Low Level Protocols

Embedded Interface Design

with **Bruce Montgomery**

Learning Objectives

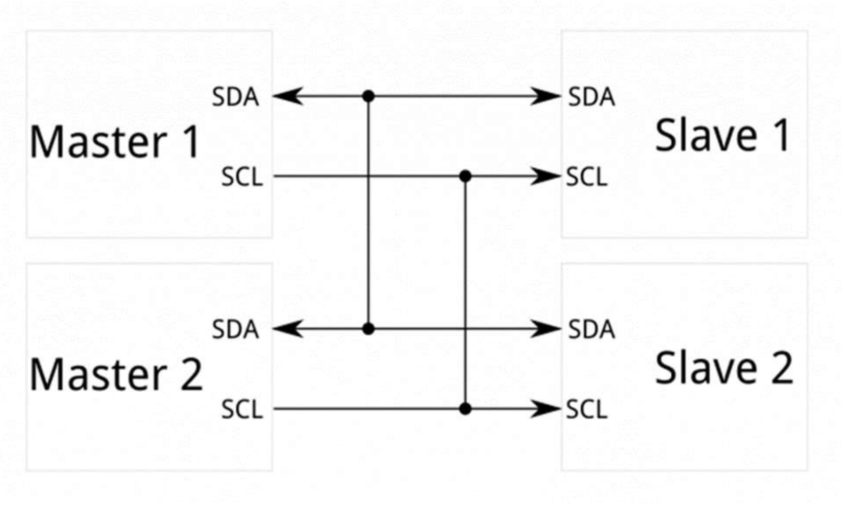
- Students will be able to…
  - Recognize common low level protocols and interfaces common to SBCs and Microcontrollers
  - Compare and contrast their applications

Low Level Protocols

- Short distance communications interfaces within (or adjacent to) a single device
- Examples
    - I2C – Inter-Integrated Circuit
    - SPI - Serial Peripheral Interface
    - UART – Universal Asynchronous Receiver/Transmitter
    - 1-Wire
    - GPIO – General Purpose Input/Output
- The ones we review at some detail are common to SBCs like the Pi, there are many others

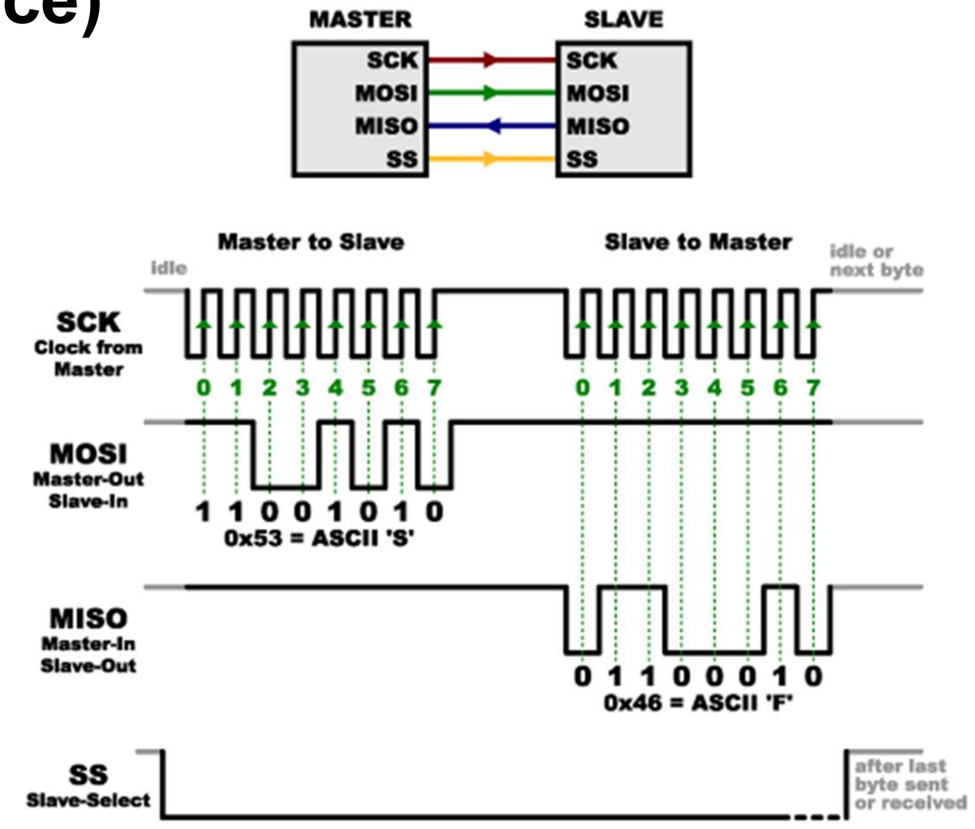
I2C (Inter-Integrated Circuit)

