assign different GPIO pin to 1-Wire interface in /boot/config.txt file, find the "dtoverlay=w1-gpio" text and replace it with:

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

If you don't need 1-Wire interface for now, you can disable it:

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

If you have 1-Wire interface enabled on GPIO-4 and installed Witty Pi's software, you may not be able to login to your Raspberry Pi because it always shuts itself down before you get the chance to login. To solve this problem, you will need to take out the micro-SD card on your Raspberry Pi, and access its file system via a card reader. You need to edit the config.txt file (as mentioned above) 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. After saving the file and put the micro SD card back to Raspberry Pi, you should be able to boot it again.

In order to install the software, you will need to have your Raspberry Pi connected to the Internet. The installation will be very simple if you run our installing script.

First step is to run this command in your home directory:

```
pi@raspberrypi ~ $ wget http://www.uugear.com/repo/WittyPi3/install.sh
```

If your Raspberry Pi has internet connection, it will immediately download the script from our website, and you will then see the "install.sh" script in your home directory. Then you just need to run it with sudo as follows:

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

Please notice that **sudo is necessary to run this script**. This script will automatically do these tasks in sequence:

1. Enable I2C on your Raspberry Pi
2. Install i2c-tools, if it is not installed yet
3. Configure Bluetooth to use mini-UART (Raspberry Pi 3 only)
4. Install wiringPi, if it is not installed yet (Raspberry Pi 4 needs version 2.52+)
5. Install Witty Pi programs, if they are not installed yet

You can also manually install these packages and make those configurations, if you prefer to. After the installation, please remember to **reboot your Raspberry Pi**, so the Realtime clock I2C hardware will be loaded correctly.

You will then see a new "wittypi" directory, and it contains 6 runnable files… viz.:

```
pi@raspberrypi ~ $ cd wittypi

pi@raspberrypi ~ /wittypi $ ls

afterStartup.sh      daemon.sh    runScript.sh    schedules      utilities.sh    wittyPi.sh
beforeShutdown.sh    init.sh      schedule.log    syncTime.sh    wittyPi.log
```

Although the **daemon.sh** is runnable, you should not run it manually. The installing script has registered it into /etc/init.d and it will run automatically after the start up.

The **syncTime.sh** script is not supposed to be manually run either; it will run automatically after the start up. It will copy the time from Raspberry Pi system to RTC if you have never set the RTC time before. If RTC has the correct time and your Raspberry Pi has the wrong time – perhaps because of the lack of Internet connection, it will copy the RTC time to your Raspberry Pi system.

The **runScript.sh** script is the one who takes charge of the schedule script running. Usually you don't need to run it manually, as it will be executed after the system is up. If there is a schedule script in use, it will schedule the next shutdown and next startup, according to the schedule script.

The **wittyPi.sh** is the software that allows you to configure your Witty Pi interactively. You can use it to copy time between Realtime clock and the system, and schedule the time for auto shutdown and/or startup. Please see the "Software Usage" chapter for more information.

The **afterStartup.sh** script will be run automatically after system is up. You can execute your own commands here if you wish them to be run after boot.

The **beforeShutdown.sh** script will be run automatically before system gets shut down by Witty Pi 3. You can also place your commands here. Please notice that, if the system is not shut down by Witty Pi 3 (tapping button, schedule shutdown or auto-shutdown due to low input voltage), this script will not be executed.

Now the software has been installed and you will need to physically mount/install Witty Pi 3 on your Raspberry Pi and **connect power supply to Witty Pi 3 only**.

Software Update/Uninstallation

If you want to update the software to newer version, you don't have to uninstall it first. Just remove or rename your "wittypi" directory and repeat the installing process, then you are all set.

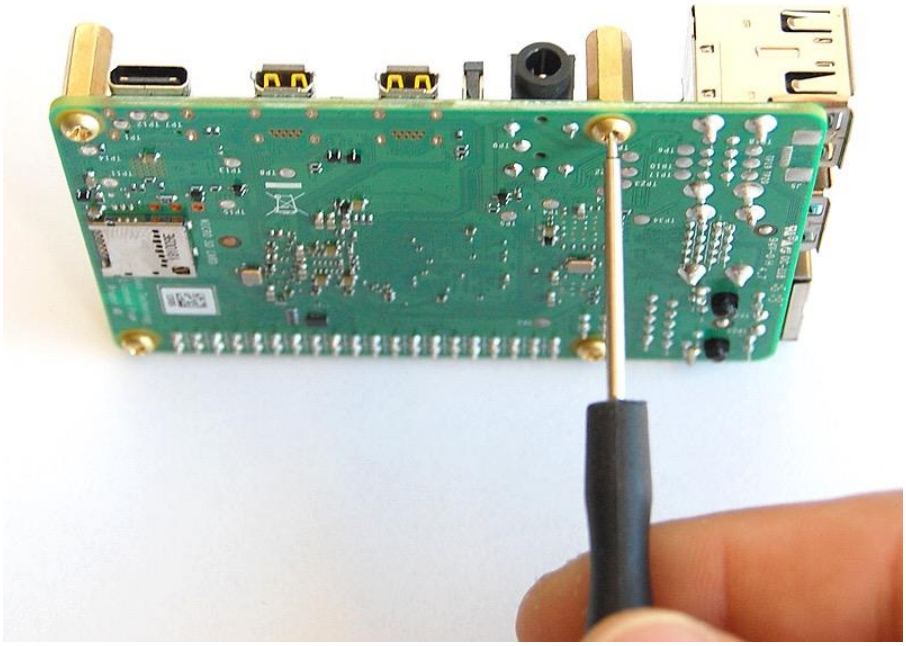
```
pi@raspberrypi ~ $ mv wittypi wittypi.bak
pi@raspberrypi ~ $ wget http://www.uugear.com/repo/WittyPi3/install.sh
pi@raspberrypi ~ $ sudo sh install.sh
```

If you prefer to completely remove the software, besides removing the "wittypi" directory, you should also remove the "/etc/init.d/wittypi" file. There are some dependencies (such as wiringPi, i2c-tools etc.), which may be installed during the software installation. In the majority of cases you don't have to remove them, but if you wish to you can check the content of "install.sh" script and do the reverse.

