Embedded Linux Device Driver Programming

Norman McEntire

Welcome!

- Welcome to Embedded Linux Device Driver Programming!
- Here you will learn how to write and run Linux Device Drivers, including:
  - User-Mode Linux Device Drivers
  - Kernel-Mode Linux Device Drivers
    - Device Drivers Built Into The Kernel
    - Device Drivers Built As Kernel Module

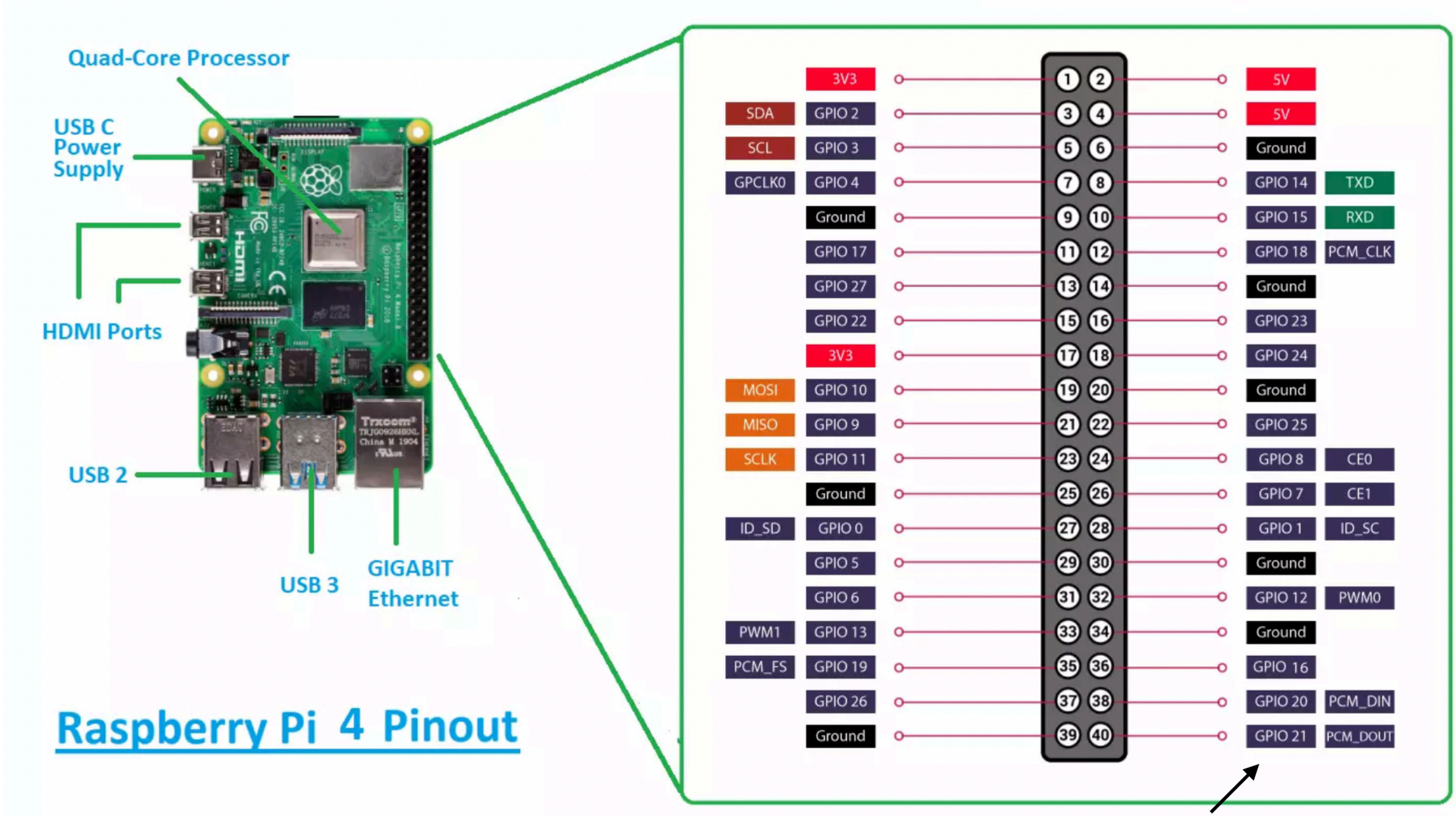
First and Foremost:

What is a Linux Device Driver?

A Linux Device Driver

- A Linux Device Driver Controls A Specific Type of Hardware Connected to a computer
  - GPIO - General Purpose I/O Hardware
  - Serial Port Hardware
  - I2C Hardware
  - SPI Hardware
  - Networking Hardware
  - And more

RPi 4 Pinout



NOTE: Broadcomm Pin Numbers (e.g. GPIO21)

