Cyber-Physical Systems

Basic I/O with RPi

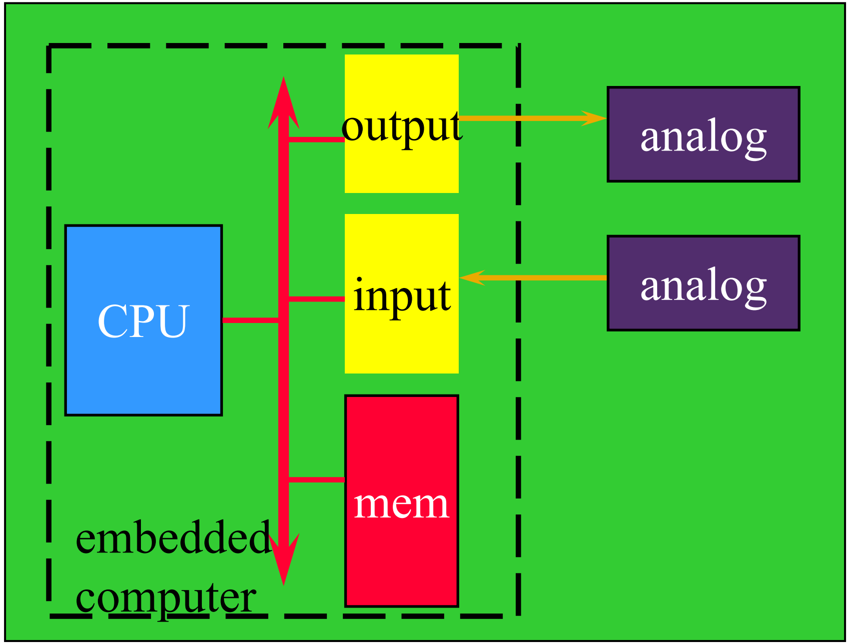
UNIVERSITY AT ALBANY
State University of New York

IECE 553/453– Fall 2021

Prof. Dola Saha

Embedded System

- Embedded computing system: any device that includes a processing system but is NOT a general-purpose computer.
- Often application specific: takes advantage of application characteristics to optimize the design
- Might have real-time requirements
- Might be power constrained



Connecting Analog and Digital Worlds

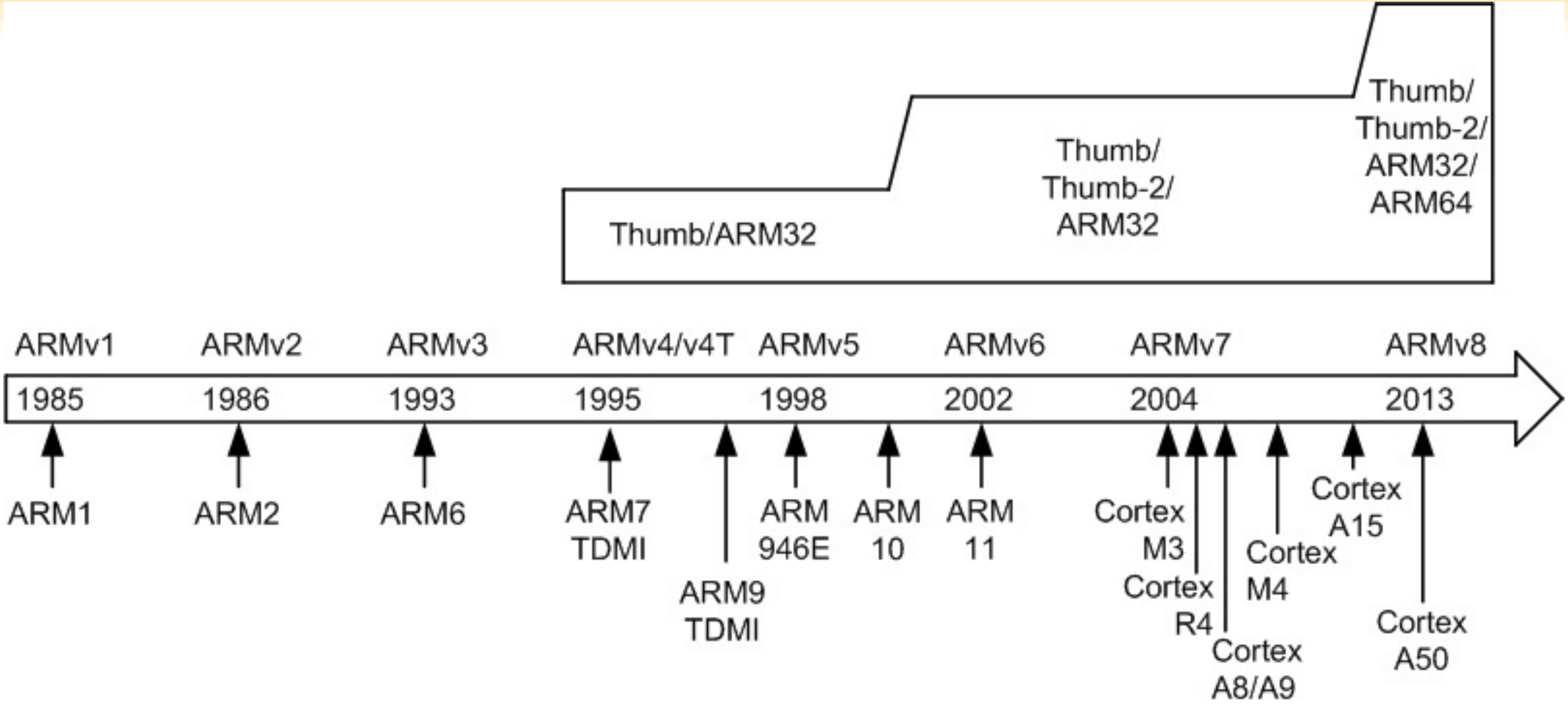
- Cyber
  - Digital
  - Discrete in Time
  - Sequential

- Physical
  - Continuum
  - Continuous in time
  - Concurrent

Practical Issues

- Analog vs. digital
- Wired vs. wireless
- Serial vs. parallel
- Sampled or event triggered
- Bit rates
- Access control, security, authentication
- Physical connectors
- Electrical requirements (voltages and currents)

History of ARM Processor



ARM Cortex Processors

ARM Cortex-A family:

Applications processors

Support OS and high-performance applications, such as smartphones, Smart TV

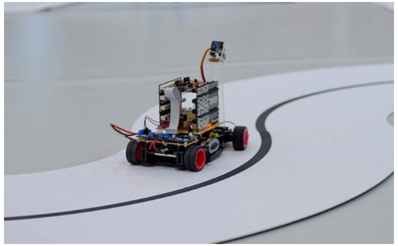
ARM Cortex-R family:

Real-time processors with high performance and high reliability

Support real-time processing and mission-critical control

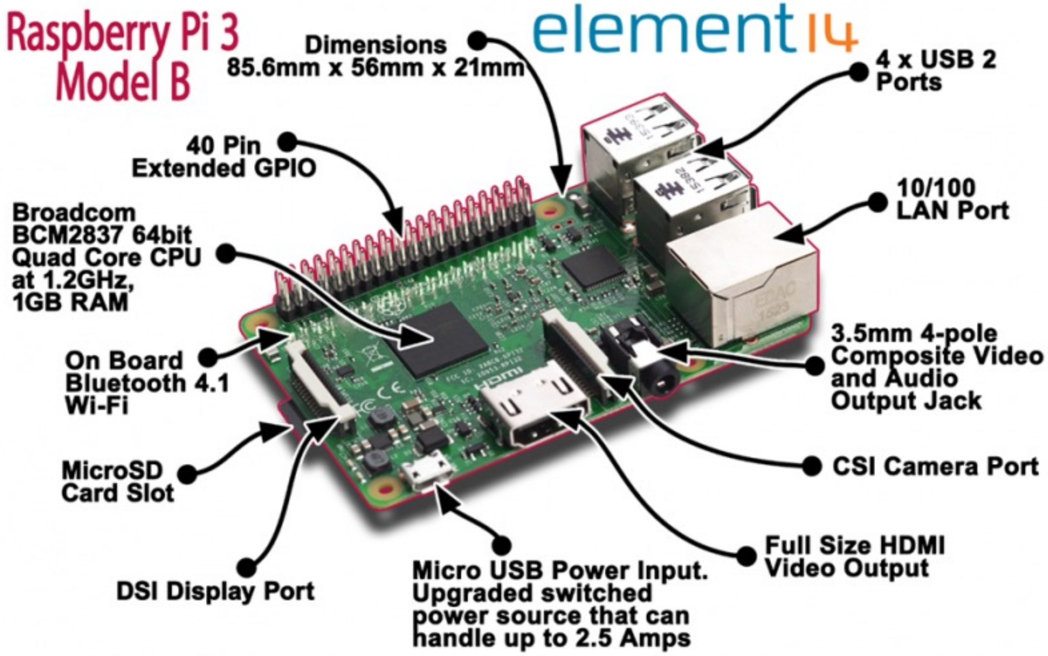
ARM Cortex-M family:

Microcontroller, energy-efficient, cost-sensitive, support SoC



Raspberry Pi – Know your board

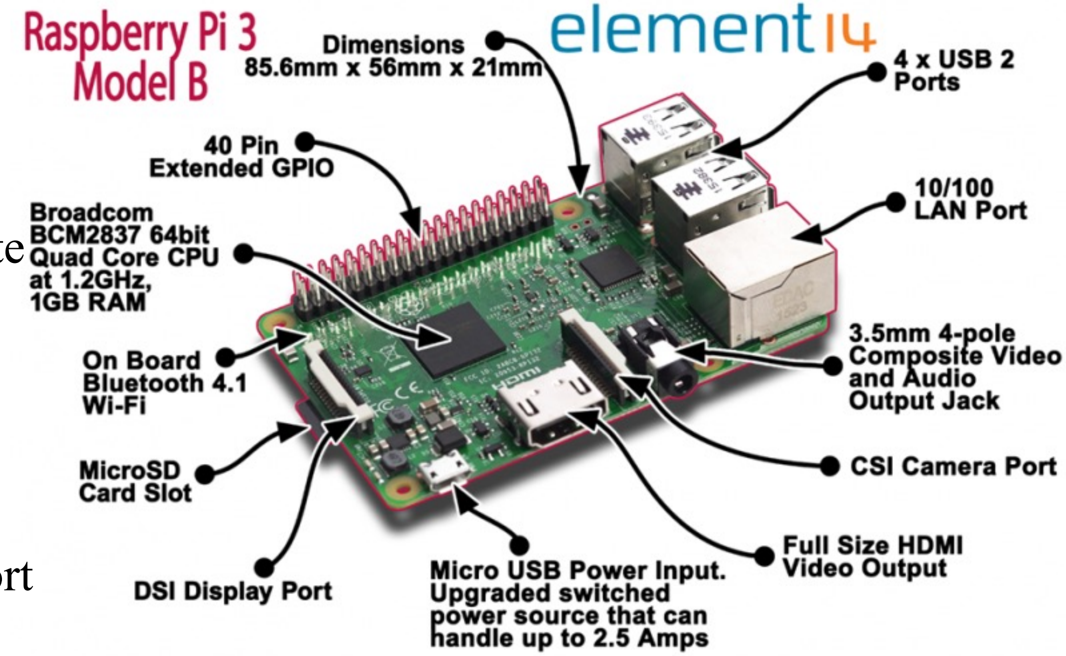
- The Raspberry Pi 3 Model B+
  - Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
  - 1GB LPDDR2 SDRAM
  - 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
  - Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
  - Extended 40-pin GPIO header
  - Full-size HDMI



