GPIO Programming

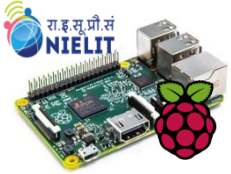
in

Raspberry Pi

Education is the kindling of a flame, not the filling of a vessel.

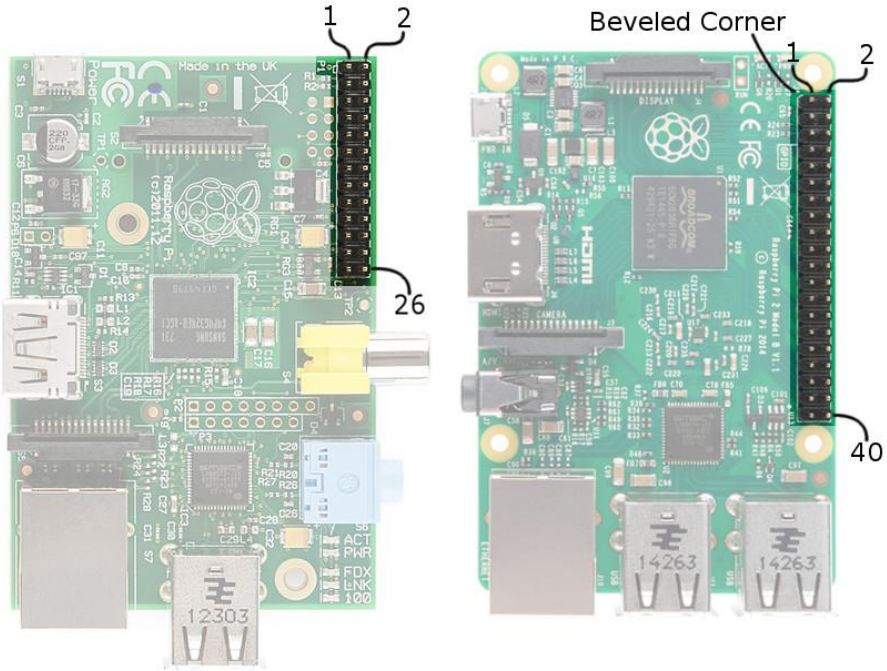
- Socrates

Raspberrry Pi



- Irrespective of its size, Raspberry Pi is a powerhorse of a computer. It can drive HDMI displays, mouse, keyboard, camera – above all it runs full featured Linux distribution.
- Not only computer it is hardware prototyping tool.
- The Pi has **bi-directional I/O pins**, which can be used to drive LEDs, spin motors, or read button presses.

GPIO Pinout

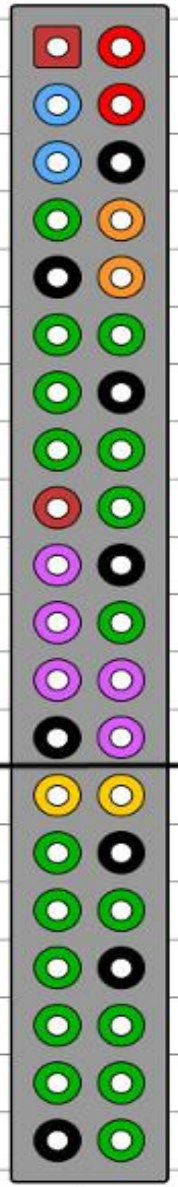


When referencing Pi pin numbers, there are two different numbering schemes:

- Broadcom chip-specific pin numbers
- P1 physical pin numbers.

Raspberry Pi2 GPIO Header

	Pin#	NAME	NAME	Pin#
Early Models	01	3.3v DC Power	DC Power 5v	02
	03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
	05	GPIO03 (SCL1 , I²C)	Ground	06
	07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
	09	Ground	(RXD0) GPIO15	10
	11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
	13	GPIO27 (GPIO_GEN2)	Ground	14
	15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
	17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
	19	GPIO10 (SPI_MOSI)	Ground	20
	21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
	23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
	25	Ground	(SPI_CE1_N) GPIO07	26
Late Models	27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
	29	GPIO05	Ground	30
	31	GPIO06	GPIO12	32
	33	GPIO13	Ground	34
	35	GPIO19	GPIO16	36
	37	GPIO26	GPIO20	38
	39	Ground	GPIO21	40



