ECE 442/510
Internet of Things and Cyber Physical Systems

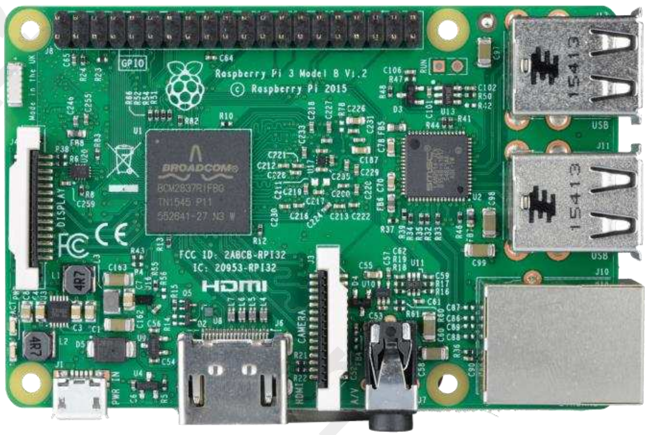
Lecture 4: Design with Raspberry Pi and Lab 2

Summer 2022

Jafar Saniie & Won-Jae Yi

What is a Raspberry Pi?

- Cost-effective microcomputer with compatibility of flexible OS
- Low cost device targeted for students, hobbyists, computer enthusiasts to learn and obtain programming skills and hardware understandings, further develop system for DIY projects
- Various operating system choices
  - Standalone Linux: Raspbian, CentOS, Fedora, Ubuntu…
  - RISC OS, Arch Linux ARM…
  - Windows 10 IoT Core, Android Things…
- Open hardware with exception of its SoC
  - Broadcom SoC including ARM processor
  - RAM (128MB to 1GB, depends on versions)
  - Networking (Ethernet, Bluetooth, Wi-Fi)
  - Peripherals (USB, GPIO pinouts, Audio)
  - Video (HDMI port)



Short History

- Around '05, Eben Upton was Director of Studies in Computer Science at University of Cambridge
- Incoming students had relatively few programming and/or hardware skills
  - Compared to "the old days", creating vision of "something like the BBC Computer", but running a modern language like Python
  - "Raspberry Pi" is a combination of "a fruit name" and a play on "Python"
- Between '06-'11, the vision turned into a high capable single board computer design
- Getting past the idea that

  "Python is enough"

Various Types of RPis

Raspberry Pi 3 Model B+

The final revision of our third-generation single-board computer

More info

Raspberry Pi 3 Model B

Our third-generation single-board computer

More info

Raspberry Pi 1 Model B+

The Model B+ is the final revision of the original Raspberry Pi

More info

Raspberry Pi 1 Model A+

The Model A+ is the low-cost variant of the Raspberry Pi

More info

Raspberry Pi Zero W

Single-board computer with wireless and Bluetooth connectivity

More info

Raspberry Pi Zero

Our lowest-cost single-board computer

More info

Raspberry Pi Zero 2 W

Your tiny, tiny $15 computer

More info

Raspberry Pi 400 Personal Computer Kit

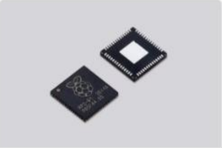
Raspberry Pi 400 is a complete personal computer, built into a compact keyboard.

More info

Raspberry Pi Pico

The new, flexible $4 microcontroller board from Raspberry Pi

More info



RP2040

A microcontroller chip designed by Raspberry Pi

More info

Raspberry Pi 400 unit

Raspberry Pi 400 is your complete personal computer, built into a compact keyboard

More info

Raspberry Pi 4 Desktop Kit

Full desktop computer kit - just connect to HDMI display(s)

More info

Raspberry Pi 4 Model B

Your tiny, dual-display, desktop computer

More info

Raspberry Pi 3 Model A+

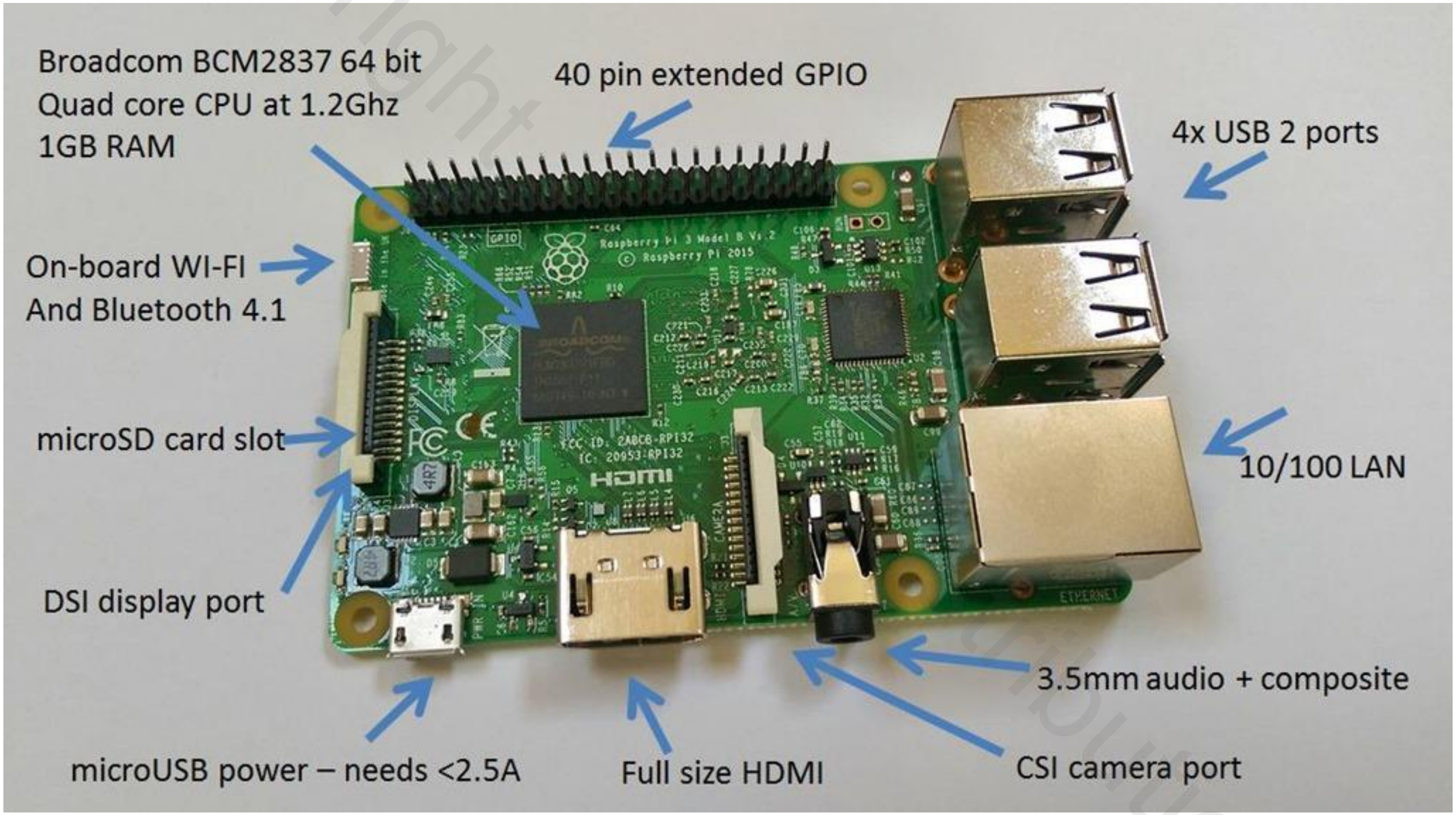
Our third-generation single-board computer, now in the A+ format

