香港城市大學
City University of Hong Kong
專業 創新 胸懷全球
Professional · Creative
For The World

EE 3070 Design Project

Technical Training 2
Laboratory Manual
(Design Modules)

2021/22 Semester A

Equipment List:

- Wifi Router with Internet connection in each laboratory
- PC with Arduino IDE (IDE Download link: http://arduino.cc/en/Main/Software
- Arduino Board (Mega 2560 or Micro)
- Wifi Module ESP8266
- Sensor Modules:
  - Light sensing modules with PGM5539
  - Temperature sensor LM35
  - Pressure sensing modules with RFP-602
  - Ultrasonic ranging module HC-SR04
- Display Module:
  - SPI bus 1.54 inch epaper GDEW0154T8 (152 x 152) with Waveshare driving board
  - I2C bus Passive OLED (0.96 inch, 128x64 dots)
- RFID System Module
  - PCD (Proximity Coupling Device) with MFRC522 Contactless Reader IC
  - PICC card (Proximity Integrated Circuit Card) with 1K memory
  - PICC tag with 1K memory

Topic Assignments:

Members of project group should discuss and choose the topics that relate to their design. The following must be noticed:

- Each member of a project group is to work on one of the 5 topics. Members within the same group should work on different topics.
- Each topic contains Basic Tasks and Advanced Tasks. You have to finish the Basic Tasks first. If time left, you proceed to Advanced Tasks.

Topic 1: Reading Sensors by Arduino

Components list:

- Arduino: Mega 2560 / Micro
- Light / pressure sensing module
- Ultrasonic distance module HC-SR04

A. Light Sensing Module

Fig. 1.1(a) and Fig. 1.1(b) show a light sensing module and its simplified circuit, respectively. The circuit is to convert the resistance attribute of light sensor (PGM5399) to voltage at o/p pin, which can then be read by an Arduino board.

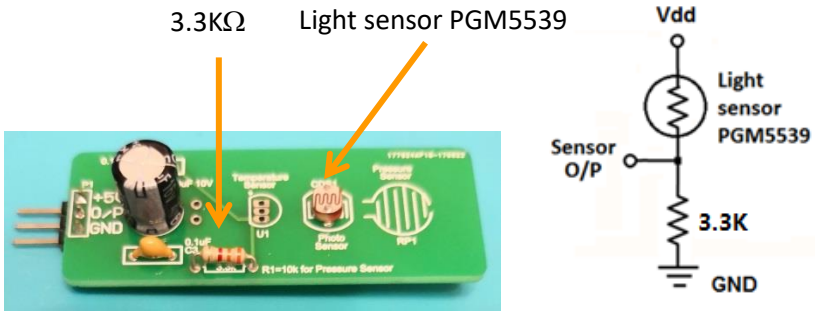


Fig. 1.1: (a) Light sensor module board and (b) the simplified circuit design

Self-Reading

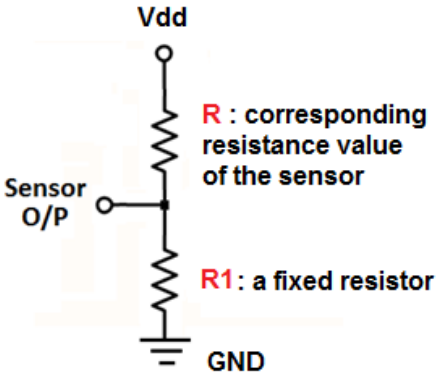
The light sensor can be considered as a resistor for which its value changes according to the light shining onto the sensing surface.

The circuit in right figure is referred as a voltage divider. Current flows from V_{dd} to GND and can be calculated by

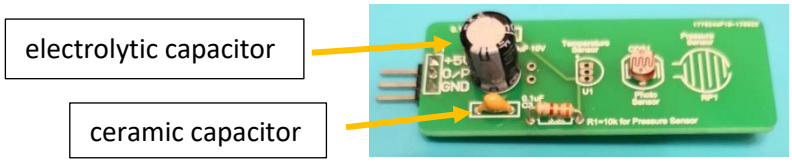
$$I = \frac{V_{dd}}{(R + R_1)}$$

Therefore, using Ohm's law, the voltage at the Sensor O/P pin (V_o) is:

$$V_o = IR_1 = V_{dd}\frac{R_1}{(R + R_1)}$$



You would find some additional capacitors on the module board. The electrolytic capacitor (e-cap) acts as charge reservoir for voltage source. The ceramic capacitor (with small value) is to resolve issue related to internal impedance (C_s in right figure) of ADC.



A.1 Hardware Connection

Fig. 1.2 shows the wiring which is to read the output from the sensing module by Arduino

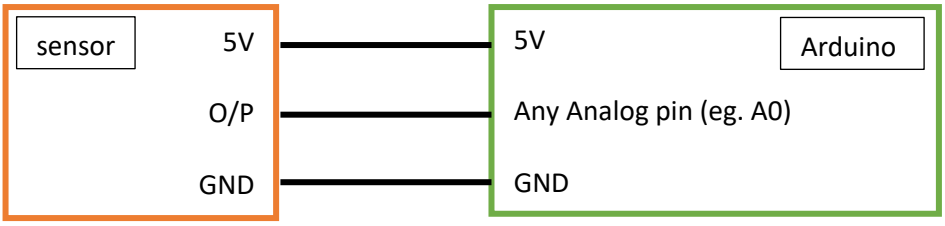


Figure 1.2 Connection between sensing module and Arduino board

A.2 Software Design: Library and functions

We can use analogRead as given in the standard library.

analogRead	Read the value from the specified analog pin.	
	analogRead(pin)	**pin:** the number of the analog input pin to read from (eg. A0 to A15 for Mega-2560) Return the value after ADC (as an integer)

The readings can be shown on the screen of a PC/notebook by launching Serial Monitor as given in Arduino IDE. Read Technical Training I Manual to review how to use Serial Monitor.

A.3 Self-Reading Materials

You can find the usage of functions and examples:
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

B. Pressure Sensing Module

Fig. 1.3 shows a pressure sensing module. The circuitry design is exactly the same as the light sensing module, except the light sensor (PGM5399) is now replaced by a pressure sensor RFP-602.

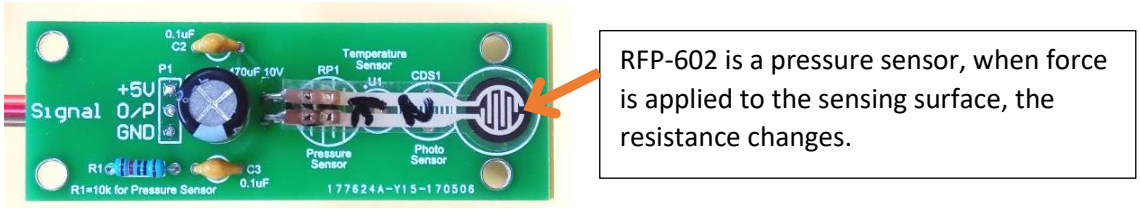


Fig. 1.3. Pressure sensing module

You can apply similar programming technique to get the readings from the pressure sensing module.

C. Ultrasonic Distance Module

HC-SR04 (see Fig. 1.4) is an ultrasonic ranging module, which measures distances based on echolocation.