- Protocol introduced by Philips Semiconductor in 1982
- Lower speed devices
- Standard mode: 100 kbit/s
- Full speed: 400 kbit/s
- Fast mode: 1 Mbit/s
- High speed: 3.2 Mbit/s
- Master/Slave half-duplex communication
- Slaves have unique address bits
- 112 devices addressable with 7 bit addresses
- 1008 devices with 10 bit addresses
- Two data lines – serial clock (SCL) and serial data (SDA)
- **Best application: Intermittently accessed devices – Example: an RFID reader**
- Reference [1]



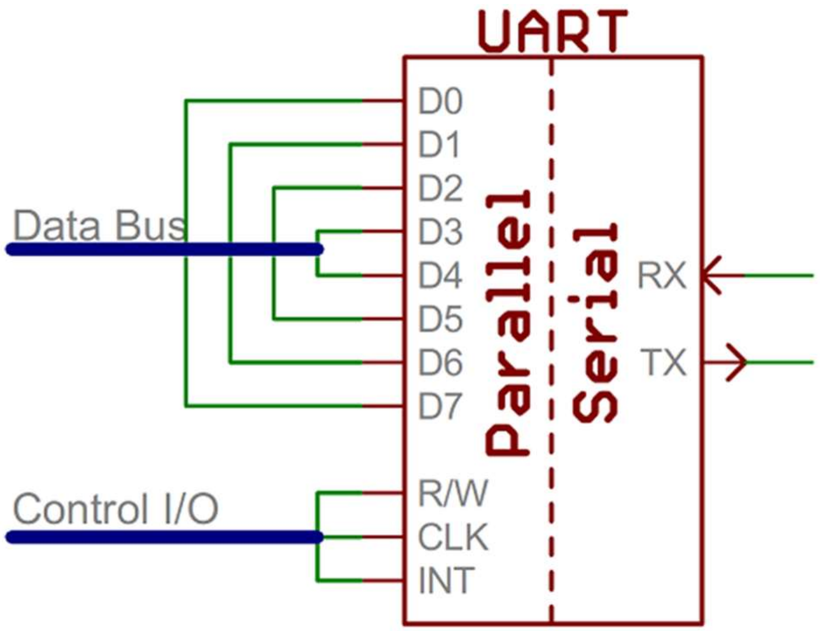
SPI (Serial Peripheral Interface)

- Protocol introduced by Motorola in late 1980s
- Full duplex serial communications
- 4 line protocol – Clock from master, Master Out Slave In, Master In Slave Out, Slave Select
- Higher throughput (and usually lower power) than I²C
- Slaves do not need a unique address
- Not limited to a maximum clock speed – in practice varies from 12 to 100 MHz
- **Best for devices that provide data streams – Examples: sensors, SD cards**
- Reference [2]

