Using the Raspberry Pi

GPIO Ports

Dr Mark Fisher,
School of Computing Sciences,
UEA, Norwich.
mark.fisher@uea.ac.uk

Aims

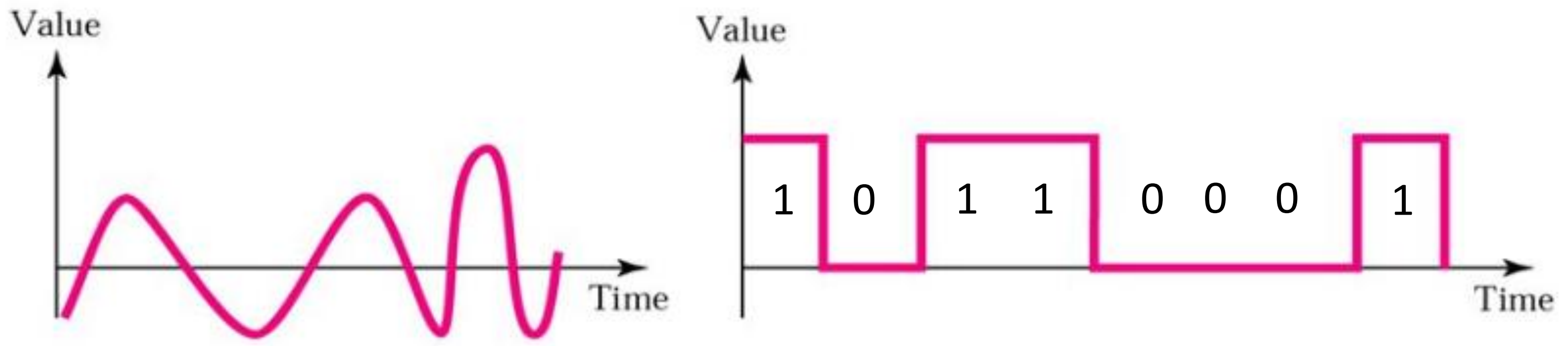
- To describe basic GPIO in detail.
  - To do this we'll need to learn some digital electronics.
- To describe GPIO's 'alternate function'
- To investigate practicalities of connecting switches and LEDs to GPIO ports.
- To discuss serial data transmission, highlighting some popular protocols

What is GPIO

- GPIO -> General Purpose Input Output
  - Essentially a versatile interface that can be configured by software.
- Does my PC / Laptop provide a GPIO interface?
  - No – PC interface standards are generally focused to the needs of humans.
  - Although USB ports can be adapted to provide GPIO
- The GPIO interface allows the computer to generate electrical outputs (O) and read electrical inputs (I).
- The type of electrical signals that can be handled by the Raspberry Pi GPIO is limited to a class of signals known as digital.
- Digital electrical signals are limited to just two voltages.
  - Logic 1 is usually represented by a high voltage (usually 3.3v or 5v - depends on the fabrication technology)
  - Logic 0 is usually represented by a low voltage (usually 0v)

Two Types of Electrical Signal

- An analogue signal is a continuous wave form that changes smoothly over time.
- A digital signal is discrete. It can have only a limited number of defined values. Often as simple as 0 and 1.



a. Analog signal

b. Digital signal

Digital Signals

- An example of digital data is the data stored in the memory of a computer. It is often represented as a series of 1s and 0s
- It is converted to an electrical signal when it is transferred from one place to another, either inside or outside the computer.

Digital Signals

- Computers manipulate digital data.
- A single data bit represents two (binary) states.
- But several data bits can be grouped together to represent numbers as binary code.
- For example, a group of 3-bits comprises 8 unique codes which can be interpreted as representing decimal numbers 0-7.

BINARY DECIMAL

MSB LSB

$0\ 0\ 0 = (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 0$

$0\ 0\ 1 = (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 1$

$0\ 1\ 0 = (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 2$

$0\ 1\ 1 = (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 3$

$1\ 0\ 0 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4$

$1\ 0\ 1 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 5$

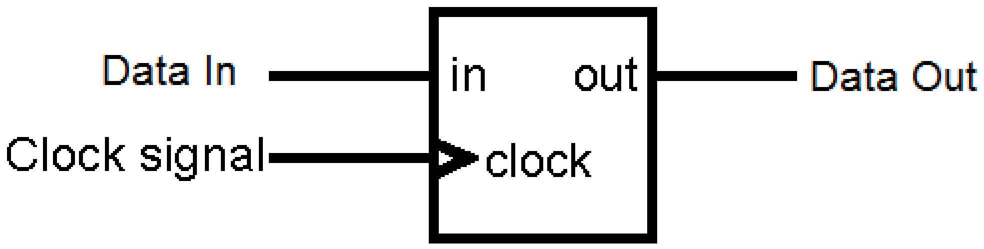
$1\ 1\ 0 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6$

$1\ 1\ 1 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 7$

Digital Logic Circuits

- Since the 1960s manufacturers have developed integrated circuits (ICs), each designed to perform specific operations on digital signals.
- Logical operations such as AND, OR, NOT etc. are basic building blocks used to create more complex functions used by digital computers.
- A register, latch or flip-flop is a simple memory device that can store 1 bit of data.

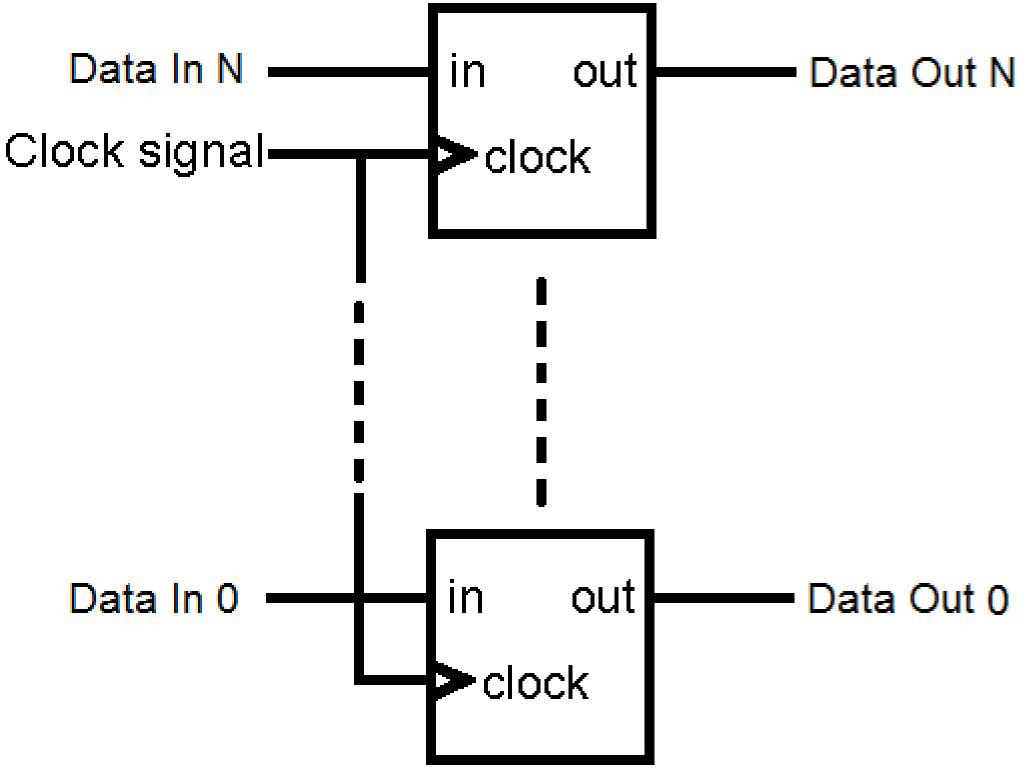
Register and Latch Circuit Model

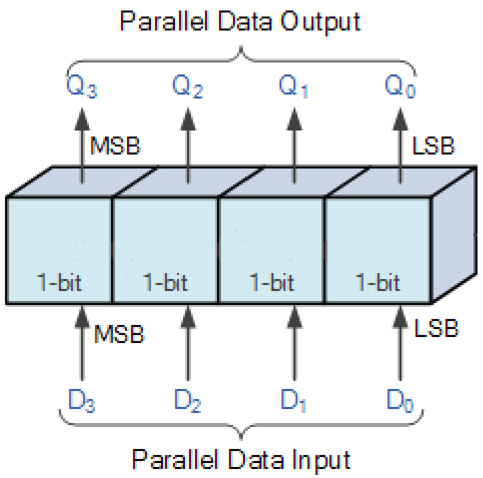


When Clock signal changes from ‘0’ to ‘1’ state of Data in is stored and transferred to Data Out.

