

EE3070 Project Design

Modular Design

Dr. Wallace Tang

Semester A 2021/22

- Inputs: “cause” of the change
- Outputs: resulting action (“effect”) that occurs on the system output due to the cause



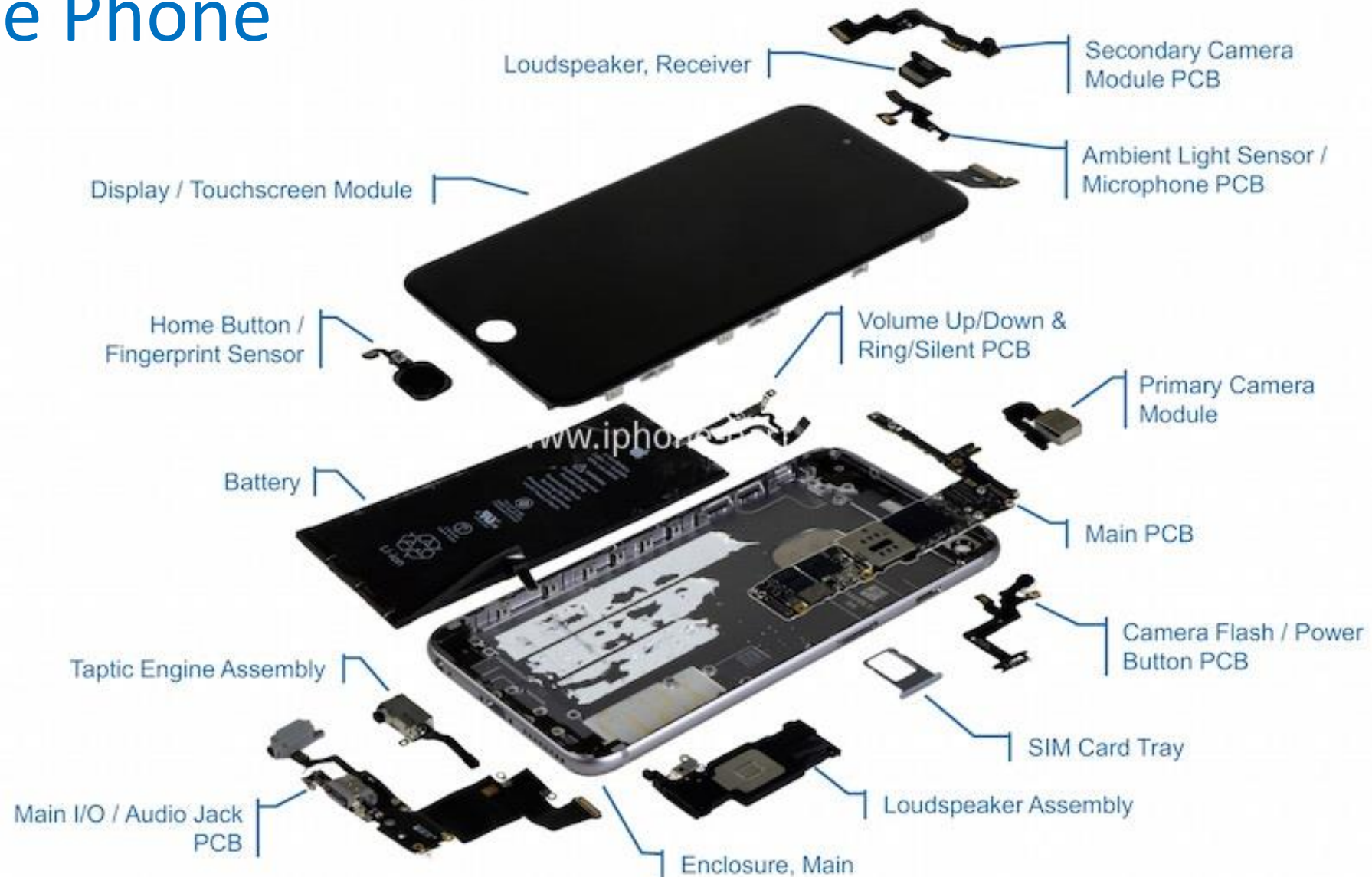
Electronic hardware + software design



Modular Design



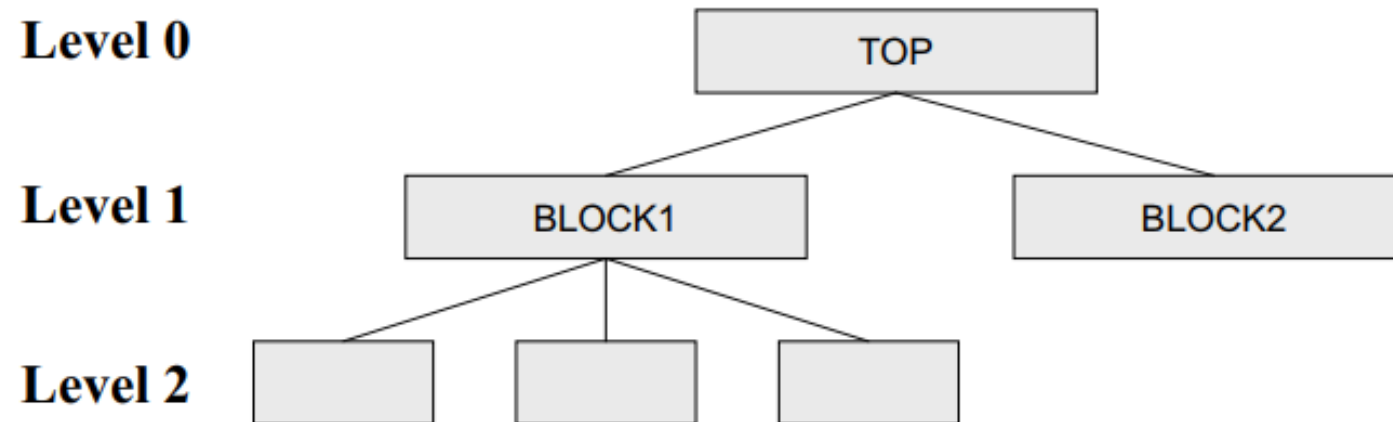
Mobile Phone



- Vehicle Manufacturing



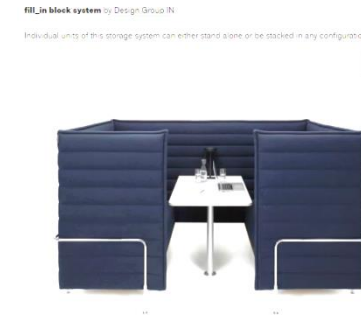
- Divide a complex system into blocks (modules)
- Blocks can be independently created and then used in different systems
- Specify the interconnection between blocks
- This concept is useful for both hardware and software designs



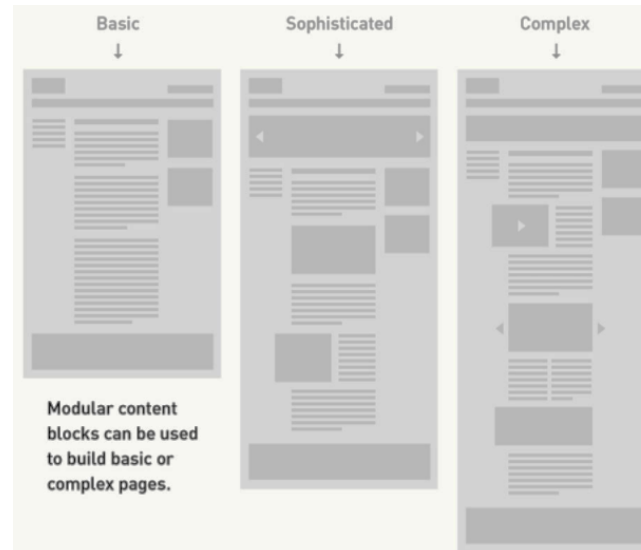
Easy (design, debug, upgrade, ...)



Flexible and Reusable



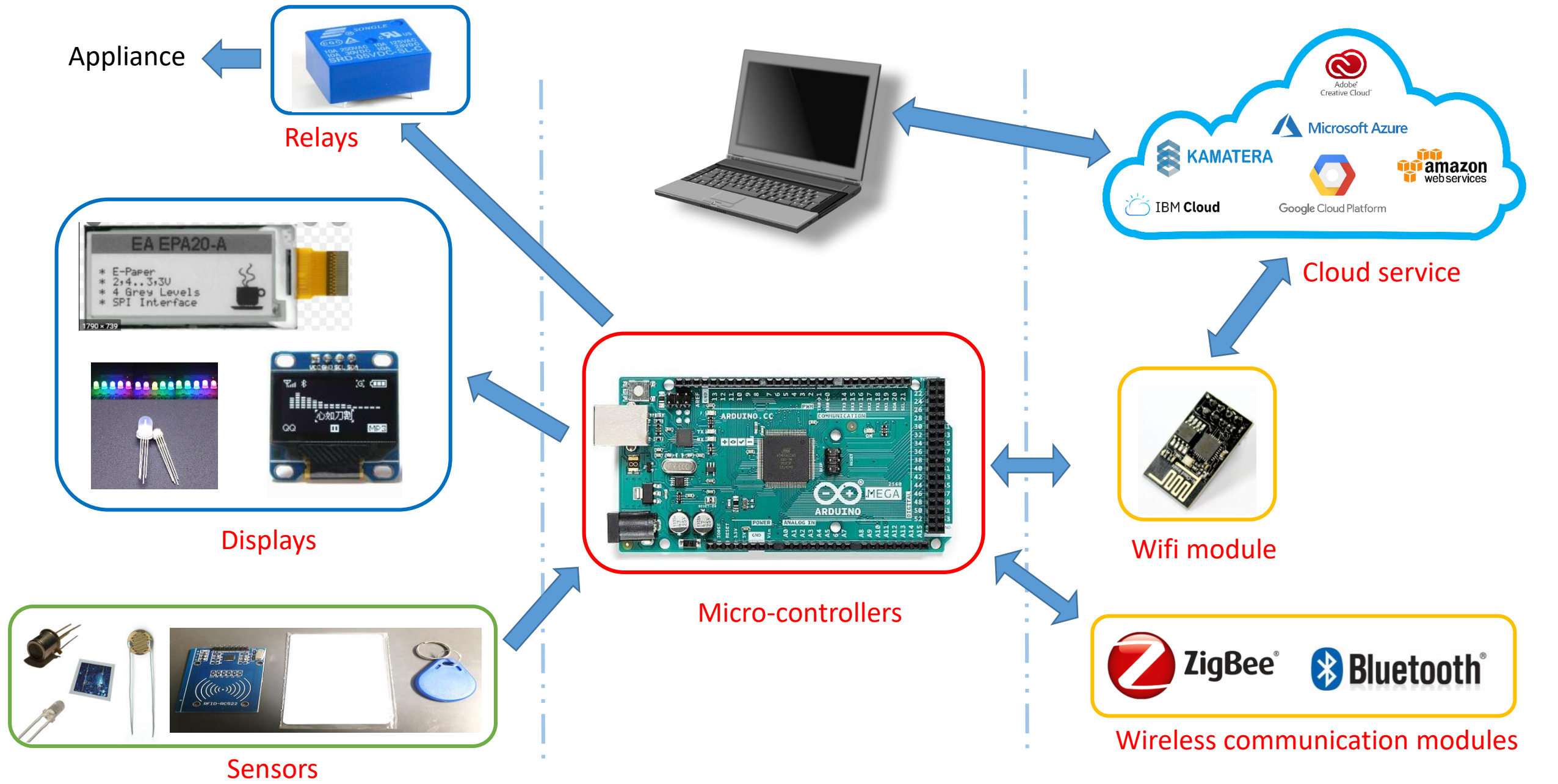
Expandable



Description of Modules

- I will not cover the technical details in the session. You may refer to the Technical Manuals
- I will
 - Describe the basic operational theory of different modules
 - Describe the functionality of the modules
 - Describe their potentials in Smart Home
- You will be given some basic units. Additional modules are only provided if they include in your proposal and there are sufficient stocks.

General System Architecture



Sensors

- Device to detect/measure a signal or stimulus
- Eg. thermometer: measure the temperature

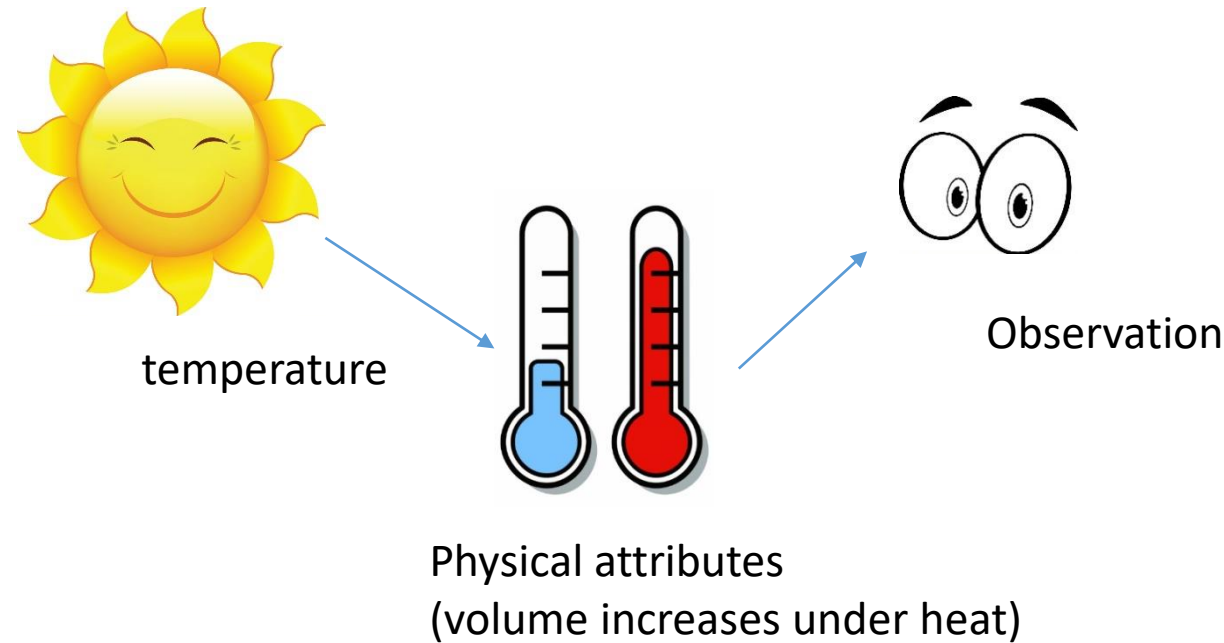
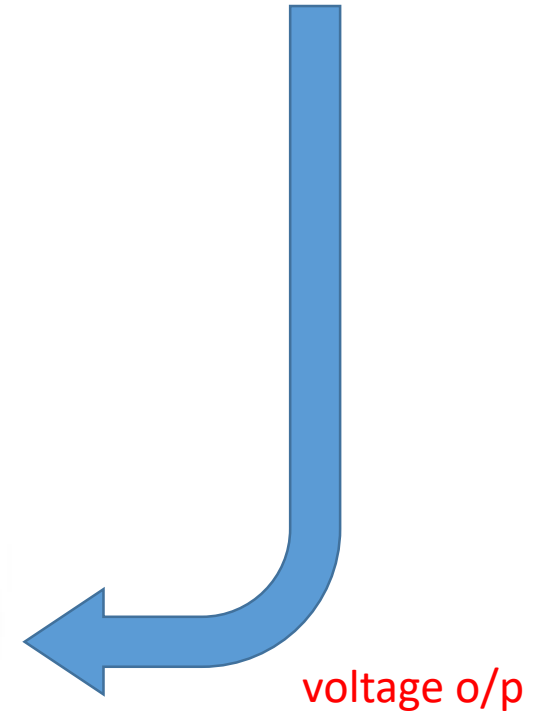