More info

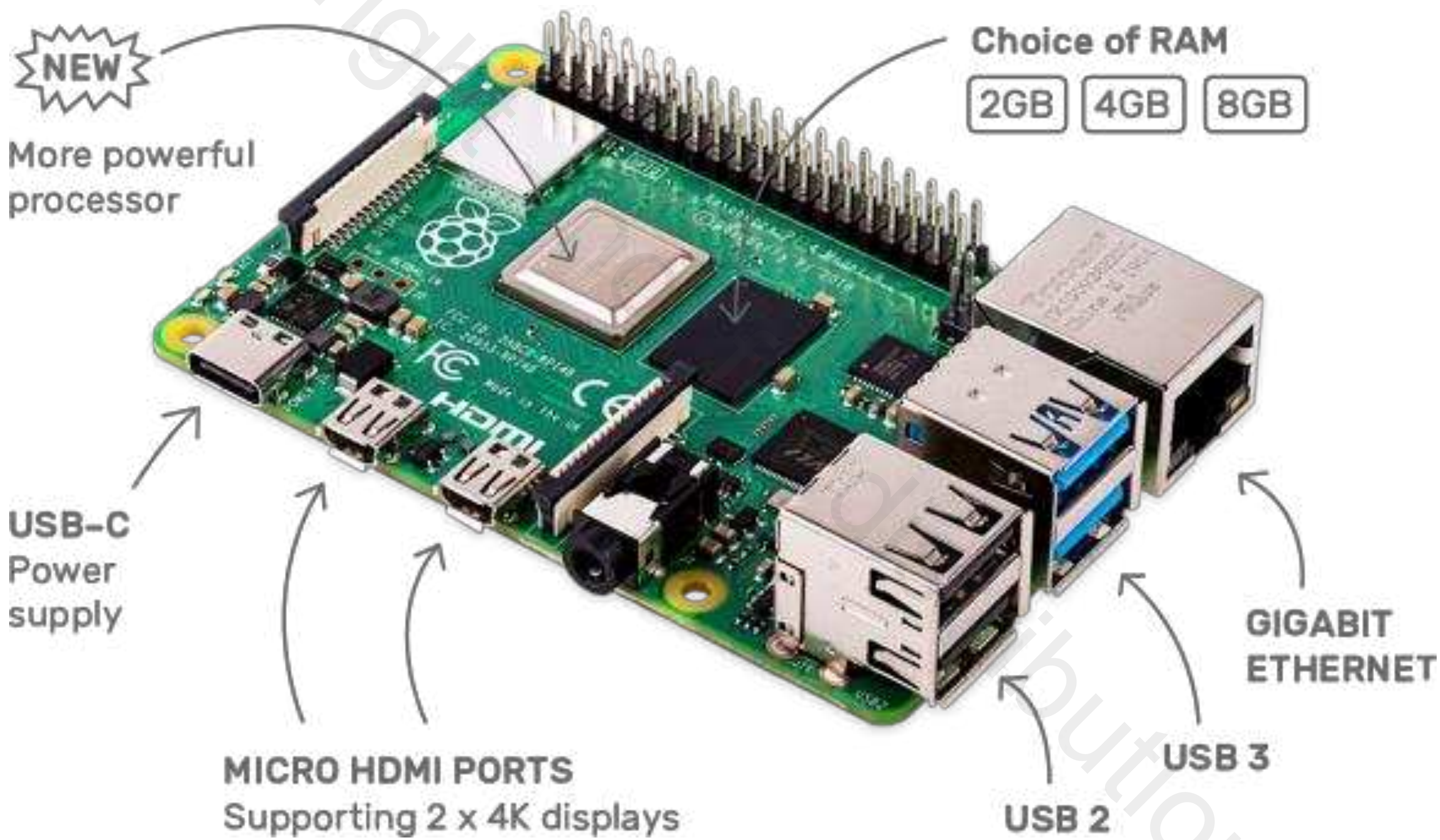
Various Types of RPis

- Pi 1 Model A+: low-cost variant, 256MB RAM, 1 USB port, 40 GPIOs, no Ethernet
- Pi 1 Model B+: 512MB RAM, 4 USB ports, 40 GPIOs, Ethernet
- Pi 2: Same spec as Model B+, but 900MHz quad-core ARM Cortex-A7 CPU, 1GB RAM
  - Model A+ and B+ are single core 700MHz ARM1176JZF-S
- Pi 3: 64-bit quad-core ARM Cortex-A53 CPU, 1GB RAM, built-in Wi-Fi and Bluetooth
  - Wi-Fi and Bluetooth on same chip… programmatically can't use them both…
- Pi Zero: Single-core 1 GHz, 512MB RAM, mini-HDMI, USB On-The-Go (OTG) ports
- Pi 4: Quad-core Cortex-A72 1.5GHz, 2/4/8GB RAM, 1Gbps Ethernet, 2 USB 3.0 ports, dual micro HDMI ports
- Pi Pico (RP2040): Dual-core Cortex-M0+ 133MHz, 264 KB SRAM, 2 MB Flash, 26 GPIOs, 3 analog inputs, 2 SPIs, 2 I2Cs, 16 PWM channels

Raspberry Pi 3



Raspberry Pi 4



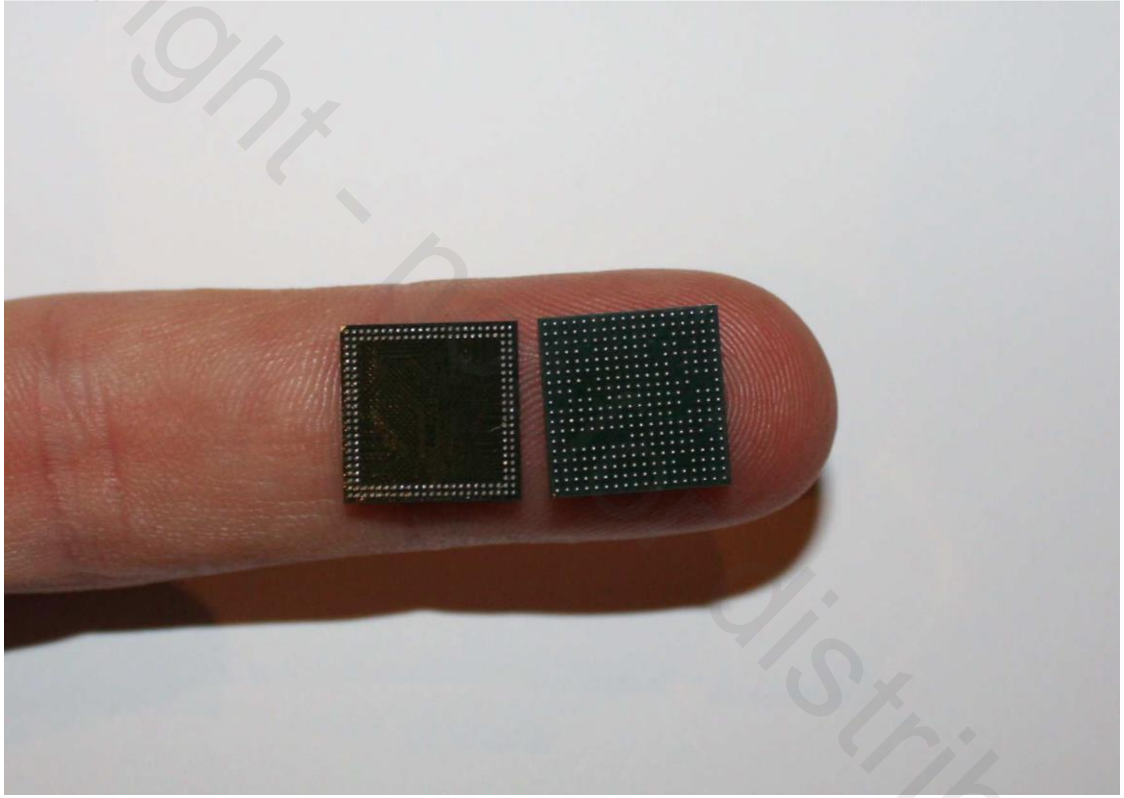
Specification (RPi3)

- CPU: Broadcom BCM2837 1.2 GHz 64-bit quad-core
  - ARMv8 Cortex-A53
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Network: 10/100 Ethernet, 2.4 GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 (Classic, Bluetooth LE)
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm audio jack, 4xUSB 2.0, Ethernet, Camera Serial Interface, Display Serial Interface

Specification (RPi4)

- CPU: Broadcom BCM2711 1.5 GHz 64-bit quad-core
  - ARMv8 Cortex-A72
- GPU: Broadcom VideoCore IV
- RAM: Up to 8 GB
- Network: 1Gbps Ethernet, 802.11ac 2.4/5GHz Wi-Fi
- Bluetooth: Bluetooth 5.0 (Classic, Bluetooth LE)
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm audio jack, 2xUSB 2.0, 2xUSB 3.0, Ethernet, Camera Serial Interface, Display Serial Interface

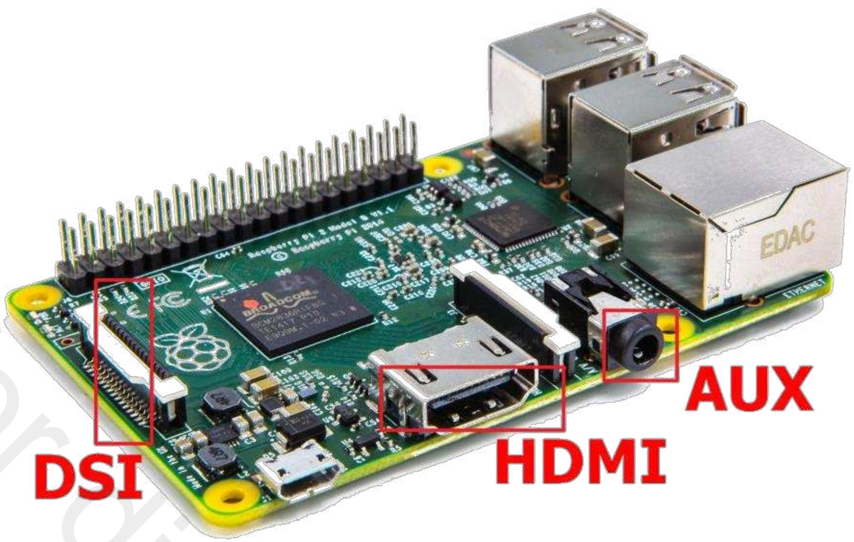
SoC in Raspberry Pi



Samsung K4P2G324ED Mobile DRAM (Left) & RPi BCM2835 SoC (Right)

Connecting to a Display/Audio

- HDMI
  - Digital signal
  - Video and audio signal
  - DVI, cannot carry audio signal
  - 1080p60
- Display Serial Interface (DSI)
  - For LCD panels
- No physical display available?
  - SSH connection through Terminal
  - Ethernet/Wi-Fi connection
  - VNC Server setup
- 3.5mm audio jack



USB (Universal Serial Bus)

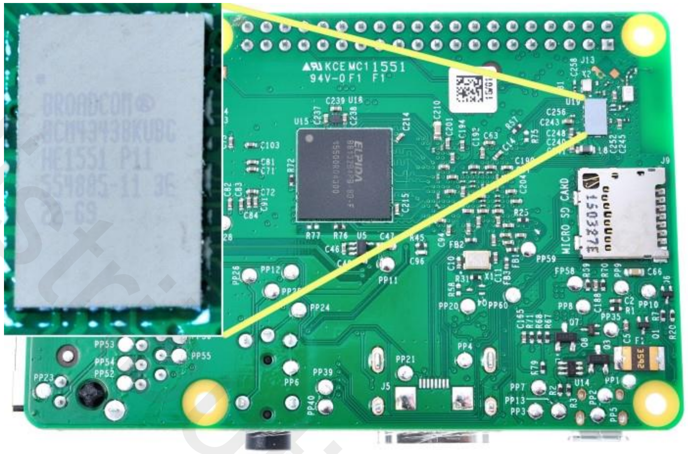
- 5 USB Ports on RPi
  - 4 USB 2.0 Ports
  - 1 USB port occupied by the Ethernet port
- Keyboard, Mouse, Wi-Fi, Bluetooth and more
- Can be extended through active USB hubs
  - Passive USB hubs aren’t ideal for current-hungry devices
    - CD drives, external hard drives
    - Recommended to use a powered USB hub
- Interfaced through the operating system’s driver

Storage

- microSD card
- Types of Card
  - SDSC (SD): 1MB to 2GB
  - SDHC: 4GB to 32GB
  - SDXD: up to 2TB
    - Class 10 and up recommended for speed
- Useful to have USB-to-microSD card reader
  - Install ROMs from your computer at any time
- External hard drive can be an option