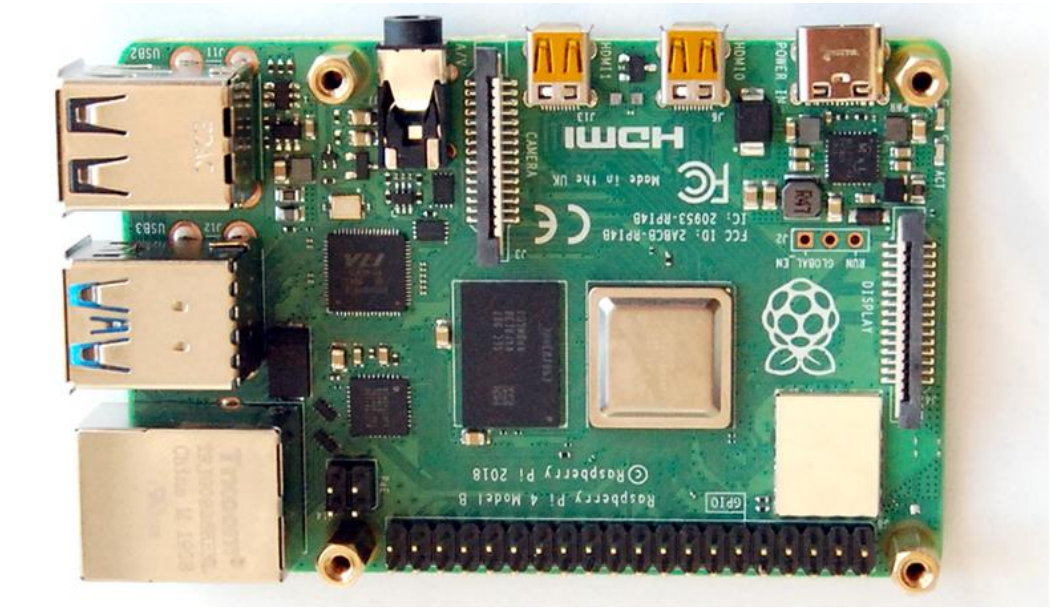
Mounting Witty Pi 3 on Raspberry Pi

You can simply plug Witty Pi 3 on your Raspberry Pi's 2x20 pin header, and it can work just like that. However, if you wish, you can use the copper standoffs and screws in the package to tightly mount Witty Pi 3 on your Raspberry Pi.

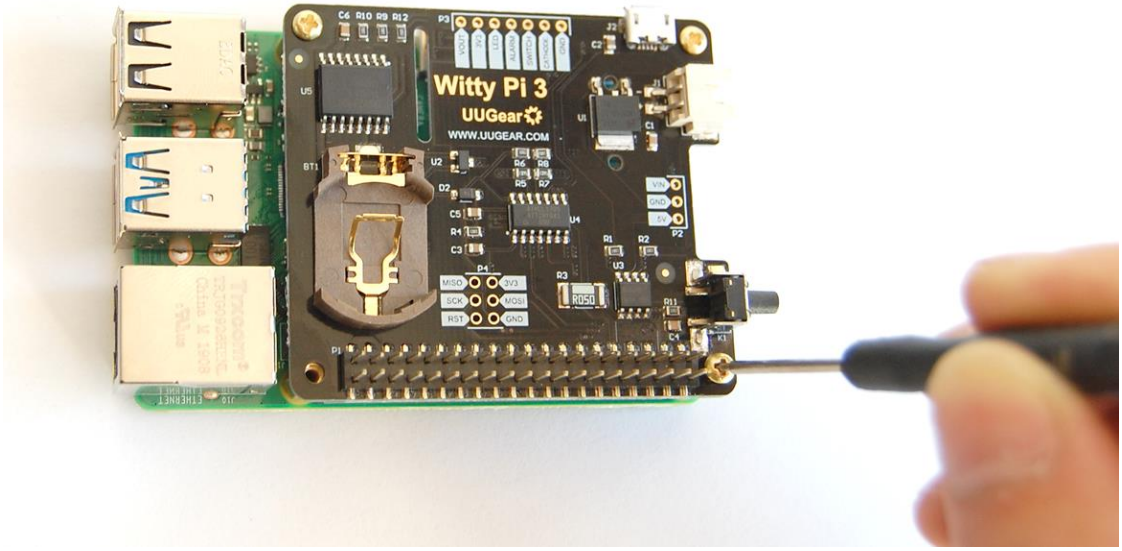
First you can mount the 3 copper standoffs on your Raspberry Pi, using the screws.



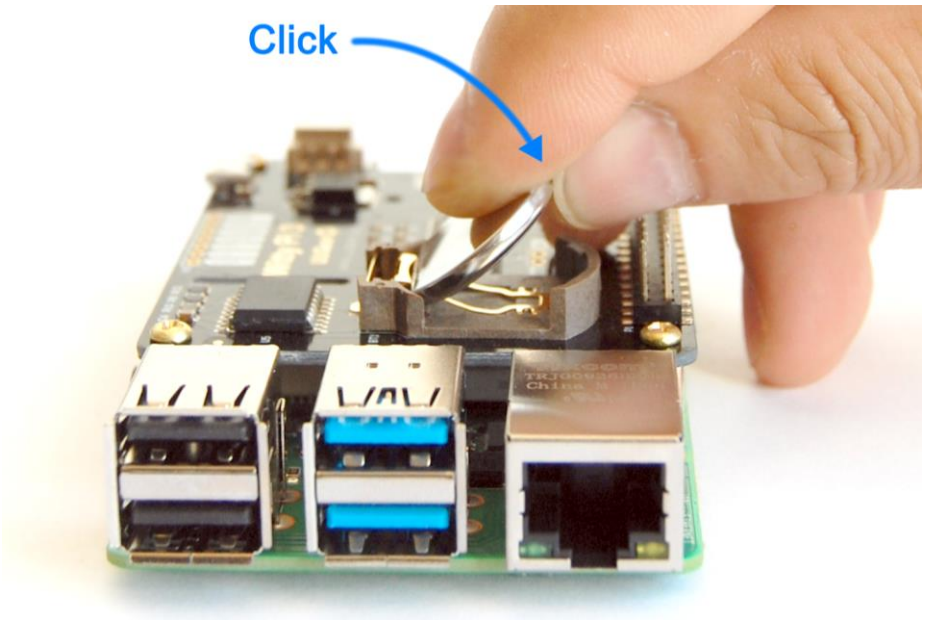
Your Raspberry Pi should look like this after mounting the 4 standoffs:



Then you can mount Witty Pi 3's stacking header on Raspberry Pi's 2x20 pin male header, and then tighten the screws.



Don't forget to put the CR2032 button battery into the battery holder, with the battery Witty Pi 3 can remember the time even after you cut its power. The RTC only draws about 4uA current from the battery to keep the time, so the battery can last years.



After mounting Witty Pi 3 on your Raspberry Pi, and connect the power supply to Witty Pi 3 (via the micro USB connector or XH2.54 connector), you can see **the white LED blinks for every few seconds, which means it is standing by**.

Now your Witty Pi 3 is ready to go.

Input Voltage

Witty Pi 3 newly adds an on board LDO voltage regulator (LM29150) and hence it can accept voltage higher than 5V (if you input power via the XH2.54 connector).

