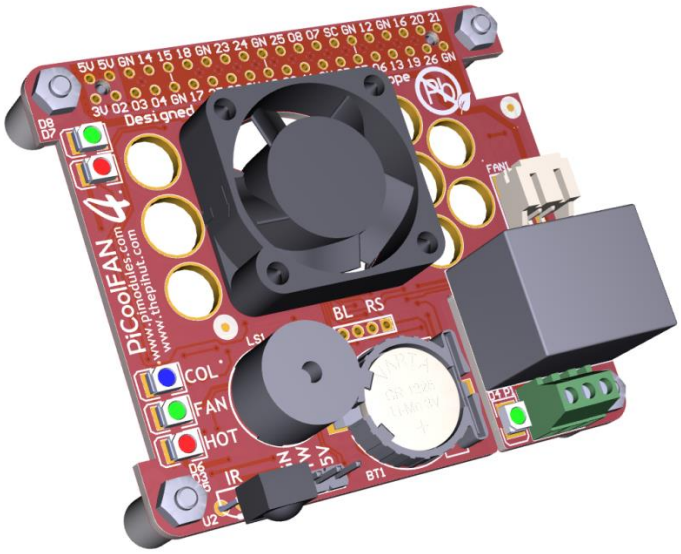
PiCoolFan 4

Advanced Active Raspberry Pi® Cooling System

with Hardware RTC

user LEDs, Sounder, Relay, IR and 1-wire interface



User Guide

Designed for the Raspberry Pi® 4 Model B

Compatible with

All former models of Raspberry Pi®

Designed by www.pimodules.com manufactured in co-operation with www.thepihut.com

Contents

System Overview

Introduction

The **PiCoolFAN4** is an **A**dvanced **A**ctive **C**ooling **S**ystem designed for the **Raspberry Pi 4** as an independent and autonomous cooling system, and can be used with all models of Raspberry Pi.

The **PiCoolFAN4** provides an enhanced cooling capability using an embedded fan and temperature sensor. The **PiCoolFAN4** temperature sensor is placed exactly 3.5 mm above the Raspberry Pi 4 CPU. This guarantees accuracy and continuity, measuring the system temperature as close to the CPU environment as possible.

The **PiCoolFAN4** does not need any software installation (*great for software such as KODI, which can also make use of the IR interface*) and requires simple assembly by the user. If required, users can make use of the provided software to enable greater control of the fan and cooling profile – including noise reduction.

The provided software allows the user to select either the Raspberry Pi's CPU temperature for fan threshold control, or the included temperature sensor on the PiCoolFAN4 PCB. In addition to this, the software also allows the user to select and edit different cooling profiles which allow either instant fan changes ('hard' cooling) or gradual changes in speed in-line with temperature changes ('mild' cooling).

The **PiCoolFAN4** does not use any GPIO pins - all communication with the Raspberry Pi is via I^2C. This leaves your GPIO pins free for your projects or other HATs, which can be stacked with the **PiCoolFAN4**.

The **PiCoolFAN4** is compatible with heatsinks when using an additional 8mm header. This allows users to benefit from both passive and active cooling at the same time.

Noise reduction is achieved through the use of a PWM fan. This allows the user to set the fan speed depending of the temperature, dramatically decreasing noise generation to an absolute minimum.

The **PiCoolFAN4** includes **Air Circulation Technology** – the fan forces cool air over the CPU, removes heat, then the warm air escapes through the special circle openings on the PCB. This design guarantees proper air circulation and maximizes the cooling efficiency even with low fan speeds (low noise).

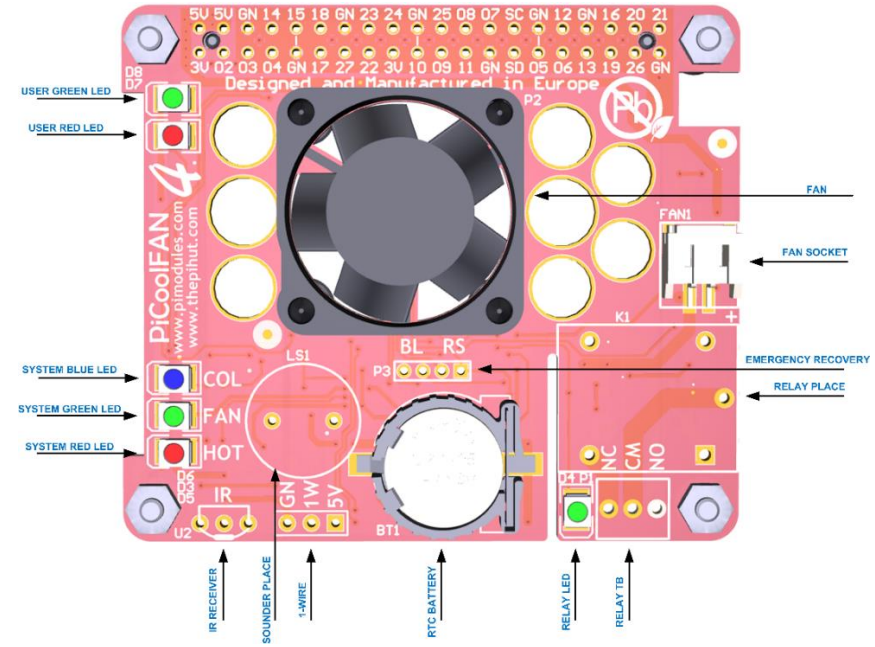
The **PiCoolFAN4** is standard equipped with:

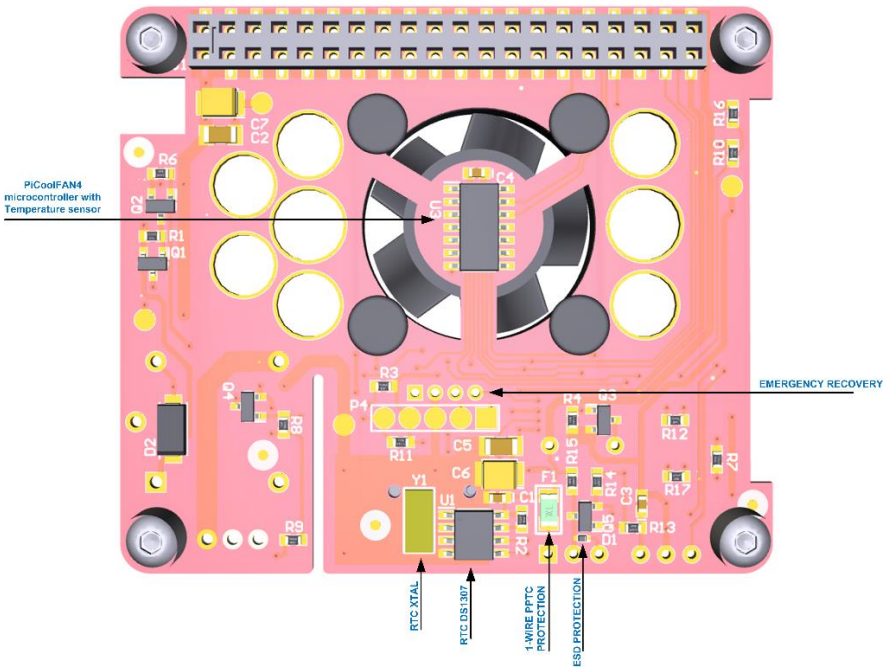
- Battery Backed Hardware Real-time Clock (RTC) DS1307 (coin battery 1220 available separately)
- 2 user programmable LEDs (Red and Green)
- 3 System LEDs (Cold, Hot and fan running)
- ESD Protected 1-wire interface (5V/3V)
- Infra-Red Receiver interface (Infra-red receiver available separately)

The **PiCoolFAN4** also includes mounting areas for the following features:

- High Current Relay Kit (available separately)
- Programmable sounder (available separately)

PiCoolFAN 4 PCB Layout





Set up Procedure

What's in the Bag?