Raspberry Pi – Know your board

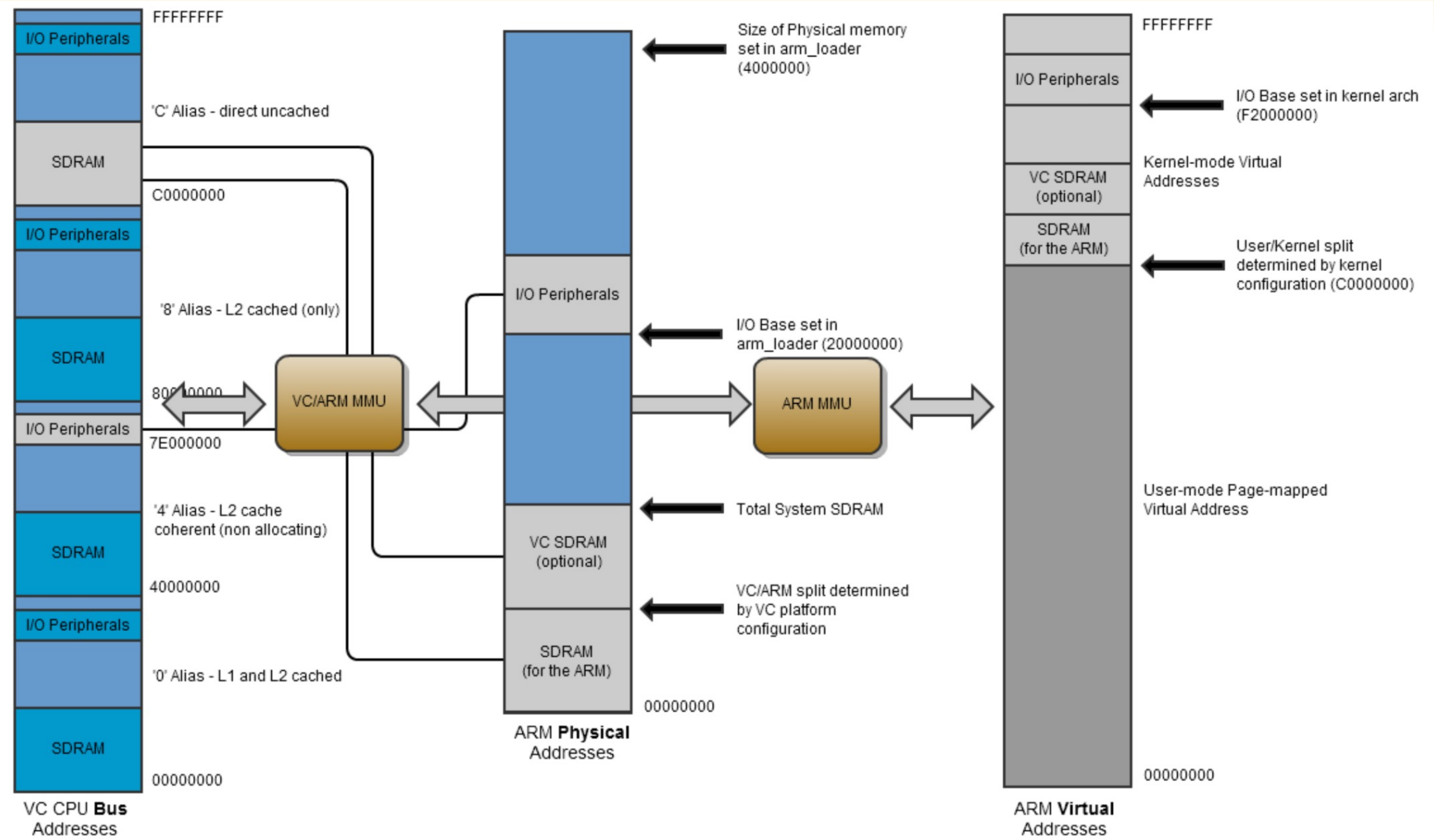
➢ The Raspberry Pi 3 Model B+

- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)



ARM Peripherals

BCM2837 Manual

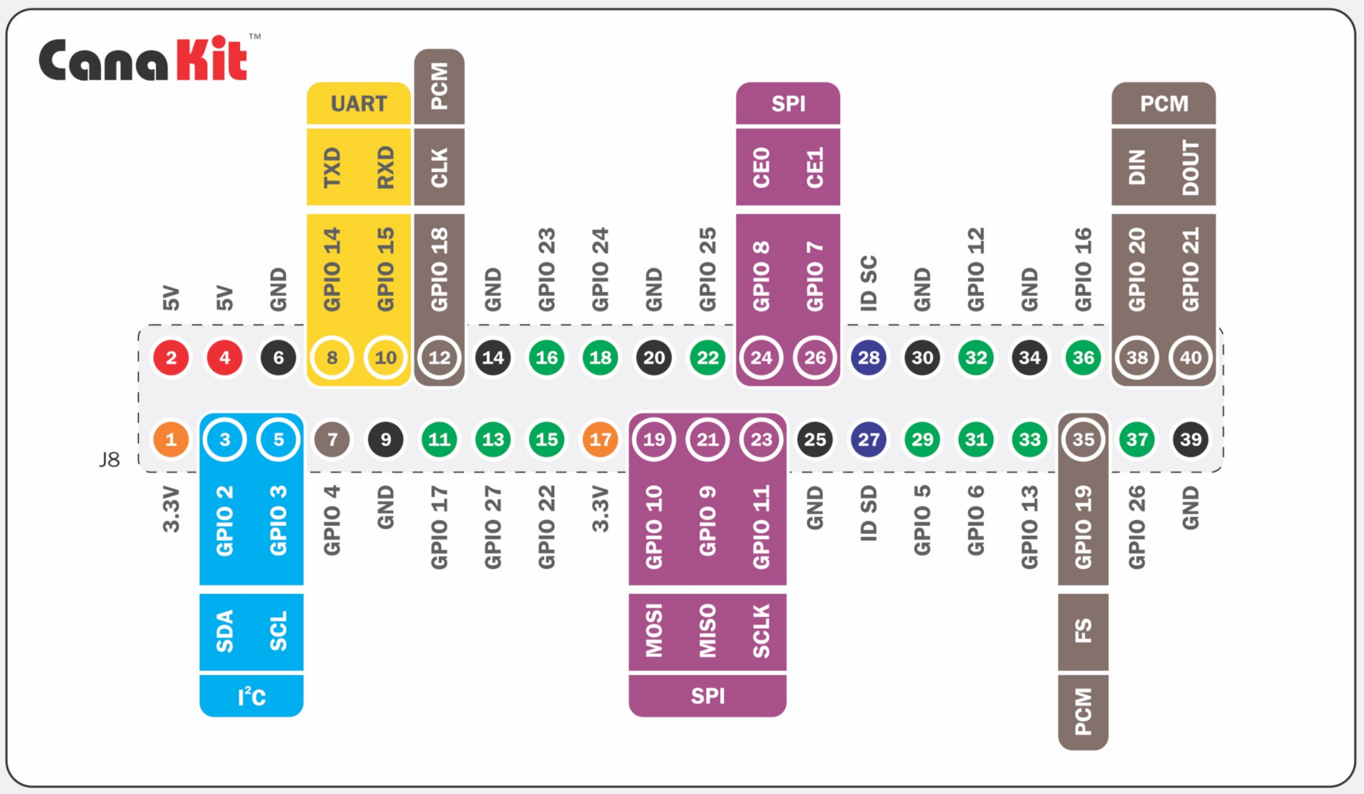


Address Mapping

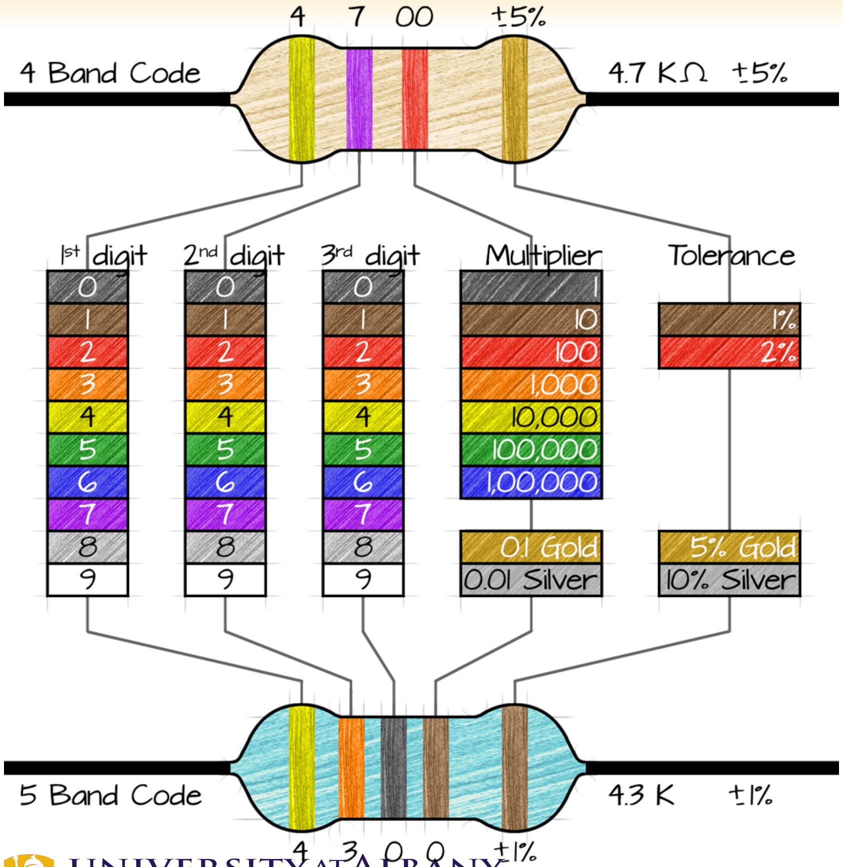
- Addresses in ARM Linux are:
  - issued as virtual addresses by the ARM core,
  - mapped into a physical address by the ARM/MMU,
  - mapped into a bus address by the ARM mapping MMU,
  - used to select the appropriate peripheral or location in RAM.