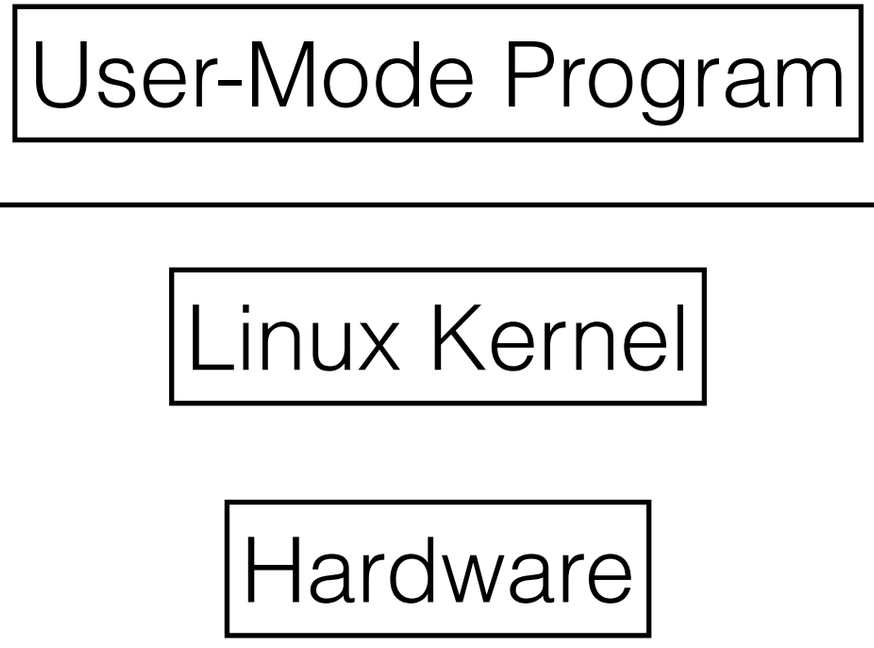
Hardware on RPi Board

- USB ports
  - USB 2
  - USB 3
- HDMI Ports
- Ethernet Port

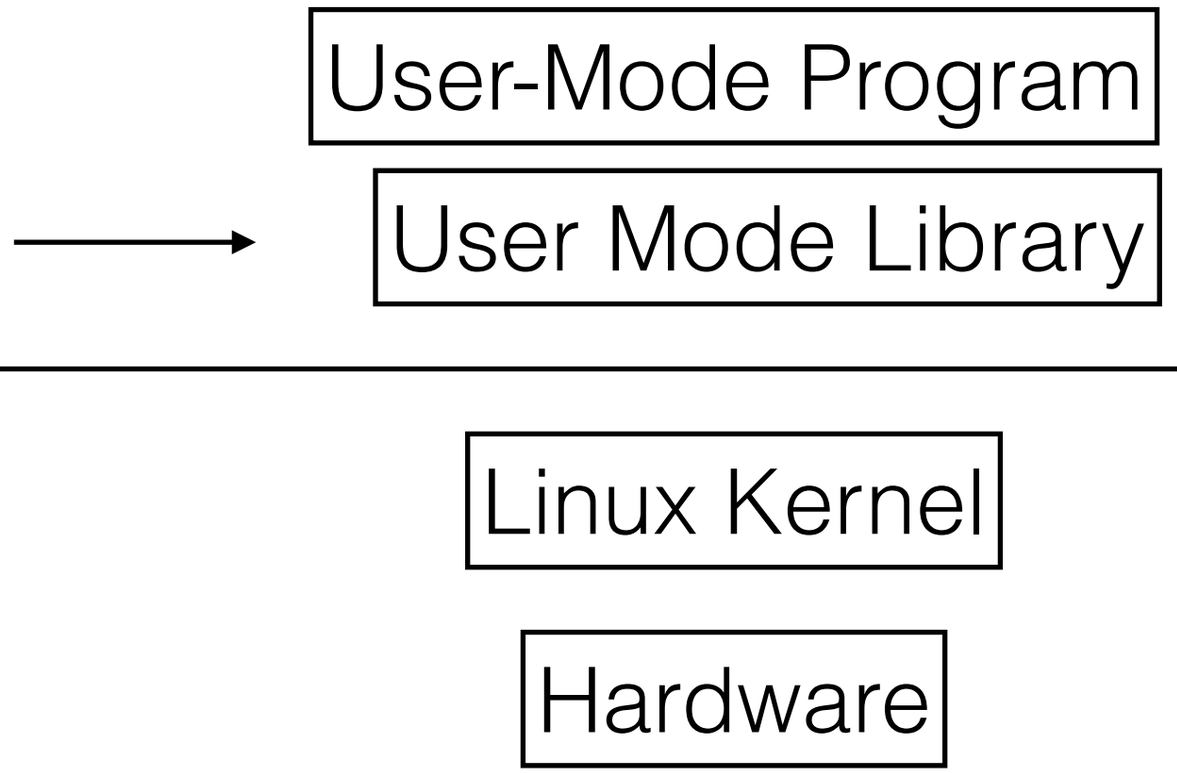
Hardware on Pinouts

- GPIO - General Purpose I/O
- Serial I/O - TxD, RxD
- I2C - Inter Integrated Circuit
  - SDA (Serial Data), SCK (Serial Clock)
- SPI - Serial Peripheral Interface
  - MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCLK (Serial Clock)

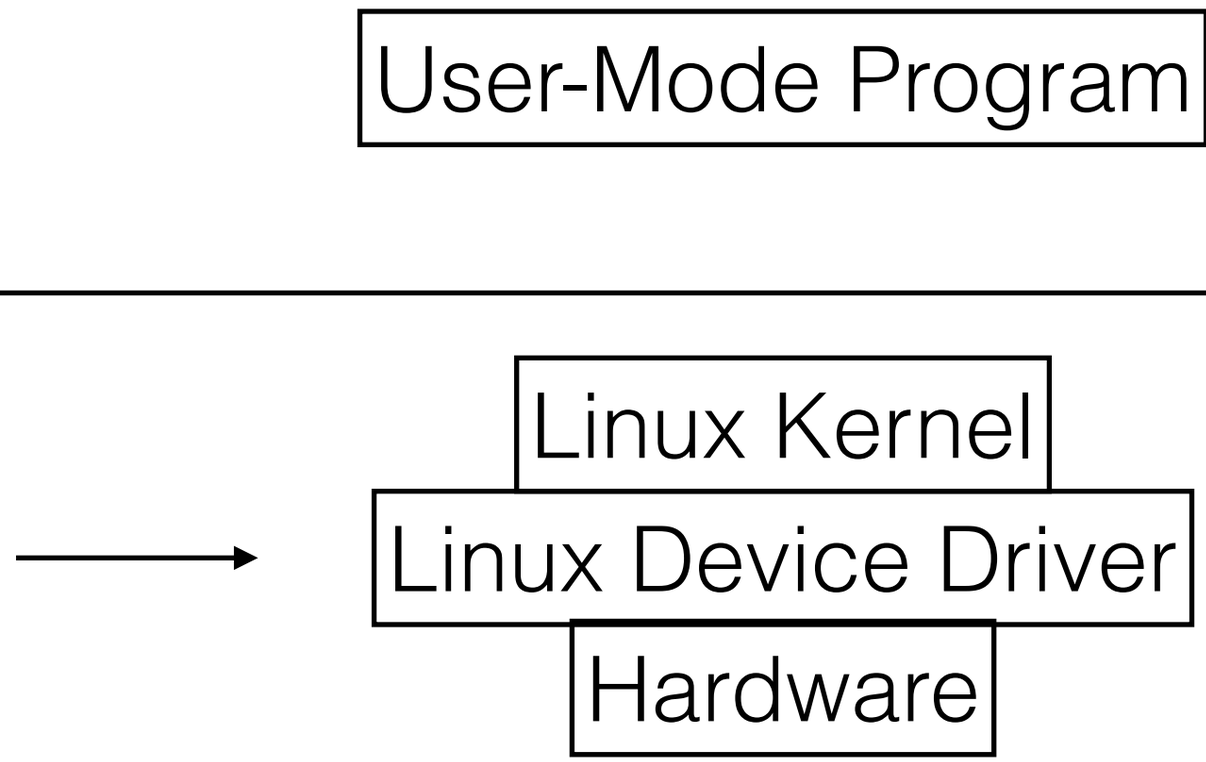
Block Diagram
View 1



Block Diagram
View 2 - User Mode Library



Block Diagram
View 3 - Kernel Mode Driver



Demo 1

RPi GPIO Access From C

Logon to RPi 4
(Your IP May Be Different)

```
$ ssh metaembedded@192.168.4.34
The authenticity of host '192.168.4.34 (192.168.4.34)' can't be established.
ED25519 key fingerprint is SHA256:QtQSfEtYwTkUAULb4FVQsCp4Z94wgbJnX8TF5/dWCOo.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:20: 192.168.1.228
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.4.34' (ED25519) to the list of known hosts.
metaembedded@192.168.4.34's password:
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May  8 15:00:09 2023 from 192.168.1.229
```

RPi 4 Baseline
(Yours may be different)

```
$ uname -a
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64 GNU/Linux

$ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 11 (bullseye)"
NAME="Raspbian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
VERSION_CODENAME=bullseye
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
```

sudo apt-get update

```
$ sudo apt-get update
Get:1 http://raspbian.raspberrypi.org/raspbian bullseye InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian bullseye InRelease [23.6 kB]
Get:3 http://raspbian.raspberrypi.org/raspbian bullseye/main armhf Packages [13.2 MB]
Get:4 http://archive.raspberrypi.org/debian bullseye/main armhf Packages [316 kB]
Get:5 http://archive.raspberrypi.org/debian bullseye/main armhf Contents (deb) [1,325 kB]
Fetched 14.9 MB in 8s (1,982 kB/s)
Reading package lists... Done
```

sudo apt-get search gpis

```
$ sudo apt-cache search gpio
airspyhf - HF+VHF software defined radio receiver - user runtime
gpiod - Tools for interacting with Linux GPIO character device - binary
ledmon - Enclosure LED Utilities
libgpiod-dev - C library for interacting with Linux GPIO device - static libraries and headers
libgpiod-doc - C library for interacting with Linux GPIO device - library documentation
libgpiod2 - C library for interacting with Linux GPIO device - shared libraries
libpigpio-dev - Client tools for Raspberry Pi GPIO control
libpigpio1 - Library for Raspberry Pi GPIO control
libpigpiod-if-dev - Development headers for client libraries for Raspberry Pi GPIO control
libpigpiod-if1 - Client library for Raspberry Pi GPIO control (deprecated)
libpigpiod-if2-1 - Client library for Raspberry Pi GPIO control
pigpio - Raspberry Pi GPIO control transitional package.
pigpio-tools - Client tools for Raspberry Pi GPIO control
pigpiod - Client tools for Raspberry Pi GPIO control
python-periphery-doc - Peripheral I/O (Documentation)
python3-libgpiod - Python bindings for libgpiod (Python 3)
python3-periphery - Peripheral I/O (Python3 version)
python3-pigpio - Python module which talks to the pigpio daemon (Python 3)
python3-rpi.gpio - Module to control Raspberry Pi GPIO channels (Python 3)
rpi.gpio-common - Module to control Raspberry Pi GPIO channels (common files)
```

dpkg -L libpigpio-dev

```
$ dpkg -L libpigpio-dev
/.
/usr
/usr/include
/usr/include/pigpio.h
/usr/lib
/usr/share
/usr/share/doc
/usr/share/doc/libpigpio-dev
/usr/share/doc/libpigpio-dev/changelog.Debian.gz
/usr/share/doc/libpigpio-dev/copyright
/usr/share/man
/usr/share/man/man3
/usr/share/man/man3/pigpio.3.gz
/usr/lib/libpigpio.so
```

sudo apt-get install libpigpio-dev

```
$ sudo apt-get install libpigpio-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libpigpio-dev is already the newest version (1.79-1+rpt1).
libpigpio-dev set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 63 not upgraded.
```