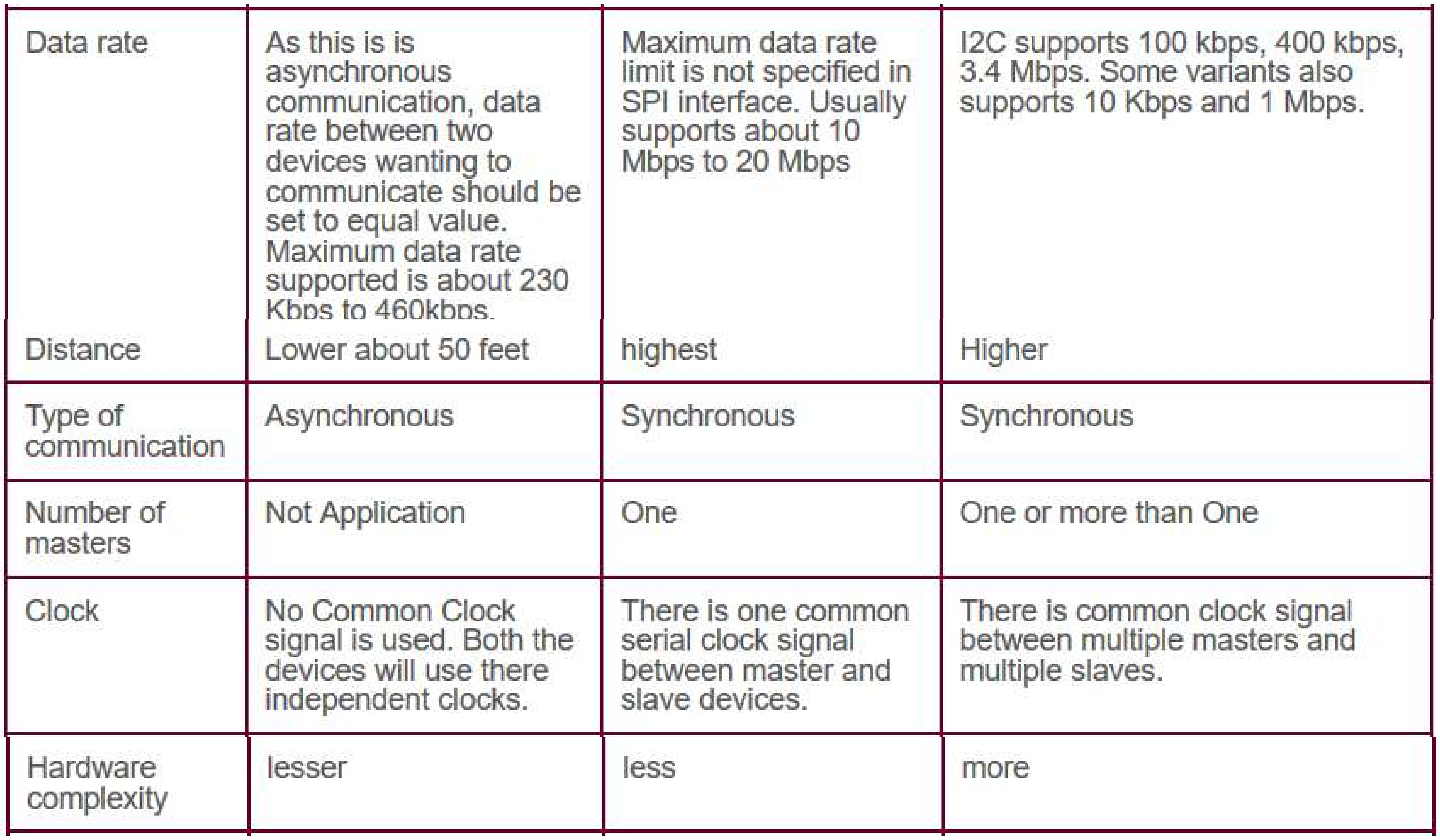


UART (Universal Asynchronous Receiver/Transmitter)

- First UART was designed by Gordon Bell of DEC in 1960s
- UARTs implement asynchronous serial communication from parallel data (sender does not have to send a clock signal)
- Usually implemented internally in microcontrollers
- Often used for RS-232 or other standard serial communication, usually with buffering FIFO queues to speed communications
- On the Pi, Raspian uses the UART for the serial console, but this can be used for your own applications
- Good interface for interconnecting to local serial devices (vs. on-board distances for I2C or SPI)
- Reference [3]