The LM29150 is 26V tolerated, but it doesn't mean you can really use 26V to power your Witty Pi 3. LM29150 is a linear regulator, which means the dropped voltage will be dissipated as heat and the heating power can be calculated by:

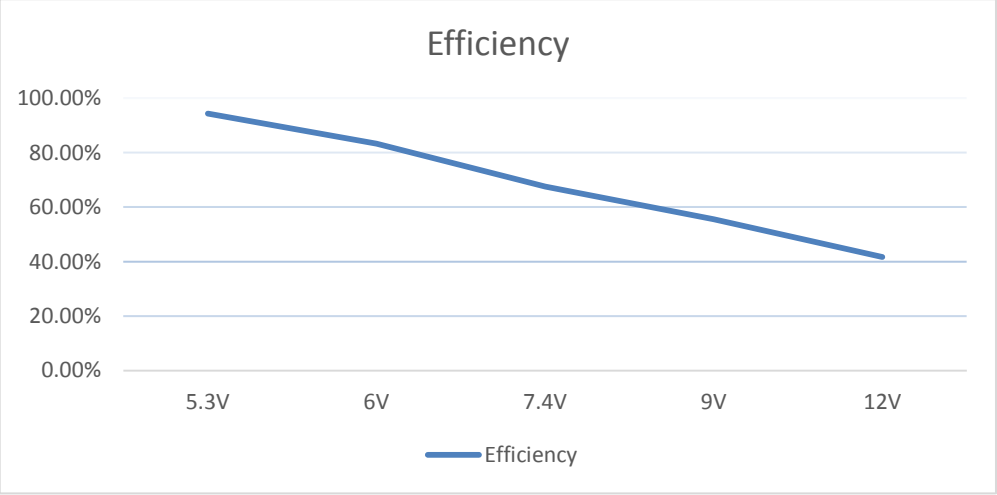
$$P_{heat} = V_{drop} * I_{out}$$

For example, when using only Raspberry Pi 4 and no other devices attached, the I_{out} is about 0.5A. If we input 9V and let the regulator to regulate it to 5V and power Raspberry Pi, the thermal dissipation power will be:

$$(9 - 5) * 0.5 = 2W$$

This is rather high power and it needs to be dissipated soon enough, otherwise the regulator will become hotter and hotter until it eventually overheated and gets damaged. You will need to install external heat sink to use higher voltage.

Also please consider the efficiency, which will drop significantly when input voltage increase. When using 6V the efficiency is about 83%, while using 9V the efficiency will be lower than 56%.



Our suggestion is to use no higher than 8V, when no external heat sink is installed. These power sources would be ideal to power Witty Pi 3:

- 6V Lead-acid battery
- 1.5V battery x 4 (in series as 6V)
- 7.4V Li-po battery (2S)
- 5~8V solar panel

About the Dummy Load

What is dummy load?

The dummy load is a functionality that can be activated when your Pi is off. You may need this functionality when you power your Witty Pi + Raspberry Pi with power bank. When your Pi is off, the current draw by Witty Pi is very small, while most of the power banks will enter sleep mode when the load is not big enough, as a result your Witty Pi may not get powered when the scheduled startup is due.

When the dummy load is activated, Witty Pi will draw rather high pulsed current from the power bank with a fixed interval; this may fool the power bank and avoid entering the sleep mode. The dummy load is implemented on firmware level, that it gives power to your Pi for short moment with a fixed interval. Not all USB current meter can capture that pulsed current, because it only lasts very short moment.

Please notice that, dummy load is just a trick that tries to fool the power bank - not all power banks can be fooled. Some of them calculate and check the integral of current, and dummy load cannot keep them awake with pulsed current.

What if it doesn't work?

You may consider physically increase the load by connecting another load resistor between the +5V and GND pins in your Pi's GPIO header. However this will also increase the load for normal usage. If your power bank needs 20mA to keep alive, you can calculate the resistor with:

$$R = V/I = 5/0.02 = 250\Omega$$

Please also mind the power rating of resistor, which should not be less than:

$$Pmin = V*I = 5*0.02 = 0.1W$$

Another approach is to avoid using power bank, and use raw battery instead. However you will need to have your own DC/DC converter to convert the battery voltage to 5V, and also have your own charging circuit. It is less convenience, but more controllable.
Choose a power bank with "always on" mode may also be an approach. The "always on" mode will prevent power bank from entering sleep mode. For example, Voltaic's V15, V25, V50, V75 and V88 support the "always on" mode. If you are going to buy a new power bank to power your Witty Pi + Raspberry Pi, they could be considered.

Software Usage

The wittyPi.sh is a bash script, and you can run it with:

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

Please notice that **sudo is not necessary here** (older version of Witty Pi software needs that). The programme displays the current temperature, the system time, the RTC time, the input/output voltage and output current:

```
pi@raspberrypi ~/wittypi $ sudo ./wittyPi.sh
================================================================================
|
|     Witty Pi - Realtime Clock + Power Management for Raspberry Pi
|
|                                < Version 3.00 >          by UUGear s.r.o.
|
================================================================================
>>> Current temperature: 29.50°C / 85.1°F
>>> Your system time is: Mon 24 Jun 2019 14:45:09 BST
>>> Your RTC time is:        Mon 24 Jun 2019 14:45:10 BST
>>> Vin=7.92V, Vout=4.96V, Iout=0.30A
Now you can:
   1. Write system time to RTC
   2. Write RTC time to system
   3. Synchronize time
   4. Schedule next shutdown
   5. Schedule next startup
   6. Choose schedule script
   7. Set low voltage threshold
   8. Set recovery voltage threshold
   9. View/change other settings...
  10. Reset data...
  11. Exit
What do you want to do? (1~11)
```

The program gives you 11 options, and you can input the number and press ENTER to confirm.

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) and yet the RTC time is not.