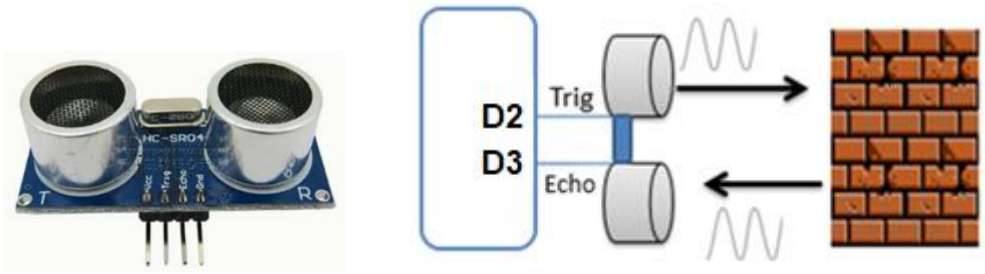


Fig. 1.4. (a) Photo of HC-SR04 (b) Measuring distance by HC-SR04 based on echolocation

The usage of this module is described in Fig. 1.5. When a HIGH pulse with width ≥10 μs is sent to Trig pin, the module will work as follows: It first emits out eight 40kHz square wave pulses, and then starts checking whether an echoed signal is received. If yes, the Echo pin will output a square pulse, for which its width equals to the time taken for the emitted wave sent forth and returned back.

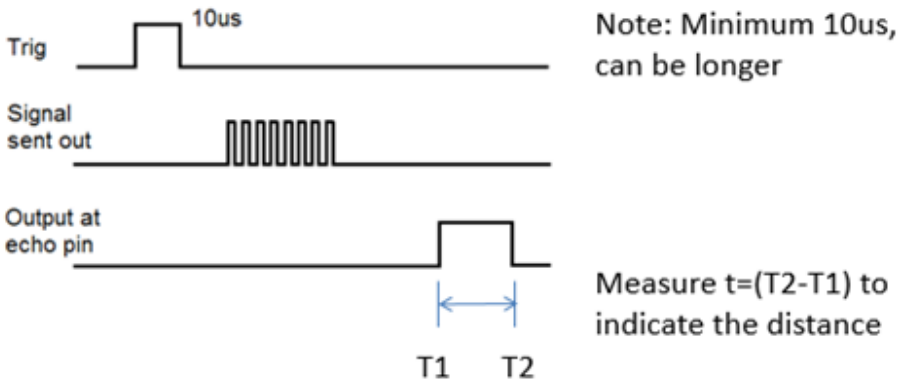


Fig. 1.5. Operational signals in HC-SR04

The distance between the sensor and the object is calculated by

$$Distance = t \times 340ms^{-1} \times 0.5$$

where $t = (T2 - T1)$ is the width of the pulse at echo pin; $340ms^{-1}$ is the speed of sound in air; multiplying 0.5 is needed because the wave travelled forth and back.

C.1 Hardware Connection

Fig. 1.6 shows the wiring for reading the output from the module.

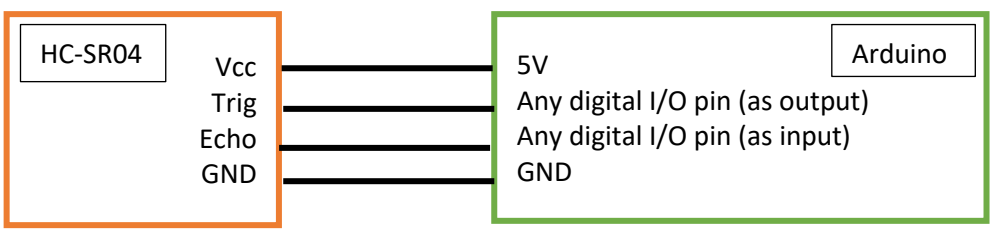


Figure 1.6 Connection between HC-SR04 and Arduino board

C.2 Software Design: Library and functions

We only need to use the standard library in Arduino. Some key functions are given.

pinMode	Configure the specified pin to behave either as an input or an output.	
	pinMode(pin, mode)	**pin:** the number of the pin whose mode you wish to set **mode**: INPUT, OUTPUT, or INPUT_PULLUP.
digitalWrite	Write a HIGH or a LOW value to a digital pin.	
	digitalWrite(pin, value)	**pin:** the pin number **value:** HIGH or LOW
delay	Pause the program for the amount of time (in milliseconds) specified as parameter.	
	delay(ms)	**ms:** the number of milliseconds to pause (unsigned long)
pulseIn	Read a pulse on a pin, return the length of the pulse in microseconds. If timeout, returns 0.	
	pulseIn(pin, value, timeout)	**pin:** the pin number for reading the pulse. **value:** HIGH or LOW. **timeout** (optional): maximum time (in ms) to wait for the pulse to start; default=1000ms.

C.3 Self-Reading Materials

To understand more, read the following material.

https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

Basic Design:

a) Design Task 1

Procedures:

1) Connect the light sensing module to an Arduino board (5V is applied to the module.)

2) Write a program to read the sensor value and output to the Serial Monitor

3) Observe the changing of value in Serial Monitor for different light levels

In normal condition, the output value is _____.

When partially covering the sensor, the output value is _____.

When totally covering the sensor, the output value is _____.

CHECKPOINT: SHOW THE RESULT AND CIRCUIT TO TUTOR

b) Design Task 2

Procedures:

1) Connect the ultrasonic distance module to Arduino board
2) Write a program to read the distance between the sensor and the table surface for every second, and output the values to the Serial Monitor,
3) Now, ADD a feature such that the color of a RGB LED changes according to the distance. Define your mapping between the color and the distance.

Describe the mapping between color and the distance

CHECKPOINT: SHOW THE RESULT AND EXPLAIN THE PROGRAM TO TUTOR

Questions to be answered in report:

Q1. Refer to Design Task 1, what do you expect for the changes of output value if you connect "5V" pin to 3.3V power bus, instead of 5V power bus in the platform board?

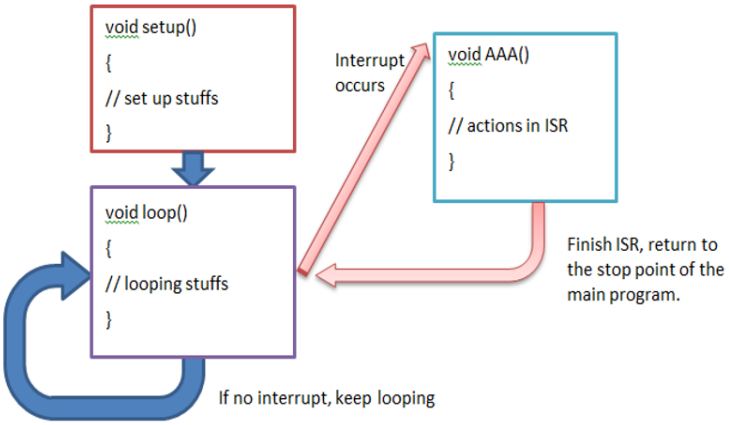
Q2. Refer to Design Task 1, what do you expect for the changes of output value if you connect "5V" pin and "GND" pin of the sensor module to GND and 5V in the platform board, respectively?

Q3. Observe the value change for different distance (from very close to very far away) in Design Task 2, and comment on the accuracy

Advanced Design: You work on this only after finished tasks in Basic Design

In Design Task 2, the disadvantage of continuously checking the return of the pulse is that we have to wait until there is a pulse, and the Arduino can do nothing while waiting.