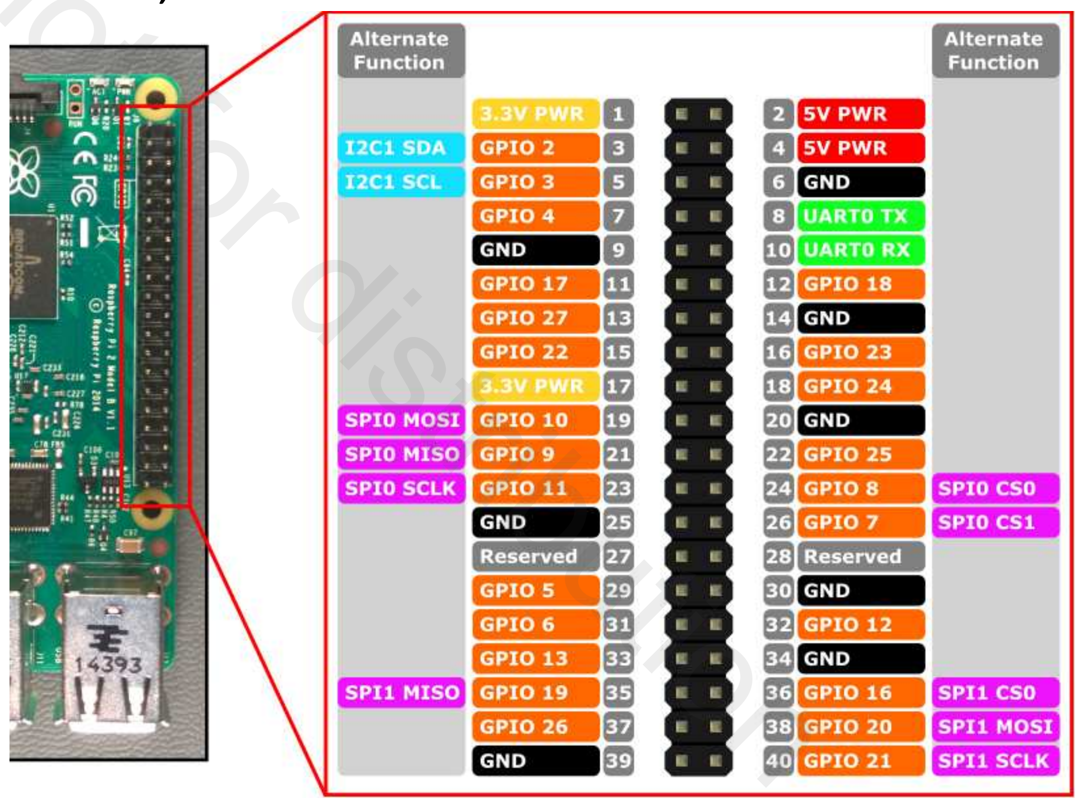
Networking (Wired/Wireless)

- Ethernet (IEEE 802.3) – 10/100Mbps
- Wi-Fi (IEEE 802.11, built-in RPi3)
  - 802.11b, up to 11 Mbps
  - 802.11g, up to 54 Mbps
  - 802.11n, up to 300 Mbps
  - 802.11ac, up to 1Gbps (*not supported on RPi3's built-in Wi-Fi*)
  - Frequency bands
    - 2.4GHz (for built-in Wi-Fi)
    - 5 GHz (USB dongle)
- Bluetooth (built-in RPi3)
  - Version 4.1
  - Support Classic & Low Energy
- All above can be added using USB adapter/converter



Low-level Peripherals

- General Purpose Input/Output (GPIO)
  - Pins can be configured to be input/output
  - Reading from various environmental sensors
    - IR, video, temperature, 3-axis orientation/acceleration
  - Writing output to DC motors, LEDs for status
- 17 GPIOs including
  - UART
  - I^{2}C bus
  - SPI bus with 2 Chip selects
  - I^{2}S audio
  - +3.3V
  - +5V



Power Consumption

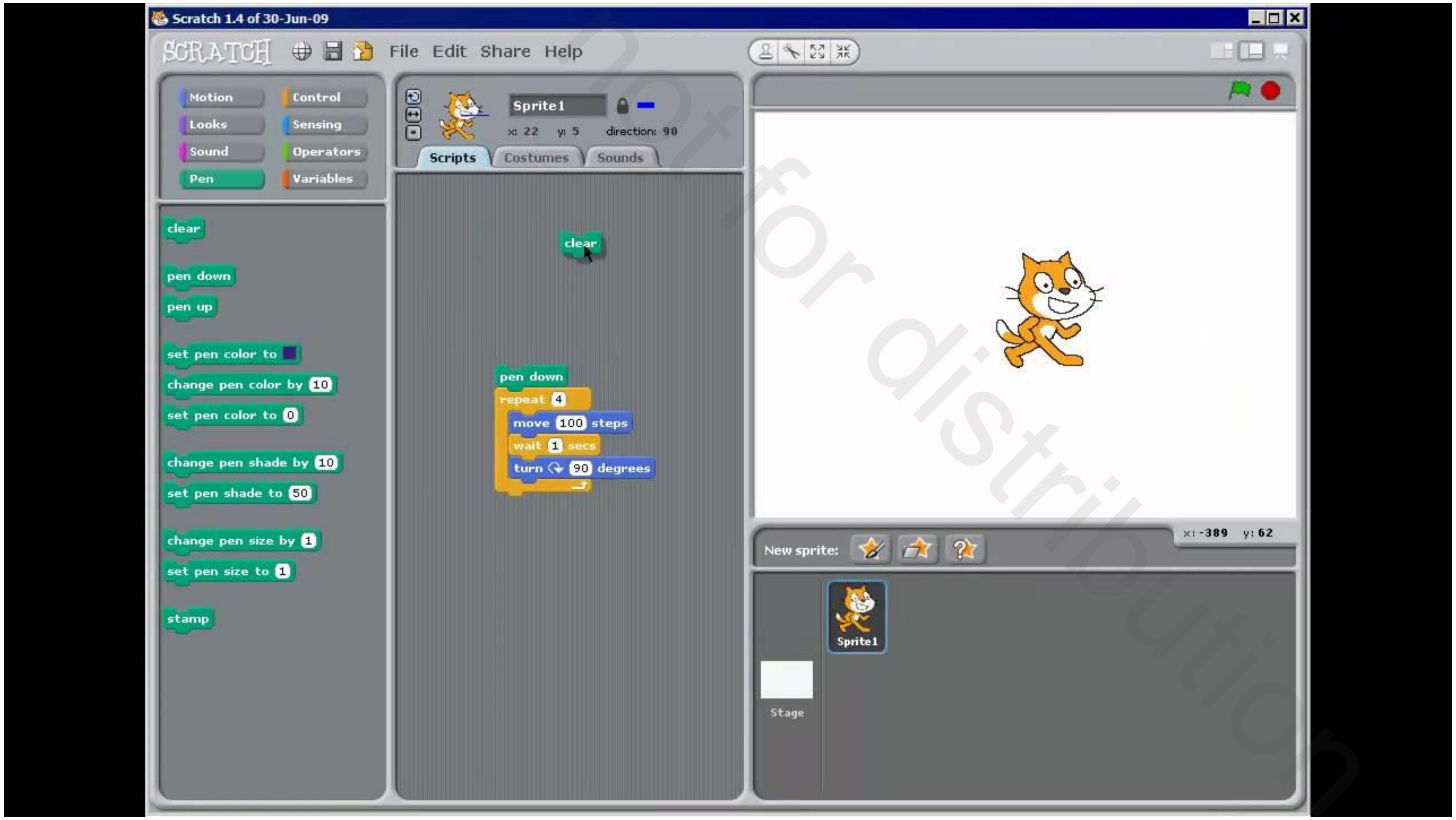
- Powered via microUSB connector
  - Wall adapter (recommended to use 5V, 2.5A)
  - USB port from PC (not recommended for power computing)
- Powered USB hub
  - To provide more power for USB peripherals

Programming Languages

- The Raspberry Pi Foundation recommends Python
- Any language which will compile for ARM can be used
- Others Installed by default on the Raspberry Pi:
  - C
  - C++
  - Java
  - Scratch
  - Ruby

Scratch

- Event-driven, block-based programming language
- Developed by MIT Media Lab, targeted mainly at children
- First appeared in 2002, public launch in 2007…



Ruby

- Object-oriented, dynamic, reflective, general-purpose script programming language
- Open source and developed by Yukihiro Matsumoto in 90s
- Classes with inheritance, mixins, iterators, closures, exception handling, garbage collection

<Example>

```
5.times { print “hello world!”}
# hello world!hello world!hello world!hello world!hello world!

if “text”.include? “ex”
		puts “match”
end
#match
puts “no match” unless “text”.include? “ttt”
#no match
```

Operating Systems

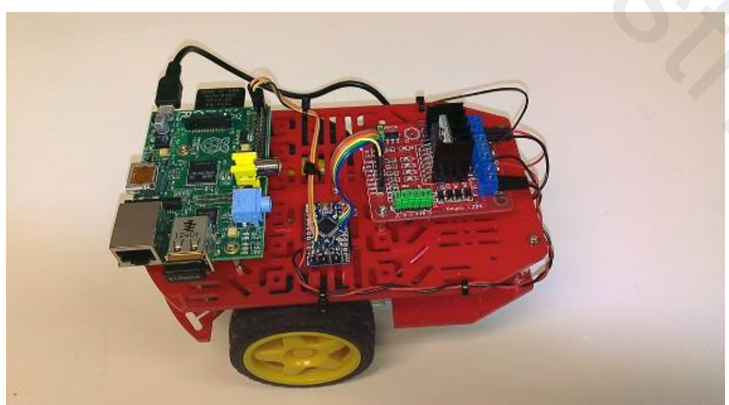
- Raspbian
  - Debian-based OS optimized for Raspberry Pi HWs
  - Easy installation by deploying its image file onto an SD card
  - At least 8GB SD Card, Class 10 (performance)
- RISC OS
- RetroPie, Moebius, OSMC and more…
- IoT-oriented OS
  - Android Things
  - Windows 10 IoT Core

GPU (Graphics Processing Unit)

- Broadcom Videocore IV GPU
  - Tile-based renderer (TBR) that use up to four cores
  - 40 nm technology
  - Integrated graphics card, thus shared memory
- Capable of Blu-ray quality of 1080p with H.264 at 40Mb/s
- Graphics performance is similar to XBOX 1
- 24 GFLOPS of general purpose computational power
- Has texture filtering and DMA infrastructure
- OpenGL ES 1.1, OpenGL ES 2.0, hardware accelerated OpenVG 1.1, Open EGL and OpenMAX