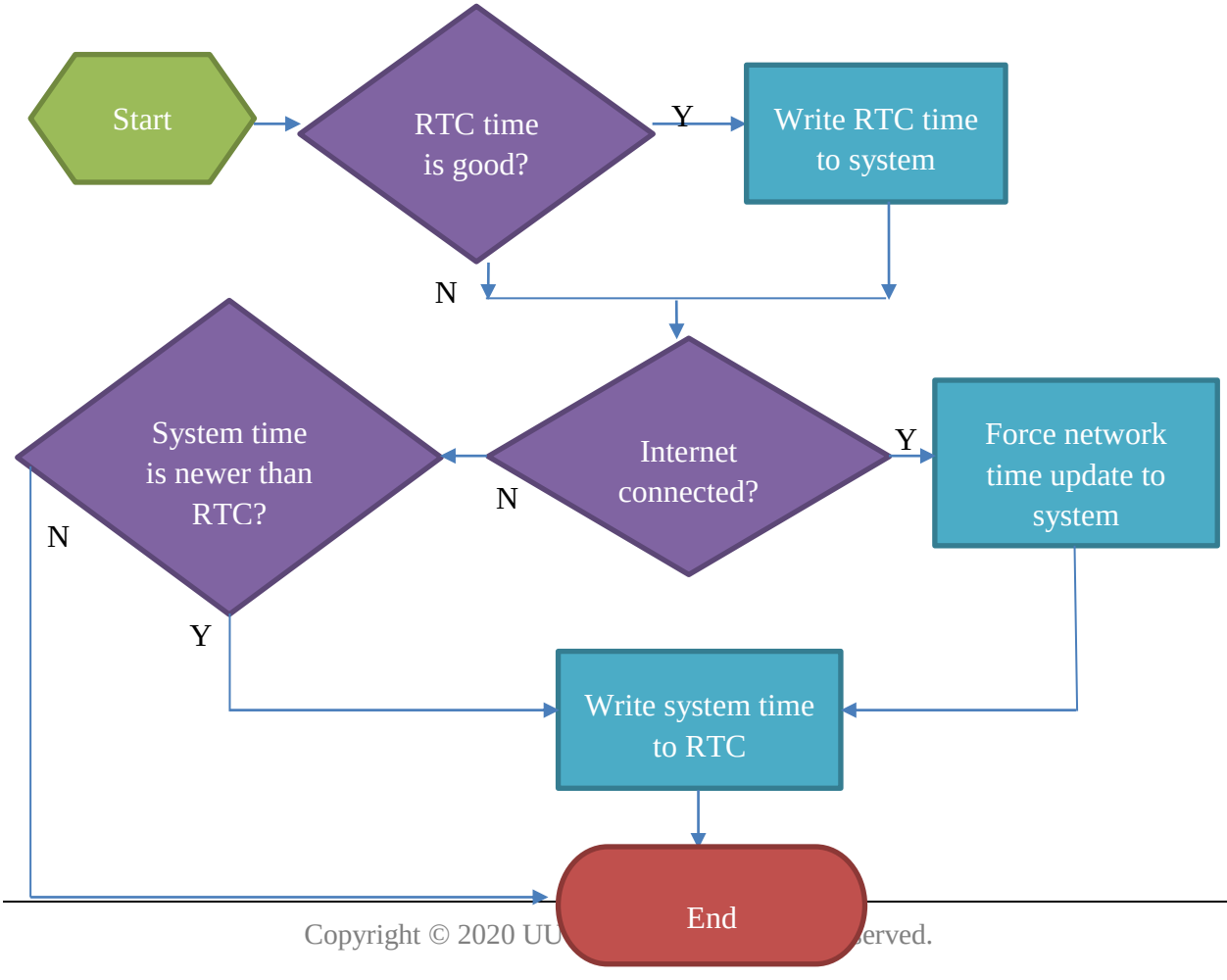
2. Write RTC time to system

This option will copy the time from the Realtime Clock on Witty Pi 3 to your Raspberry Pi system. This option should be used when you find the RTC time is correct while the system time is not.

3. Synchronize time

If you choose this option, it will run the "syncTime.sh" script explicitly, which should have been executed once after the system is up.

This script will detect if Internet is connected, and apply network time to both system and RTC. The



flow chart below shows what this script actually do:

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". DD means the day in the month, HH is the hour, MM is the minute. All these should be 2 digits and 24-hour system is used. Here you can not specify the second. This is a hardware limitation on the RTC chip, and only day, hour and minute could be specified for scheduled shutdown.

You can use "??" as wildcard, which gives you the possibility to make a repeatable schedule. Please see the table below:

Repeatable Shutdown Schedule			
Day (dd)	Hour (HH)	Minute (MM)	Result
??	??	??	Minutely Schedule **(DON'T USE IT!)**
??	??	Number	Hourly Schedule
??	Number	Number	Daily Schedule

Please don't use "?? ??:??" to schedule the next shutdown, or your Raspberry Pi will keep being shutdown and you hardly have a chance to change this setting (unless you remove the battery and force RTC to forget it).

According to the hardware limitation, not all patterns with wildcards are supported. The rule is: wildcards have to show up from let to right, and there is no number between two wildcards. So "?? ??:38" is OK, while "?? 16:??" is not supported.

Here are some examples of scheduling the shutdown:

- **15 21:45** means 9:45 at night, on 15th in this month.
- **?? 23:30** means 23:30 at night everyday (daily schedule)
- **?? ??:15** means the 15th minute every hour (hourly schedule)

5. *Schedule next startup*

This option allows you to specify when your Raspberry Pi should startup 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 is used. Different than the shutdown scheduling, you can specify the second here.

You can also use "??" as wildcard, which gives you the possibility to make a repeatable schedule. Please see the table below:

Repeatable Startup Schedule			
Day (dd)	Hour (HH)	Minute (MM)	Result
??	??	??	Minutely Schedule
??	??	Number	Hourly Schedule
??	Number	Number	Daily Schedule

According to the hardware limitation, not all patterns with wildcards are supported. The rule is: wildcards have to show up from let to right, and there is no number between two wildcards. So "?? ??:??:12" is OK, while "?? 15:??:25" is not supported.

If you input an unsupported pattern, Witty Pi 3 will try to change it to the closest one that could be supported. You will see the message on the console.

Here are some examples of scheduling the startup:

- **15 07:30:00** means 7:30 in the morning, on 15th in this month.
- **?? 23:30:00** means 23:30:00 at night everyday (daily schedule)
- **?? ??:15:00** means the 15th minute every hour (hourly schedule)
- **?? ??:??:05** means the 5th second every minute (minutely schedule)

6. *Choose Schedule Script*

What if you want to define a complex ON/OFF sequence for your Raspberry Pi? The answer is "schedule script".

A schedule script (.wpi file) defines a loop, with all states and their durations inside. By automatically running "runScript.sh" after booting, Witty Pi 3 will automatically 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 3 will take care of the rest.

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 on.

If you want to create your own schedule script, please read the "Making Schedule Script" chapter.

7. Set low voltage threshold

If you are powering your Witty Pi 3 from the XH2.54 connector, and the input voltage is lower than the low voltage threshold, Witty Pi 3 will shutdown your Raspberry Pi. Here you can specify the low voltage from 2.0V to 25.0V. If you want to disable the low voltage threshold, just set it to 0.

8. Set recovery voltage threshold

If you are powering your Witty Pi 3 from the XH2.54 connector, and the input voltage is higher than the recovery voltage threshold, Witty Pi 3 will turn on your Raspberry Pi if it was previously shut down due to too low input voltage. Here you can specify the recovery voltage from 2.0V to 25.0V. If you want to disable the recovery voltage threshold, just set it to 0.

Remarks: the low voltage threshold and recovery voltage threshold can be set individually and their combinations will make Witty Pi 3 work differently.

Low Voltage	Recovery Voltage	Result
Not Set	**Not Set**	Witty Pi 3 doesn't care about the input voltage changing.
Set	**Not Set**	Witty Pi 3 will shutdown 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 3 will shutdown your Raspberry Pi if input voltage is too low. Once this happens, Witty Pi 3 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 3 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 turn on Raspberry Pi again just after shutting it down.

9. 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 3, and you will need to tap the button on Witty Pi 3 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 3 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 8.0.

- **Pulsing interval during sleep**
  This parameter will decide how long Witty Pi 3's micro control 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. Here you can only pick a value from 1, 2, 4 and 8.

