

物聯網與微處理機系統設計

Internet of Things and Microprocessor System Design

Lecture 03 – RPi GPIO

Lecturer: 陳彥安 Chen, Yan-Ann

YZU CSE

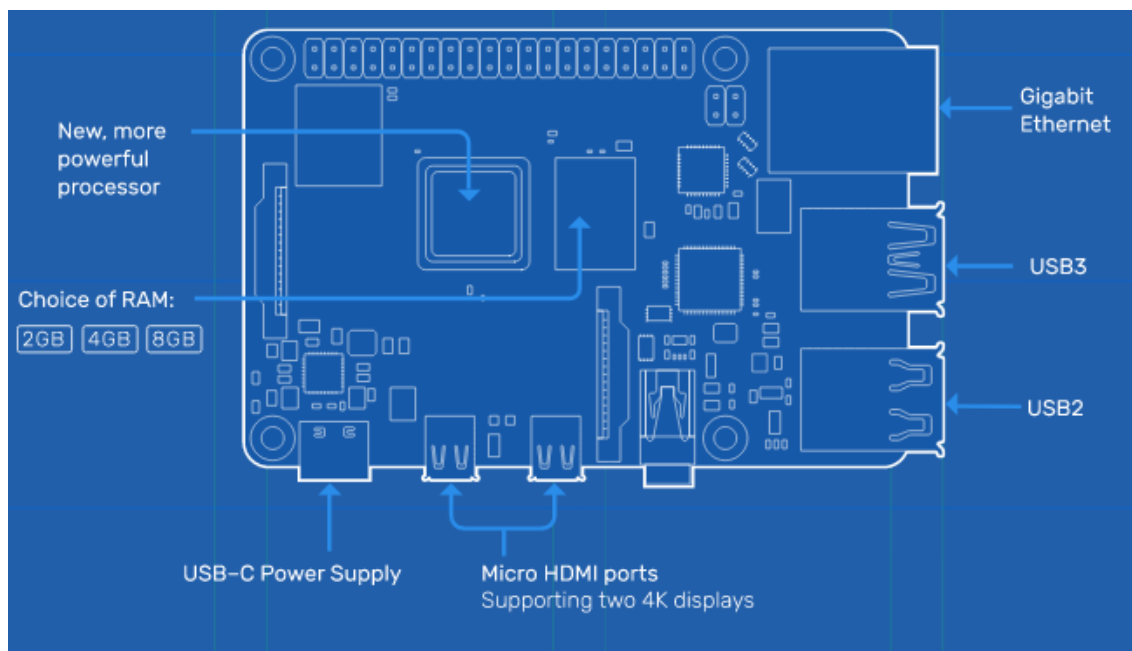
Outline

- GPIO Introduction
- GPIO Output
- GPIO Input
- Lab

Outline

- GPIO Introduction
- GPIO Output
- GPIO Input
- Lab

RPi Spec



Specifications

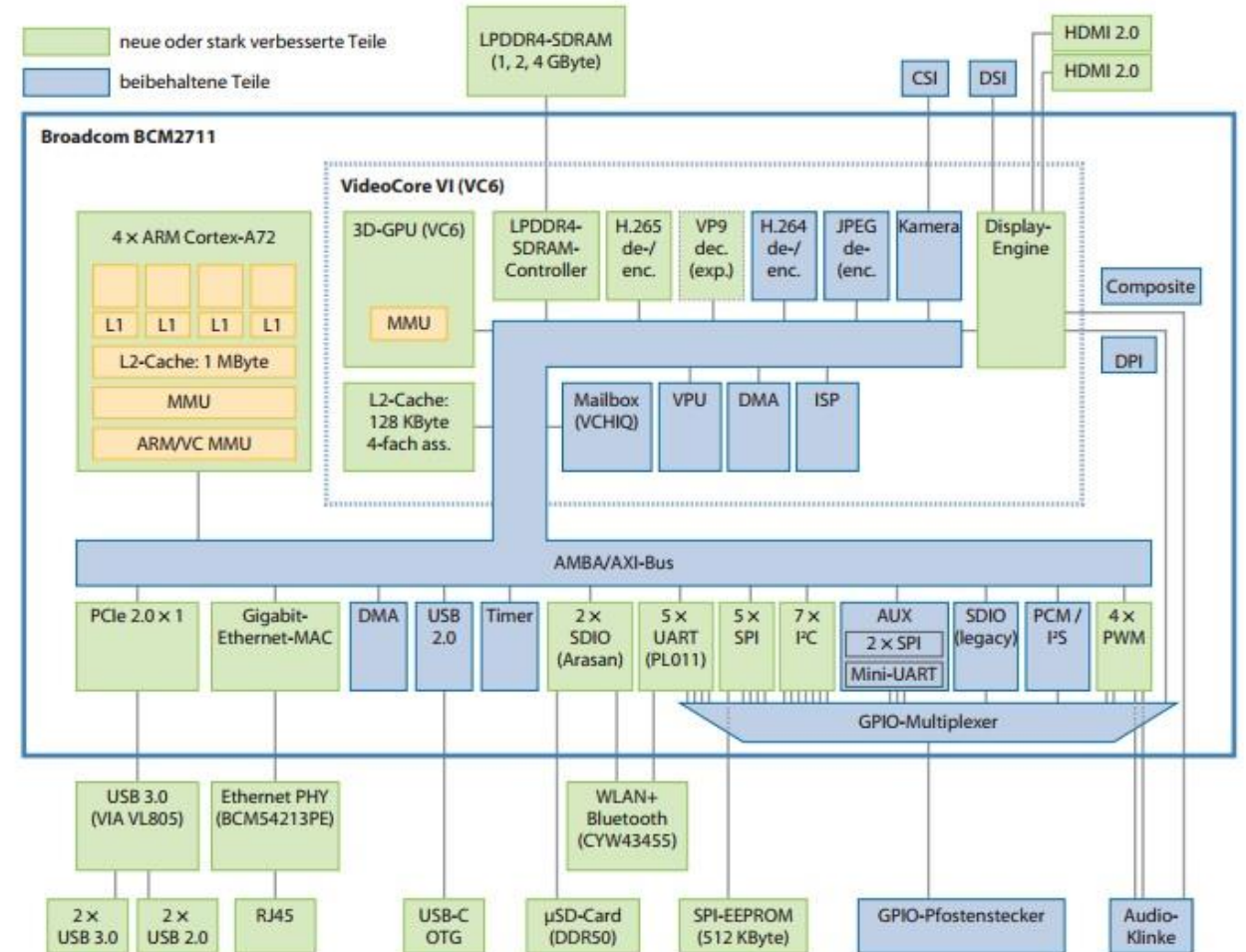
- **Broadcom BCM2711**, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- **Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)**
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

RPi Block Diagram

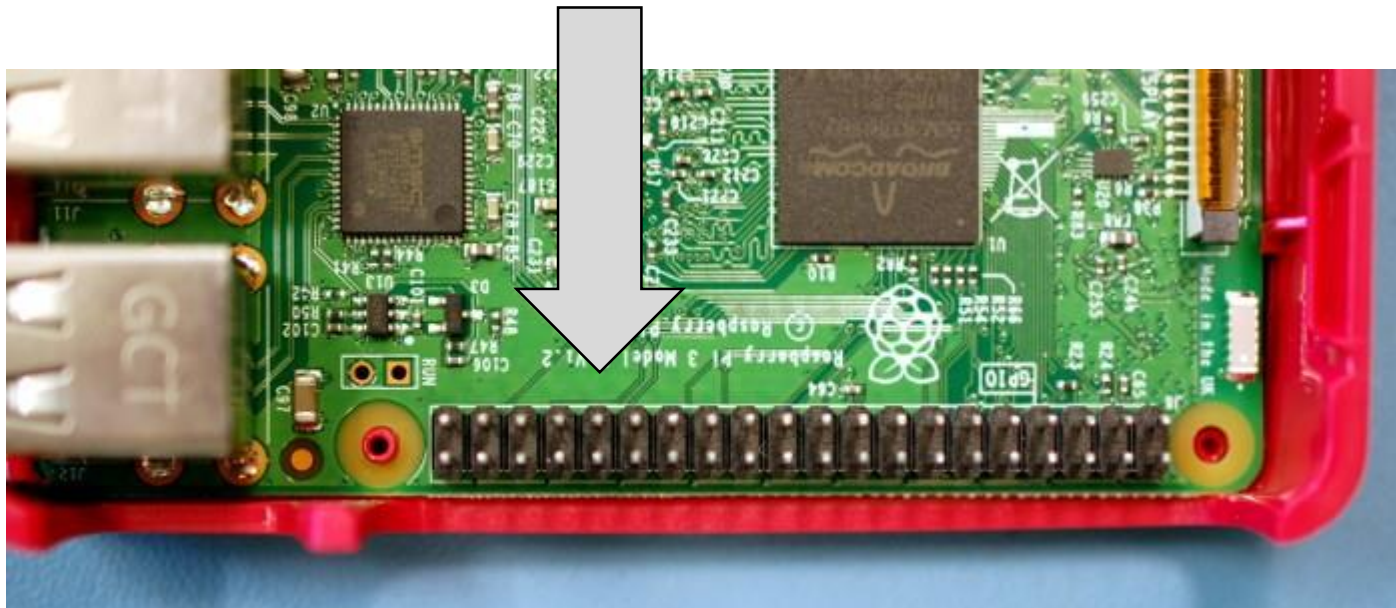
Herz des Raspberry Pi 4: Broadcom BCM2711

Das System-on-Chip (SoC) BCM2711 vereint nicht nur vier CPU-Kerne mit einer GPU, sondern enthält auch Controller für viele Schnittstellen.



GPIO

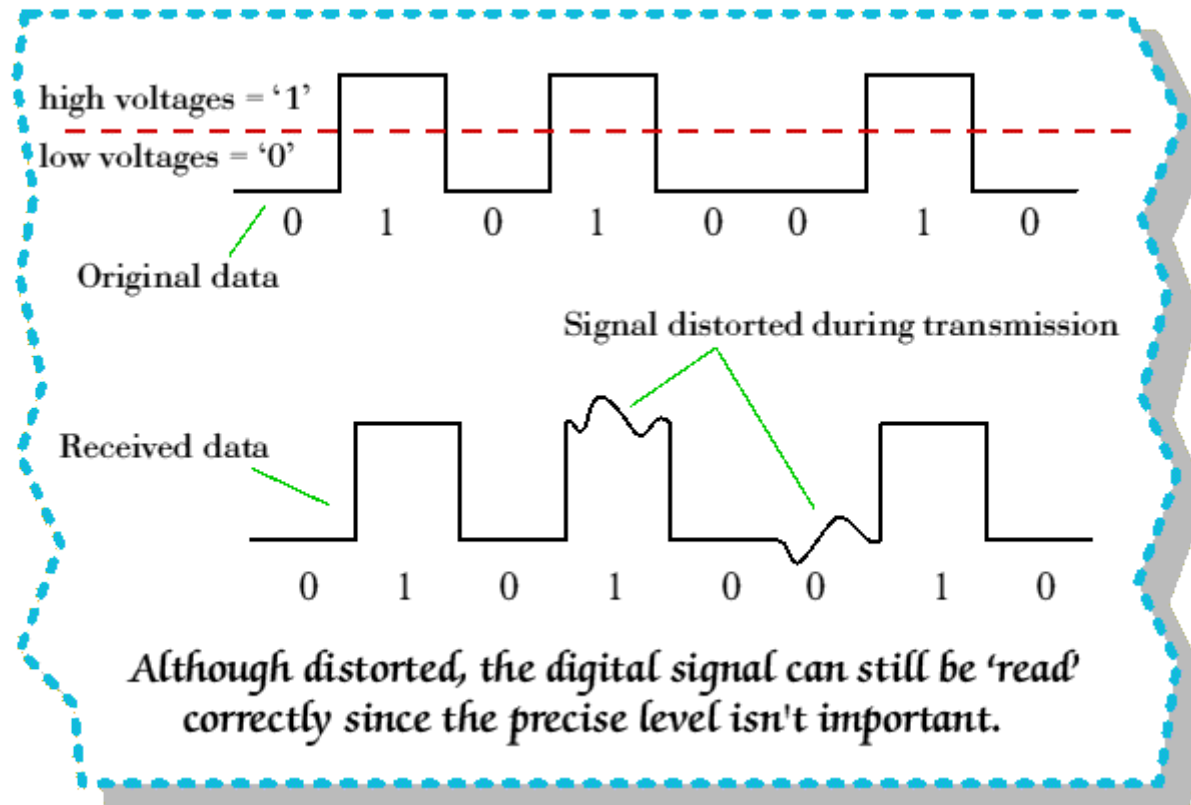
- General-purpose input/output (GPIO)
- You can set PIN as Input or Output or Both
 - Input: Read a value (voltage) from a PIN
 - Output: Write a value (voltage) to a PIN



src: <https://www.rs-online.com/designspark/interfacing-hardware-with-the-raspberry-pi>

Digital Signal

- Digital '1': High voltages (3.3v)
- Digital '0': Low voltages (0v)



src: https://www.st-andrews.ac.uk/~www_pa/Scots_Guide/info/signals/digital/digital.htm

GPIO

■ Voltages

- Two 5V pins and two 3V3 pins are present on the board
- A number of ground pins (0V), which are unconfigurable.
- The remaining pins are all general purpose 3V3 pins, meaning **outputs are set to 3V3** and **inputs are 3V3-tolerant**.