Examples of Raspberry Pi Applications

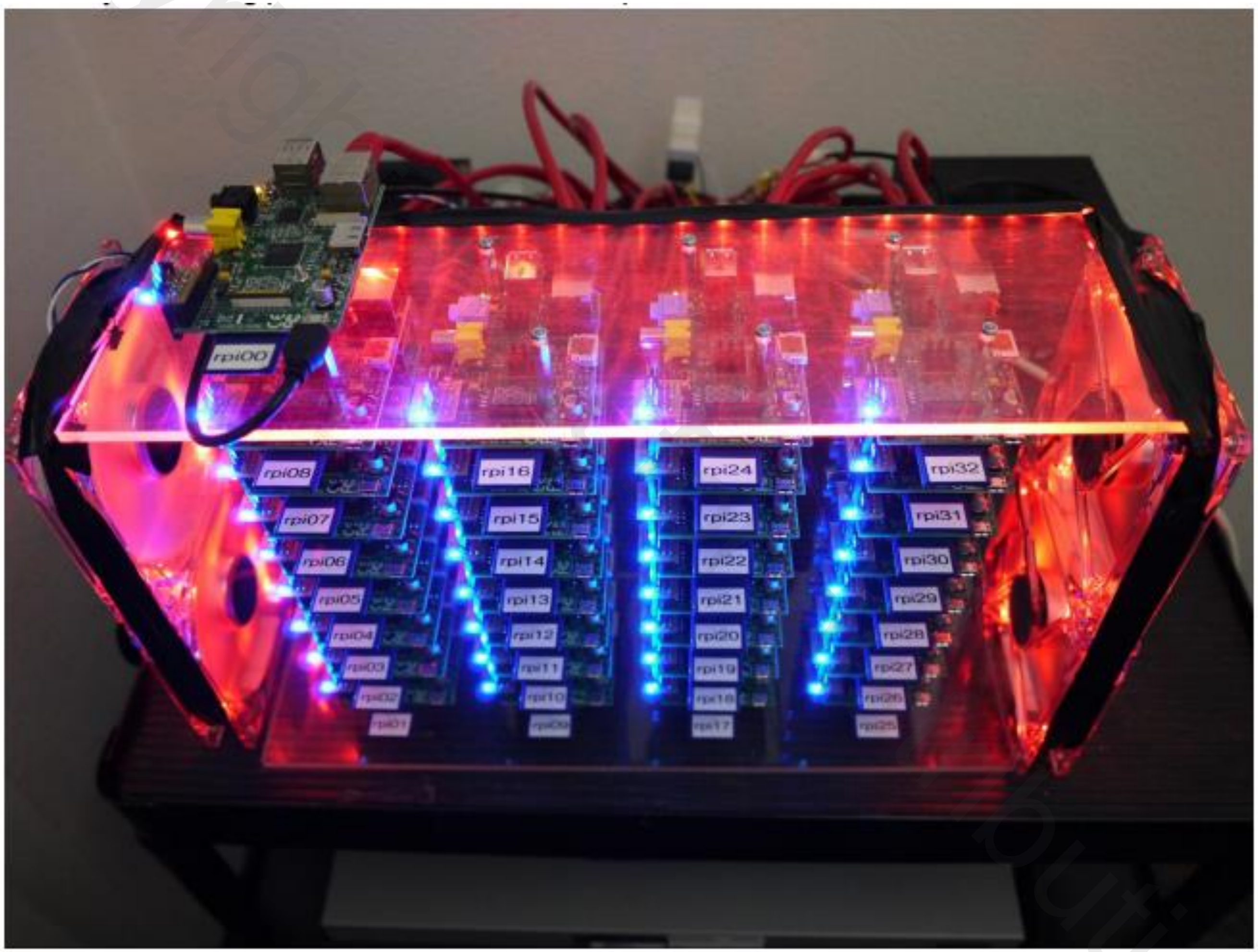
- Word Processor (use Libre Office)
- Programming: Python, C, Java, Scratch, etc.
- Game Console (https://youtu.be/pOqiqRHrAHE)
- Web Server: Apache HTTP Server, Database Management System
- NAS, LDAP server, Edge router, DNS server
- HTPC (Home Theater PC)
- Home Automation Central System, Alexa, Google Home
- Raspberry Pi-powered laptop (https://youtu.be/DTWe7EwriiA)
- Pi Robot



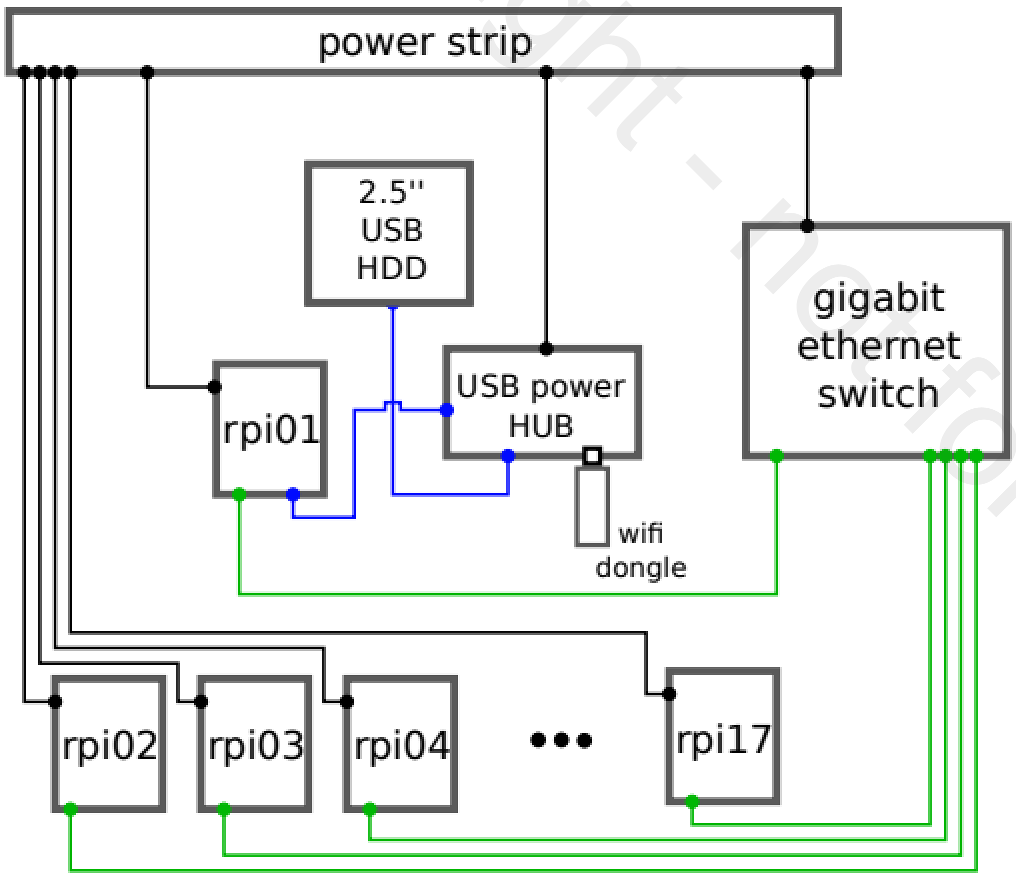

RPi NAS (Network Attached Storage)

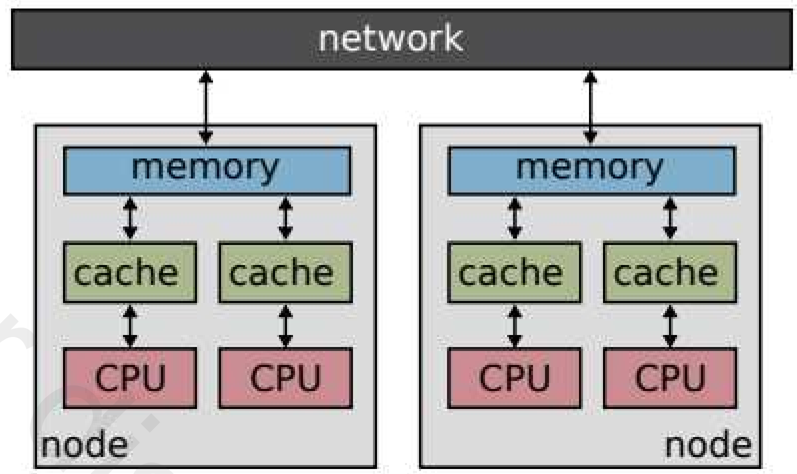
- NAS: External hard drives connected via LAN
- Smaller version of Google Drive, Dropbox, OneDrive
    - File server (FTP, NFS, SMB/CIFS…)
    - Streaming server (music or video over the network)
    - Personal web server (web hosting)
    - Local seedbox (for torrent file downloading)
    - USB to SATA controllers to connect hard drives
    - Could set up as RAID (0, 1, 5…)
    - May not be secure as the commercialized cloud service

Raspberry Pi for Cluster Computing



Raspberry Pi Cluster





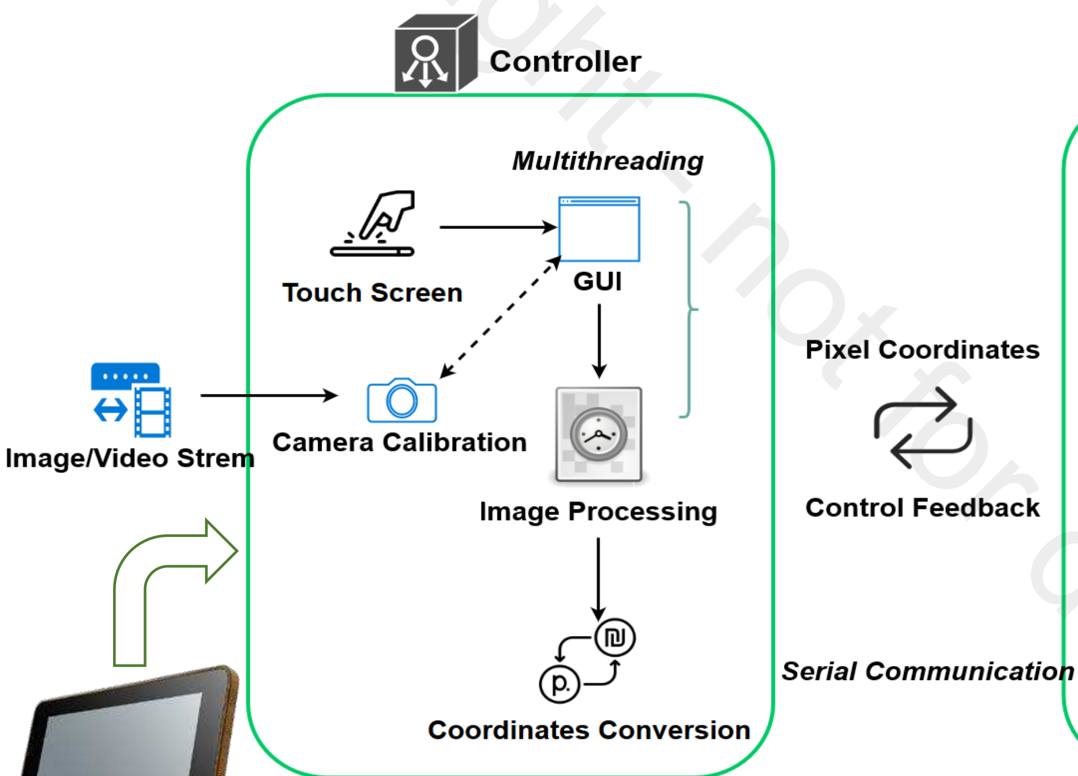
Raspberry Pi Cluster

- HPC (High-Performance Computing)
  - Connecting multiple computers to get higher performance
  - Scalability, availability, power efficiency
- Process-level parallelism
  - High throughput for independent jobs
- Parallel processing program
  - Single program run on multiple processors
  - Multiple RPis acting as multiple cores/processors
- Utilizing multiple RPi computing resources for parallel processing
  - One particular RPi governs all other RPis within the cluster to send commands and receive results
- Batch processing, complex numerical simulations/calculations
- Data transaction, appliance, web application servers