To solve this, we can use Interrupt. The Arduino can be triggered by an external event using a digital pin. When event occurs, interrupt will be issued, and the Interrupt Service Routine will be run. (Note: You can refer to the Lecture Notes for Arduino Programming to understand more about Interrupt)

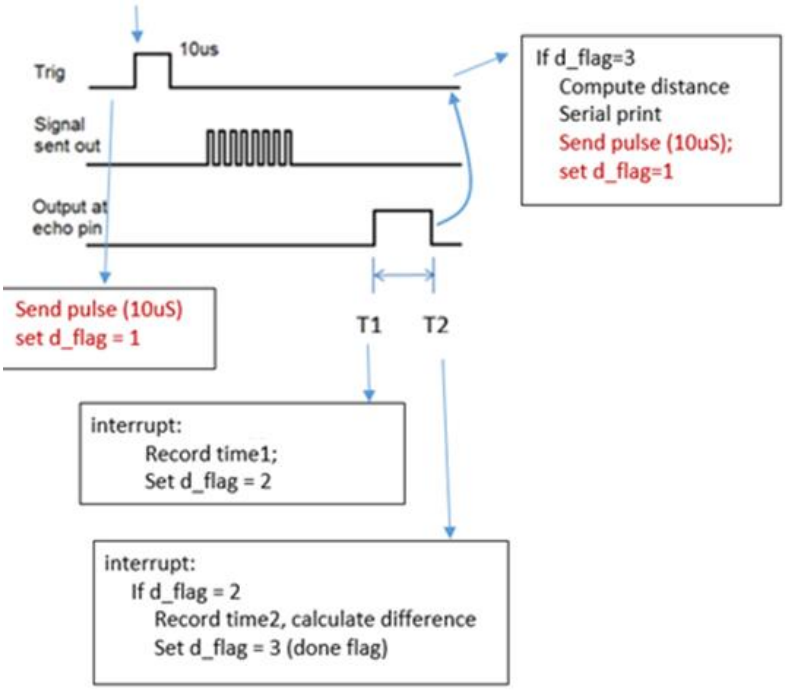


There are several modes to define when the interrupt should be triggered.

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

a) Advanced Task 1:

You are given the following design example. Use interrupt to redo Design Task 2.



CHECKPOINT: SHOW THE RESULT AND EXPLAIN THE PROGRAM TO TUTOR

Topic 2: Display texts and image on OLED by Arduino

Components list:

- Arduino: Mega 2560 / Micro
- 0.96 inch passive OLED, part number SSD1306

A. Introduction of OLED and I2C:

OLED is based on organic materials that emit light when electric current is applied. Fig. 2.1 shows the OLED (Part number: SSD1306) used in the project. It has a resolution of 128 x 64 (dots), and its communication bus is Inter-Integrated Circuit (I2C).

An I2C bus consists of two signal wires, SCL (Serial clock) and SDA (Serial data). In I2C, the master unit controls the data communication, so that data can be transferred between the master and the slave. To drive an OLED, we can use an Arduino board as the master unit while the OLED is a slave. To specify a particular slave, a distinct address is needed. For the OLED, it is hard-wired in the module, for which by default, the address is 0x3C. Arduino can then send data to OLED via the bus, and then the OLED displays accordingly.

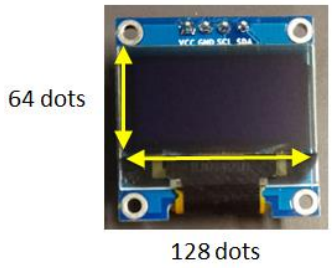
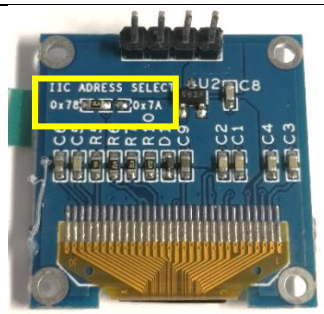


Fig. 2.1. Passive OLED: 0.96” in dimension and 128×64 in resolution

Note 1:
The I2C address is 7 bits. As shown at the back of the module (see the right figure), you can see a mark “0x78” or “0x7A”.

- The address 3C $_h$ which is 011 1100 $_b$, then appending the R/W bit, one has 78 $_h$ (0111 1000 $_b$).
- The address 3D $_h$ which is 011 1101$_b$, then appending the R/W bit, one has 7A $_h$ (0111 1010 $_b$).



B. Self-Reading Materials

To understand more, read the following materials

- English version: https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/

C. Hardware Connection

Arduino boards have dedicated SDA/SCL pins and a short summary is given in Table 2.1. (Self-reading: https://www.arduino.cc/en/reference/wire.h)

Table 2.1 Pin assignment for SDA and SCL in Arduino boards

Board Type	Mega 2560	Micro
SDA	20 (marked as SDA)	D2
SCL	21 (marked as SCL)	D3

Fig. 2.2 shows the connections between Arduino and OLED

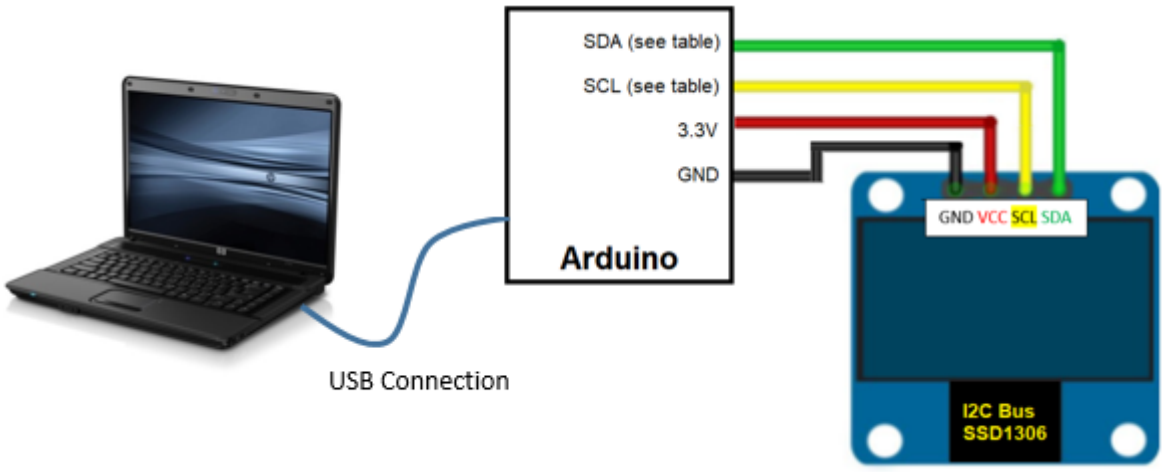


Fig. 2.2. Connecting OLED (SSD1306) to Arduino board. The USB connection to PC is for programming and providing power to Arduino.

D. Software Design and Library

D.1 Library

You'll need the following library to program the OLED SSD1306

1. Adafruit_SSD1306.h : It is a library to interface with the OLED SSD1306
   Self-Reading: https://github.com/adafruit/Adafruit_SSD1306

The library <Adafruit_SSD1306.h> needs another library. You'll need to install them if they were not included in your Arduino IDE:

2. Adafruit_GFX.h : It is a library for drawing and display
   Self-Reading https://github.com/adafruit/Adafruit-GFX-Library

Adafruit_SSD1306.h also uses another two libraries (we called this as dependency), which are bundled with the hardware packages. They are Wire.h and SPI.h. For students who like to know more, they may visit

- Wire.h: https://www.arduino.cc/en/reference/wire
- SPI.h: https://www.arduino.cc/en/Reference/SPI

Note: See Laboratory Manual (Arduino IDE and Programming) to learn how to add a library

Below shows the calling of the library and initialize the display (as an object in C++).

```
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

D.2 Program Example

You can open an example file to observe different functions (ssd1306_128x64_i2c.ino). Refer to Technical Training Manual I to review how to open an example file.

Below are some important notes:

1. The address of the I2C should be 0x3C. [This is configured in the OLED board.]

