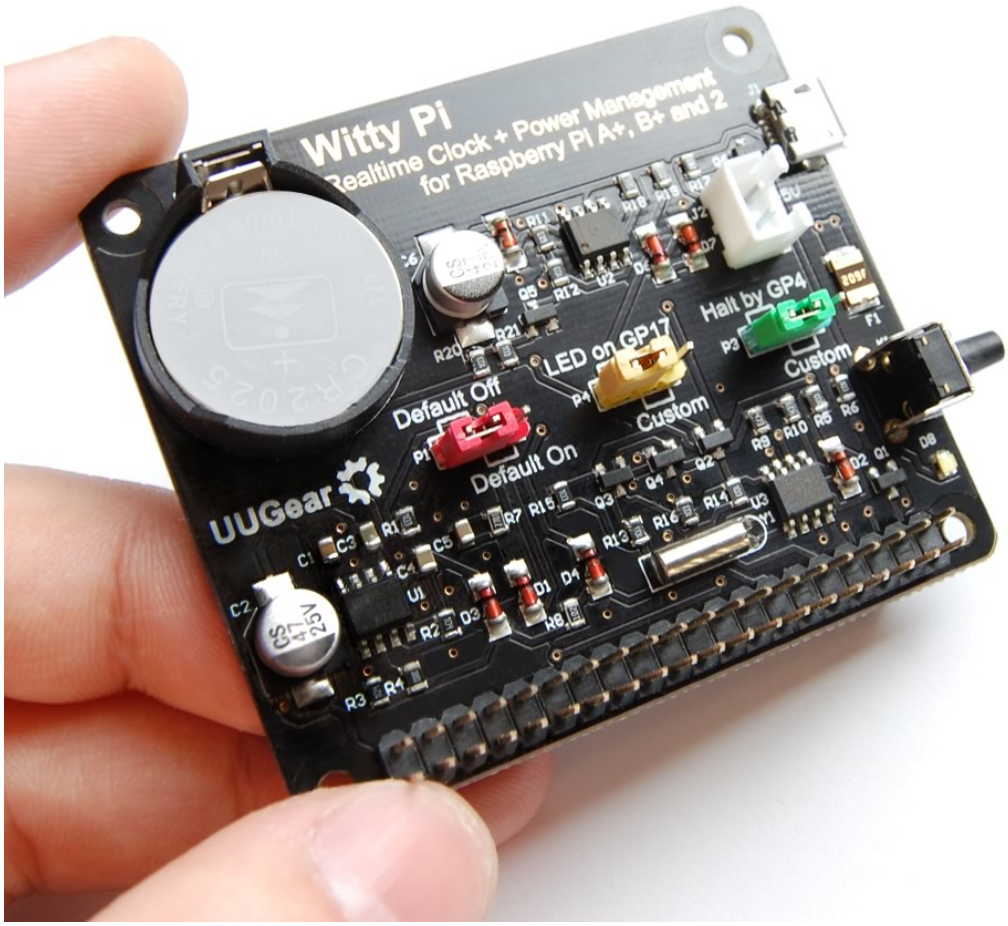
UUGear

Witty Pi

Realtime Clock and Power Management for Raspberry Pi

User Manual



UUGear

Table of Content

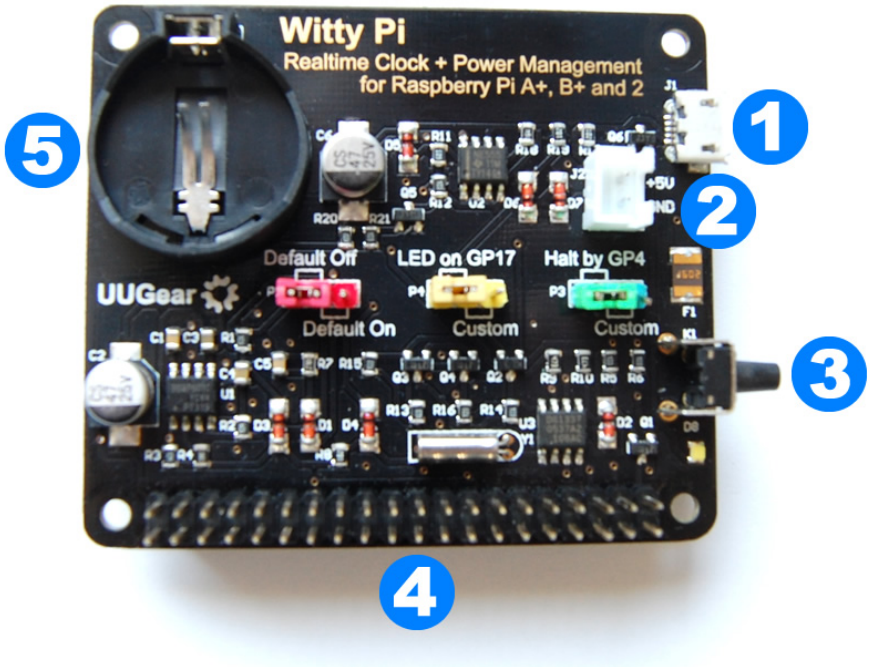
What is Witty Pi?