OpenCV with Raspberry Pi

- Open Source Computer Vision
  - A library of programming functions for real-time computer vision apps
  - Facial recognition, gesture recognition, robotics application
  - Augmented reality, motion tracking, stereo vision (3D), surveillance system
  - C++, Python, Java, Perl, MATLAB/OCTAVE…
  - Windows, Linux, macOS, FreeBSD, Android, iOS…
- Raspberry Pi Camera or USB Camera
  - Real-time image processing using OpenCV libraries
  - Many image processing related projects can be developed…

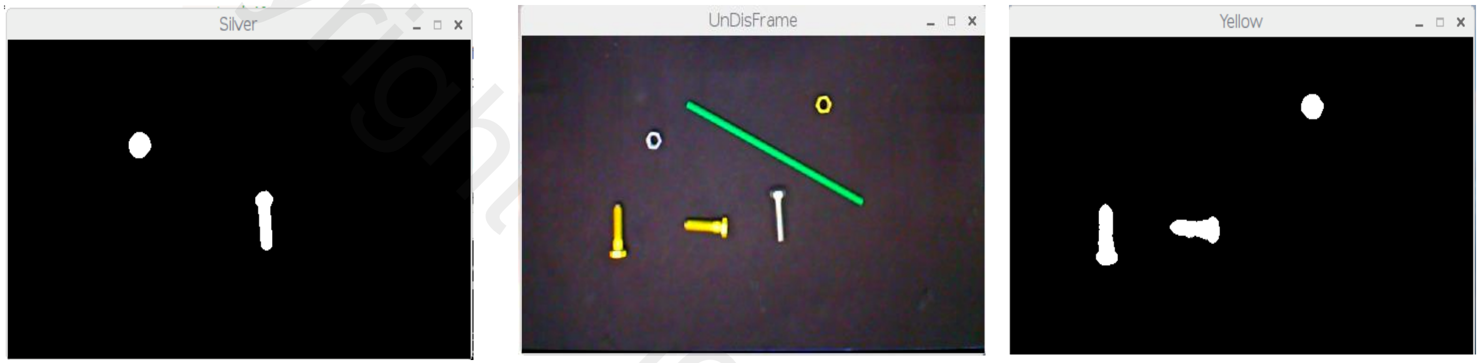
OpenCV with RPi – Robotic Arm



Architecture of Robotic Arm

http://ecasp.ece.iit.edu/video/summer2019/ecasp_robotic_arm_2019_summer.mp4

OpenCV with RPi – Robotic Arm

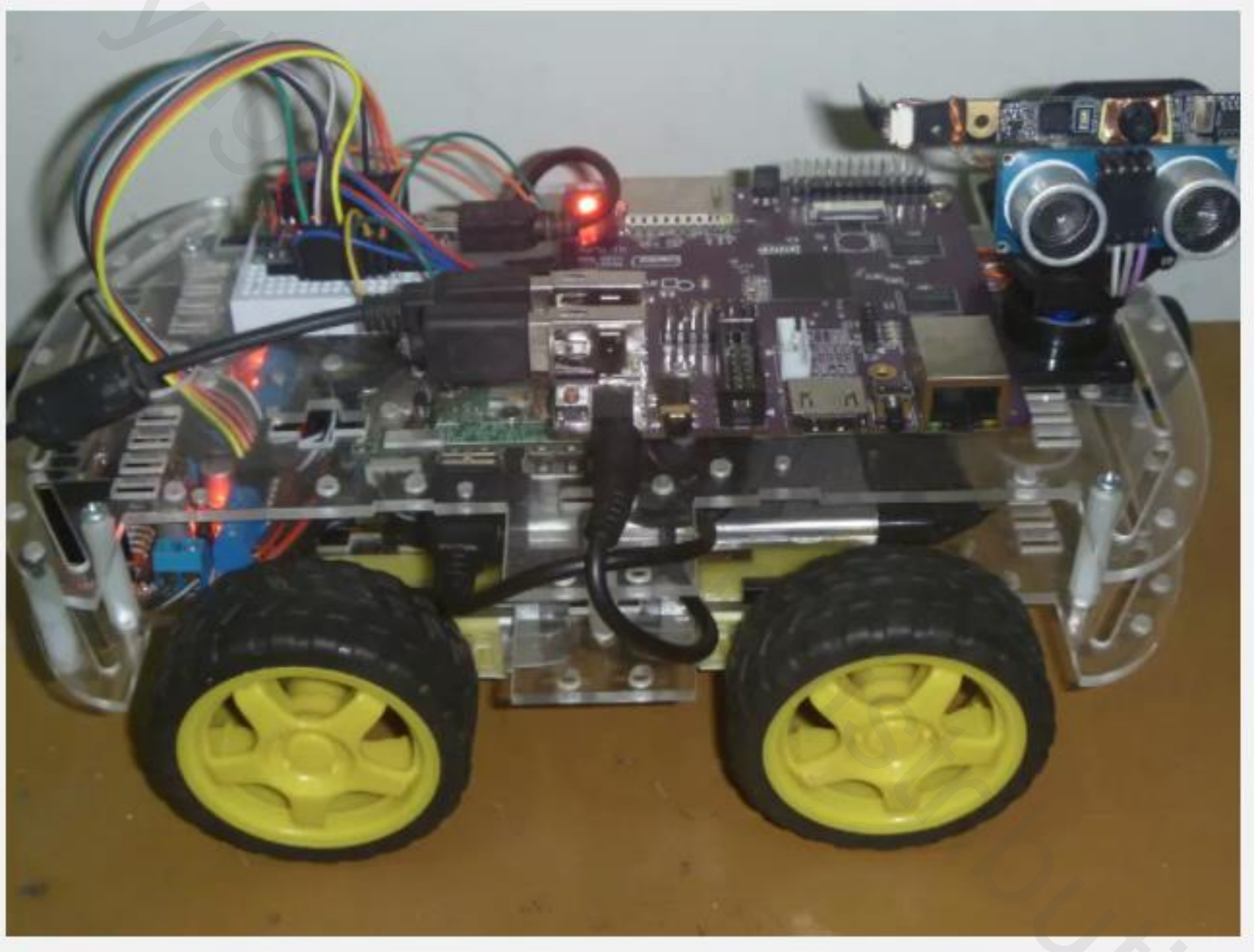


Color Segmentation using OpenCV

User Interface and Object Recognition

OpenCV with RPi – Object Recognition

OpenCV with RPi – Smart Face Tracker



LINK: https://youtu.be/KCavJ6M486I

Lab 2
Motion Sensing System Implementation using Raspberry Pi

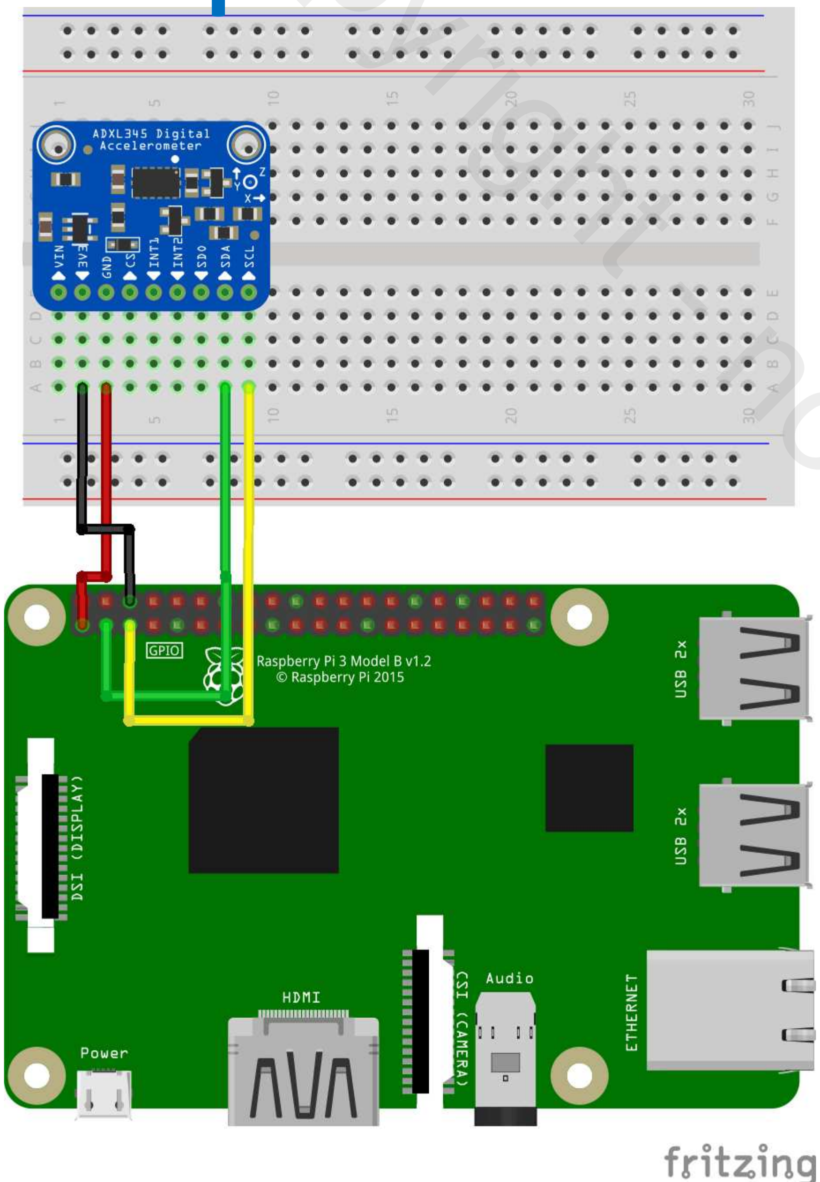
Lab explanation for Raspberry Pi

https://youtu.be/lzsWUj11JVs

Motion Sensing System Implementation using Raspberry Pi

- Understand how computer interface with I/O
  - Understand I^2C in C++
  - Understand SPI in C++
  - Understand Linux developing environment
- Understand client-server architecture
  - Brief introduction on socket in C++ and Python
  - Brief introduction on AES (Advanced Encryption Standard)