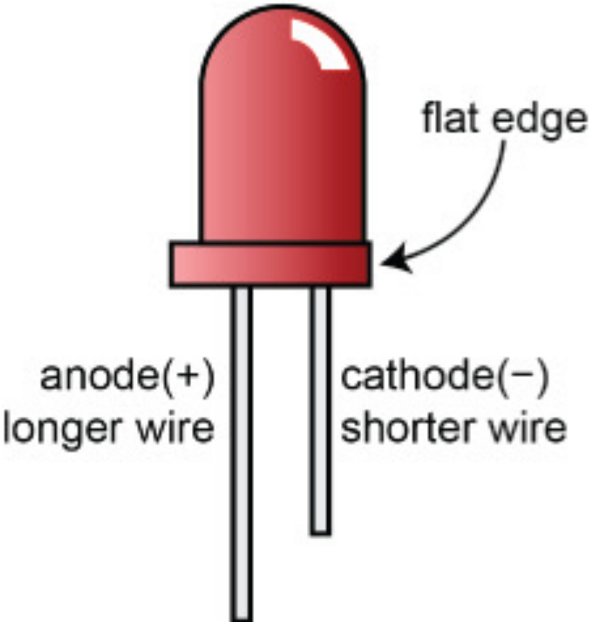
GPIO Pins

- ➢ https://pinout.xyz

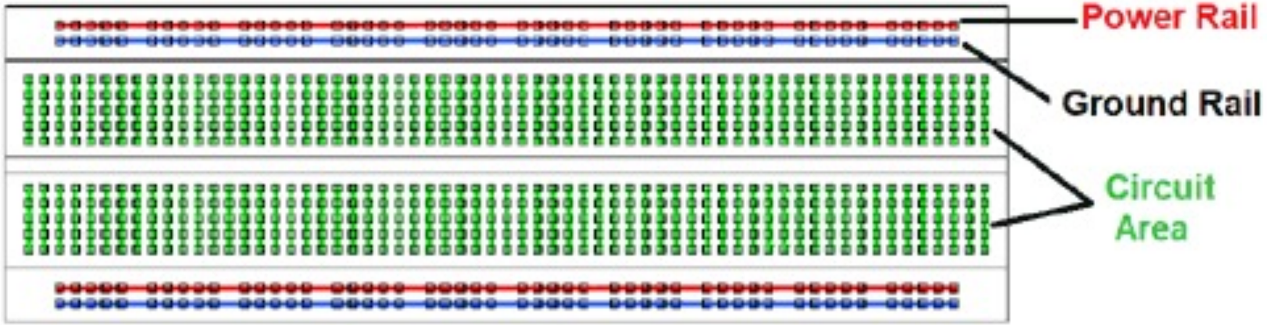


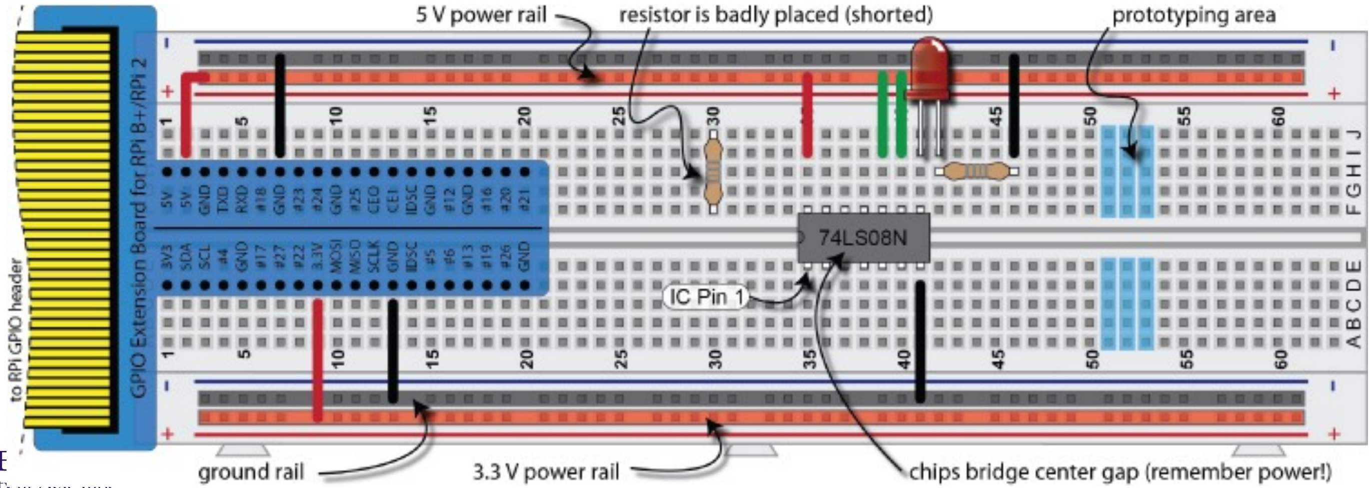
Resistors and LEDs



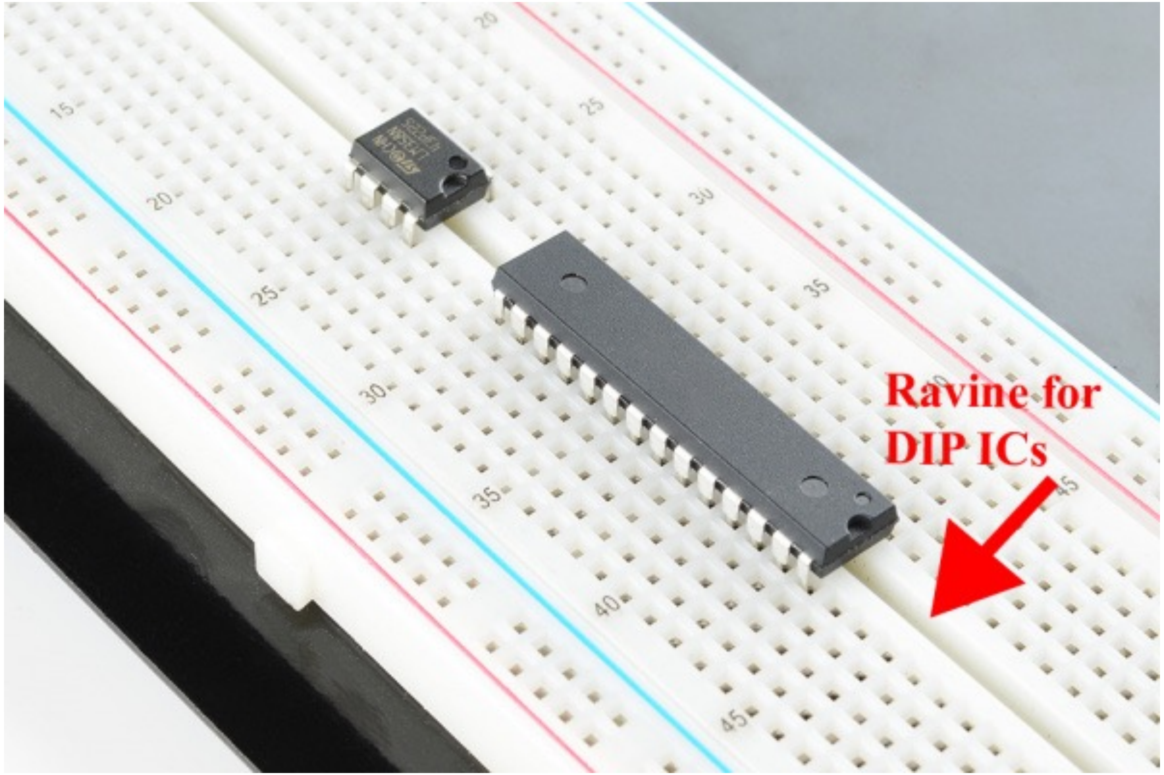


Breadboard Connections



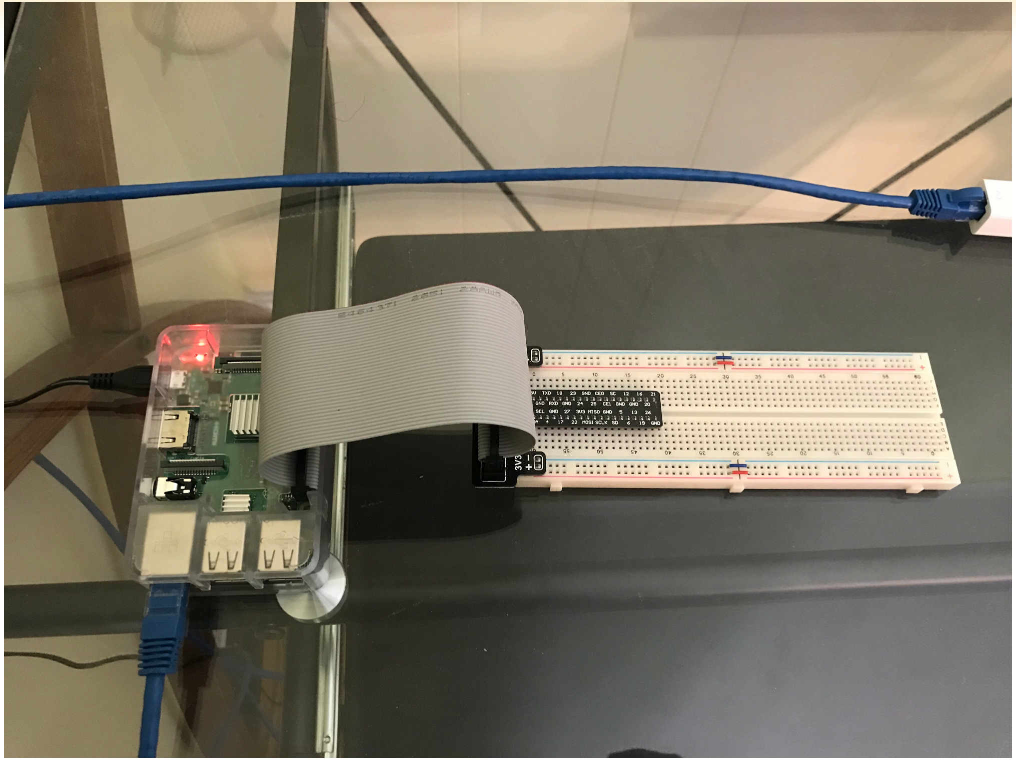


Dual In-Line Package or DIP

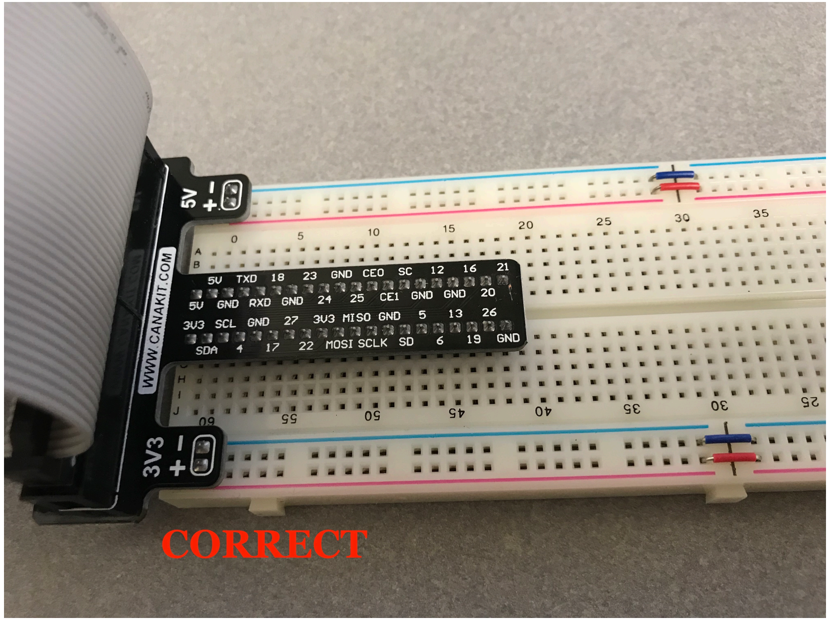


GPIO

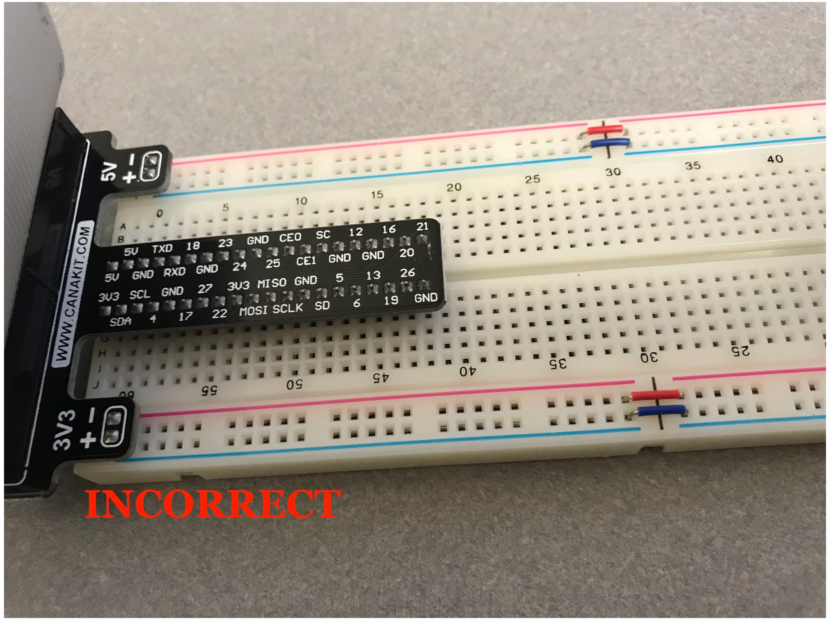
- GPIO to Breadboard Interface Board
- GPIO Ribbon Cable
- Breadboard