Parallel In / Parallel Out (PIPO) Register

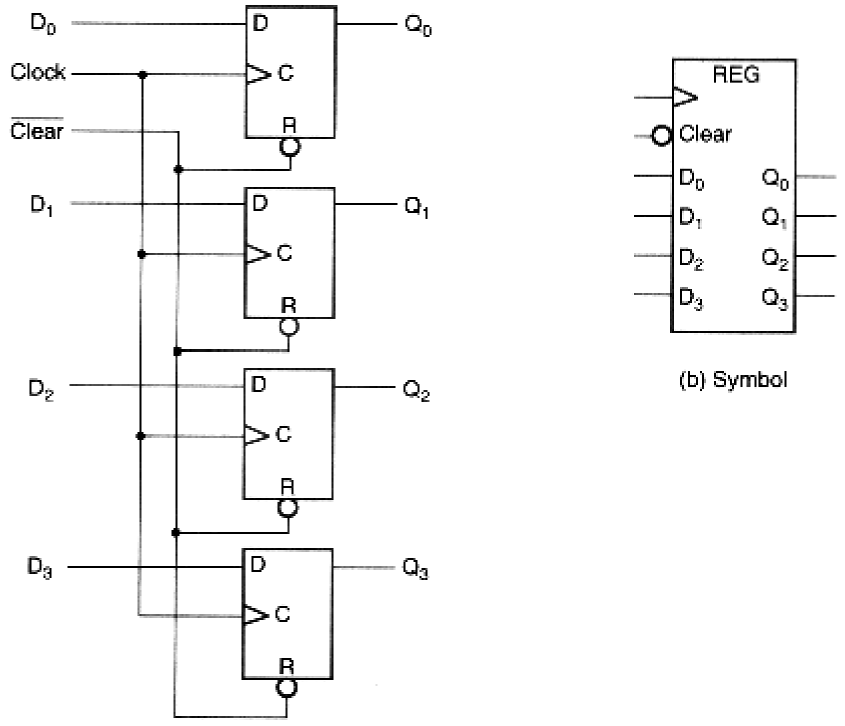
- Individual registers can be concatenated so multiple bits can be read/written in parallel.





Simple Digital Output Port

- A simple output port comprises a register.
- Registers are logic circuits that can store binary data (bits).
- A flip-flop circuit can store 1-bit of data.
- D-types represent one of a number of types of flip-flop circuit.
- To write to the flip-flop we present data on the 'D' input and then toggle the clock (the write occurs on the low-high transition)

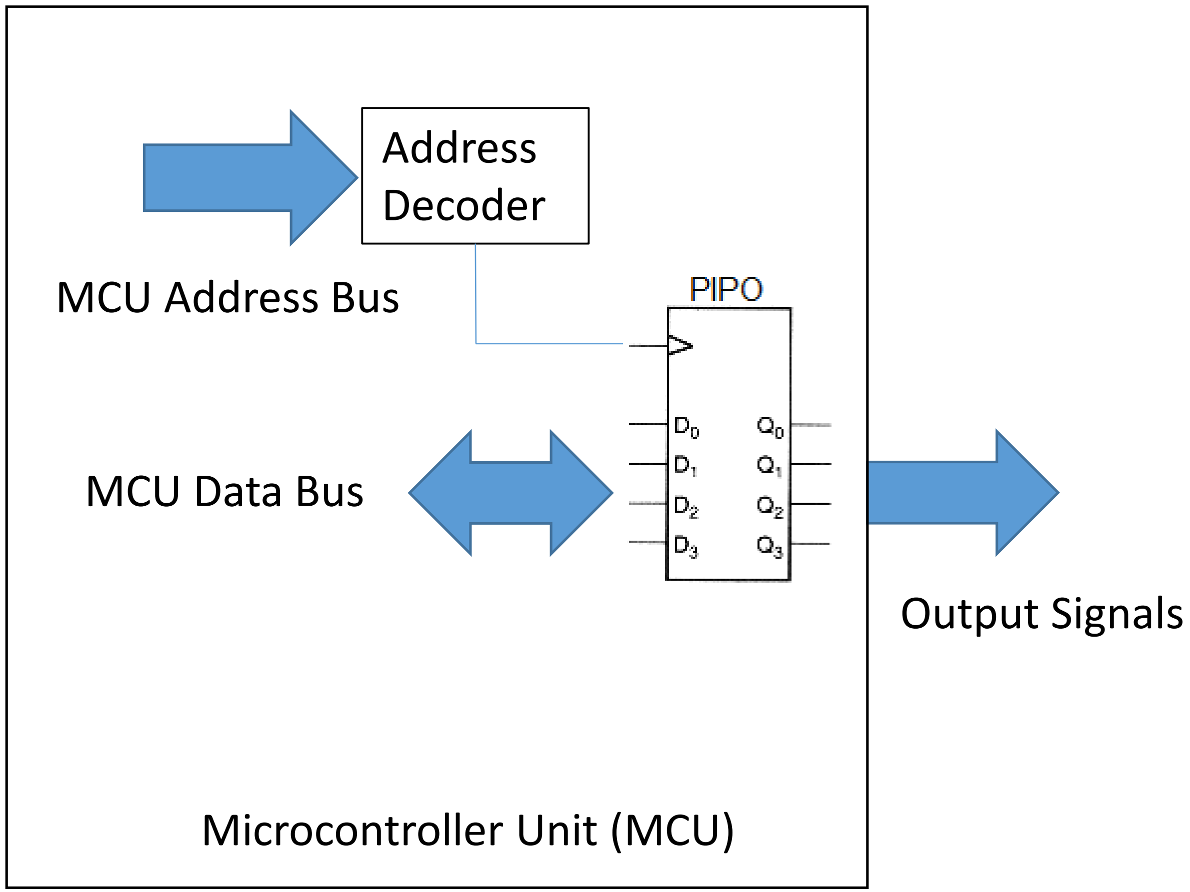


(a) Logic diagram

(b) Symbol

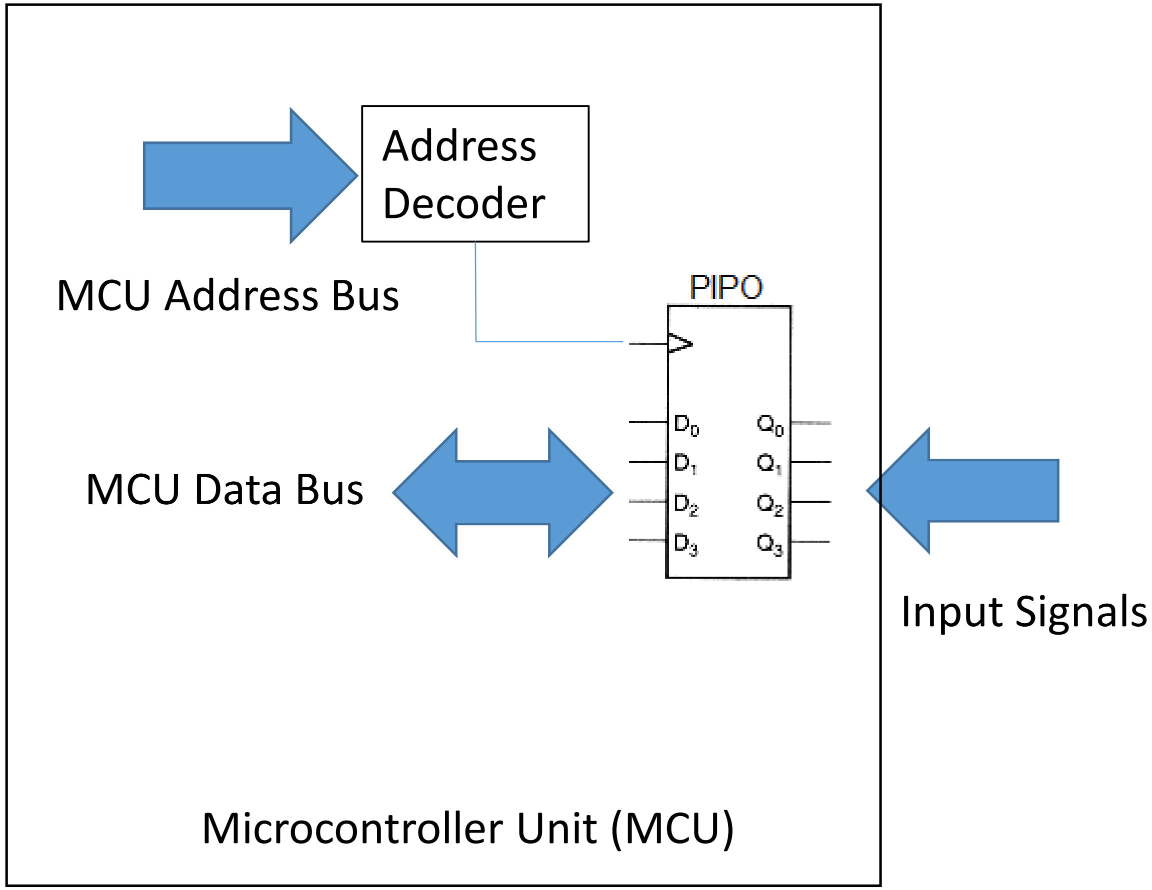
Simple Output Port

- To make a simple output port we just need to add a address decoder.
- The address decoder recognises a specific address.
- The decoder generates a clock when this address is issued by the MCU.
- The decoder itself is another very simple logic circuit.