Motion Sensing System Implementation using Raspberry Pi



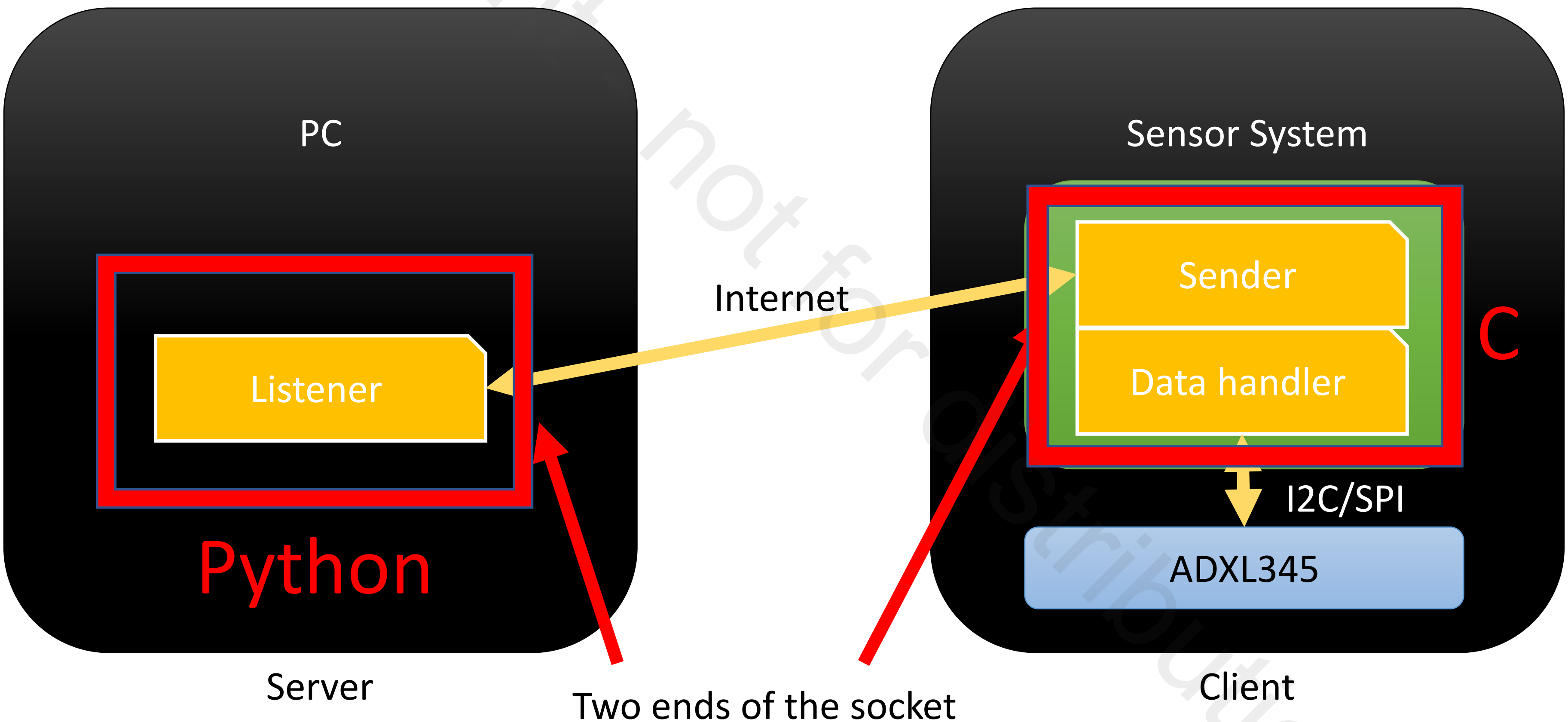
Hardware Configuration

- Wire corresponding pinouts from ADXL345(accelerometer) to RPi3
- 3.3V, GND, SDA, SCL
- X,Y,Z axes data sent to RPi over I^2C/SPI
  - 16-bit data per axis

Software Configuration

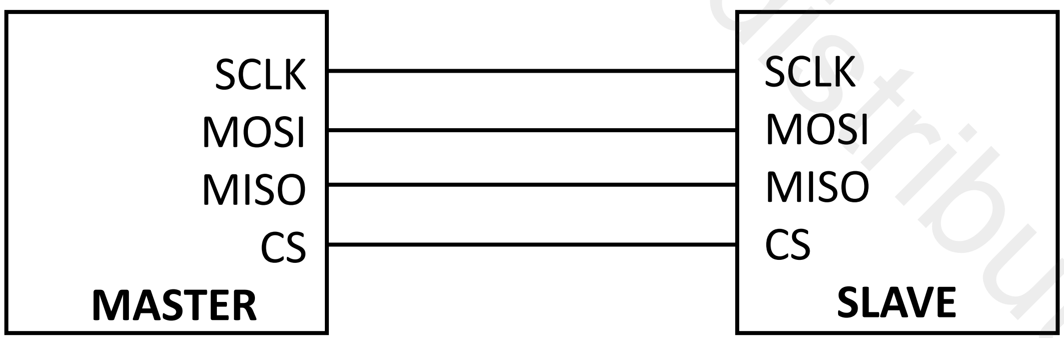
- Include predefined header from Adafruit for ADXL345
- C++ programming for reading ADXL345
- You can also use other languages, for example Python, but we will go over the C++ code in this lecture
- C++ code communicates with a server over UDP connection

Motion Sensing System Implementation using Raspberry Pi



Communication Protocols - SPI

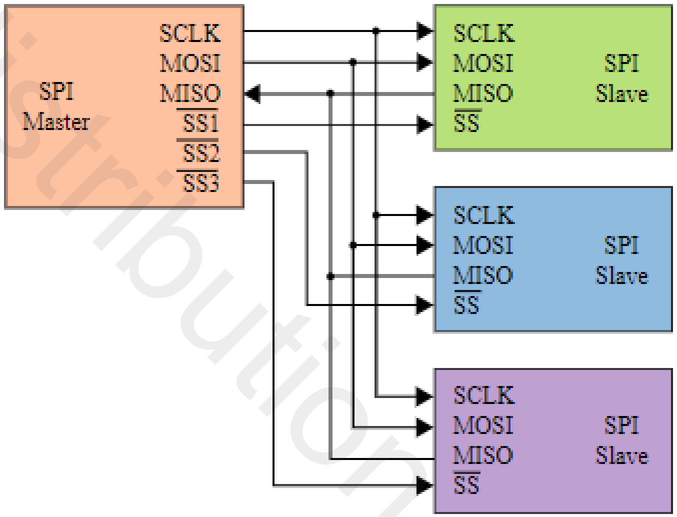
- Serial Peripheral Interface (SPI)
  - Full-duplex, short-distance, single-master protocol
  - Synchronous communication protocol
- Simple master-slave SPI connection
  - SCLK: Serial Clock (output from Master)
  - MOSI: Master Out Slave In (output from Master)
  - MISO: Master In Slave Out (output from Slave)
  - CS: Chip Select (also known as Slave Select, output from Master)



Simple master-slave SPI connections

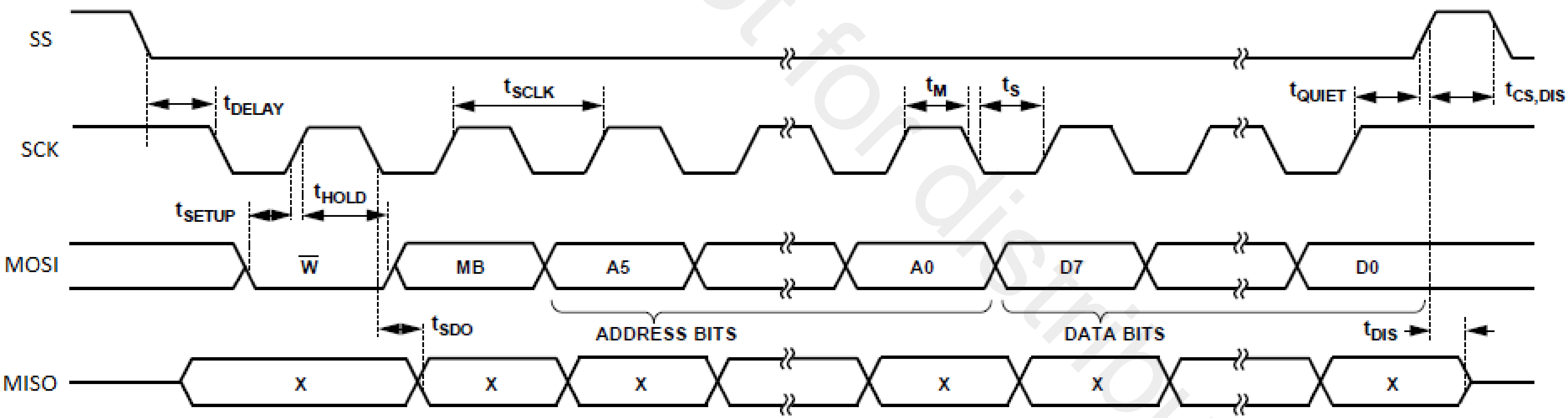
Communication Protocols - SPI

- Master ALWAYS initiates data frame and clock
- Clock frequencies vary, depending on master/slave
  - Typically from 1 MHz to 40 MHz or higher
- Some slave devices trigger on active low input
  - Logic zero signal from master → slave chip is ON → accepts clock and data
- Multiple slaves can be connected to a master device
  - Additional CS(SS) lines from master to multiple slaves
- NO ACK available
  - No way to know if data is received correctly



Communication Protocols - SPI

- SPI Bus
  - 4-wire mode is used in this lab
  - Require 4 wires for communication

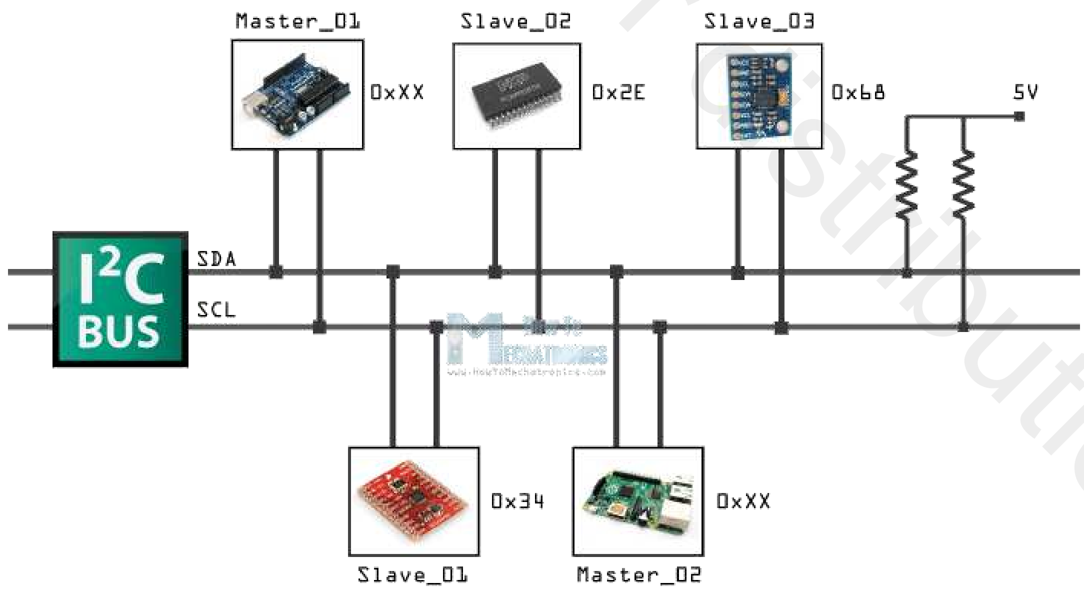


Communication Protocols – I^2C

- Inter-Integrated Circuit Bus
- Developed and patent by Philips Semiconductors
  - Original purpose: Connect a CPU to peripheral chips in a TV-set
- Peripheral devices in embedded systems
  - Connected to microcontroller as memory-mapped I/O devices
- Requires only two wires, data (SDA) and clock (SCL)
  - 2-wire communication bus to provide communication link between integrated circuits
  - Always pulled up via resistors to the input voltage
- Half duplex: sender sends the command, the receiver just listens and cannot transmit anything; and vice versa
- Three speeds available
  - High speed (3.4 MBps)
  - Fast speed (400 KBps)
  - Slow (<100 KBps)