Table 1.1: Sensors' functions and output attributes

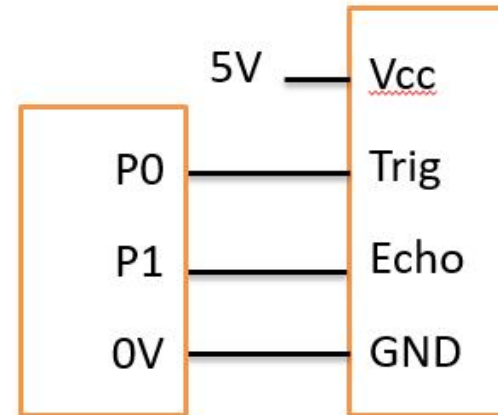
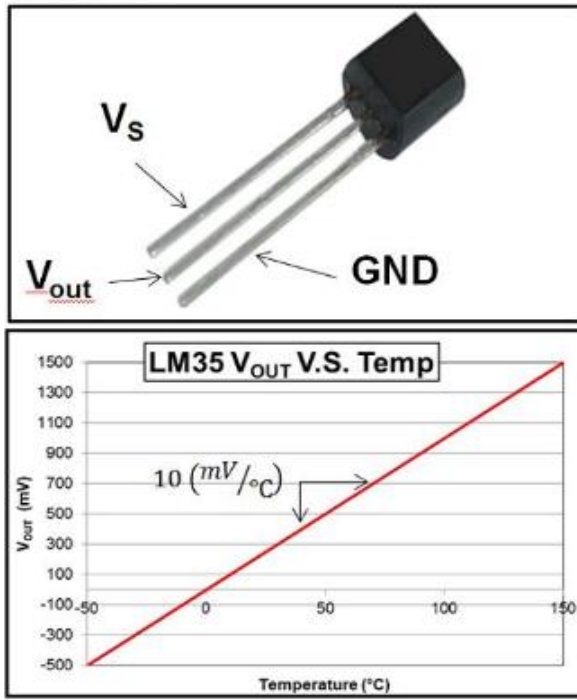
Sensor Part No.	Functions	Attributes
PGM5399	Measure light intensity	Resistance
RFP-602	Measure pressure applied to the surface	Resistance
HC-SR04	Measure distance	Signal Pulse width
LM35	Measure ambient temp	Voltage



Is Power Needed?

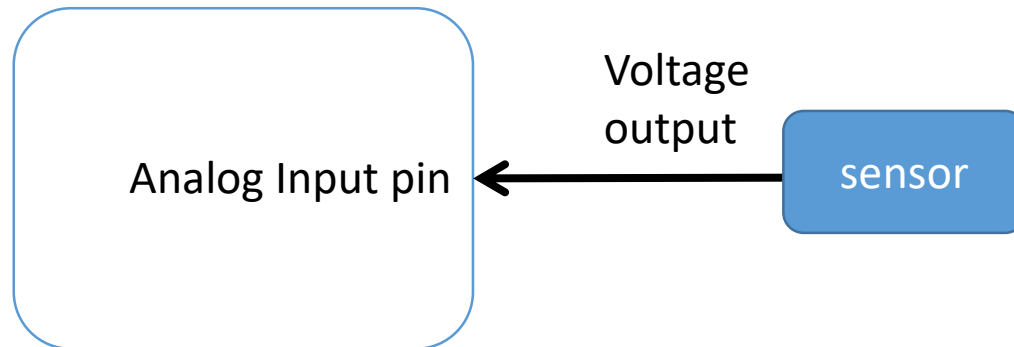
- Some sensors need power supply and the voltage level may vary (3.3V, 5V, etc)

LM35

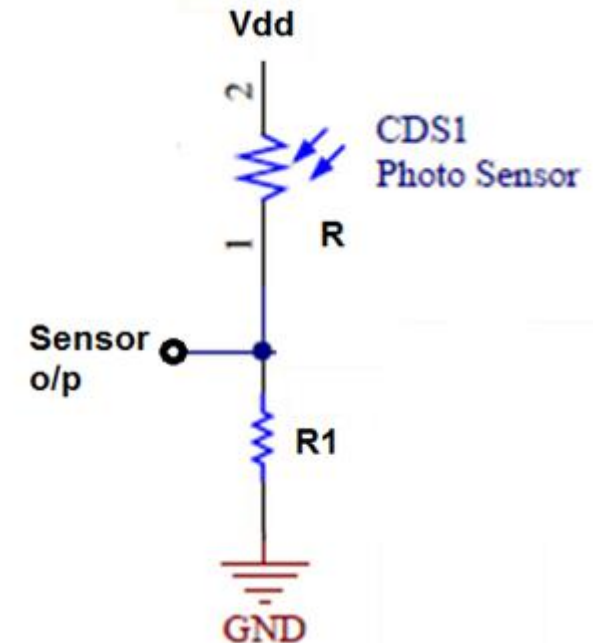


Interfacing with Micro-Controller

- Arduino (I/O pin) can only read a voltage level
 - Using ADC (Function: analogread) to convert into digits

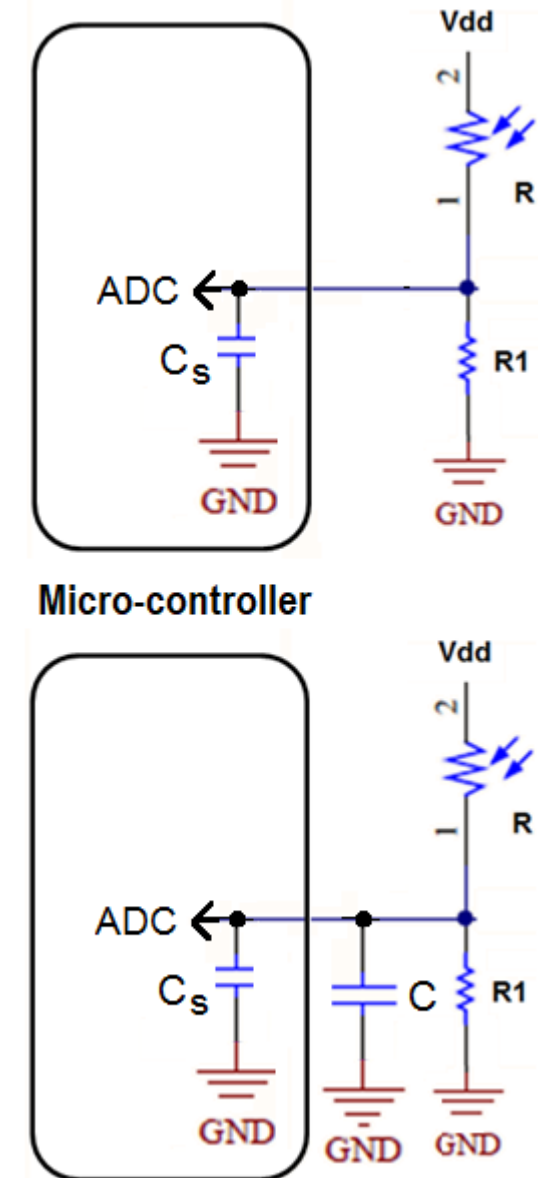


- Resistance $\rightarrow$ Voltage (using voltage divider)
- Sensor o/p = $V_{dd} \frac{R1}{(R+R1)}$
- When R increases (decreases),
 - sensor o/p decreases (increases), but it is not a linear map



Additional Hardware Design

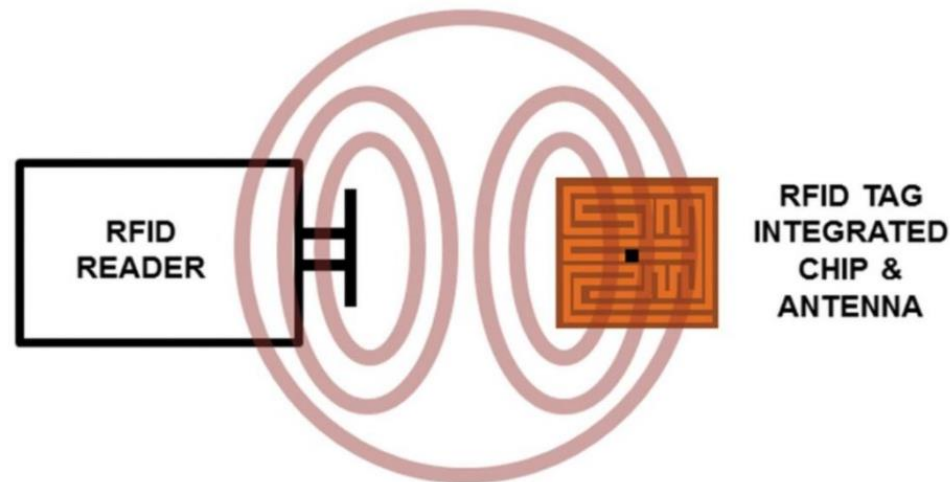
- Problem with internal impedance of the ADC [There is a small sampling cap C_s , in pF, which needs to be charged up in an ADC operation]
- If R and $R1$ are large, current goes to charge C_s is too small, introducing errors to ADC output
- Solution: Add a capacitor C (eg. 0.1 μ F) as a charge reservoir
 - error can be reduced
 - But need to reduce sampling frequency



- Environment Measurement
- Tracking a person
- Buttons
- ...

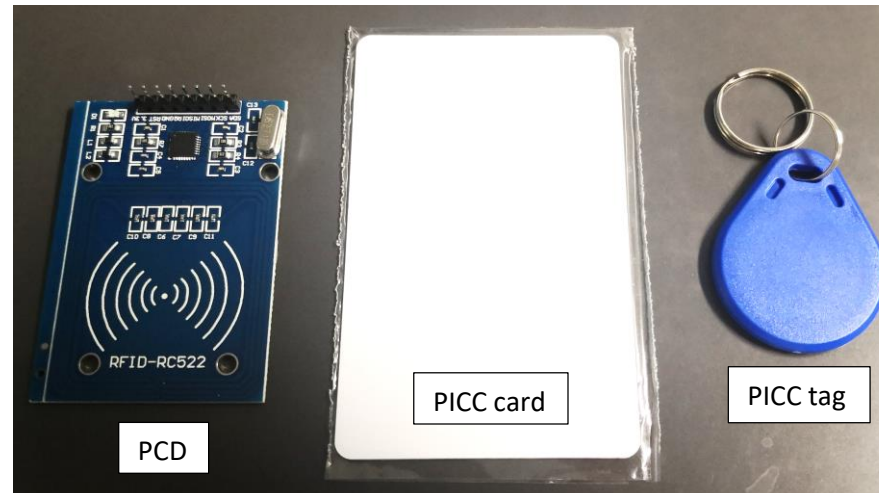
Radio Frequency Identification

- A contactless communication using radio waves, working in a very short range
- Passive and Active
- A Passive RFID: Has an antenna to receive RF waves; and circuitry (IC) to process/store data, as well as modulating and demodulating RF received/sent by the antenna



Passive RFID Modules

- Part Number: MFRC522
- A RFID reader (PCD): to read/write the passive RFID card (PICC card or tag).
- Modulating frequency 13.56MHz
- SPI Interface



Talk about SPI later

How It Works

- PCD transfers energy to PICC by emitting electromagnetic waves through the air.
- PICC uses RF energy to charge up.
- PICC receives command/data signal and responds accordingly.
- PCD receives PICC response and process accordingly, say sends to a host computer or external devices through its control lines.

Storage in PICC

- 1K memory
- Divided into 16 sectors
- Each sector has 4 blocks
- Each block has 16 byte
- Some blocks for special data (we don't use them)

Sector	Block	Byte Number within Block																Description
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	[0] 0																	Manufacturer Block
	[1] 1																	Data Block
	[2] 2																	Data Block
	[3] 3	KeyA						Access Bits				KeyB						Sector Trailer 0
1	[0] 4																	Data Block
	[1] 5																	Data Block
	[2] 6																	Data Block
	[3] 7	KeyA						Access Bits				KeyB						Sector Trailer 1
...	[0]																	Data Block
	[1]																	Data Block
	[2]																	Data Block
	[3]	KeyA						Access Bits				KeyB						Sector Trailer ...
...	[0]																	Data Block
	[1]																	Data Block
	[2]																	Data Block
	[3]	KeyA						Access Bits				KeyB						Sector Trailer ...
14	[0] 56																	Data Block
	[1] 57																	Data Block
	[2] 58																	Data Block
	[3] 59	KeyA						Access Bits				KeyB						Sector Trailer 14
15	[0] 60																	Data Block
	[1] 61																	Data Block
	[2] 62																	Data Block
	[3] 63	KeyA						Access Bits				KeyB						Sector Trailer 15

Table 2.1 MIFARE Classic 1K - Memory Organization

- Mifare 1K EEPROM is arranged of 16 sectors. Each sector has 4 blocks and each block has 16-byte.
- Block 0 is a special read-only data block keeps the manufacturer data and the UID of the tag.
- Sector Trailer block, the last block of the sector, holds the access conditions and two of the authentication keys for that particular sector.

Information in Memory

- The first block of the first sector (Block 0 of Sector 0) is called manufacturer block. It is read-only and holds UID and IC manufacturer data.

Byte Number within Block 0/Sector0															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NUID[0..3]				Manufacturer Data											

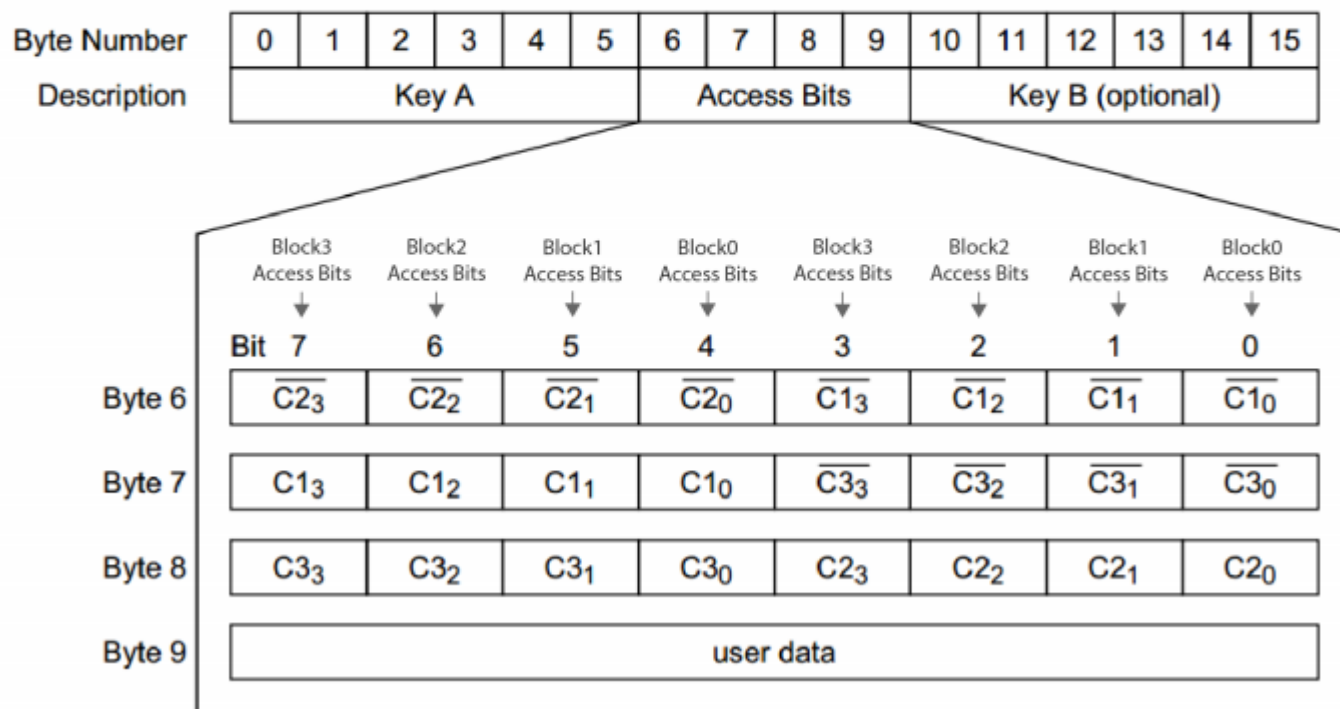
- To read/write the data in the PICC card, authentication is needed. There are two keys, key A (for read) and key B (for write). It is given in the sector trailer of each sector. At chip delivery, all keys are set to 0x FF FF FF FF FF FF, and the bytes 6,7,8, are set to 0xFF0780.