■ Outputs

- A GPIO pin designated as an output pin can be set to **high (3V3)** or **low (0V)**.

■ Inputs

- A GPIO pin designated as an input pin can be read as **high (3V3)** or **low (0V)**.
- This is made easier with the use of internal pull-up or pull-down resistors.
- Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

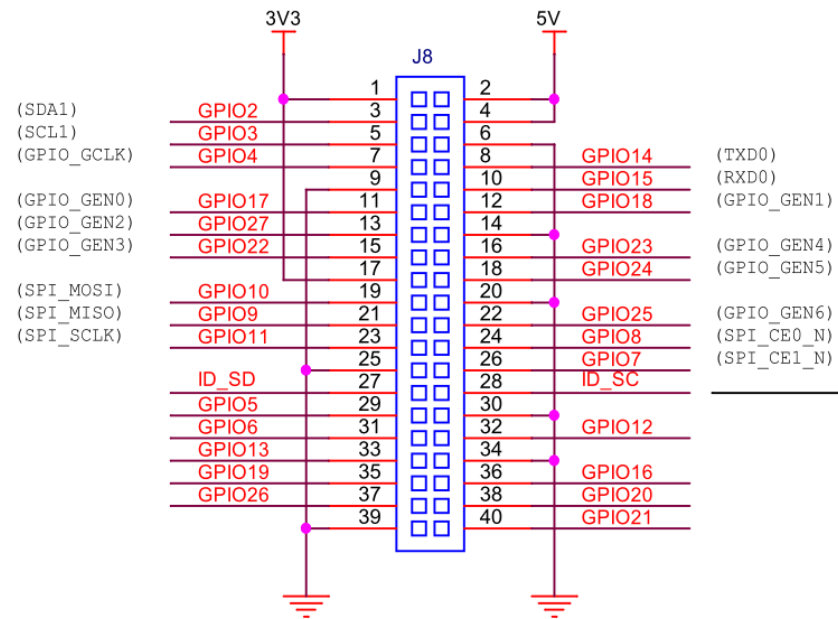
Alternative Functions

- PWM (pulse-width modulation)
 - Software PWM available on all pins
 - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI
 - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
 - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
 - Data: (GPIO2); Clock (GPIO3)
 - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
 - TX (GPIO14); RX (GPIO15)

GPIO Pinout



Broadcom BCM2711



GPIO EXPANSION

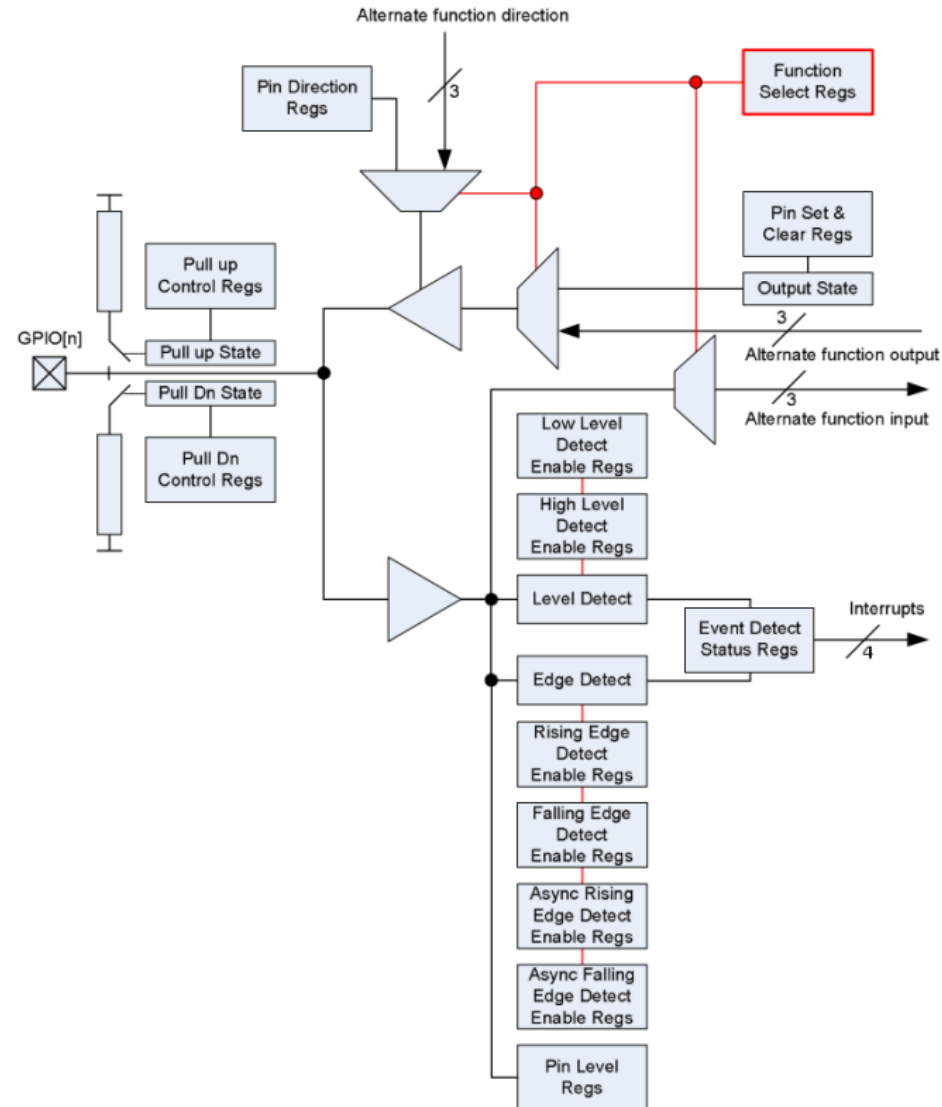
ID_SD and ID_SC PINS:

These pins are reserved for HAT ID EEPROM.

At boot time this I2C interface will be interrogated to look for an EEPROM that identifies the attached board and allows automatic setup of the GPIOs (and optionally, Linux drivers).

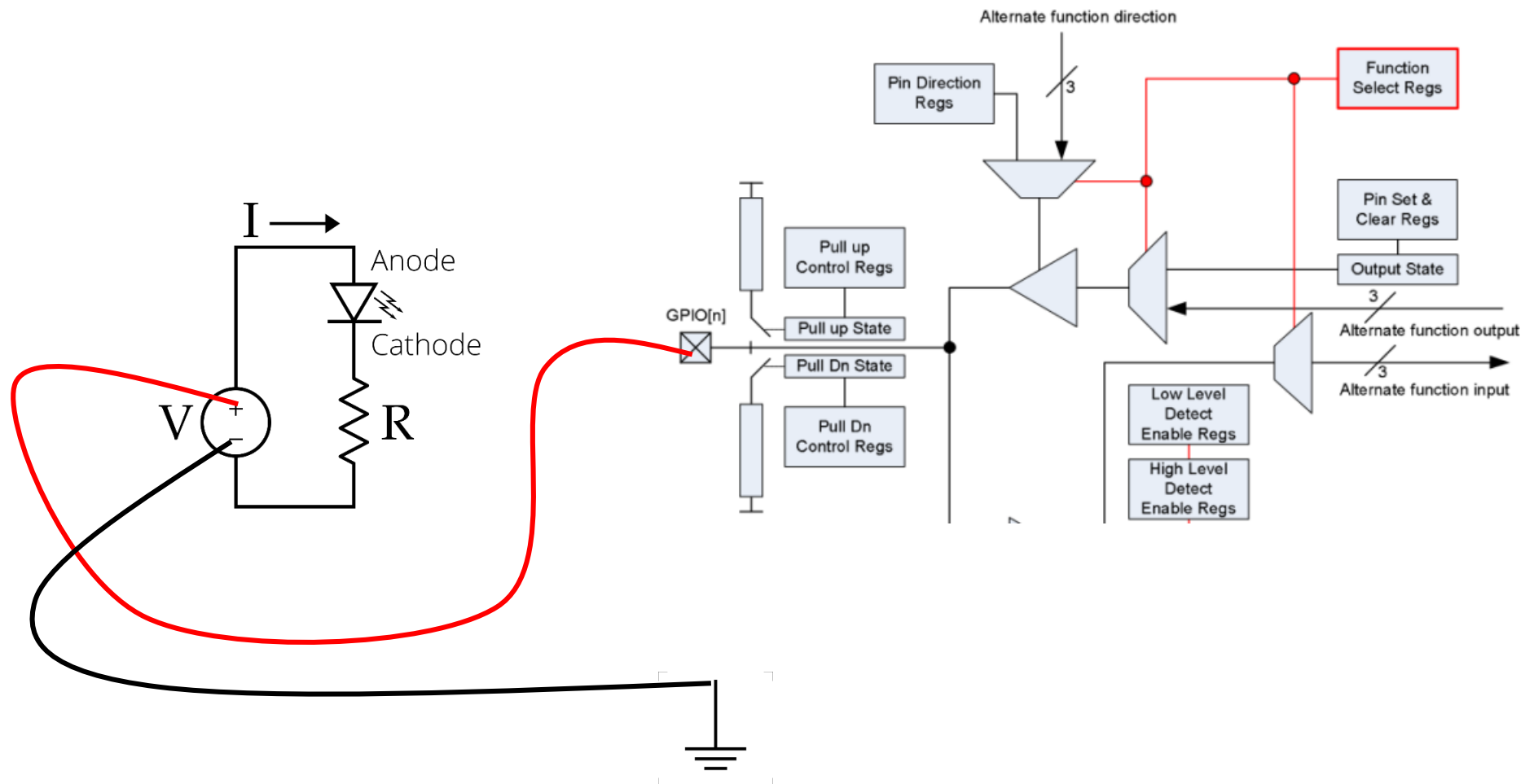
DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

GPIO Block Diagram



GPIO Output

■ Output example



RPi Pinout

■ <https://pinout.xyz/>



Raspberry Pi
Pinout

3v3 Power	1		2	5v Power
GPIO 2 (I2C1 SDA)	3		4	5v Power
GPIO 3 (I2C1 SCL)	5		6	Ground
GPIO 4 (GPCLK0)	7		8	GPIO 14 (UART TX)
Ground	9		10	GPIO 15 (UART RX)
GPIO 17	11		12	GPIO 18 (PCM CLK)
GPIO 27	13		14	Ground
GPIO 22	15		16	GPIO 23
3v3 Power	17		18	GPIO 24
GPIO 10 (SPI0 MOSI)	19		20	Ground
GPIO 9 (SPI0 MISO)	21		22	GPIO 25
GPIO 11 (SPI0 SCLK)	23		24	GPIO 8 (SPI0 CE0)
Ground	25		26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27		28	GPIO 1 (EEPROM SCL)
GPIO 5	29		30	Ground
GPIO 6	31		32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33		34	Ground
GPIO 19 (PCM FS)	35		36	GPIO 16
GPIO 26	37		38	GPIO 20 (PCM DIN)
Ground	39		40	GPIO 21 (PCM DOUT)

Example:

GPIO 17

Alt0	Alt1	Alt2	Alt3	Alt4	Alt5
FL1	SMI SD9	DPI D13	UART0 RTS	SPI1 CE1	UART1 RTS

- Physical/Board pin 11
- GPIO/BCM pin 17
- Wiring Pi pin 0

GPIO	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
GPIO1	High	SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
GPIO2	High	SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
GPIO3	High	SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
GPIO4	High	GPCLK0	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
GPIO5	High	GPCLK1	SA0	DPI_D1	SPI4_MISO	RXD3	SCL3
GPIO6	High	GPCLK2	SOE_N / SE	DPI_D2	SPI4_MOSI	CTS3	SDA4
GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
GPIO8	High	SPI0_CE0_N	SD0	DPI_D4	BSCSL / CE_N	TXD4	SDA4
GPIO9	Low	SPI0_MISO	SD1	DPI_D5	BSCSL / MISO	RXD4	SCL4
GPIO10	Low	SPI0_MOSI	SD2	DPI_D6	BSCSL_SDA / MOSI	CTS4	SDA5
GPIO11	Low	SPI0_SCLK	SD3	DPI_D7	BSCSL_SCL / SCLK	RTS4	SCL5
GPIO12	Low	PWM0_0	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
GPIO13	Low	PWM0_1	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
GPIO14	Low	TXD0	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
GPIO15	Low	RXD0	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
GPIO16	Low	<reserved>	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	PWM0_0
GPIO19	Low	PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM0_1
GPIO20	Low	PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPI1_MOSI	GPCLK0