- **White LED duration**
  This parameter will decide how long the white LED should stay on, when Witty Pi 3 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 3 + Raspberry Pi, after turning off Raspberry Pi and cut the power, Witty Pi 3 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 3 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 to use 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.

10. *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 3 will not auto-start your Raspberry Pi.

- **Clear auto shutdown time:**
  The auto-shutdown time will be erased and Witty Pi 3 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.

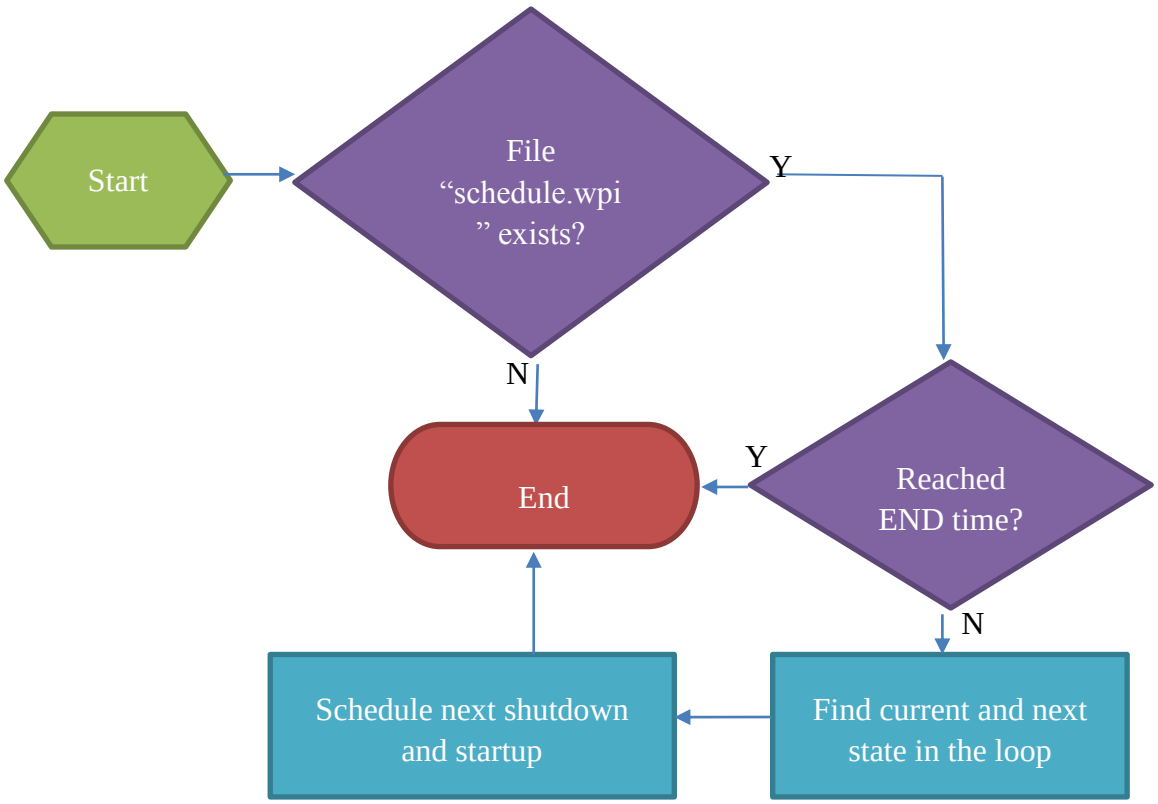
- **Perform all actions above:**
  Clear all scheduled times, remove the “schedule.wpi” file and unset the low voltage and recovery voltage thresholds.

11. *Exit*

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

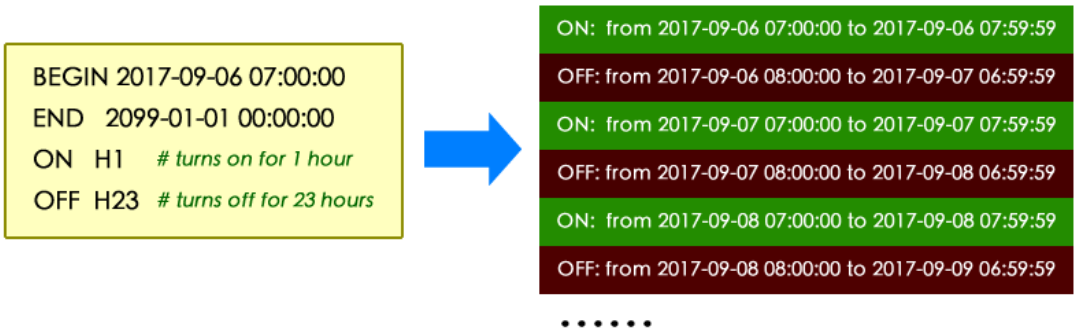
How Schedule Script Works?

A schedule script defines a serial of ON/OFF states and specify the duration of each state. At the end of each state, there should be a scheduled shutdown (for ON state) or scheduled startup (for OFF state). All states in the schedule script will be executed in sequence and repeatedly, until the END time is reached.



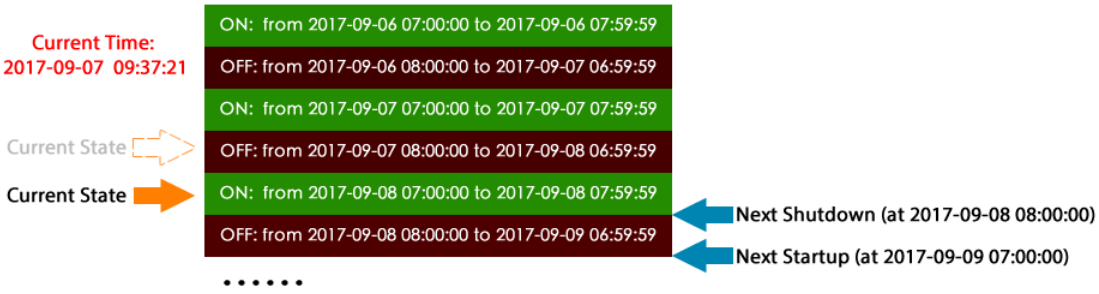
Every time your Raspberry Pi wakes up, it has a chance to run the runScript.sh file, which loads the schedule script (“schedule.wpi” file). If the current time doesn’t reach the END time defined by the schedule script, the next shutdown and next startup will be scheduled automatically. When your Raspberry 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, as it knows Raspberry Pi is currently ON.



Make Schedule Script

A schedule script is a text file with. Wpi file extension. You can use any text editor to create and edit it. In the major of cases, using "nano" will be very convenient.

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  2015-08-01 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 actually 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 are responsible for the shutdown of your Raspberry Pi. Also 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.