Witty Pi is small extension board that can add realtime clock and power management to your Raspberry Pi. After installing Witty Pi on your Raspberry Pi, you get some amazing new features:

- You can properly turn on/off Raspberry Pi with single tap on the switch.
- Raspberry Pi and all its USB peripherals get fully power cut after shutdown.
- Raspberry Pi knows the correct time, even without accessing the Internet.
- You can schedule the startup/shutdown of your Raspberry Pi.
- You can even write a script to define complex ON/OFF sequence.

Witty Pi supports Raspberry Pi model A+, B+ and Raspberry 2.

The picture below shows how is Witty Pi look like:



1) DC 5V power in
2) Alternative DC 5V power in
3) On/off switch
4) 40-pin header (for connecting to Raspberry Pi)
5) Battery holder (for button cell lithium battery CR2032/CR2025)

What is in the Package?

Each Witty Pi package contains:

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

UUGear

Witty Pi Specifications

Dimension:	65mm x 56mm x 19mm
Weight	23g (net weight without battery)
Realtime Clock Chip	DS1337 (datasheet)
LED Indicator	A white LED, which lights up for a few seconds when a shutdown command is received, or fade in and out slowly (breathing) when Witty Pi is standing by.
Connector	40-pin dual row 2.54 mm pitch female & male header
Battery	CR2032 or CR2025 (for time keeping)
Power In	DC 5V (via micro USB port or white power jack)
Output Current	Maximum 2A for Raspberry Pi and its peripherals
Static Current	< 10mA
Operating Temperature	0℃~50℃
Storage Temperature	-20℃~60℃
Humidity	0~80%RH, no condensing

Software Installation

We strongly recommend to install the software for Witty Pi before physically mount Witty Pi on your Raspberry Pi.

You will need to have your Raspberry Pi connected to the Internet. The installation will be very simple if you run the installing script from us. The wiringPi utility is required by the software so the script will install it for you, if you don't have it yet.

The software has been tested under **Raspbian Wheezy** and **Raspbian Jessie**. It might also works on other operating systems with or without modification.

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

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

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

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

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

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
5. Install Witty Pi programs, if they are not installed yet

You can also manually install these packages 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 see a new "wittyPi" directory, and it contains 3 runnable scripts:

```
pi@raspberrypi ~ $ cd wittyPi
pi@raspberrypi ~ /wittyPi $ ls
daemon.sh   init.sh   syncTime.sh   utilities.sh   wittyPi.sh
```

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 suppose to be manually run either, it will run automatically at about one minute 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 correct time and your Raspberry Pi hasn't (because of the lacking of Internet), it will copy the RTC time to your Raspberry Pi system.

If you wish the **syncTime.sh** script run as early as possible, you can edit the **/etc/init.d/wittypi** file, line 22:

```
sudo /home/pi/wittyPi/syncTime.sh 60 &
```

The number 60 means running the script 60 seconds after the startup, you can set it to 0 to trigger an immediate run after the boot.

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.

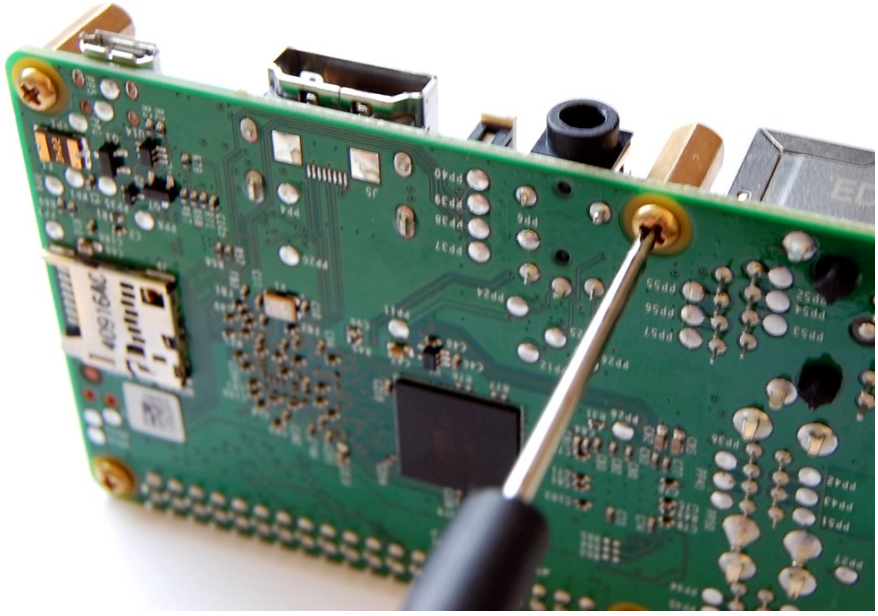
Now the software has been installed, and you will need to physically mount Witty Pi on your Raspberry Pi.