```
void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
```

2. After calling any drawing functions, you need to call display.display to update the screen so as to make them show on-screen.
3. Function delay() is added after display.display because it takes times for the screen to update.

```
// Clear the buffer
display.clearDisplay();

// Draw a single pixel in white
display.drawPixel(10, 10, SSD1306_WHITE);

// Show the display buffer on the screen. You MUST call display() after
// drawing commands to make them visible on screen!
display.display();
delay(2000);
// display.display() is NOT necessary after every single drawing command,
// unless that's what you want...rather, you can batch up a bunch of
// drawing operations and then update the screen all at once by calling
// display.display(). These examples demonstrate both approaches...
```

D.3 Brief Descriptions on Drawing Functions

A brief summary of drawing functions is given below. Details can be referred to the following link:

https://learn.adafruit.com/adafruit-gfx-graphics-library/graphics-primitives

Since the OLED is monochrome, color = SSD1306_WHITE to turn on the dot.

1. drawPixel(x, y, color): (x,y) is pixel location
2. drawLine(x1, y1, x2, y2, color): (x1,y1) and (x2,y2) are the starting and end points
3. drawRect(x, y, w, h, color): (x,y) is top-left corner of the rectangle, w and h are its width and height, respectively.
4. fillRect(x, y, w, h, color): same as drawRect but the rectangle is filled
5. drawRoundRect(x, y, w, h, r, color): (x,y) is top-left corner of the rectangle, w and h are the width and height, r is the corner radius.
6. fillRoundRect(x, y, w, h, r, color): same as drawRoundRect, except the rectangle is filled
7. drawCircle(x, y, r, color): (x,y) is the center of the circle and r is its radius.
8. fillCircle(x, y, r, color): same as drawCircle, except the circle is filled.
9. drawTriangle(x1, y1, x2, y2, x3, y3, color): (x1,y1), (x2,y2), (x3,y3) are the corner locations of the triangle
10. fillTriangle(x1, y1, x2, y2, x3, y3, color): same as drawTriangle, except the triangle is filled
11. setTextSize(size): Set the text size, 1 is normal size
12. setTextColor(color): Set the text color
13. setCursor(x, y): move the cursor to (x,y)
14. println("…") or println(F("…")): write the text in "" starting at current cursor location; with the symbol 'F', the string is stored in flash memory, not in SRAM
15. drawBitmap(x, y, *bmpfile, w, h, color): (x,y) is the top-left corner, w and h are the width and height, bitmap data must be located in program memory using PROGMEM directive. The color can be inversed using SSD1360_INVERSE as color.
16. clearDisplay(): clear the current screen

Basic Design:

a) Design Task 1:

Understand the program example and successfully run the program according to your hardware configuration and settings. Also try at least 3 functions as given in Sect. A.4.3.

b) Design Task 2:

Draw a square with size 40 x 40 at the center of the OLED and write your name below

c) Design Task 3:

Create a logo using bitmap which represents yourself and display it onto the OLED

Do the tasks with separated program or screens. Show the results to Tutor and explain how you do the design.

CHECKPOINT: SHOW THE RESULT AND EXPLAIN TO TUTOR FOR EACH TASK

Questions to be answered in report:

Q1. Explain the usage of 3 functions (that you tried in Design Task 1) and their outputs.

Q2. How to determine the x- and y-coordinates of the center of an OLED, if the screen side is given as X and Y?

Q3. How many memory has been used for different tasks? Which one occupies most and Why?

Advanced Design: You work on this only after finished tasks in Basic Design

a) Advanced Task 1:

Design a program to create an animation: Make a solid square with side 20x20 dots moving from left to right, then from right to left, repetitively. The square changes direction, once its edge reaches the boundary of OLED display.

CHECKPOINT: SHOW THE RESULT AND EXPLAIN THE PROGRAM TO TUTOR

Topic 3: Display texts and image on E-paper using Arduino

Components list:

- Arduino: Mega 2560 / Micro
- 1.54 inch e-paper, part number GDEH0154D67 (Waveshare driver board)

A. Introduction of Epaper and SPI:

An e-paper contains many tiny capsules with black or white particles, which are negatively or positively charged, respectively. When positive charge is applied at the top, the black particles are drawn to the top and become visible. Similarly, if negative charge is applied to top, the screen turns to white. The particles would stay at their positions even removing the charge, meaning that the image can be kept even disconnecting the voltage supply.

An e-paper screen displays the patterns by reflecting the ambient light, so no background light is required. It also has high visibility with a wide viewing angle of 180 degrees.

Fig. 3.1 shows a photo of the e-paper and the driver board (manufacturer Waveshare). To communicate with the e-paper in this project, Serial Peripheral Interface (SPI) is used. SPI is an interface bus, using clock (SCK) and data lines (MOSI – Master-Output-Slave-Input, MISO - Master-Input-Slave-Output) for data communications. Additionally, a slave select line (SS) is used to choose the device that to be communicated with. Some devices use a half-duplex interface, having only a single data line. This kind of SPI is commonly referred as 3-wire SPI.

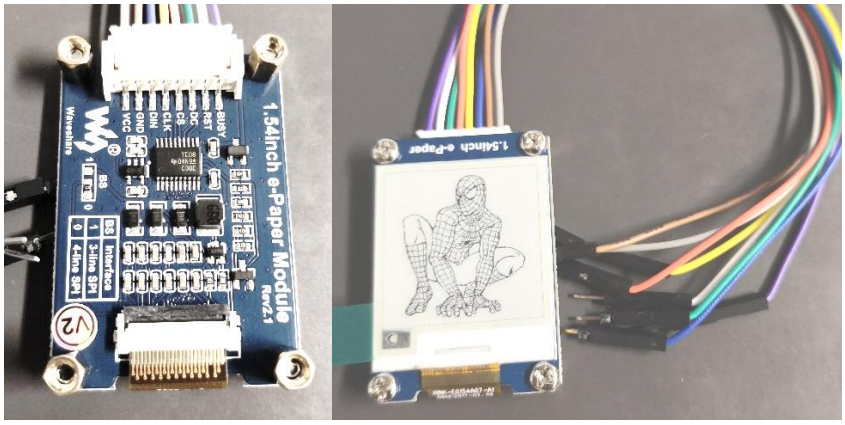


Fig. 3.1 (a) Back view: driver (b) Front view: Epaper panel

B. Self-Reading Materials:

To understand more, read the following materials
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

C. Hardware Connections

Some Arduino boards have dedicated SCK/MOSI/MISO/SS pins, and a short summary is given in the following table (Self-Reading: https://www.arduino.cc/en/reference/SPI)

Table 3.1 Pin assignment for SCK/MOSI/MISO/SS in Arduino boards

Board Type	MOSI	MISO	SCK	SS
Mega 2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53
Micro	16 or MOSI	14 or MISO	15 or SCK	17 or SS

Note: ICSP refer to the socket on the Arduino boards. The pins are defined as follows. [Note: You'll see a dot at pin 1 of the socket.]

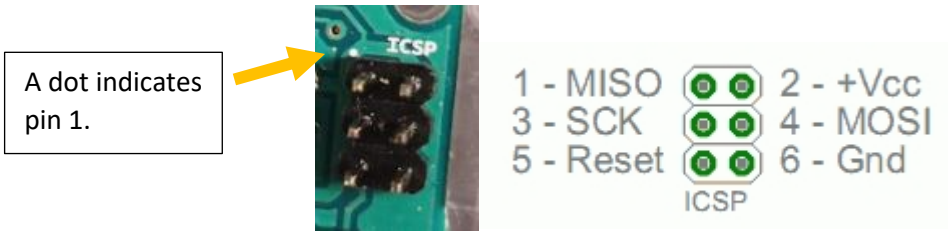


Fig. 3.2 ICSP socket on Arduino boards

Since we only write data from the master (Arduino) to the slave (epaper), only MOSI is needed. Also, to program an epaper, 3 more pins are needed. They are DC (Data/Command pin), RST (Reset pin) and BUSY (Busy detect pin). You can freely assign I/O pins to SS, DC, RST, and BUSY. An example of wiring is given in Table 3.2.

Table 3.2: Wiring connections between Epaper and Arduino board

GDEW0154T8	Arduino Mega	Arduino Micro
VCC	3.3V	3.3V
GND	GND	GND
DIN=MOSI	MOSI(51)	MOSI
CLK=SCK	SCK(52)	SCK
CS=SS	10	SS
DC	9	5
RST	8	6
BUSY	7	7

Note: Other pin numbers can be used for CS, DC, RST and BUSY, which is also defined in program.

D. Software Design

D.1 Library

You'll need to install the following library to program the epaper:

1. GxEPD2 : It is a library to interface with epaper manufactured by Waveshare.

To use this library, the following library needs to be pre-installed in the Arduino IDE.

2. Adafruit_GFX.h : It is a library for drawing and graphics.

Note: See Technical Training I Manual to review how to add a library

GxEPD2 also use another library SPI.h (we called this as dependency), which is bundled with the hardware packages. For students who like to know more, they may visit

- SPI.h: https://www.arduino.cc/en/Reference/SPI

Self-Reading Materials:

About GxEPD2: https://github.com/ZinggJM/GxEPD2

About Adafruit_GFX:

https://github.com/adafruit/Adafruit-GFX-Library

https://learn.adafruit.com/adafruit-gfx-graphics-library/graphics-primitives

About text fonts: https://learn.adafruit.com/adafruit-gfx-graphics-library/using-fonts

D.2 Use of Library and Initialization

Below shows the calling of the library and initialization of the display