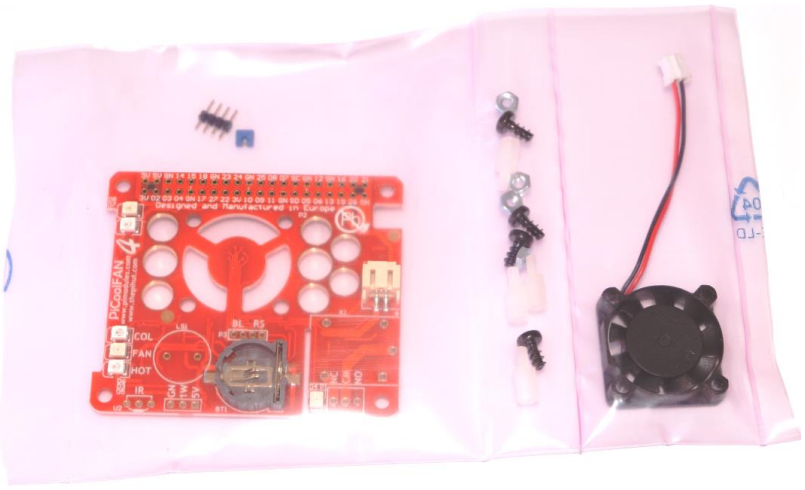
This package comes with everything you need to start using the **PiCoolFAN4** right out of the box. It's assembled, tested and contains all required accessories. Minor assembly is required by the user and no soldering is necessary.

Each Package contains the following parts:

1. Assembled **PiCoolFAN4** PCB
2. 2mm Header and Jumper
3. Necessary fan screws (4), bolts(4) and plastic spacers(4)

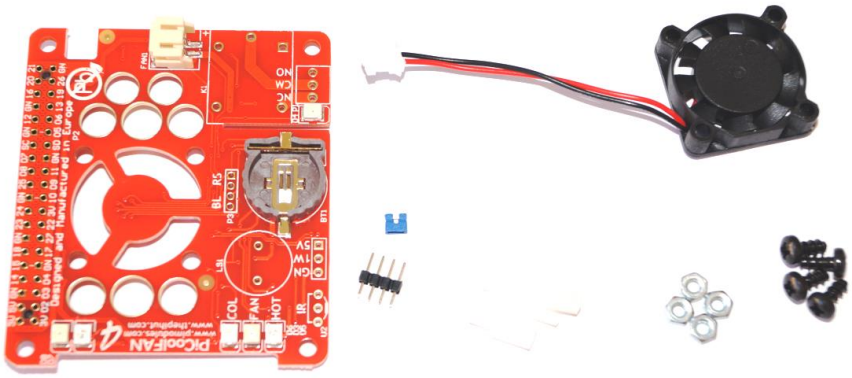
***Note**: The kit includes a 2mm header and jumper, but this is only required for the rare scenario where the user needs to reset the bootloader and is otherwise not necessary to be soldered.*

The **PiCoolFAN4** assembly requires a pair of scissors and a cross-head screwdriver.

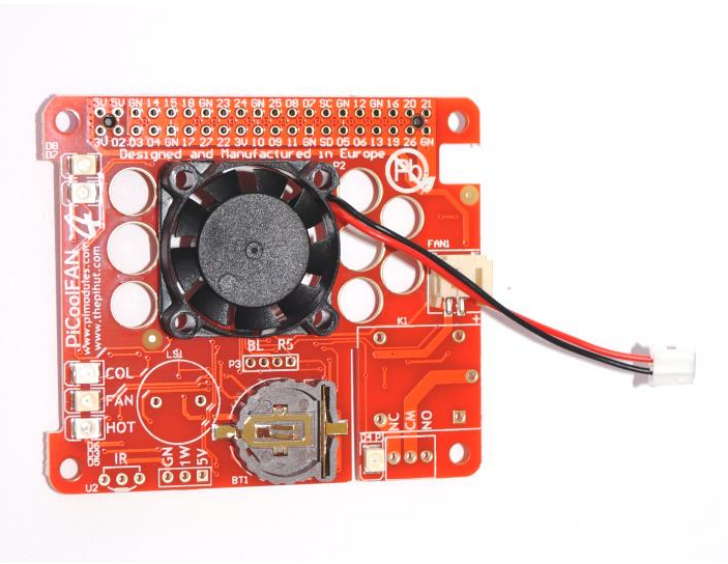


Kit Assembly

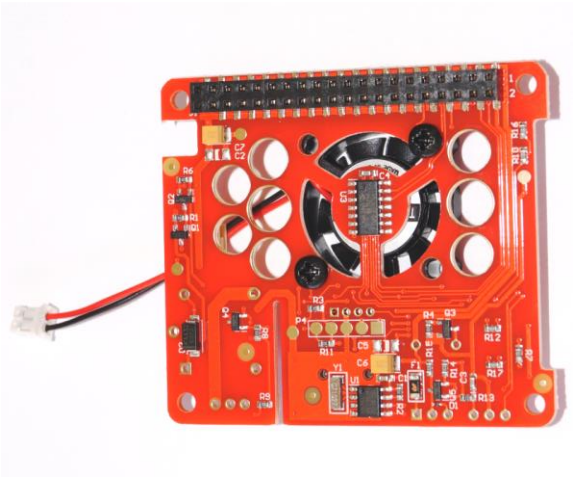
1. Cut the bag on upper side by scissors and remove components from the bag:



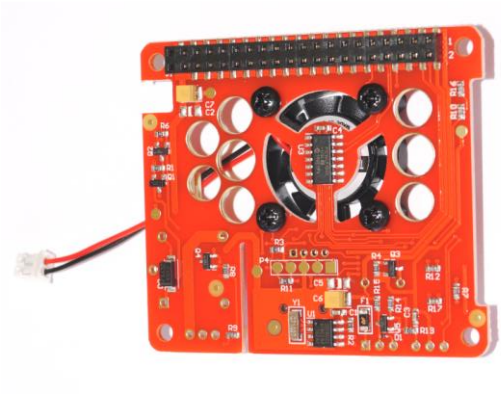
2. Place the fan on the top of the PCB with the cable to the right. It is important to place the fan this way, as there are slots on the PCB matching the fan position. Incorrect placement of the fan will reduce cooling performance:



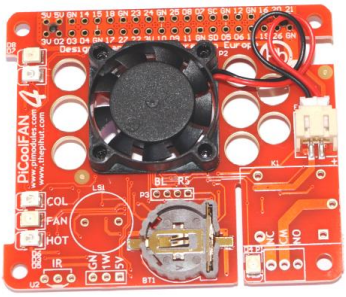
3. Turn the PCB and fan over, and screw the first two screws in opposing corners as per the image below. At this stage there is no need to full tighten the screws, just a couple of turns in order to keep the fan in place. Do not push the center of the fan – this could cause breakage.