UUGear

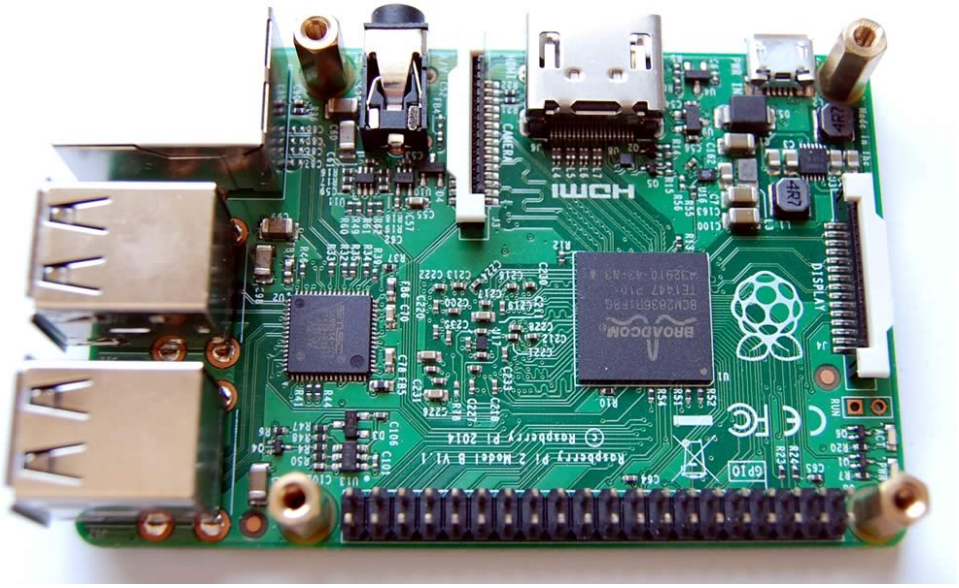
Mounting Witty Pi on Raspberry Pi

You can simply plug Witty Pi on your Raspberry Pi's 40-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 on your Raspberry Pi.

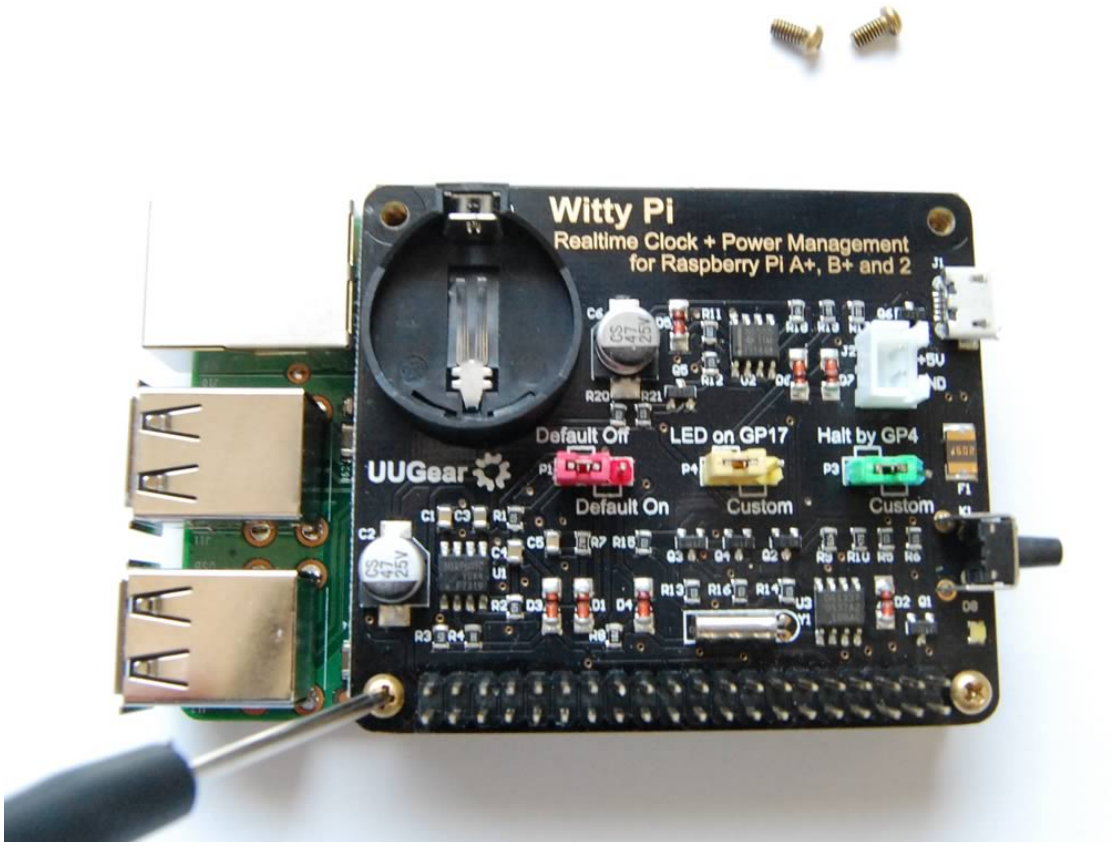
First you can mount the 4 copper standoffs on your Raspberry Pi, using the screws.