/usr/include/pigpio.h Part 1

```
pigpio is a C library for the Raspberry which allows control of the GPIO.

*Features*

o hardware timed PWM on any of GPIO 0-31

o hardware timed servo pulses on any of GPIO 0-31

o callbacks when any of GPIO 0-31 change state

o callbacks at timed intervals

o reading/writing all of the GPIO in a bank as one operation

o individually setting GPIO modes, reading and writing

o notifications when any of GPIO 0-31 change state

o the construction of output waveforms with microsecond timing

o rudimentary permission control over GPIO

o a simple interface to start and stop new threads

o I2C, SPI, and serial link wrappers

o creating and running scripts
```

/usr/include/pigpio.h
Part 2

```
*GPIO*

ALL GPIO are identified by their Broadcom number.
```

```
*Usage*

Include <pigpio.h> in your source files.
```

```
. .
gcc -Wall -pthread -o prog prog.c -lpigpio -lrt
sudo ./prog
```

/usr/include/pigpio.h
Part 3

```
[*gpioInitialise*] must be called before all other library functions
with the following exceptions:

. .
[*gpioCfg**]
[*gpioVersion*]
[*gpioHardwareRevision*]
```

```
If the library is not initialised all but the [*gpioCfg**],
[*gpioVersion*], and [*gpioHardwareRevision*] functions will
return error PI_NOT_INITIALISED.

If the library is initialised the [*gpioCfg**] functions will return
error PI_INITIALISED.
```

/usr/include/pigpio.h
Part 4

```
If you intend to rely on signals sent to your application, you should
turn off the internal signal handling as shown in this example:
```

```
. .
int cfg = gpioCfgGetInternals();
cfg |= PI_CFG_NOSIGHANDLER;  // (1<<10)
gpioCfgSetInternals(cfg);
int status = gpioInitialise();
. .
```

/usr/include/pigpio.h
Part 5

```
/*OVERVIEW

ESSENTIAL

gpioInitialise              Initialise library
gpioTerminate               Stop library

BASIC

gpioSetMode                 Set a GPIO mode
gpioGetMode                 Get a GPIO mode

gpioSetPullUpDown           Set/clear GPIO pull up/down resistor

gpioRead                    Read a GPIO
gpioWrite                   Write a GPIO
```

/usr/include/pigpio.h
gpioInitalize

```
/*F*/
int gpioInitialise(void);
/*D
Initialises the library.

Returns the pigpio version number if OK, otherwise PI_INIT_FAILED.

gpioInitialise must be called before using the other library functions
with the following exceptions:

. .
[*gpioCfg**]
[*gpioVersion*]
[*gpioHardwareRevision*]
. .
```

```
...
if (gpioInitialise() < 0)
{
   // pigpio initialisation failed.
}
else
{
   // pigpio initialised okay.
}
```

/usr/include/pigpio.h
gpioTerminate

```
/*F*/
void gpioTerminate(void);
/*D
Terminates the library.

Returns nothing.

Call before program exit.

This function resets the used DMA channels, releases memory, and
terminates any running threads.

...
gpioTerminate();
...
D*/
```

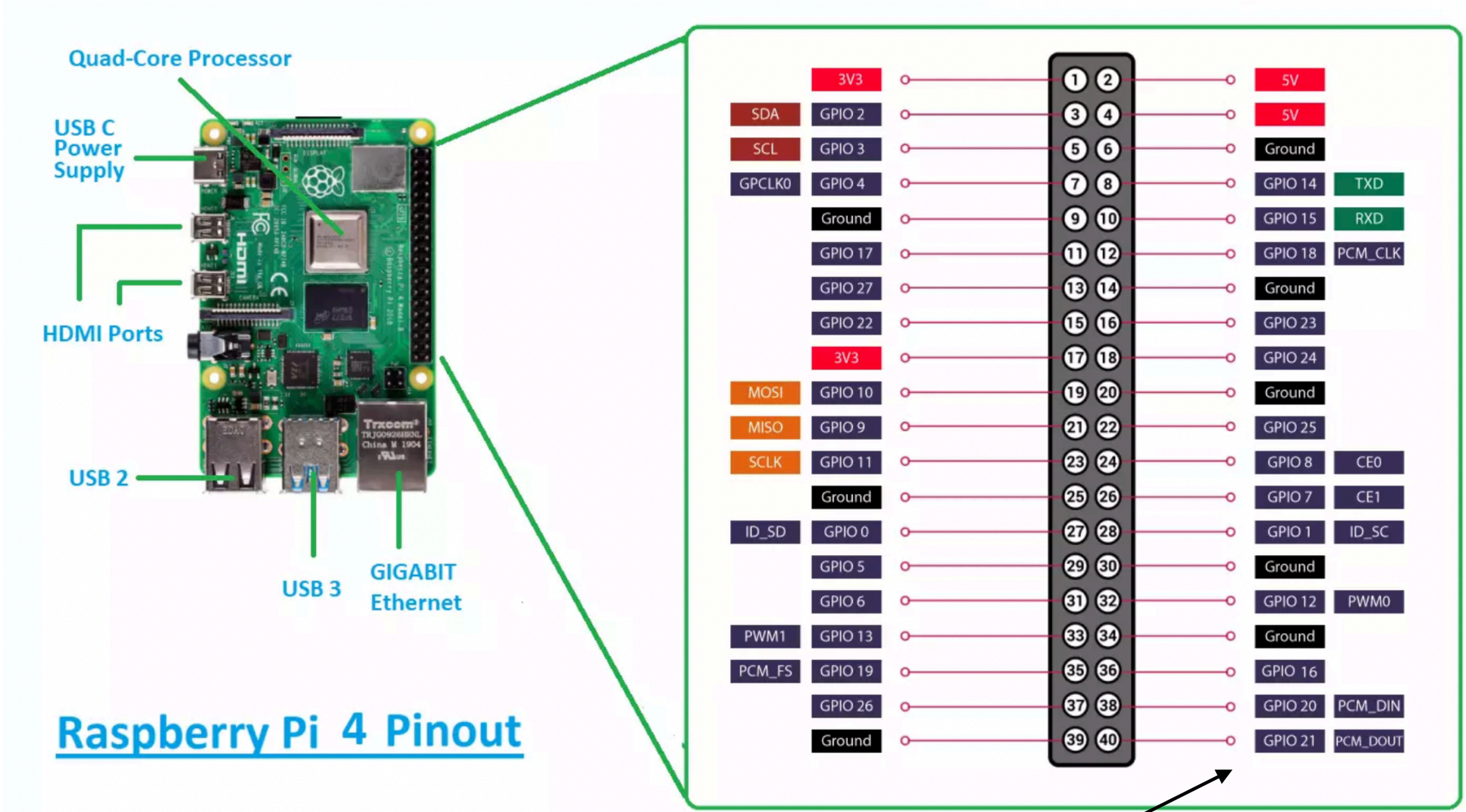
/usr/include/pigpio.h
gpioSetMode

```
/*F*/
int gpioSetMode(unsigned gpio, unsigned mode);
/*D
Sets the GPIO mode, typically input or output.

. .
gpio: 0-53
mode: 0-7
. .

Returns 0 if OK, otherwise PI_BAD_GPIO or PI_BAD_MODE.

Arduino style: pinMode.

...
gpioSetMode(17, PI_INPUT);  // Set GPIO17 as input.

gpioSetMode(18, PI_OUTPUT); // Set GPIO18 as output.

gpioSetMode(22,PI_ALT0);    // Set GPIO22 to alternative mode 0.
```

/usr/include/pigpio.h
gpioSetMode

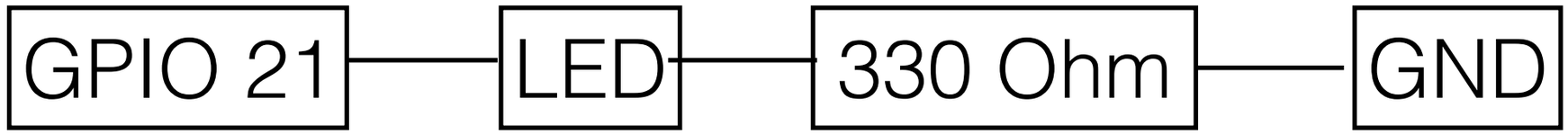
```
/*F*/
int gpioWrite(unsigned gpio, unsigned level);
/*D
Sets the GPIO level, on or off.

. .
 gpio: 0-53
level: 0-1
. .

Returns 0 if OK, otherwise PI_BAD_GPIO or PI_BAD_LEVEL.

If PWM or servo pulses are active on the GPIO they are switched off.

Arduino style: digitalWrite

...
gpioWrite(24, 1); // Set GPIO24 high.
...
D*/
```