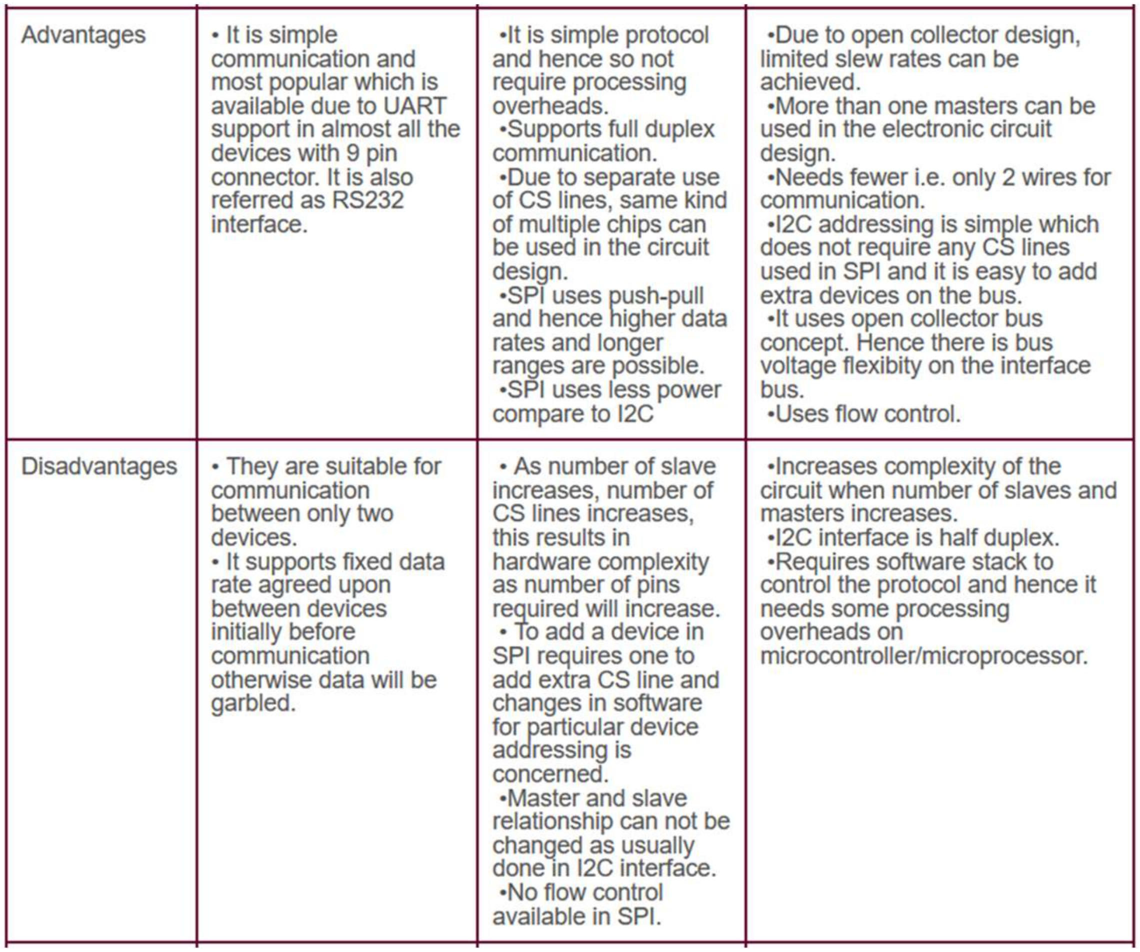
UART vs SPI vs I2C



Data rate	As this is is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.
Distance	Lower about 50 feet	highest	Higher
Type of communication	Asynchronous	Synchronous	Synchronous
Number of masters	Not Application	One	One or more than One
Clock	No Common Clock signal is used. Both the devices will use there independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal between multiple masters and multiple slaves.
Hardware complexity	lesser	less	more

Reference [4]