Convention



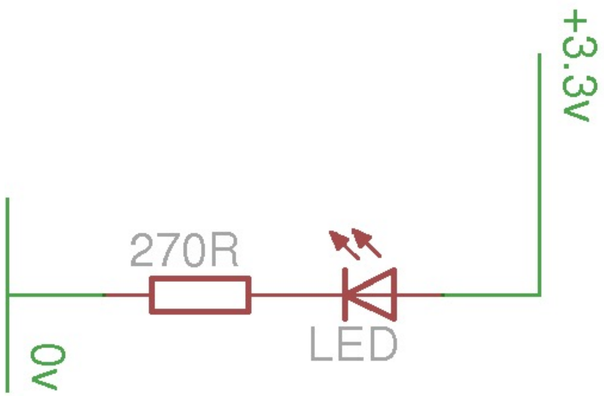
5V
3V3
WWW.CANAKIT.COM
CORRECT

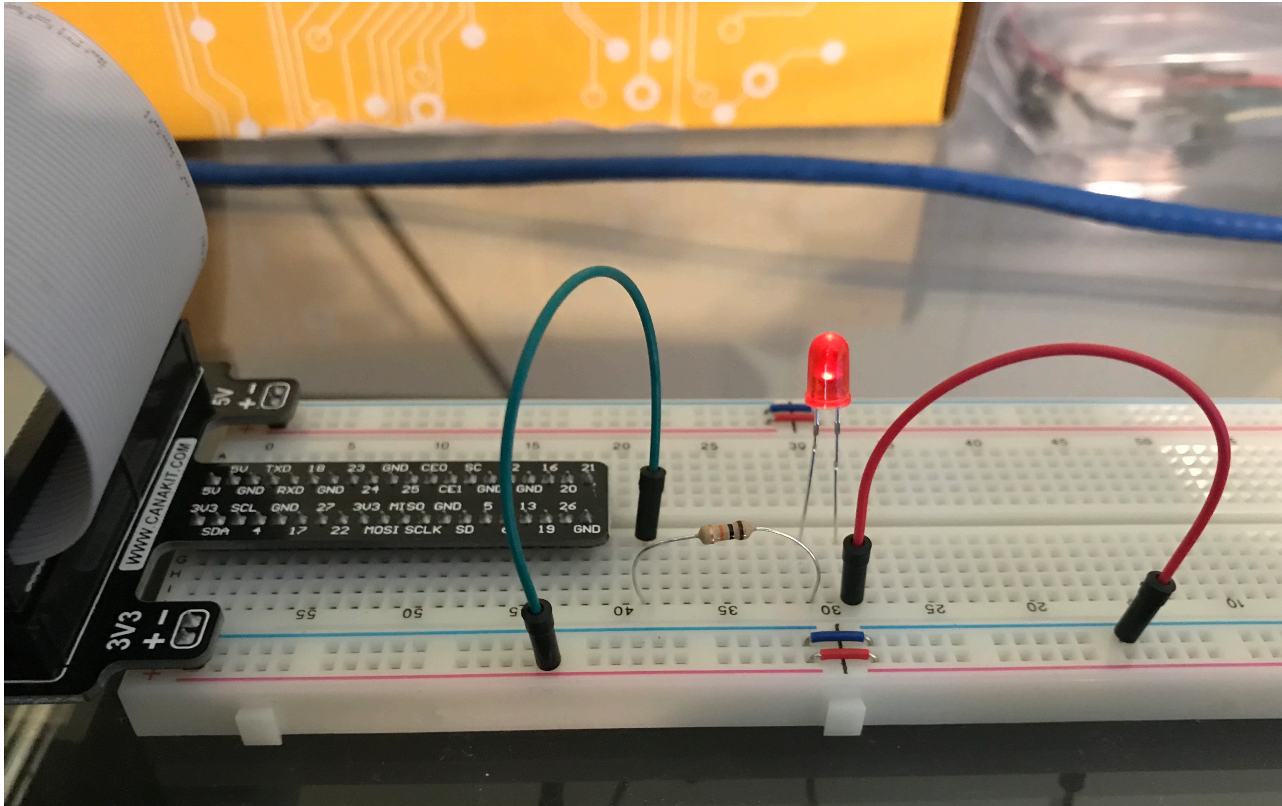


5V
3V3
WWW.CANAKIT.COM
INCORRECT

Circuit to Breadboard

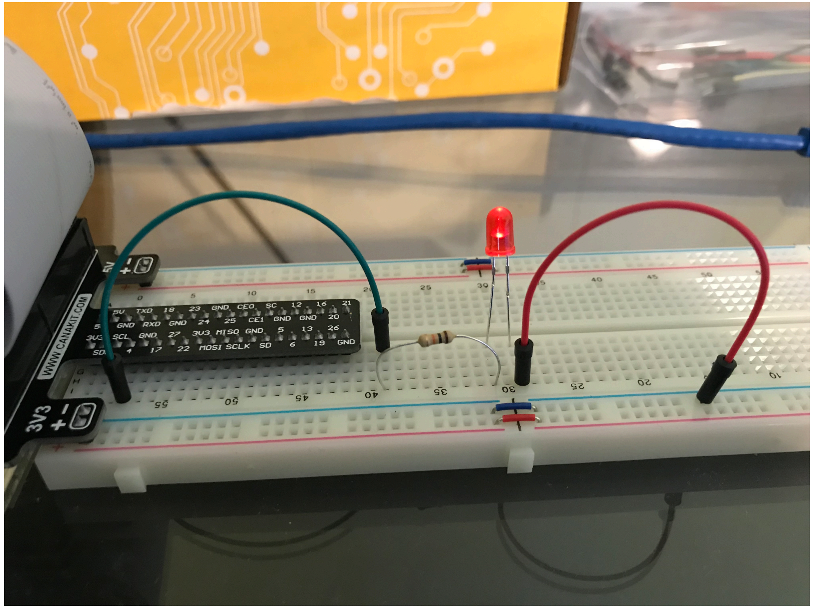
- Use 3V





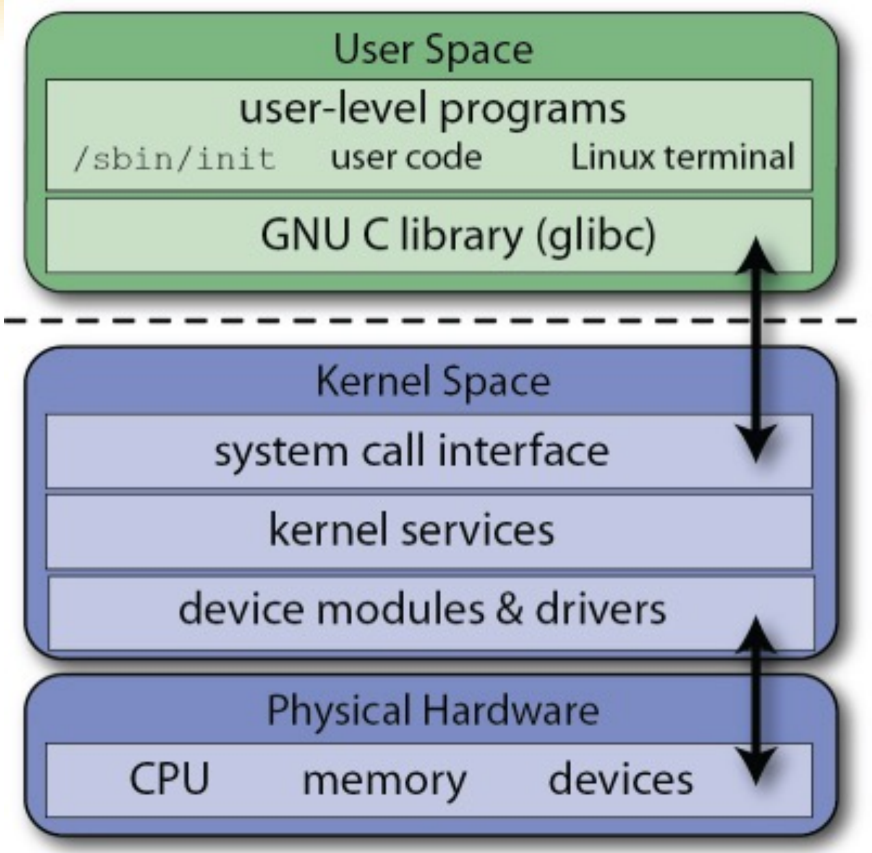
Circuit to Breadboard

- Use GPIO pin



sysfs – pseudo-filesystem

- The **sysfs** filesystem is a pseudo-filesystem which provides an interface to kernel data structures.
- The files under **sysfs** provide information about devices, kernel modules, filesystems, and other kernel components.



Linux Kernel vs User Space

- The Linux kernel runs in an area of system memory called the *kernel space*
- Regular user applications run in an area of system memory called *user space*
- A hard boundary between these two spaces prevents
  - User applications from accessing memory and resources required by the Linux kernel
  - Linux kernel from crashing due to badly written user code
  - Interfering one user's applications with another
  - Provides a degree of security.

sysfs

- Paths in sysfs (/sys/class/gpio)
  - Control interfaces used to get userspace control over GPIOs
    - export
    - unexport
  - GPIOs themselves
  - GPIO controllers ("gpiochip" instances)
- GPIO signals have paths like /sys/class/gpio/gpioN/
  - "direction" - reads as either "in" or "out”
  - "value" - reads as either 0 (low) or 1 (high)
  - "edge" - reads as either "none", "rising", "falling", or "both”
  - "active_low" - reads as either 0 (false) or 1 (true)

Steps to perform I/O using sysfs

- Export the pin.
- Set the pin direction (input or output).
- If an output pin, set the level to low or high.
- If an input pin, read the pin's level (low or high).
- When done, unexport the pin.

Exporting GPIO control to userspace

- "export"
  - Userspace may ask the kernel to export control of a GPIO to userspace by writing its number to this file.
  - Example: "echo 19 > export" will create a "gpio19" node for GPIO #19, if that's not requested by kernel code.
- "unexport"
  - Reverses the effect of exporting to userspace.
  - Example: "echo 19 > unexport" will remove a "gpio19" node exported using the "export" file.

Control GPIO with Linux

- Become the sudo user
  - dsaha@sahaPi:~ $ sudo su
- Go to the GPIO folder and list the contents
  - root@sahaPi:/home/dsaha# cd /sys/class/gpio/
  - root@sahaPi:/sys/class/gpio# ls
  - export gpiochip0 gpiochip128 unexport
- Export gpio 4
  - root@sahaPi:/sys/class/gpio# echo 4 > export
  - root@sahaPi:/sys/class/gpio# ls
  - export gpio4 gpiochip0 gpiochip128 unexport

Control GPIO with Linux

- Go to the gpio4 folder and list contents
  - root@sahaPi:/sys/class/gpio# cd gpio4/
  - root@sahaPi:/sys/class/gpio/gpio4# ls
  - active_low device direction edge power subsystem uevent value
- Set direction (in or out) of pin
  - root@sahaPi:/sys/class/gpio/gpio4# echo out > direction
- Set value to be 1 to turn on the LED
  - root@sahaPi:/sys/class/gpio/gpio4# echo 1 > value

Control GPIO with Linux

- Set value to be 0 to turn off the LED
  - root@sahaPi:/sys/class/gpio/gpio4# echo 0 > value
- Check the status (direction and value) of the pin
  - root@sahaPi:/sys/class/gpio/gpio4# cat direction
  - out
  - root@sahaPi:/sys/class/gpio/gpio4# cat value
  - 0

Control GPIO with Linux

- Ready to give up the control? Get out of gpio4 folder and list contents, which shows gpio4 folder
  - root@sahaPi:/sys/class/gpio/gpio4# cd ../
  - root@sahaPi:/sys/class/gpio# ls
  - export gpio4 gpiochip0 gpiochip128 unexport
- Unexport gpio 4 and list contents showing removal of gpio4 folder
  - root@sahaPi:/sys/class/gpio# echo 4 > unexport
  - root@sahaPi:/sys/class/gpio# ls
  - export gpiochip0 gpiochip128 unexport

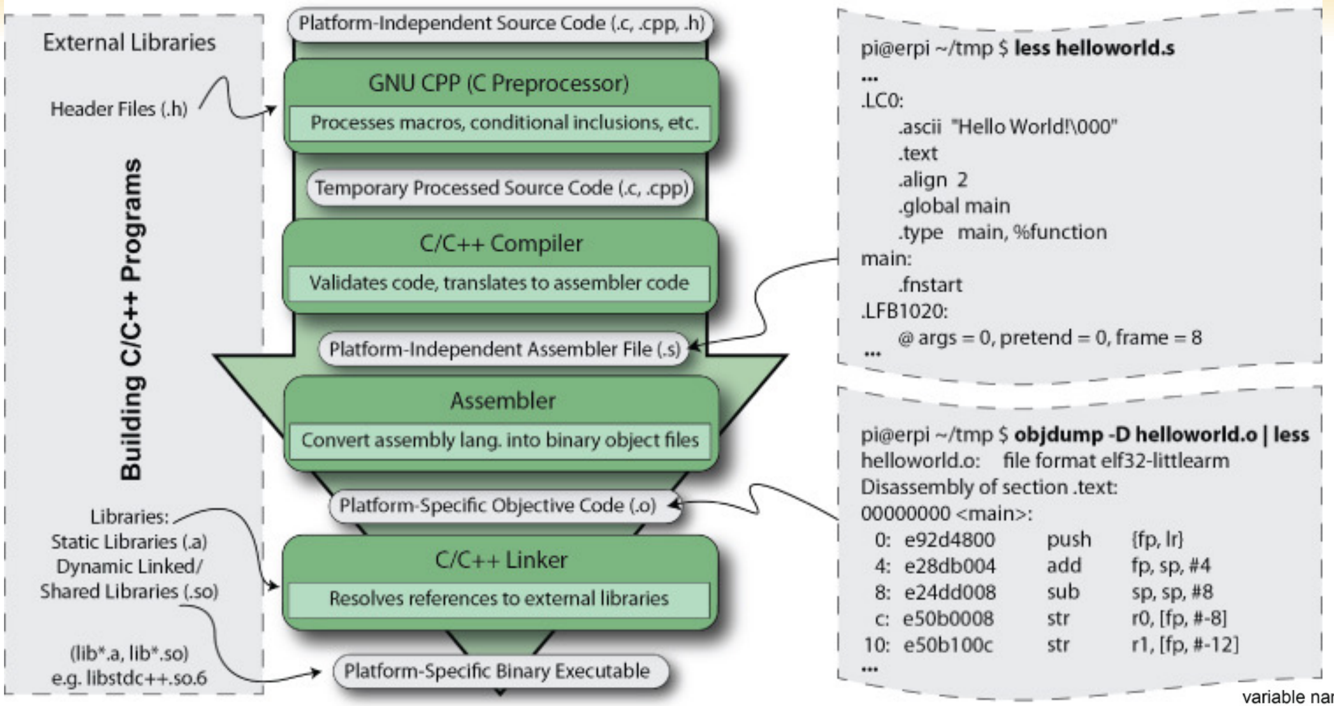
Program

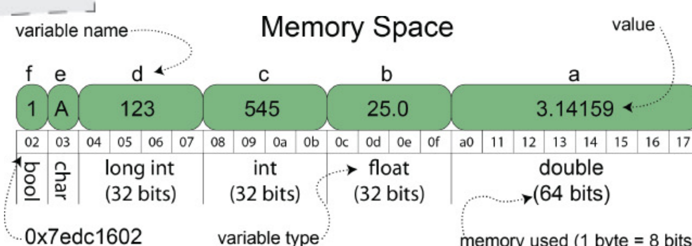
- Bash Script
  - exploringrpi/chp05/bashLED/bashLED
- Python Code
  - exploringrpi/chp05/pythonLED/python2LED.py
- C code
  - exploringrpi/chp05/makeLED/makeLED.c

C/C++

ADVANTAGES	DISADVANTAGES
You can build code directly on the RPi or you can **cross-compile code** using professional toolchains. Runtime environments do not need to be installed.	Compiled **code is not portable**. Code compiled for your x86 desktop will not run on the RPi ARM processor.
C++ has full **support** for procedural programming, OOP, and support for generics through the use of STL (Standard Template Library).	Many consider the languages to be complex to master. There is a tendency to need to know everything before you can do anything.
It gives the best **computational performance**, especially *if optimized*. However, optimization can be difficult and can reduce the portability of your code.	The use of pointers and the low-level control available makes code prone to memory leaks. With careful coding these can be avoided and can lead to efficiencies over dynamic memory management schemes.
Can be used for high-performance user-interface application development on the RPi using third-party libraries. Libraries such as Qt and Boost provide extensive additional libraries for components, networking, etc.	By default, C and C++ do not support graphical user interfaces, network sockets, etc. Third-party libraries are required.
Offers **low-level access** to glibc for integrating with the Linux system. Programs can be setuid to root.	Not suitable for scripting (there is a C shell, csh, that does have syntax like C). You can integrate with Lua. Not ideal for web development either.
The **Linux kernel is written in C** and having knowledge of C/C++ can help if you ever have to write device drivers or contribute to Linux kernel development.	C++ attempts to span from low-level to high-level programming tasks, but it can be difficult to write very scalable enterprise or web applications.
The C/C++ languages are ISO standards, not owned by a single company.	