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



Then you can connect Witty Pi's female header (at bottom) with Raspberry Pi's 40-pin male header, and then tighten the screws.



Don't forget to put the button battery into the battery holder, with the battery Witty Pi can remember the time even after you cut its power.

After mounting Witty Pi on your Raspberry Pi, and plug the power supply to the micro USB port on Witty Pi, you can see **the white LED fading in and out slowly, which means it is standing by**.

Now your Witty Pi is ready to go.

Software Usage

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

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

Please notice that sudo is required. Once the script is run, it will tell you the system time and RTC (Realtime clock) time, so you can decide how to copy the time.

```
pi@raspberrypi ~/wittyPi $ sudo ./wittyPi.sh
================================================================================
|                                                                              |
|     Witty Pi - Realtime Clock + Power Management for Raspberry Pi            |
|                                                                              |
|                              < Version 2.16 >          by UUGear s.r.o.      |
|                                                                              |
================================================================================
>>> Your system time is:    Sun 01 Nov 2015 16:51:29 CET
>>> Your RTC time is:       Sun 01 Nov 2015 16:51:29 CET
Now you can:
   1. Write system time to RTC
   2. Write RTC time to system
   3. Set time for auto startup
   4. Set time for auto shutdown
   5. Choose schedule script
   6. Reset data...
   7. Exit
What do you want to do? (1~7)
```

The program gives you 7 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 system time is correct (may get

synchronized from Internet) and RTC time is not.

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.

3. Set time for auto startup

This option allows you to specify when your Raspberry Pi should auto startup. 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 hours system is used.

You can also use "??" as wildcard while inputting the time. This gives you the possibility to make a repeatable schedule. Please see the table below:

Day (dd)	Hour (HH)	Minute (MM)	Result
??	??	??	Minutely Schedule
??	??	number	Hourly Schedule
??	number	number	Daily Schedule

Here are some examples:

- **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)

Please be informed that not all patterns with wildcards are supported. According to the hardware limitation, only a few combinations are supported. The rule is, in short: wildcards have to show up from let to right, and no any number appears between two wildcards. So "?? ??:??:12" is OK and "?? 15:??:25" is not supported.

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

4. Set time for auto shutdown

This option allows you to specify when your Raspberry Pi should auto shutdown. Please notice the input format should be “DD HH:MM”, it is basically the same with the previous option but without the second in the end. It is a hardware limitation and only day, hour and minute could be specified for auto shutdown.

Here are some examples:

- **15 21:45** means 9:45 at night, on 15th in this month.

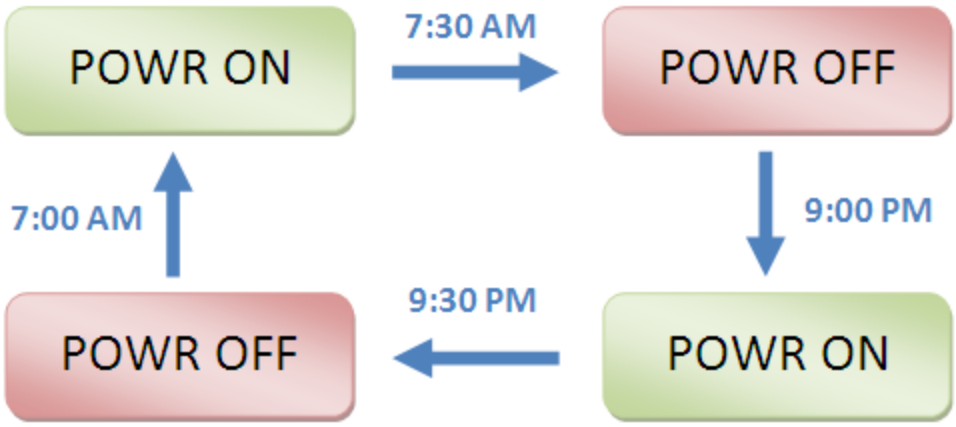
Similar with the auto startup schedule, you can use “??” as wildcard as well, and the rules to use wildcards are the same.

- **?? 23:30** means 23:30 at night everyday (daily schedule)
- **?? ??:15** means the 15th minute every hour (hourly schedule)

5. Choose Schedule Script

This cool feature brings so many new possibilities to use Witty Pi.

The RTC chip on Witty Pi only has two alarms, and we use one for auto startup and the other for auto shutdown. Manually setting the auto shutdown and startup alarm might be enough for the major of use cases, but sometimes you may want a more complex schedule. For example, you want your Raspberry Pi to wake up on 7:00 AM -> shutdown on 7:30 AM -> wake up again on 9:00 PM -> shutdown again on 9:30 PM. The process repeats everyday, as shown in the state chart below.



Obviously you could not achieve this by just setting the auto startup/shutdown time, because there will be two auto-startups and two auto-shutdowns everyday, and you can only set them once.

The solution is to use the schedule script, which is designed for scenarios like this. A schedule script (.wpi file) defines a loop, with all states and their durations inside. By automatically running the script after booting, Witty Pi will automatically schedule the next shutdown and startup for you.