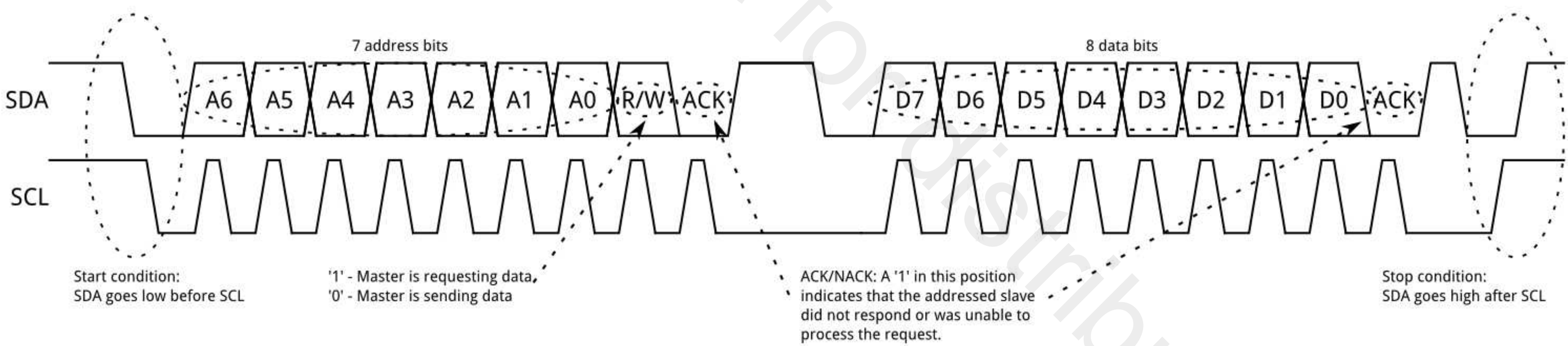
Communication Protocols – I²C

- Most Widely used protocol for sensor interfacing in embedded systems
  - Accelerometer, gyroscope, temperature, humidity sensors
  - Stepper motor control, proximity sensor and more
- Useful for low-pin-count devices
  - All sensors can be tied to a single I²C bus
  - How differentiated?



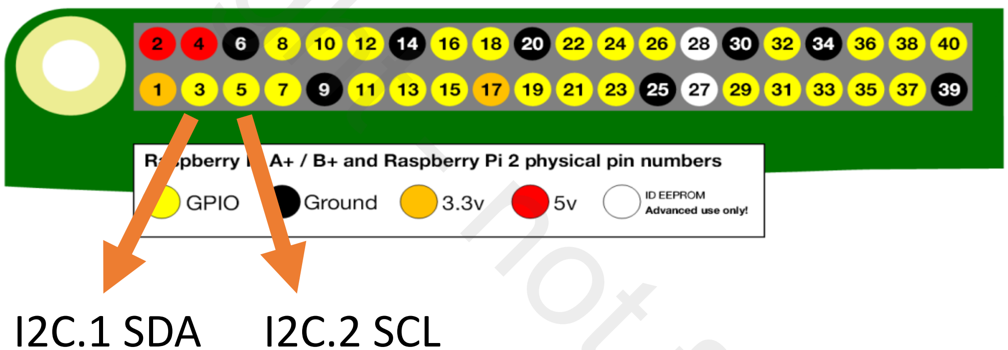
Communication Protocols – I^{2}C

- I2C Bus
  - 7-bit addressing mode is used
  - Require only 2 wires for communication

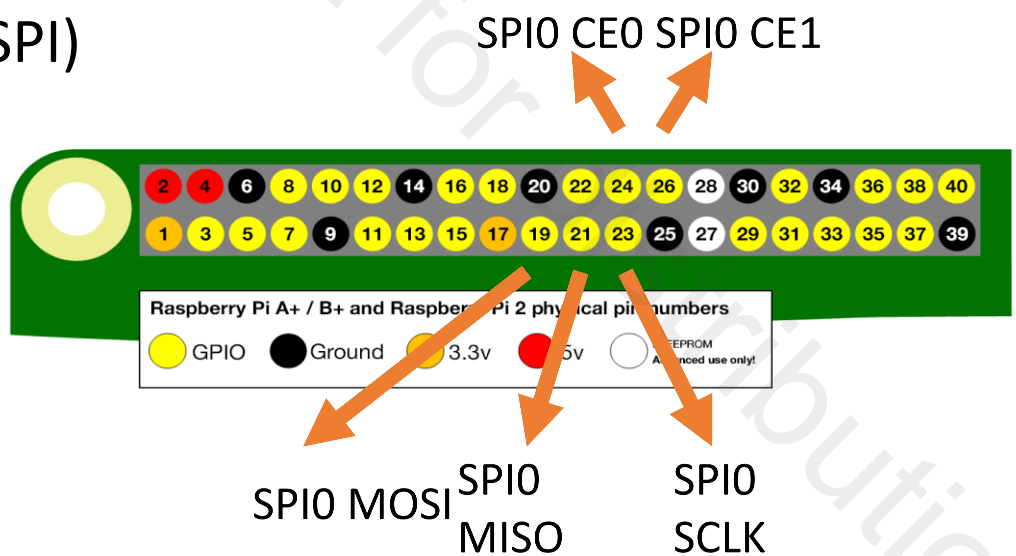


Motion Sensing System Implementation using Raspberry Pi

- Raspberry Pi (I^2C)

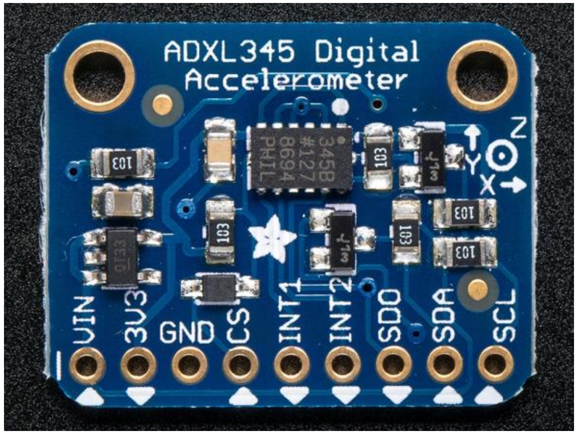


- Raspberry Pi (SPI)



- Do you want to modify these pin-maps? Re-compile the kernel!

Motion Sensing System Implementation using Raspberry Pi



- Adafruit ADXL345
  - Accelerometer
  - As low as 40 μA in measurement mode
  - 0.1 μA in standby mode
  - 4 sensitivity levels: $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$
  - Occupies I^2C 7-bit address: 0x53
  - V_{CC} takes up to 5V in, and regulates it to 3.3V
  - Capable of SPI interface also
- Where to get these specs?
  - To know your device, you need to look into the datasheet
  - https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf

Motion Sensing System Implementation using Raspberry Pi

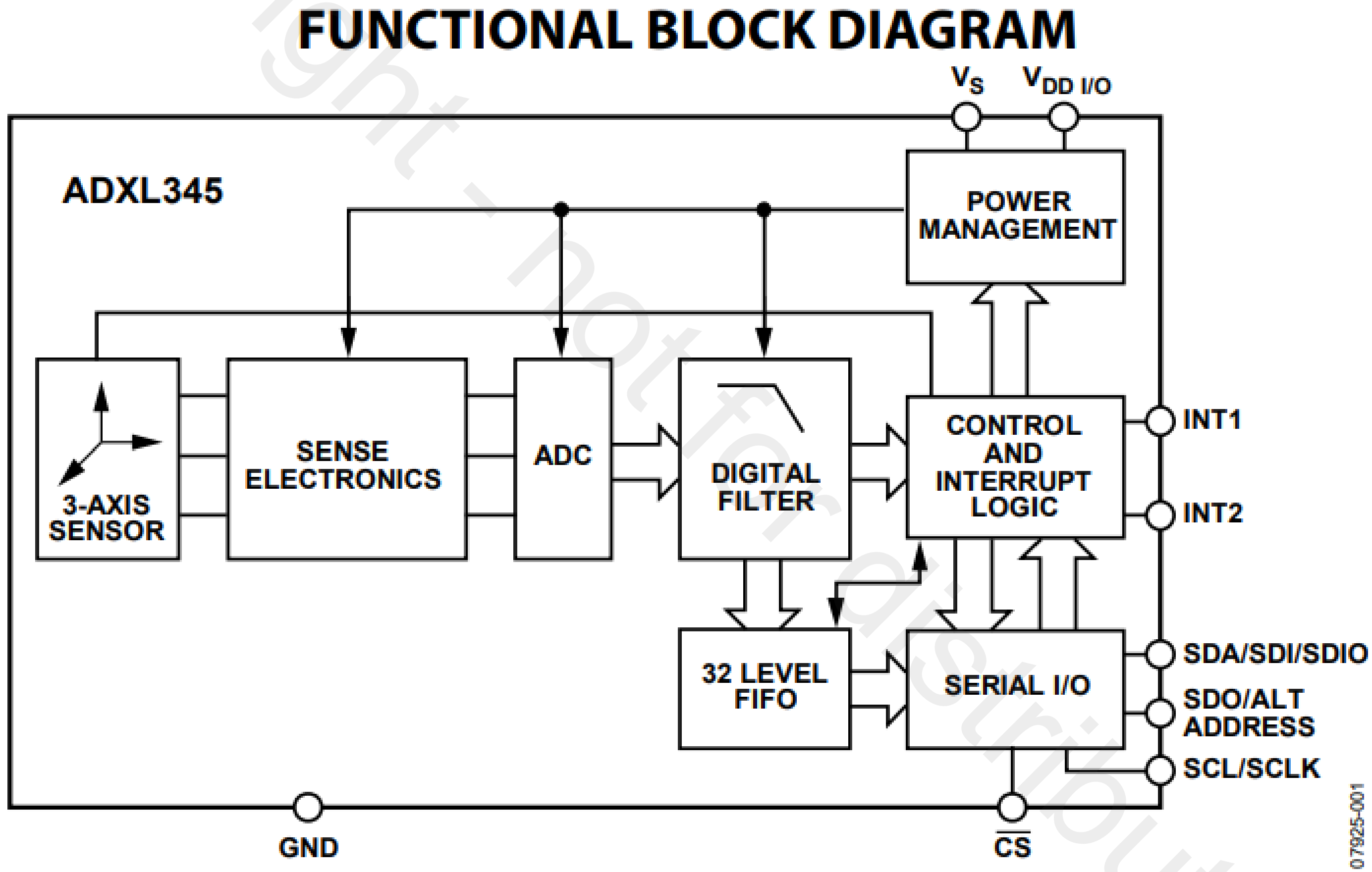


Figure 1.