GPIO	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO21	Low	PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1
GPIO22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
GPIO23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
GPIO24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
GPIO25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
GPIO26	Low	SD0_DAT2	<reserved>	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
GPIO27	Low	SD0_DAT3	<reserved>	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N
GPIO28	-	SDA0	SA5	PCM_CLK	<reserved>	MII_A_RX_ER R	RGMII_MDIO
GPIO29	-	SCL0	SA4	PCM_FS	<reserved>	MII_A_TX_ER R	RGMII_MDC
GPIO30	Low	<reserved>	SA3	PCM_DIN	CTS0	MII_A_CRD	CTS1
GPIO31	Low	<reserved>	SA2	PCM_DOUT	RTS0	MII_A_COL	RTS1
GPIO32	Low	GPCLK0	SA1	<reserved>	TXD0	SD_CARD_PR ES	TXD1
GPIO33	Low	<reserved>	SA0	<reserved>	RXD0	SD_CARD_W RPROT	RXD1
GPIO34	High	GPCLK0	SOE_N / SE	<reserved>	SD1_CLK	SD_CARD_LE D	RGMII_IRQ
GPIO35	High	SPI0_CE1_N	SWE_N / SRW_N		SD1_CMD	RGMII_STAR T_STOP	

GPIO	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO36	High	SPI0_CE0_N	SD0	TXD0	SD1_DAT0	RGMII_RX_O K	MII_A_RX_ER R
GPIO37	Low	SPI0_MISO	SD1	RXD0	SD1_DAT1	RGMII_MDIO	MII_A_TX_ER R
GPIO38	Low	SPI0_MOSI	SD2	RTS0	SD1_DAT2	RGMII_MDC	MII_A_CR_S
GPIO39	Low	SPI0_SCLK	SD3	CTS0	SD1_DAT3	RGMII_IRQ	MII_A_COL
GPIO40	Low	PWM1_0	SD4		SD1_DAT4	SPI0_MISO	TXD1
GPIO41	Low	PWM1_1	SD5	<reserved>	SD1_DAT5	SPI0_MOSI	RXD1
GPIO42	Low	GPCLK1	SD6	<reserved>	SD1_DAT6	SPI0_SCLK	RTS1
GPIO43	Low	GPCLK2	SD7	<reserved>	SD1_DAT7	SPI0_CE0_N	CTS1
GPIO44	-	GPCLK1	SDA0	SDA1	<reserved>	SPI0_CE1_N	SD_CARD_VO LT
GPIO45	-	PWM0_1	SCL0	SCL1	<reserved>	SPI0_CE2_N	SD_CARD_P WRO
GPIO46	High	<Internal>					
GPIO47	High	<Internal>					
GPIO48	High	<Internal>					
GPIO49	High	<Internal>					
GPIO50	High	<Internal>					
GPIO51	High	<Internal>					
GPIO52	High	<Internal>					
GPIO53	High	<Internal>					
GPIO54	High	<Internal>					
GPIO55	High	<Internal>					
GPIO56	High	<Internal>					
GPIO57	High	<Internal>					

RPi Pinout

■ BCM numbering

24x - GPIO pins

1x - Serial UARTs (RPi3 only includes mini UART)

2x - SPI bus

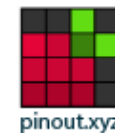
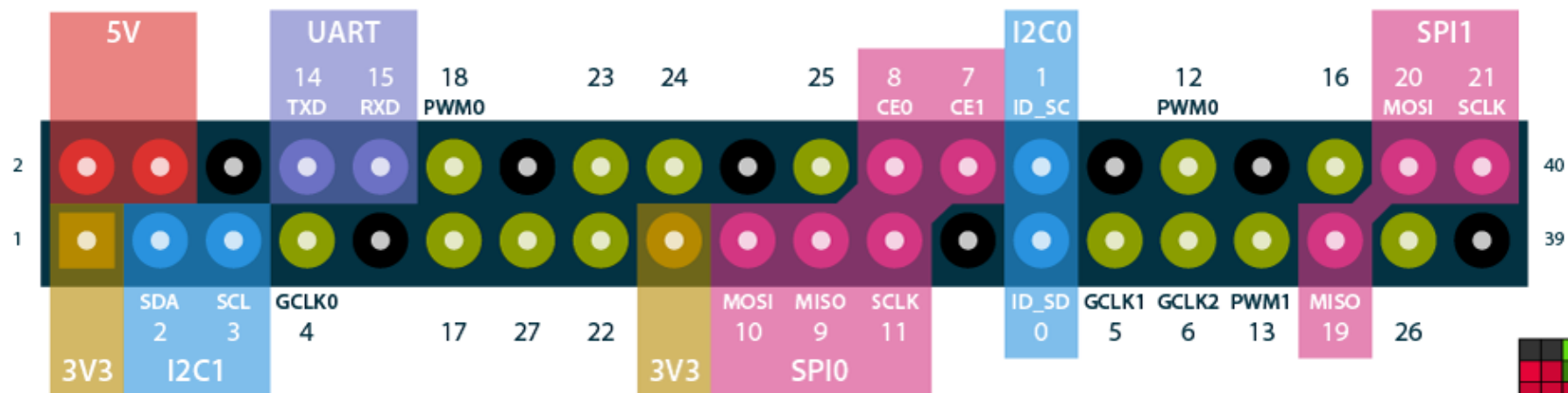
1x - I2C bus

2x - 5V power pins

2x - 3.3V power pins

8x - Ground pins

Raspberry Pi GPIO BCM numbering



The GPIO has the following registers. All accesses are assumed to be 32-bit. The GPIO register base address is **0x7E21 5000**.

Address Offset	Register Name	Description	Size
0x00	GPFSEL0	GPIO Function Select 0	32
0x04	GPFSEL1	GPIO Function Select 1	32
0x08	GPFSEL2	GPIO Function Select 2	32
0x0C	GPFSEL3	GPIO Function Select 3	32
0x10	GPFSEL4	GPIO Function Select 4	32
0x14	GPFSEL5	GPIO Function Select 5	32
0x18	-	Reserved	-
0x1C	GPSET0	GPIO Pin Output Set 0	32
0x20	GPSET1	GPIO Pin Output Set 1	32
0x24	-	Reserved	-
0x28	GPCLR0	GPIO Pin Output Clear 0	32
0x2C	GPCLR1	GPIO Pin Output Clear 1	32
0x30	-	Reserved	-
0x34	GPLEV0	GPIO Pin Level 0	32
0x38	GPLEV1	GPIO Pin Level 1	32
0x3C	-	Reserved	-
0x40	GPEDS0	GPIO Pin Event Detect Status 0	32
0x44	GPEDS1	GPIO Pin Event Detect Status 1	32
0x48	-	Reserved	-
0x4C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32
0x50	GPREN1	GPIO Pin Rising Edge Detect Enable 1	32
0x54	-	Reserved	-
0x58	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32

Address Offset	Register Name	Description	Size
0x5C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	32
0x60	-	Reserved	-
0x64	GPHEN0	GPIO Pin High Detect Enable 0	32
0x68	GPHEN1	GPIO Pin High Detect Enable 1	32
0x6C	-	Reserved	-
0x70	GPLEN0	GPIO Pin Low Detect Enable 0	32
0x74	GPLEN1	GPIO Pin Low Detect Enable 1	32
0x78	-	Reserved	-
0x7C	GPAREN0	GPIO Pin Async. Rising Edge Detect 0	32
0x80	GPAREN1	GPIO Pin Async. Rising Edge Detect 1	32
0x84	-	Reserved	-
0x88	GPAFEN0	GPIO Pin Async. Falling Edge Detect 0	32
0x8C	GPAFEN1	GPIO Pin Async. Falling Edge Detect 1	32
0x90	-	Reserved	-
0xE4	GPIO_PUP_PDN_CNTRL_REG0	GPIO Pull-up / Pull-down Register 0	32
0xE8	GPIO_PUP_PDN_CNTRL_REG1	GPIO Pull-up / Pull-down Register 1	32
0xEC	GPIO_PUP_PDN_CNTRL_REG2	GPIO Pull-up / Pull-down Register 2	32
0xF0	GPIO_PUP_PDN_CNTRL_REG3	GPIO Pull-up / Pull-down Register 3	32

Memory Map I/O

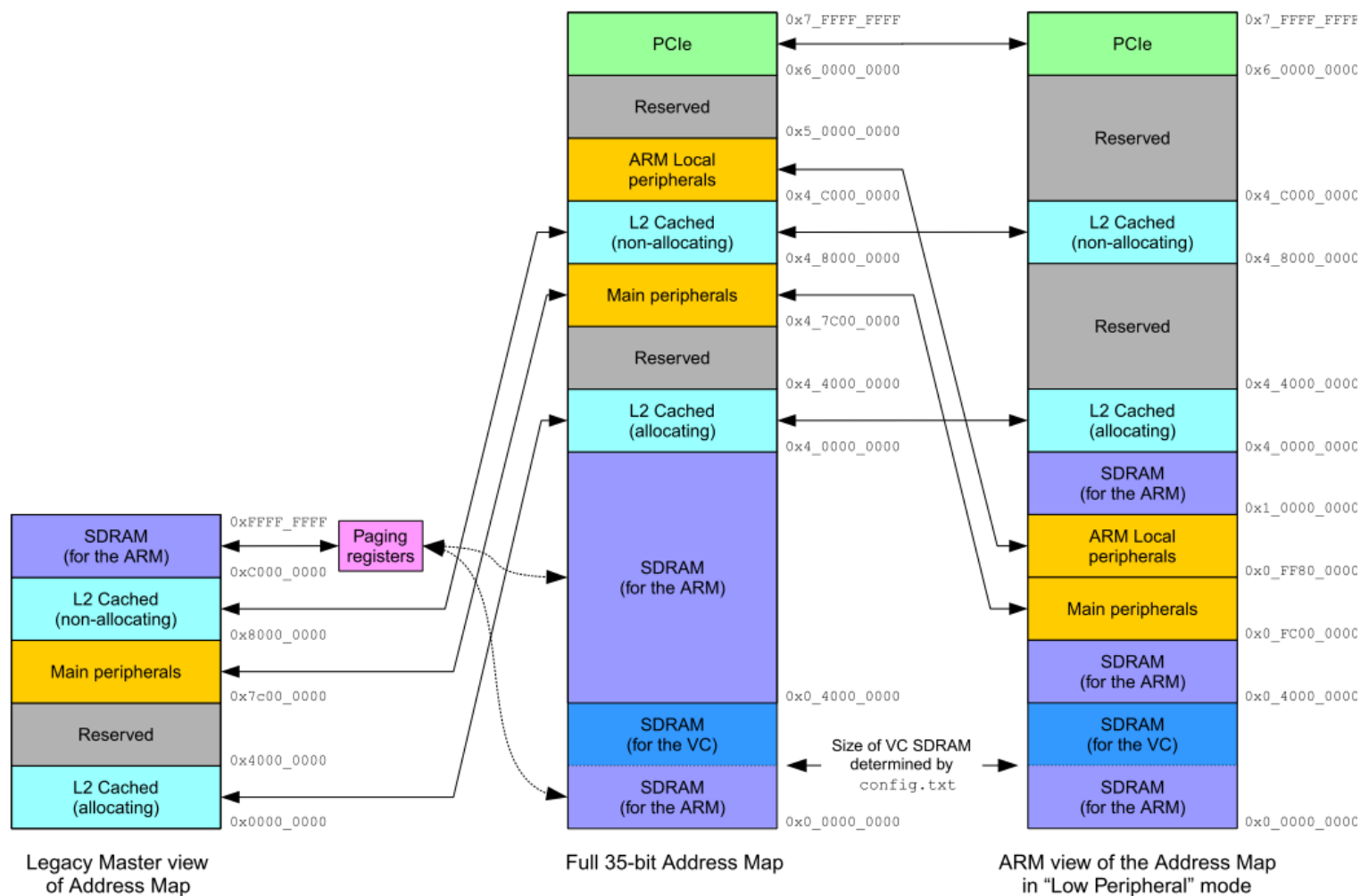


Figure 1. BCM2711 Address Maps


GPIO Limitations

Obey these simple rules to **reduce the risk of damaging your Raspberry Pi** when using the GPIO connector:

- Do not put **more than 3.3V** on any GPIO pin being used as an input.
- Do not draw more than 16mA per output and keep the total for all outputs below 50mA in total for an older 26-pin Raspberry Pi, and below **100mA** on a 40-pin Raspberry Pi.
- When using LEDs, 3mA is enough to light a red LED reasonably brightly with a 470 Ω series resistor.
- **Do not poke at the GPIO connector** with a screwdriver or any metal object when the Pi is powered up.
- Do not power the Pi with more than 5V.
- Do not draw more than a total of 250mA from the 5V supply pins.