```
// GxEPD2_MinimumExample.ino by Jean-Marc Zingg
// epaper 1.54" GDEH0154D67
// size 200 x 200

//#define ENABLE_GxEPD2_GFX 1

#include <GxEPD2_BW.h>
#include <Fonts/FreeMonoBold9pt7b.h>
#define ENABLE_GxEPD2_GFX 0

// copy constructor for your e-paper from GxEPD2_Example.ino, and for AVR needed #defines
 #define MAX_DISPLAY_BUFFER_SIZE 800 //
 #define MAX_HEIGHT(EPD) (EPD::HEIGHT <= MAX_DISPLAY_BUFFER_SIZE / (EPD::WIDTH / 8) ? EPD::HEIGHT : MAX_DISPLAY_BUFFER_SIZE / (EPD::WIDTH / 8))

GxEPD2_BW<GxEPD2_154_D67, MAX_HEIGHT(GxEPD2_154_D67)> display(GxEPD2_154_D67(/*CS=10*/ 10, /*DC=*/ 9, /*RST=*/ 8, /*BUSY=*/ 7)); // GDEH0154D67 200x200
```

Explanation

2 header files are included

- GxEPD2_BW.h : use for the monochrome epaper
- Fonts/FreeMonoBold9pt7b.h : A font type used in text

Remark: If you don't use Paged display, no need to add

Define ENABLE_GxEPD2_GFX as 0 because we use Adafruit_GFX instead of this.

Define MAX_DISPLAY_BUFFER_SIZE and MAX_HEIGHT(EPD) which are for the Paged display, and it will be described in more details soon.

The last line is to define the pins for CS, DC, RST and BUSY

Note: The pin assignment should match with your own wiring.

D.3. Brief descriptions on drawing functions

The followings are some drawing functions supported by Adafruit_GFX.h. You can observe their outcomes by insert them in the program example (as given in next page). Details can be referred to the following link:

https://learn.adafruit.com/adafruit-gfx-graphics-library/graphics-primitives

Also, since the epaper in-use is monochrome, color = GxEPD_WHITE or GxEPD_BLACK

1. fillScreen(color): fill the whole defined screen with color
2. setRotation(angle): angle is 0,1,2,3; rotate the display by 0, 90, 180, 270 degree in clockwise
3. drawPixel(x, y, color): (x,y) is pixel location
4. drawLine(x1, y1, x2, y2, color): (x1,y1) and (x2,y2) are the starting and end points
5. drawRect(x, y, w, h, color): (x,y) is top-left corner, w and h are the width and height
6. fillRect(x, y, w, h, color): same as drawRect but the rectangle is filled
7. drawRoundRect(x, y, w, h, r, color): (x,y) is top-left corner, w and h are the width and height, r is the corner radius
8. fillRoundRect(x, y, w, h, r, color): same as drawRoundRect, except the rectangle is filled
9. drawCircle(x, y, r, color): (x,y) is the center and r is the radius.
10. fillCircle(x, y, r, color): same as drawCircle, except the circle is filled.
11. drawTriangle(x1, y1, x2, y2, x3, y3, color): (x1,y1), (x2,y2), (x3,y3) are the corner locations
12. fillTriangle(x1, y1, x2, y2, x3, y3, color): same as drawTriangle, except the triangle is filled
13. drawBitmap(x, y, *bmpfile, w, h, color): (x,y) is the top-left corner, w and h are the width and height, bitmap data must be located in program memory using PROGMEM directive.
14. setCursor(x, y): move the cursor to (x,y)
15. setTextColor(color): Set the text color
16. setFont(*font): Set the text font, eg. display.setFont(&FreeMonoBold9pt7b);
17. setTextSize(size): Set the text size, 1 is normal size
18. print("…"): write the text in "…", starting at current cursor location

C.4 Paged Drawing

To reduce buffer size, we display something on an epaper using pages (i.e. page by page). The page size is specified by the buffer size, as defined in the program (see last page). A display is given in a picture loop, specified by firstPage() and nextPage() frames.

The standard procedures are as follows

1. init(): Initialize the epaper
2. firstPage: draw the first page (the buffer)
3. nextPage: use a while loop to keep on drawing next Page
4. powerOff(): Turn off generation of panel driving voltage. This is essential when finish your display.

A program example is given in next page.