Byte Number within the Sector Trailer Block															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
KeyA						Access Bits				KeyB (Optional)					

Access Bits

Access Bits	Valid Commands	Block	Description
C13,C23,C33	read/write	3	Sector Trailer
C12,C22,C32	read/write/increment/decrement	2	Data Block
C11,C21,C31	read/write/increment/decrement	1	Data Block
C10,C20,C30	read/write/increment/decrement	0	Data Block

Table 4 Access Bits



Access Bits			Access Conditions for							Remark
			KeyA		Access Bits		KeyB			
C1	C2	C3	Read	Write	Read	Write	Read	Write		
0	0	0	Never	KeyA	KeyA	Never	KeyA	KeyA	KeyB may be read[1]*	
0	1	0	Never	Never	KeyA	Never	KeyA	Never	KeyB may be read[1]*	
1	0	0	Never	KeyB	KeyA/B	Never	Never	KeyB		
1	1	0	Never	Never	KeyA/B	Never	Never	Never		
0	0	1	Never	KeyA	KeyA	KeyA	KeyA	KeyA	KeyB may be read, transport configuration[1]*	
0	1	1	Never	KeyB	KeyA/B	KeyB	Never	KeyB		
1	0	1	Never	Never	KeyA/B	KeyB	Never	Never		
1	1	1	Never	Never	KeyA/B	Never	Never	Never		

Table 4.1 Access Conditions for the Sector Trailer Block

Access Bits			Access Conditions for				Application
C1	C2	C3	Read	Write	Increment	Decrement Transfer Restore	
0	0	0	KeyA/B	KeyA/B	KeyA/B	KeyA/B	transport configuration[1]*
0	1	0	KeyA/B	Never	Never	Never	read/write block[1]*
1	0	0	KeyA/B	KeyB	Never	Never	read/write block[1]*
1	1	0	KeyA/B	KeyB	KeyB	KeyA/B	value block with recharging[1]*
0	0	1	KeyA/B	Never	Never	KeyA/B	value block non-rechargeable[1]*
0	1	1	KeyB	KeyB	Never	Never	read/write block[1]*
1	0	1	KeyB	Never	Never	Never	read/write block[1]*
1	1	1	Never	Never	Never	Never	read/write block

Table 4.2 Access Conditions for the data blocks

What does it mean?

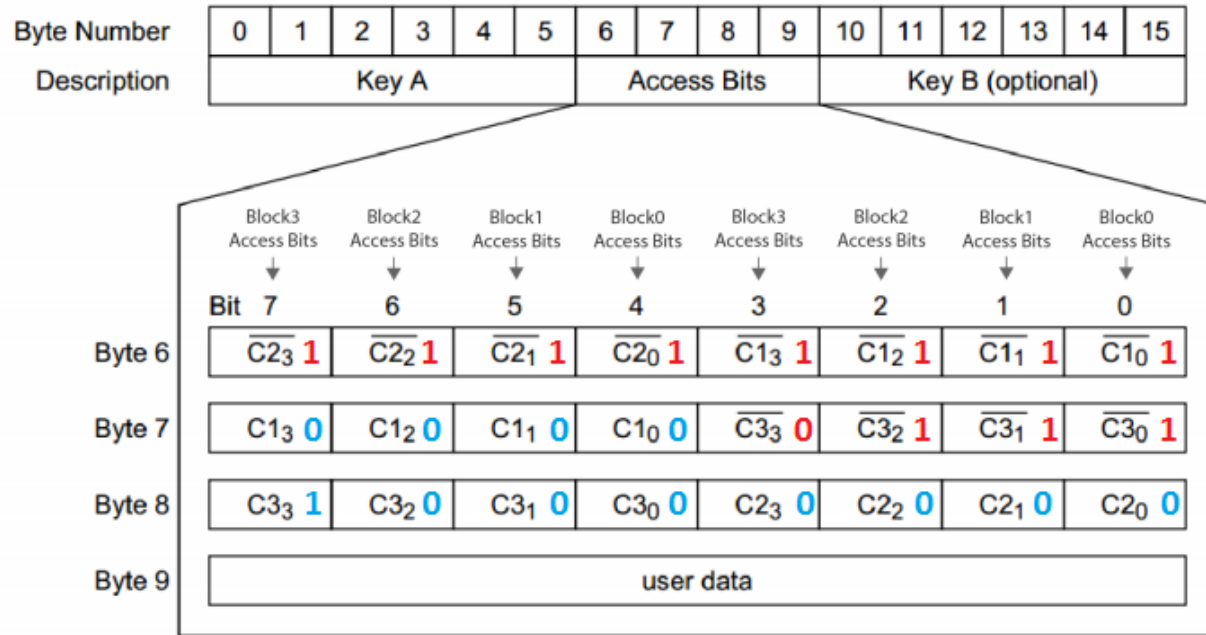
1	7	00 00 00 00 00 00	FF 07 80 69	FF FF FF FF FF FF	[0 0 1]
	6	00 00 00 00 00 00	00 00 00 00	00 00 00 00 00 00	[0 0 0]
	5	00 00 00 00 00 00	00 00 00 00	00 00 00 00 00 00	[0 0 0]
	4	00 00 00 00 00 00	00 00 00 00	00 00 00 00 00 00	[0 0 0]

C13, C23, C33 = 001

C12, C22, C32 = C11, C21, C31 = C10, C20, C30 = 000

Access Bits	Valid Commands	Block	Description
C13,C23,C33	read/write	3	Sector Trailer
C12,C22,C32	read/write/increment/decrement	2	Data Block
C11,C21,C31	read/write/increment/decrement	1	Data Block
C10,C20,C30	read/write/increment/decrement	0	Data Block

Table 4 Access Bits



Access Bits			Access Conditions for							Remark
			KeyA		Access Bits		KeyB			
C1	C2	C3	Read	Write	Read	Write	Read	Write		
0	0	0	Never	KeyA	KeyA	Never	KeyA	KeyA	KeyB may be read[1]*	
0	1	0	Never	Never	KeyA	Never	KeyA	Never	KeyB may be read[1]*	
1	0	0	Never	KeyB	KeyA B	Never	Never	KeyB		
1	1	0	Never	Never	KeyA B	Never	Never	Never		
0	0	1	Never	KeyA	KeyA	KeyA	KeyA	KeyA	KeyB may be read, transport configuration[1]*	
0	1	1	Never	KeyB	KeyA B	KeyB	Never	KeyB		
1	0	1	Never	Never	KeyA B	KeyB	Never	Never		
1	1	1	Never	Never	KeyA B	Never	Never	Never		

Table 4.1 Access Conditions for the Sector Trailer Block

Example

- UID: BA 9C A9 1A

```

Card UID: BA 9C A9 1A
PICC type: MIFARE 1KB
Authenticating using key A...
Current data in sector:
    0      3  00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
    2      2  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
    1      1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
    0      0  BA 9C A9 1A 95 08 04 00 62 63 64 65 66 67 68 69 [ 0 0 0 ]

Current data in sector:
    1      7  00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
    6      6  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
    5      5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
    4      4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
  
```

The Access Bits (bytes 6-8) 0xFF0780 implies

- Key A never be read, but can write (change) itself. So, it shows 00 00 00 00 00 00.
- Key A can read/write Access Bits and KeyB
- Key A/B is allowed to read/write/increment/decrement for the data blocks
- To read the data, firstly need to authenticate using key A, and the default key is FF FF FF FF FF FF.

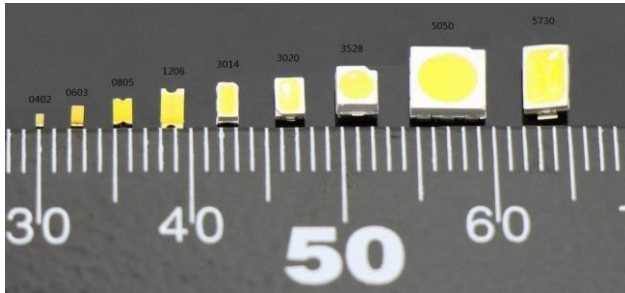
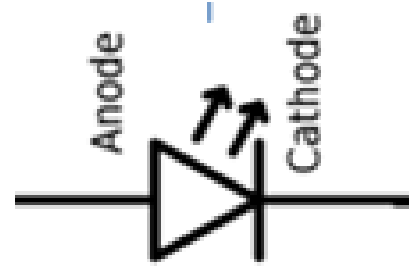
Usage

- UID can help us to identify the ID
- 1K memory, can read/write data in blocks of sectors (eg. Block 0-2 of Sector 1-15)

Display

LED

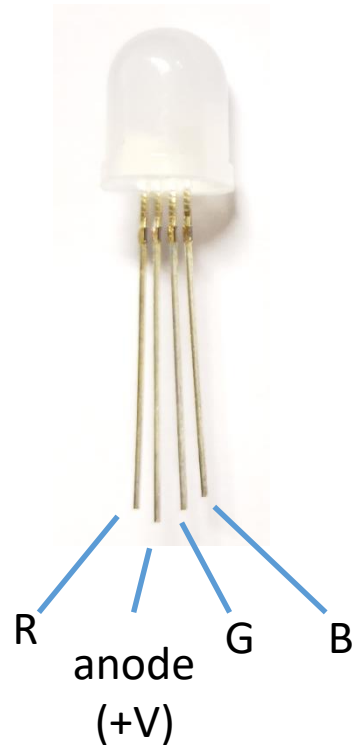
- LED – Light emitting diode
- To light up, LED is forward biased
 - Voltage at anode is made higher than the cathode (forward voltage)
 - Current should be sufficient to light up the LED, but should not be too high



Different packages

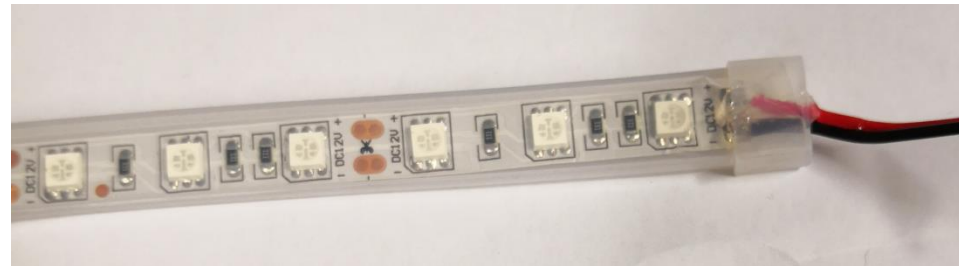
Different Package

- RGB LED: Packing red/green/blue LEDs together into a single one

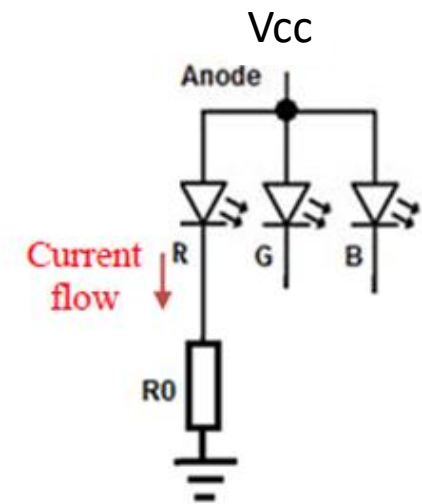


Common anode RGB LED

- LED Strip Light: Packing a series of LEDs in parallel



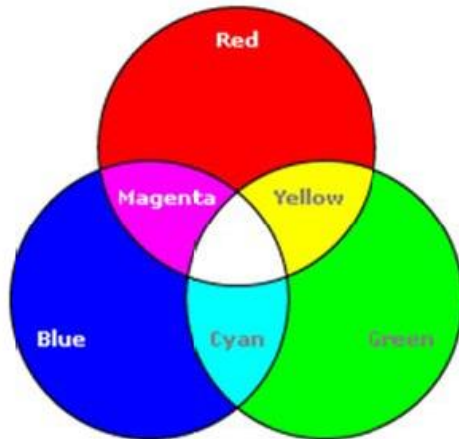
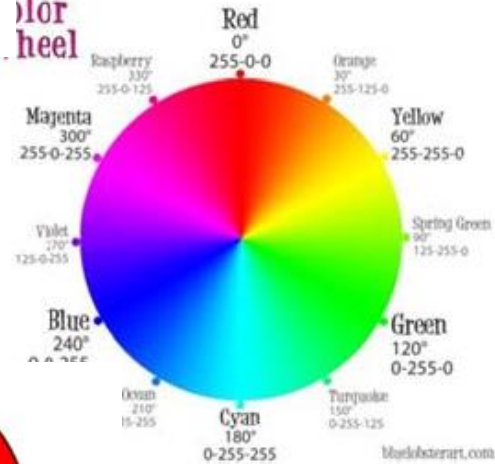
Display: RGB-LED



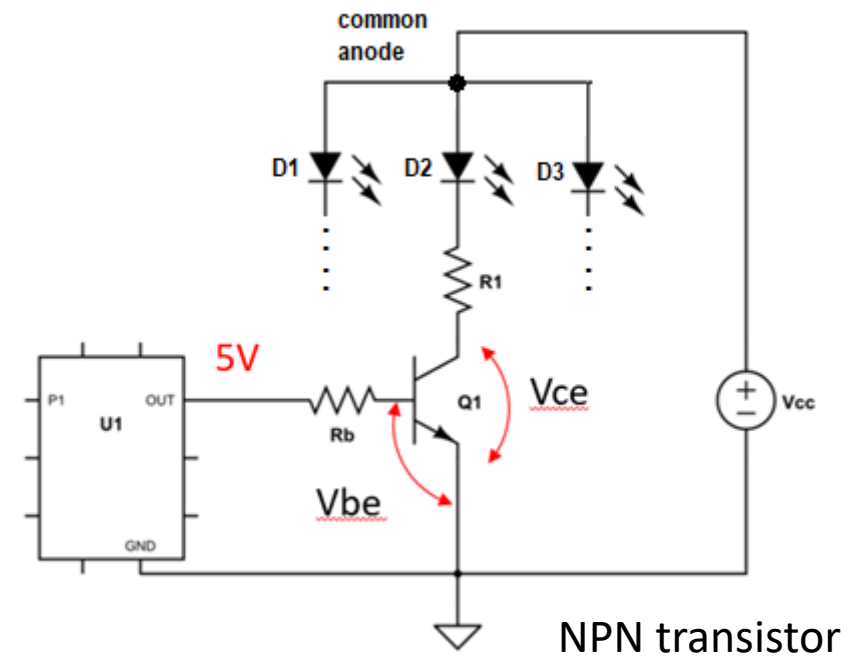
R
A
G
B
(anode)

Common anode: Longest lead is the Anode (A)