UART vs SPI vs I2C

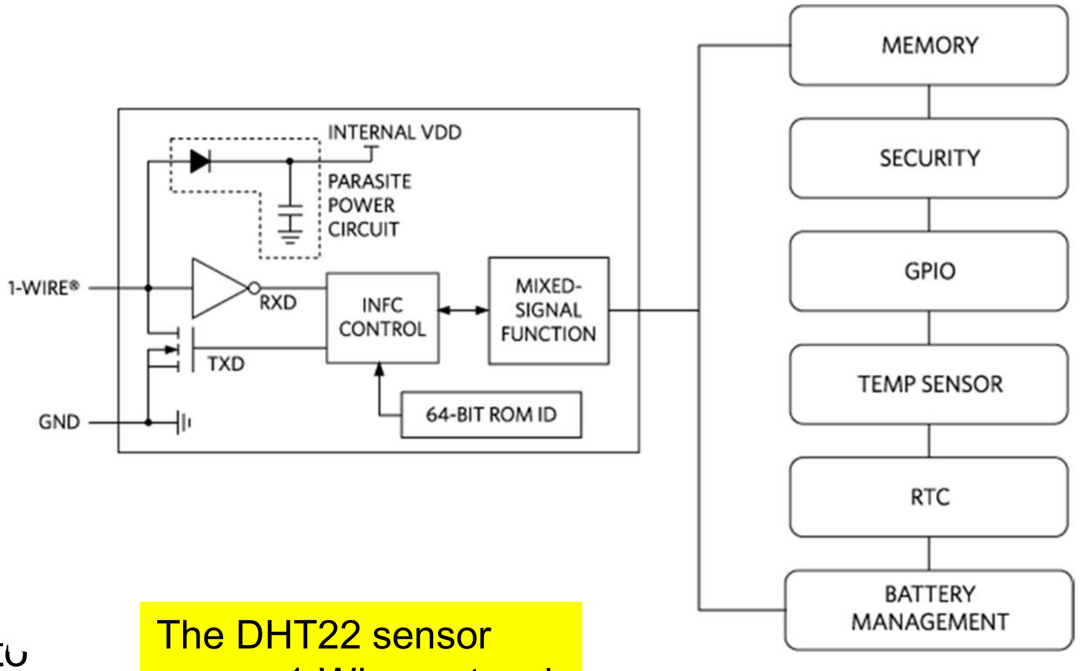


Advantages	• It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface.	•It is simple protocol and hence so not require processing overheads. •Supports full duplex communication. •Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. •SPI uses push-pull and hence higher data rates and longer ranges are possible. •SPI uses less power compare to I2C	•Due to open collector design, limited slew rates can be achieved. •More than one masters can be used in the electronic circuit design. •Needs fewer i.e. only 2 wires for communication. •I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. •It uses open collector bus concept. Hence there is bus voltage flexibity on the interface bus. •Uses flow control.
Disadvantages	• They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.	• As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. •Master and slave relationship can not be changed as usually done in I2C interface. •No flow control available in SPI.	•Increases complexity of the circuit when number of slaves and masters increases. •I2C interface is half duplex. •Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor.

Reference [4]