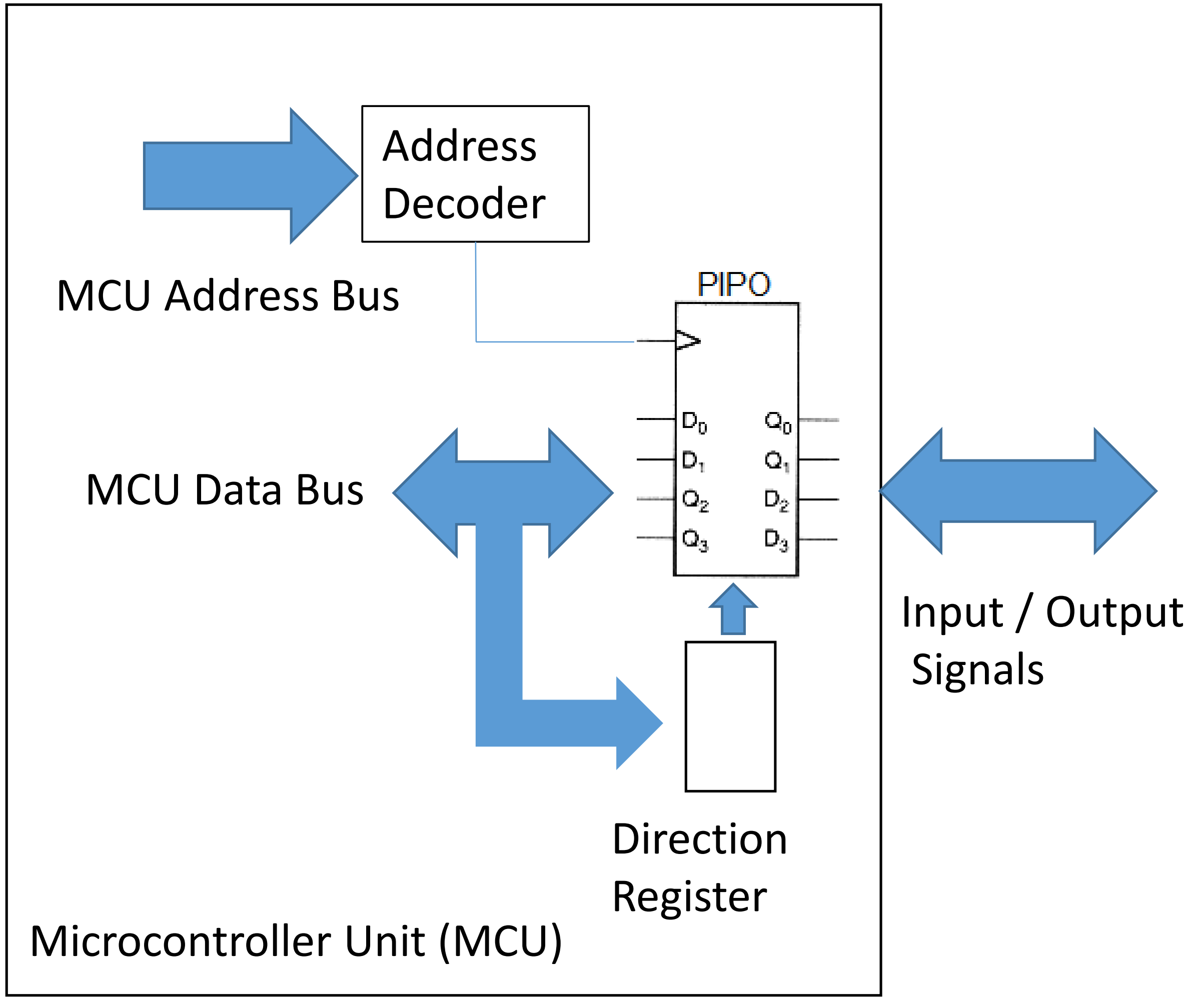
Simple Input Port

- To make a simple input port we just need to rewire our register.



Input / Output Port

- If we make the I/O datapath programmable we can configure individual bits as outputs or inputs.
- By writing to individual bits of the configuration register we can change the I/O interconnect.

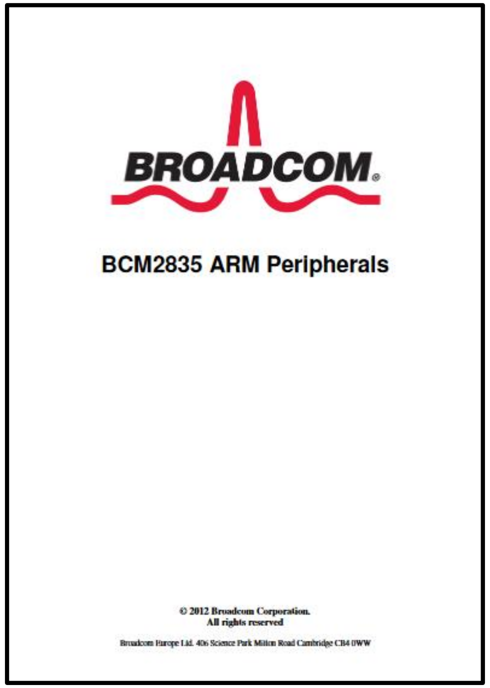


Raspberry Pi SoC

- The Raspberry Pi is implemented as a custom System on Chip (SoC) designed / produced by Broadcom. https://www.broadcom.com/
- The design is documented at

  https://www.raspberrypi.org/documentation/hardware/raspberrypi/

- There are 4 Raspberry Pi variants:
  - Pi Zero
  - Pi One - BCM2835
  - Pi Two – BCM2836
  - Pi Three – BCM2837
- Only documentation of BCM2835 is generally available.



BROADCOM.

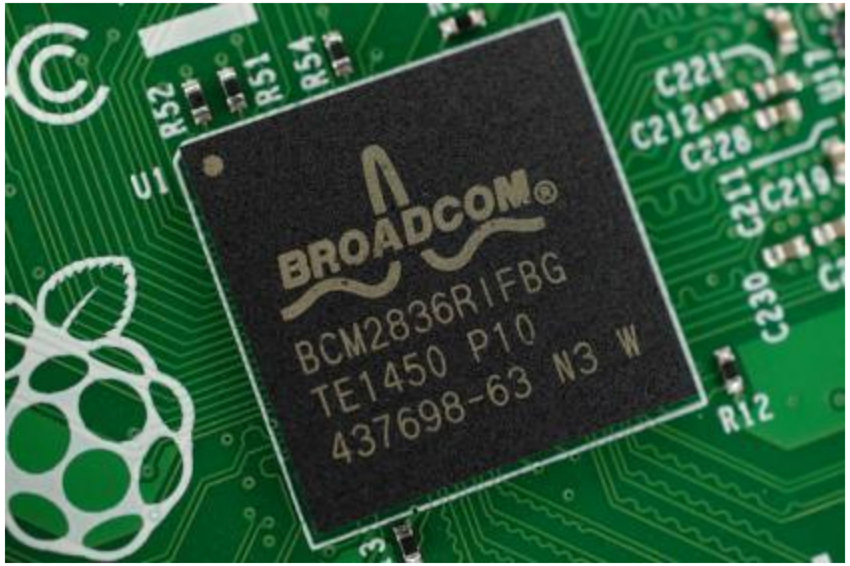
BCM2835 ARM Peripherals

© 2012 Broadcom Corporation.
All rights reserved

Broadcom Europe Ltd. 406 Science Park Milton Road Cambridge CB4 0WW

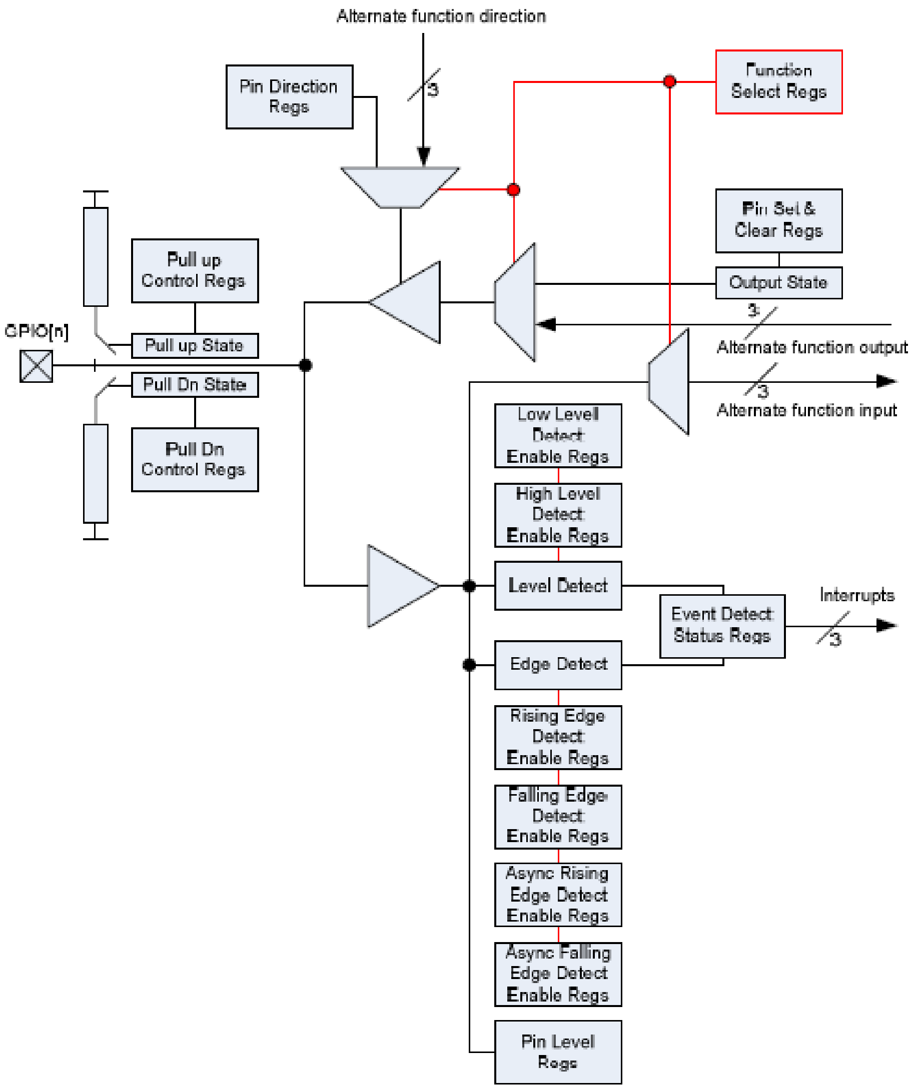
Raspberry Pi SoC

- BCM2835/6/7 contains the following peripherals which may safely be accessed by the Quad Core ARM Cortex-A53 Processor.
  - Timers
  - Interrupt controller
  - GPIO
  - USB
  - PCM / I2S
  - DMA controller
  - I2C master
  - I2C / SPI slave
  - SPI0, SPI1, SPI2
  - PWM
  - UART0, UART1