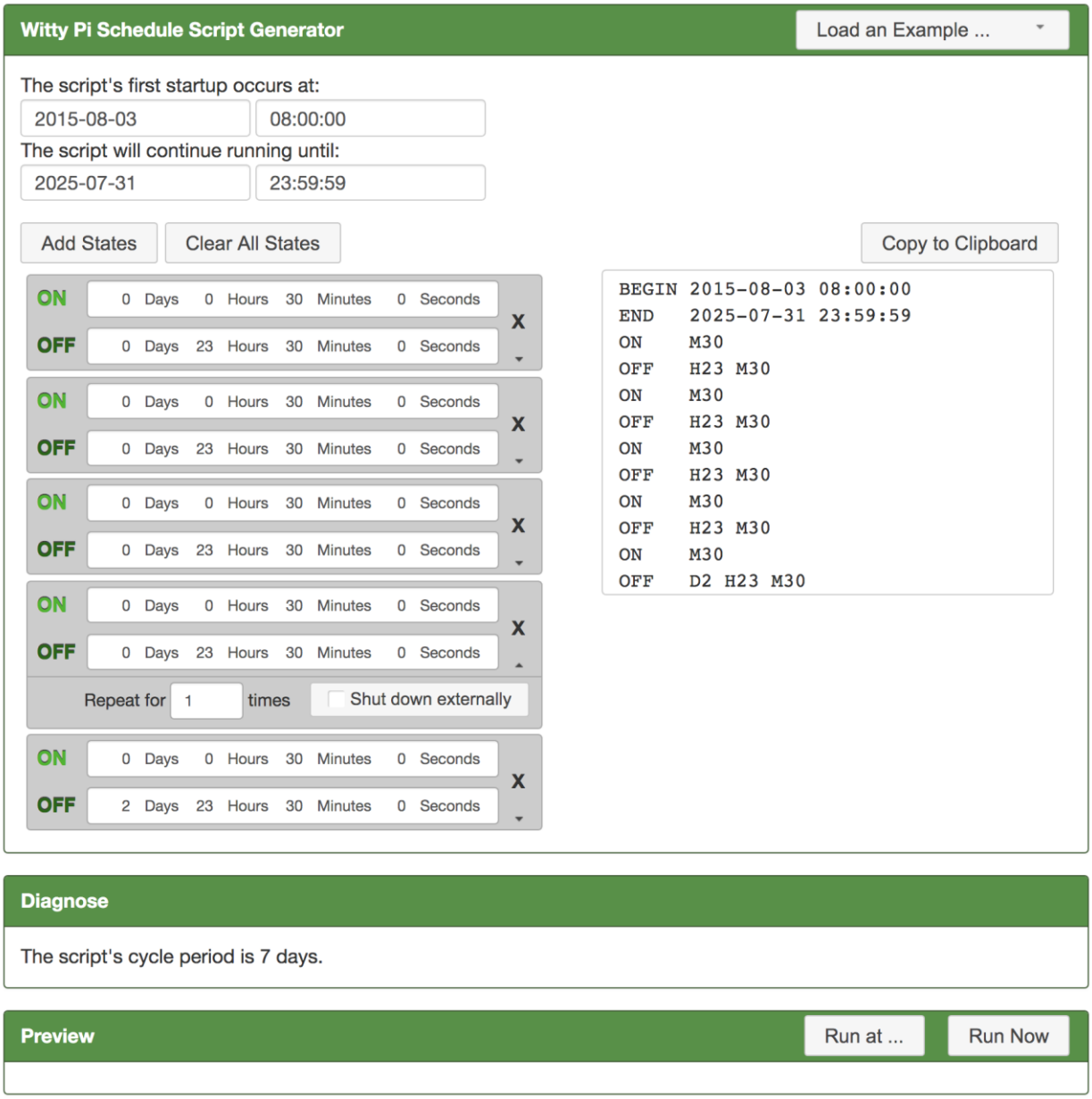
Using Schedule Script Generator

You can use our web application to create your schedule script. Just simply open this URL in your web browser and you are ready to go:

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

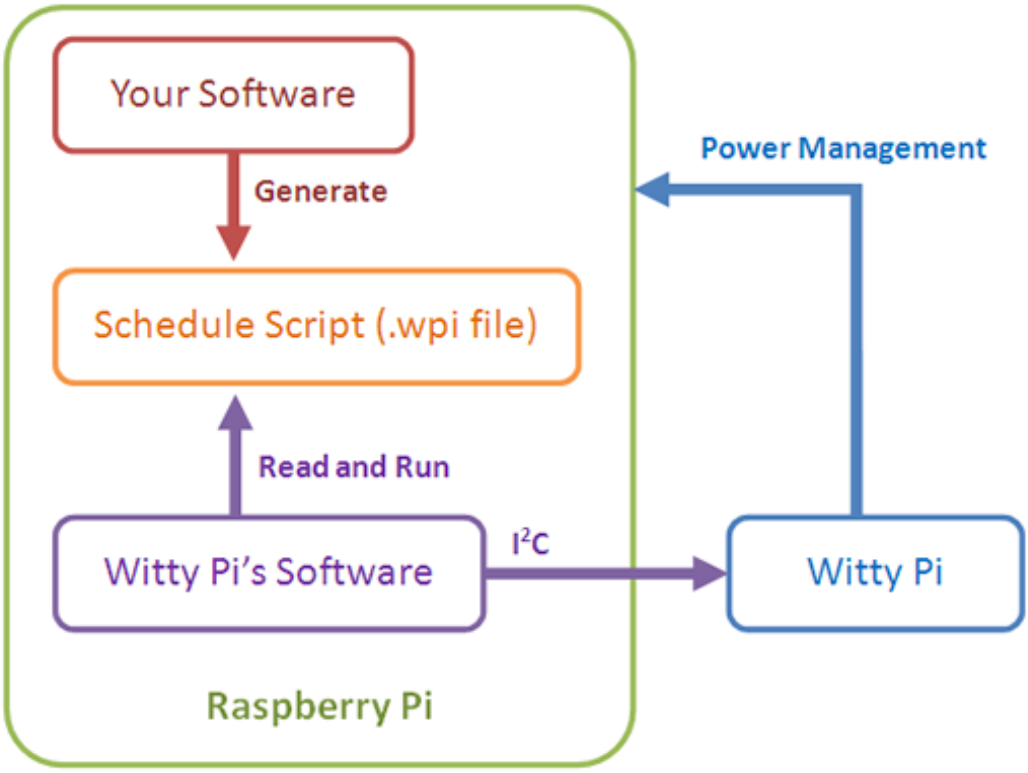
This web application allows you to visually create the schedule script, it immediately generate the final schedule script (on the right).

You can also click the “Run” 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.



Advanced Usage of Schedule Script

Although the schedule script can be chosen by wittyPi.sh, you can use it 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 3 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.