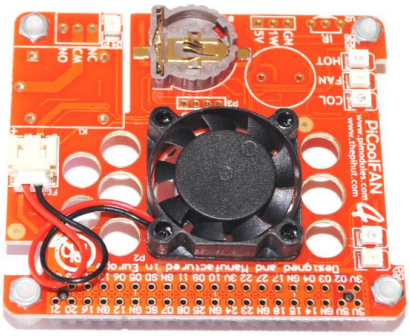
4. Fit the remaining fan screws, and then tighten all of them (finger-tight – do not overtighten!)



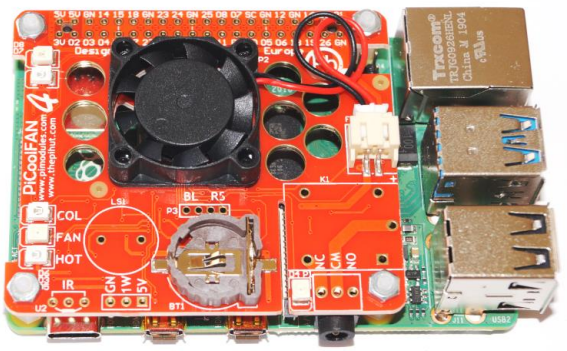
5. Turn the PCB over again and connect the fan cable to the socket:



6. Screw the 4 x 7mm spacers to the rear of the PCB with the provided nuts as per the image below:



7. Fit your **PiCoolFAN4** on to the Raspberry Pi 4. For basic usage there is no need for any software installation.



You are ready to go!

Using a Heatsink

It is possible to use a heatsink with the **PiCoolFAN4** and the combination of these two cooling tools provides excellent results.

To use a heatsink with the **PiCoolFan4**, the user needs to fit a GPIO header/extender to raise the height of the PCB (as standard, the PCB is close to the Raspberry Pi CPU to ensure the included temperature sensor can function as per design).

Users should ensure that the **PiCoolFAN4** is 3.5mm away from the heatsink, and test/adjust cooling profile thresholds to account for the different conditions (see page 13).

Alternatively, users can make use of the included software to use the Raspberry Pi temperature sensor for fan control (see page 18).

Basic Functionality

Factory Defaults

The **PiCoolFAN4** is set to a factory default state at production. These defaults are:

- **Cooling profile** = Mild
- **Temperature sensor** = PiCoolFAN4 temperature sensor
- **Temperature threshold** (*ttemp*) = 55°C *(50°C if using new software version V1.3)*
- **Temperature step** (*tstep*) = 1

***Note**: More information on these values and Cooling Profiles can be found on page 13*

Fan LEDs

The **PiCoolFAN4** is equipped with 3 System LEDs (Blue, Green, and Red) which are also enabled at this factory default state:

- If the current temperature is above the threshold, the **red LED** will light
- If the current temperature is below the threshold, the **blue LED** will light
- If the fan is running, the **green LED** will light.

***Note**: Green LED is driven by PWM. The intensity of the LED will increase with the fan speed for a visual indication of the current cooling requirements.*

The user can change this factory setup whenever they like (see page 11), and they can also revert back to factory settings at any time (see page 29).

Identifying the Current Setup

A simple script is provided which indicates the current setup alongside temperature readings.

The scripts can be downloaded at the following address and should be moved to the Raspberry Pi home directory:

https://pimodules.com/download/picoolfan4-daemons-and-python-scripts

The user will likely need to enable I^2C at this stage. This can be achieved by enabling I^2C within the interfacing section of the Raspi-config tool (***sudo raspi-config***).

With the scripts in place and I^2C enabled, run the following command:

sudo python pcf4_status.py

The output of the command should be similar to the image below (albeit values may be different):

```
PiCoolFAN 4 Internal registers status
  PCB/FM Version:.......................  60
  System Mode [0x08] smode:.............  00
  RPi Core Tempemparature:..............  48
  PCF4 Received Core Temp [0x00] ctemp:..  48
  PCF4 Internal Temp. [0x0E] itemp:.....  45
  PCF4 Threshold Temp. [0x01] ttemp:.....  45
  PCF4 Temp. Step [0x04] tstep:.........  01
  PCF4 Fan Speed [0x02] fspeed:.........  0500
  Running time in Seconds:..............  2
**********************************
```

A description of each line can be found below:

- **PCB/FM Version**: *Hardware and firmware version number*
- **System Mode:** *shows which sensor and cooling profile has been selected (see page 14)*
- **RPi Core Temperature:** *Temperature reading from the Raspberry Pi's temperature sensor*
- **PCF4 Received Core Temp:** *Temperature reading from the Raspberry Pi's temperature sensor*
- **PCF4 Internal Temp**: *Temperature reading from the PiCoolFAN4 temperature sensor*
- **PCF4 Threshold Temp**: *Current temperature threshold setting*
- **PCF4 Temp. *Step***: *Current temperature step for Mild cooling profiles (see page 13)*
- **PCF4 Fan Speed:** *Current fan speed*
- **Running time:** *Time the script has been running*

Tuning the Factory Default Temperature Threshold

The **PiCoolFAN4** is ready to be used without any software installation, making use of the temperature sensor on the **PiCoolFan4** PCB.

This factory default setup (threshold) has been tuned for use inside an official Raspberry Pi case (in the local environment of the production facility), however if you are using a different case, in a different environment, or just simply want to tune the settings - this can be achieved easily with the provided software.

Users who are not using a case with their Raspberry Pi are also strongly advised to tune the fan threshold.

Remember: The **PiCoolFAN4** temperature sensor is 3.5mm above the Raspberry Pi CPU. It is not a direct reading from the Raspberry Pi's CPU temperature sensor (*although this can also be set up – see page 18*).

Different ambient temperatures, draughts or enclosures usually require tuning of the temperature threshold to ensure the fan is activated at the right time.