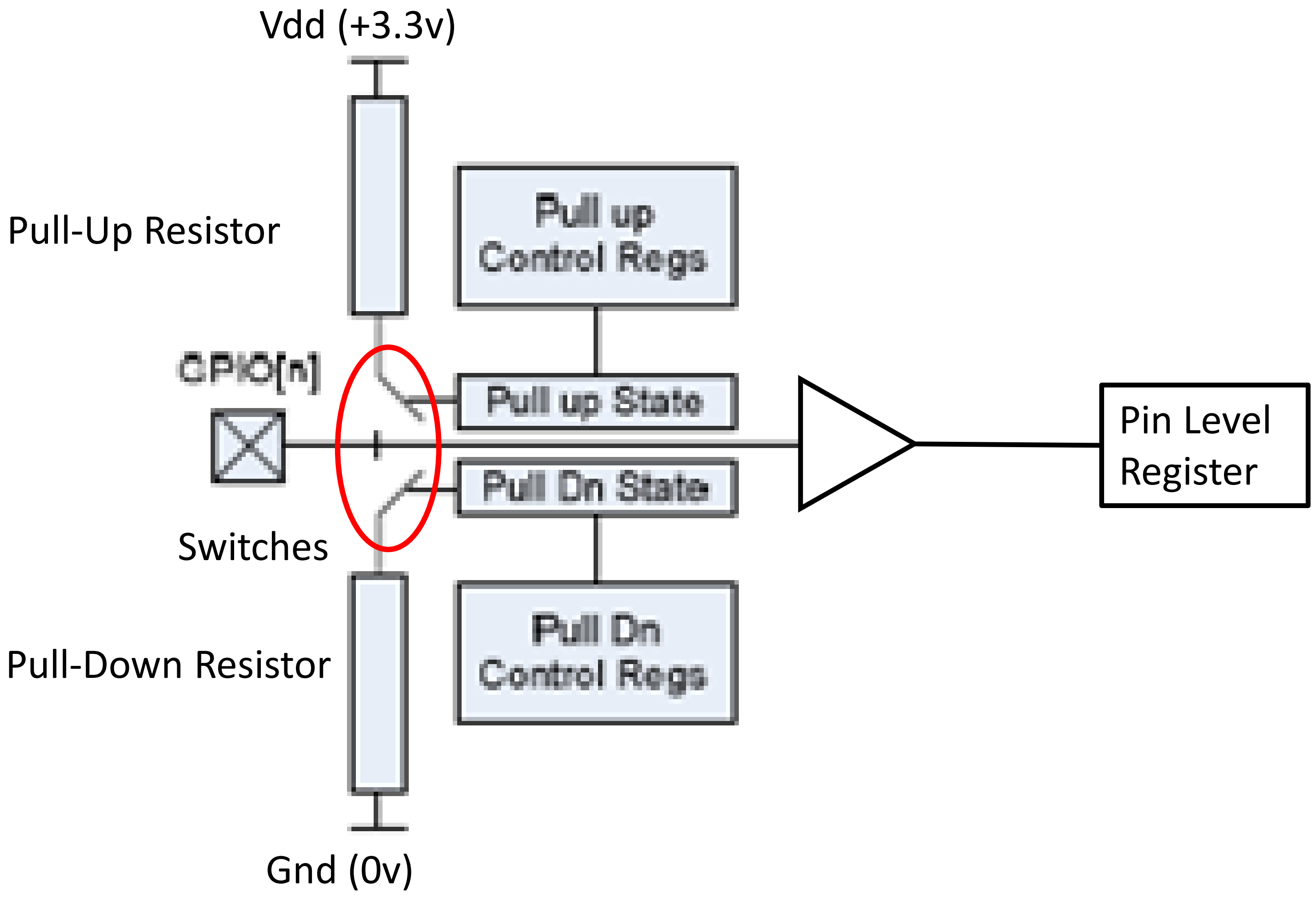


Raspberry Pi GPIO

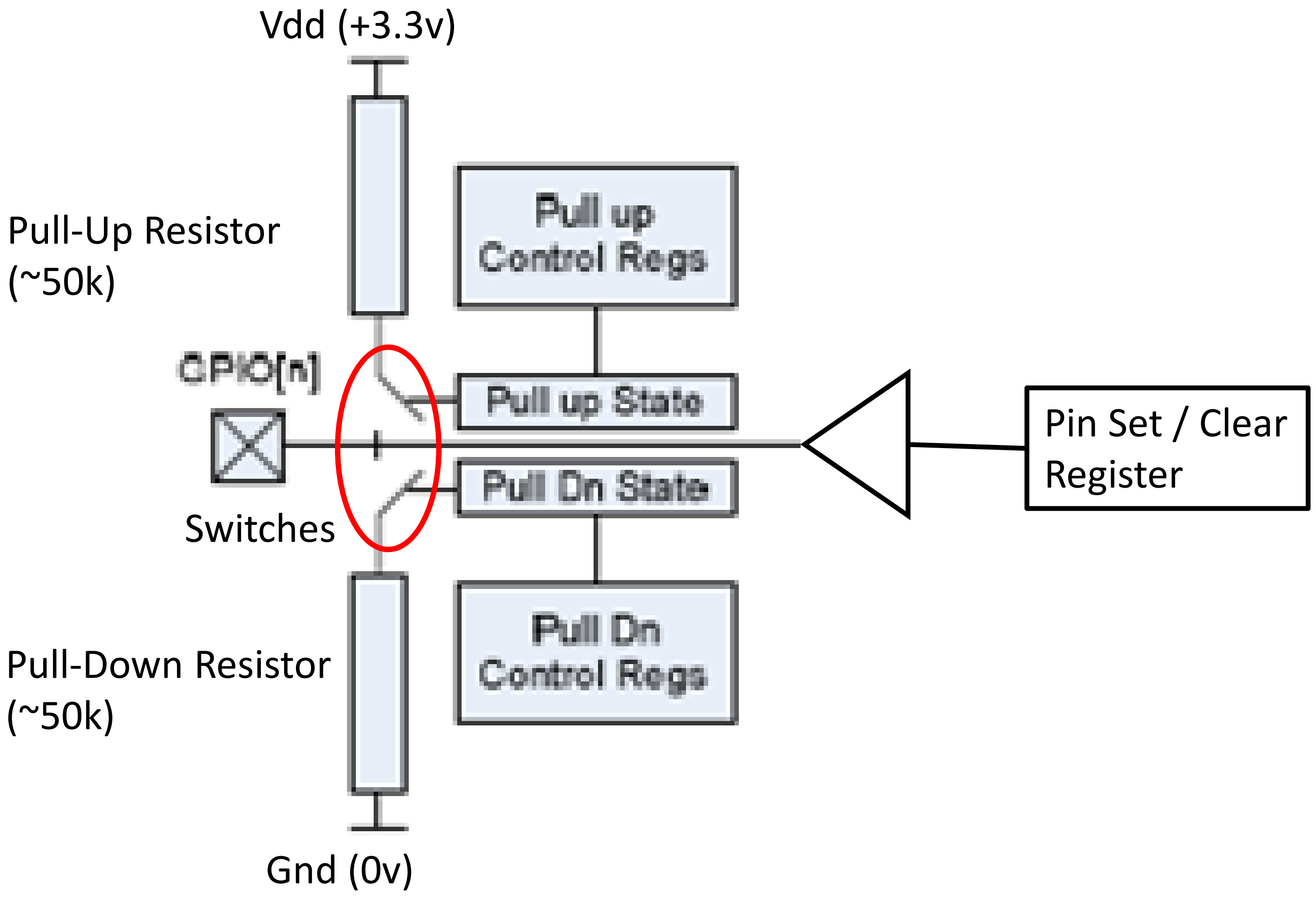
- Pins are expensive!
- Pins are multi-functional
- Shared between
  - GPIO
  - UART
  - SPI
  - I2C
  - PWM
- Configured by software
- Makes programming I/O at the lowest level very challenging.
- I/O Libraries manage the complexity



GPIO Input Data Path



GPIO Output Data Path

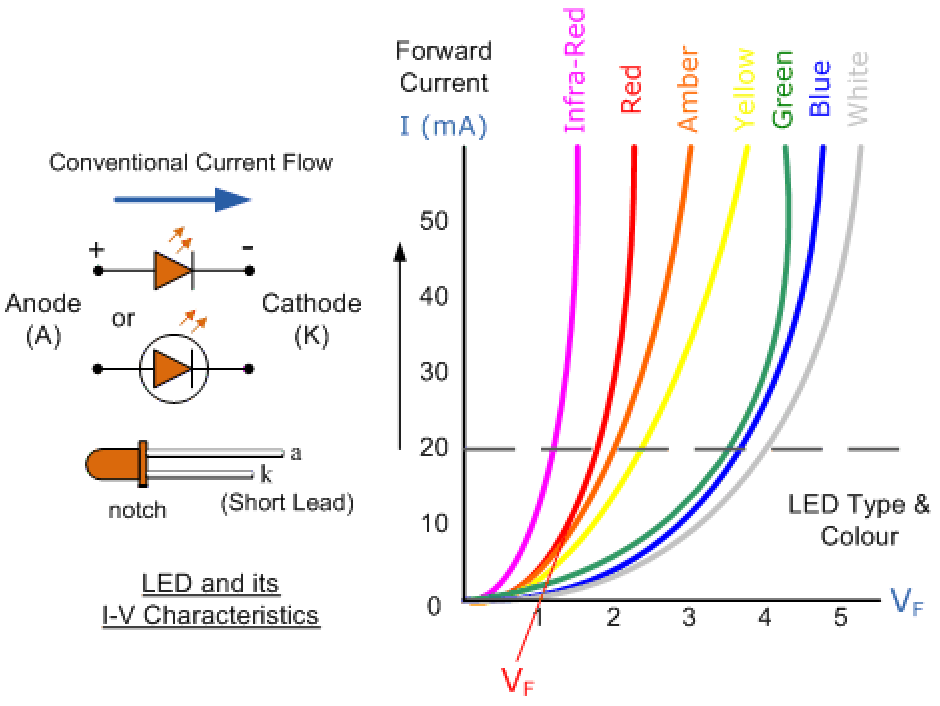


Pull-up / down control

- Essentially each of 54 I/O pins (channels) can be programmed
  - 00 = Off – disable pull-up/down
  - 01 – Enable Pull Down Control
  - 10 – Enable Pull Up Control
  - 11 – Reserved
- Note: The pull-up/down resistors are implemented as semiconductors so their resistance somewhere between 10k – 100k (typically ~50k).