The Unpopulated 3-Pin Header (P2)

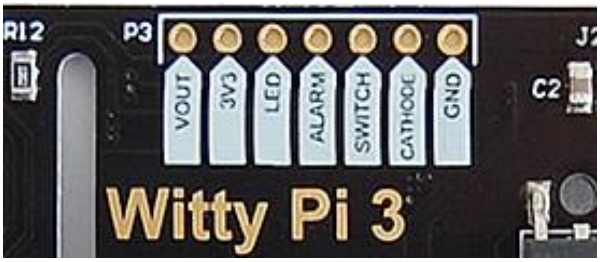
On the right edge of Witty Pi 3 board, there is an unpopulated 3-pin header, and you can decide to use male or female header here.

This header is for inputting voltage to power Witty Pi 3. It will be useful if you neither want to input 5V via the micro USB 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.

The Unpopulated 7-Pin Header (P3)

On the upper edge of Witty Pi 3 board, there is an unpopulated 7-pin header, and you can decide to use male or female header here.



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 on Raspberry Pi, which is the output voltage of Witty Pi 3 board. By detecting its voltage, you will know if your Raspberry Pi is in ON state.

Please mind the actual voltage applied to your Raspberry Pi is between VOUT and CATHODE. The GND in this header is for Witty Pi 3, not for Raspberry Pi.

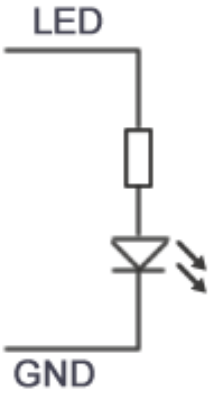
If there is another device need to get powered (by 5V) with Raspberry Pi together, you can connect it to this VOUT and CATHODE pins.

3V3

It is 3.3V voltage on board that powers the micro controller and the RTC.

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.



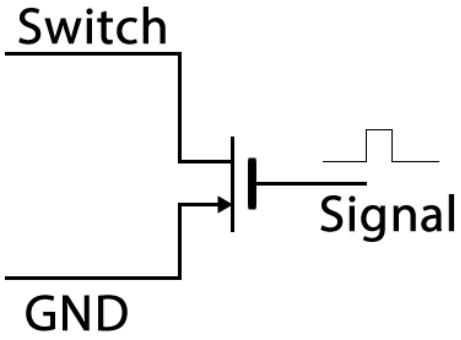
ALARM

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 3's software detects and processes it.

SWITCH

It is the signal line that connects to the switch (button) on Witty Pi 3. If you want to connect your own (2-lead) switch, just wire the two leads to Switch and GND pins.

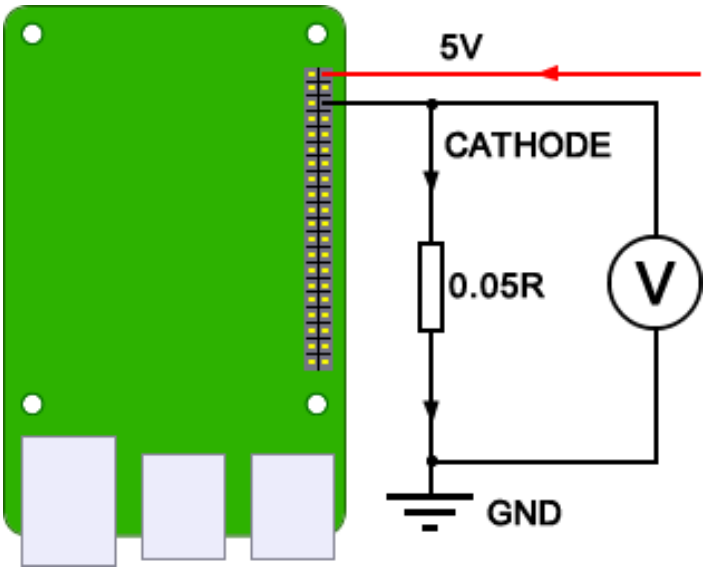
Alternatively, if you wish to trigger Witty Pi 3 with external signal, you can use a N-channel MOSFET to achieve this:



The signal should be a positive pulse, and the pulse length should be longer than 300ms. Processing a pulse will be equal to taping the switch once, so it will turn on your Raspberry Pi if your Pi is off, or turn off your Raspberry Pi if your Pi is on.

CATHOD

It is the cathode 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 like this:

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

GND

It is the ground of Witty Pi 3 board, and it directly connects to the GND wire of power supply.

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 get 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 debugging.

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 might not be correct as the system time has not been updated by RTC yet. You will see the correct time after the RTC time has been written to the system. For example:

```
[2020-05-09 11:10:14] Witty Pi daemon (v3.11) is started.
……
[2020-05-09 11:10:16] Synchronizing time between system and Witty Pi...
[2020-05-09 11:10:16] Writing RTC time to system...
[2020-05-09 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…". If there is Internet connection, the NTP time will be applied to the system and RTC later.

Again, 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.

Frequently Asked Questions (FAQ)

What I2C Address is used by Witty Pi 3?

Raspberry Pi communicates with the RTC chip (DS3231) on Witty Pi 3 via I^{2}C protocol. The DS3231 chip has a fixed I^{2}C address: **0x68**.

Also Witty Pi 3 has a micro controller (ATtiny841), which also works as an I^{2}C slave, and the micro controller's I^{2}C address is **0x69**.

If you have Witty Pi 3 connected to Raspberry Pi and run "sudo i2cdetect –y 1" in the console, you will see this:

```
pi@raspberrypi ~ $ sudo 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: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi ~ $
```

This RTC address (0x68) is fixed and you can never change it. If you want to use other I^{2}C devices on your Raspberry Pi, please make sure they have different I^{2}C addresses.

The micro controller can use other address than 0x69 by configuring the I²C register at position #9. However, that also need to modify the software accordingly.

What I²C Registers Are Provided by Witty Pi 3?

The micro controller Witty Pi 3 works as an I²C slave and Raspberry Pi can read/write some registers via I²C interface. The software configures Witty Pi 3 by setting the I²C register accordingly.

The table below shows the 20 registers provided by Witty Pi 3. As you can see, some of them are read-only (can not be changed, or can only be updated by the firmware itself):

Address	Description	Range	Default	Accessible
0	Firmware ID	--	0x22	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 micro USB = 0	1 or 0	1	Read-only
8	A flag indicates that whether the previous shutdown was due to low voltage: • Yes = 1 • No = 0	1 or 0	0	Read-only

9	I2C slave address: default=0x69	0x08 ~0x77	0x69	Read & Write
10	State when power connected: • Default On = 1 • Default Off = 0	1 or 0	0	Read & Write
11	Pulse interval when Raspberry Pi is off: • 1 second = 6 • 2 seconds = 7 • 4 seconds = 8 • 8 seconds = 9 This parameter affects the white LED blinking and dummy load.	6,7,8 or 9	8	Read & Write
12	Low voltage threshold (multiple 10 times, 255=disabled)	0~255	255	Read & Write
13	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
14	The delay (multiple 10) before power cut: default=50 (5 seconds)	0~255	50	Read & Write
15	Recovery voltage threshold (multiple 10 times, 255=disabled)	0~255	255	Read & Write
16	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
17	Adjustment for measured input	-127	20	Read &

	voltage (multiple 100 times)	~127		Write
18	Adjustment for measured output voltage (multiple 100 times)	-127 ~127	20	Read & Write
19	Adjustment for measured output current (multiple 100 times)	-127 ~127	0	Read & Write

Below is an example to read the register at address 7, to know the current power mode (0x00 means input 5V via the micro USB connector):