RPi 4 Pinout



Let's Connect LED to GPIO Pin 21

Block Diagram



Build

```
$ gcc -Wall -o led21-blink led21-blink.c -lpthread -lpigpio -lrt

$ ldd led21-blink
   /usr/lib/arm-linux-gnueabihf/libarmmem-${PLATFORM}.so => /usr/lib/arm-linux-gnueabihf/libarmmem-v8l.so
(0xf7d2d000)
   libpigpio.so.1 => /lib/libpigpio.so.1 (0xf7c35000)
   libc.so.6 => /lib/arm-linux-gnueabihf/libc.so.6 (0xf7ae1000)
   libpthread.so.0 => /lib/arm-linux-gnueabihf/libpthread.so.0 (0xf7ab5000)
   /lib/ld-linux-armhf.so.3 (0xf7d42000)
```

lpthread - POSIX Thread Library

-lpigpio - Pi GPIO Library

-lrt -Real-time Extensions Library

led21-blink.c
Part 1

```
/* led21-blink.c
 *
 * Connect LED to GPIO 21 and make it blink
 *
 * To build:
 *
 * gcc -Wall -o led21-blink  led21-blink.c -lpigpio -lrt
 *
 * To run:
 *
 * sudo ./led21-blink
 *
 */

#include <stdio.h>
#include <pigpio.h>
#include <signal.h>
#include <unistd.h>

#define LED_PIN 21

int running = 1;
void handle_sig_int(int sig) {
                running = 0;
}
```

led21-blink.c
Part 2

```c
int main() {
        int result = gpioInitialise();
        if (result < 0) {
                fprintf(stderr, "gpioInitialise() failed\n")
                result = 1;
                goto getOut;
        }

result = gpioSetMode(LED_PIN, PI_OUTPUT);
if (result < 0) {
        fprintf(stderr, "gpioSetMode() failed\n");
        result = 2;
        goto getOut;
}

// We need to use signals
int cfg = gpioCfgGetInternals();
cfg |= PI_CFG_NOSIGHANDLER;
gpioCfgSetInternals(cfg);

signal(SIGINT, handle_sig_int);
```

led21-blink.c
Part 3

```
        int toggle = 1;
        while (running) {
                sleep(1);
                result = gpioWrite(LED_PIN, toggle);
                if (result < 0) break;
                toggle ^= 1; //Toggle bit
        }

getOut:
        gpioTerminate();
        return 0;
}
```

Build and Run

```
$ ls -l /dev/gpiomem
crw-rw---- 1 root gpio 245, 0 Apr 26 13:37 /dev/gpiomem
```

```
$ gcc -Wall -o led21-blink led21-blink.c -lpthread -lpigpio -lrt
```

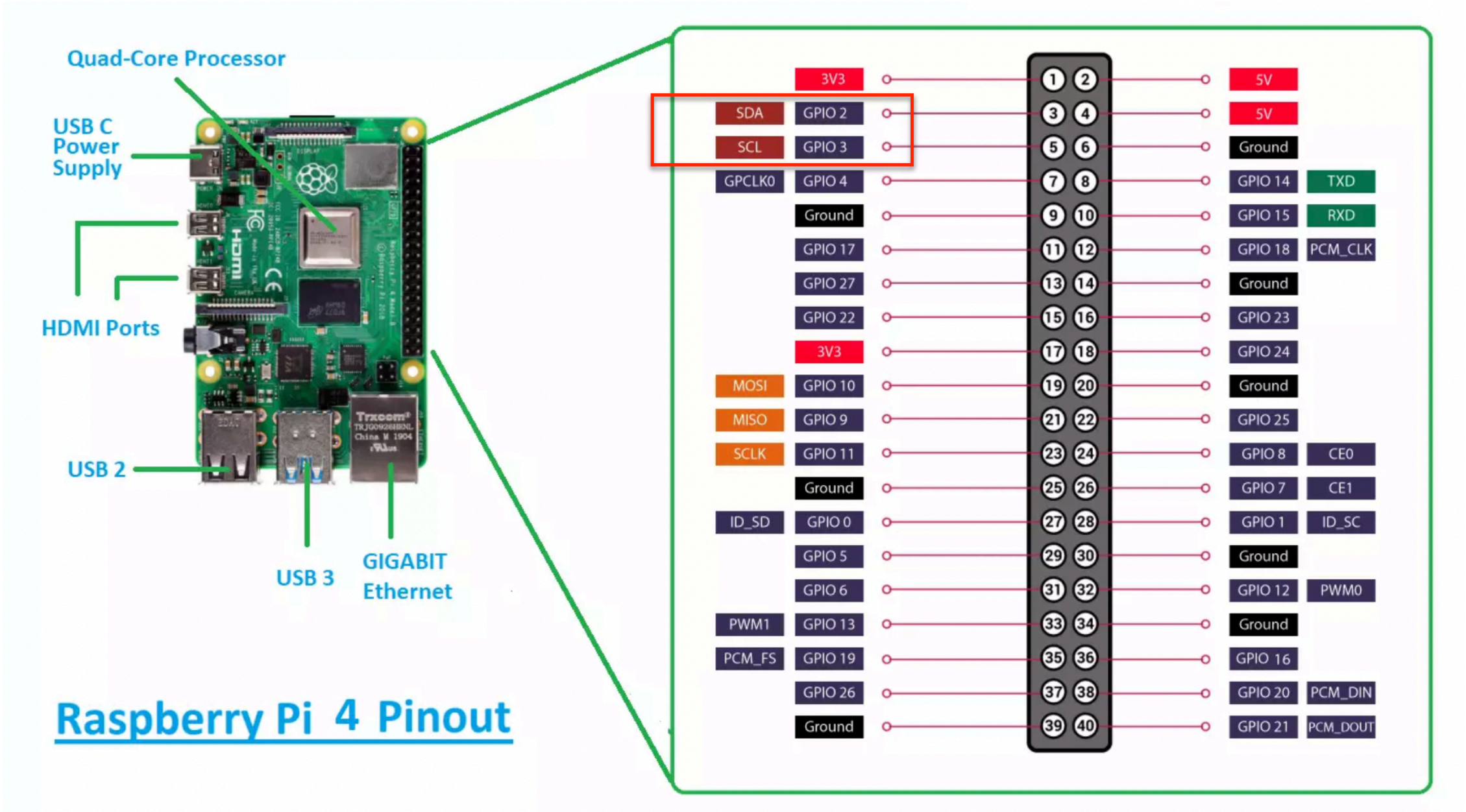
```
$ sudo ./led21-blink
```

NOTE: Result is LED blinking

Questions about GPIO Access from C?

I2C Access from C

RPi 4 Pinout
I2C Pins (SDA, SCL)



/usr/include/pigpio.h

```
I2C

i2cOpen                   Opens an I2C device
i2cClose                  Closes an I2C device

i2cWriteQuick             SMBus write quick

i2cReadByte               SMBus read byte
i2cWriteByte              SMBus write byte

i2cReadByteData           SMBus read byte data
i2cWriteByteData          SMBus write byte data

i2cReadWordData           SMBus read word data
i2cWriteWordData          SMBus write word data

i2cReadBlockData          SMBus read block data
i2cWriteBlockData         SMBus write block data

i2cReadI2CBlockData       SMBus read I2C block data
i2cWriteI2CBlockData      SMBus write I2C block data

i2cReadDevice             Reads the raw I2C device
i2cWriteDevice            Writes the raw I2C device
```

/usr/include/pigpio.h
i2cOpen

```
/*F*/
int i2cOpen(unsigned i2cBus, unsigned i2cAddr, unsigned i2cFlags);
/*D
This returns a handle for the device at the address on the I2C bus.

. .
  i2cBus: >=0
 i2cAddr: 0-0x7F
i2cFlags: 0
. .

No flags are currently defined.  This parameter should be set to zero.

Physically buses 0 and 1 are available on the Pi.  Higher numbered buses
will be available if a kernel supported bus multiplexor is being used.

The GPIO used are given in the following table.

      @ SDA @ SCL
I2C 0 @  0  @  1
I2C 1 @  2  @  3

Returns a handle (>=0) if OK, otherwise PI_BAD_I2C_BUS, PI_BAD_I2C_ADDR,
PI_BAD_FLAGS, PI_NO_HANDLE, or PI_I2C_OPEN_FAILED.
```