GB
lor
heel



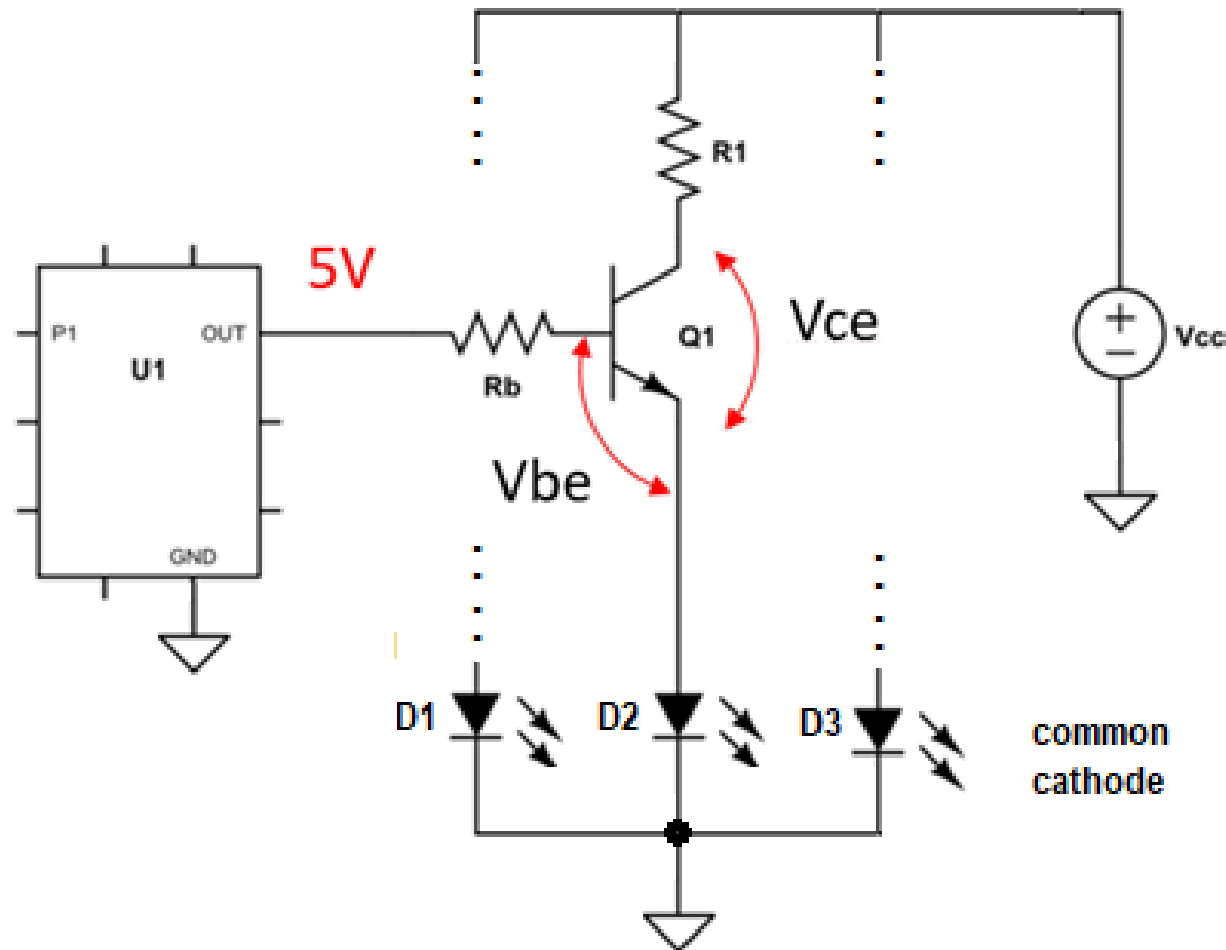
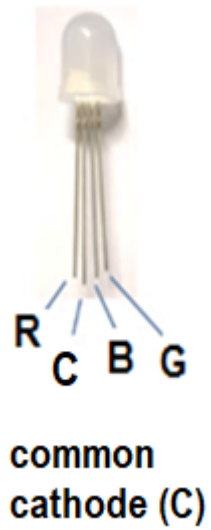
- To turn on, give high voltage to anode (A) and low voltage to R / G / B.



- Turn on a combination of R/G/B to get different colors.

Display: RGB-LED

- Common Cathode



E-Paper (Electronic Paper)

- First developed in 1974

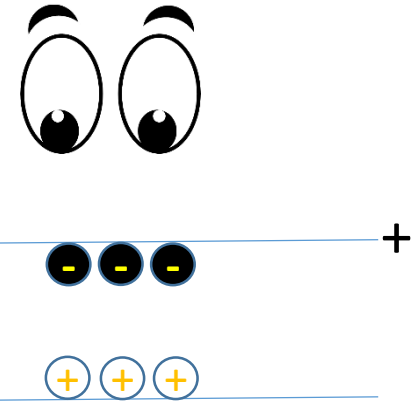
<https://sites.google.com/site/fl1electronicpaper/Home/an-interview-with-nick-sheridon>

- Flexible, light weight, high visibility under light
- Observable even after power is off

Model number: GDEH0154D67 (200 x 200 dots)

- Bus: Serial Peripheral Interface (SPI)
- Control display by sending data/command to E-paper via the SPI bus
- Allow partial update

Talk about SPI later

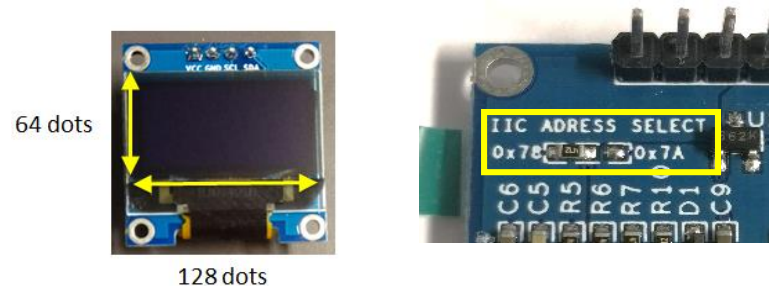


OLED

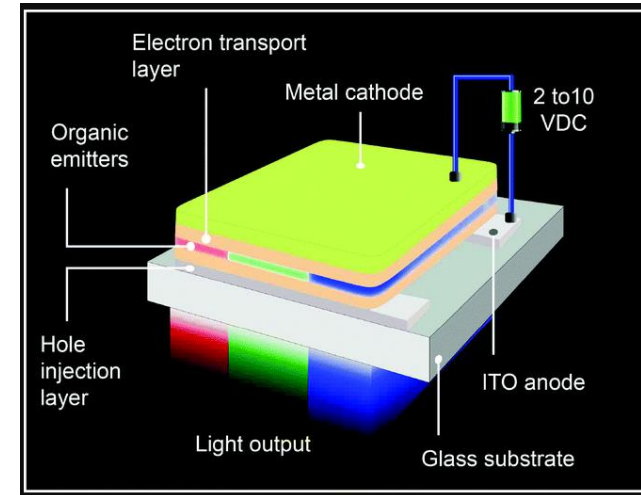
- Organic thin films between conductors
- Light emitted when current applied
- Good visibility even in dark
- Always need power

Model number: SSD1306 (128 x 64 dots)

- Bus: Inter-Integrated Circuit (**I2C**)
- Control display by sending data/command via I2C
Can change the hard-wired address with soldering skill), and control 2 OLEDs with the same bus



Talk about SPI later



DOI: [10.1039/C3CS60449G](https://doi.org/10.1039/C3CS60449G) (Review Article) *Chem. Soc. Rev.*, 2014, **43**, 3259-3302



What can we do with epaper/OLED display?

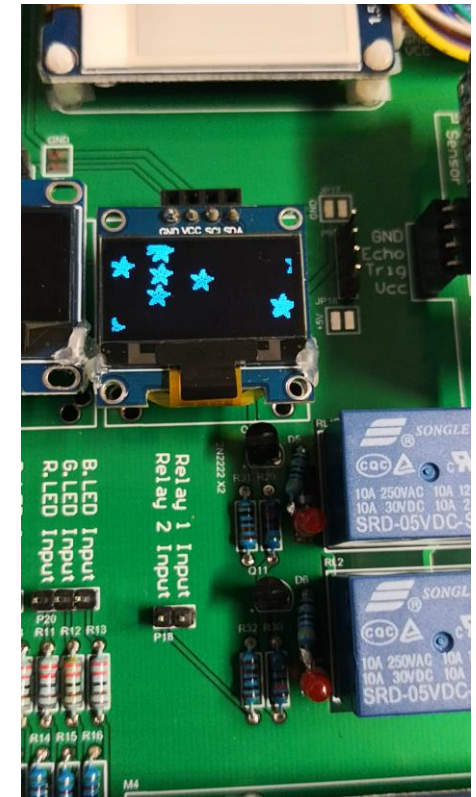
- Write texts (may control font / font size)
- Draw lines, rectangles, circles, pixels
- Load image to display
 - Create bitmap image data; stored in memory; load to display
- Animation: draw image -> delete/refresh -> draw modified image

How to create and upload image?

- <https://educ8s.tv/arduino-bitmap-graphics-tutorial/>
1. Draw an image (using Paint), monochrome, with proper size
 2. Convert an image (bmp) file to codes: eg. online program
<https://javl.github.io/image2cpp/>
 3. Copy the code and store in the program (as in memory)
 4. Use image drawing command
- Eg. `display.drawImage(appleimage,0,0,152,152,false,false,true);`

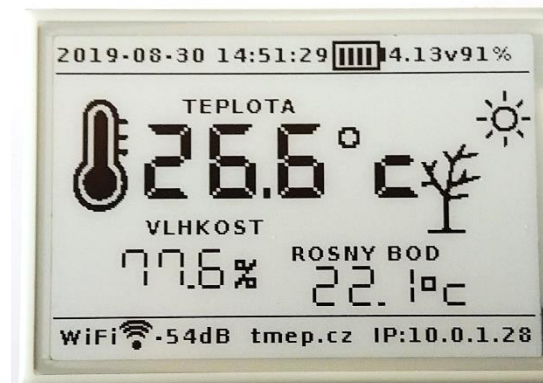
How to create an animation?

- Create a series of images
- Display one-by-one
- Note: Limitation in memory size



Examples of Applications

- Signaling (eg. RGB LED)
- Control Panel
- Environmental data
- ...

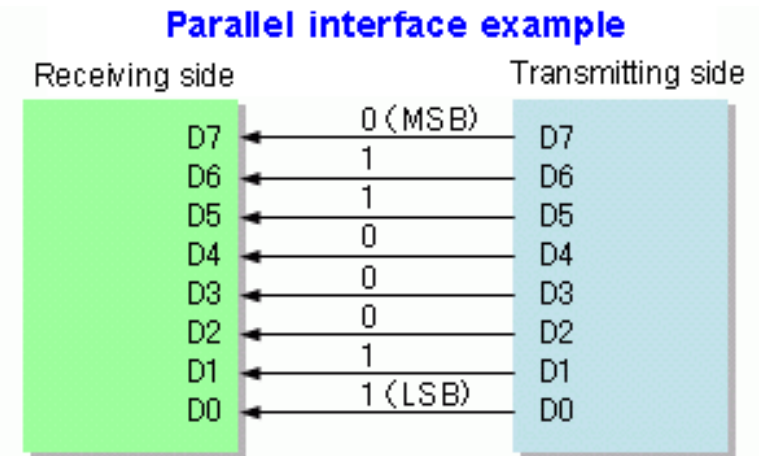
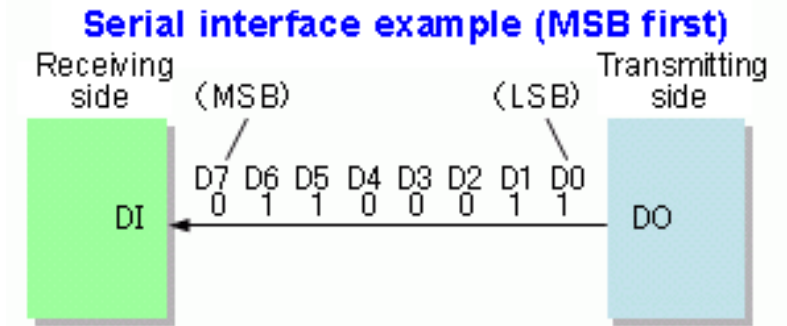


Communication I:

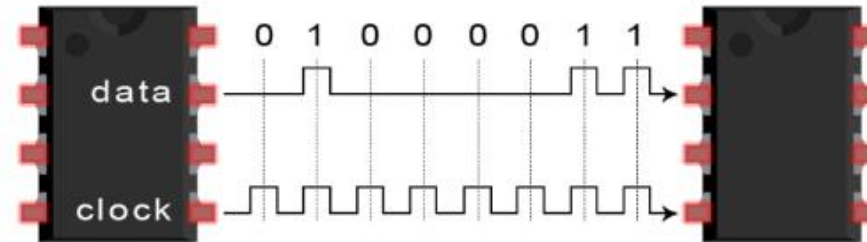
Wired Communication

- I'll talk about different types of communications
- Wired Communications
 - Serial Communications
 - Asynchronous: UART
 - Synchronous: SPI, I2C
- Wireless Communications
 - Wifi
 - ZigBee
 - Bluetooth

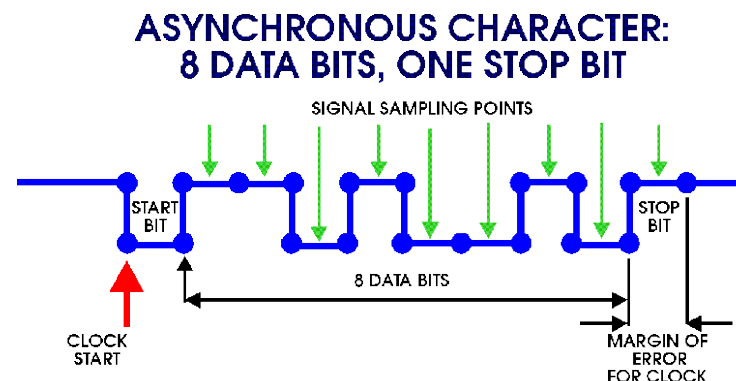
- Data in bytes (1 byte = 8 bits)
(We have a 8-bit controller in Arduino)
- Serial communication
 - Bits are sent one by one (eg. 8 bits needs 8 cycles), hence slow
 - Require very few pins (wires) for communications
- Parallel communication:
 - Several bits (eg. a word of 8 bits) can be sent in one cycle, hence fast
 - Require many pins (wires)



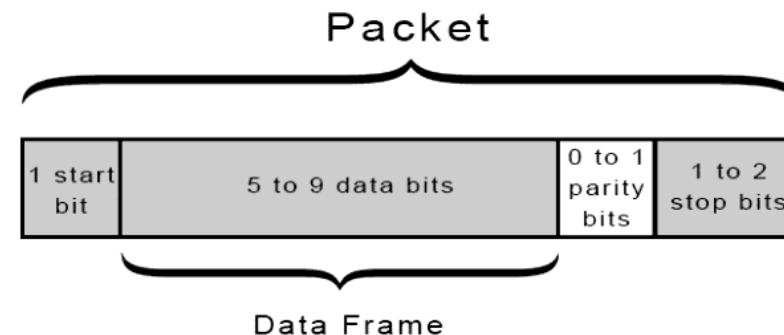
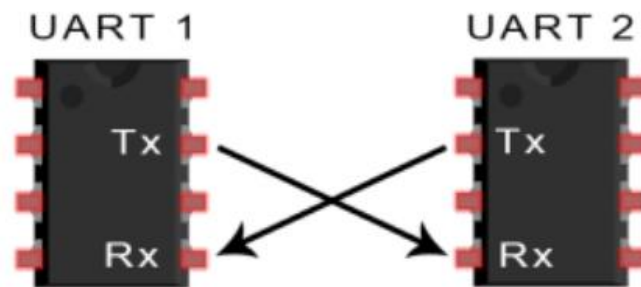
- **Synchronous:** A clock to synchronize between transmitter and receiver so that the receiver can receive the data correctly.



