API for GPIO Library

The Jetson GPIO library provides all the public APIs provided by the RPi. GPIO library. The following discusses the usage of each API:

1.import library

To import the Jetson.GPIO module, use:

import Jetson.GPIO as GPIO

In this way, you can refer to this module as GPIO in the rest of the application. Modules can also be imported using the name RPi. GPIO replaced Jetson. GPIO is used to use existing code from the RPi library.

2.Pin number

The Jetson GPIO library provides four methods for numbering IO pins. The first two correspond to the modes provided by the RPi. GPIO library, namely BOARD and BCM, respectively referencing the pin numbers of the 40 pin GPIO connector and Broadcom SoC GPIO numbers. The other two modes CVM and TEGRA_ SOC uses strings instead of numbers, which correspond to signal names on the CVM CVB connector and Tegra SoC, respectively.

To specify which mode you are using (mandatory), use the following function calls:

GPIO.setmode(GPIO.BOARD)# or

GPIO.setmode(GPIO.BCM)# or

GPIO.setmode(GPIO.CVM)# or

GPIO.setmode(GPIO.TEGRA_SOC)

To check the set mode, you can call:

mode = GPIO.getmode()

This mode must be GPIO.BOARD, GPIO.BCM, GPIO.CVM, GPIO.TEGRA_ SOC or none.

3. Warning

The GPIO you are trying to use may already be in use outside of the current application. In this case, if the GPIO configuration used is any value other than the default direction (input), Jetson The GPIO library will issue a warning to you. If you attempt to clean up before setting the mode and channel, it will also warn you. To disable warnings, use:

GPIO.setwarnings(False)

4.Set channel

Before being used as input or output, the GPIO channel must be set first. To configure a channel as input, call:

GPIO.setup(channel, GPIO.IN)

To set the channel to output, call:

GPIO.setup(channel, GPIO.OUT)

You can also specify an initial value for the output channel:

GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)

When setting one channel as output, multiple channels can also be set at once:

channels = [18, 12, 13]

GPIO.setup(channels, GPIO.OUT)

5.input

To read the value of a channel, use:

GPIO.input(channel)

这将返回GPIO.LOW或GPIO.HIGH。

6.输出

This will return GPIO.LOW or GPIO.HIGH.

GPIO.output(channel, state)

The status can be GPIO.LOW or GPIO.HIGH.

You can also output to a channel list or tuple:

channels = [18, 12, 13] # or use tuples

GPIO.output(channels, GPIO.HIGH) # or GPIO.LOW

set first channel to HIGH and rest to LOW

GPIO.output(channel, (GPIO.LOW, GPIO.HIGH, GPIO.HIGH))

7.clear

At the end of the program, it is best to clean the channels in order to set all pins to the default state. To clear all used channels, use:

GPIO.cleanup()

If you do not want to clear all channels, you can also clear individual channels or channel lists or tuples:

GPIO.cleanup(chan1) # cleanup only chan1

GPIO.cleanup([chan1, chan2]) # cleanup only chan1 and chan2

GPIO.cleanup((chan1, chan2)) # does the same operation as previous statement

8.Jetson module information and library version

To obtain information about the Jetson module, please use/read:

GPIO.JETSON_INFO

This provides the following keys for the Python dictionary: P1_ REVISION, RAM, REVISION, TYPE, MANUFACTURE, and PROCESS. All values in the dictionary are strings, but P1_ REVISION is an integer.

To obtain information about library versions, please use/read:

GPIO.VERSION

This provides a string in XYZ version format.

9.interrupt

In addition to busy polling, the library also provides three methods for monitoring input events:

wait_for_edge（）

This function blocks the calling thread until the provided edge is detected. This function can be called as follows:

GPIO.wait_for_edge(channel, GPIO.RISING)

The second parameter specifies the edge to be detected, which can be GPIO.RISING, GPIO.FALLING, or GPIO.BOTH. If you only want to limit the waiting time to the specified time, you can choose to set a timeout:

timeout is in milliseconds

GPIO.wait_for_edge(channel, GPIO.RISING, timeout=500)

This function returns the channel that detected the edge; If a timeout occurs, return none.

event_detected（）

This function can be used to regularly check if any events have occurred since the last call. This function can be set and called as follows:

set rising edge detection on the channel

GPIO.add_event_detect(channel, GPIO.RISING)

run_other_code()

if GPIO.event_detected(channel):

do_something()

As before, you can detect events for GPIO.RISING, GPIO.FALLING, or GPIO.BOTH.

Run callback function when edge is detected

This feature can be used to run a second thread for the callback function. Therefore, in response to the edge, the callback function can run concurrently with the main program. You can use this feature in the following ways:

define callback function

def callback_fn(channel):

print("Callback called from channel %s" % channel)

add rising edge detection

GPIO.add_event_detect(channel, GPIO.RISING, callback=callback_fn)

If necessary, multiple callbacks can also be added:

def callback_one(channel):

print("First Callback")

def callback_two(channel):

print("Second Callback")

GPIO.add_event_detect(channel, GPIO.RISING)

GPIO.add_event_callback(channel, callback_one)

GPIO.add_event_callback(channel, callback_two)

In this case, these two callbacks run sequentially, rather than simultaneously, as only the thread runs all callback functions.

To prevent multiple calls to callback functions by folding multiple events into one, you can choose to set the bounce time:

bouncetime set in milliseconds

GPIO.add_event_detect(channel, GPIO.RISING,
callback=callback_fn,bouncetime=200)

If edge detection is no longer needed, it can be removed as follows:

GPIO.remove_event_detect(channel)

10.Check the functionality of the GPIO channel

This feature allows you to check the functionality of the provided GPIO channel:

GPIO.gpio_function(channel)

This function returns GPIO. IN or GPIO. OUT.

11. PWM

Please refer to samples/simple_ PWM. py provides detailed information on how to use PWM channels.

The Jetson.GPIO library only supports PWM on pins that come with hardware PWM controllers. Unlike the RPi. GPIO library, the Jetson. GPIO library does not implement PWM for software simulation.

The system pin multiplexer must be configured to connect the hardware PWM controller to the relevant pins. If pinmux is not configured, the PWM signal will not reach the pin! The Jetson.GPIO library does not dynamically modify the pinmux configuration to achieve this goal. Read the L4T documentation for detailed information on how to configure pinmux

LINKS: https://github.com/NVIDIA/jetson-gpio