Once you select the “Choose schedule script” option, it will list all schedule scripts in the “schedules” directory. You can choose one, and then Witty Pi will take care of the rest.

```
pi@raspberrypi ~/wittyPi $ sudo ./wittyPi.sh
================================================================================
|                                                                              |
|     Witty Pi - Realtime Clock + Power Management for Raspberry Pi            |
|                                                                              |
|                              < Version 2.16>          by UUGear s.r.o.       |
|                                                                              |
================================================================================
>>> Your system time is:    Sun 01 Nov 2015 16:51:29 CET
>>> Your RTC time is:       Sun 01 Nov 2015 16:51:29 CET
Now you can:
   1. Write system time to RTC
   2. Write RTC time to system
   3. Set time for auto startup
   4. Set time for auto shutdown
   5. Choose schedule script
   6. Reset data...
   7. Exit
What do you want to do? (1~7) 5
   I can see 5 schedule scripts in the "schedules" directory:
   [1] 7:00_on_7:30_off_21:00_on_21:30_off.wpi
   [2] on_10m_every_2h.wpi
   [3] on_1h_every_2d.wpi
   [4] on_30m_everyday_but_weekend.wpi
   [5] on_5m_every_20m.wpi
   Which schedule script do you want to use? (1~5)
```

The first schedule script (7:00_on_7:30_off_21:00_on_21:30_off.wpi) can make a schedule for the example above with state chart.

From the file name you can know what the script will do. For example,

on_10m_every_2h.wpi will turn on your Raspberry Pi for 10 minutes, in every two hours.

If you want to stop using the schedule script, please choose the 6th item in the menu ("Reset Data…") and then select "Perform all actions above". This will delete the "schedule.wpi" file in Witty Pi's install directory, and clear the auto-startup/shutdown time that already gets scheduled.

Remarks: if you use a schedule script, please make sure to let Witty Pi to shutdown your Pi, and do not shut it down manually. Otherwise Witty Pi will try to shutdown your Pi when your Pi is actually off, and that may cause a dead lock. You may need to reset your Witty Pi if that happens.

If you wish to confirm what has been done by the script, you can check the "schedule.log" file in the "~/wittyPi" directory, when your Raspberry Pi is on.

You can create your own schedule script too. If you are interested, please read the "Making Schedule Script" chapter.

6. Reset Data…

If you want to erase the data you already set into Witty Pi (auto-startup time, auto-shutdown time, currently used schedule script), 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 removed.
- Perform all actions above:
  Clear all scheduled times and remove the "schedule.wpi" file.

The figure below shows these sub menu options.

```
pi@raspberrypi ~/wittyPi $ sudo ./wittyPi.sh
================================================================================
|                                                                              |
|     Witty Pi - Realtime Clock + Power Management for Raspberry A+, B+ and 2  |
|                                                                              |
|                              < Version 2.16>          by UUGear s.r.o.       |
|                                                                              |
================================================================================
>>> Your system time is:    Sun 01 Nov 2015 16:51:29 CET
>>> Your RTC time is:       Sun 01 Nov 2015 16:51:29 CET
Now you can:
   1. Write system time to RTC
   2. Write RTC time to system
   3. Set time for auto startup
   4. Set time for auto shutdown
   5. Choose schedule script
   6. Reset data...
   7. Exit
What do you want to do? (1~7) 6
Here you can reset some data:
   [1] Clear auto startup time
   [2] Clear auto shutdown time
   [3] Stop using schedule script
   [4] Perform all actions above
Which action to perform? (1~4)
```

7. Exit

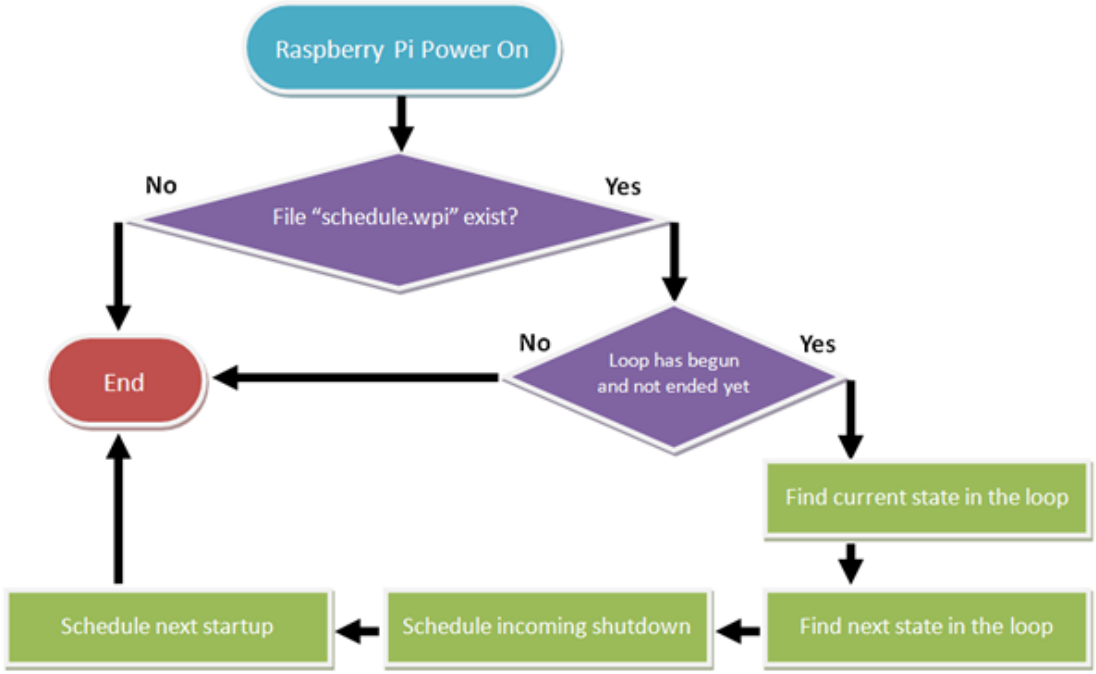
Selecting this option will simply exit the software and return to the console.