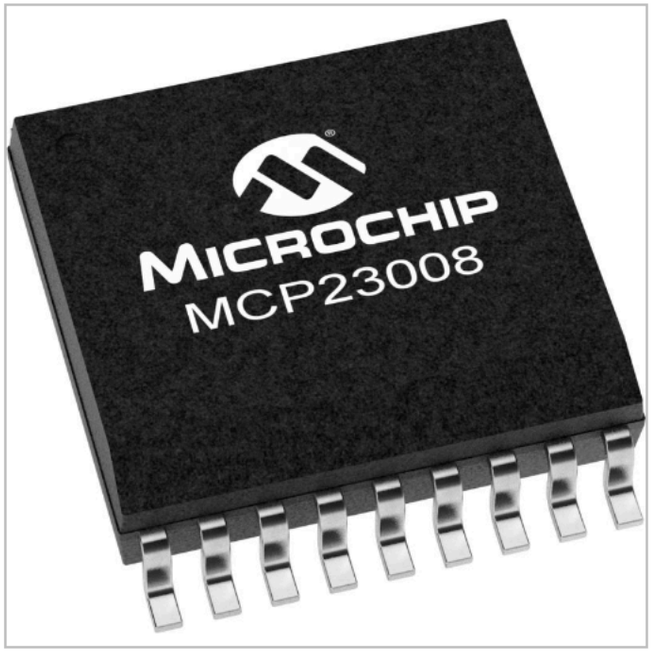
/usr/include/pigpio.h
i2cClose

```
/*F*/
int i2cClose(unsigned handle);
/*D
This closes the I2C device associated with the handle.

. .
handle: >=0, as returned by a call to [*i2cOpen*]
. .

Returns 0 if OK, otherwise PI_BAD_HANDLE.
D*/
```

/usr/include/pigpio.h
i2cWriteByteData

```
/*F*/
int i2cWriteByteData(unsigned handle, unsigned i2cReg, unsigned bVal);
/*D
This writes a single byte to the specified register of the device
associated with handle.

. .
handle: >=0, as returned by a call to [*i2cOpen*]
i2cReg: 0-255, the register to write
  bVal: 0-0xFF, the value to write
. .

Returns 0 if OK, otherwise PI_BAD_HANDLE, PI_BAD_PARAM, or
PI_I2C_WRITE_FAILED.

Write byte. SMBus 2.0 5.5.4
. .
S Addr Wr [A] i2cReg [A] bVal [A] P
. .
D*/
```

MCP23008



MCP23008 ☆

8-Bit I2C I/O Expander with Serial Interface

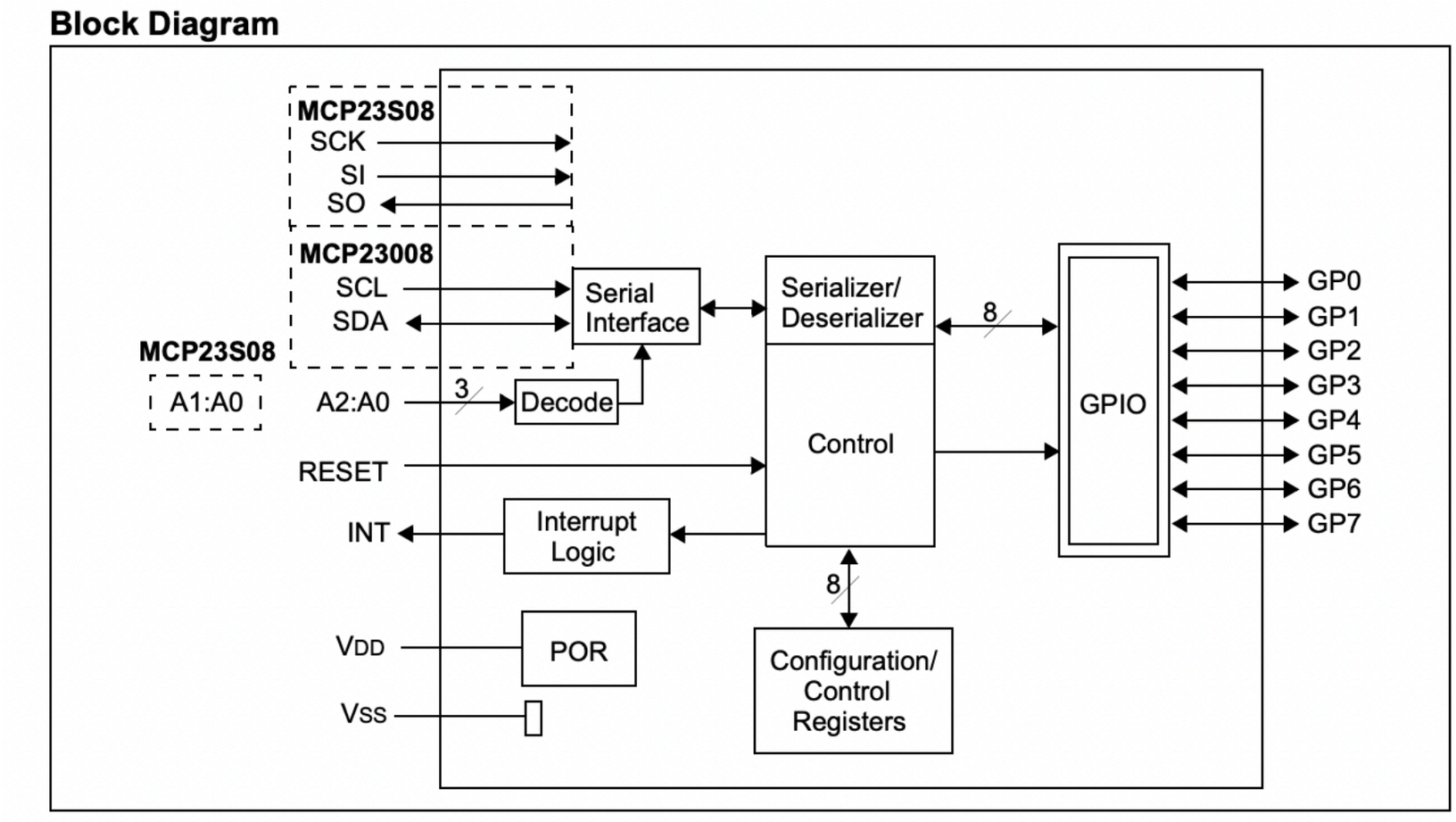
Status: In Production.