Building C/C++ Applications





Bash and Python Script

```bash
LED_GPIO=4   # Use a variable -- easy to change GPIO number

# An example Bash functions
function setLED
{ # $1 is the first argument that is passed to this function
  echo $1 >> "/sys/class/gpio/gpio$LED_GPIO/value"
}

# Start of the program -- start reading from here
if [ $# -ne 1 ]; then   # if there is not exactly one argument
  echo "No command was passed. Usage is: bashLED command,"
  echo "where command is one of: setup, on, off, status and close"
  echo -e " e.g., bashLED setup, followed by bashLED on"
  exit 2     # error that indicates an invalid number of arguments
fi
echo "The LED command that was passed is: $1"
if [ "$1" == "setup" ]; then
  echo "Exporting GPIO number $1"
  echo $LED_GPIO >> "/sys/class/gpio/export"
  sleep 1    # to ensure gpio has been exported before next step
  echo "out" >> "/sys/class/gpio/gpio$LED_GPIO/direction"
elif [ "$1" == "on" ]; then
  echo "Turning the LED on"
  setLED 1   # 1 is received as $1 in the setLED function
elif [ "$1" == "off" ]; then
  echo "Turning the LED off"
  setLED 0   # 0 is received as $1 in the setLED function
elif [ "$1" == "status" ]; then
  state=$(cat "/sys/class/gpio/gpio$LED_GPIO/value")
  echo "The LED state is: $state"
elif [ "$1" == "close" ]; then
  echo "Unexporting GPIO number $LED_GPIO"
  echo $LED_GPIO >> "/sys/class/gpio/unexport"
fi
```

```python
import sys
from time import sleep
LED4_PATH = "/sys/class/gpio/gpio4/"
SYSFS_DIR = "/sys/class/gpio/"
LED_NUMBER = "4"

def writeLED ( filename, value, path=LED4_PATH ):
   "This function writes the value passed to the file in the path"
   fo = open( path + filename,"w")
   fo.write(value)
   fo.close()
   return

print "Starting the GPIO LED4 Python script"
if len(sys.argv)!=2:
   print "There is an incorrect number of arguments"
   print "  usage is:  pythonLED.py command"
   print "  where command is one of setup, on, off, status, or close"
   sys.exit(2)
if sys.argv[1]=="on":
   print "Turning the LED on"
   writeLED (filename="value", value="1")
elif sys.argv[1]=="off":
   print "Turning the LED off"
   writeLED (filename="value", value="0")
elif sys.argv[1]=="setup":
   print "Setting up the LED GPIO"
   writeLED (filename="export", value=LED_NUMBER, path=SYSFS_DIR)
   sleep(0.1);
   writeLED (filename="direction", value="out")
elif sys.argv[1]=="close":
   print "Closing down the LED GPIO"
   writeLED (filename="unexport", value=LED_NUMBER, path=SYSFS_DIR)
elif sys.argv[1]=="status":
   print "Getting the LED state value"
   fo = open( LED4_PATH + "value", "r")
   print fo.read()
   fo.close()
else:
   print "Invalid Command!"
print "End of Python script"
```

C Program

```c
#define GPIO_NUMBER "4"
#define GPIO4_PATH "/sys/class/gpio/gpio4/"
#define GPIO_SYSFS "/sys/class/gpio/"

void writeGPIO(char filename[], char value[]){
   FILE* fp;                              // create a file pointer fp
   fp = fopen(filename, "w+");            // open file for writing
   fprintf(fp, "%s", value);              // send the value to the file
   fclose(fp);                            // close the file using fp
}

int main(int argc, char* argv[]){
   if(argc!=2){                           // program name is argument 1
      printf("Usage is makeLEDC and one of:\n");
      printf("   setup, on, off, status, or close\n");
      printf(" e.g. makeLEDC on\n");
      return 2;                           // invalid number of arguments
   }
   printf("Starting the makeLED program\n");
   if(strcmp(argv[1],"setup")==0){
      printf("Setting up the LED on the GPIO\n");
      writeGPIO(GPIO_SYSFS "export", GPIO_NUMBER);
      usleep(100000);                     // sleep for 100ms
      writeGPIO(GPIO4_PATH "direction", "out");
   }
   else if(strcmp(argv[1],"close")==0){
      printf("Closing the LED on the GPIO\n");
      writeGPIO(GPIO_SYSFS "unexport", GPIO_NUMBER);
   }
```

```c
   else if(strcmp(argv[1],"on")==0){
      printf("Turning the LED on\n");
      writeGPIO(GPIO4_PATH "value", "1");
   }
   else if (strcmp(argv[1],"off")==0){
      printf("Turning the LED off\n");
      writeGPIO(GPIO4_PATH "value", "0");
   }
   else if (strcmp(argv[1],"status")==0){
      FILE* fp;        // see writeGPIO function above for description
      char line[80], fullFilename[100];
      sprintf(fullFilename, GPIO4_PATH "/value");
      fp = fopen(fullFilename, "rt");           // reading text this time
      while (fgets(line, 80, fp) != NULL){
         printf("The state of the LED is %s", line);
      }
      fclose(fp);
   }
   else{
      printf("Invalid command!\n");
   }
   printf("Finished the makeLED Program\n");
   return 0;
}
```

Use Rpi Library

- https://sourceforge.net/projects/raspberry-gpio-python/
- Note: Current release does not support SPI, I2C, 1-wire or serial functionality on the RPi yet

```
import RPi.GPIO as GPIO
from time import sleep

ledPin = 4                    # GPIO Pin Number, where LED is connected

GPIO.setmode(GPIO.BCM)        # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT)  # LED pin set as output

GPIO.output(ledPin, GPIO.HIGH) # Turn the LED on
sleep(1)                       # Sleep for 1 sec
GPIO.output(ledPin, GPIO.LOW)  # Turn the LED off
```

Use gpiozero Library

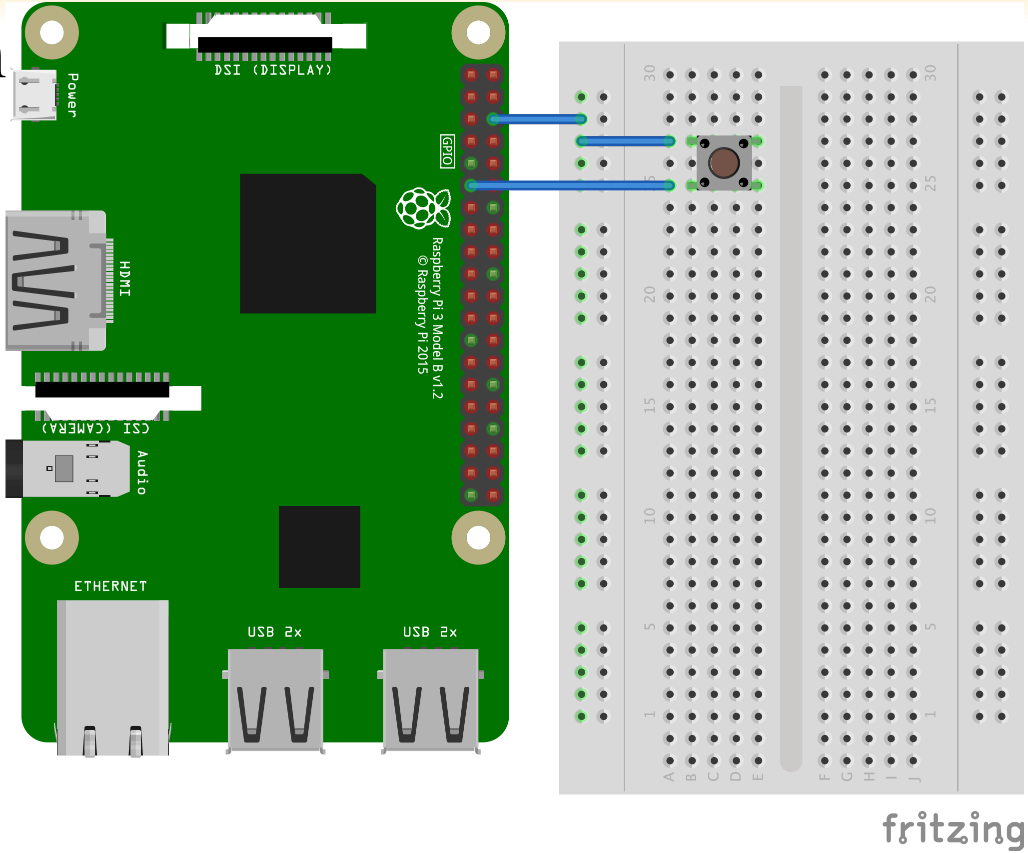
- https://gpiozero.readthedocs.io/en/stable/

```
from gpiozero import LED
from time import sleep

led = LED(4)      # GPIO Pin Number
led.on()          # Turn on LED
sleep(1)          # Sleep for 1 sec
led.off()         # Turn off LED
```

GPIO as Input

- Push-button Switch



Reading GPIO

```
import RPi.GPIO as GPIO
import time

buttonPin=17     # GPIO Pin Number where Button Switch is connected

GPIO.setmode(GPIO.BCM)            # Broadcom pin-numbering scheme
GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Button pin set as input

while True:                               # Monitor continuously
    input_state = GPIO.input(buttonPin)   # Get the input state
    if input_state == False:              # Check status
        print('Button Pressed')           # Print
        time.sleep(0.2)                   # Sleep before checking again
```

```
from gpiozero import Button
import time

button = Button(17) # GPIO Pin Number where Button Switch is connected

while True:                               # Monitor continuously
    if button.is_pressed:                 # Check Status
        print("Button Pressed")           # Print
        time.sleep(0.2)                   # Sleep before checking again
```

Block Diagram

- Pg 89 of BCM2837
- Alternate functions
- Dedicated interrupt
- Pull-up/pull-down state

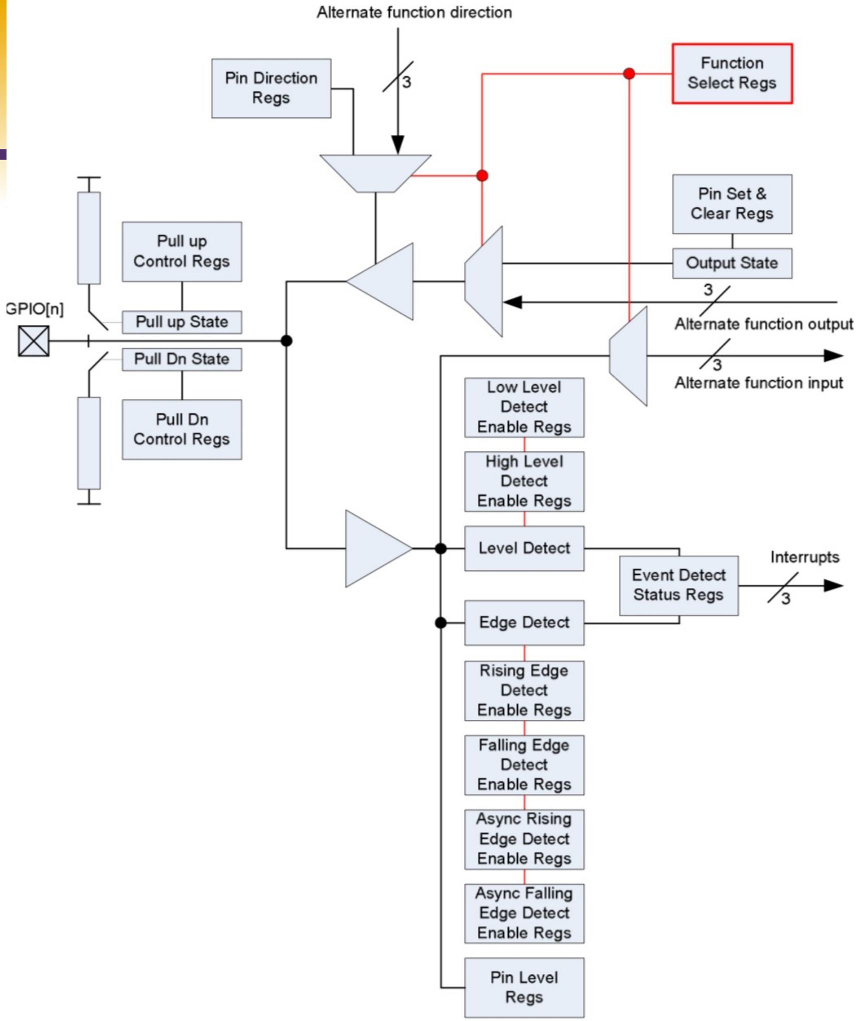
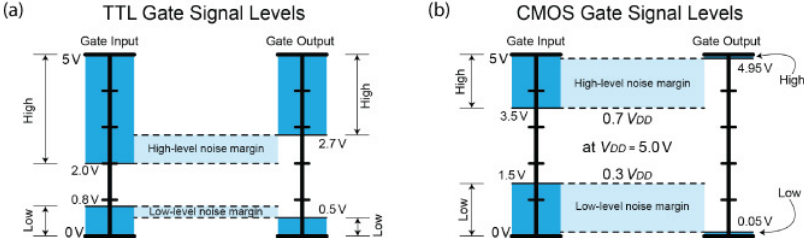