Ref: <https://books.google.com.tw/books?id=0skvDAAAQBAJ&pg=PT270&lpg=PT270#v=onepage&q&f=false>

Programming Language for GPIO

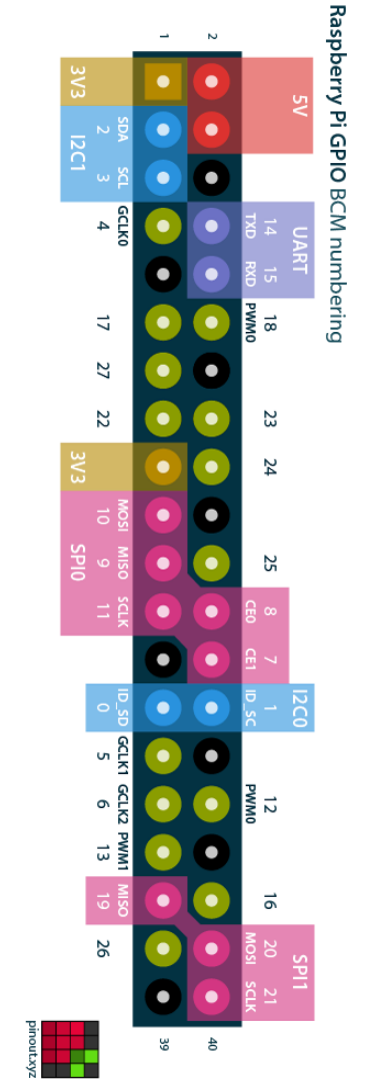
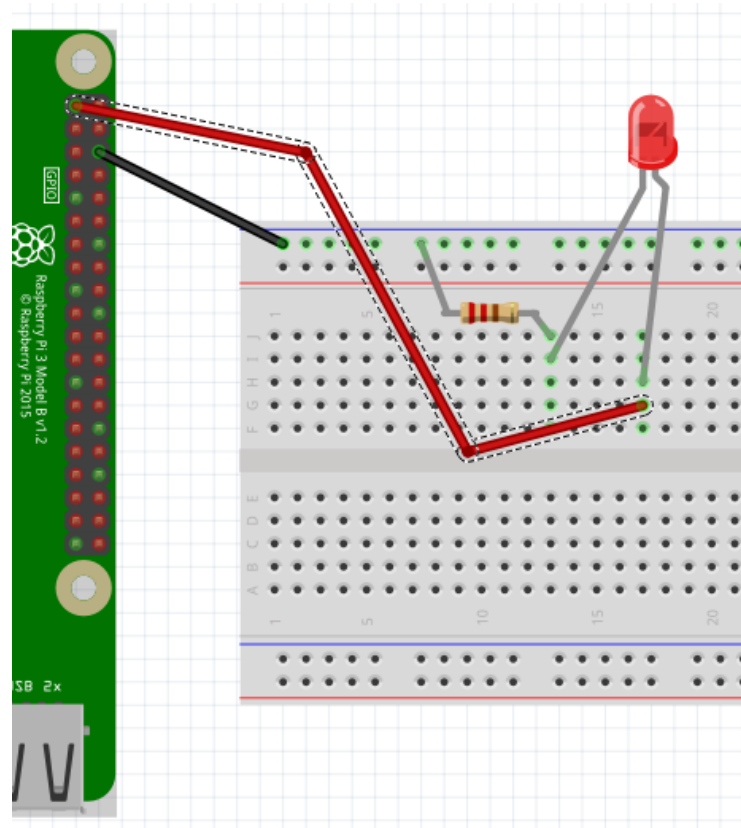
- C
- C + wiringPi
- C#
- Ruby
- Perl
- **Python** 
- Scratch
- Java Pi4J Library
- Shell script

Outline

- GPIO Introduction
- GPIO Output
- GPIO Input
- Lab

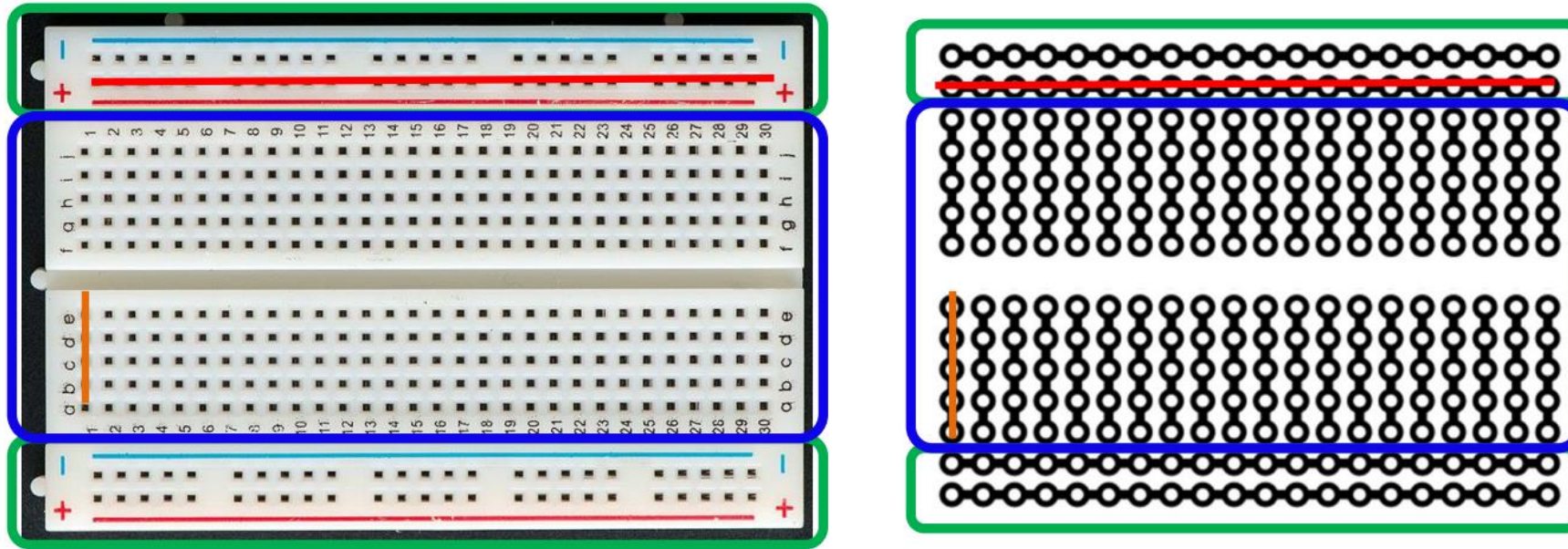
Simple Circuit

- PIN 1 - LED(+)
- PIN 6 - LED(-)



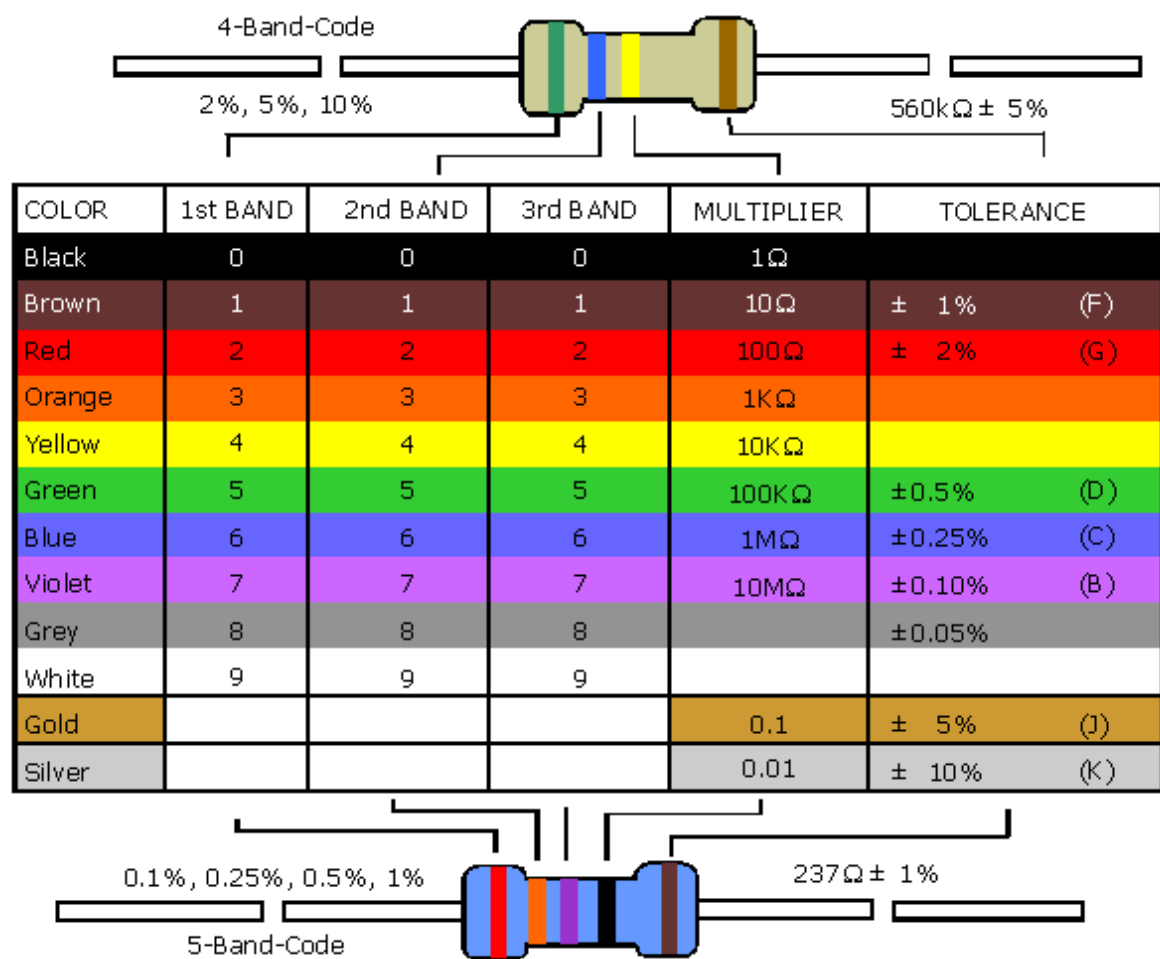
Breadboard

Bus strips: one for ground (-) and one for a supply voltage (+)



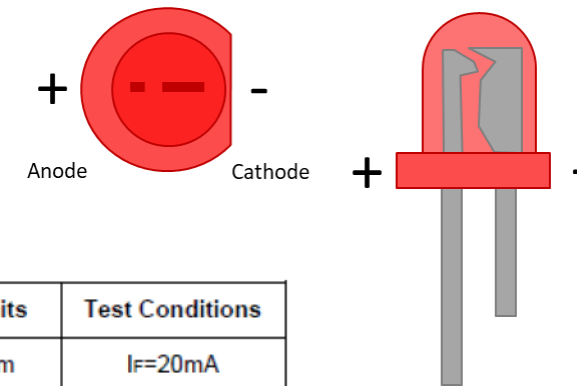
Terminal strips (ex: a1, b1, c1, d1 and e1 are connected)

Resistor



LED

- The LED (Light Emitting Diode) is a simple, digital actuator.
- LEDs have a **short leg (-)** and a **long leg (+)**
- To prevent damage, LEDs are used together with a resistor.



Electrical / Optical Characteristics at TA=25°C

Symbol	Parameter	Device	Typ.	Max.	Units	Test Conditions
λ_{peak}	Peak Wavelength	Super Bright Red	660		nm	$I_f=20\text{mA}$
λ_D [1]	Dominant Wavelength	Super Bright Red	640		nm	$I_f=20\text{mA}$
$\Delta\lambda_{1/2}$	Spectral Line Half-width	Super Bright Red	20		nm	$I_f=20\text{mA}$
C	Capacitance	Super Bright Red	45		pF	$V_F=0\text{V}; f=1\text{MHz}$
V_F [2]	Forward Voltage	Super Bright Red	1.85	2.5	V	$I_f=20\text{mA}$
I_R	Reverse Current	Super Bright Red		10	μA	$V_R = 5\text{V}$

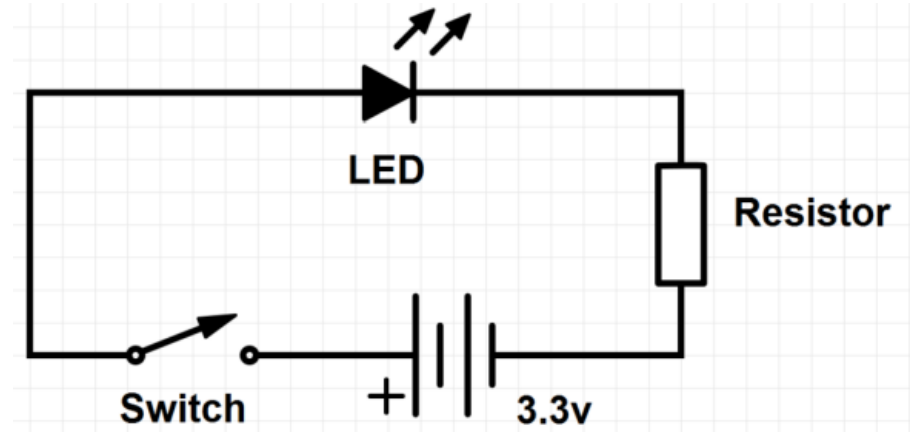
Notes:

1. Wavelength: $\pm 1\text{nm}$.

2. Forward Voltage: $\pm 0.1\text{V}$.

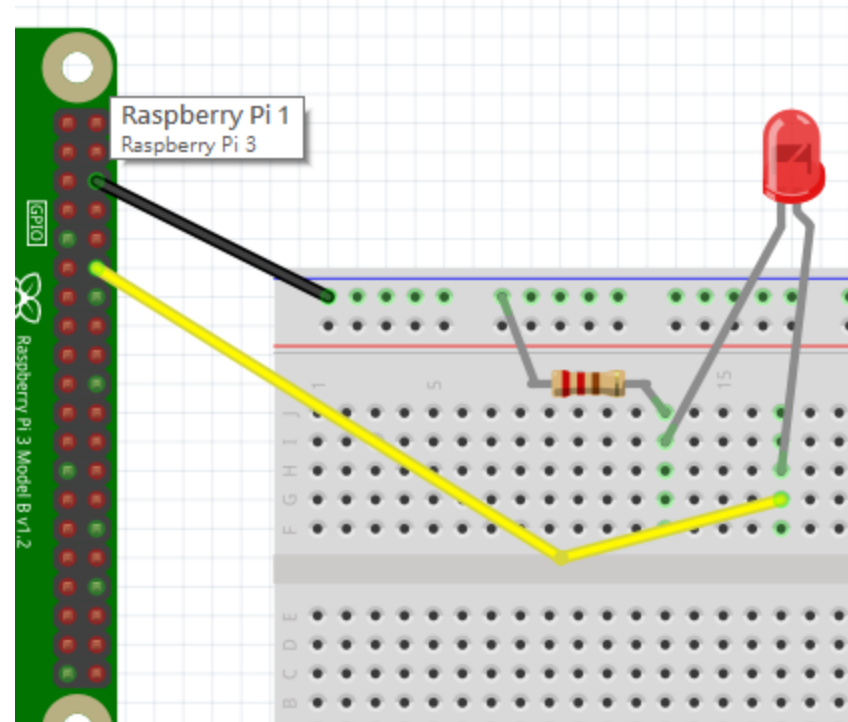
LED

- Parameters
 - GPIO provides 3.3v
 - Forward voltage of LED is 1.85v
 - Current for LED is 20mA
- Ohm's law
 - $I = V/R$
- The resistor which make 20mA is
 - $R = (3.3 - 1.85) / 0.02 = 72.5 \text{ Ohm}$
 - If you use a resistor which is smaller than 72.5 Ohm, the current will be larger than 20mA and it may damage the LED.