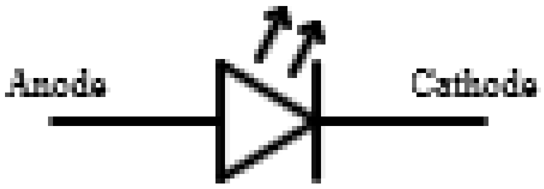
Light Emitting Diodes

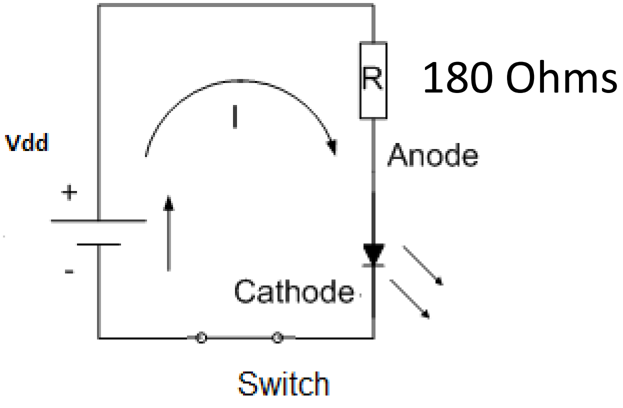
- The forward current to voltage characteristic of a Light Emitting Diode (LED) is shown.
- A LED is essentially a Diode.
- Diodes only pass current when the voltage applied to the Anode is +ve with respect to the Cathode



Connecting a LED to GPIO

- GPIO outputs can only supply (drive) a limited amount of current.
- The value of ‘drive’ for each pin is programmable.
  - https://www.scribd.com/doc/101830961/GPIO-Pads-Control2
- Never demand any output pin handles more than 16 mA.
- Minimum value of series Resistor



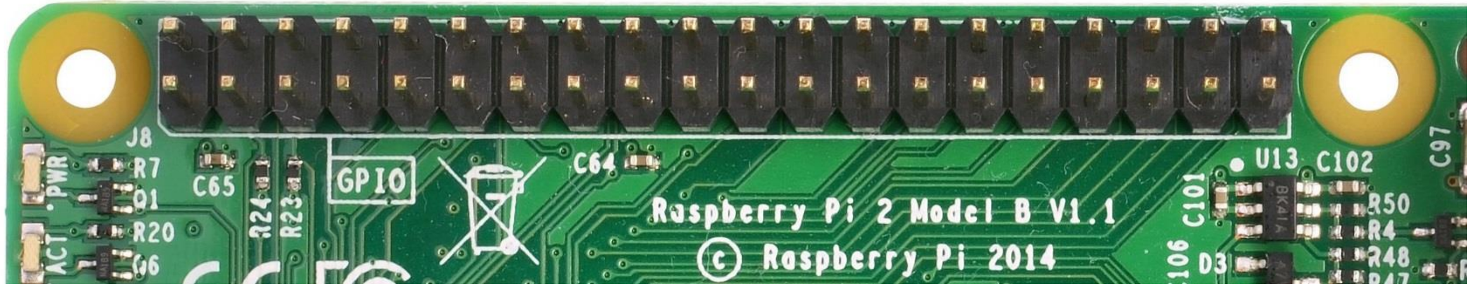


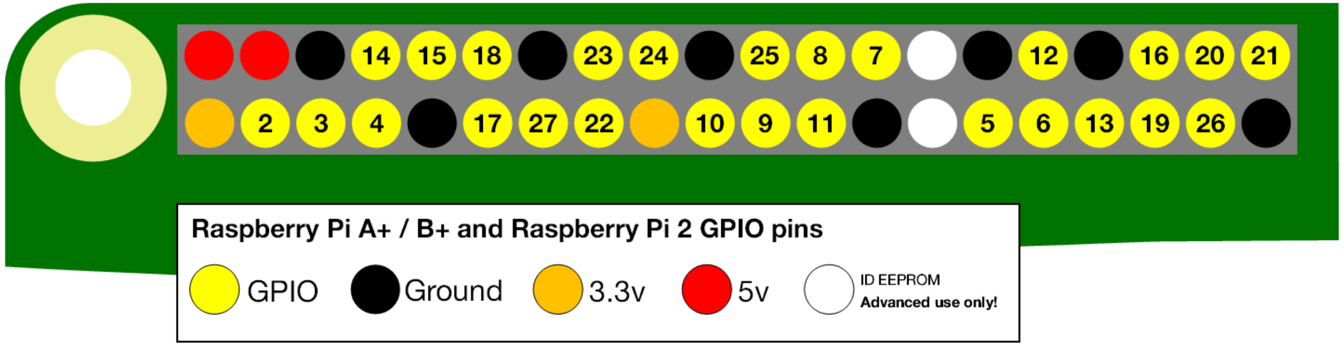
$$R_{min} = \frac{V}{I} = \frac{(3.3-0.7)V}{16\ mA}$$ = 162 Ohms

$$P = VI = (3.3 - 0.7) \times 0.016 = 41\ mW$$

- Introduction
- Digital Signals
- Simple Model of Parallel I/O
- Raspberry Pi SoC
- **Raspberry Pi GPIO**
- Serial Data Protocols
- Integrated Digital Temperature Sensor
- Software
- Summary

Raspberry Pi GPIO Connector

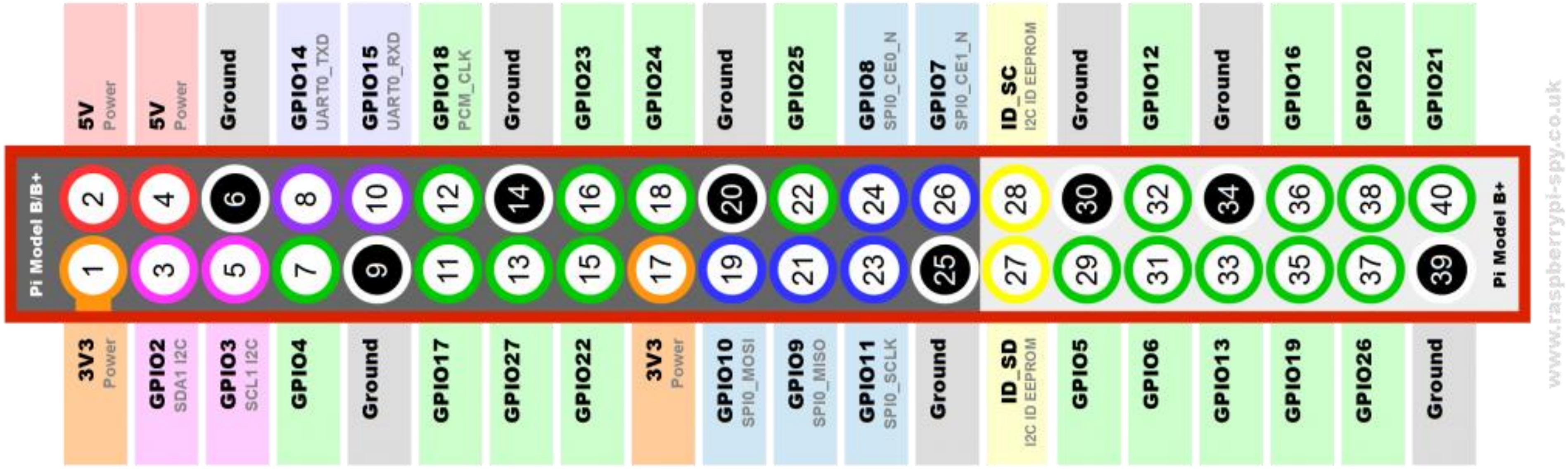




- Physical connection to GPIO is via a 40-pin connector.
- Note: Raspberry Pi 1 model A & B used a 26-pin connector.