Figure 6-1 GPIO Block Diagram

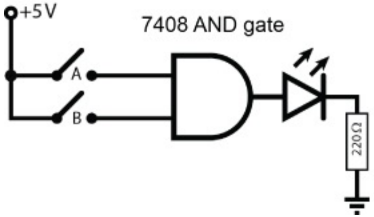
TTL & CMOS

- TTL (Transistor-transistor logic) & CMOS (complementary metal-oxide-semiconductor)



Margin for 5V

- Noise Margin: absolute difference between the output voltage levels and the input voltage levels.
- Floating Input: leave unused logic gate inputs "floating," or disconnected
  - TTL: Float high, should tie to ground
  - CMOS
    - inputs are sensitive to the high voltages
    - from static electricity & electrical noise
    - should never be left floating

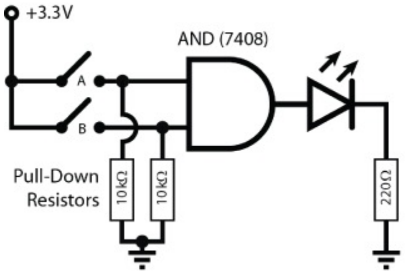


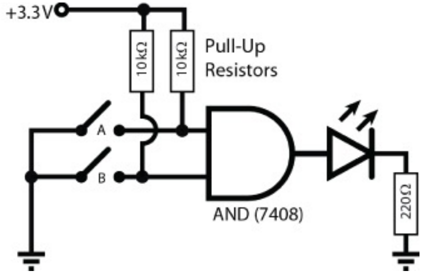
7408 AND gate

Switch A	Switch B	Required (A.B)	74LS08 TTL Output	74HC08 CMOS Output
Closed	Closed	On	On	On
Closed	Open	Off	**On**	**~Off**
Open	Closed	Off	**On**	**~Off**
Open	Open	Off	**On**	**~Off**

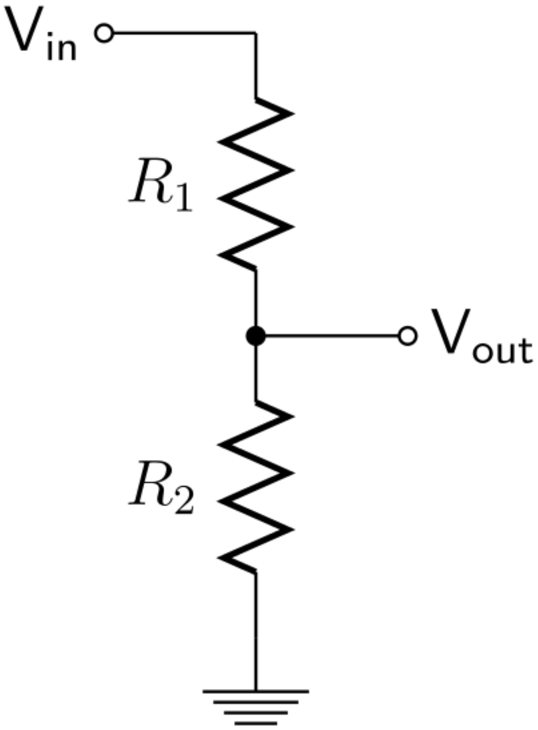
Pull-down and Pull-up Resistors

- Used to ensure that the switches do not create floating inputs
- Pull-down resistors:
  - used to guarantee that the inputs to the gate are **low** when the switches are open
- Pull-up resistors:
  - used to guarantee that the inputs are **high** when the switches are open.
- The RPi has weak internal pull-up and pull-down resistors that can be used for this purpose.





Voltage Divider Circuit



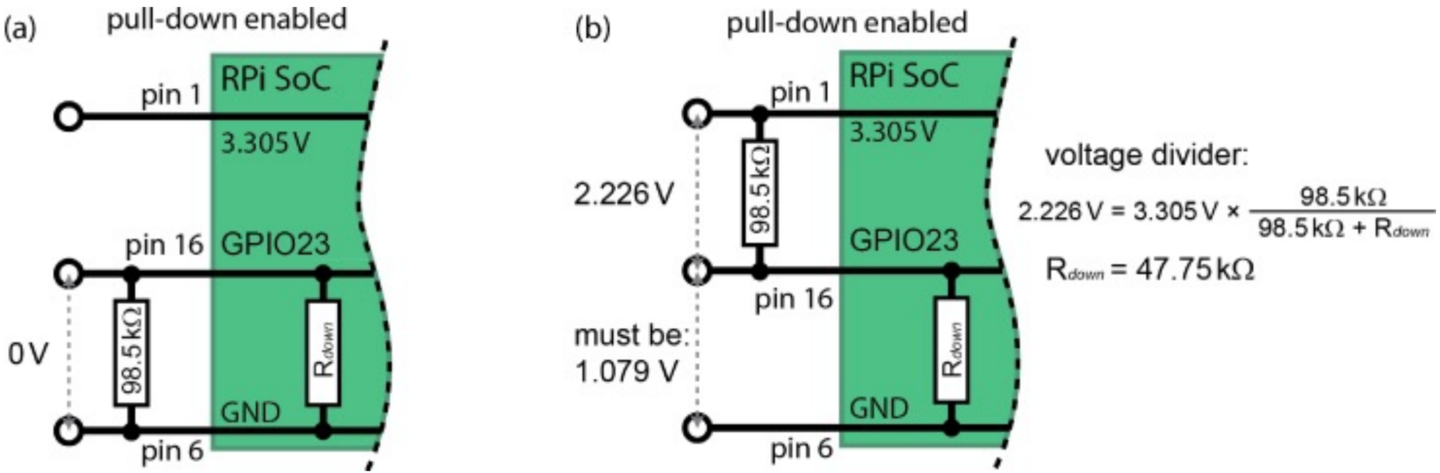
$$V_{in} = I(R_1 + R_2)$$

$$V_{out} = IR_2$$

$$V_{out} = \frac{R_2}{R_1 + R_2} . V_{in}$$

Calculate Internal Resistor Value

- Voltage Divider



- Pin 3 (GPIO2) & Pin 5 (GPIO3) have two permanent onboard 1.8 kΩ "strong" pull-up resistors attached on the PCB (R23 and R24)

Internal pull-up/pull-down Resistors

- Can be configured using memory based GPIO control
- **cat /proc/iomem**
- 00000000-3b3fffff : System RAM
- ….
- 3f200000-3f2000b3 : /soc/gpio@7e200000
- ….

Address Mapped

/dev/mem

- *\/dev/mem* is a character device file that is an image of the main memory of the computer.
- Byte addresses in */dev/mem* are interpreted as physical memory addresses.
- References to nonexistent locations cause errors to be returned.

Use /dev/mem directly

- **wget** http://www.lartmaker.nl/lartware/port/devmem2.c
- **gcc devmem2.c -o devmem2**
- **./devmem2**

Usage: ./devmem2 { address } [type [data]]

address : memory address to act upon

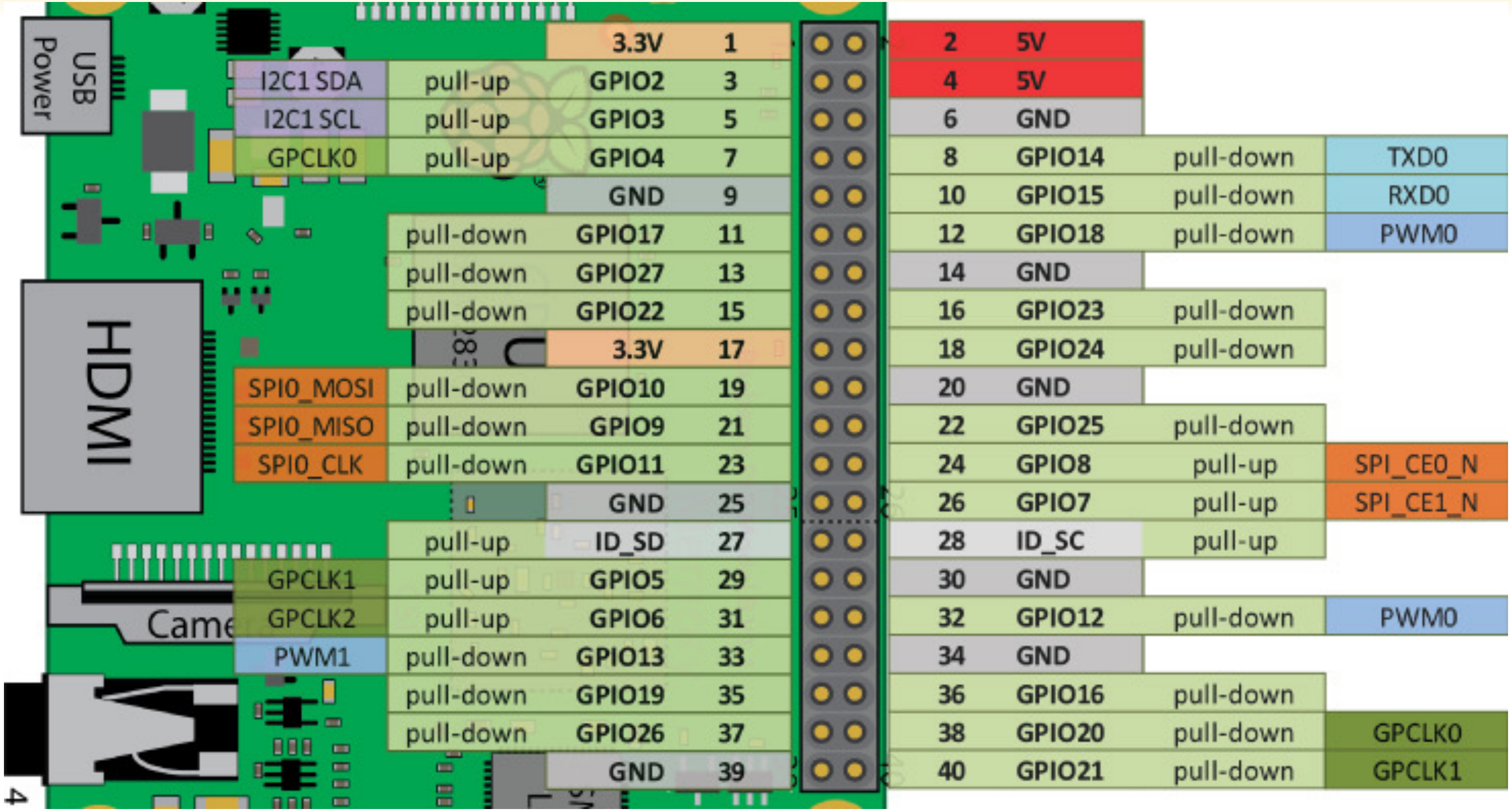
type : access operation type : [b]yte, [h]alfword, [w]ord

data : data to be written

GPIO Pull-up/down Register Control

- The **GPIO Pull-up/down Register** controls the actuation of the internal pull-up/down control line to ALL the GPIO pins. This register must be used in conjunction with the **2 GPPUDCLKn registers**.
- Note that it is **not possible to read back** the current Pull-up/down settings and so it is the users' responsibility to 'remember' which pull-up/downs are active. The reason for this is that GPIO pull-ups are maintained even in power-down mode when the core is off, when all register contents is lost.