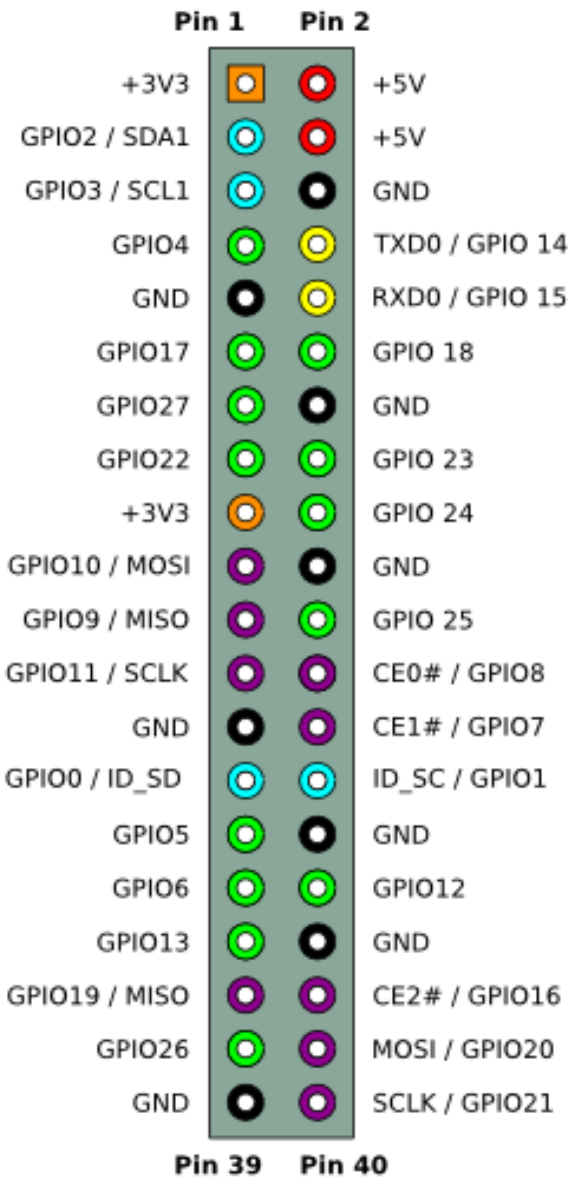
Rev. 1
26/01/2014

http://www.element14.com

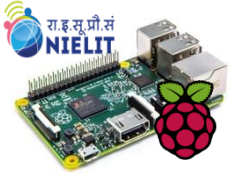
Wedge Silk	Python (BCM)	Wiring gPi GPIO	Name	P1 Pin Number		Name	Wiring gPi GPIO	Python (BCM)	Wedge Silk
			3.3v DC Power	**1**	**2**	5v DC Power			
SDA		8	GPIO02 (SDA1, I2C)	**3**	**4**	5v DC Power			
SCL		9	GPIO03 (SCL1, I2C)	**5**	**6**	Ground			
G4	4	7	GPIO04 (GPIO_GCLK)	**7**	**8**	GPIO14 (TXD0)	15		TXO
			Ground	**9**	**10**	GPIO15 (RXD0)	16		RXI
G17	17	0	GPIO17 (GPIO_GEN0)	**11**	**12**	GPIO18 (GPIO_GEN1)	1	18	G18
G27	27	2	GPIO27 (GPIO_GEN2)	**13**	**14**	Ground			
G22	22	3	GPIO22 (GPIO_GEN3)	**15**	**16**	GPIO23 (GPIO_GEN4)	4	23	G23
			3.3v DC Power	**17**	**18**	GPIO24 (GPIO_GEN5)	5	24	G24
MOSI		12	GPIO10 (SPI_MOSI)	**19**	**20**	Ground			

Wedge Silk	Python (BCM)	WiringPi GPIO	Name	P1 Pin Number		Name	WiringPi GPIO	Python (BCM)	Wedge Silk
MISO		13	GPIO09 (SPI_MISO)	**21**	**22**	GPIO25 (GPIO_GEN6)	6	25	G25
		(no work y 14)	GPIO11 (SPI_CLK)	**23**	**24**	GPIO08 (SPI_CE0_N)	10		CD0
			Ground	**25**	**26**	GPIO07 (SPI_CE1_N)	11		CE1
IDSD		30	ID_SD (I2C ID EEPROM)	**27**	**28**	ID_SC (I2C ID EEPROM)	31		IDSC
G05	5	21	GPIO05	**29**	**30**	Ground			
G6	6	22	GPIO06	**31**	**32**	GPIO12	26	12	G12
G13	13	23	GPIO13	**33**	**34**	Ground			
G19	19	24	GPIO19	**35**	**36**	GPIO16	27	16	G16
G26	26	25	GPIO26	**37**	**38**	GPIO20	28	20	G20
			Ground	**39**	**40**	GPIO21	29	21	G21