1. Install the **PiCoolFAN4**
2. Close your system in desired case, with heatsink (*see page 8*) or without
3. Place your system in the desired location where it will be used
4. Run your system for 5 – 10 minutes
5. Run the **pcf4_status.py** script (*in your home directory as per page 10)*
6. Observe both the **PCF4 Internal Temp itemp** (*PiCoolFAN4 temperature sensor*) and the **RPi Core Temperature** (*Raspberry Pi CPU temperature sensor*) readings as indicated in the image below:

```
PiCoolFAN 4 Internal registers status
  PCB/FM Version:.......................  60
  System Mode [0x08] smode:.............  00
  RPi Core Tempemparature:..............  48
  PCF4 Received Core Temp [0x00] ctemp:..  48
  PCF4 Internal Temp. [0x0E] itemp:.....  45
  PCF4 Threshold Temp. [0x01] ttemp:....  45
  PCF4 Temp. Step [0x04] tstep:.........  01
  PCF4 Fan Speed [0x02] fspeed:.........  0500
  Running time in Seconds:..............  2
**************************************
```

These readings give an indication of the link between the direct temperature of the Raspberry Pi CPU and the **PiCoolFAN4** temperature sensor readings.

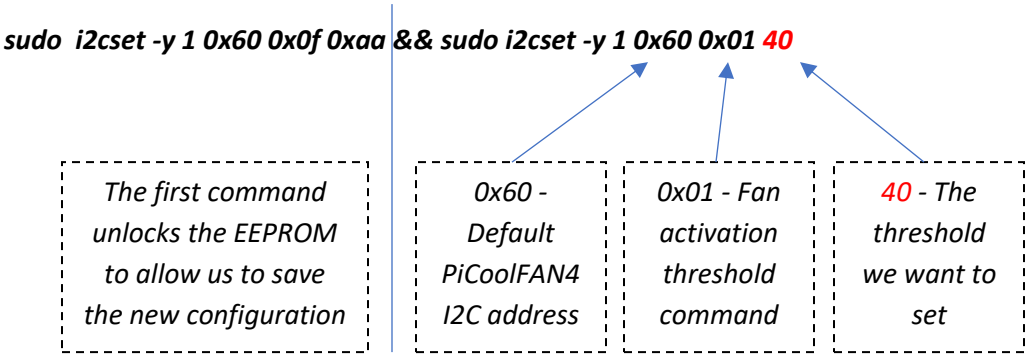
For ***example***, the Raspberry Pi temperature ***might*** be 60°C and the PCF4 temperature ***may*** be 50°C.

This indicates that the PCF4 sensor will likely always be around 10°C lower that the Raspberry Pi sensor – so to set a threshold for the Raspberry Pi sensor to 50°C, we would set the PCF4 threshold to 40°C.

To set a new threshold (whatever you decide), we use an I²C command, changing the red value to your desired temperature threshold:

sudo i2cset -y 1 0x60 0x0f 0xaa && sudo i2cset -y 1 0x60 0x01 40

We have broken down the command below so that you can understand what each part does:



This example is setting the threshold to 40°C, however the user can change this to whatever value is right for their requirements.

Remember, in this scenario based on the factory default setup, we're setting the PiCoolFAN4's temperature sensor reading for the threshold. Should you wish to use the Raspberry Pi's internal temperature sensor as the threshold sensor, see page 18.

Advanced Functionality

This section will guide the user in using the advanced functionality of the **PiCoolFAN4**, such as configuring the software to use the Raspberry Pi's temperature sensor, changing the cooling mode (hard or mild), adjusting the temperature threshold, fan speed changes and more.

Cooling Profiles

There are two type of cooling profile available to users:

Hard Cooling

A simple 'on/off' cooling profile.

The fan is initially off, and turns on at 100% speed when the temperature threshold is hit. When then temperature is lower than the threshold, it turns off again.

This profile is extremely effective at avoiding over overheating, however comes at the cost of fan noise.

Mild Cooling (recommended)

This cooling profile is more advanced in that it gradually speeds the fan up and down depending on how far the temperatures is from the threshold.

Users can adjust the settings of this profile to change the scale of the fan speed changes.

The Mild Cooling Profile is the recommend cooling method as it guarantees the lowest noise, best cooling and lowest current consumption, and as such, is the default profile setting from the factory.

Cooling Profile Parameters

Each Cooling Profile contains a set of three parameters which control how the profile reacts to temperature changes. These parameters are:

- Sensor and cooling profile selection (***Smode***)
- Temperature threshold (***Ttemp***)
- Fan speed change rate I.e. how the speed of the fan will change depending on temperature changes (***Tstep***)

Users can change any one (or all) of these parameters to customize cooling to their exact needs. All changes are stored in the EEPROM i.e. the settings are saved, even when the Raspberry Pi is turned off and on again.

This section will describe the different options for each parameter, then describe how to structure and issue this command to the Raspberry Pi.

Smode

System Mode (**smode**) defines both the sensor to be used and the cooling profile to be used with that sensor.

The following values are available:

smode value	Temperature Sensor	Cooling Profile
0x00	PiCoolFan4	Mild
0x02	Raspberry Pi	Mild
0x10	PiCoolFan4	Hard
0x12	Raspberry Pi	Hard
0x20	Manual System - User needs to start/stop the fan manually (via commands) and check the temperature manually	

***Note**: In order to use the Raspberry Pi temperature sensor, some software setup is required. Please see page 18.*

***Note**: 0x00 is the factory default **smode** setting*

Ttemp and Tstep

Ttemp defines the temperature threshold to be used in the cooling profile.

Tstep defines the number of steps (temperature in degrees) it takes to increase/decrease fan speed to the next range.

When used with a **Hard** cooling profile, **Ttemp** is simply the point at which the fan turns on/off. **Tstep** is not used with Hard cooling profiles.

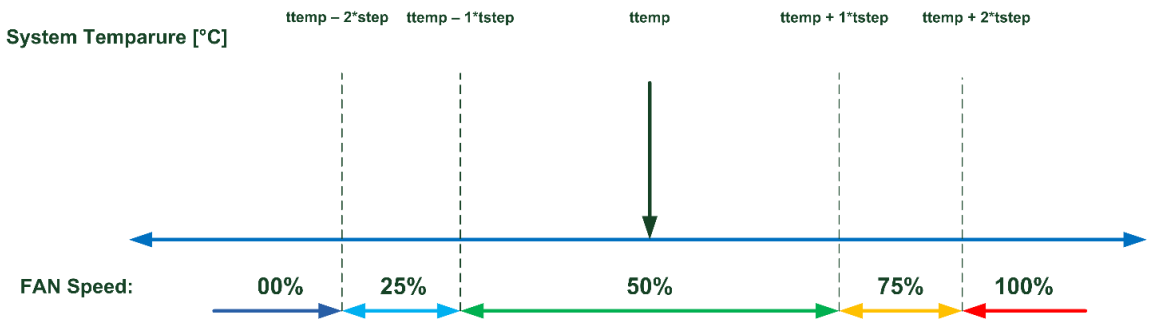
When used with a **Mild** cooling profile, the **Ttemp** is the center point at which the gradual fan changes are based on, which works in conjunction with **Tstep**.

In the example below, the **ttemp** is on the center of temperature monitoring window and **tstep** is set to 1.

When the temperature moves one degree lower or one degree higher than **ttemp**, the fan runs at 50% of its speed (lower noise). This is because tstep is set to 1.