Default Configuration



Alternate Functions of GPIO Pins

The RPi GPIO header (note 1)

Pi A/B Rev 2 (P1), B+ (J8), Pi 2/3(J8)

ALT5	ALT4	ALT3	ALT2	ALT1	ALT0	WPi	Pull	Mode	Pin Numbers	Pin Numbers	Mode	Pull	WPi	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
50mA maximum on 3.3V supply								3.3V	1	2	5V	max current draw ~300mA (cover when not in use)							
(note 2)			reserved	SA3	I2C1 SDA	8	up	GPIO2	3	4	5V	max current draw ~300mA (cover when not in use)							
(note 2)			reserved	SA2	I2C1 SCL	9	up	GPIO3	5	6	GND								
ARM_TDI			reserved	SA1	GPCLK0	7	up	GPIO4	7	8	GPIO14	down	15	TXD0	SD6				TXD1
								GND	9	10	GPIO15	down	16	RXD0	SD7				RXD1
RTS1	SPI1_CE1_N	RTS0	reserved	SD9	reserved	0	down	GPIO17	11	12	GPIO18	down	1	PCM_CLK	SD10		BSCSL_SDA/MOSI	SPI1_CE0_N	PWM0
	ARM_TMS	SD1_DAT3	reserved	reserved	reserved	2	down	GPIO27	13	14	GND								
	ARM_TRST	SD1_CLK	reserved	SD14	reserved	3	down	GPIO22	15	16	GPIO23	down	4	reserved	SD15	reserved	SD1_CMD	ARM_RTCK	
50mA maximum on 3.3V supply								3.3V	17	18	GPIO24	down	5	reserved	SD16	reserved	SD1_DAT0	ARM_TDO	
			reserved	SD2	SPI0_MOSI	12	down	GPIO10	19	20	GND								
			reserved	SD1	SPI0_MISO	13	down	GPIO9	21	22	GPIO25	down	6	reserved	SD17	reserved	SD1_DAT1	ARM_TCK	
			reserved	SD3	SPI0_CLK	14	down	GPIO11	23	24	GPIO8	up	10	SPI_CE0_N	SD0	reserved			
								GND	25	26	GPIO7	up	11	SPI_CE1_N	SWE_N / SRW_N	reserved			
Do not use (GPIO0) -- see note 3			reserved	SA5	SDA0	30	up	ID_SD	27	28	ID_SC	up	31	SCL0	SA4	reserved	Do not use (GPIO1) -- see note 3		
ARM_TDO			reserved	SA0	GPCLK1	21	up	GPIO5	29	30	GND								
ARM_RTCK			reserved	SOE_N/SE	GPCLK2	22	up	GPIO6	31	32	GPIO12	down	26	PWM0	SD4	reserved			ARM_TMS
ARM_TCK			reserved	SD5	PWM1	23	down	GPIO13	33	34	GND								
			reserved	SD11	PCM_FS	24	down	GPIO19	35	36	GPIO16	down	27	reserved	SD8	reserved	CTS0	SPI1_CE2_N	CTS1
	ARM_TDI	SD1_DAT2	reserved	reserved	reserved	25	down	GPIO26	37	38	GPIO20	down	28	PCM_DIN	SD12	reserved	BSCSL / MISO	SPI1_MOSI	GPCLK0
								GND	39	40	GPIO21	down	29	PCM_DOUT	SD13	reserved	BSCSL / CE_N	SPI1_SCLK	GPCLK1

RPi A, B, A+, B+, 2/3 (pins 1–26)

RPi A+, B+, 2/3 Only (pins 27–40)

Type	Linux DT	Description
GPIO	sysfs	general purpose input/output
SPI	spi	serial peripheral interface
I2C	i2c0/i2c1	I^2C Bus
UART	uart0	UART
PWM	pwm	Pulse Width Modulation
GPCLK	gp_clk	General purpose clock (GPCLK1 is reserved)
PCM	pcm	PCM audio
SA	smi	Secondary Memory Interface
ARM_	arm_jtag	ARM JTAG debugger

Note 1: The data in this table was created from the www.eLinux.org web pages, system information, and datasheets where available.

Note 2: On early models of the RPi, Pin 3 is GPIO0 and Pin 5 is GPIO1. Also, these pins have permanent on-board 1.8 KΩ pull-up resistors attached (for the I^2C bus).

Note 3: ID_SD and ID_SC pins are reserved for the ID EEPROM (for different HATs). This is an I^2C interface that is probed at boot time in order to detect attached boards. This allows Linux to load the correct drivers for a HAT. See Chapter 8.

BCM 2837 Manual

- Table 6-1

0x 7E20 0094	GPPUD	GPIO Pin Pull-up/down Enable	32	R/W
0x 7E20 0098	GPPUDCLK0	GPIO Pin Pull-up/down Enable Clock 0	32	R/W
0x 7E20 009C	GPPUDCLK1	GPIO Pin Pull-up/down Enable Clock 1	32	R/W

- Table 6-28

31-2	---	Unused	R	0
1-0	PUD	PUD - GPIO Pin Pull-up/down 00 = Off – disable pull-up/down 01 = Enable Pull Down control 10 = Enable Pull Up control 11 = Reserved *Use in conjunction with GPPUDCLK0/1/2	R/W	0

BCM 2837 Manual

Bit(s)	Field Name	Description	Type	Reset
(31-0)	PUDCLKn (n=0..31)	0 = No Effect 1 = Assert Clock on line *(n)* *Must be used in conjunction with GPPUD	R/W	0

Table 6-29 – GPIO Pull-up/down Clock Register 0

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	PUDCLKn (n=32..53)	0 = No Effect 1 = Assert Clock on line *(n)* *Must be used in conjunction with GPPUD	R/W	0

Table 6-30 – GPIO Pull-up/down Clock Register 1

Control Pull-up/down (from BCM2837)

1. Write to GPPUD to set the required control signal (i.e. Pull-up or Pull-Down or neither to remove the current Pull-up/down)
2. Wait 150 cycles – this provides the required set-up time for the control signal
3. Write to GPPUDCLK0/1 to clock the control signal into the GPIO pads you wish to modify – NOTE only the pads which receive a clock will be modified, all others will retain their previous state.
4. Wait 150 cycles – this provides the required hold time for the control signal
5. Write to GPPUD to remove the control signal
6. Write to GPPUDCLK0/1 to remove the clock

Pull Down Resistor is enabled

- Set bit 4 on the GPPUDCLK0 register, clear the GPPUD register, and then remove the clock control signal from GPPUDCLK0
  - GPIO4 is bit 4, which is 10000_2 ($0x10_{16}$)
- Get the Value in GPIO 4
  - sudo su
  - cd /sys/class/gpio/
  - echo 4 > export
  - cd gpio4
  - cat value

Pull Down Resistor is enabled

- GPPUD Enable Pull-down
  - sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x01
- GPPUDCLK0 enable GPIO 4
  - sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
- GPPUD Disable Pull-down
  - sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
- GPPUDCLK0 disable Clk signal
  - sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
- cat value
  - 0

Pull up Configuration

- GPPUD Enable Pull-up
  - sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x02
- GPPUDCLK0 enable GPIO 4
  - sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
- GPPUD Disable Pull-up
  - sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
- GPPUDCLK0 disable Clk signal
  - sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
- cat value
  - 1

GPIO Function Select

0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W

- The function select registers are used to define the operation of the general-purpose I/O pins. Each of the 54 GPIO pins has at least two alternative functions as defined in section 16.2. The FSEL{n} field determines the functionality of the nth GPIO pin. All unused alternative function lines are tied to ground and will output a “0” if selected. All pins reset to normal GPIO input operation.

GPIO Function Select Register

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL19	FSEL19 - Function Select 19 000 = GPIO Pin 19 is an input 001 = GPIO Pin 19 is an output 100 = GPIO Pin 19 takes alternate function 0 101 = GPIO Pin 19 takes alternate function 1 110 = GPIO Pin 19 takes alternate function 2 111 = GPIO Pin 19 takes alternate function 3 011 = GPIO Pin 19 takes alternate function 4 010 = GPIO Pin 19 takes alternate function 5	R/W	0
26-24	FSEL18	FSEL18 - Function Select 18	R/W	0
23-21	FSEL17	FSEL17 - Function Select 17	R/W	0
20-18	FSEL16	FSEL16 - Function Select 16	R/W	0
17-15	FSEL15	FSEL15 - Function Select 15	R/W	0
14-12	FSEL14	FSEL14 - Function Select 14	R/W	0
11-9	FSEL13	FSEL13 - Function Select 13	R/W	0
8-6	FSEL12	FSEL12 - Function Select 12	R/W	0
5-3	FSEL11	FSEL11 - Function Select 11	R/W	0
2-0	FSEL10	FSEL10 - Function Select 10	R/W	0

Table 6-3 – GPIO Alternate function select register 1

GPIO Pin Output Set Registers

Address

0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W

Values

Bit(s)	Field Name	Description	Type	Reset
31-0	SETn (n=0..31)	0 = No effect 1 = Set GPIO pin *n*	R/W	0

Table 6-8 – GPIO Output Set Register 0

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	SETn (n=32..53)	0 = No effect 1 = Set GPIO pin *n.*	R/W	0

Table 6-9 – GPIO Output Set Register 1

GPIO Pin Output Clear

0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W

Bit(s)	Field Name	Description	Type	Reset
31-0	CLRn (n=0..31)	0 = No effect 1 = Clear GPIO pin *n*	R/W	0

Table 6-10 – GPIO Output Clear Register 0

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	CLRn (n=32..53)	0 = No effect 1 = Clear GPIO pin *n*	R/W	0

Table 6-11 – GPIO Output Clear Register 1

GPIO Pin Level

- The pin level registers return the actual value of the pin. The LEV{n} field gives the value of the respective GPIO pin.

0x 7E20 0034	GPLEV0	GPIO Pin Level 0	32	R
0x 7E20 0038	GPLEV1	GPIO Pin Level 1	32	R

Bit(s)	Field Name	Description	Type	Reset
31-0	LEVn (n=0..31)	0 = GPIO pin *n* is low 1 = GPIO pin *n* is high	R/W	0

Table 6-12 – GPIO Level Register 0

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	LEVn (n=32..53)	0 = GPIO pin *n* is low 1 = GPIO pin *n* is high	R/W	0

Table 6-13 – GPIO Level Register 1

WiringPi

```
dsaha@sahaPi:~/wiringPi $ gpio readall
 +-----+-----+---------+------+---+---Pi 3+--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3+--+---+------+---------+-----+-----+
```

The gpio Command (WiringPi)

Command	Example	Description
gpio read <pin>	`gpio read 2`	Read a binary value from a WPi numbered pin. Use `-g` to use GPIO numbers. Example reads button state.
gpio write <pin> <value>	`gpio write 0 1`	Set a binary value on a WPi numbered pin. Example sets the LED on. <value> is either `1` or `0`.
gpio mode <pin> <mode>	`gpio mode 1 pwm`	Example sets the h/w PWM outputs on (WPi pin 1, GPIO 18). <mode> is one of `in`, `out`, `pwm`, `up`, `down`, or `tri`.
gpio pwm <pin> <value>	`gpio pwm 1 256`	Set a PWM value on the PWM output pin.
gpio clock <pin> <freq>	`gpio mode 7 clock` `gpio clock 7 2400000`	Sets up a clock signal (i.e., 50% duty cycle) on a pin with general purpose clock capabilities. The signal is derived by dividing the 19.2 MHz clock, so integer divisors of this frequency are optimum.
gpio readall	`gpio readall`	Reads all of the pins and prints a chart of their numbers, modes, and values.
gpio unexportall	`gpio unexportall`	Unexport all GPIO sysfs entries.
gpio export <gpio> <mode>	`gpio export 4 input`	Exports a pin using the GPIO numbering. <mode> is either `in/input` or `out/output`.
gpio exports	`gpio exports`	Lists all sysfs exported pins.
gpio unexport <gpio>	`gpio unexport 4`	Unexport a pin using the GPIO numbering.
gpio edge <pin> <mode>	`gpio edge 4 rising`	Enables the GPIO pin for edge interrupt triggering. <mode> is one of `rising`, `falling`, `both`, or `none`.
gpio wfi <pin> <mode>	`gpio wfi 2 both`	Wait on a state change. <mode> is one of `rising`, `falling`, or `both`.
gpio pwm-bal	`gpio pwm-bal`	Set the PWM mode to be balanced.
gpio pwm-ms	`gpio pwm-ms`	Set the PWM mode to be mark-space.
gpio pwmr <range>	`gpio pwmr 512`	Set the PWM range. <range> is not limited - typically less than 4,095.
gpio pwmc <divider>	`gpio pwmc 10`	Set the PWM clock divider. PWM frequency = 19.2 MHz / (range × divider).

wiringPi

- ➢ Functions

Return	Function Call	Description
Setup		
int	`wiringPiSetup(void)`	Initializes wiringPi. Must be used with root privileges. Returns 0 if successful.
int	`wiringPiSetupGpio(void)`	Same as above. Uses GPIO rather than WPi numbers. Must use root privileges.
int	`wiringPiSetupSys(void)` (use one)	Uses sysfs. Root not required if udev rules in place (see end of chapter). You must manually export pins. Slower, as memory-mapping does not work.
int	`wiringPiSetupPhys(void)`	Uses the physical pin numbering on the RPi.
int	`piBoardRev(void)`	Returns the board version (0=n/a, 1=A, 2=B, 3=B+, 4=compute, 5= A+, 6=RPi 2)
GPIO Control		
void	`pinMode(int pin, int mode)`	Sets the pin to be one of `INPUT`, `OUTPUT`, or `PWM_OUTPUT` (on the hardware PWM pins only). Not available if `wiringPiSetupSys()` is used.
int	`getAlt(int pin)`	Get the ALT mode for a pin.
void	`pinModeAlt(int pin, int mode)`	Set the ALT mode for a pin.
void	`digitalWrite(int pin, int value)`	Sets the pin to be one of `HIGH` (1) or `LOW` (0). The pin mode must be OUTPUT.
void	`digitalWriteByte(int value)`	Fast parallel write of 8 bits to the first eight GPIO pins.
int	`digitalRead(int pin)`	Returns the input on a pin and returns either `HIGH` (1) or `LOW` (0).
void	`pullUpDnControl(int pin, int pud)`	Sets the pull-up or pull-down resistor type to be one of `PUD_OFF` (none), `PUD_UP` (pull up), or `PUD_DOWN` (pull down). Not available in sysfs mode.
PWM and Timers		
void	`pwmWrite(int pin, int value)`	Sets the PWM output for a h/w PWM pin. Not available in sysfs mode.
void	`pwmSetMode(int mode)`	RPi PWM has two modes `PWM_MODE_BAL` (balanced) or `PWM_MODE_MS` (mark-space ratio). MS mode is most commonly used. BAL affects PWM frequency.
void	`pwmSetRange(unsigned int range)`	Sets the PWM range register. Valid values 2-4,095. Range and divisor affect frequency.
void	`pwmSetClock(int divisor)`	Sets the PWM clock divisor. PWM frequency = 19.2MHz / (divisor × range)
void	`pwmToneWrite(int pin, int freq)`	Set the frequency using the hardware PWM pin.
void	`gpioClockSet(int pin, int freq)`	Sets the frequency on a GPIO clock pin.
Interrupts		
int	`waitForInterrupt(int pin,int timeout)`	Waits for an interrupt. Timeout is set in ms where -1 is none. You must initialize the pin from outside the program, or using `system()` and the `gpio` command.
int	`wiringPiISR(int pin, int edgeType, void (*function)(void));`	Set a callback function (ISR) to be called on an interrupt event, which is one of `INT_EDGE_FALLING`, `INT_EDGE_RISING`, `INT_EDGE_BOTH`, or `INT_EDGE_SETUP`.
int	`piHiPri(int priority)`	Sets the priority of the program (0 to 99) allowing for a reduction in latency. Must be run as root. Returns 0 for success and -1 otherwise.
Helper Functions		
int	`wpiPinToGpio(int wPiPin)`	Converts WPi numbers into GPIO numbers.
int	`physPinToGpio(int physPin)`	Converts physical pin numbers to GPIO numbers.
uint32_t	`millis(void)`	Returns the number of milliseconds since a setup function was called.
uint32_t	`micros(void)`	Returns the number of microseconds since a setup function was called.
void	`delay(unsigned int t_ms)`	Delays for `t_ms` milliseconds. Delay is non-blocking and will exhibit latency.
void	`delayMicroseconds(unsigned int t_us)`	Delays for a number of microseconds.