Blink

- Circuit
 - PIN 12 - LED(+)
 - PIN 6 - LED(-)



Blink

```
1 import RPi.GPIO as GPIO
2 import time
3
4 LED_PIN = 12
5 GPIO.setmode(GPIO.BOARD)
6 GPIO.setup(LED_PIN, GPIO.OUT)
7 try:
8     while True:
9         print("LED in on.")
10        GPIO.output(LED_PIN, GPIO.HIGH)
11        time.sleep(1)
12        print("LED is off.")
13        GPIO.output(LED_PIN, GPIO.LOW)
14        time.sleep(1)
15 except KeyboardInterrupt:
16     print("Exception: KeyboardInterrupt")
17 finally:
18     GPIO.cleanup()
```

Load GPIO library

LED is on pin 12 by physical pin numbering (z-shape)

the try clause (the statement(s) between the try and except keywords) is executed.

a user-generated interruption is signaled (ctrl + c)

A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

Syntax

- `GPIO.setmode(GPIO.BOARD)` # two parameters
 - `GPIO.BOARD`: Define the pins by the number of the pin plug (**z-shape**)
 - `GPIO.BCM`: Define the pins by the "Broadcom SOC channel" number
- `GPIO.setup(LED_PIN, GPIO.OUT)`
 - Set `LED_PIN` to output mode
- `GPIO.cleanup()`
 - clean up all the ports you've used
- Notice
 - when you have a port set HIGH as an output and you accidentally connect it to GND (LOW), which would short-circuit the port and possibly fry it.

How to Run on RPi

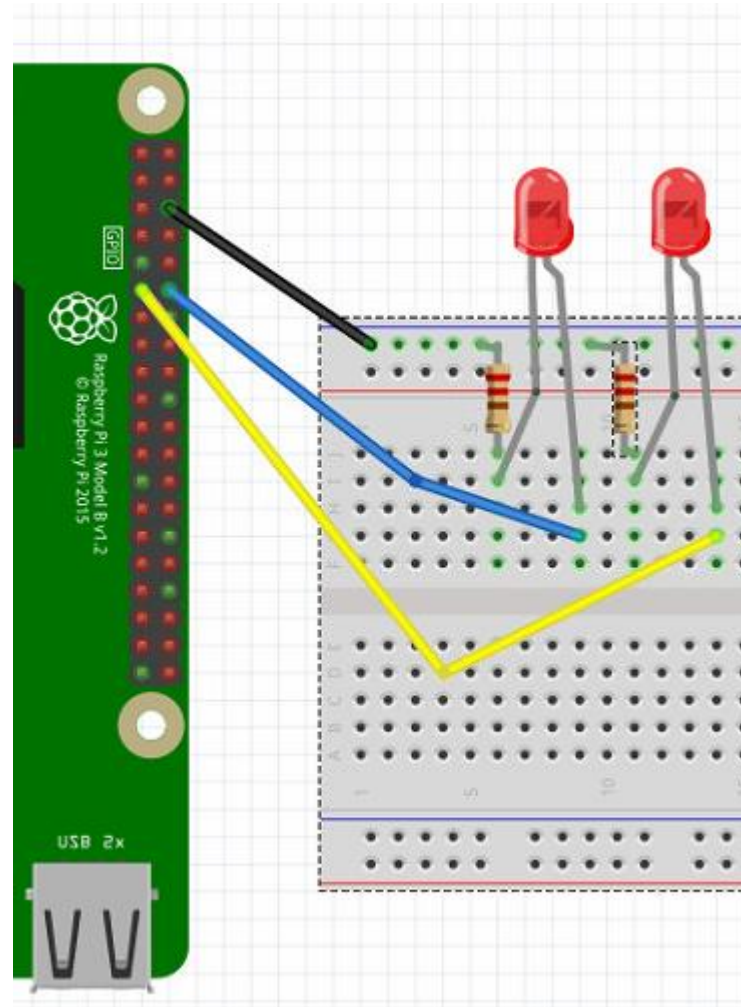
- Write the code with editor on RPi (ex: nano, vim, ... etc).
 - Ex: `$ nano blink.py`
- Write the code with editor on PC and then transmit to RPi via WinSCP.
- Execution

`$ python3 blink.py`

```
pi@rpi-embedded:~ $ python3 blink.py
LED in on.
LED is off.
LED in on.
LED is off.
^CException: KeyboardInterrupt
```

Blink 2 LEDs

- Circuit



Blink 2 LEDs

- Blink two LEDs alternatively.

\$ python3 blink2.py

```
import RPi.GPIO as GPIO
import time

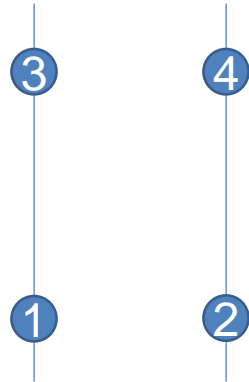
LED_PINS = [12, 11]
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_PINS[0], GPIO.OUT)
GPIO.setup(LED_PINS[1], GPIO.OUT)
try:
    counter = 1
    while True:
        for i in range(2):
            if (counter >> i) & 0x00000001:
                GPIO.output(LED_PINS[i], GPIO.HIGH)
            else:
                GPIO.output(LED_PINS[i], GPIO.LOW)
            counter = counter << 1
        if counter > 2:
            counter = 1
        time.sleep(1)
except KeyboardInterrupt:
    print("Exception: KeyboardInterrupt")
finally:
    GPIO.cleanup()
```

Outline

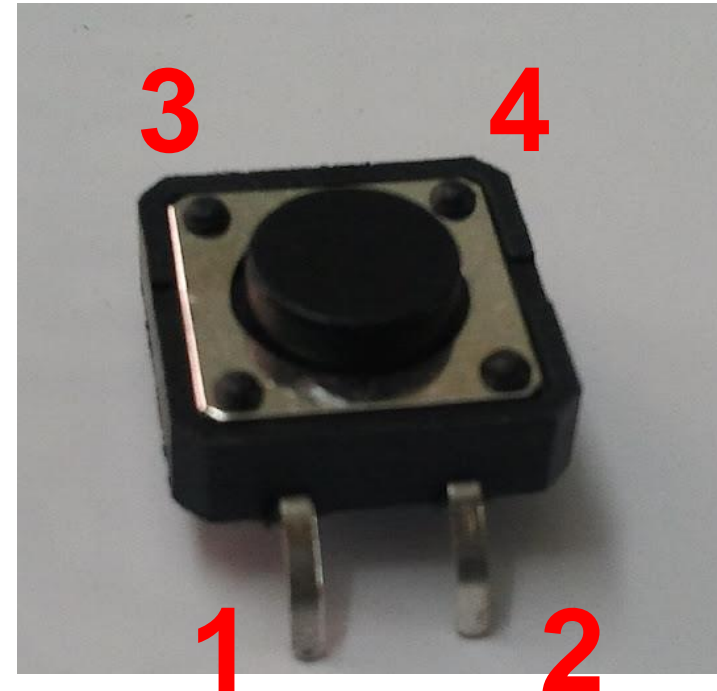
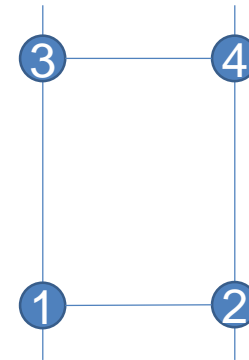
- GPIO Introduction
- GPIO Output
- GPIO Input
 - Polling
 - Interrupt
- Lab

Push button

- There are 4 pins in a button.
- Relationship (connect, disconnect) between these pins?
 - Before press

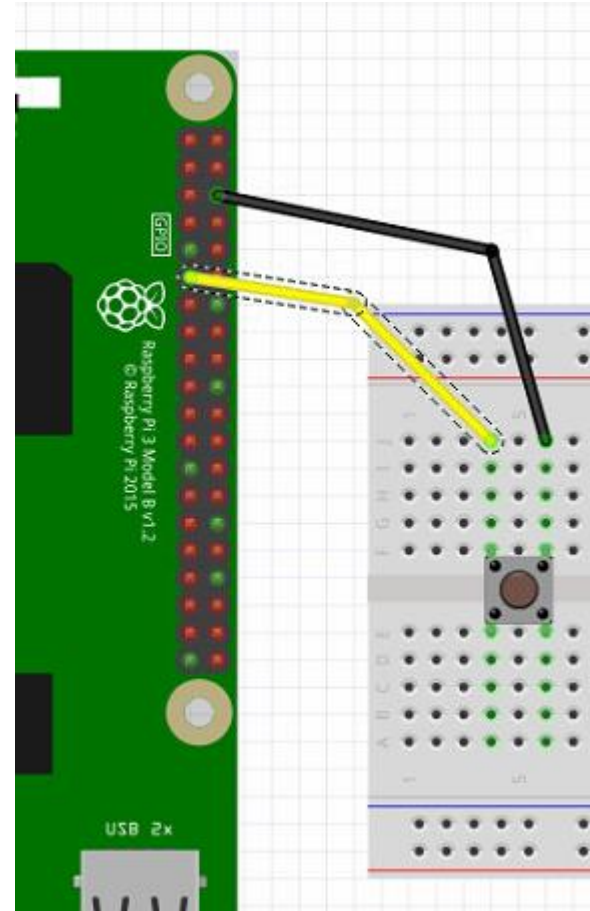


- After press

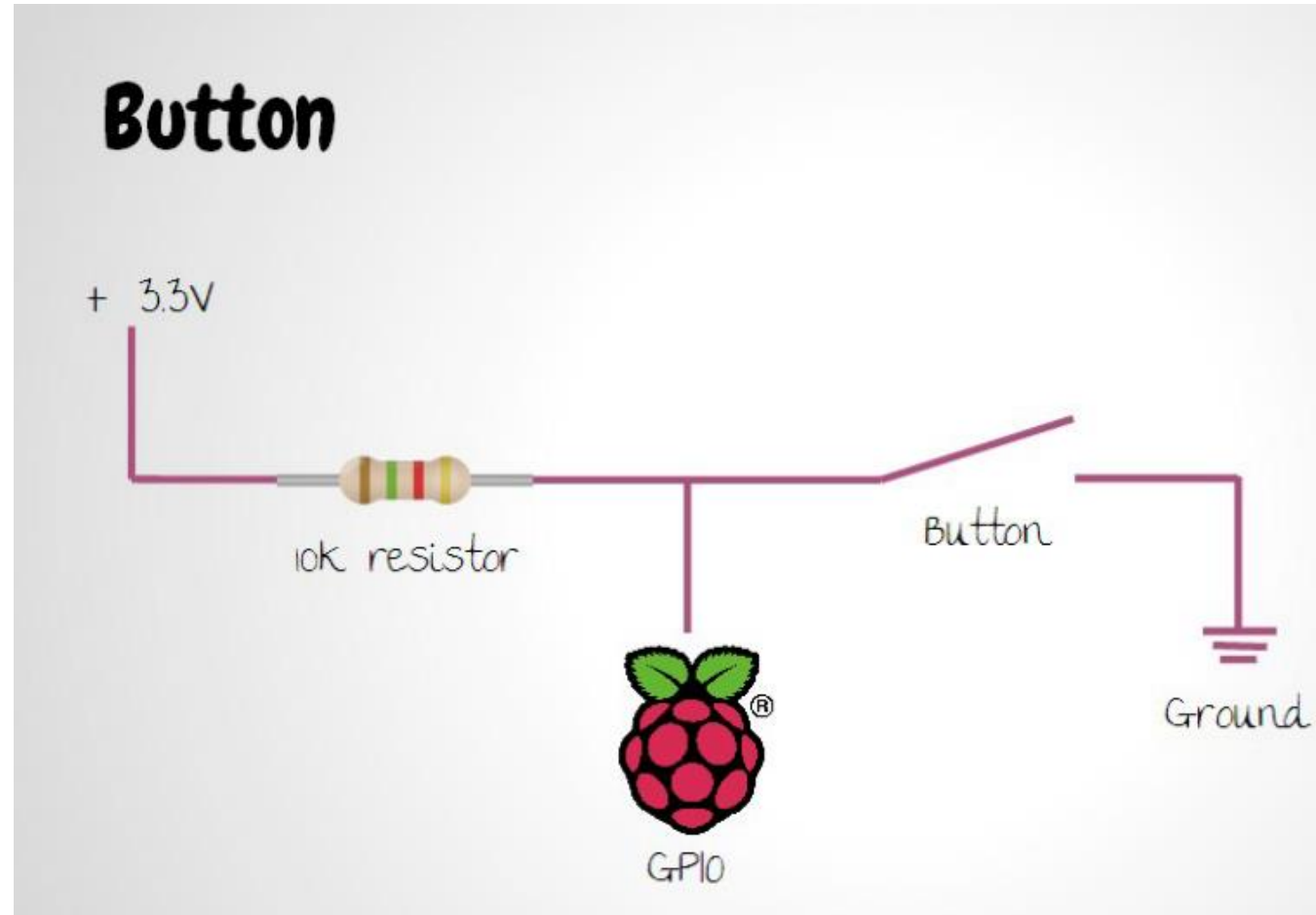


Circuit

- PIN 11 - Pushbutton
- GND - Pushbutton



Hardware Pull-Up



Push Button

- We use internal pull-up resistor.