Pi GPIO header



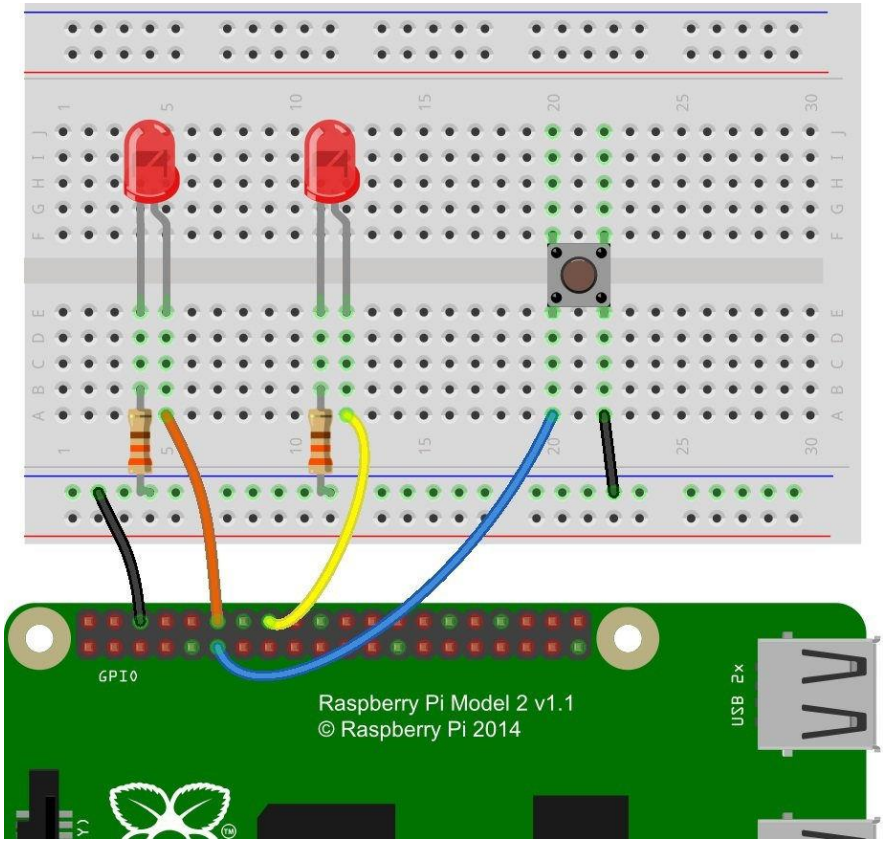
Pin Number	Pin Name Rev2	Pin Number	Pin Name Rev2
P1-01	3.3 V	P1-02	5V0
P1-03	GPIO 2	P1-04	5V0
P1-05	GPIO 3	P1-06	GND
P1-07	GPIO 4	P1-08	GPIO 14
P1-09	GND	P1-10	GPIO 15
P1-11	GPIO17	P1-12	GPIO 18
P1-13	GPIO27	P1-14	GND
P1-15	GPIO22	P1-16	GPIO23
P1-17	3.3 V	P1-18	GPIO24
P1-19	GPIO10	P1-20	GND
P1-21	GPIO9	P1-22	GPIO25
P1-23	GPIO11	P1-24	GPIO08
P1-25	GND	P1-26	GPIO07


रा.इ.सू.प्रौ.सं
NIELIT

GETTING READY WITH HARDWARE

Circuit

- **Two LEDs** are connected to the **Pi's GPIO 18 and GPIO 23** – those are the Broadcom chip-specific numbers.
- P1 connector pin numbers, 12 and 16.
- The **button** is connected to Broadcom **GPIO 17**, aka P1 pin 11.