If the temperature continuous to increase, the fan speed will increase 75% of its speed, as this is another 1 'step' as defined by **tstep** (which is set to 1). The fan will eventually run at 100% if the temperature increases by another step.

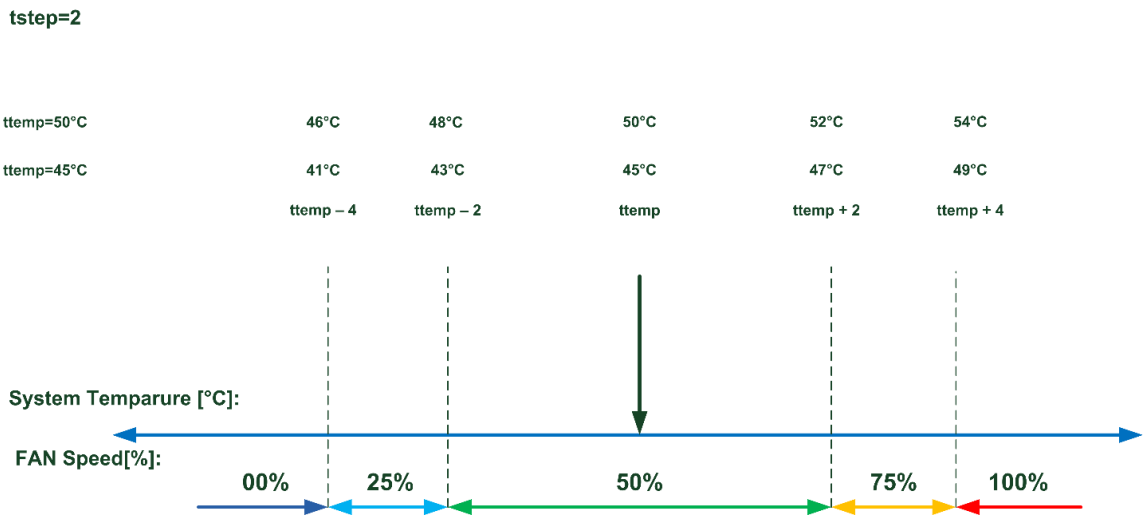
The same works for decreasing temperature with the same steps/range, lowering fan speed until it eventually turns off the fan when not required. This is ideal for minimizing noise output.



The user can change the **ttemp** to any value, effectively moving the window up or down in the temperature scale.

Tstep can also be changed, to increase/decrease the window of temperature increase/decrease before moving to the next fan speed.

Here is an example of **Ttemp** set to 45°C and **Tstep** set to 2:



How to set a Cooling Profile & Parameters

Each Cooling Profile parameter can be set using an I²C command, and users can choose to change one or many (or all) settings to meet their needs.

Most of the command is the same for each parameter, with just a different I²C address for each parameter entered at the end (along with the desired setting).

Setting Smode

Smode is set by using **0x08** in the I²C command followed by the desired **setting**:

sudo i2cset -y 1 0x60 0x0f 0xaa && sudo i2cset -y 1 0x60 0x08 0x10

In this example, the setting is 0x10 which uses the **PiCoolFAN4** sensor and a Hard cooling profile.

Other **smode** settings can be seen in the table on page 14.

Setting Ttemp

Ttemp is set by using **0x01** in the I²C command followed by the desired **temperature**:

sudo i2cset -y 1 0x60 0x0f 0xaa && sudo i2cset -y 1 0x60 0x01 45

In this example, we use 45 which sets the temperature threshold to 45 degrees.

Setting Tstep

Tstep is set by using **0x04** in the I²C command followed by the desired **setting**:

sudo i2cset -y 1 0x60 0x0f 0xaa && sudo i2cset -y 1 0x60 0x04 3

In this example, we are setting the step to 3. Step values permitted are 1, 2, 3, 4 or 5.

Using the Raspberry Pi Temperature Sensor

In most cases using the **PiCoolFAN4** sensor – once set up for the user's case/environment – provides excellent temperature monitoring without the overhead of running additional software (daemon).

However, users can select to use the Raspberry Pi temperature sensor for temperature monitoring rather than the **PiCoolFAN4** sensor. This option accommodates users in specific scenarios/environments where the **PiCoolFan4** sensor is not practical.

Enable the daemon

To enable the software to use the Raspberry Pi sensor as the temperature monitor (as part of the cooling profile), the following steps are required:

1. Copy provided python **pcf4.py** script to the root directory, if not already done so.
2. Create a configuration file that tells SystemD what we want it to do and when:

   sudo nano /lib/systemd/system/pcf4.service

3. Add the text below to this file, and then exit by saving it:

   [Unit]

   Description=PiCoolFAN4 Service supplying RPi core temperature to it

   After=multi-user.target

   [Service]

   Type=idle

   ExecStart=/usr/bin/python /home/pi/pcf4.py

   StandardOutput=inherit

   StandardError=inherit

   Restart=always

   [Install]

   WantedBy=multi-user.target

4. Setup the file permissions

   sudo chmod 644 /lib/systemd/system/pcf4.service

5. Reload, enable, start the daemon

sudo systemctl daemon-reload

sudo systemctl enable pcf4.service

sudo systemctl start pcf4.service

6. Reboot the Pi and your custom service should run:

sudo reboot

Once this service is activated, the user has the option to use both Cooling Profiles (Mild or Hard) based on the Raspberry Pi temperature sensor (see page 13).

***Note**: Even with the Daemon running, the user can still use the **PiCoolFAN4** internal sensor if desired.*

***Note**: To check your current system setup, please see page 10*

Low Noise Fan Cooling Techniques

Keeping your Raspberry Pi cool whilst also keeping fan noise to a minimum can be challenging.

Users who aren't concerned with noise can of course run their fan at full speed for the best cooling performance, however most users will benefit from the techniques below to strike a good balance:

1. In most cases there is no need to run the fan at full speed at all times. This is not necessary as after one step, the Raspberry Pi has been cooled and additional fast cooling is not usually required.
2. Use the best cooling level that fits the needs of your application, that balances good cooling and a low level of noise.
3. Never select the **ttemp** (threshold temperature) at a very low level. This will cause the fan to run at full speed for no reason.
4. Remember that even if your fan is running at low speed, The **PiCoolFAN4** is a smart device, and if needed, will run the fan at 100% speed (if temperatures hit that level).
5. Try to select an appropriate **ttemp** after observing your system temperatures within your specific case, environment and system usage scenario.
6. The **PiCoolFAN4** Cooling Profiles are designed to run the fan at 50% speed most of the time. Try to select a **ttemp** and **tstep** that keeps your system cool and fan speed low for the majority of the running time. Larger **tstep** values increase the width of the 50% window but care should be taken to ensure the window does not allow high temperatures (linked to **ttemp**).