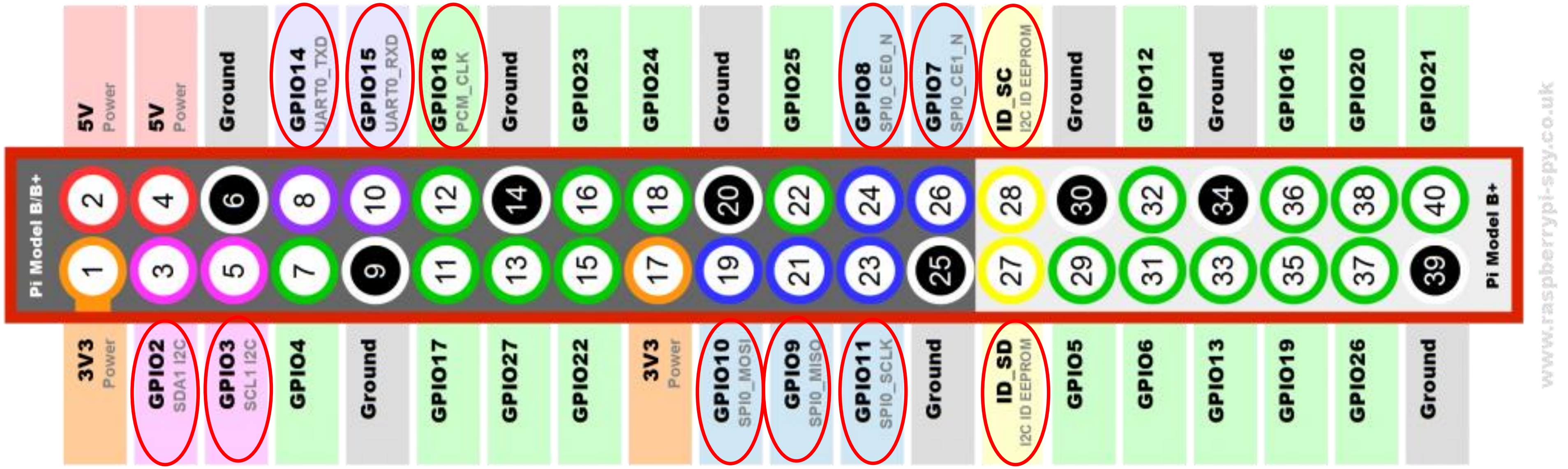
Pin Numbering

- Individual GPIO pins (‘channels’) are referenced using one of two conventions known as GPIO numbering and physical numbering.
  - GPIO numbering refers to the pin by its logical channel name (e.g. GPIO9)
  - Physical numbering refers to the channel by its pin ID (e.g. Pin 21)
- Before writing or reading a channel you must first select the pin numbering scheme being used.



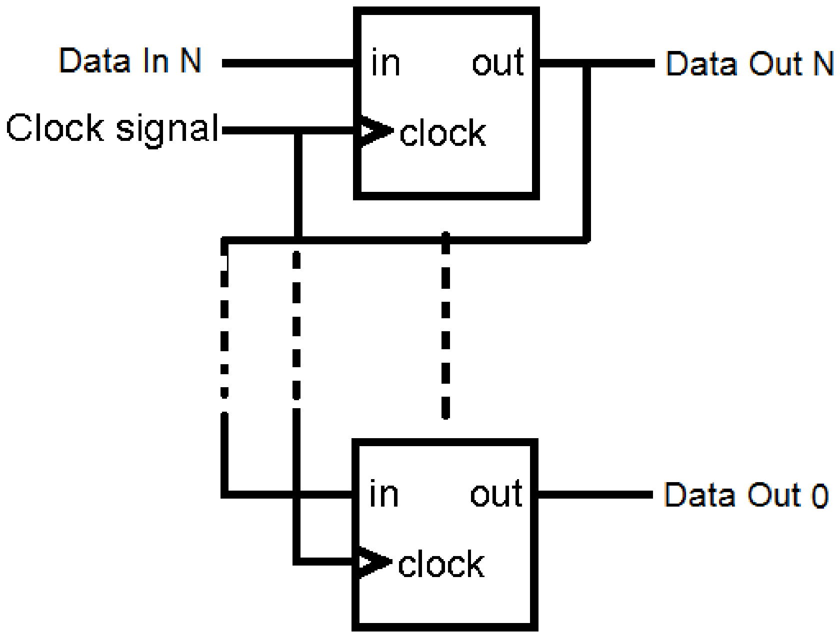
Alternate Function

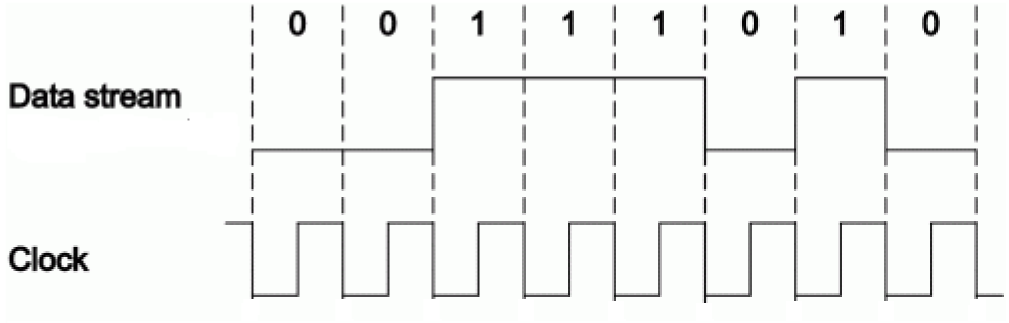
- To reduce the overall pin count (i.e. cost of fabricating the SoC) some GPIO pins are shared with other peripherals; many providing serial data interface.
  - UART
  - I2C
  - SPIO
- The big advantage of serial data transmission is it only requires 2 (or 3) signals.

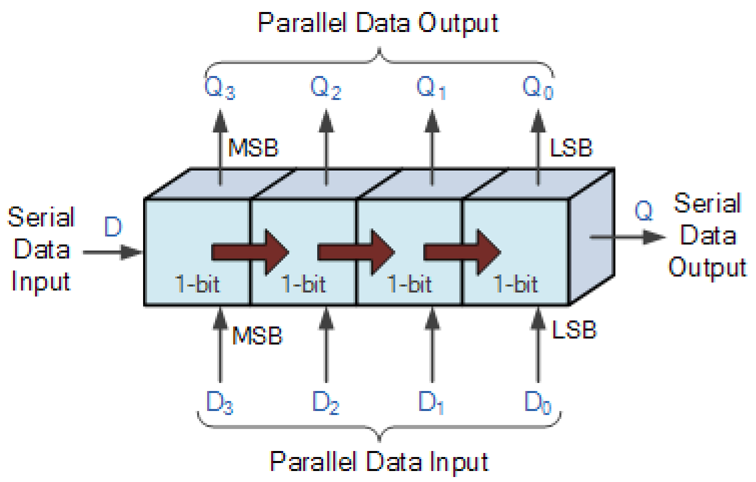


Serial In / Serial Out (SISO) Register

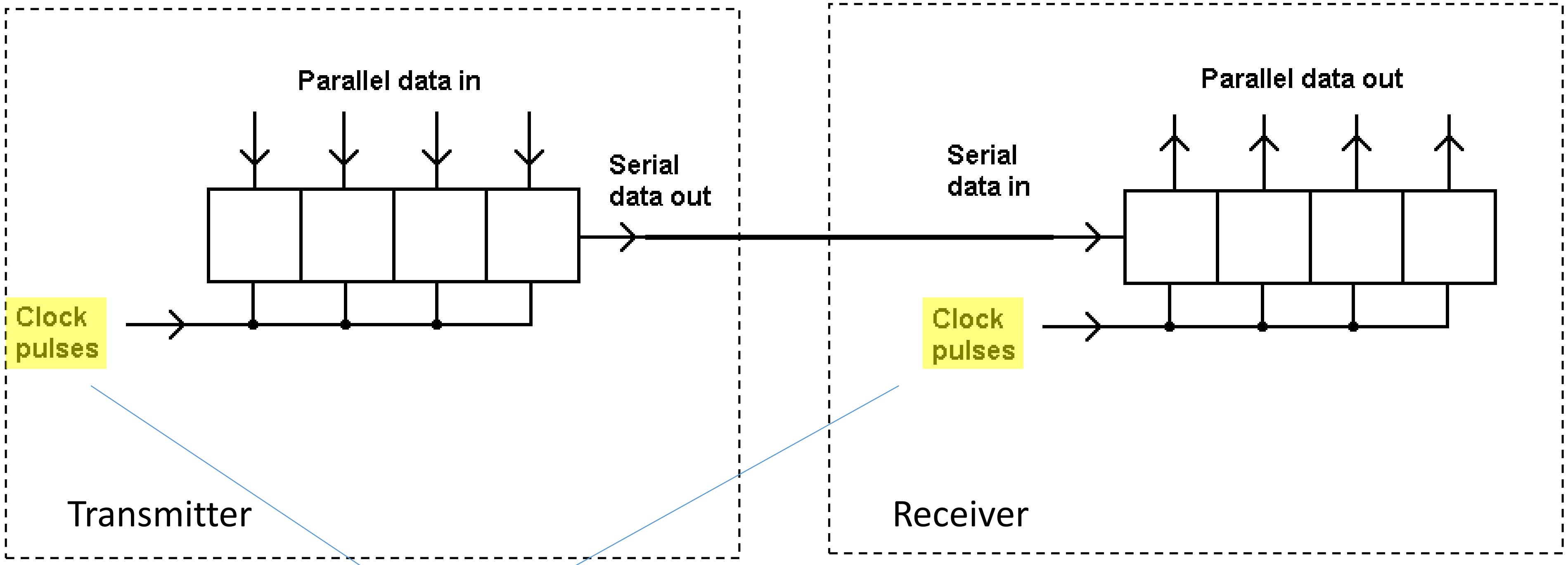
- Individual registers can be cascaded so multiple bits can be read/written serially.