Making 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 Raspberry Pi, using “nano” will be very convenient.

A schedule script defines a loop, and there are at least one ON state and one OFF state in the loop. Every time your Raspberry Pi wakes up, it has a chance to run the schedule script (“schedule.wpi” file). If the current time is in the time range defined by the schedule script, the incoming shutdown and next startup will be scheduled

automatically.



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 language, Witty Pi 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. Usually 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, 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 hour, 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:

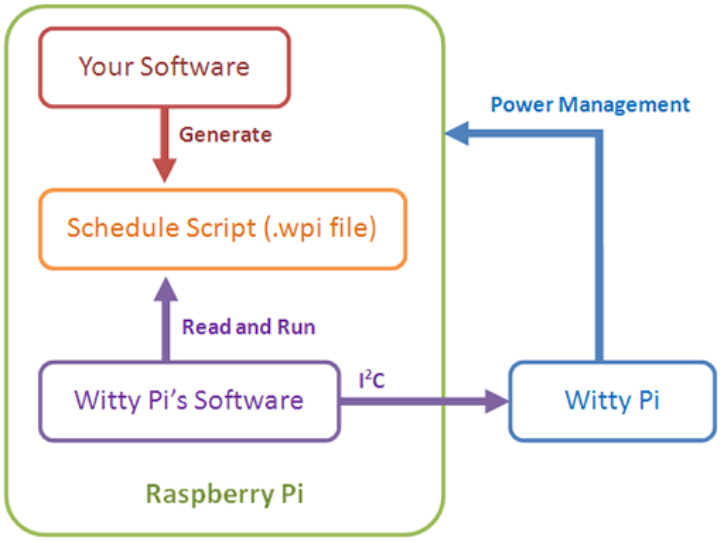
```
OFF     D2
```

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

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.

Advanced Usage:

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 "sudo ./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.

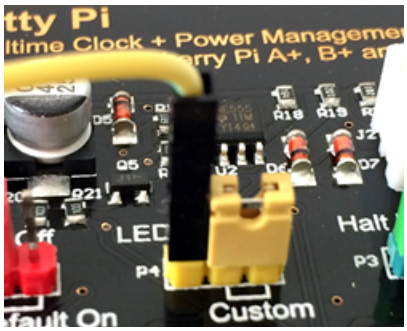


Hardware Configuration

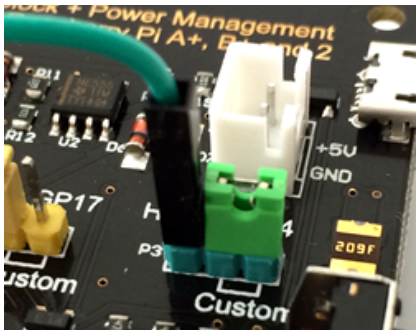
There are three colorful jumpers on the board, and they allow you to make some customization on your Witty Pi.

The red jumper can decide if you Raspberry Pi get powered immediately when you connect the 5V power supply to Witty Pi. By default, this jumper is set to “Default Off”, so you have to click the switch once to power on your Raspberry Pi.

The yellow jumper allows you to specify which GPIO pin is used to drive the LED indicator. GPIO-17 is used by default, and you can change it to any GPIO pin you want. Just put the jumper cap on the right side and wire the left pin of the jumper to the target GPIO pin.



The green jumper is to configure which GPIO pin will be used to shutdown your Raspberry Pi. GPIO-4 is used by default. Please notice that, not every GPIO pin could be used for this purpose. The GPIO pin should have initial high voltage level after the booting, otherwise your Raspberry Pi will get shutdown immediately. The way to configure this jumper is the same with the yellow one.