```
// this part of code has been explained in Sec.D.2

#include <GxEPD2_BW.h>

#define ENABLE_GxEPD2_GFX 0

#define MAX_DISPLAY_BUFFER_SIZE 800
#define MAX_HEIGHT(EPD) (EPD::HEIGHT <= MAX_DISPLAY_BUFFER_SIZE / (EPD::WIDTH / 8) ? EPD::HEIGHT : MAX_DISPLAY_BUFFER_SIZE / (EPD::WIDTH / 8))
GxEPD2_BW<GxEPD2_154_D67, MAX_HEIGHT(GxEPD2_154_D67)> display(GxEPD2_154_D67(/*CS=10*/ 10, /*DC=*/ 9, /*RST=*/ 8, /*BUSY=*/ 7)); // GDEH0154D67 200x200
//

void setup()
{
  display.init();
  delay(1000);

  display.firstPage();
  do {
    // this part of code is executed multiple times, as many as needed, in case of full buffer it is executed once

// set the background to white (fill the buffer with value for white)
      display.fillScreen(GxEPD_WHITE);
      delay(500);

      display.fillCircle(20,60, 10, GxEPD_BLACK);  // draw a circle
      delay(500);
  } while (display.nextPage());

// tell the graphics class to transfer the buffer content (page) to the controller buffer the graphics class will command
the controller to refresh to the screen when the last page has been transferred
// returns true if more pages need be drawn and transferred; returns false if the last page has been transferred and the screen
refreshed for panels without fast partial update

  display.powerOff();
}
void loop()
{}
```

Basic Design:

a) Design Task 1:

Understand the program example and successfully run the program according to your hardware configuration and settings. Also try at least other 3 functions as in Sect. A.2.2.

b) Design Task 2:

Draw a rectangle with size 100 x 50 at the center of the epaper and then write your name inside; try to put your name at the center of the rectangle

c) Design Task 3:

Create a logo using bitmap which represents yourself and display it onto the epaper (attempt with and without paged drawing)

Do the tasks with separated screens. Show the results to Tutor and explain how you do the design.

CHECKPOINT: SHOW THE RESULT TO TUTOR FOR EACH TASK

Questions to be answered in report:

Q1. Explain the usage of 3 functions that you tried in Design Task 1 and their outputs.

Q2. What is the x- and y-location of the left-top corner (dot) of your rectangle in Design Task 2?

Q3. How many memory has been used for different tasks? Which one occupies most and Why?

Advanced Design: You work on this only after finished tasks in Basic Design

a) Advanced Task 1

It is possible to update a partial window of an epaper. You may explore it by checking the library. In this advanced design,

- Draw a square with size 100 x 50 dots at the center of the epaper, and then display your name and your members' names alternatively inside the rectangle, without refreshing the whole screen
- It can be further advanced: instead of displaying the members' name, the texts to be displayed are changed according to the input from Serial monitor

CHECKPOINT: SHOW THE RESULT AND EXPLAIN THE PROGRAM TO TUTOR

Topic 4: Read / write RFID card using Arduino

Components list:

- Arduino: Mega 2560 / Micro
- MFRC522 package
  - PCD (Proximity Coupling Device), based on NXP MFRC522 Contactless Reader IC
  - PICC card (Proximity Integrated Circuit Card): MIFARE Card with 1K memory
  - PICC tag: MIFARE Tag with 1K memory

A. Introduction of RFID

RFID system divided into passive and active RFID. In this project, we focus on passive RFID, and the part number is MFRC522.

A passive RFID system has an antenna and circuitry that houses a unique code, but has no power source. A RFID reader (PCD) is needed to read/write the passive RFID card (PICC card or tag). MFRC522 is useful for contactless communication (with a very short range. A reader/writer antenna is used to communicate with the cards), by modulating signal of frequency 13.56MHz. The receiver module then demodulates and decodes the signals.

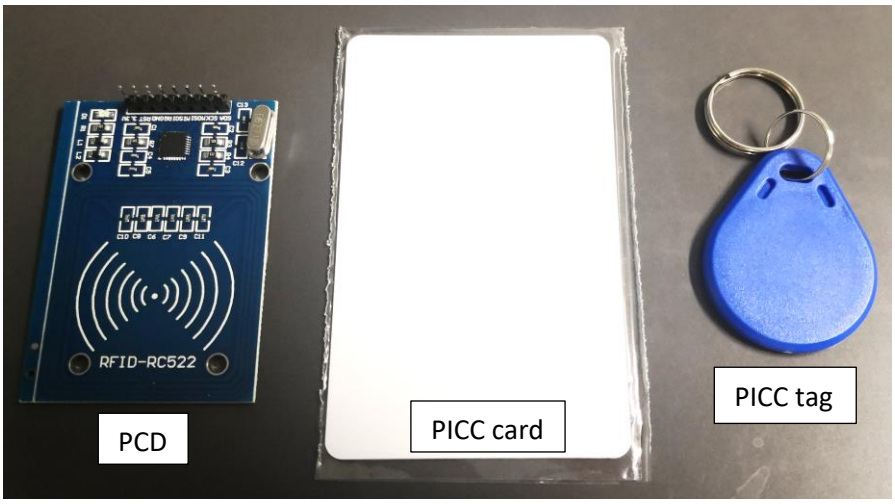


Fig. 4.1. Photo of PCD, PICC card and tag in MFRC522 package

Communications between Arduino boards and the PCD are based on Serial Peripheral Interface (SPI). SPI is an interface bus, which uses clock (SCK) and data lines (MOSI – Master-Output-Slave-Input, MISO - Master-Input-Slave-Output) for data communications. Additionally, a slave select line (SS) is used to choose the device that to be communicated with. Some devices use a half-duplex interface, having only a single data line. This kind of SPI is commonly referred as 3-wire SPI.

B. Self-Reading Materials

Device datasheet: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

Introduction on hardware connection and program:

https://medium.com/autonomous-robotics/an-introduction-to-rfid-dc6228767691

C. Hardware Connections

Some Arduino boards have dedicated SCK/MOSI/MISO/SS pins, and a short summary is given in the following table (Self-Reading: https://www.arduino.cc/en/reference/SPI)

Table 4.1 Pin assignment for SCK/MOSI/MISO/SS in Arduino boards

Board Type	MOSI	MISO	SCK	SS
Mega	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53
Micro	16 or MOSI	14 or MISO	15 or SCK	17 or SS

Note: ICSP refer to the socket on Arduino boards. The pins are defined as follows. [Note: You'll see a dot at pin 1 of the socket.]

Fig. 4.2. ICSP on Arduino boards

To communicate between PCD and Arduino board, another pin, RST (Reset pin), is needed. You can freely choose an I/O pin for RST. Examples are given in the following table.

Table 4.2 Example of wiring between PCD and Arduino boards

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

C. Software Design

C.1 Library

You'll need to install the following library to program the RFID reader and cards:

1. MFRC522 : It is a library to interface with RFID reader using MFRC522.

To use this library, the following library needs to be pre-installed in the Arduino IDE.

2. Adafruit_GFX.h : It is a library for drawing and graphics

MFRC522 also use the library SPI,h. For students who like to know more, they may visit

- SPI.h: https://www.arduino.cc/en/Reference/SPI

Self-Reading Materials

About RFID: https://github.com/miguelbalboa/rfid

About MFRC522 library: https://www.arduino.cc/reference/en/libraries/mfrc522/

C.2 Brief descriptions on functions, enum and structures

Functions for setting up the Arduino

```
MFRC522(SSPin, resetPin);    // set up the Arduino by defining the SS and RESET pins
```

Functions for manipulating the MFRC522

```
PCD_Init()
PCD_Reset()
```

Functions for communicating with PICCs

```
byte PICC_Select(Uid *uid, byte validBits = 0);
byte PICC_HaltA();
```

Functions for communicating with MIFARE PICCs

```
byte PCD_Authenticate(byte command, byte blockAddr, MIFARE_Key *key, Uid *uid);
void PCD_StopCrypto1();
byte MIFARE_Read(byte blockAddr, byte *buffer, byte *bufferSize);
byte MIFARE_Write(byte blockAddr, byte *buffer, byte bufferSize);
```

Convenience functions - does not add extra functionality

```
bool PICC_IsNewCardPresent();
bool PICC_ReadCardSerial();
```

Detailed documentation – enum and structures

PICC types we can detect.

