API usage of GPIO library

The Jetson GPIO library provides all the public APIs provided by the RPi.GPIO library. The usage of each API is discussed below:

1. Importing the library

To import the Jetson.GPIO module, use:

```
import Jetson.GPIO as GPIO
```

In this way, you can refer to the module as GPIO in the rest of the application. The module can also be imported using the name RPi.GPIO instead of Jetson.GPIO is used for existing code that uses the RPi library.

2. Pin numbering

The Jetson GPIO library provides four ways to number IO pins. The first two correspond to the modes provided by the RPi.GPIO library, namely BOARD and BCM, which refer to the pin numbers of the 40-pin GPIO header connector and the Broadcom SoC GPIO numbers respectively. The remaining two modes, CVM and TEGRA_SOC, use strings instead of numbers, and the numbers correspond to the signal names on the CVM CVB connector and the Tegra SoC respectively.

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

```
GPIO.setmode(GPIO.BOARD)# or

GPIO.setmode(GPIO.BCM)# or

GPIO.setmode(GPIO.CVM)# or

GPIO.setmode(GPIO.TEGRA_SOC)
```

To check which mode has been set, you can call:

```
mode = GPIO.getmode()
```

The 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, the Jetson
GPIO library will warn you if the GPIO used is configured for anything other than the default direction (input). It will also warn you if you try to clean up before setting the mode and channel. To disable warnings, use:

```
GPIO.setwarnings(False)
```

4. Setting up channels

GPIO channels must be set up before using them as input or output. To configure a channel as an input, call:

```
GPIO.setup(channel, GPIO.IN)
```

To set a channel as an output, call:

```
GPIO.setup(channel, GPIO.OUT)
```

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

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

When setting a channel as an output, you can also set multiple channels at once:

```
channels = [18, 12, 13]

GPIO.setup(channels, GPIO.OUT)
```

5. Input

To read the value of a channel, use:

```
GPIO.input(channel)
```

This will return either GPIO.LOW or GPIO.HIGH.

6. Output

To set the value of a pin configured as an output, use:

```
GPIO.output(channel, state)
```

State can be either GPIO.LOW or GPIO.HIGH.

You can also output to a list of channels or a 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. Cleanup

At the end of your program, it is a good idea to clean up the channels so that all pins are set to their default state. To clean up all used channels, use:

```
GPIO.cleanup()
```

If you don't want to clean up all channels, you can also clean up a single channel or a list or tuple of channels:

```
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 get information about the Jetson module, use/read:

GPIO.JETSON_INFO

This gives a Python dictionary with the following keys: P1_REVISION, RAM, REVISION, TYPE, MANUFACTURER, and PROCESSOR. All values in the dictionary are strings, except P1_REVISION which is an integer.

To get information about the library version, use/read:

GPIO.VERSION

This provides a string in the format of XYZ version.

9. Interrupts

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

wait_for_edge() function

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

GPIO.wait_for_edge(channel, GPIO.RISING)

The second argument specifies the edge to detect, which can be GPIO.RISING, GPIO.FALLING, or GPIO.BOTH. You can optionally set a timeout if you only want to limit the wait to a specified amount of time:

timeout is in milliseconds

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

This function returns the channel on which the edge was detected, or None if a timeout occurred.

event_detected() function

This function can be used to periodically check if an event has occurred since the last call. The function can be set up 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 detected

This function can be used to run a second thread for the callback function. Thus, the callback function can run concurrently with the main program in response to an edge. This function can be used as follows:

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)

You can also add multiple callbacks if you want:

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, the two callbacks are run sequentially, not simultaneously, because there is only one thread running all the callback functions.

To prevent the callback function from being called multiple times by collapsing multiple events into one, a bounce time can be optionally set:

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. Checking the functionality of a GPIO channel

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

GPIO.gpio_function(channel)

The function returns either GPIO.IN or GPIO.OUT.

11. PWM

See samples/simple_pwm.py for details on how to use PWM channels.

The Jetson.GPIO library only supports PWM on pins that come with a hardware PWM controller. Unlike the RPi.GPIO library, the Jetson.GPIO library does not implement software-emulated PWM. The Jetson
Nano supports 2 PWM channels, while the Jetson AGX Xavier supports 3 PWM channels. The Jetson
TX1 and TX2 do not support any PWM channels.

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

Full English instructions are available at: https://github.com/NVIDIA/jetson-gpio