Coding

- Import the RPi.GPIO module

  import RPi.GPIO as GPIO

  GPIO is the local name of the module in the code

- Choosing Pin Numbering Declaration
  - GPIO.BOARD – Board numbering scheme. The pin numbers follow the pin numbers on header P1.
  - GPIO.BCM – Broadcom chip-specific pin numbers. These pin numbers follow the lower-level numbering system defined by the Raspberry Pi's Broadcom-chip brain.

Coding

- Command to select pin numbering selection

  e.g. **GPIO.setmode(GPIO.BCM)**

* GPIO.BCM numbers are silkscreened on the PCB

- Function used for setting the Pin Mode

  setup([pin], [GPIO.IN, GPIO.OUT]

  e.g. **GPIO.setup(18, GPIO.OUT)**

Coding - output

- Digital Output

  Function used to write a pin high or low
  GPIO.output([pin], [GPIO.LOW, GPIO.HIGH])

  e.g. **GPIO.output(18, GPIO.HIGH)**

- PWM ("Analog") Output

  function used is GPIO.PWM([pin], [frequency])

  and to start PWM pwm.start([duty cycle])

  pwm = GPIO.PWM(18, 1000)

  pwm.start(50)

Coding - output – contd…

- PWM ("Analog") Output…

Function used to adjust the value of the PWM output, is pwm.ChangeDutyCycle([duty cycle])

[duty cycle] can be any value between 0 (i.e 0%/LOW) and 100 (ie.e 100%/HIGH). E.g. to set a pin to 75%

pwm.ChangeDutyCycle(75)

Command to turn PWM off is, **pwm.stop()**

* Pin has to be set as an output before using it as PWM

Coding - input

- Function to read the value of pin, if it is configured as an input is - GPIO.input([pin])

*Function will return True or False, indicating High or low

```
if GPIO.input(17):
    print("Pin 11 is HIGH")
else:
    print("Pin 11 is LOW")
```

Coding - Pull-Up/Down Resistors

- Command used to set a pull-up resistor on GPIO 17,

  GPIO.setup(17, GPIO.IN,
  pull_up_down=GPIO.PUD_UP)

* the pull_up_down can have two values

  pull_up_down=GPIO.PUD_DOWN

  pull_up_down=GPIO.PUD_UP

* If nothing is declared in that third value, both pull-resistors will be disabled.

Delays

To slow down the execution of Python script:

include time

time.sleep([seconds])

To have delay of 250 milliseconds

time.sleep(0.25)

- Garbage Collecting

  GPIO.cleanup()

Create a file

pi@raspberrypi ~/code $ mkdir python
pi@raspberrypi ~/code $ cd python

- Create a file with **.py** extension.
- Then open it in editor

pi@raspberrypi ~/code/python $ touch blinker.py
pi@raspberrypi ~/code/python $ leafpad blinker.py &

codify

```
# External module imports
import RPi.GPIO as GPIO
import time
                    # Pin Definitons:
pwmPin = 18    # Broadcom pin 18 (P1 pin 12)
ledPin = 23    # Broadcom pin 23 (P1 pin 16)
butPin = 17    # Broadcom pin 17 (P1 pin 11)
dc = 95 # duty cycle (0-100) for PWM pin
# Pin Setup:
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
GPIO.setup(pwmPin, GPIO.OUT) # PWM pin set as output
```

Codify …

```
pwm = GPIO.PWM(pwmPin, 50)
        # Initialize PWM on pwmPin 100Hz frequency
GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
        # Button pin set as input w/ pull-up

GPIO.output(ledPin, GPIO.LOW)
    # Initial state for LEDs pwm.start(dc)

print("Here we go! Press CTRL+C to exit")
```

Codify …

```
try:
    while 1:
        if GPIO.input(butPin): # button is released
            pwm.ChangeDutyCycle(dc)
            GPIO.output(ledPin, GPIO.LOW)
        else: # button is pressed:
            pwm.ChangeDutyCycle(100-dc)
            GPIO.output(ledPin, GPIO.HIGH)
            time.sleep(0.075)
            GPIO.output(ledPin, GPIO.LOW)
            time.sleep(0.075)
except KeyboardInterrupt: # If CTRL+C is pressed, exit cleanly:
    pwm.stop() # stop PWM
    GPIO.cleanup() # cleanup all GPIO
```

Run code

- To run your “blinker.py” script, type:

  $ sudo python blinker.py

- With the code running, press the button to turn on the digital LED. The PWM-ing LED will invert its brightness when you press the button as well.