Documentation Symbols

Recommended for Automotive Design

MCP23008/MCP23S08 Data Sheet: PDF

MCP23008
Block Diagram

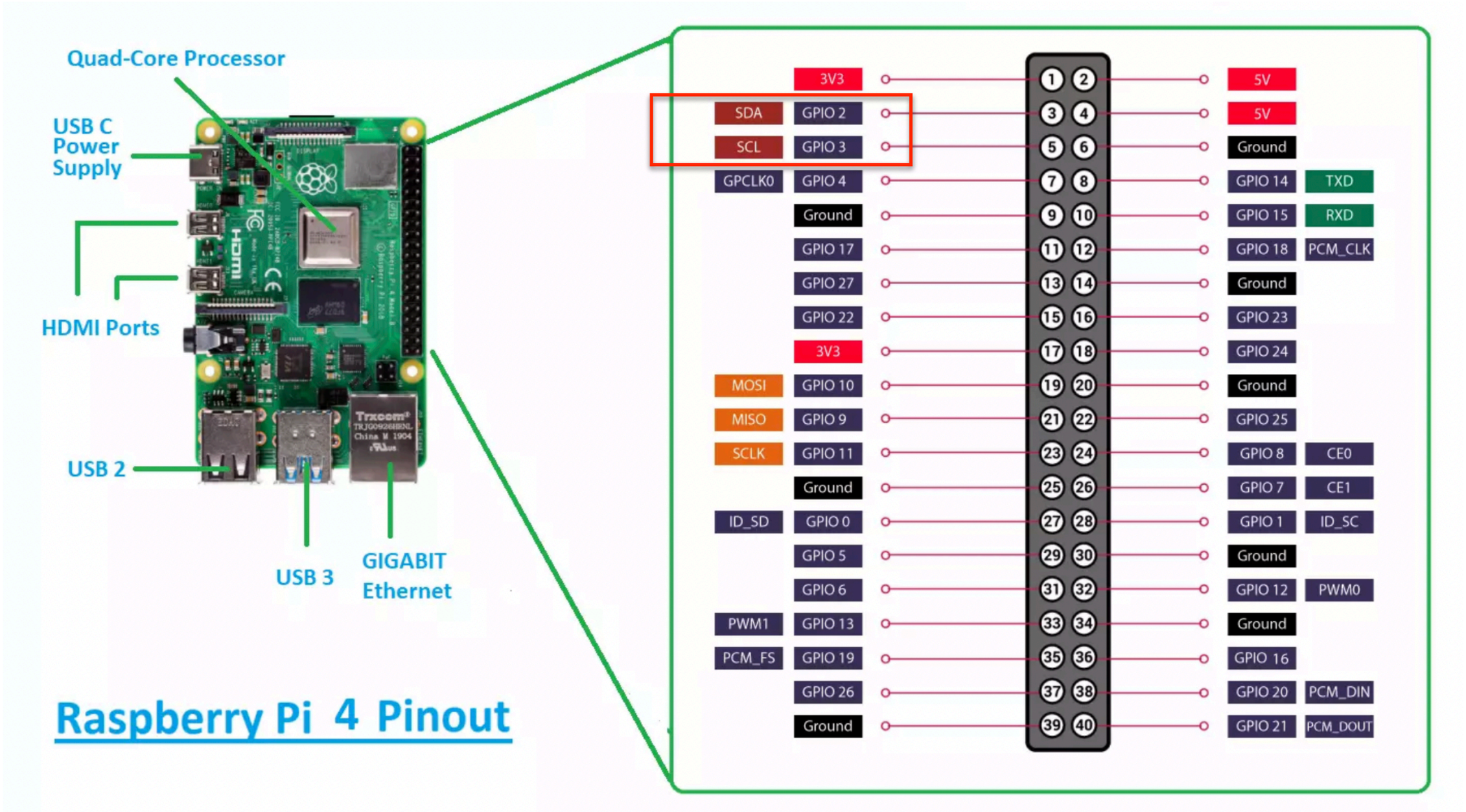


MCP23008 Registers

TABLE 1-2: REGISTER ADDRESSES

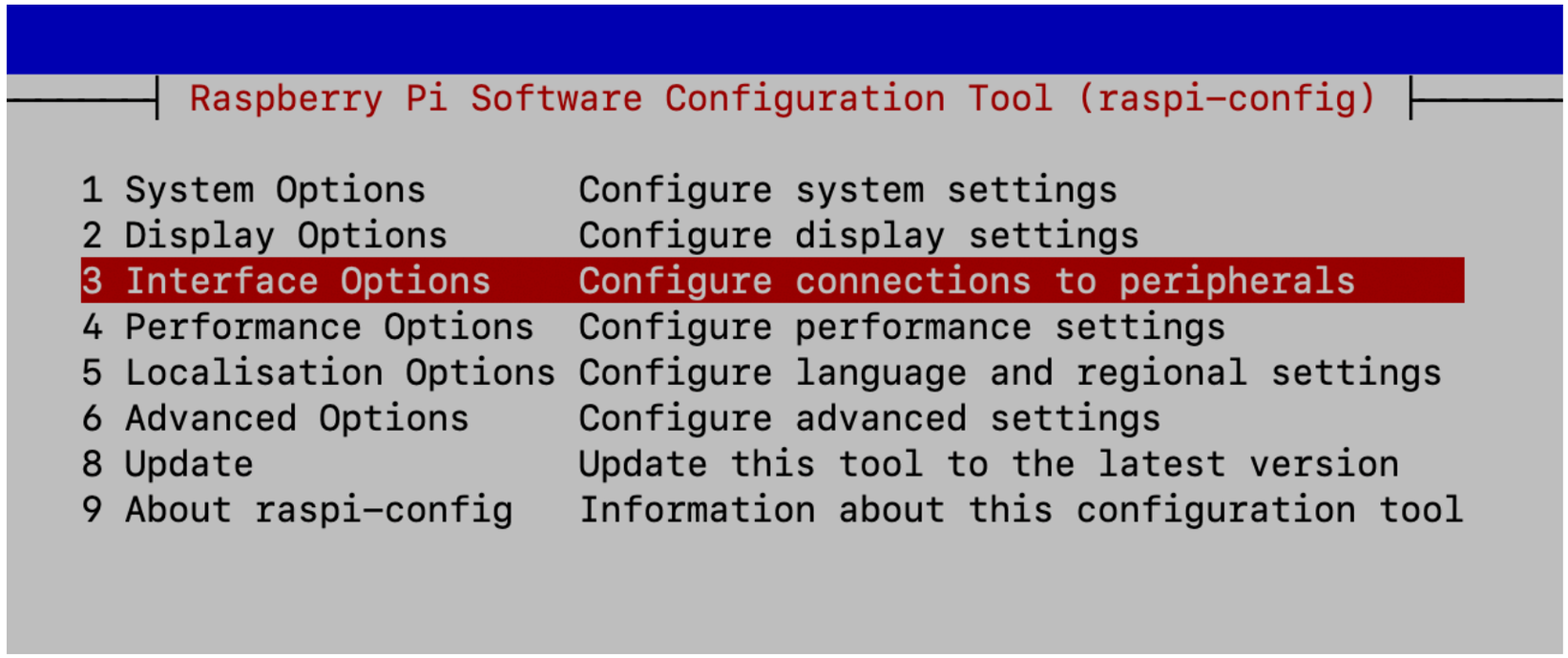
Address	Access to:
00h	IODIR
01h	IPOL
02h	GPINTEN
03h	DEFVAL
04h	INTCON
05h	IOCON
06h	GPPU
07h	INTF
08h	INTCAP (Read-only)
09h	GPIO
0Ah	OLAT

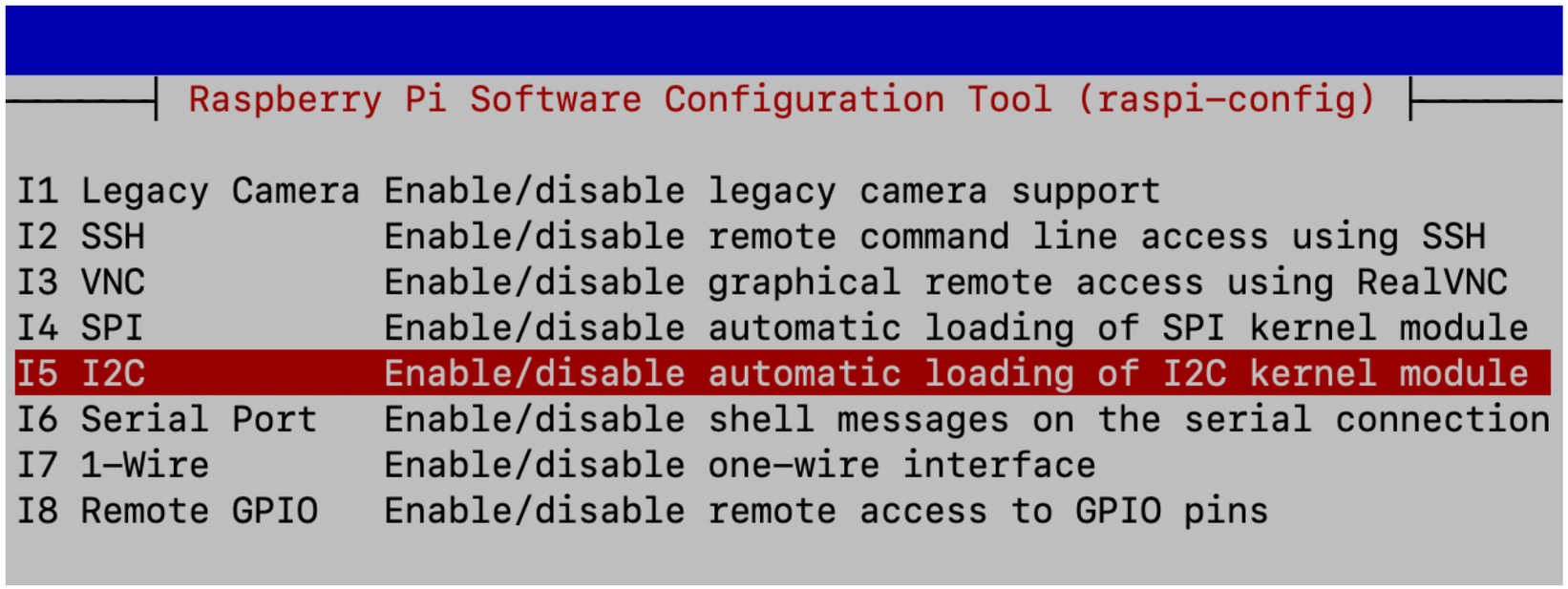
RPi 4 Pinout
I2C Pins (SDA, SCL)