1-Wire

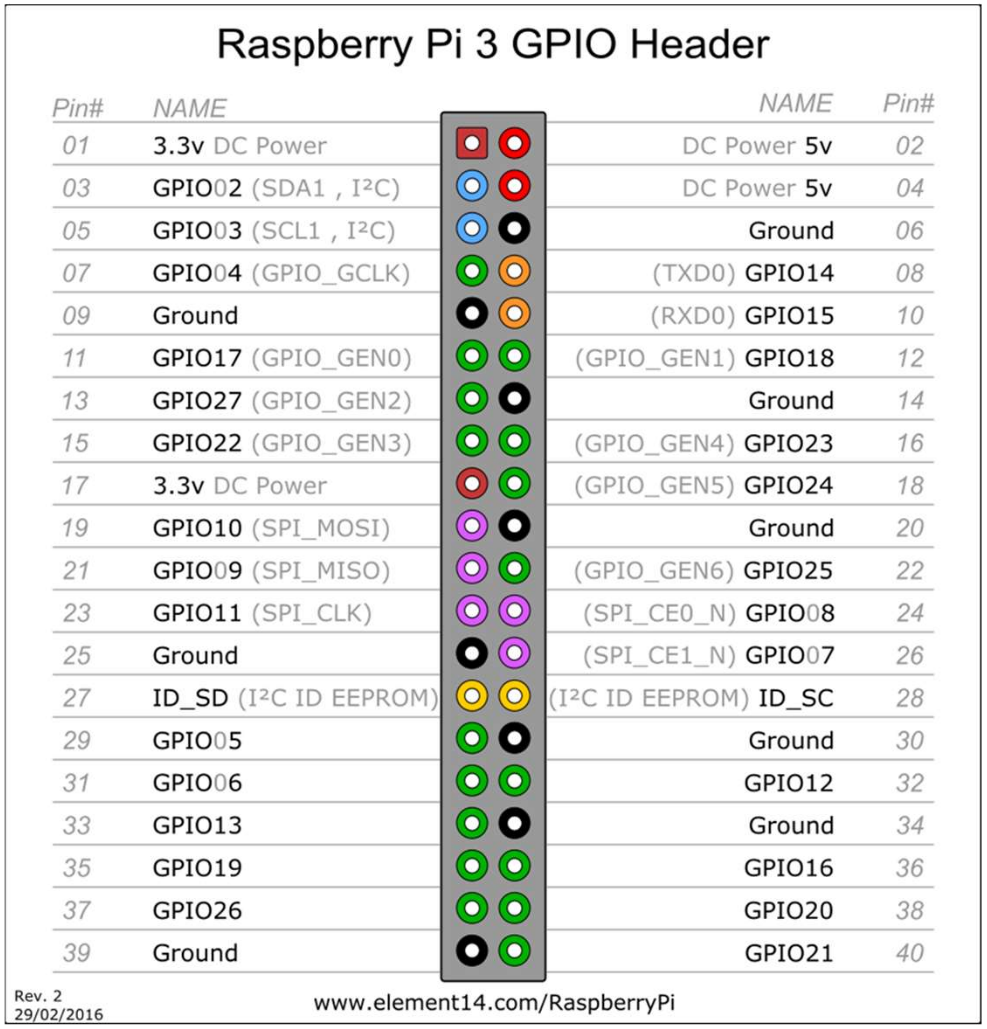
- Protocol introduced by Dallas semiconductors (now Maxim)
- Interfacing to 1-Wire devices on a RPi3 can use a single GPIO pin for data, and often provide a unique serial code to identify the device
- Two contacts, data and ground, provide for half-duplex bidirectional communication
- **Designed for contact applications**
- Disconnecting from the 1-Wire bus (or a loss of contact) puts the 1-Wire slaves intu a defined reset state
- When the voltage returns, the slaves wake up and signal their presence



The DHT22 sensor uses a 1-Wire protocol, but not the one by Dallas/Maxim

GPIO (General Purpose Input/Output)

- GPIO interfaces represent generic pins that can be controlled at run time
- GPIO pins can be configured:
  - as input or output
  - as enabled or disabled
  - as interrupts
- On the RPi3 for example, the GPIO header provides for I2C, SPI, and 1-Wire interfaces