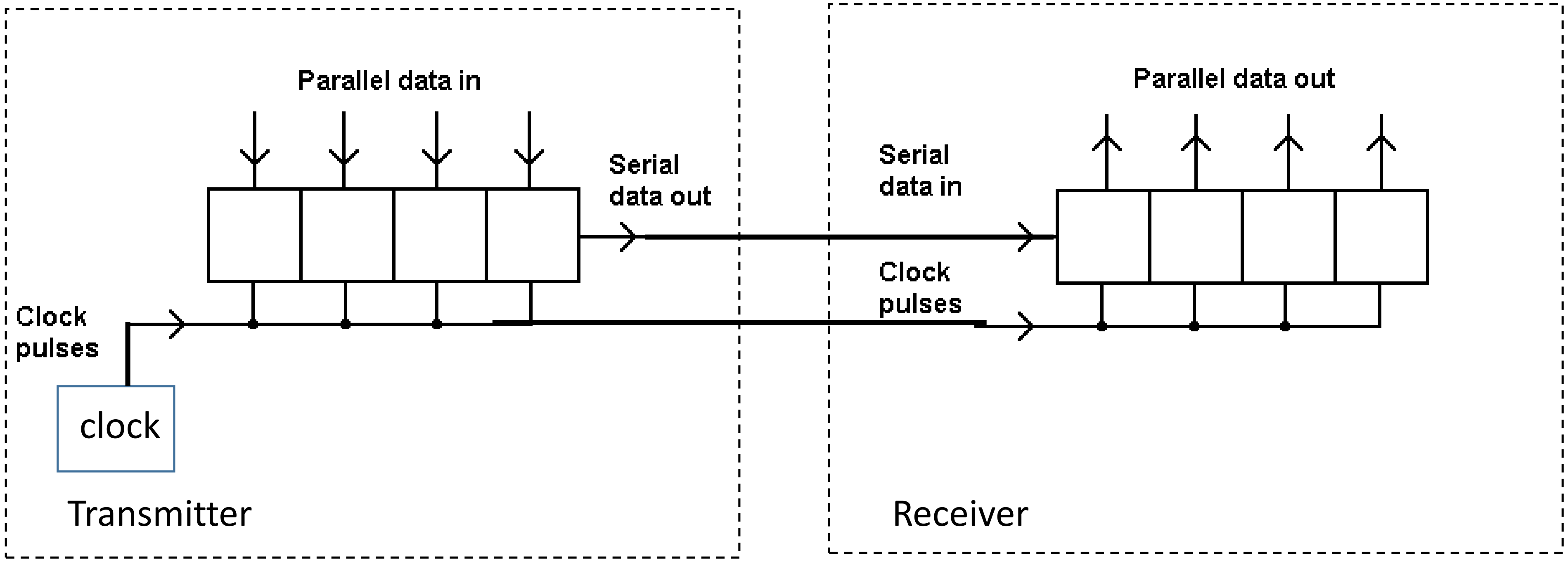


Basic Serial Port

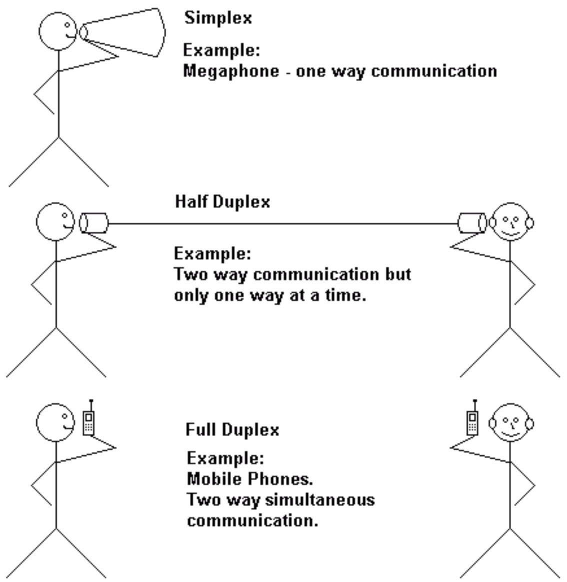


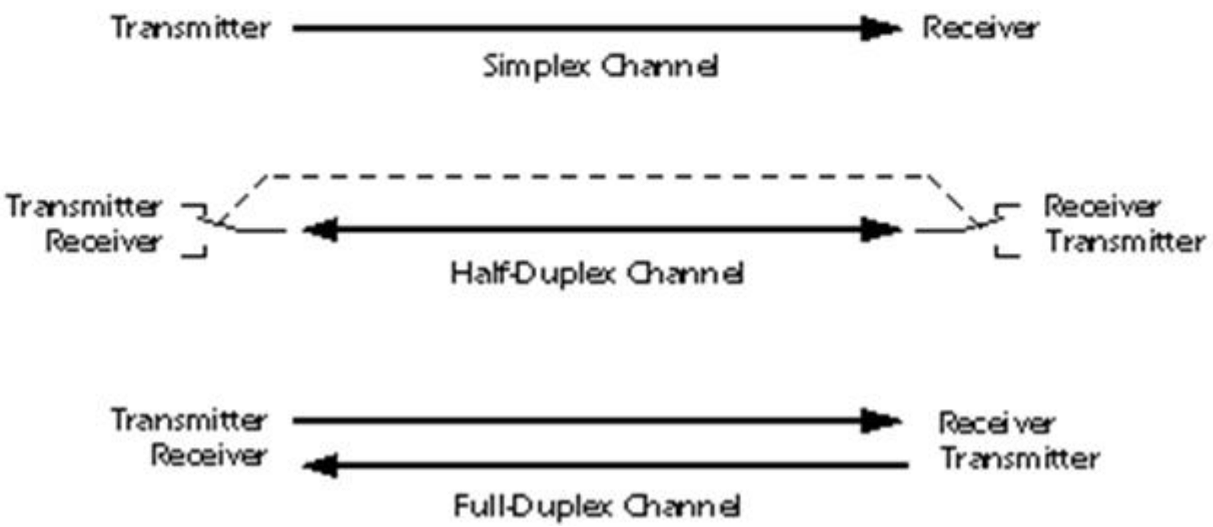
These must be synchronised (same frequency and phase) to avoid errors

Synchronous Operation



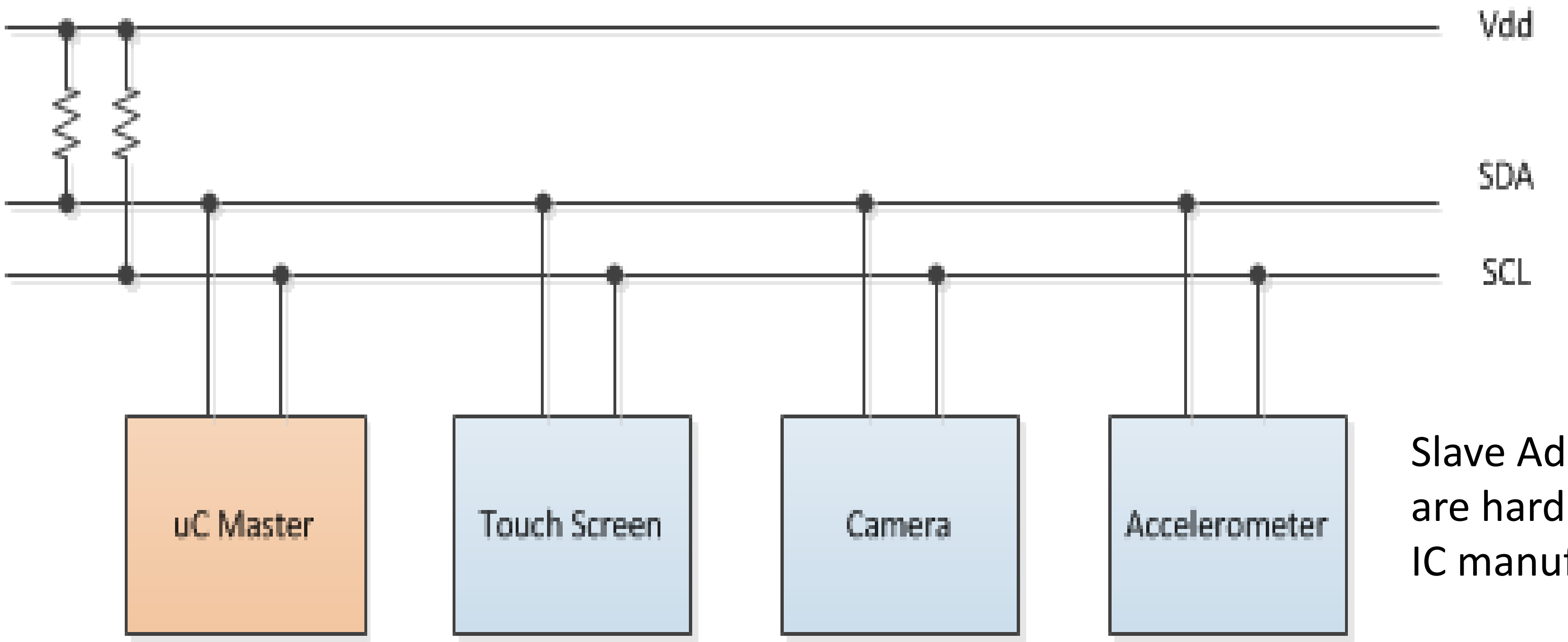
Channel Types





Inter-Integrated Circuit Bus (I^2C, I2C)

- I^2C (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, half-duplex, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor.
- The bus comprises clock (SCL) and serial data (SDA) lines



Slave Addresses are hard coded by IC manufacturers

I^{2}C Message Protocol

- I2C is a 2-wire interface designed to connect low speed peripherals.
- Bit rate is 100 kbit/s but since most transactions involve passing slave address actual data rate is about half this figure.
- Uses a 7-bit or 10-bit address space
- The master initially sends a start bit followed by the slave address, followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave.
- If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in its complementary mode (receive or transmit, respectively).