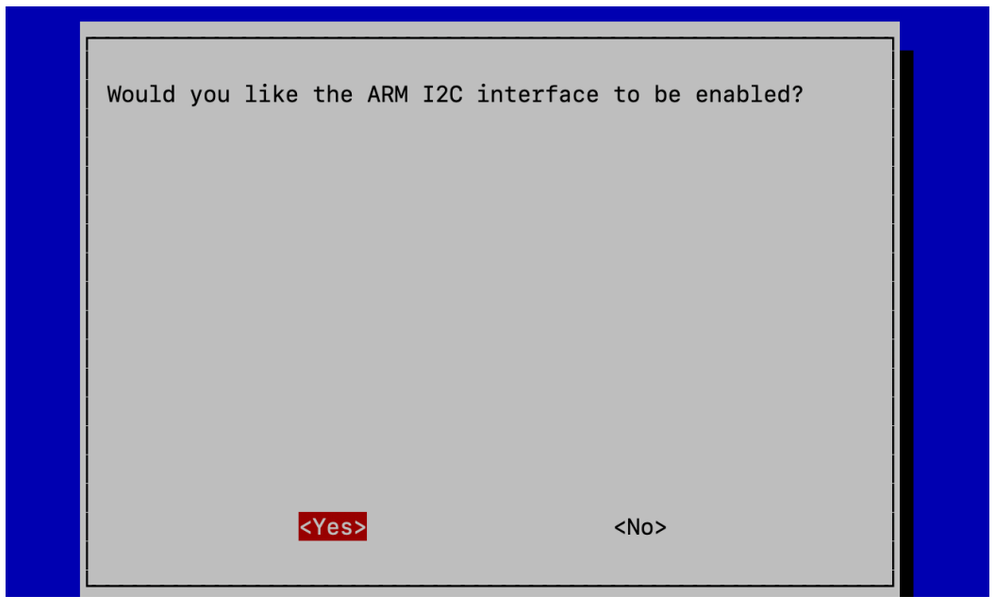
Configuration of I2C Part 1

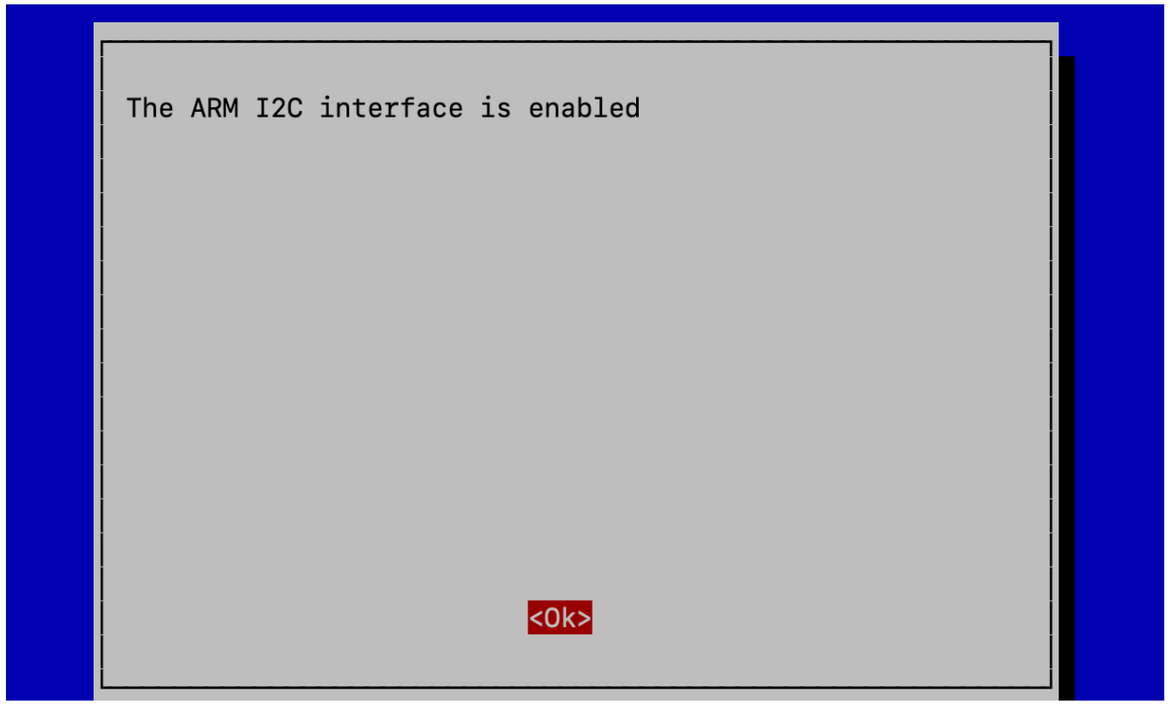
```
sudo raspi-config
```





Configuration of I2C Part 2





lsmod

```
$ lsmod | head
Module                  Size  Used by
i2c_bcm2835            16384  0
xt_CHECKSUM            16384  1
nft_chain_nat          16384  1
xt_MASQUERADE          16384  1
nf_nat                 49152  2 nft_chain_nat,xt_MASQUERADE
nf_conntrack          143360  2 nf_nat,xt_MASQUERADE
nf_defrag_ipv6         20480  1 nf_conntrack
nf_defrag_ipv4         16384  1 nf_conntrack
xt_tcpudp              16384  5
```

i2c-led.c
Part 1

```c
/* i2c-blink.c
 *
 * Blink LEDs connected to I2C Device
 *
 * To build:
 *
 * gcc -Wall -o i2c-blink  i2c-blink.c -lpigpio -lrt
 *
 * To run:
 *
 * sudo ./i2c-blink
 *
 */

#include <stdio.h>
#include <pigpio.h>
#include <signal.h>
#include <unistd.h>

#define MCP23008_I2C_ADDR 0x20

int running = 1;
void handle_sig_int(int sig) {
                running = 0;
}
```

i2c-led.c
Part 2

```
int main() {
        int result = gpioInitialise();
        if (result < 0) {
                fprintf(stderr, "gpioInitialise() failed\n");
                result = 1;
                goto getOut;
        }

        // We need to use signals
        int cfg = gpioCfgGetInternals();
        cfg |= PI_CFG_NOSIGHANDLER;
        gpioCfgSetInternals(cfg);

        signal(SIGINT, handle_sig_int);

        unsigned i2cBus = 0;
        unsigned i2cAddr = MCP23008_I2C_ADDR;
        unsigned i2cFlags = 0;
        int i2cHandle = i2cOpen(i2cBus, i2cAddr, i2cFlags);
        if (i2cHandle < 0) {
                fprintf(stderr, "i2cOpen() failed\n");
                goto getOut;
        }
```

i2c-led.c
Part 3

```
        // Set MCP23008 I/O direction for all pins as output (0x00)
        unsigned i2cReg = 0;
        unsigned i2cValue = 0;
        result = i2cWriteByteData(i2cHandle, i2cReg, i2cValue);
        if (result != 0) {
                fprintf(stderr, "i2cWriteByteData() failed\n");
                goto getOut;
        }

        // Toggle the pins connected to the MCP23008 I/O expander
        i2cReg = 0x09;
        while (1) {
                i2cValue = 0xff;
                i2cWriteByteData(i2cHandle, i2cReg, i2cValue);
                sleep(5);
                i2cValue = 0;
                i2cWriteByteData(i2cHandle, i2cReg, i2cValue);
                sleep(5);
        }

getOut:
        if (i2cHandle >= 0) i2cClose(i2cHandle);
        gpioTerminate();
        return 0;
}
```