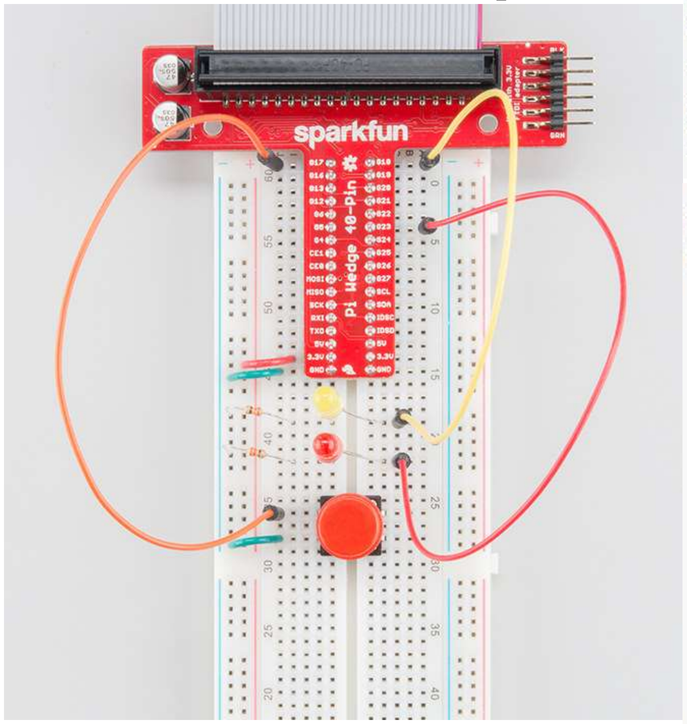
Raspberry Pi 3 GPIO Header

Pin#	NAME			NAME	Pin#
01	3.3v DC Power	🟥	🔴	DC Power 5v	02
03	GPIO02 (SDA1 , I^2C)	🔵	🔴	DC Power 5v	04
05	GPIO03 (SCL1 , I^2C)	🔵	⚫	Ground	06
07	GPIO04 (GPIO_GCLK)	🟢	🟠	(TXD0) GPIO14	08
09	Ground	⚫	🟠	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	🟢	🟢	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	🟢	⚫	Ground	14
15	GPIO22 (GPIO_GEN3)	🟢	🟢	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	🔴	🟢	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	🟣	⚫	Ground	20
21	GPIO09 (SPI_MISO)	🟣	🟢	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	🟣	🟣	(SPI_CE0_N) GPIO08	24
25	Ground	⚫	🟣	(SPI_CE1_N) GPIO07	26
27	ID_SD (I^2C ID EEPROM)	🟡	🟡	(I^2C ID EEPROM) ID_SC	28
29	GPIO05	🟢	⚫	Ground	30
31	GPIO06	🟢	🟢	GPIO12	32
33	GPIO13	🟢	⚫	Ground	34
35	GPIO19	🟢	🟢	GPIO16	36
37	GPIO26	🟢	🟢	GPIO20	38
39	Ground	⚫	🟢	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

GPIO Example



Reference [7]

```
import RPi.GPIO as GPIO
```

```
# Pin Setup:
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
GPIO.setup(pwmPin, GPIO.OUT) # PWM pin set as output
pwm = GPIO.PWM(pwmPin, 50)  # Initialize PWM on pwmPin 100Hz frequency
GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Button pin set as input w/ pull-up
```

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

Other Common Board/Chip-Level Protocols

- ADC/DAC Analog-Digital Converter – often used for digital microprocessor interfaces to analog devices [8]
- One oft-sited difference between Arduinos and Pis, the Pi must have an ADC/DAC add-on board for such interfaces
- JTAG – Test/verification interface and protocol [9]
- PCI, PCI-X, AGP, PCI Express – for video, sound, network, controller specific interfaces [10]
- AXI Advanced eXtensible Interface, APB Advanced Peripheral Bus, and AHB Advanced High-performance Bus – connects blocks on SoC (System on Chip) chip-level designs [11,12]
- More on these in other ESE classes

References

[1] https://learn.sparkfun.com/tutorials/i2c
[2] https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi
[3] https://learn.sparkfun.com/tutorials/serial-communication
[4] http://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html
[5] https://www.maximintegrated.com/en/app-notes/index.mvp/id/1796
[6] https://learn.sparkfun.com/tutorials/raspberry-gpio
[7] https://learn.sparkfun.com/tutorials/raspberry-gpio#python-rpigpio-example
[8] https://www.digikey.com/en/articles/techzone/2017/sep/adc-dac-tutorial
[9] https://www.xjtag.com/about-jtag/jtag-a-technical-overview/
[10] https://www.lifewire.com/pci-express-pcie-2625962
[11] http://www.iosrjournals.org/iosr-jece/papers/Vol8-Issue5/A0850109.pdf
[12] https://www.design-reuse.com/articles/24123/amba-ahb-to-axi-bus-comparison.html