Table information gleaned from `wiringPi.h` and `wiringPi.c`, which are distributed in the `/wiringPi/` directory of the wiringPi repository

wiringPi Blink LED

```
#include <wiringPi.h>
int main (void)
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (500) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

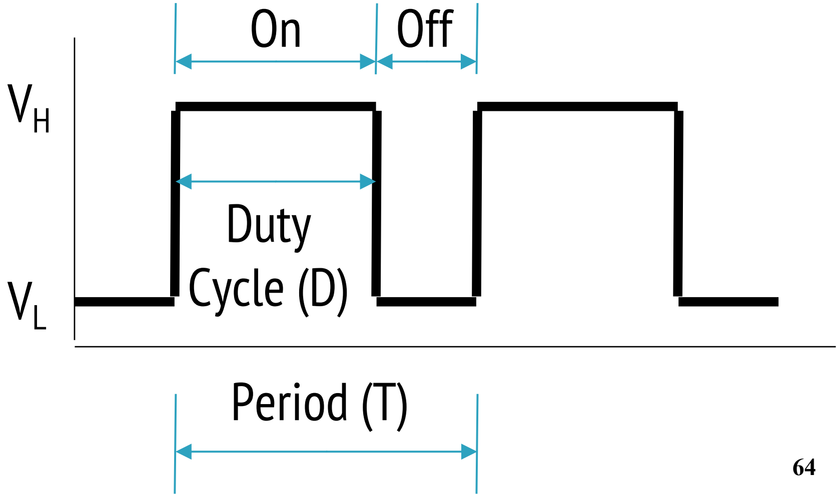
http://wiringpi.com/examples/blink/

Analog Output

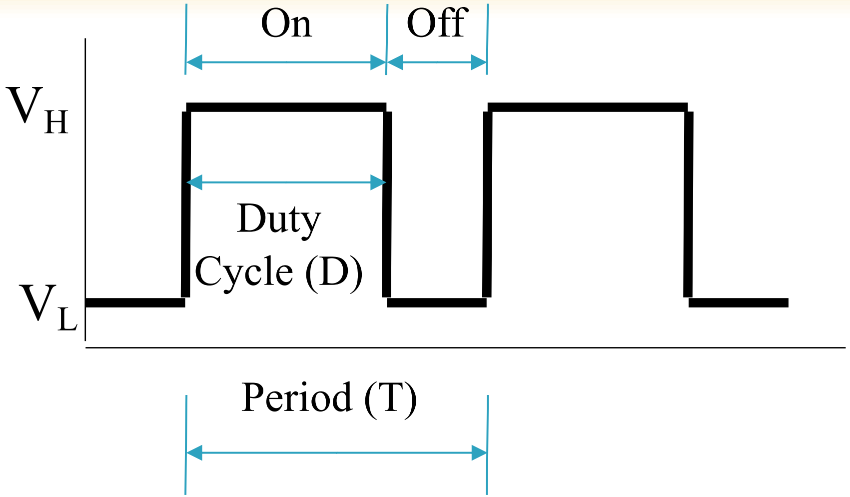
- Pulse Width Modulation (PWM)
  - Technique that conforms a signal width, generally pulses
  - The general purpose is to control power delivery
  - The on-off behavior changes the average power of signal
  - Output signal alternates between on and off within a specified period.
  - If signal toggles between on and off quicker than the load, then the load is not affected by the toggling

PWM – Duty Cycle

- A measure of the time the modulated signal is in its “high” state
- Generally recorded as the percentage of the signal period where the signal is considered on



Duty Cycle Formulation



Duty Cycle is determined by:

$$Duty\ Cycle = \frac{On\ Time}{Period} \times 100\%$$

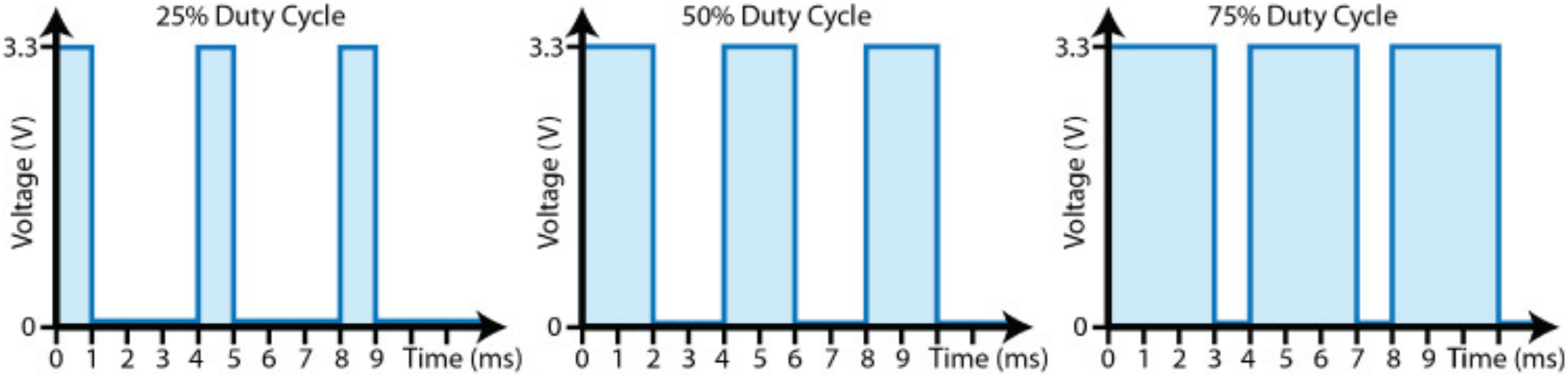
*Average value of a signal can be found as:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt$$

$$V_{avg} = D \cdot V_H + (1 - D) \cdot V_L$$

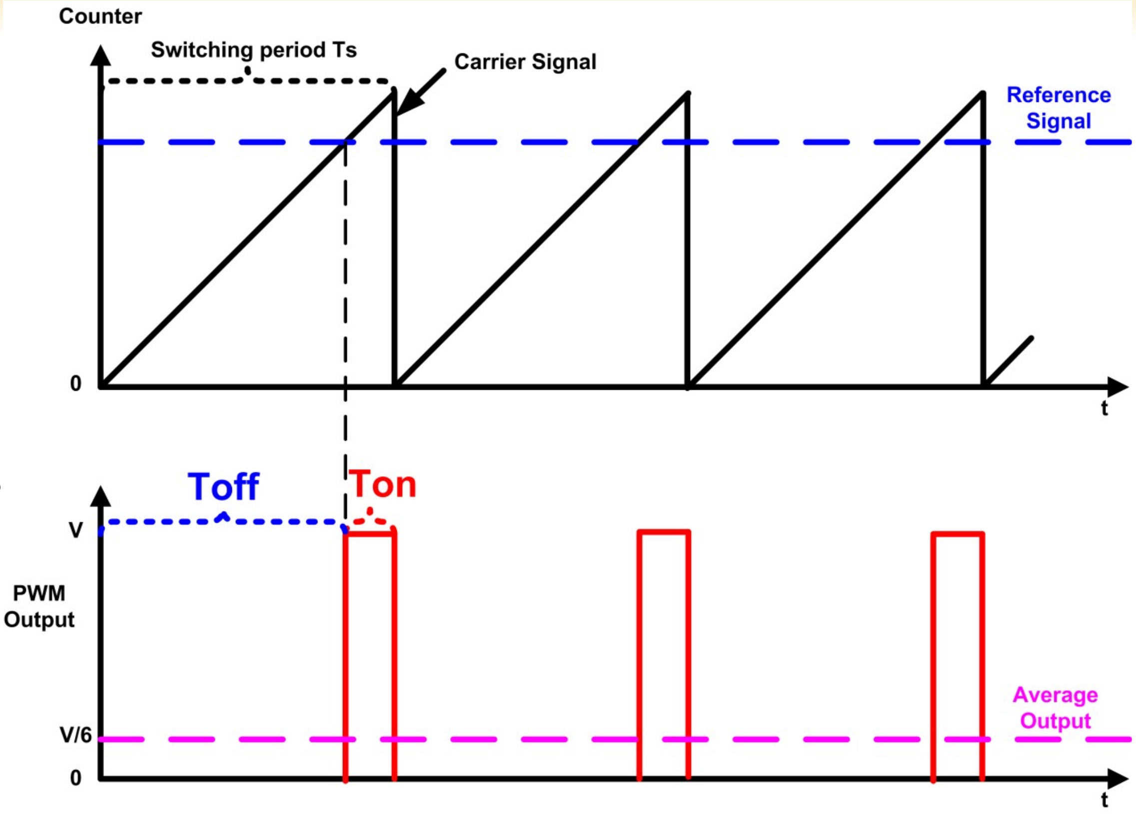
*In general analysis, V_L is taken as zero volts for simplicity.

PWM Duty Cycle



PWM Mode

- Counter counts up to the range provided
- When the counter value is higher than set value, output is high



PWM Duty Cycle Calculation

- The PWM device on the RPi is clocked at a fixed base-clock frequency of 19.2 MHz
- Integer divisor and range values are used to tailor the PWM frequency according to application requirements
- $f_{PWM} = 19.2MHz/(divisor \times range)$
- If f_{PWM} is 10KHz (0.01MHz), and range is 128,
  - $divisor = \frac{19.2MHz}{f_{PWM} \times range} = 15$

PWM0 and PWM1 Map

	PWM0	PWM1
GPIO 12	Alt Fun 0	-
GPIO 13	-	Alt Fun 0
GPIO 18	Alt Fun 5	-
GPIO 19	-	Alt Fun 5
GPIO 40	Alt Fun 0	-
GPIO 41	-	Alt Fun 0
GPIO 45	-	Alt Fun 0
GPIO 52	Alt Fun 1	-
GPIO 53	-	Alt Fun 1

9.6 Control and Status Registers

PWM Address Map			
Address Offset	**Register Name**	**Description**	**Size**
0x0	CTL	PWM Control	32
0x4	STA	PWM Status	32
0x8	DMAC	PWM DMA Configuration	32
0x10	RNG1	PWM Channel 1 Range	32
0x14	DAT1	PWM Channel 1 Data	32
0x18	FIF1	PWM FIFO Input	32
0x20	RNG2	PWM Channel 2 Range	32
0x24	DAT2	PWM Channel 2 Data	32

exploringPi/chp06/wiringPi/pwm.cpp

```
#include <iostream>
#include <wiringPi.h>
using namespace std;
#define PWM0       12                          // this is physical Pin 12
#define PWM1       33                          // only on the RPi B+/A+/2/3
int main() {                                   // must be run as root
   wiringPiSetupPhys();                        // use the physical pin numbers
   pinMode(PWM0, PWM_OUTPUT);                  // use the RPi PWM output
   pinMode(PWM1, PWM_OUTPUT);                  // only on recent RPis
   // Setting PWM frequency to be 10kHz with a full range of 128 steps
   // PWM frequency = 19.2 MHz / (divisor * range)
   // 10000 = 19200000 / (divisor * 128) => divisor = 15.0 = 15
   pwmSetMode(PWM_MODE_MS);                    // use a fixed frequency
   pwmSetRange(128);                           // range is 0-128
   pwmSetClock(15);                            // gives a precise 10kHz signal
   cout << "The PWM Output is enabled" << endl;
   pwmWrite(PWM0, 32);                         // duty cycle of 25% (32/128)
   pwmWrite(PWM1, 64);                         // duty cycle of 50% (64/128)
   return 0;                                   // PWM output stays on after exit
}
```