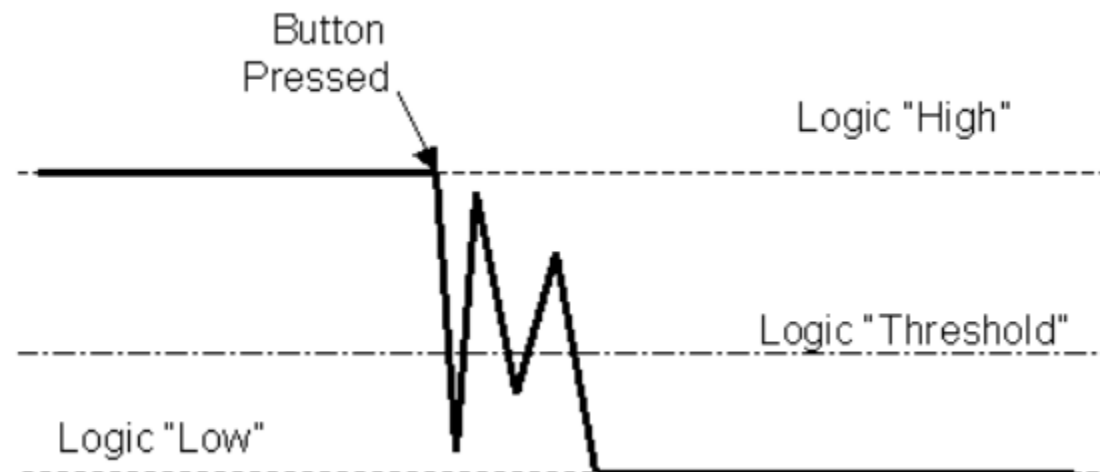
\$ python3 button.py

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BOARD)
5  BTN_PIN = 11
6  GPIO.setup(BTN_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
7  previousStatus = None
8
9  try:
10     while True:
11         input = GPIO.input(BTN_PIN)
12         if input == GPIO.LOW and previousStatus == GPIO.HIGH:
13             print("Button pressed @", time.ctime())
14             previousStatus = input
15     except KeyboardInterrupt:
16         print("Exception: KeyboardInterrupt")
17
18     finally:
19         GPIO.cleanup()
```

Bounce Problem

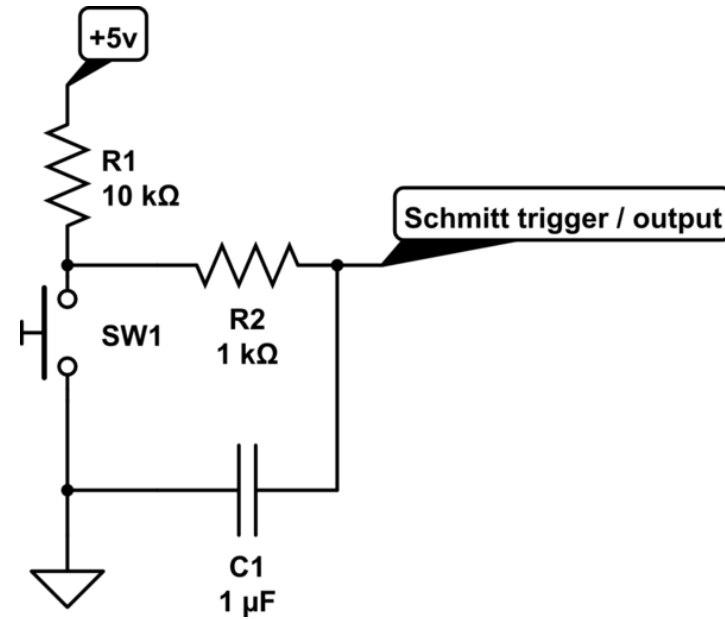
- Press one time with multiple print messages.

```
pi@rpi-embedded:~ $ python3 button.py
Button pressed @ Fri Oct 4 02:46:21 2019
Button pressed @ Fri Oct 4 02:46:22 2019
Button pressed @ Fri Oct 4 02:46:23 2019
Button pressed @ Fri Oct 4 02:46:24 2019
Button pressed @ Fri Oct 4 02:46:24 2019
Button pressed @ Fri Oct 4 02:46:24 2019
```



Debounce

- Hardware
- Software
 - Delay the measurement of triggering.



Push Button with Debounce

\$ python3 button_debounce.py

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
BTN_PIN = 11
WAIT_TIME = 0.2
GPIO.setup(BTN_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
previousStatus = None
previousTime = time.time()
currentTime = None

try:
    while True:
        input = GPIO.input(BTN_PIN)
        currentTime = time.time()
        if input == GPIO.LOW and previousStatus == GPIO.HIGH and (currentTime - previousTime) > WAIT_TIME:
            previousTime = currentTime
            print("Button pressed @", time.ctime())
            previousStatus = input

except KeyboardInterrupt:
    print("Exception: KeyboardInterrupt")

finally:
    GPIO.cleanup()
```

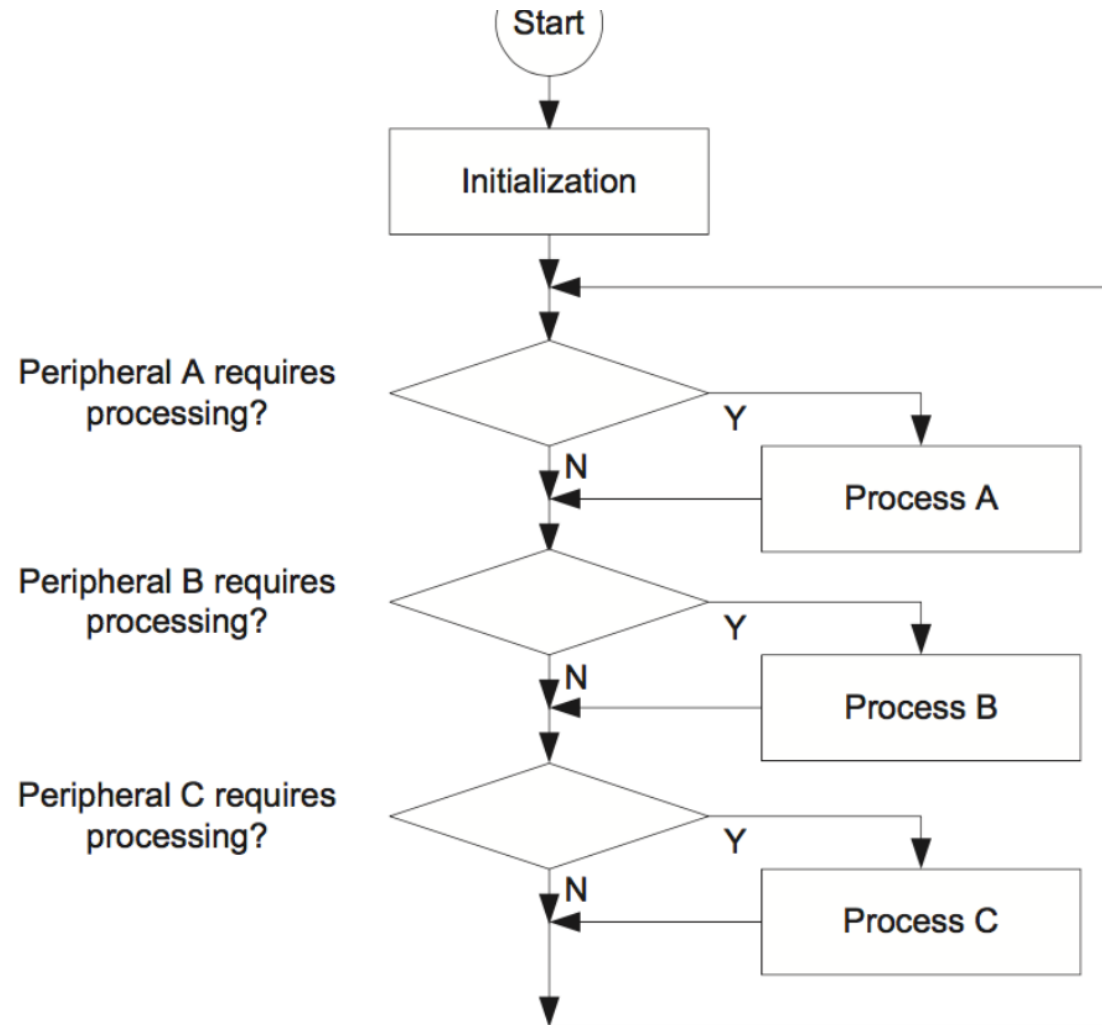
Outline

- GPIO Introduction
- GPIO Output
- GPIO Input
 - Polling
 - Interrupt
- Lab

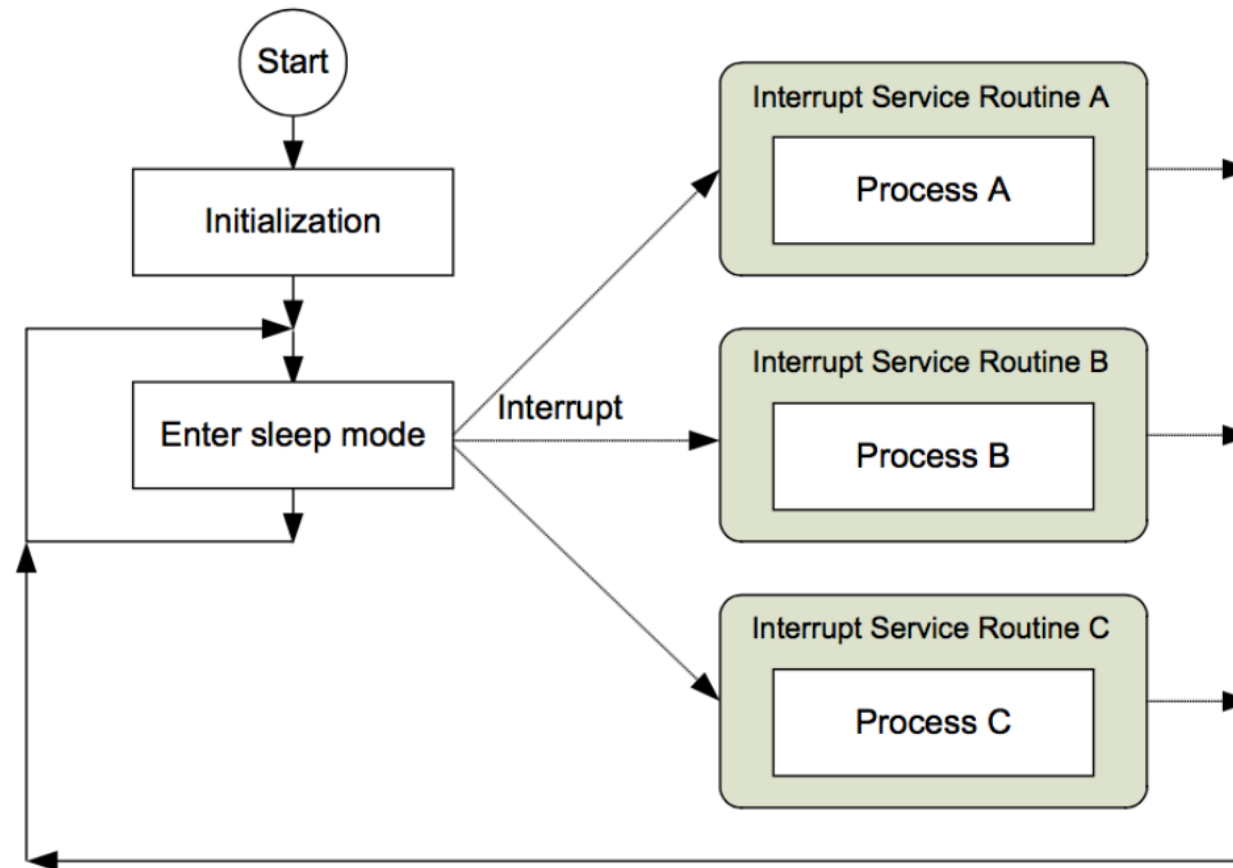
Interrupt

- Previously, we do polling by software to check the state of button.
- How could you handle the following tasks **at the same time**?
 - User inputs
 - Display (including animation)
 - Timeout (including timer)
 - Music
 - Network
 - Incoming calls and LINE messages