- **Asynchronous:**
 - No synchronous clock
 - Add start and stop bits in the data packet being transferred to define the beginning and end of the data packet. With such, the receiver can know when to start reading the bits.



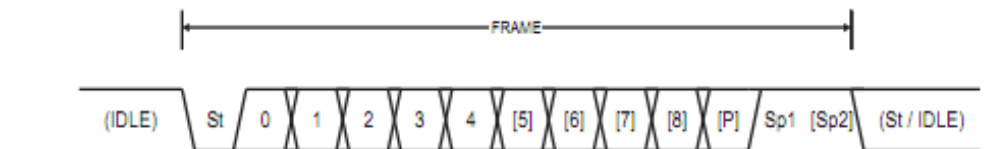
- UART: **U**niversal **A**synchronous **R**eceiver/**T**ransmitter
- A kind of serial communications (Very commonly used)
- General flow:
 - 2 UARTs: one acts as transmitter and one is receiver
 - Transmitter: The data (in bytes) is firstly converted into a serial form (bits), then transmitted one by one at Tx pin.
 - Receiver: After received the serial data one-by-one from Rx pin, the receiver converts the serial data back as byte.



More Details

- The speed of data transfer (called baud rate, unit: bits per second or bps) must be the same between transmitter and receiver.
 - Typical 9600 bps, and can be up to 115200 bps
- Need additional bits so that the receiver knows where the data starts and ends
 - **Start bit** and **stop bit** are added
- Additional bit for error check, called **parity**
 - even number of 1's in the data frame => parity bit = 0
 - odd number of 1's in the data frame => parity bit = 1

Figure 19-4. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Built-In Communications to PC

- Provide a means for communication between the Arduino board and a PC
- Via USB cable
- Communication between PC and Arduino board (via USB) is based on a built-in serial monitor. (Click the serial monitor button in the toolbar)
 - Select the same baud rate used in the program
 - The RX and TX LEDs on the board will flash when data is being transmitted via the USB connection to the computer
- **Serial** class is used in programming.

Asynchronous Serial Communication

Example

```
BlinkWithoutDelay | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help

BlinkWithoutDelay $

// constants won't change:
const long interval = 1000;          // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, if the difference
  // between the current time and last time you blinked the LED is bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
      Serial.println("ON");
    } else {
      ledState = LOW;
      Serial.println("OFF");
    }

    // set the LED with the ledState of the variable:
  }
}
```

BlinkWithoutDelay | Arduino 1.8.13 (Windows Store 1.8.42.0)

File Edit Sketch Tools Help

- Auto Format Ctrl+T
- Archive Sketch
- Fix Encoding & Reload
- Manage Libraries... Ctrl+Shift+I
- Serial Monitor Ctrl+Shift+M**
- Serial Plotter Ctrl+Shift+L
- WiFi101 / WiFiNINA Firmware Updater
- Board: "Arduino Micro" >
- Port: "COM9 (Arduino Micro)" >
- Get Board Info
- Programmer: "ArduinoISP" >
- Burn Bootloader

COM9

15:10:04.973 -> OFF
15:10:05.948 -> ON
15:10:06.957 -> OFF
15:10:07.963 -> ON
15:10:08.967 -> OFF
15:10:09.972 -> ON

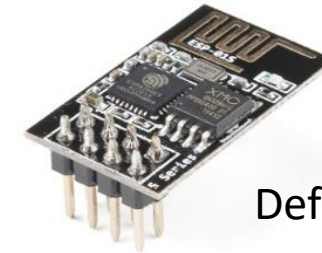
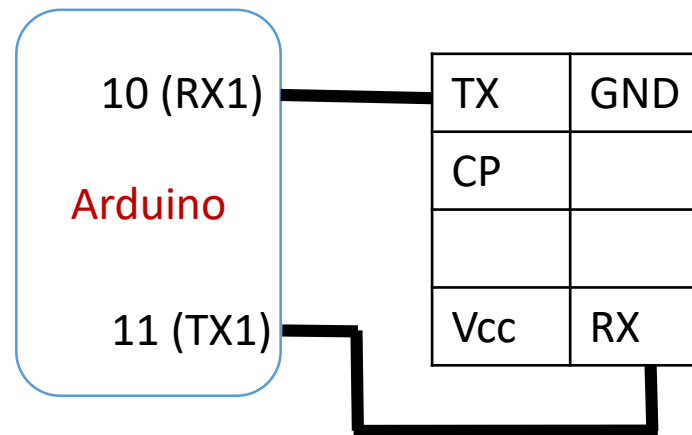
☒ Autoscroll ☒ Show timestamp Newline 115200 baud Clear output

Note on Arduino and Communication Module

- Arduino boards have specific RX and TX pins to support Serial communications
- If other pins are used, a library **SoftwareSerial.h** can be used, to replicate the functionality of the hard-wired RX and TX pins
- This also allows the communication between Arduino board and other devices
- Baud rate must be the same on both sides (as set in software and hardware)

Example

- Set up communication between Arduino board and a Wifi module



Default baud rate = 19200bps

- Initialization the ESP8266

```
SoftwareSerial Serial1(10, 11); // Define RX, TX pin  
#define ESP_BAUDRATE 19200
```

```
// initialize serial for ESP module. The default baud rate of the ESP is set as 19200 bps  
setEspBaudRate(ESP_BAUDRATE);
```

```
// initialize ESP module  
WiFi.init(&Serial1);
```

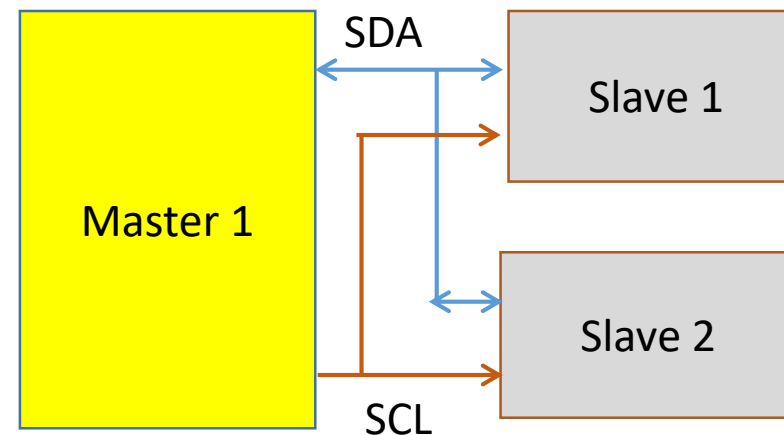
What is I2C (I Square C)?

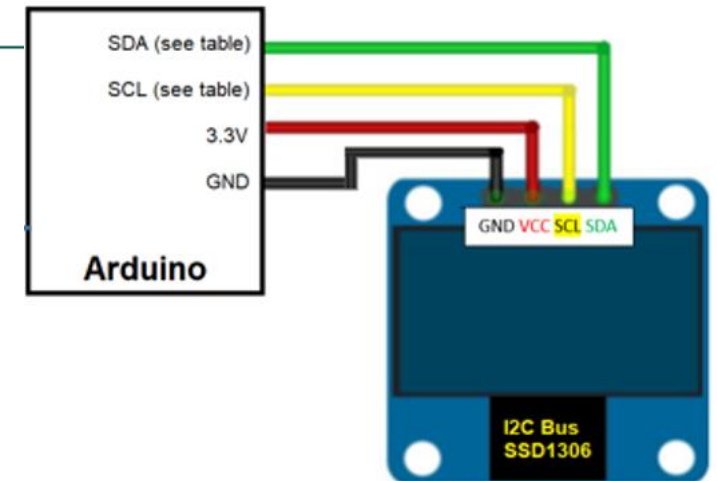
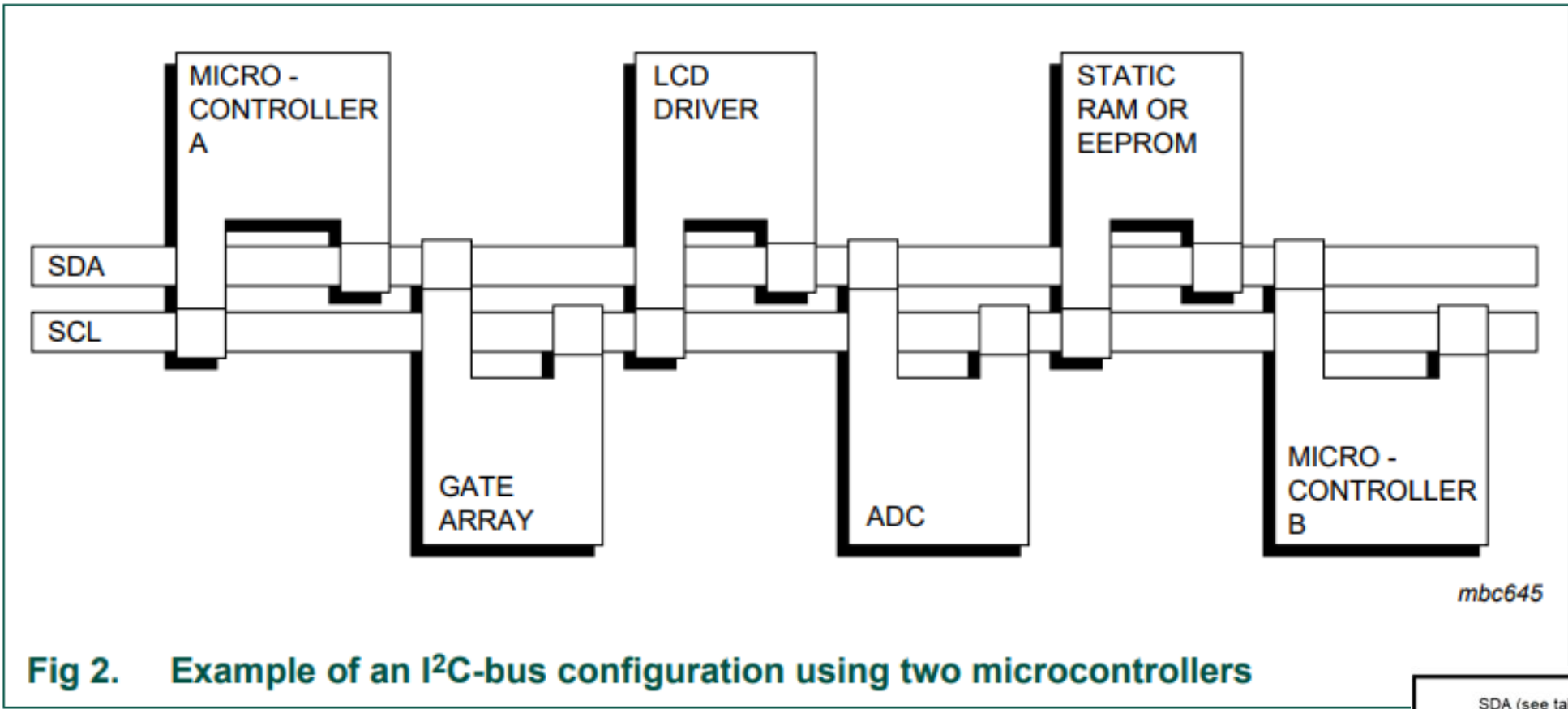
- I2C : Inter-Integrated Circuit
- A **synchronous serial communication**, multi-master, multi-slave, wired communications
- Only required two signal wires: SCL (Serial Clock) and SDA (Serial Data)
 - SCL is the clock signal, and SDA is the data signal.

Master and Slave in I2C

- The device that initiates a transaction on the I²C bus is called the master. The master controls the clock signal. A device being addressed by the master is called a slave.
- Using just 2 bus lines (SDA and SCL), a master can connect to multiple slaves (up to 1008)
- We can also have multiple masters / multiple slaves.

Example of 1 master and 2 slaves

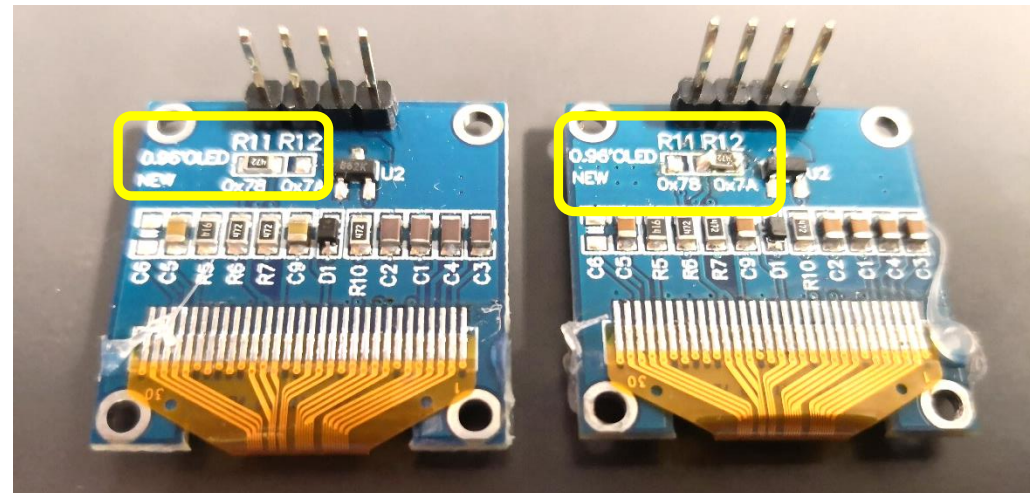




<https://www.nxp.com/docs/en/application-note/AN10216.pdf>
<https://www.nxp.com/docs/en/user-guide/UM10204.pdf>

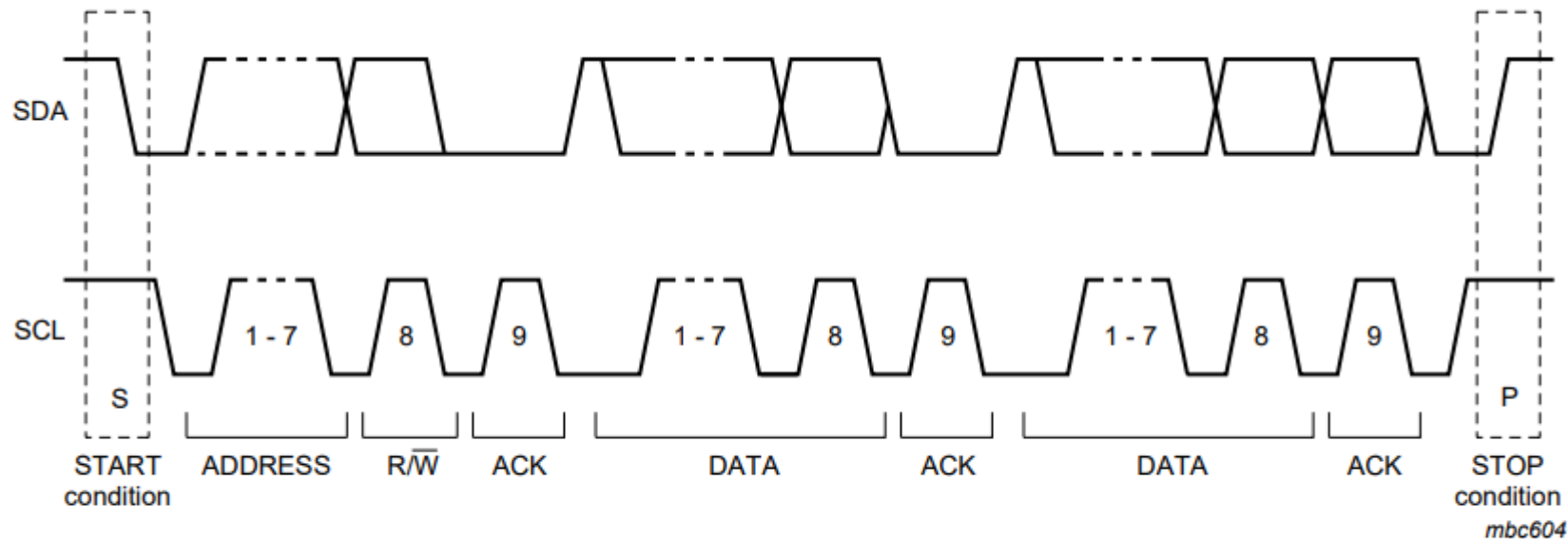
Address of Slave Devices

- How to differentiate different devices if they connected to same master?
 - Each I²C-compatible hardware slave device comes with a **predefined device address**. Eg. our OLED is 60d (3Ch)
 - To allow multiple identical devices connecting to one I2C bus, user can alter some lower bits of the address.
 - For example, our OLED's address can be modified by re-soldering a resistor at the back.