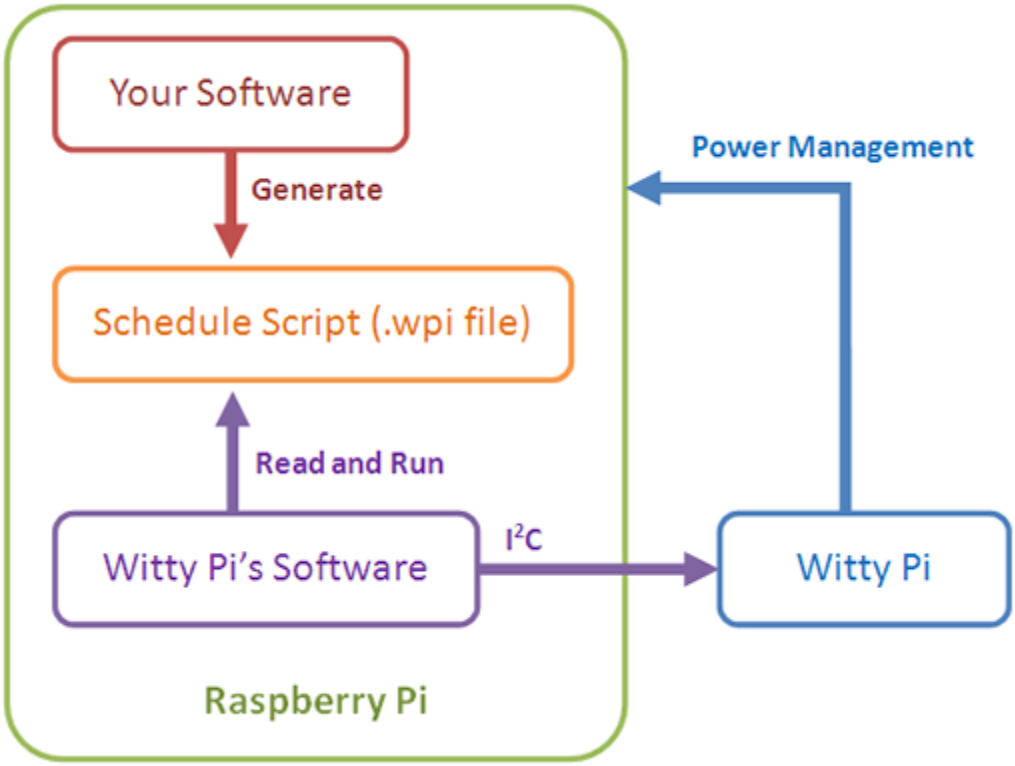


You will also need to modify the software a little bit to use different GPIO pins. More details can be found here:
http://www.uugear.com/portfolio/change-the-pin-that-used-by-witty-pi/

Software Customization

If You like programming, you could customize or even write your own software for Witty Pi. The software from us is a good example to start learning how to write program for Witty Pi. We believe the most interesting part is to program the RTC chip and make different setups of alarms to schedule the Pi's startup/shutdown. We will keep writing tutorials for this part, and update the examples in this manual. Meanwhile you could also checkout the datasheet for DS1337 chip, and its "ALARMS" section has the information that needed for programming the alarms.

There is another choice, which we think is even better. That is to use the schedule script as an interfacing language. You don't need to really understand how Witty Pi's software works; instead you just create your own schedule script, and let Witty Pi's software to take care of the rest. Programming Witty Pi becomes so easy because your application only need to generate a text file, according to user's input.



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**.

The “schedule.log” file contains the history of schedule script executions. Here 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 right, this log file might provide more information for debugging.

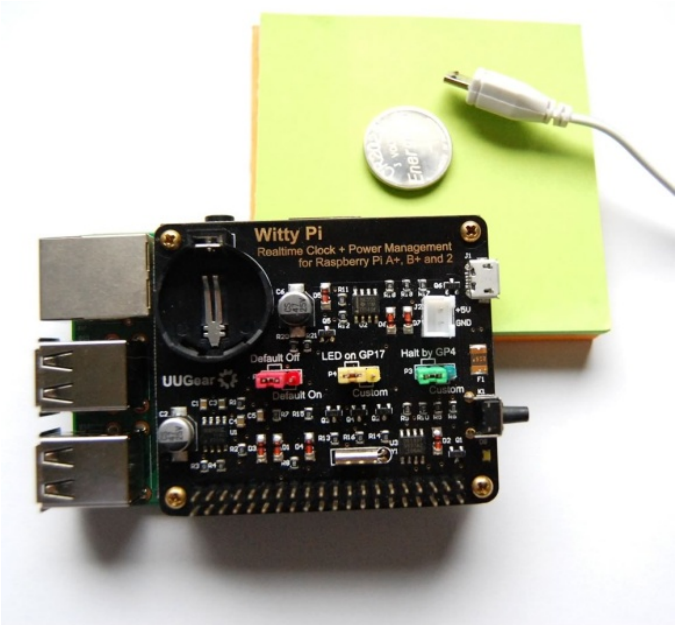
If you need our help for solving a problem, please kindly put the log files in email attachment too. This will help us to help you better.

Witty Pi gets Dead Locked?

If you have power supply connected to Witty Pi, and pressing the button on Witty Pi could not startup your Raspberry Pi (you may or may not see your Raspberry Pi get powered for 1~ 2 seconds only), then your Witty Pi gets dead locked.