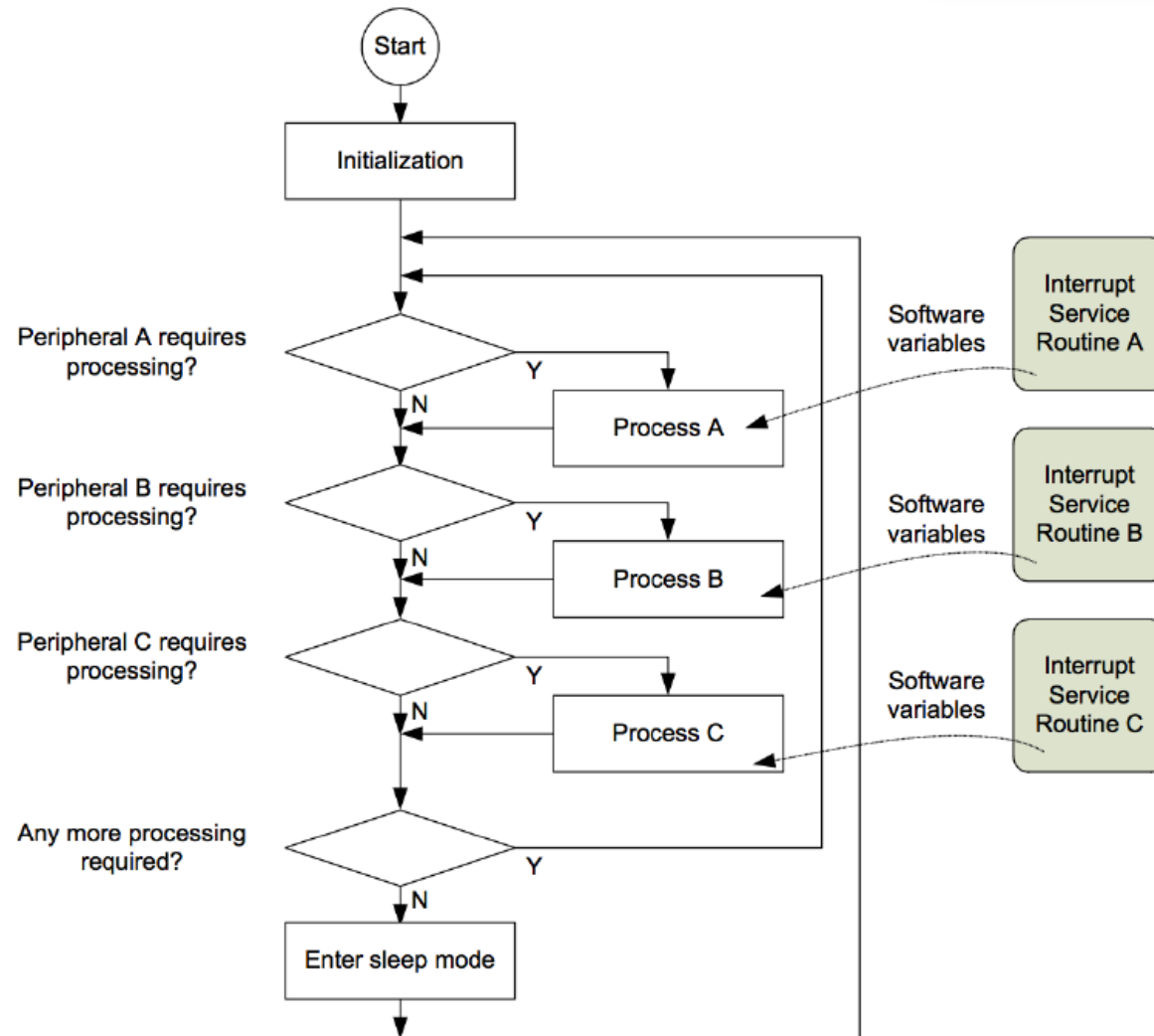
Application Polling



Interrupt Driven



Application-Combined



Exception

- Exceptions are events that cause changes in program flow control **outside a normal code sequence**.
- When it happens, the program that is currently executing is **suspended**, and a piece of code associated with the event (the **exception handler**) is **executed**.
- The events could either be external or internal.
- When an event is from an external source
 - commonly known as an **interrupt** or **interrupt request (IRQ)**.

BCM2711

■ Interrupts

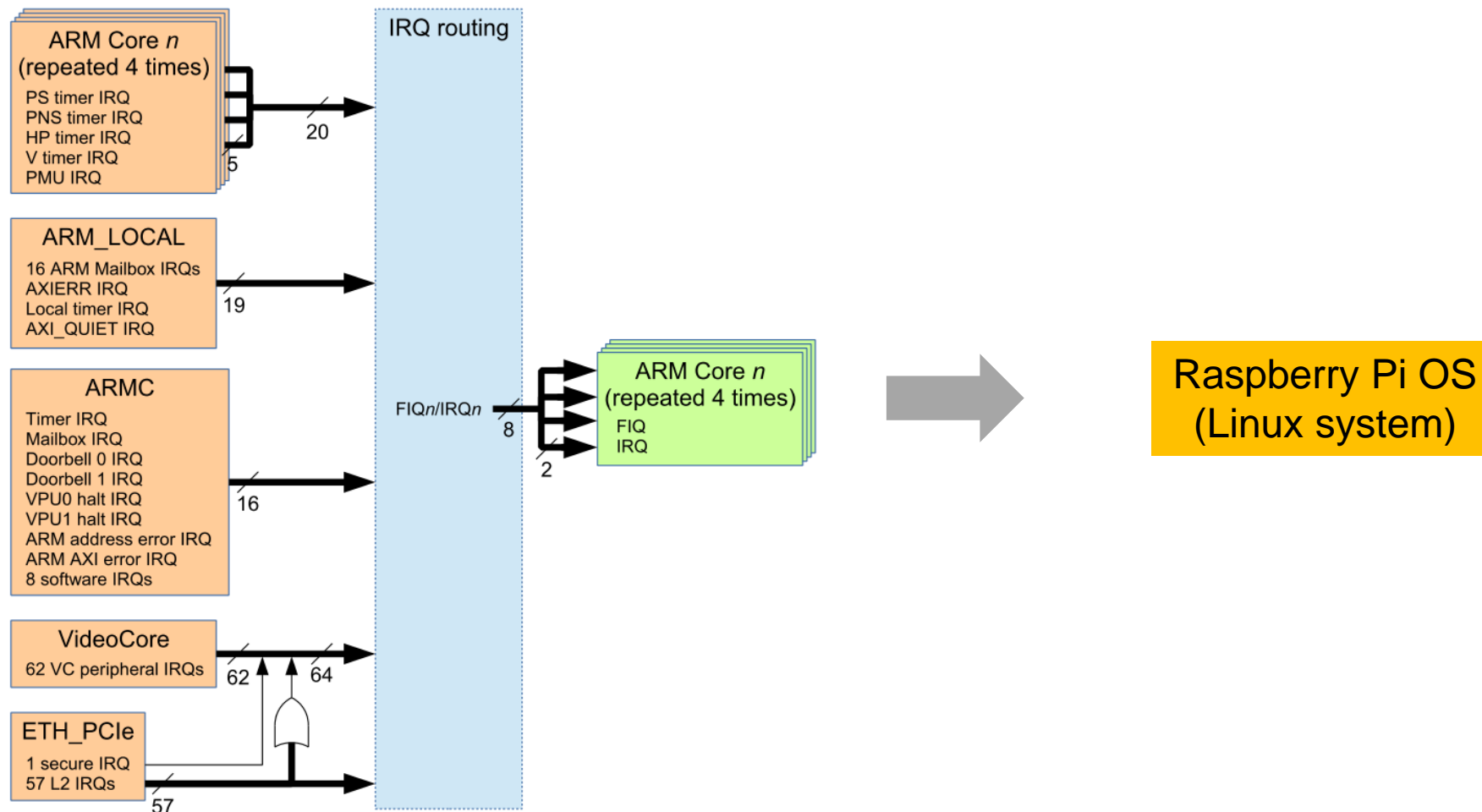
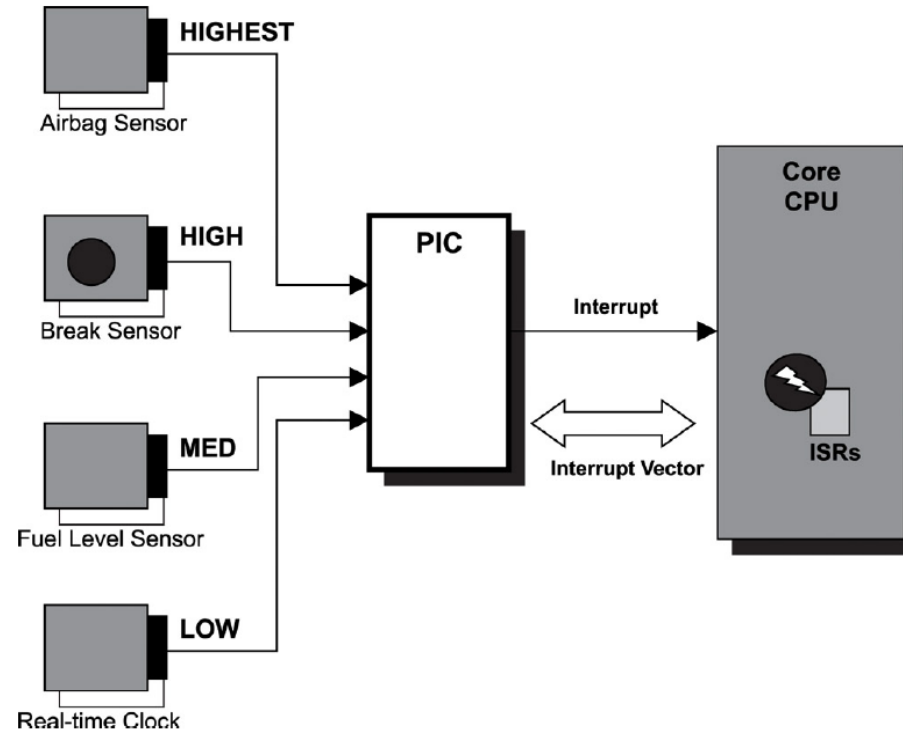


Figure 5. Interrupt sources and destinations

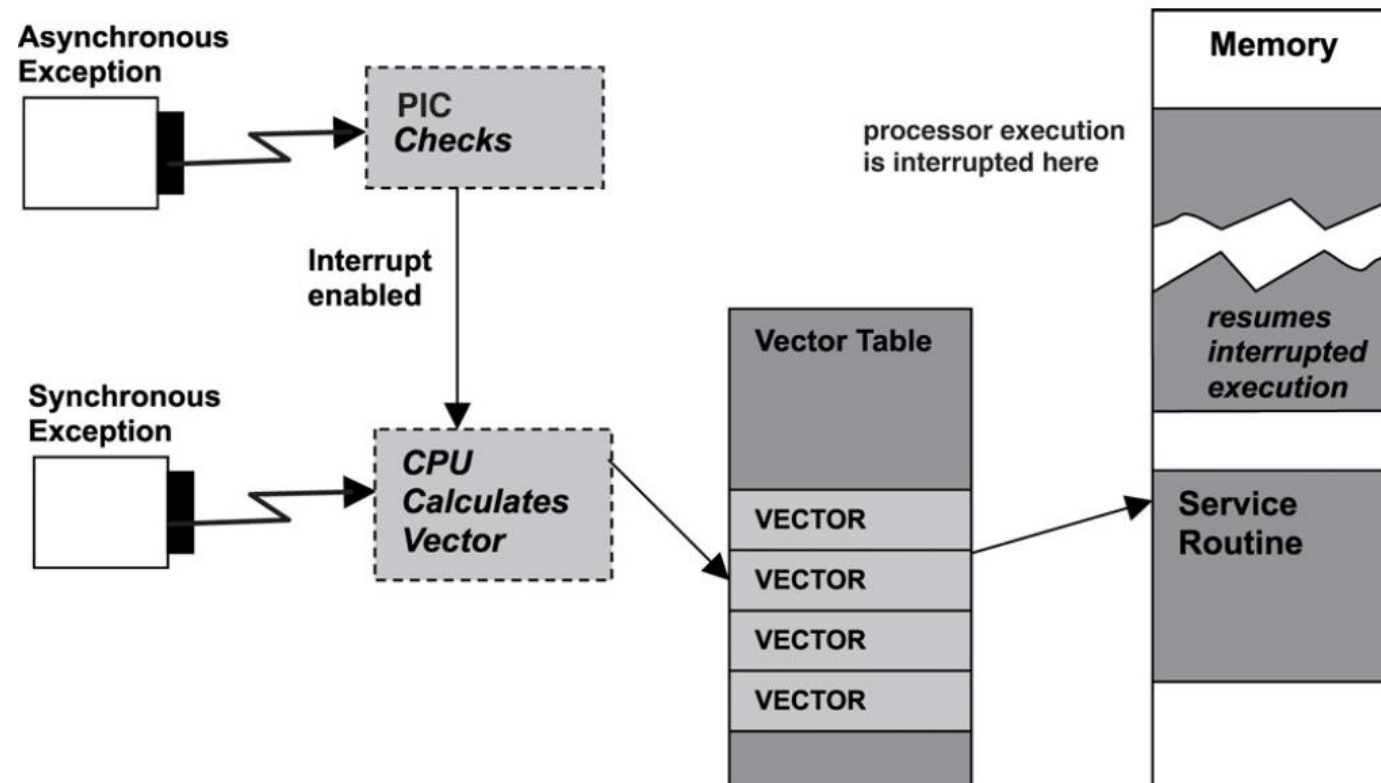
Interrupt Vector (1/2)