```
enum PICC_Type
```

Return codes from the functions in this class.

```
enum StatusCode
```

A struct used for passing the UID of a PICC.

```
typedef struct {
byte size; Number of bytes in the UID. 4, 7 or 10.
byte uidByte[10];
byte sak; The SAK (Select acknowledge) byte returned
from the PICC after successful selection.
} Uid
```

A struct used for passing a MIFARE Crypto1 key

```
typedef struct {
byte keyByte[MF_KEY_SIZE];
} MIFARE_Key;
```

Member variables

```
Uid uid; Used by PICC_ReadCardSerial().
```

C.3 Example: ReadNUID in library

The following is an example program (ReadNUID)

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN);        // Create object Instance

MFRC522::MIFARE_Key key;              //create variable “key” with MIFARE_Key structure

// Init array that will store new NUID
byte nuidPICC[4];

void setup() {
   Serial.begin(9600);
   SPI.begin(); // Init SPI bus
   rfid.PCD_Init();      // Init the MFRC522 chip

   for (byte i = 0; i < 6; i++) {
      key.keyByte[i] = 0xFF;
   }

   Serial.println(F("This code scan the MIFARE Classsic NUID."));
   Serial.print(F("Using the following key:"));
   printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
}

void loop() {

   // Reset the loop if no new card present on the sensor/reader. This saves the entire process when
idle.
   if ( ! rfid.PICC_IsNewCardPresent())        // Check whether there is PICC card
      return;

   // Verify if the NUID has been read
   if ( ! rfid.PICC_ReadCardSerial())          //Read the card, if true, then continue
      return;

   Serial.print(F("PICC type: "));
   MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
   Serial.println(rfid.PICC_GetTypeName(piccType));

   // Check whether the card type is Classic MIFARE type or not
   if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
      piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
      piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
      Serial.println(F("Your tag is not of type MIFARE Classic."));
      return;
   }
   // check whether the UID of the card is the same as the stored one
```

```
    if (rfid.uid.uidByte[0] != nuidPICC[0] ||
        rfid.uid.uidByte[1] != nuidPICC[1] ||
        rfid.uid.uidByte[2] != nuidPICC[2] ||
        rfid.uid.uidByte[3] != nuidPICC[3] ) {
        Serial.println(F("A new card has been detected."));

        // Store NUID into nuidPICC array
        for (byte i = 0; i < 4; i++) {
            nuidPICC[i] = rfid.uid.uidByte[i];
        }

        Serial.println(F("The NUID tag is:"));
        Serial.print(F("In hex: "));
        printHex(rfid.uid.uidByte, rfid.uid.size);
        Serial.println();
        Serial.print(F("In dec: "));
        printDec(rfid.uid.uidByte, rfid.uid.size);
        Serial.println();
    }
    else Serial.println(F("Card read previously."));

    // Halt PICC
    rfid.PICC_HaltA();

    // Stop encryption on PCD
    rfid.PCD_StopCrypto1();
}


/**
  * Helper routine to dump a byte array as hex values to Serial.
  */
void printHex(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? " 0" : " ");
        Serial.print(buffer[i], HEX);
    }
}

/**
  * Helper routine to dump a byte array as dec values to Serial.
  */
void printDec(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? " 0" : " ");
        Serial.print(buffer[i], DEC);
    }
}
```

Basic Design:

a) Design Task 1:

Understand the sample program and successfully run the program with the project platform (Note: Modify it to match with your hardware and configuration).

b) Design Task 2:

Write a program to output "Yes" in Serial monitor and light up a RGB-LED with green color if the ID is matched with a stored value; otherwise, display "No" and light up a RGB-LED with red color

c) Design Task 3:

Write a program to count how many times a particular card has been read. You have at least two cards. Output the counts to Serial monitor, and also, light up RGB-LED with different colors for different cards.

Show the results to Tutor and explain how you do the design.

CHECKPOINT: SHOW THE RESULT AND EXPLAIN TO TUTOR FOR EACH TASK

Questions to be answered in report:

Read the following material for MIFARE classic 1K smart card.
https://shop.sonmicro.com/Downloads/MIFARECLASSIC-UM.pdf

Q1. Where is the UID of a RFID tag stored in the memory?

Q2. In total, how many bytes can be used for storing data? [Remark: Byte 9 in sector trailer can also be used for general purpose 1 byte user data.]

Q3. What happens if the sector trailer is written with improper format?

Advanced Design: You work on this only after finished tasks in Basic Design

a) Advanced Task 1:

Write a program for the following task

> Read a RFID tag and check its ID. If it matches with a pre-stored ID, MFincrement the byte 0 of 1st block of 1st sector by one, and display "Increment xxx" in Serial monitor.
> Otherwise, display "Wrong RFID xxx" in Serial monitor.
> "xxx" is the ID of the tag's ID.

(Note: There is an example in the library about read/write a data block. The flow chart is given in Appendix 4A.)