If you are using Raspberry Pi 3, this problem most probably comes from the Bluetooth, which is newly on board since Pi3. In the newly released firmware, GP14 and GP15 pins are mapped to UART1, since the UART0 is used by Bluetooth now. As a result, the TXD pin (GP14) is in “INPUT” state after the boot, and it gets internally pulled down, which causes the dead lock status of Witty Pi. **The solution is to install the latest software** (version 2.16 or higher), which has taken care of this situation.

When dead lock happens, you will need to reset your Witty Pi. Just remove the battery on Witty Pi, and unplug the power supply. Then you wait for 1 minute so the RTC chip forgets the alarm state, later you will be able to power it on again.



There are two scenarios that could cause the “dead lock” status:

- Witty Pi schedules the next auto shutdown, and then Raspberry Pi gets shutdown in a way that without pulling down the GPIO-4 pin, such as disconnecting power supply directly, or calling “sudo shutdown –h now” command by the user. Since the GPIO-4 pin is not pulled down, Witty Pi’s daemon is not aware of the shutdown, and hence it will not disable the auto shutdown alarm before cutting the power. When the shutdown alarm gets activated while Witty Pi and Raspberry Pi are not powered, Witty Pi’s software doesn’t have a chance to clear the alarm state, and that equals to long hold the button without releasing it, which prevents your Raspberry Pi to get powered on again.
- The scheduled shutdown comes when your Raspberry Pi is on the process of shutting down. Since the software may have been closed when the shutdown alarm comes, it may not be able to clear the shutdown alarm, and the result will be the same as the previous scenario.

To avoid the dead lock, please make sure not to call the “sudo shutdown –h now” command by yourself, and not to unplug/disconnect the power supply for Witty Pi before finishing the gracefully shutdown process. This can avoid the first scenario mentioned above.

If you use schedule script, or manually schedule a shutdown in very short future, it is better to let Witty Pi to shutdown your Raspberry Pi, otherwise it will become the second scenario that causes the dead lock, when the scheduled shutdown comes during the manually shutdown process.

If you are expecting an auto startup, please make sure your Witty Pi has power supply connected.

If you want to shutdown your Raspberry Pi manually, please **tap the button** on Witty Pi. If you want to shutdown Raspberry Pi in your own script, you can call “**gpio mode 7 out**” (GPIO-4 pin is pin 7 in the wiringPi).

Witty Pi Immediately Shutdown after Startup?

Some customers meet the situation that after installing the software and reboot, Witty Pi immediately shutdown Raspberry Pi after successfully boot, with or without having the white LED lighted up. This could repeat no matter how many times you press the button to startup.

You may need to disconnect Witty Pi from your Raspberry Pi, and power on Raspberry Pi only for troubleshooting.

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

1. Scripts do not get executed 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 will cut the power of Raspberry Pi, **without lighting up the white LED**. 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.

2. 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 don’t have to replace the power supply. Just 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 44:

```
while [ $counter -lt 10 ]; 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.

What I2C Address is Used by Witty Pi?

Raspberry Pi communicates with the RTC chip (DS1337) on Witty Pi via I^{2}C protocol. The DS1337 chip has a fixed I^{2}C address: **0x68**.

If you have Witty Pi 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 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi ~ $
```

This address 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.

What GPIO Pins are Used by Witty Pi?

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

Please notice the same pin may have different pin number in different numbering systems. In documents we always use BCM numbering, while in the software (bash script) we use the wiring Pi numbering.

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

As you can see in the table, Witty Pi uses **GPIO-4**, **GPIO-17**, **GPIO-2 (SDA 1)**, **GPIO-3 (SCL 1)** and **GPIO-14 (TXD)**.

The usage of GPIO-4 and GPIO-17 are customizable. If you want to use other GPIO pins to do their job, you can follow this tutorial: http://www.uugear.com/portfolio/change-the-pin-that-used-by-witty-pi/

GPIO-2 and GPIO-3 are for I^2C communication between Raspberry Pi and the RTC chip (DS1337). I^2C devices are indentified by I^2C address, and they can share the I^2C pins as long as they have different I^2C addresses.

GPIO-14, as the TXD pin for serial communication, is quite special in Raspberry Pi. It will keep on HIGH state after system is up, and will become LOW after system has been shut down. It is used for detecting the system up or down, so Witty Pi can fully cut the power after system has been shut down. Since it is the only pin that could be used for this purpose, it is not customizable either.

Is Witty Pi Compatible with "Other Hardware"?

We have got a lot of emails asking this question, and the only difference is the "Other Hardware" got replaced by various 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. 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" doesn't use any GPIO pin that used by Witty Pi, and it doesn't use 0x68 I^2C address, then it should be compatible with Witty Pi.

If the "Other Hardware" uses GPIO-4 or GPIO-17, you still have chance to make it work, since Witty Pi doesn't have to use these two GPIO pins, and you can choose other pins to do the same job. However, if the "Other Hardware" uses GPIO-14 (TXD), there is a big chance that it could not work with Witty Pi together, unless it keeps pulling up GPIO-14 and only pull it down for short-time data transferring. Otherwise, when the "Other Hardware" pulls down GPIO-14 long enough, Witty Pi will think the system has been shut down and cut the power directly.

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.