Software and Tools

- Linux Developing Environment
  - No dedicated IDE (Integrated Developer Environment)
  - Use text editor to write codes
  - GDB to debug and G++ to compile
- Python Runtime
  - Script language
  - No need to compile, but will be interpreted
- AES
  - Symmetric-key encryption/decryption
  - Encrypt block(s) of data
  - Use IV (Initial Vector) to avoid eavesdropping

ADXL345.h

```
class ADXL345 {
 public:
      ADXL345 (int fdx, unsigned char addx=0x53) { fd=fdx; myAddr=addx;}
      int init();
      bool readXYZ(short &ax, short &ay, short &az);
  private:
      bool selectDevice();
      bool writeToDevice(char * buf, int len);
      unsigned char myAddr; // ADXL345 device address
      int fd;// File descriptor
};
```

- Constructor: *fd* (file descriptor value of I^{2}C from main function), ADXL345 I^{2}C device address (0x53) set to *myAddr*
- init(): initializing ADXL345 device over I^{2}C bus
- readXYZ(): reading X, Y, Z axes data, storing in *short* variables
- selectDevice(): verifies if ADXL345 is connected correctly over I^{2}C bus
- writeToDevice(): writes commands to ADXL345 over I^{2}C bus

ADXL345.cpp


```
int ADXL345::init() {
        assert (fd>0);                  // crash if port was not opened earlier
        char buf[6];                    // buffer for data being read/written on I^2C bus
        if(!selectDevice()) return -1;  // if I2C device is not present, return -1

        buf[0] = 0x2d;                  // select Register 0x2D – POWER_CTL (Read/Write)
        buf[1] = 0x18;                  // write 0x18 to Register 0x2D
        if(!writeToDevice(buf,2)) return -2;    // if writing buf to I2C device fail, return -2

        buf[0] = 0x31;                  // select Register 0x31 – DATA_FORMAT (Read/Write)
        buf[1] = 0x0A;                  // write 0x0A to Register 0x31
        if(!writeToDevice(buf,2)) return -3;    //if writing buf to I^2c device fail, return -3

        printf("ADXL345:init() OK\n");
        return 0;
}
```

- ADXL345 initialization process required before retrieving accelerometer data
- Refer to ADXL345 Datasheet for Register information (https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf)

ADXL345.cpp

```
……
    buf[0] = 0x2d;                   // select Register 0x2D – POWER_CTL (Read/Write)
    buf[1] = 0x18;                   // write 0x18 to Register 0x2D
    if(!writeToDevice(buf,2)) return -2;      // if writing buf to I2C device fail, return -2
…….
```

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	
0	0	0	1	1	0	0	0

- Link Bit = 0; inactivity and activity functions are concurrent
- AUTO_SLEEP = 1; switch to sleep mode when inactivity is detected
- Measure = 1; sets to measurement mode
- Sleep = 0; sets to normal mode of operation
- Wakeup = 00; sets frequency of readings in sleep mode to 8 Hz

ADXL345.cpp

```
……
    buf[0] = 0x31;        // select Register 0x31 – DATA_FORMAT (Read/Write)
    buf[1] = 0x0A;        // write 0x0A to Register 0x31
    if(!writeToDevice(buf,2)) return -3;      //if writing buf to I2c device fail, return -3
…….
```

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	
0	0	0	0	1	0	1	0

- SELF_TEST = 0; disables self-test force
- SPI = 0; sets the device to 4-wire SPI mode
- INT_INVERT = 0; sets the interrupts to active high
- FULL_RES = 1; full resolution mode, where the output resolution increases with the g range set by the range bits to maintain a 4 m*g*/LSB scale factor
- Justify = 0; right (LSB) justified mode with sign extension
- Range = 10; ±8 *g*, 10 bit mode

ADXL345.cpp

```
bool ADXL345::selectDevice() {                    // select I2C device from bus
    if (ioctl(fd, I2C_SLAVE, myAddr) < 0){        // check if myAddr is available as I2C slave
        fprintf(strderr, “Device ADXL345 not present\n”);
        return false;
    } return true;
}

bool ADXL345::writeToDevice(char* buf, int len) { // write to I2C device’s register
    if (write(fd, buf, len) != len) {             // check if writing went through
        fprintf(strderr, “can’t write to device ADXL345 buf=%s, len=%d\n”,fd, buf,
len);
        return false;
    } return true;
}
```

- Both functions called during initialization process
- ioctl(): a system call for device-specific *input/output control* operations
- write(): a system call that writes data from a buffer to a given device referred by file descriptor *fd*

ADXL345.cpp

```
bool  ADXL345::readXYZ( short &x , short &y, short &z) {
        assert(fd>0);                           // crash if port was not opened earlier
        if(!selectDevice())                     // check if device is alive
                return false;
        char buf[7];
        buf[0] = 0x32;                          // starting register for accelerometer data
        if(!writeToDevice(buf,2))
                return false;
        if (read(fd, buf, 6) != 6) {  // Read back data into buf[]
                printf("Unable to read from slave for ADXL345\n");
                return false;
        }  else {
                x = (buf[1]<<8) |  buf[0];      // X axis data
                y = (buf[3]<<8) |  buf[2];      // Y axis data
                z = (buf[5]<<8) |  buf[4];      // Z axis data
        }
        return true;
}
```

- *readXYZ()*: reading in X, Y, Z axes accelerometer data to corresponding *short* variables

ADXL345.cpp

```
…
    buf[0] = 0x32;                    // starting register for accelerometer data
    if(!writeToDevice(buf,2))
        return false;
    if (read(fd, buf, 6) != 6) {  // Read back data into buf[]
        printf("Unable to read from slave for ADXL345\n");
        return false;
    }  else {
        x = (buf[1]<<8) |  buf[0];     // X axis data
        y = (buf[3]<<8) |  buf[2];     // Y axis data
        z = (buf[5]<<8) |  buf[4];     // Z axis data
    }
…
```

- To start retrieving data from ADXL345, request data by writing to 0x32
- Register 0x32, 0x33 – DATAX0(least significant byte), DATAX1(most significant byte)
- Register 0x34, 0x35 – DATAY0(least significant byte), DATAY1(most significant byte)
- Register 0x36, 0x37 – DATAZ0(least significant byte), DATAZ1(most significant byte)

main.cpp

```
…
#include "ADXL345.h"
#define I2C_FILE_NAME "/dev/i2c-1"

int main(int argc, char **argv) {
	// Open a connection to the I2C userspace control file
	int i2c_fd = open(I2C_FILE_NAME, O_RDWR);
	if (i2c_fd < 0) {			// check if it successfully opened the connection
		printf("unable to open I2C control file, err=%d\n",i2c_fd);
		exit(1);
	}
…
```

- Declare ADXL345 header *ADXL345.h*
- In Linux system, I2C device will be loaded under */dev/i2c-**, * depends on the device
  - *ls /dev/i2c-** : find out device number
- Open the connection with */dev/i2c-1* in read/write mode

main.cpp

```
…
    ADXL345 myAcc(i2c_fd);        // myAcc class of ADXL345
    int ret = myAcc.init();       // initialize myAcc/ADXL345 device
    if(ret) {
        printf("Failed to init ADXL345, ret=%d\n",ret); exit(1);
    }

    myAcc.readXYZ(ax,ay,az);      // reads data, stored in short variables ax, ay, az
    printf("Ax: %hi\t Ay: %hi\t Az: %hi\n", ax, ay, az);  // print X, Y, Z axes data
…
```

- Declare myAcc class of ADXL345, constructor executed with I2C file descriptor *(fd)*
- Initialize ADXL345 device, settings are defined in *ADXL345.cpp*
- Call readXYZ() function to retrieve data, store in *short int (16-bit)* variables
- *%hi*: to display short variable
    - h: for short/unsigned short
    - i: signed decimal notation

Sample Output – ADXL345 on RPi

pi@raspberrypi: ~/Desktop/Project

File Edit Tabs Help

```
---------------------
Loop Time 20    Ax : 17         Ay : -247       Az : -95
---------------------
Loop Time 20    Ax : 18         Ay : -247       Az : -95
---------------------
Loop Time 20    Ax : 17         Ay : -248       Az : -96
---------------------
Loop Time 20    Ax : 16         Ay : -247       Az : -95
---------------------
Loop Time 20    Ax : 18         Ay : -247       Az : -95
---------------------
Loop Time 20    Ax : 16         Ay : -246       Az : -95
---------------------
Loop Time 20    Ax : 19         Ay : -250       Az : -95
---------------------
Loop Time 20    Ax : 17         Ay : -249       Az : -94
---------------------
Loop Time 20    Ax : 18         Ay : -249       Az : -95
---------------------
Loop Time 20    Ax : 18         Ay : -248       Az : -95
---------------------
Loop Time 20    Ax : 18         Ay : -248       Az : -95
---------------------
```

- Continuous data retrieval from ADXL345 every 20ms
- Each axis raw data ranges from -512 to +511 (10-bit data)
- Refer to datasheet for m*g*/LSB for each *g* range
  - 15.6m*g*/LSB in this example (±8 *g* mode)
  - Ax = 18 → 18 x 15.6m*g*/LSB = <u>0.28 *g*</u>
  - Ay = -248 → -248 x 15.6m*g*/LSB = <u>-3.87 *g*</u>
  - Az = -95 → -95 x 15.6m*g*/LSB = <u>1.48 *g*</u>

References

[1] A more general tutorial on C programming on Raspberry Pi

https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/rpii/cintro.html

[2] More insight on socket programming

https://www.geeksforgeeks.org/socket-programming-cc

[3] Raspberry Pi hardware documents

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

[4] Common Linux commands

https://www.raspberrypi.org/documentation/linux/usage/commands.md

[5] Adafruit ADXL345 accelerometer program guides

https://learn.adafruit.com/adxl345-digital-accelerometer/programming

[6] GDB debugger quick manual

http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

[7] Python 3.6 tutorial

https://docs.python.org/3/tutorial/