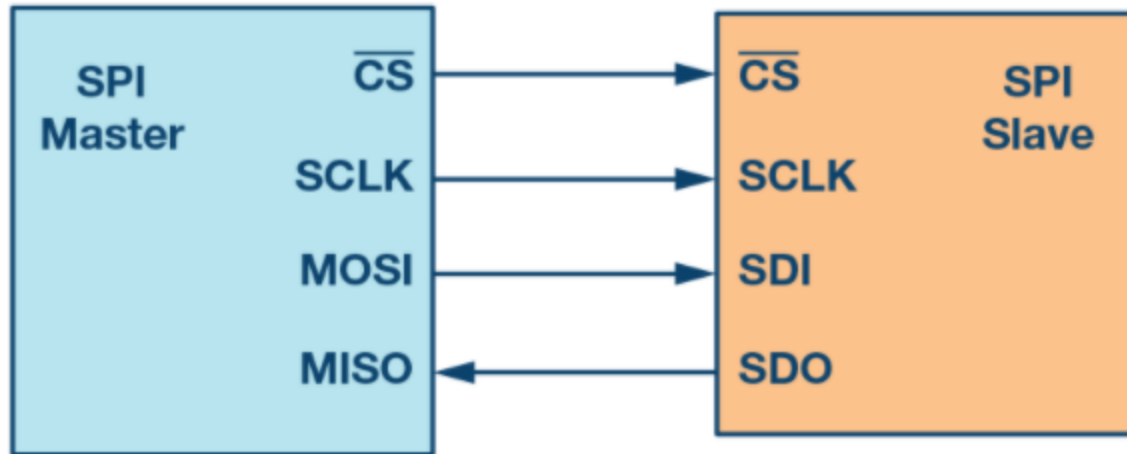
How I2C Works?

- The master transmits the device address of the intended slave at the beginning of every transaction.
- Each slave is responsible for monitoring the bus, but only the slave with that address will respond.



A complete data transfer

- Serial Peripheral Interface (SPI)
- Synchronous, full duplex master-slave based interface
- 3-wire or **4-wire**



- Chip Select (CS)
- Clock (CLK)
- Master out, slave in (MOSI)
- Master in, slave out (MISO)

<https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>

<https://community.nxp.com/t5/MQX-Software-Solutions-Knowledge/Introduction-to-SPI-interface/ta-p/1104630>

- Support higher clock frequency than I2C
- One master and multiple slave (Use CS (low) to select the slave)
- MOSI and MISO are data lines

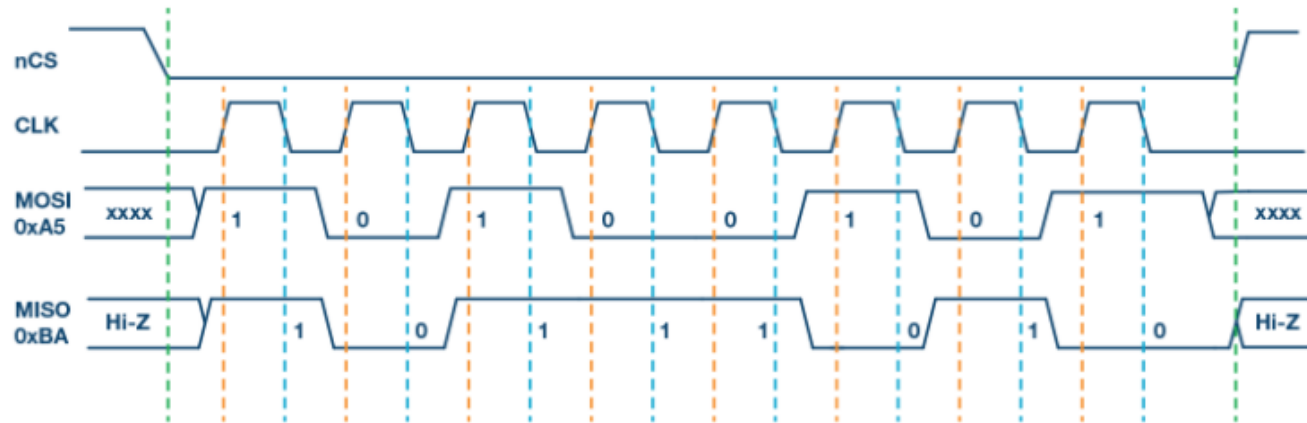
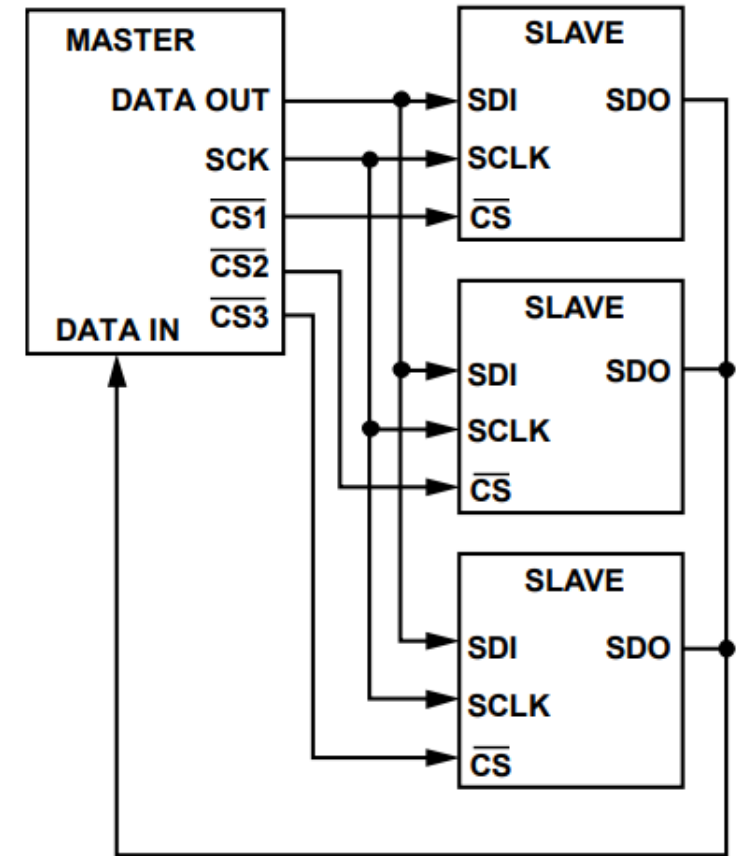
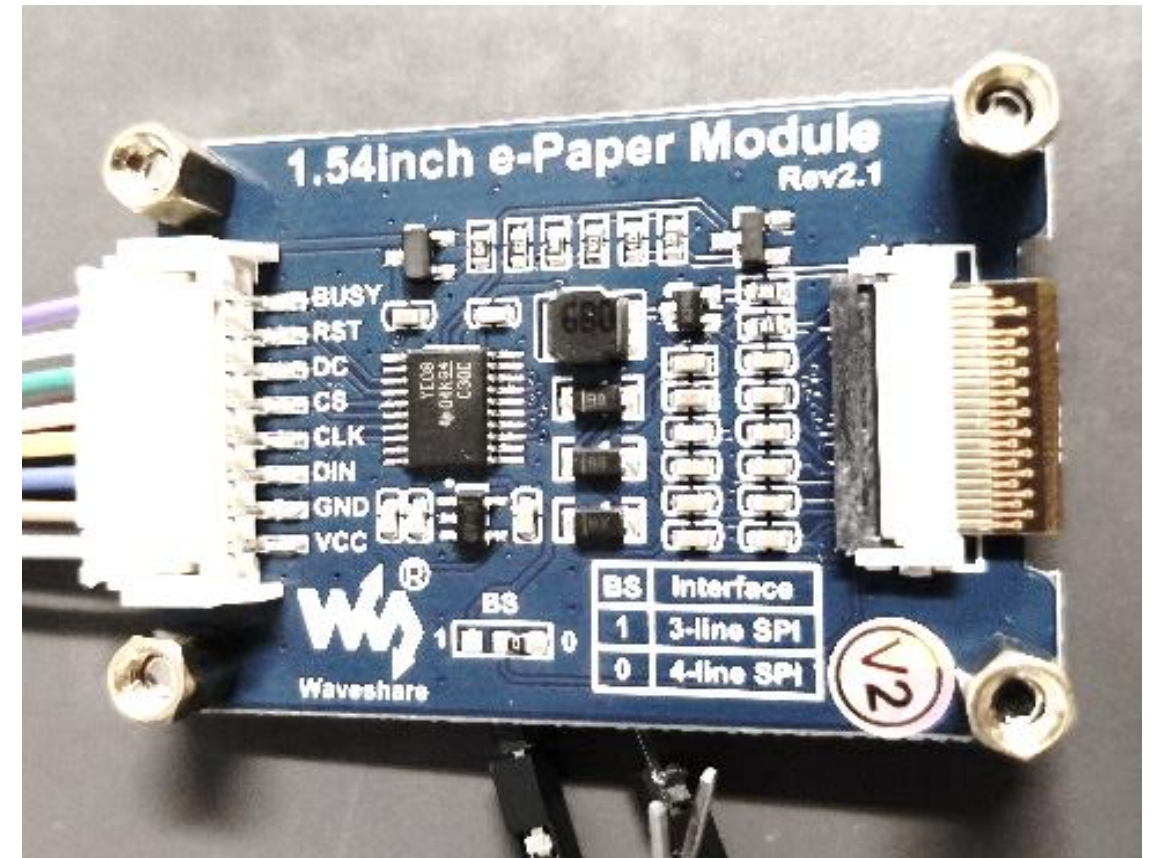


Figure 2. SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.



Example: Epaper driver

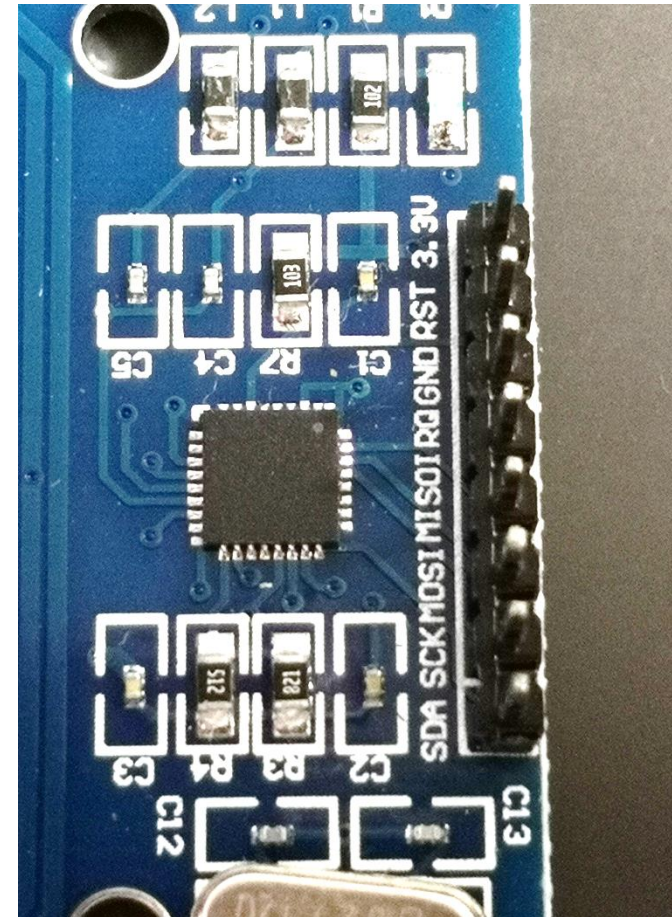
GDEW0154T8	Arduino Mega
VCC	3.3V
GND	GND
DIN=MOSI	MOSI(51)
CLK=SCK	SCK(52)
CS=SS	10
DC	9
RST	8
BUSY	7



Additional control pins: DC (Data/Command pin), RST (Reset pin) and BUSY (Busy detect pin)

Example: RFID PCD

PCD	Arduino Mega	Arduino Micro
SDA = SS	53	SS (17)
SCK	SCK (52)	SCK (15)
MOSI	MOSI (51)	MOSI (16)
MISO	MISO (50)	MISO (14)
IRQ	-	-
GND	GND	GND
RST	5	5
3.3V	3.3V	3.3V



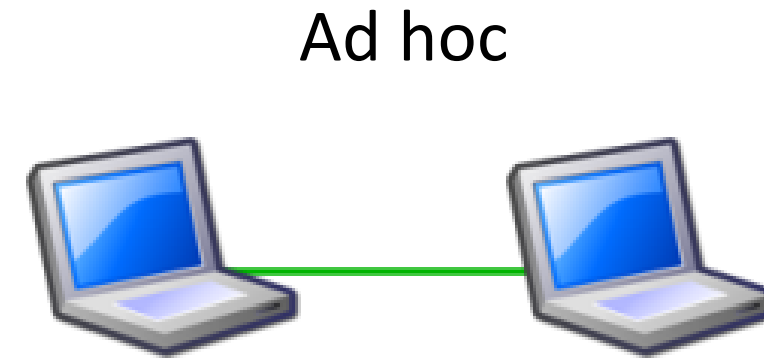
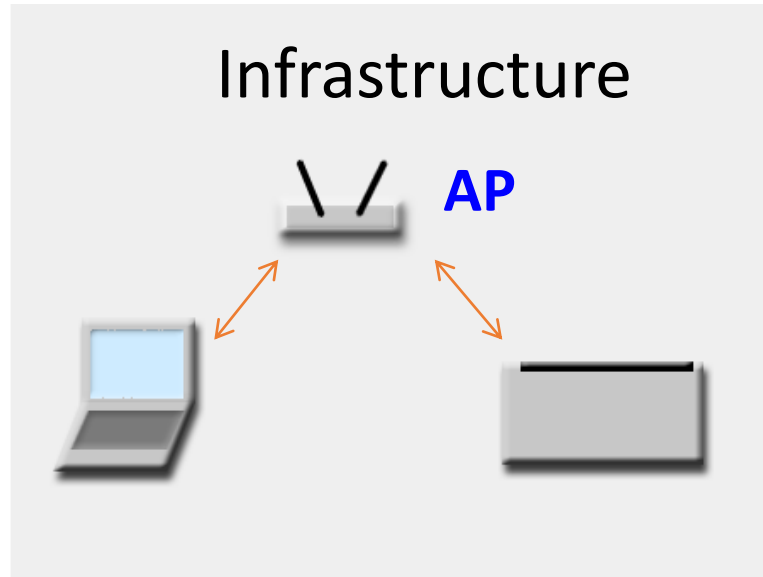
Additional control pins: RST (Reset pin) and IRQ (interrupt)

Communication II:

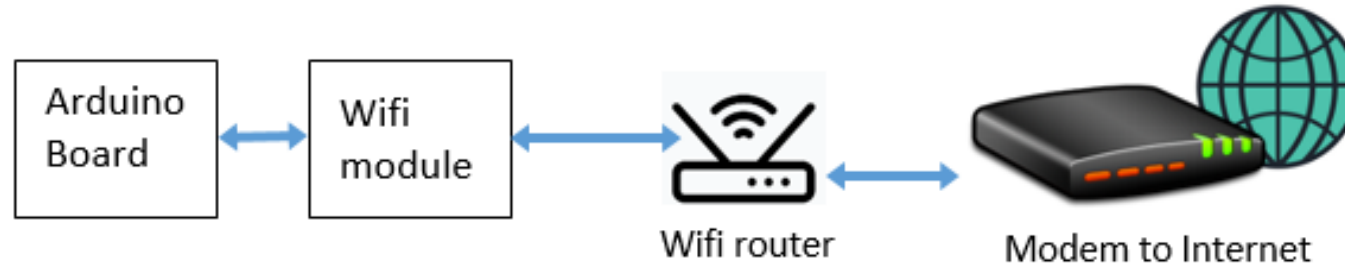
Wireless Communication

- **WiFi**: (Wireless Fidelity) is a technology for wireless local area networking (WLAN) with devices based on the **IEEE 802.11** standards.
- **Station** (STA): a device that has the capability to use the 802.11 protocol; or all components that can connect into a wireless medium in a network.
 - **Access Point (AP)**: normally wireless router, is a base station.
 - **Client**: mobile devices (e.g., laptops, smartphones) or non-portable devices (e.g., desktop computers with a wireless network interface).
- **SSID**: or Service Set Identifier
 - A unique ID that consists of 32 characters
 - To name a WLAN, and so you can connect to a particular one

- **Infrastructure**: Devices communicate through an Access Point
- **Ad hoc**: Devices transmit directly based on a peer-to-peer structure



- Router in each Lab -> Access Internet



- Wifi module
 - Part Number: ESP8266
 - Default BAUD: 19200 bps
- Connect to Router
 - SSID (Service Set Identifier)
 - Password
- Communications between Arduino and Wifi Module by wired communications (Serial Communications)

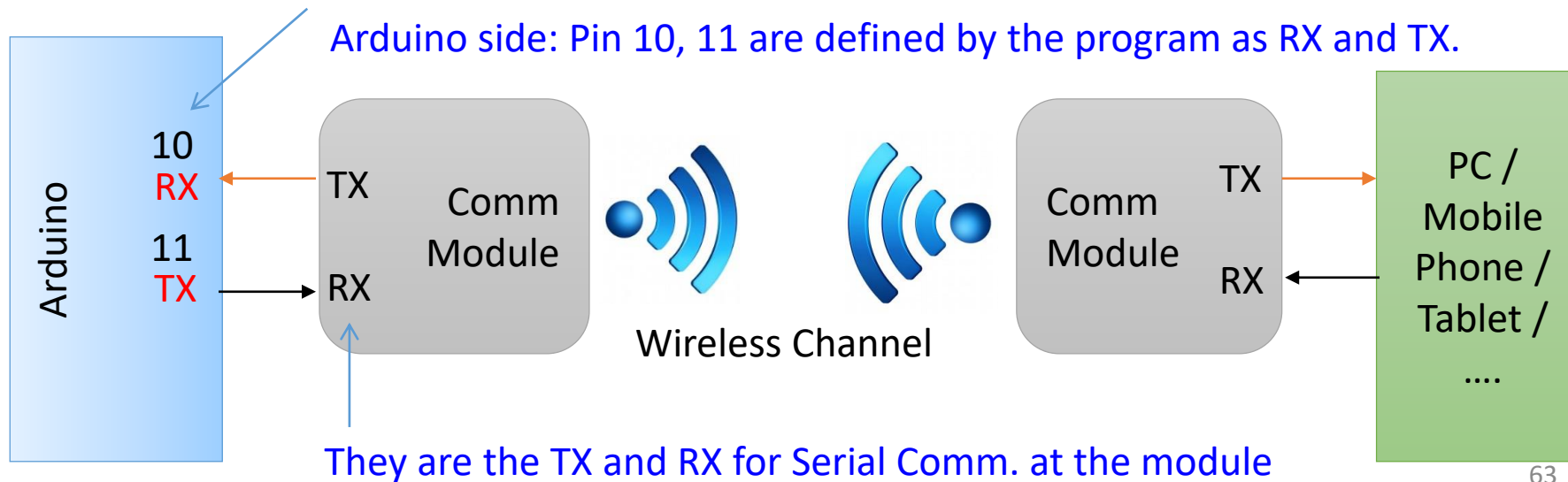
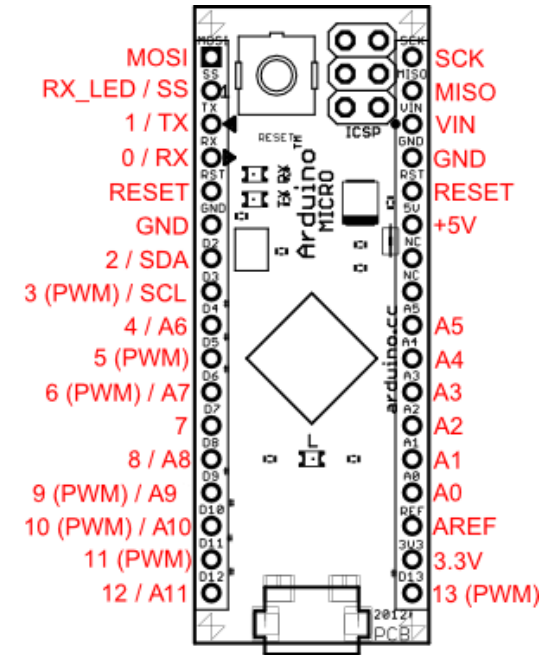


How WiFi being used ?

- Provide Internet access to enable message communications and promotions
- User locating
- Real time update / analysis (via cloud / server)

Wireless Communications Between Arduino and Other Devices

- An external wireless communication module is needed (of course, another side should also have the wireless communication capability)
- Only 8, 9, 10, 11, MISO, SCK and MOSI can be used as RX in Arduino Micro
- Learn two things:
 - Select and setup the modules
 - Use **SoftwareSerial** Library for programming



SoftwareSerial Library

- Allow serial communication (UART) on other digital pins of the Arduino (other than default TX and RX), using software to replicate the functionality

```
#include <SoftwareSerial.h>
```

← Include the library

```
const byte rxPin = 2;
```

```
const byte txPin = 3;
```

```
// set up a new serial object
```

```
SoftwareSerial mySerial (rxPin, txPin);
```

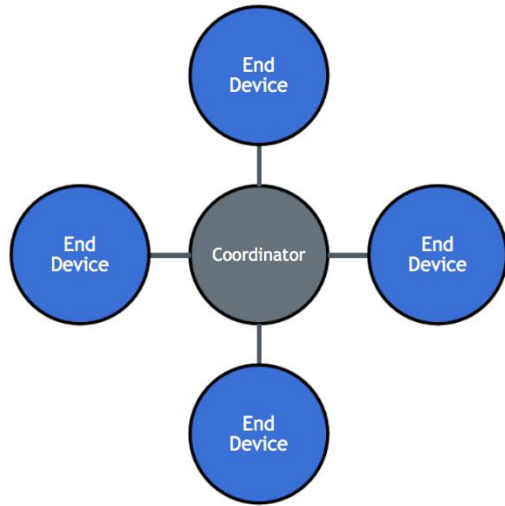
Create mySerial

- Pin 2 for receive data
- Pin 3 for transmit data

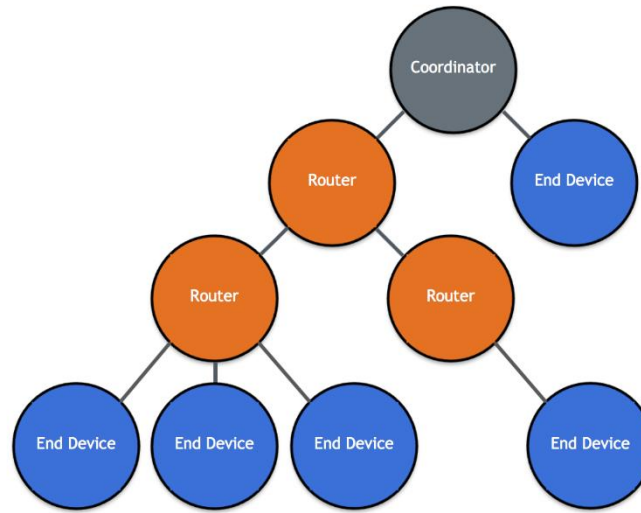
- Usages are similar with those in **Serial** class

Note: For Mega, we can use Serial1, Serial2, Serial3.
Also, corresponding class is available.

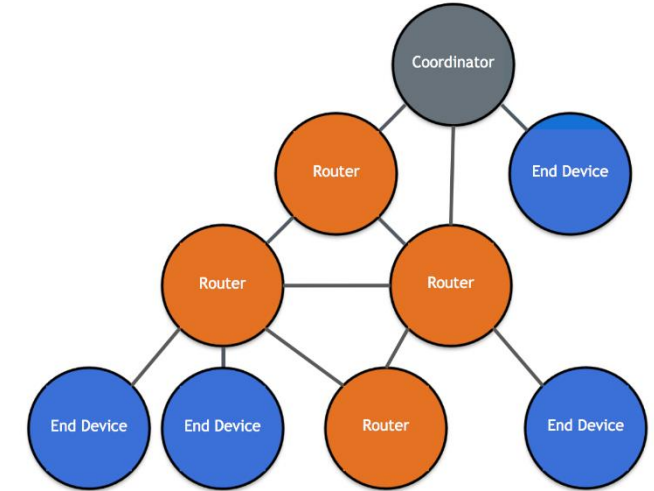
- ZigBee is an **IEEE 802.15.4**-based specifications for a suite of high-level communication protocols used to create personal area networks , such as for home automation
- Low cost, low power, low data rate
- Can form different network topologies
- According to the network, each node may have different role
 - **Coordinator**: first node to start up and initializes the rest of the network
 - **Router**: relay message to other nodes
 - **End device**: only send and receive messages, no other special functions



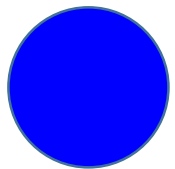
Star



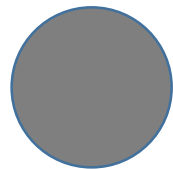
Tree



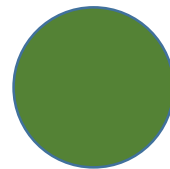
Mesh



End device



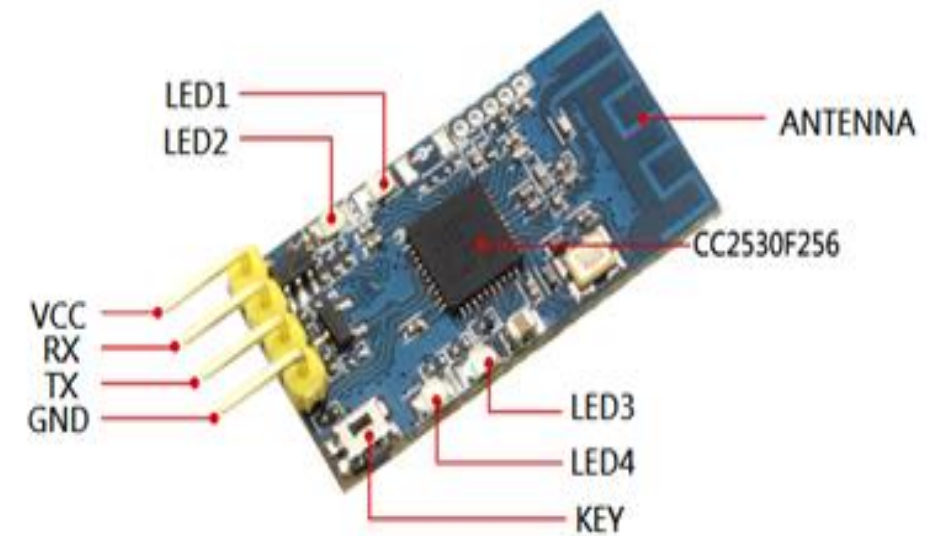
Coordinator



Router

- For the convenience of users, some manufactures produce ZigBee modules, which can be easily used for communication.
- The role of nodes are simplified but the topologies are limited.
- Eg. ZigBee modules CC2530F256
 - **Peer-to-Peer**: allow nodes to communicate directly with each other.
 - **Broadcast**: When a message is sent, any node in range will receive.

- There are four LEDs, which indicate the settings during the process of configuration, on the ZigBee module.
- To enter configuration mode,
 - Press and hold the KEY, then give power.
 - All the four LEDs begin to flash circularly
 - Release the KEY to enter the 1st configuration mode which is for baud rate

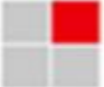
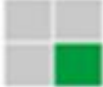


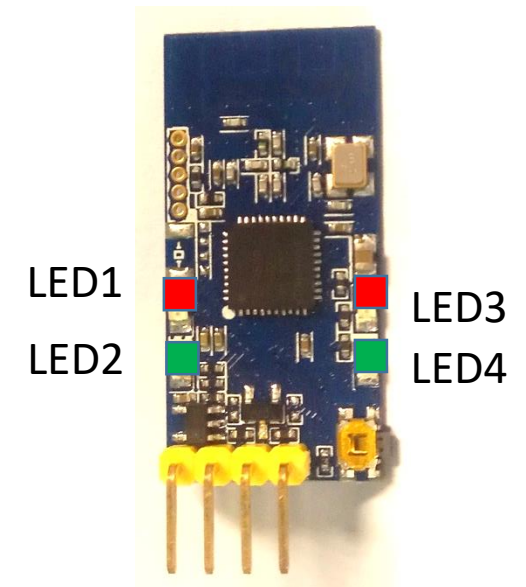
ZigBee Module

Setting 1 – Baud Rate

- Press the KEY shortly to set the baud rate (LED will change accordingly, from one setting to another)
- The relationship between the status of four LEDs and baud rate is shown below.

Note: LED1 and LED3 is red; LED2 and LED4 is green.

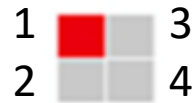
	2400		4800		9600		14400
	19200		38400		57600		115200



Setting 2 - Channel

- Next, enter the channel configuration mode
 - Press and hold the key again until the four LEDs begin to flash circularly
 - Release the KEY to enter the 2nd configuration mode which is for channel selection
- Press the KEY shortly to set the channel
 - There are $2^4 = 16$ statuses for the four LEDs, corresponding to 16 channels (0 – 15).

Eg.



LED 1 on: (0001) = channel 1



LED 2 and LED 3 on: (0110) = 4+2 = channel 6

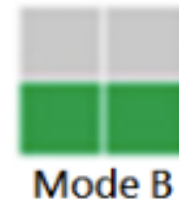
More About the Channels

- The standard (IEEE 802.15.4) specifies operation in the unlicensed 2.4 to 2.4835 GHz (worldwide), 902 to 928 MHz (Americas and Australia) and 868 to 868.6 MHz (Europe) [ISM bands](#). **Sixteen** channels are allocated in the **2.4 GHz** band, with each channel spaced 5 MHz apart, though using only 2 MHz of bandwidth.