I²C Address Table

This table details the I²C addresses used for the various features/setting for the PiCoolFan4:

Address	Name	Explanation
0x00	ctemp	**C**ore Raspberry Pi **temp**erature – user by software (Daemon) to provide exact Raspberry Pi temperature if Raspberry Pi sensor is configured to be used. User should use the Register for writing, however can read for monitoring purposes. In decimal, and Celsius degrees. **Do not use HEX codes**
0x01	ttemp	**T**hreshold **Tem**perature for fan activation
0x02	fspeed	**F**AN **Speed** used for fan speed monitoring or setting up of fan speed. Allowed values are: • 250 (means 25%) • 500 (means 50%) • 750 (means 75%) • 1000 (means 100%)
0x04	tstep	**T**emperature **Step** defines speed of the fan which will be running in the Mild Cooling Profiles. Allowed values are: • 1 (default) • 2 • 3 • 4 • 5
0x05	PCF4_IO	Used for PCF4 I/O peripherals handling. Described in details in the following chapters
0x06	bfreq	**B**uzzer (sounder) **Freq**uency. Allowed values are 0 – 10000 Hz
0x08	smode	**S**ystem **Mode** setup various PCF4 modes. Described in detail within this document
0x0A	version	Specify V**ersion** of PCB and firmware
0x0E	itemp	**I**nternal Sensor **Temp**erature
0x0F	EEPROM unlock	Used to unlock PCF4 EEPROM when written (for security reasons)

Additional Feature Setup

HW Real Time Clock

As it has been mentioned above there is no need to do anything in software installation for the basic operation of the **PiCoolFAN4.** However this small PCB is equipped with plenty of additional features, that; if used, user can benefits of their advantages. The most frequently used is the Hardware Battery Backed Real Time Clock. Follow below description in order to make it easy used on your system.

After plug-in the **PiCoolFAN4** and start the Raspberry Pi, you can see existing the following addresses on the I^2C space simply by typing the following command:

sudo i2cdetect -y 1

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

If an RTC has been installed previously the address 0x68 will be visible as UU, or if it is the first installation it will be visible as 68. In addition the other visible address will be the 60 (0x60) that is the basic address where Raspberry Pi is interacting with the **PiCoolFAN4**. The address 0x60 could be different if other alternative firmware has been loaded. The available addresses are : 0x30, 0x40, and 0x50. Please kindly notice that these addresses if used are part of Advanced Installation procedures.

In order to use HW RTC user will need to fit-in a coin backup battery CR1220 or CR1225. It is 3V Lithium battery. User can use any other locally ordered similar batteries, however as we and all distributors are selecting always the best parts therefore we suggest to buy from them directly when buying the **PiCoolFAN4**.

Then, after coin battery installation it is needed to follow the standard HW RTC installation. The used IC is the DS1307, here below is described a sample installation of it.

Please kindly notice that functionality of Hardware RTC is independent of any other functionalities of **PiCoolFAN4**.

In order to make the HW RTC installed follow the below procedure step by step.

1. Ensure to run below line

   sudo apt-get -y install i2c-tools

2. Edit by running the following line

   sudo nano /etc/modules

   and check, make sure to have the following items in the file and add what is missing:

   i2c-bcm2708

   i2c-dev

   rtc-ds1307

3. Edit by running the following line

   sudo nano /boot/config.txt

4. and add the following to this file:

   enable_uart=1

   dtoverlay=i2c-rtc,ds1307

5. Edit by running the following line

   sudo nano /etc/rc.local

6. and add the following line before "**exit 0**"

   sleep 4; hwclock -s &

7. Reboot system by

   sudo reboot

8. Remove the **fake-hwclock** which interferes with the RTC **hwclock**

   sudo apt-get -y remove fake-hwclock

   sudo update-rc.d -f fake-hwclock remove

9. Run

   sudo nano /lib/udev/hwclock-set

10. and comment out these three lines**:**

```
#if [ -e /run/systemd/system] ; then
# exit 0
#fi
```

11. Run **date** to verify the time is correct.

12. Plug in Ethernet or WiFi (if not plugged before) to let the Raspberry Pi sync the right time from the Internet. Once that's done, run:

```
sudo hwclock -w
```

13. to write the time, and another

```
sudo hwclock -r
```

13. to read the time

That's it! Next time you boot the time will automatically be synced from the RTC module.

Sound Generation System

The **PiCoolFAN4** can be optionally equipped with Simple Sound Generation System. It is can be used dedicated user applications offering the whole range of acoustic frequencies full programmable by user.

There is 1 register **bfreq** that are responsible for the generating sound. The user needs to enter the requested frequency and just after that, sounder will be generated it. If the user need to stop frequency generation, need just to enter 0, so the sounder will stop the sound generation.

This simply mechanism allows the user even to play music.

The default value of Frequency is 0. Data can be entered in HEX or DEC format, but need to be always word (16 bits).

0x0E	**bfreq**	Word	Common	R/W	**Frequency of sound in Hz**

Example of use

sudo i2cset -y 1 0x60 0x06 0x0000 w Deactivate permanently the buzzer (no sounds will be played)

sudo i2cset -y 1 0x60 0x06 1000 w Generating 1000Hz frequency

sudo i2cset -y 1 0x60 0x06 1047 w Generating the C (1047 Hz) note

sudo i2cset -y 1 0x60 0x06 0500 w Generating 500Hz frequency

sudo i2cset -y 1 0x60 0x06 0000 w Deactivate permanently the buzzer (no sounds will be played)

High Current Relay

The **PiCoolFAN4** can be optionally equipped with High Current Relay. This Relay is supported on PCB by Green LED that, when it lit inform user that Relay is ON for optical monitoring.

The **PiCoolFAN4** High Current Relay kit contains:

- High Current Relay
- 40 pin Stack Header
- 4 x 17 mm spacers
- 4 x 25mm screws/nuts
- 1 x 3 way Terminal Block

Due to construction of the **PiCoolFAN4** PCB **we do not recommend** to use this Relay for switching of higher voltages/currents other than **32 VDC/3.5A** per switching contacts.

As each High Current Relay is equipped in addition with high 17mm spacers, due to used spacers the internal **PiCoolFAN4** temperature sensor is placed too far from the Raspberry Pi CPU, therefore user need to use Raspberry Pi core temperature measurements and the provided Daemon to have a proper fan activity.

It is absolutely not allowed to use the Integrated Relay for switching 220 V AC at any current, even very low due to security reasons.

For **PiCoolFAN4** I/0 Embedded Handler (User LEDs and Relay) is using a common register, that each bit is dedicated to different I/O device. Therefore bit 2nd is assigned to the Relay.

0x05	**I/O Relay**	Byte	Common	R/W	**Relay ON/OFF 0b00000100**

Example of use

sudo i2cset -y 1 0x60 0x05 0x04 activate the Relay (ON)

sudo i2cset -y 1 0x60 0x05 0x00 *de*-activate the Relay (OFF)

User LEDs

The **PiCoolFAN4** is standard equipped with 2 LEDs for user applications: Green and Red. They can be switched ON or OFF each one respectively.

Therefore bit 0 and 0th and 1st is assigned to the user LEDs.