```
pi@raspberrypi ~ $ i2cget -y 1 0x69 7
0x00
pi@raspberrypi ~ $
```

And below is an example to write the register at address 11, to set the pulsing interval to 1 second (using value 6):

```
pi@raspberrypi ~ $ i2cset -y 1 0x69 11 6
pi@raspberrypi ~ $
```

Remarks: although the register at address 9 is writable, **changing the I2C slave address is more than writing an I²C register**. You will need to modify the software accordingly (change the I2C_MC_ADDRESS value in utilities.sh file), and reconnect the power supply to make it work.

What GPIO Pins Are Used by Witty Pi 3?

The GPIO pins used by Witty Pi 3 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 3 uses **GPIO-4**, **GPIO-17**, **GPIO-2 (SDA 1)** and **GPIO-3 (SCL 1).**

Witty Pi 3 doesn't actually use the TXD pin, but it will monitor its voltage. The TXD pin is supposed to be HIGH when 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 behaviour, otherwise Witty Pi 3 doesn't know when the system is off, and can not fully cut the power.

GPIO-2 and GPIO-3 are for I^2C communication between Raspberry Pi and the RTC chip (DS3231). I^2C devices are identified by I^2C address, and they can share the I^2C pins as long as they have different I^2C addresses.

Is Witty Pi 3 Compatible with "Other Hardware"?

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

Please understand that we might not have the hardware you have on hand, and even if we have, it is difficult for us to make tests on all these hardware with Witty Pi 3. Fortunately, it is not that difficult to figure out the answer by yourself. Basically you just need to consider the I^2C address and GPIO pins used by the "Other Hardware".

If the "Other Hardware" uses 0x68 I^2C address, it will be a confliction with Witty Pi's RTC chip and you can not use it with Witty Pi together. If the "Other Hardware" uses 0x69 I^2C address, you will need to change the I^2C address used by Witty Pi's micro controller, and change the software accordingly.

If the "Other Hardware" doesn't use any GPIO pin that used by Witty Pi, and it doesn't use 0x68 or 0x69 I^2C address, then it should be compatible with Witty Pi.

If you have no idea which I^2C 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.

Witty Pi 3 does not boot?

Some customers meet the situation that Witty Pi 3 only boot Raspberry Pi for a few seconds, and then shutdown Raspberry Pi or cut the power directly.

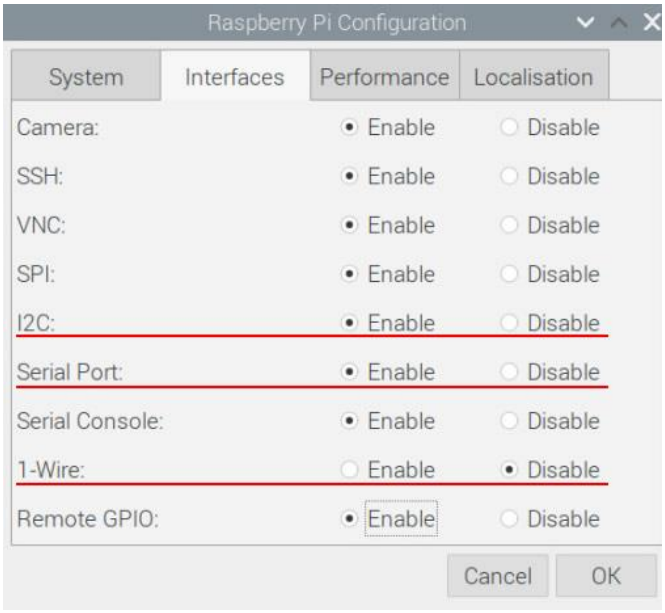
Remarks: you may need to disconnect Witty Pi 3 from your Raspberry Pi, and power on Raspberry Pi only for troubleshooting.

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 has GUI installed, you can review/configure these interfaces via Raspberry Pi Configuration:



Also you can check all GPIO pin states with "gpio readall" command. 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. 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 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 a 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.

2. Software does not run automatically after Raspberry Pi system is on

After installing the software, you will have "daemon.sh", "syncTime.sh" and "runScript.sh" scripts in the directory that has software installed. These scripts should be executed automatically after the operating system is up.

If these scripts are not automatically executed for any reason, Witty Pi 3 will cut the power of Raspberry Pi. In this case, the first place to check is the "/etc/init.d/wittypi" file, and there is possibility that it contains wrong paths to these scripts.

If you are running a special distribution of OS, make sure to check if the commands in the "/etc/init.d/wittypi" file do exist in the system, or it will fail silently. For example, Minibian doesn't have "sudo" command.

3. Witty Pi 3 is unexpectedly woken up by shutdown alarm

This could happen, after you scheduled a shutdown in the future, and then Witty Pi 3 lost power connection. Raspberry Pi is off ungracefully and later the power connection is resume. If the shutdown alarm occurs when Raspberry Pi is still off, it will wake it up, which is unexpected. The software will detect this situation and shutdown your Raspberry Pi immediately.

Next time when you turn on your Raspberry Pi, it will be normal again, and you can find this message in the log, for previous shutdown:

```
Seems I was unexpectedly woken up by shutdown alarm, must go back to sleep…
```

Sometimes the same case can happen when you tap the button to turn on your Raspberry Pi, if the shutdown alarm occurs during the time that no power supply was connected.

You may ask, how comes the shutdown alarm could act like a startup alarm? This situation exists since Witty Pi 2. Witty Pi 2 and 3 use the same RTC chip (DS3231), which is way better than the one used in on Witty Pi (DS1337). The downside however, is the new RTC chip merges two alarm pins into one. Hence without turning on your Raspberry Pi and check the RTC registers, Witty Pi doesn't know if the current alarm is a shutdown alarm or not.

4. 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 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.

5. 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 10 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 the given 10 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 the GPIO-4 listener to shutdown your Raspberry Pi.

If it is the case, you can try to delay the starting of GPIO-4 listener, and in the major 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 10 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, or you can use another pin to replace GPIO-4.

Revision History

Revision	Date	Description
1.00	2019.06.14	Initial revision
1.01	2019.06.24	Add information for Raspberry Pi 4B
1.02	2019.07.10	Add description of provided I2C registers
1.03	2019.07.28	Information for confliction with 1-Wire interface.
1.04	2020.01.13	Change default firmware ID from 0x20 to 0x22. Add more information for “Witty Pi 3 does not boot?”
1.05	2020.05.20	Add description of Witty Pi log files.
1.06	2020.07.03	Add description of dummy load.