No.	Freq(MHz)	No.	Freq(MHz)	No.	Freq(MHz)	No.	Freq(MHz)
1	2405	5	2425	9	2445	13	2465
2	2410	6	2430	10	2450	14	2470
3	2415	7	2435	11	2455	15	2475
4	2420	8	2440	12	2460	16	2480

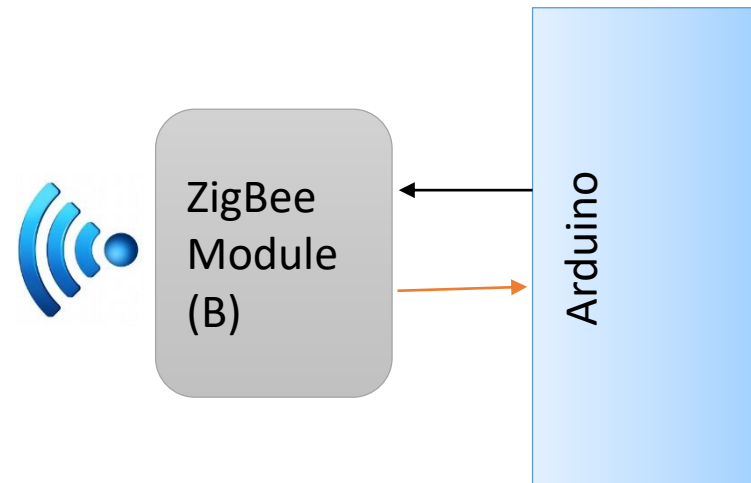
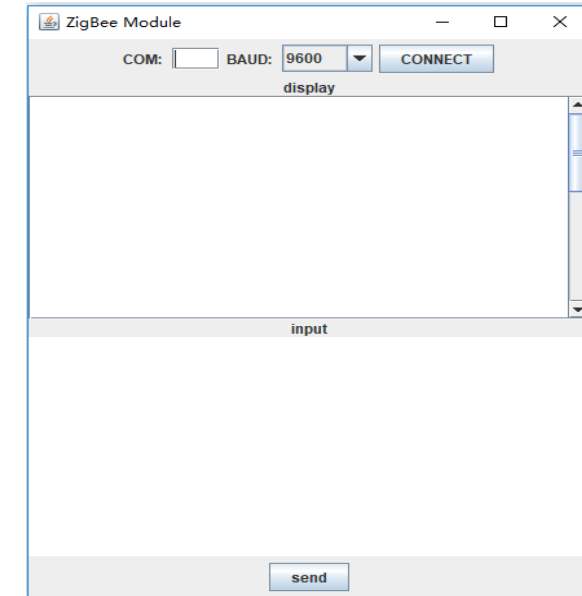
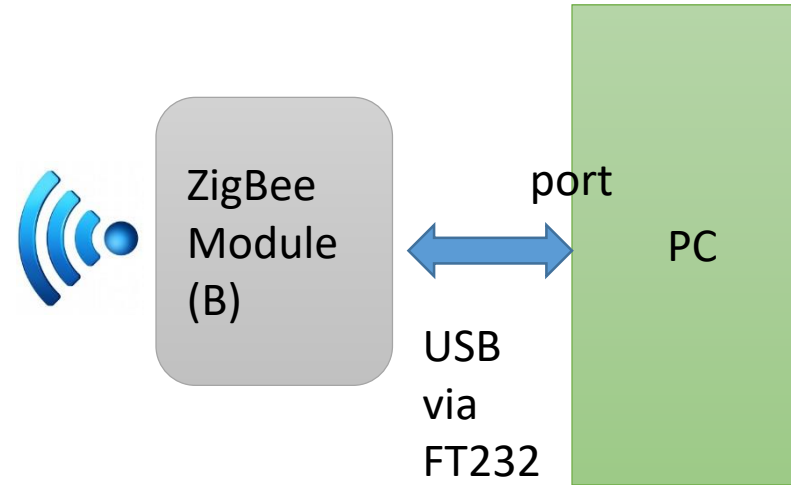
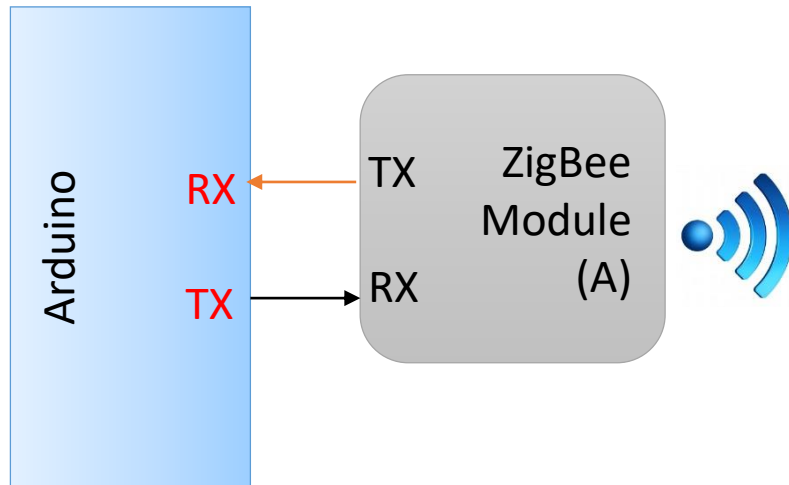
Setting 3 – Work Mode

- Lastly, enter work mode configuration
 - Press and hold the key again until the four LEDs begin to flash circularly
 - Release the KEY to enter the 3rd configuration mode which is for work mode selection
- Press the KEY shortly to set the working mode
- There are three modes:



- Peer-to-Peer communication
One module is in mode A; the other is in mode B
- Broadcast Communication
All are in mode C (can have more than two).

Examples of Configuration



- Bluetooth is a wireless technology standard for exchanging data over short distances from fixed and mobile devices, and building personal area networks (PANs).
- Based on **IEEE 802.15.1**
- Packet-based protocol with a **master/slave** architecture
- Form of Connections
 - Peer-to-Peer: A master Bluetooth device can communicate with a slave Bluetooth device
 - Piconet (an ad-hoc computer network): A master Bluetooth device can communicate with a maximum of seven slave devices

Typical Applications of Bluetooth



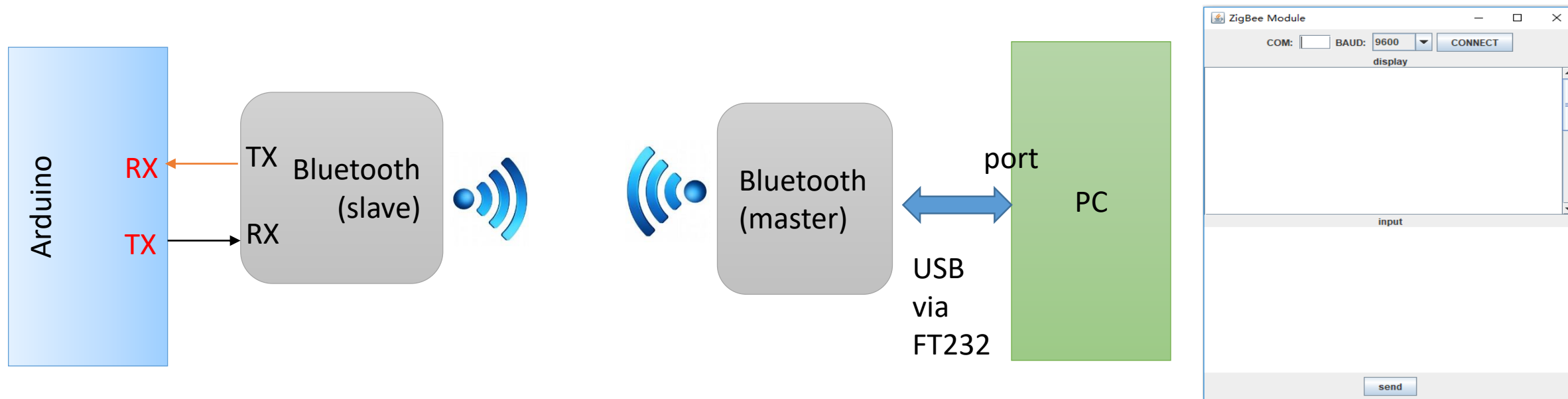
Mobile and HC05 Bluetooth Module

- Bluetooth Module: BT05



- <https://makersportal.com/blog/2019/10/14/bluetooth-module-with-arduino>

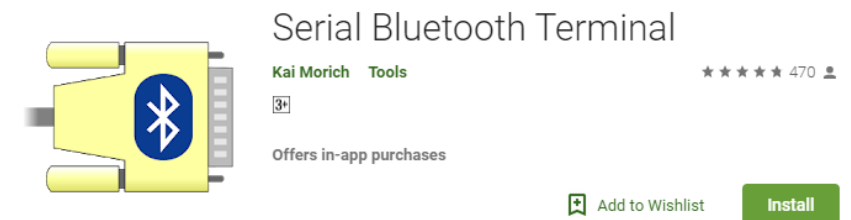
Examples of Configuration



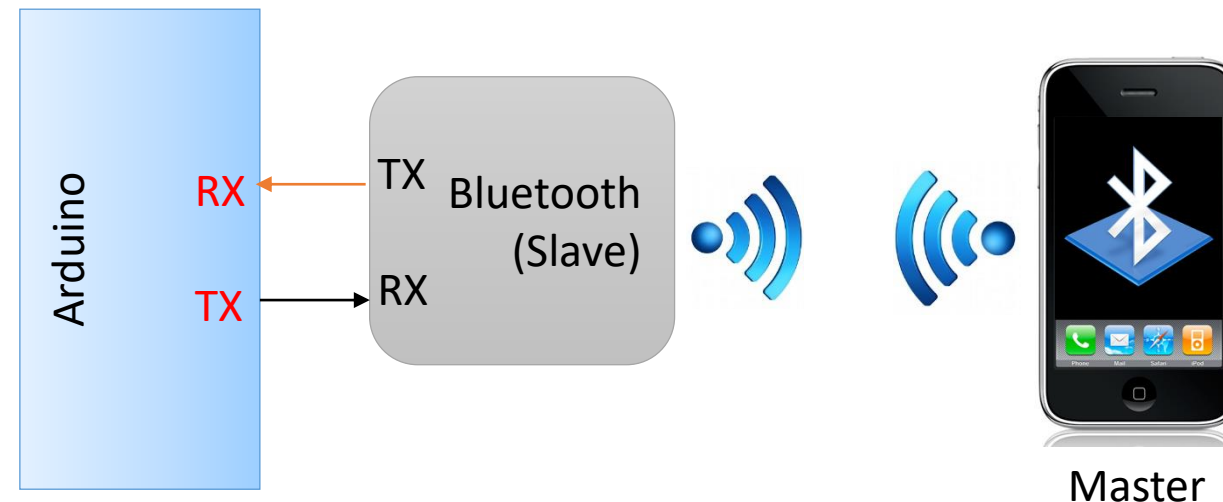
Note: You'll need to use AT command to set the mode; baud rate is 9600bps (default); need to know the BT local address.



Note: You'll need APP at Mobile side to scan and communication with the BT.
Example for android phone: Serial Bluetooth Terminal



- If you use a mobile to connect the Bluetooth (BT) module
 - **The role of BT module must be slave**, otherwise it cannot be scanned by your phone
 - When the mobile phone scanned a device, the device returns its name. Therefore, you should know **the name of your BT module** (Note: There may be many other BT modules nearby). If necessary, you can also change the name of a BT module.



Cloud Storage:

ThingSpeak

Cloud Service: ThingSpeak

- An IoT platform in cloud: <http://thingspeak.com>
- User account is associated with MathWorks account, and CityU student can have it
- Read the following site to create a MathWorks account

<https://www.cityu.edu.hk/csc/deptweb/facilities/central-sw-tah.htm>

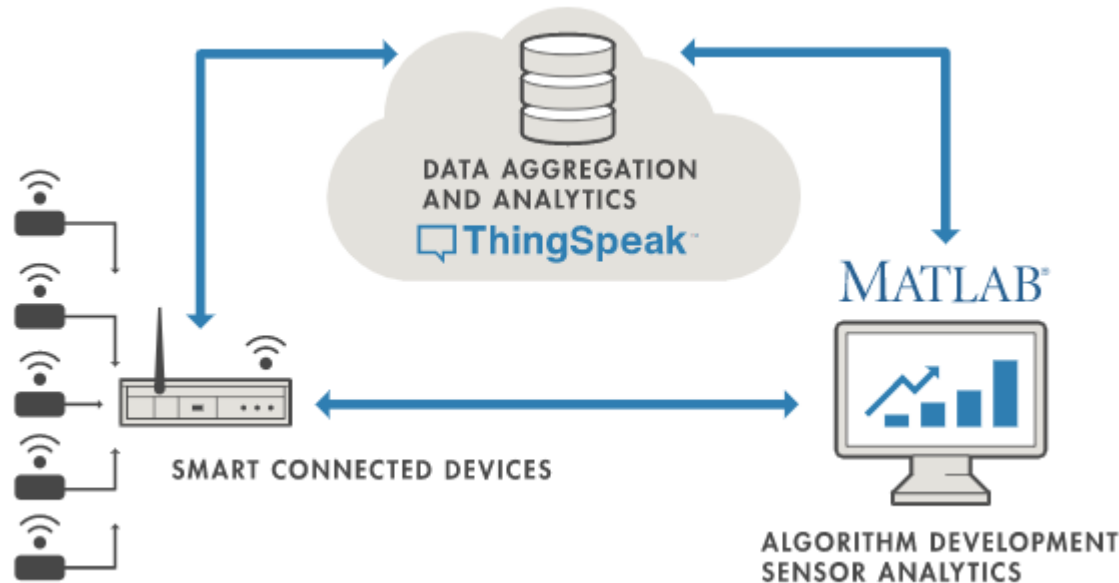
Individual License for Personally-owned Computers

The TAH license allows faculty, staff, and students to install MATLAB software on their personally-owned computers and laptops. Below are the steps for getting the installer and activating the individual license:

1. Go to [CityU's MATLAB Portal](#).
2. Click "**Sign in to get started**" in the Get MATLAB and Simulink section.
3. Sign in with your CityU EID and password.
4. Create a MathWorks account using your CityU email address.
5. Download the installer from MathWorks website.
6. Run the installer.
7. In the installer, select **Log in with a MathWorks Account** and follow the online instructions.
8. When prompted to do so, select the **Academic – Total Headcount** license labeled **Individual**.
9. Select the products to download and install.
10. After downloading and installing your products, keep the **Activate MATLAB** checkbox selected and click **Next**.
11. When asked to provide a user name, verify that the displayed user name is correct. Continue with the process until activation is complete.

Channels

- Channels: Public or Private
 - Public channel can be used to share data with others
- Allow to upload / download at most 8 fields via a channel.
- Once data is in a ThingSpeak channel, you can analyze and visualize it, calculate new data, or interact with social media, web services, and other devices



Channel Settings

Percentage complete 50%

Channel ID 1230455

Name

Description

Field 1	T1	☒
Field 2	T2	☒
Field 3	T3	☒
Field 4		☐
Field 5		☐
Field 6		☐
Field 7		☐
Field 8		☐

Upload / Download

- Set up a Channel and create fields (Channel ID is assigned)
- Upload (write) and Download (read) the fields
- Write / Read API Keys are needed for Private Channels

Testng Channel

Channel ID: 1230455
Author: eekstang
Access: Private

arduino, lab

[Private View](#) [Public View](#) [Channel Settings](#) [Sharing](#) [API Keys](#) [Data Import / Export](#)

Write API Key

Key

[Generate New Write API Key](#)

Read API Keys

Key

Note

[Save Note](#) [Delete API Key](#)

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click [Generate New Write API Key](#).
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click [Generate New Read API Key](#) to generate an additional read key for the channel.
- **Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

Channel Settings

Percentage complete 50%

Channel ID 1230455

Name

Description

Field 1	T1	☒
Field 2	T2	☒
Field 3	T3	☒
Field 4		☐
Field 5		☐
Field 6		☐
Field 7		☐
Field 8		☐

Tools to Access the Channels

- PC with MATLAB -> Internet
- Arduino with WIFI module -> Internet

End