0x05	**I/O LED**	Byte	Common	R/W	**R**ed **L**ED **ON/OFF 0b0000000**1

Example of use

sudo i2cset -y 1 0x60 0x05 0x01 activate the **Red LED** (ON)

sudo i2cset -y 1 0x60 0x05 0x00 *de*-activate the **Red LED** (OFF)

0x05	**I/O LED**	Byte	Common	R/W	**G**reen LED **ON/OFF 0b000000**1**0**

Example of use

sudo i2cset -y 1 0x60 0x05 0x02 activate the **Green LED** (ON)

sudo i2cset -y 1 0x60 0x05 0x00 *de*-activate the **Green LED** (OFF)

the user can switch all LEDs and Relay together by sending common commands or separately. If user like to keep i.e. one I/O and switch another one, the “OR” operation is required.

Infra-Red Receiver Interface

The **PiCoolFAN4** is equipped with IR receiver interface. It is directly routed to the GPIO18. It can be used for any application like Media Player. If IR Receiver is not soldered, then the GPIO18 is free for any other application. An excellent tutorial how to use IR is provided by www.thepihut.com at below link:

https://thepihut.com/blogs/raspberry-pi-tutorials/raspberry-pis-remotes-ir-receivers

ESD protected 1-wire Interface

The **PiCoolFAN4** is equipped with ESD protected 5/3V interface. It is directly routed to the GPIO4. It can be used for any application user like. If the 1-wire device is not used (soldered), then the GPIO4 is free for any other application. An excellent tutorial how to use 1-wire is provided by www.thepihut.com at below link:

https://thepihut.com/blogs/raspberry-pi-tutorials/ds18b20-one-wire-digital-temperature-sensor-and-the-raspberry-pi

Bootloader Reset Pins (4-pin header)

The **PiCoolFAN4** bootloader is invoked by a PICo (I^2C) command, and then the new firmware is uploaded by Serial Port. However very, very unusual, for some undefined reason the firmware cannot enter the bootloader mode. This case is not possible in any normal use, just in a case that firmware is not reacting, our company added the additional hardware bootloader invoking method. On that case user need to solder the 4 pin header and put the jumper on the place **BL**. Then restart the system. Alternatively you can short the **RS** pins if not like to restart the system – this will cause reset of **PiCoolFAN4** and enter the bootloader. **PiCoolFAN 4** will enter the boot loading mode (User Green LED lit) and remain on this until new firmware properly uploaded. Then user should use this command for new/old firmware upload. After firmware uploading user need to remove the jumper from the **BL** points and reboot the system and run factory defaults after that.

sudo python PCF4_serial0_9600_FU1.0.py -v -f PiCoolFAN.hex

Setting Factory Defaults

There is a Factory Defaults set, that user can recall at any time. The Factory Defaults are the following:

- **Smode=00**
- **Ttemp=55** (changed in the new firmware v1.3 to 50)
- **Tstep=1**

If user like to recall Factory Defaults the following command need to be executed

sudo i2cset -y 1 0x60 0x08 0xaa

Bootloader Operations

Nowadays, all embedded and computer based devices are under continuously software upgrading. We can see it on the continuously Raspberry Pi new NOOBS releases, with many, many upgrades/updates. As the **PiCoolFAN4** use embedded processor, bootloader is a mechanism that always user to have always up-to-date system with firmware. The bootloader is used for uploading of a new version of firmware is such exist, or for upgrading of firmware that is handling different I^2C address. On such request user need to make sure that required firmware as also bootloader script is located on the Raspberry Pi root, and if so, execute the following command. Always replacing the ***PiCoolFAN.hex*** with a current name of the firmware release.

sudo i2cset -y 1 0x60 0x08 0xbb && sleep 1 && sudo python PCF4_serial0_9600_FU1.0.py -v -f PiCoolFAN.hex

Executing bootloader command the following message will be visible on the terminal. It's important to note that boot loading procedure is protected and if for any case will be interrupted or firmware not uploaded properly, **PiCoolFAN4** will remain in the boot loading mode until uploading will be completed properly. There is no way to exit from bootloader mode until properly firmware will be loaded. On such case a green user LED will all the time lit, and user need to run only the second part of bootloader command like shown below.