- How do exceptions and interrupts work ?
 - Programmable interrupt controller



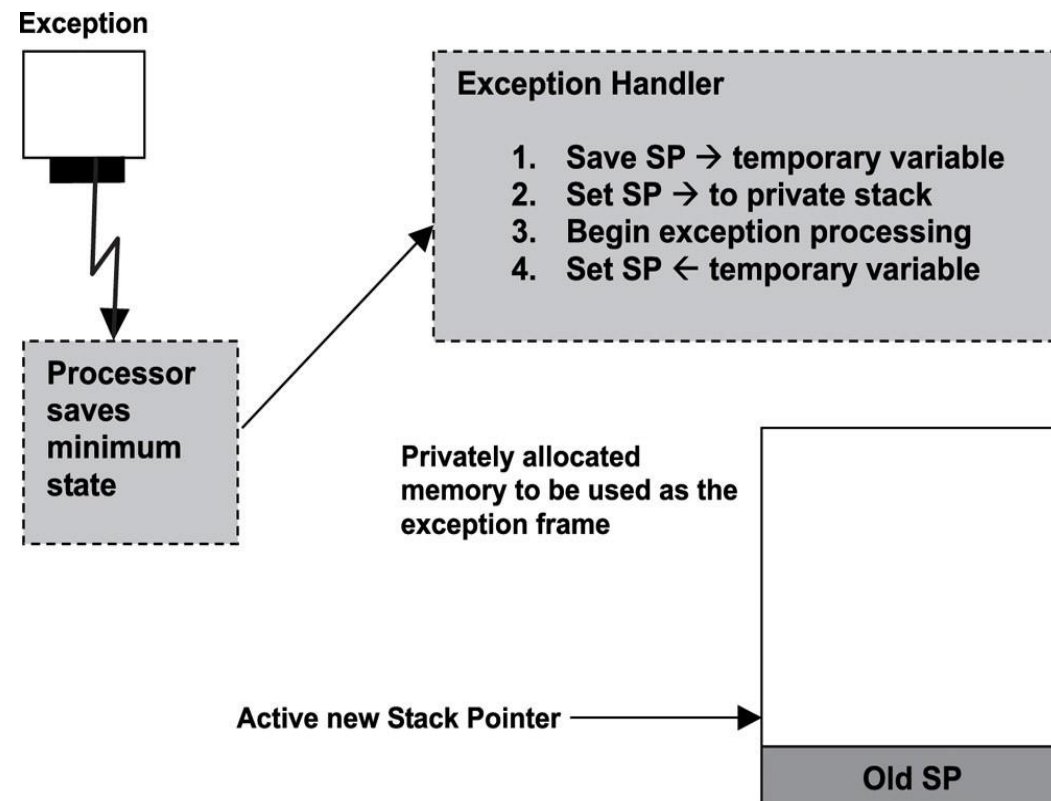
Interrupt Vector (2/2)

- Processing general exceptions
 - Loading and invoking exception handlers



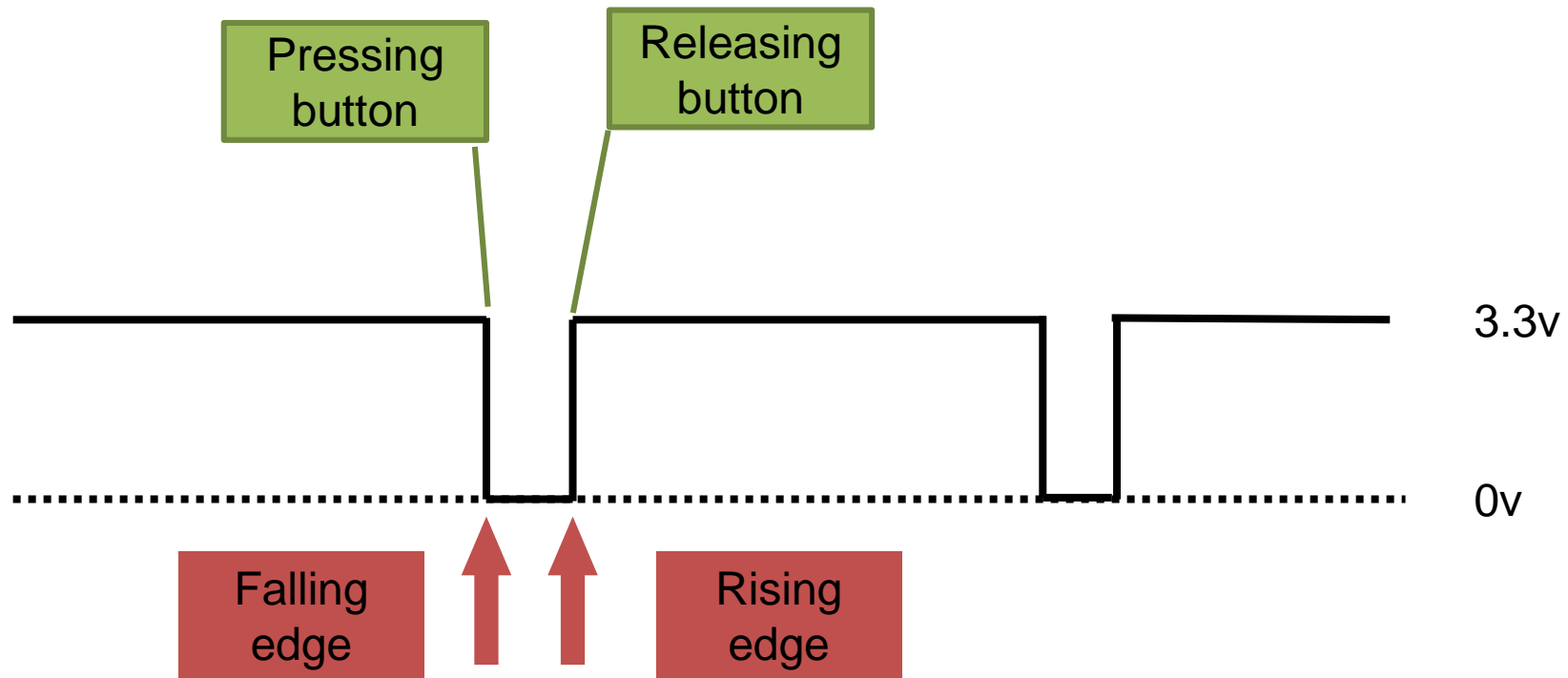
Exception Handlers

- Exception frame
 - The exception frame is also called the **interrupt stack** in the context of asynchronous exceptions.



Triggering

- When to trigger an interrupt?



Interrupts and Edge Detection

- <https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/>
- wait_for_edge() function

```
# wait for up to 5 seconds for a rising edge (timeout is in milliseconds)
channel = GPIO.wait_for_edge(channel, GPIO.RISING, timeout=5000)
if channel is None:
    print('Timeout occurred')
else:
    print('Edge detected on channel', channel)
```

- event_detected() function

```
GPIO.add_event_detect(channel, GPIO.RISING) # add rising edge detection on a channel
do_something()
if GPIO.event_detected(channel):
    print('Button pressed')
```

- Threaded callbacks
 - RPi.GPIO runs a second thread for callback functions.

button_int.py

```
import RPi.GPIO as GPIO
import time

def ButtonPressed(btn):
    print("Button pressed @", time.ctime())

GPIO.setmode(GPIO.BOARD)
BTN_PIN = 11
GPIO.setup(BTN_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(BTN_PIN, GPIO.FALLING, ButtonPressed, 200)

try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("Exception: KeyboardInterrupt")

finally:
    GPIO.cleanup()
```

\$ python3 button_int.py

- You may have to **restore the indentation** of python code after copying.

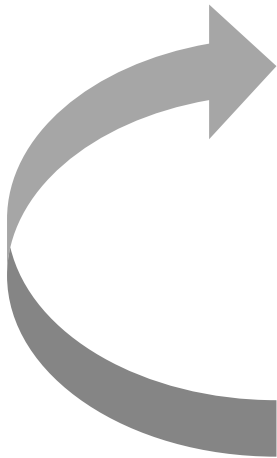
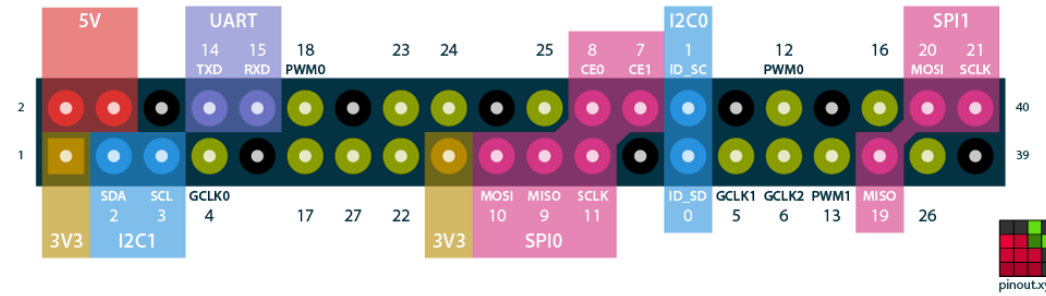
Multithreading

- RPi has multiple cores in CPU.
- Software multithreading is another methods for processing these tasks simultaneously.
- In python, it supports thread-based parallelism.
 - <https://docs.python.org/3/library/threading.html>

CS348A Lab

- Combine LED and push button.
- Choose **different PINs** to replace PIN 12, 11 for the LED and button.
- Push the button to turn on or off. (on -> off or off -> on)
- Change to the next pattern** when the blinking function is turned on. (1 -> 2 -> 3 -> 1 -> 2 ...)
- Demonstrate it to TA.

Raspberry Pi GPIO BCM numbering



Pattern 1



Blink every 2 second.

Pattern 2



Blink every 1 seconds.

Pattern 3

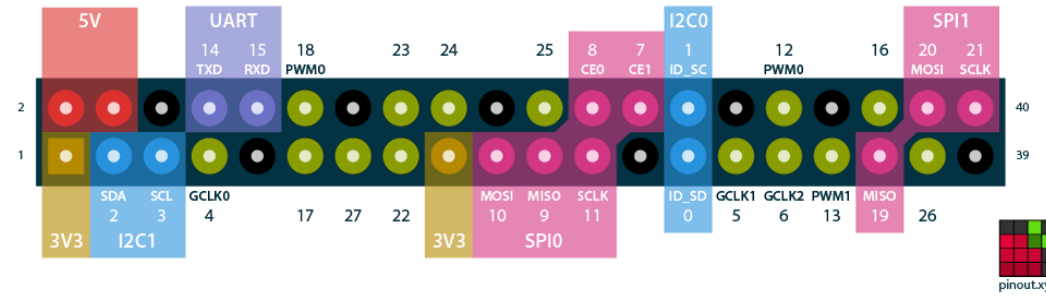


Blink every 0.5 seconds.

CS348B Lab

- Combine LED and push button.
- Choose **different PINs** to replace PIN 12, 11 for the LED and button.
- Push the button one time to **switch the LED pattern**. (1 -> 2 -> 3 -> 4 -> 1 -> 2 ...)
- Demonstrate it to TA.

Raspberry Pi GPIO BCM numbering



Pattern 1



No LED

Pattern 2



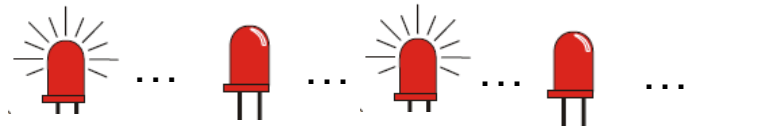
Blink every 1 second.

Pattern 3



Blink every 0.5 seconds.

Pattern 4



Blink every 0.25 seconds.

Reference

- BCM2711 ARM Peripherals
 - https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0.pdf
- raspberry-gpio-python
 - <https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>