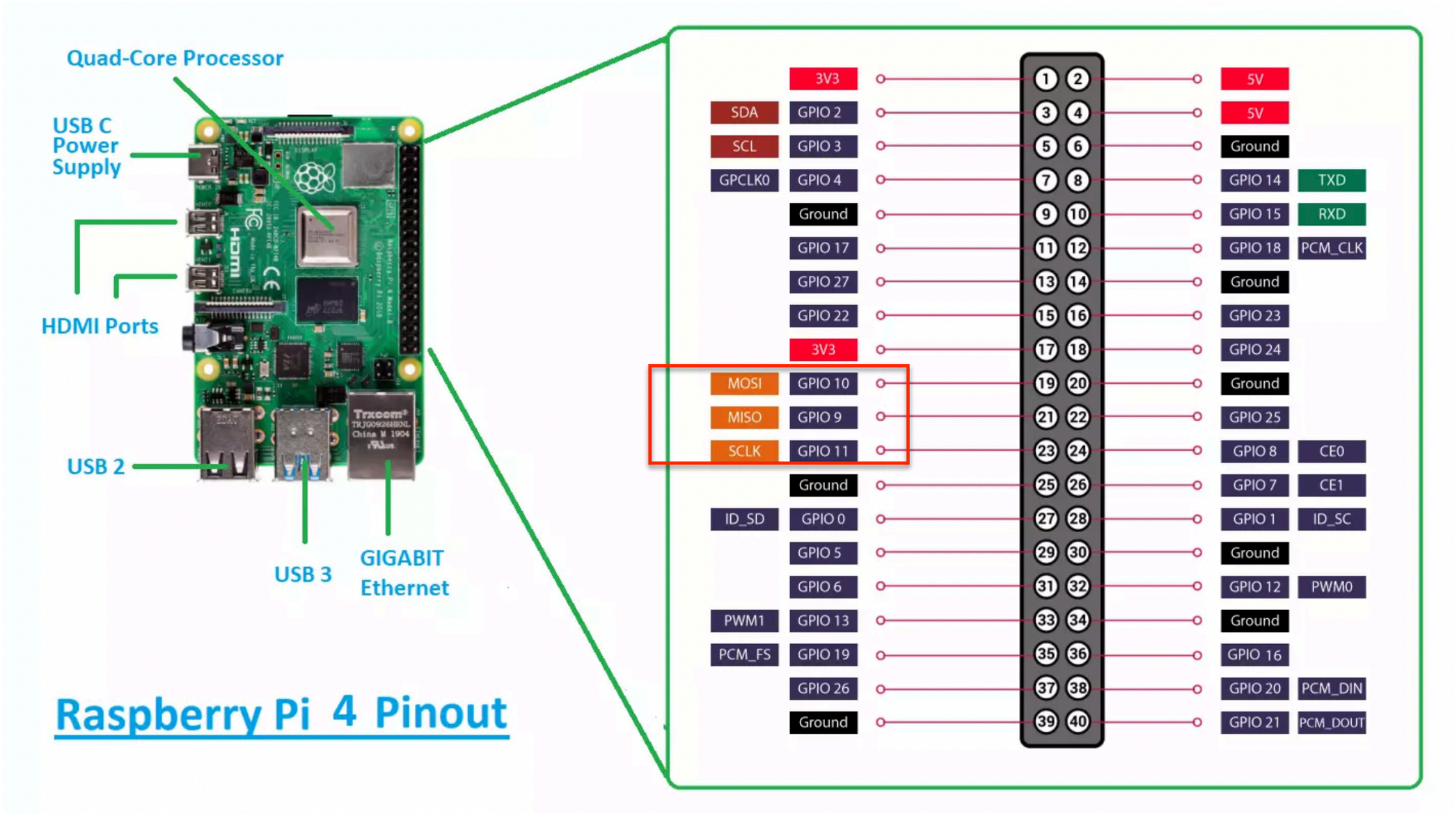
Build and Run

```
$ !gcc
gcc -Wall -o i2c-blink i2c-blink.c -lpthread -lpigpio -lrt
```

```
$ sudo ./i2c-blink
i2cOpen() failed
```

SPI Access from C

RPi 4 Pinout
SPI Pins (MOSI,MISO,SCLK)

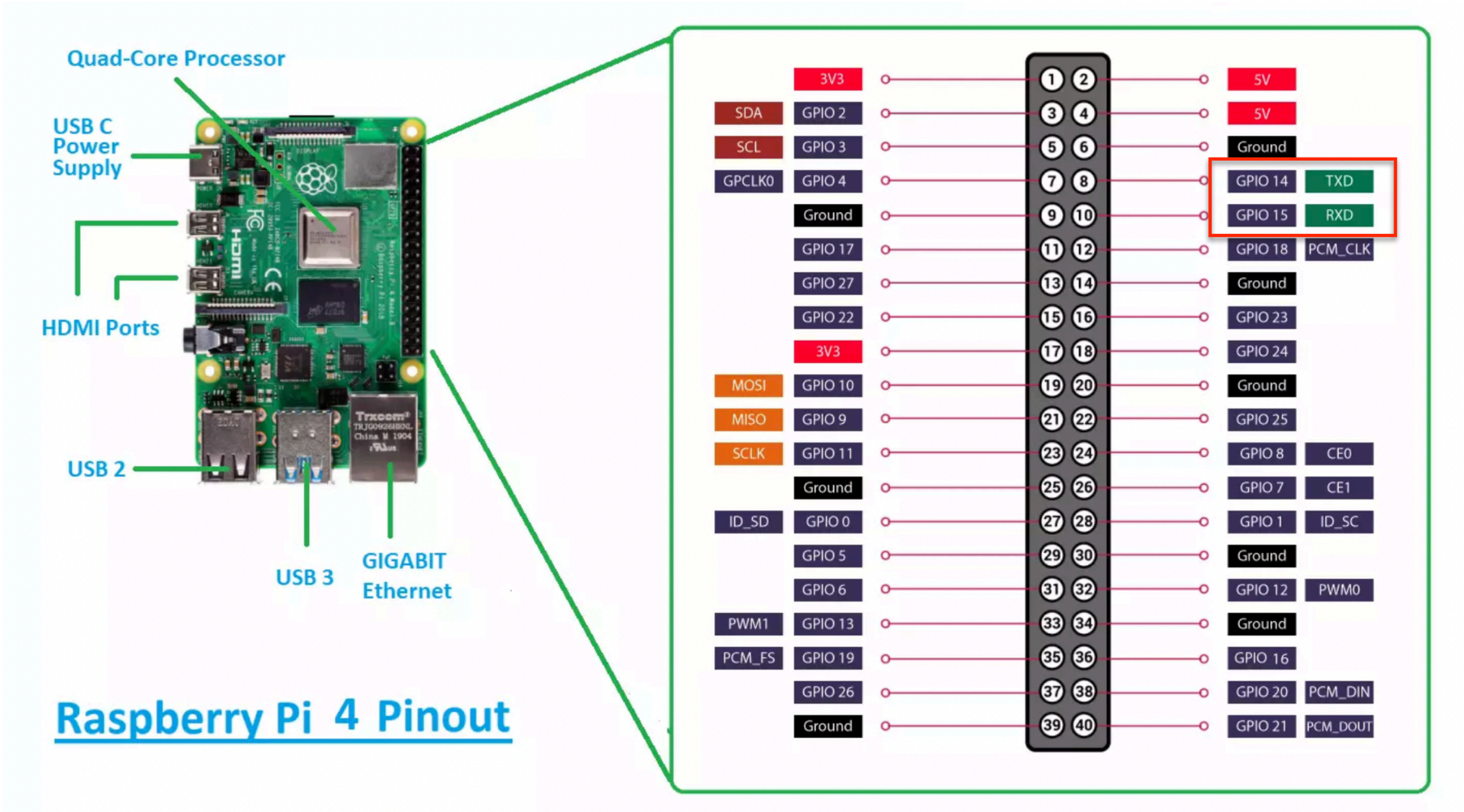


/usr/include/pigpio.h

```
SPI

spiOpen                   Opens a SPI device
spiClose                  Closes a SPI device

spiRead                   Reads bytes from a SPI device
spiWrite                  Writes bytes to a SPI device
spiXfer                   Transfers bytes with a SPI device
```

Serial Access from C

RPi 4 Pinout
Serial Pins (TXD, RXD)



/usr/include/pigpio.h

```
SERIAL

serOpen                         Opens a serial device
serClose                        Closes a serial device

serReadByte                     Reads a byte from a serial device
serWriteByte                    Writes a byte to a serial device

serRead                         Reads bytes from a serial device
serWrite                        Writes bytes to a serial device

serDataAvailable                Returns number of bytes ready to be read

SERIAL_BIT_BANG_(read_only)

gpioSerialReadOpen              Opens a GPIO for bit bang serial reads
gpioSerialReadClose             Closes a GPIO for bit bang serial reads

gpioSerialReadInvert            Configures normal/inverted for serial reads
```