sudo python PCF4_serial0_9600_FU1.0.py -v -f PiCoolFAN.hex

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ sudo i2cset -y 1 0x60 0x08 0xbb && sleep 1 && sudo python PCF4_serial0_9600_FU1.0.py  -v -f PiCoolF
AN_V1_2.hex
Validating firmware: OK
Checking communication with bootloader: OK
Uploading firmware: 0% ******** 3.0% ********* 7.0% ********* 11.0% ********* 15.0% ********* 19.0% ********* 23.0% **
******* 27.0% ********* 31.0% ********* 34.0% ********* 38.0% ********* 42.0% ********* 46.0% ********* 50.0% ********
* 54.0% ********* 58.0% ********* 62.0% ********* 65.0% ********* 69.0% ********* 73.0% ********* 77.0% ********* 81.0
% ********* 85.0% ********* 89.0% ********* 93.0% ********* 96.0% ****** Done uploading...
Invoking factory defaults of PIco...
send factory reset command ...
ALL Done :) Ready to go...
pi@raspberrypi:~ $
```

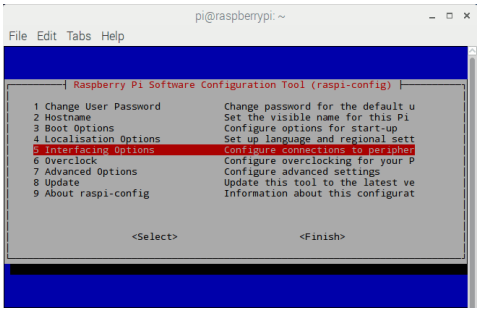
In order to execute boot loading process user must take care to have available and free serial port on the Raspberry Pi. That mean the Serial Port on pins GPIO14 and GPIO15 must be free from any other hardware HAT, and activated in system. After successfully boot loading process **PiCoolFAN4** is self-testing (including the Relay interface) so user need to have disconnected any loads from the Relay Terminal Blocks to avoid unexpected results.

Serial Port activation and checking

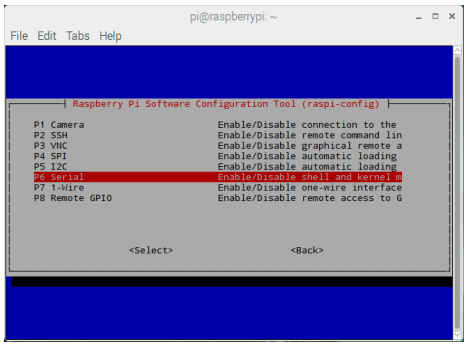
In order to use bootloader, user need to have activated and free Serial Port on Raspberry Pi. The below procedure is showing how to make sure that Serial Port is free and available for Bootloader process.

sudo raspi-config

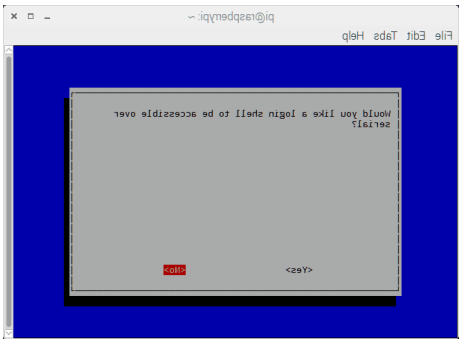
Select -> **Interfacing Options**



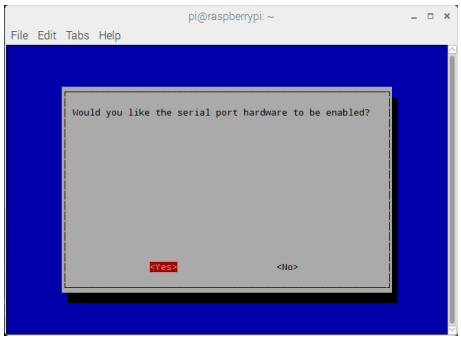
After selecting Interfacing option, select **Serial** option to enable UART



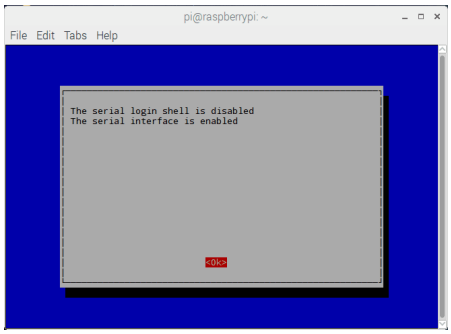
Then it will ask for login shell to be accessible over Serial, select **No** shown as follows.



At the end, it will ask for enabling Hardware Serial port, select **Yes**,



Finally, our UART is enabled for Serial Communication on RX and TX pin of Raspberry Pi.



Then, reboot the Raspberry Pi.

Using Different I²C address

As mentioned before the **PiCoolFAN4** can be used with different addresses other than 0x60. This is because some HATs are using different addresses and there is a possibility to have a conflict between them and **PiCoolFAN4**.

The available addresses are the 0x30, 0x40, and 0x50.

All handling of the **PiCoolFAN4** is identical, the only changes are needed are to bootload new firmware addressed to selected new address.

Due to ultra-small memory footprint on the used micro controller (due to price reasons) in the **PiCoolFAN4**, there is no space to implement I²C selection on a single firmware. Therefore the final version of the **PiCoolFAN4** firmware is released for all available address separately. When doing boot loading user need to know that some functionalities (factory setup) cannot run as they are referred to different addresses. Similar with Daemon and status monitoring python script. These programs are provided in a single package together with firmware(s).

As using of the different addresses is advanced usage user need to be first familiar with the product before start experimenting with different I²C addresses and different Daemons.

There have been implemented the following **PICo** addresses assigned to the following entities:

- 0x60 default I²C address
- 0x50 alternative I²C address with firmware upload (used only if other HAT occupy the 0x60)
- 0x40 alternative I²C address with firmware upload (used only if other HAT occupy the 0x60)
- 0x30 alternative I²C address with firmware upload (used only if other HAT occupy the 0x60)

Common Problems & Solutions

The temperature show on the PiCoolFAN4 is much different than the Raspberry Pi core

The Internal Temperature sensor is a very simple and not very accurate sensor however enough for such application. It measures temperature of the environment around of the Raspberry Pi ® CPU so it cannot be the same.

The Temperature of the Raspberry Pi is going high without fan activation, what should I do?

The threshold temperature has been set 55 °C (changed in the new firmware V1.3 and up to 50°C), the temperature threshold is fine if the Raspberry Pi is closed in a case (especially in the official one), however it could be too high (so fan not activate) for Raspberry Pi running alone outside of case. Then user need to decrease this threshold to 50 °C or even 45 °C, to have a proper cooling process. Such post installation process is needed in order to set the best threshold temperature for your exact system. Alternatively you can use the daemon then fan will be activated on very accurate CPU temperature.

sudo i2cset -y 1 0x60 0x0f 0xaa && sudo i2cset -y 1 0x60 0x01 45

Why does the PiCoolFAN4 internal temperature sensor have a big difference?

The PiCoolFAN4 temperature sensor is not accurate, it is internal micro controller temperature sensor, and it can be used on as an indicator and not as an exact measure tool. However it is very good for such simple application as triggering of fan. User need to adjust little bit the threshold if like to have more exact indications. The reason we selected this sensor is the price. It is not possible to have low price and very exact external temperature measurements. If user still required to have very exact temperature activation, then the simplest way is to use daemon and Raspberry Pi temperature sensor. This approach is also low cost however, need to have installation the daemon.

What is better to use as a temperature sensor: the Internal PiCoolFAN4 or Raspberry Pi?

On our opinion such simple application is not needed to have an accurate temperature measurements. Maybe user will need to make some threshold adjustment according to environment where his system is placed, but still it is. If really the user like to have very accurate temperature activation of fan, then the penalty is to install the Daemon that will push the CPU actual temperature to the **PiCoolFAN4** and adjust it exactly to be very near to the real conditions. We are offering both solution at the same very low cost, so it is up to user to decide what better fits-in to his needs.

The system is not entering the boot loading process

The boot loading process is a very easy task. However many people report on our support, problems. This is because most of them is not following exact our recommendations, so not all required steps are fulfilled. If your bootloader is not entering (you have a message from bootloader script) the most possible reason is that you typed something wrong is bootloader command or the Serial Port of the Raspberry Pi has been

not activated. Please follow exactly the Serial Port activation procedure described in our manual. Also there is a strong recommendation, before you start doing boot loading, install and use this product to be more familiar with it. Unfortunately many, many people the first think they are doing is to boatload of the new firmware. That is definitely not the best approach of handling of a new unknown device, even it is simple one.

During boot loading process system hang up (or for any reason and not completed uploading)

The implemented in the **PiCoolFAN4** bootloader is very secure. If the bootloaded new firmware not finished successfully, **PiCoolFAN4** will not start. Will wait for repeated loading until completed properly. The Green User LED will lit then. If something like that happen, then user need to repeat the bootloader command without recalling (bootloader itself) it, like below. Remember to write the proper and current firmware name instead this typed.

sudo python PCF4_serial0_9600_FU1.0.py -v -f PiCoolFAN.hex

Document Revisions

Version	Date	Modified Sections	Comments
N.A.	02/09/2019	N.A.	First Preliminary Public Document Release
N.A.	06/09/2019	N.A.	Updated Version
N.A.	05/11/2019	Corrected some mistakes in the manual text. Added section for heatsink usage	No update firmware released – as not needed When threshold temp is set, must be done in decimal not in hex **0x45**, but just **45 should be used**

Firmware Revisions

Version	Date	Modified Sections Bugs fixes	Comments
1	02/09/2019	N.A.	First Preliminary Public Release
3	06/09/2019	1. Size Compacted 2. Corrected bug with store in internal EEPPROM 3. Default value for **ttemp** is set to 50 °C	Updated Version Release
4	08/09/2019	1. Corrected bug with I^2C timing 2. Added maximum for the **ttemp**, that cannot be exceeded by user setting, to 75°C	Updated Version Release