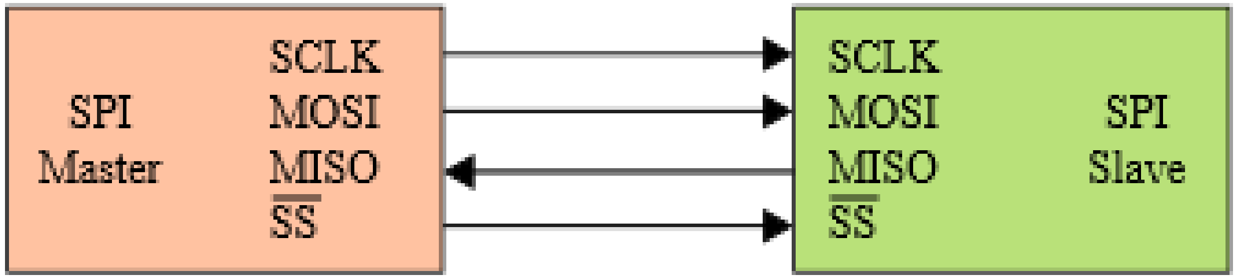
I^{2}C Message Protocol

- I^{2}C defines basic types of messages, each of which begins with a START and ends with a STOP:
  - Single message where a master writes data to a slave;
  - Single message where a master reads data from a slave;
  - Combined messages, where a master issues at least two reads and/or writes to one or more slaves.
- In a combined message, each read or write begins with a START and the slave address. After the first START, these are also called repeated START bits; repeated START bits are not preceded by STOP bits, which is how slaves know the next transfer is part of the same message.

Serial Peripheral Bus (SPI)

- SPI is a full-duplex, synchronous, serial bus developed by Motrola.
- SPI devices communicate using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

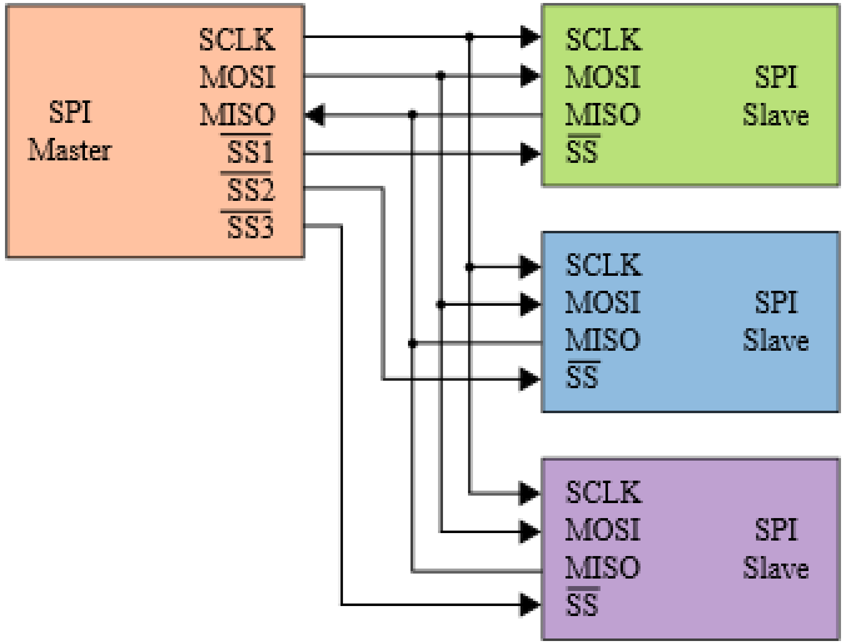
Shows a single master, single slave configuration



- Introduction
- Digital Signals
- Simple Model of Parallel I/O
- Raspberry Pi SoC
- Raspberry Pi GPIO
- **Serial Data Protocols**
- Integrated Digital Temperature Sensor
- Software
- Summary

Serial Peripheral Bus (SPI)

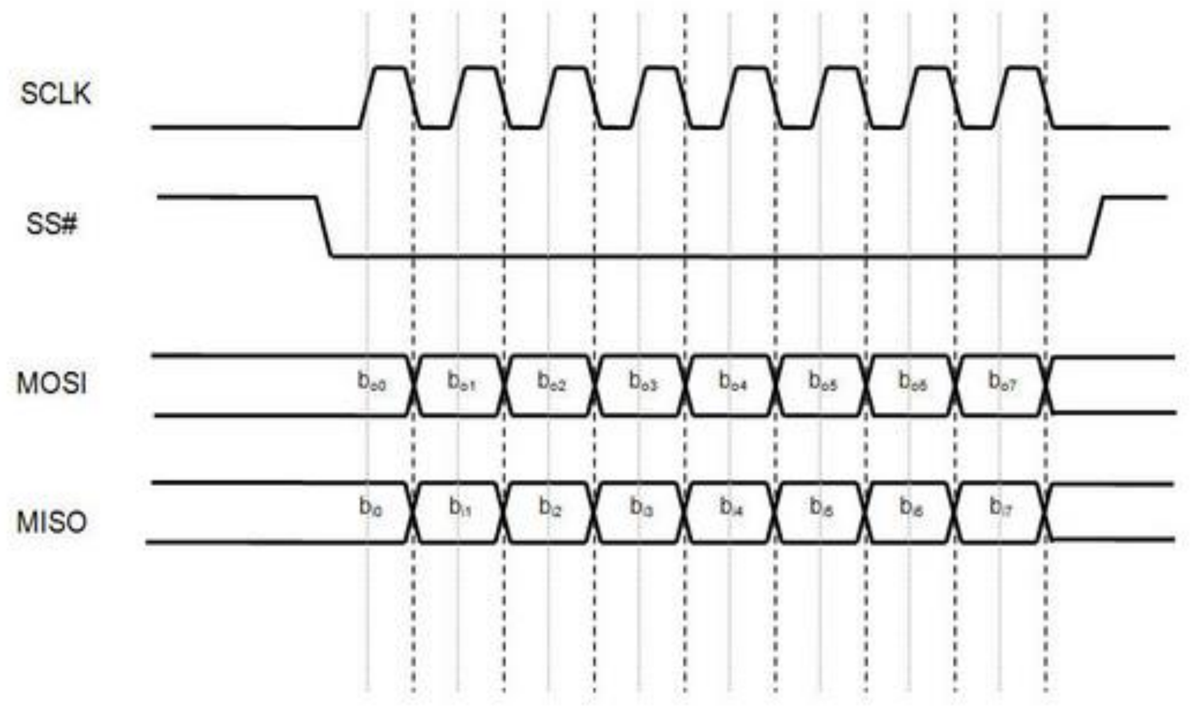
- Slave devices are not allocated addresses.
- Instead, multiple slave devices are supported by multiple Slave Select lines.



- Introduction
- Digital Signals
- Simple Model of Parallel I/O
- Raspberry Pi SoC
- Raspberry Pi GPIO
- **Serial Data Protocols**
- Integrated Digital Temperature Sensor
- Software
- Summary

SPI Signals

- SCLK : clock signal (all SPI signals are synchronised to this)
- SSLn : slave select signal (used to select slave that master communicates with)
- MOSI : Master Out Slave In (data from master to slave)
- MISO : Master In Slave Out (data from Slave to master)



SPI Communication (Timing Diagram)

- Introduction
- Digital Signals
- Simple Model of Parallel I/O
- Raspberry Pi SoC
- Raspberry Pi GPIO
- Serial Data Transmission
- **Integrated Digital Temperature Sensor**
- Software
- Summary

DS18B20 Temperature Sensor

- Some sensor manufacturers develop their own serial interface protocols.
- E.g. DS18B20 Temperature Sensor (www.maximintegrated.com) is a digital thermometer that uses Maxim's exclusive 1-Wire bus protocol.
- The temperature senor combines a analogue sensor with an Analogue-to-Digital Converter (ADC).

http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

DS18B20 Temperature Sensor

- The temperature senor resolution (9-12-bit) is configured by software.
- Temperature data is stored as a 16-bit sign-extended two's complement number

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	**BIT 14**	**BIT 13**	**BIT 12**	**BIT 11**	**BIT 10**	**BIT 9**	**BIT 8**
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

DS18B20 Temperature Sensor

- The Temperature / Data relationship is shown below (assumes 12-bit resolution)

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

**The power-on reset value of the temperature register is +85°C.*

Software Libraries

- Software libraries make using GPIO, SPIO, I2C etc. much simpler.
- Need to select a library with functions that can easily integrated within programs written in a target language.
- Bare GPIO and Serial Bus Standards are generally supported by the Open Source community.
- Manufacturers using bespoke protocols provide their own libraries – in many cases these are ported to Raspberry Pi by the user community.

Raspberry Pi GPIO Library

- A number of libraries exist for interfacing to GPIO
  - C / C++ - http://www.airspayce.com/mikem/bcm2835/
  - Java - http://pi4j.com/
  - Python - https://sourceforge.net/projects/raspberry-gpio-python/
    https://pythonhosted.org/RPIO/
- Raspbian ships with RPi.GPIO Python library but you might wish to update to the latest version
  http://www.raspberrypi-spy.co.uk/2012/05/install-rpi-gpio-python-library/
- $ find /usr | grep –i gpio will recursively search /usr directory for files containing the string “gpio”.
  Look for lines containing 0.6.3.egg-info (0.6.3 is the version).
- Consult https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

Summary

- The GPIO peripheral is complicated because manufacturers are trying to squeeze as much functionality as possible onto a few device pins (pins are expensive).
- Many devices use a serial data bus to save pins.
- To successfully use the GPIO we need to use a library.