CHECKPOINT: SHOW THE RESULT AND EXPLAIN THE PROGRAM TO TUTOR

Appendix 4A: Flow chart to Read/Write Block

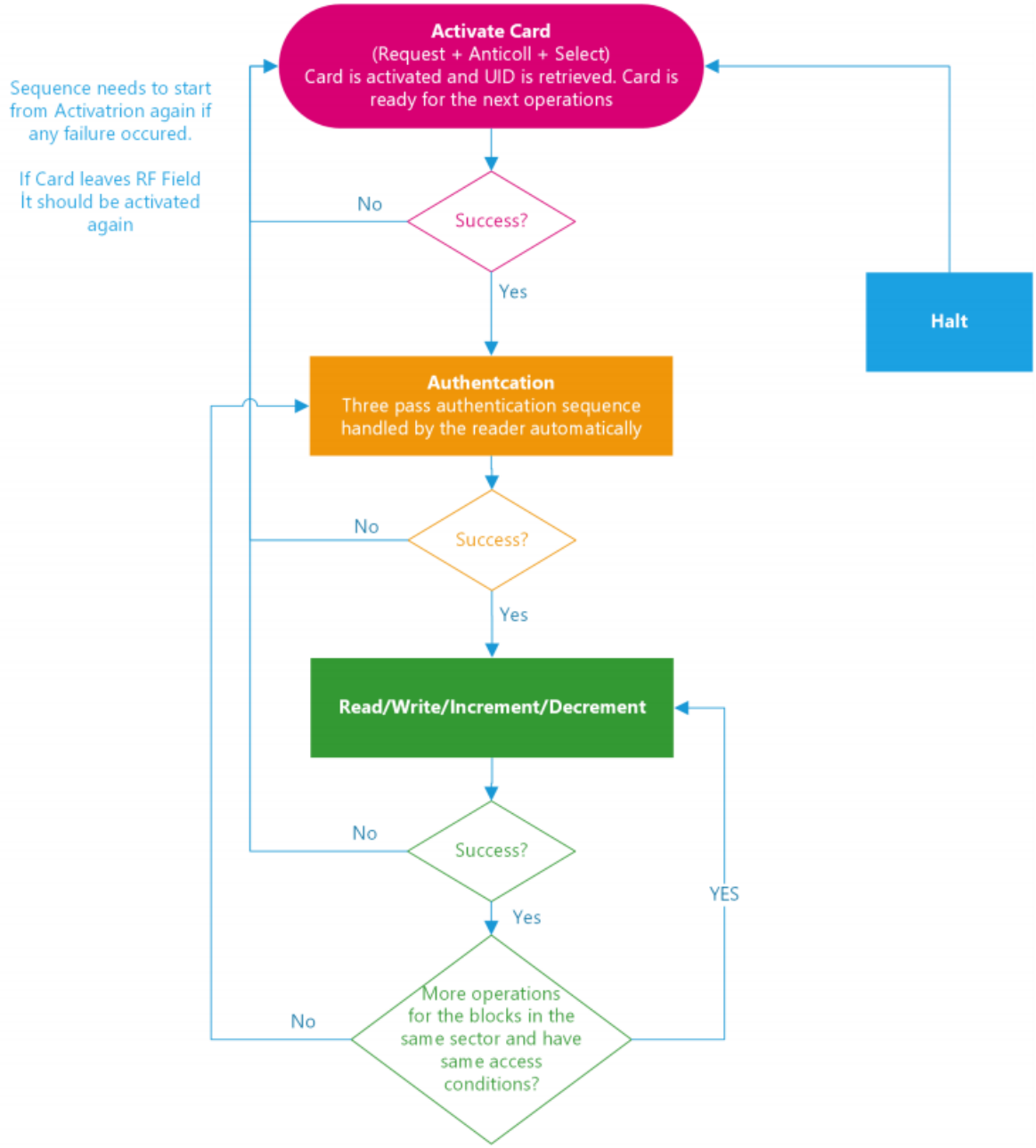


Fig. 4.4 Flow chart to read/write data block

Topic 5: Read/Write Cloud Data by Arduino

Components list:

- Arduino: Mega 2560
- Wireless transceiver ESP8266 module

A. Introduction of ThingSpeak

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts. Its website

ThingSpeak offers free data storage and analysis of time-stamped numeric or alphanumeric data. Users can access ThingSpeak by visiting http://thingspeak.com and creating a ThingSpeak user account. The user account is associated with the MathWorks account and students' CityU email account has already been registered as MathWorks account. If not, students can register free accounts.

To register MathWorks, you can visit

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

Documentation for ThingSpeak

- https://www.mathworks.com/help/thingspeak/index.html

A.1 Upload/Download Data

ThingSpeak stores data in channels. Channels support an unlimited number of timestamped observations (think of these as rows in a spreadsheet). Each channel has up to 8 fields and each field store one value. Fig. 5.1 shows an example of a channel with 3 fields. The channel ID is set by ThingSpeak system and the channel name and field names can be set by user.

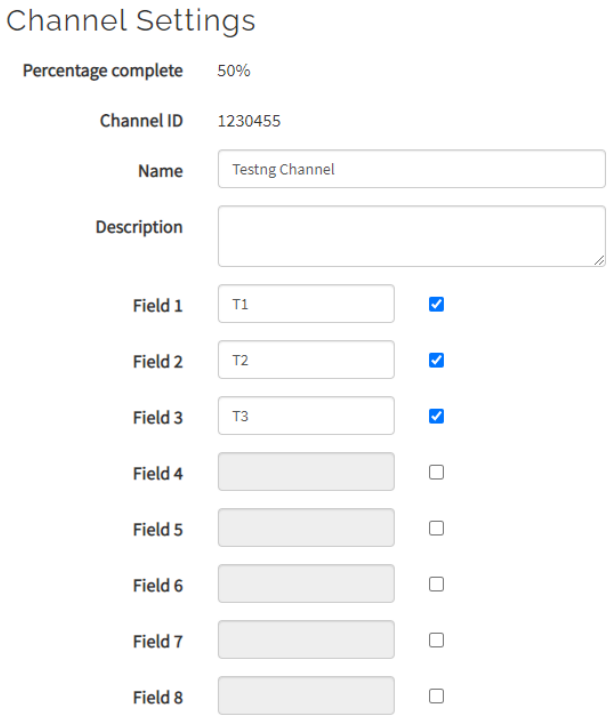


Fig.5.1: Example of Channel settings in ThingSpeak

Self-Reading Materials

Read the following materials to understand more about the channel settings.
https://www.mathworks.com/help/thingspeak/collect-data-in-a-new-channel.html

To read/write the field, you'll need to have the Read/Write API keys. For example.

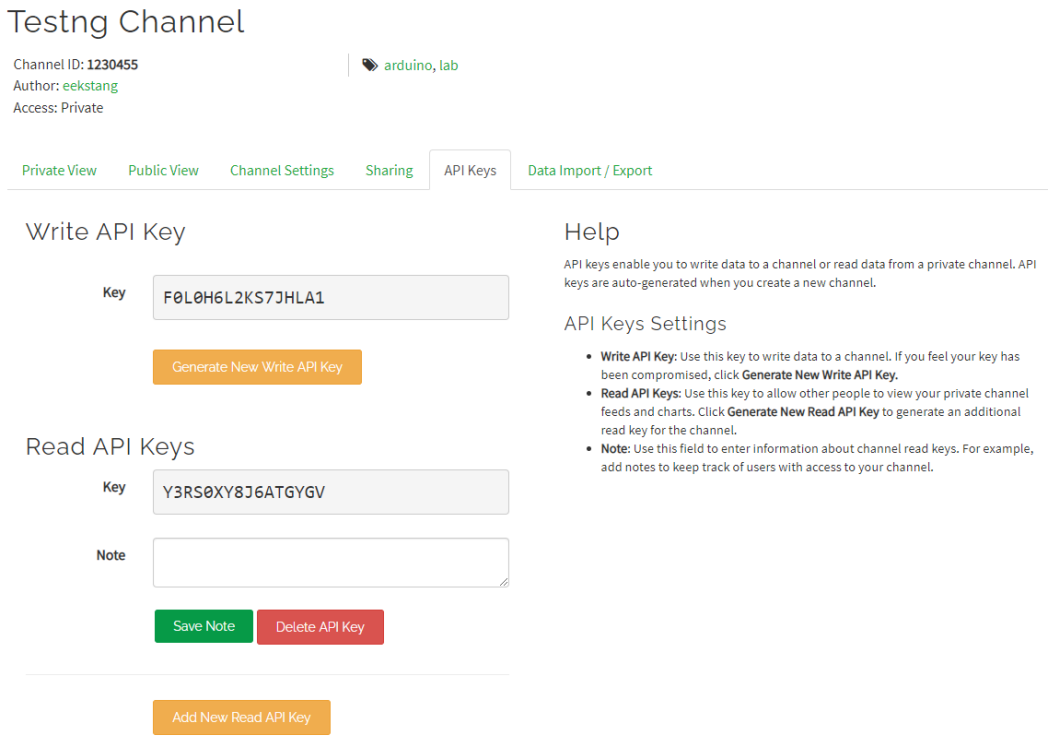


Fig.5.2: Example of the page shown at the API Keys submenu

In the cloud website, we can view and analysis those unlimited historic data. The channel can be set as private (API keys are needed to read/write) or public (everyone) or some specified ThingSpeak users.

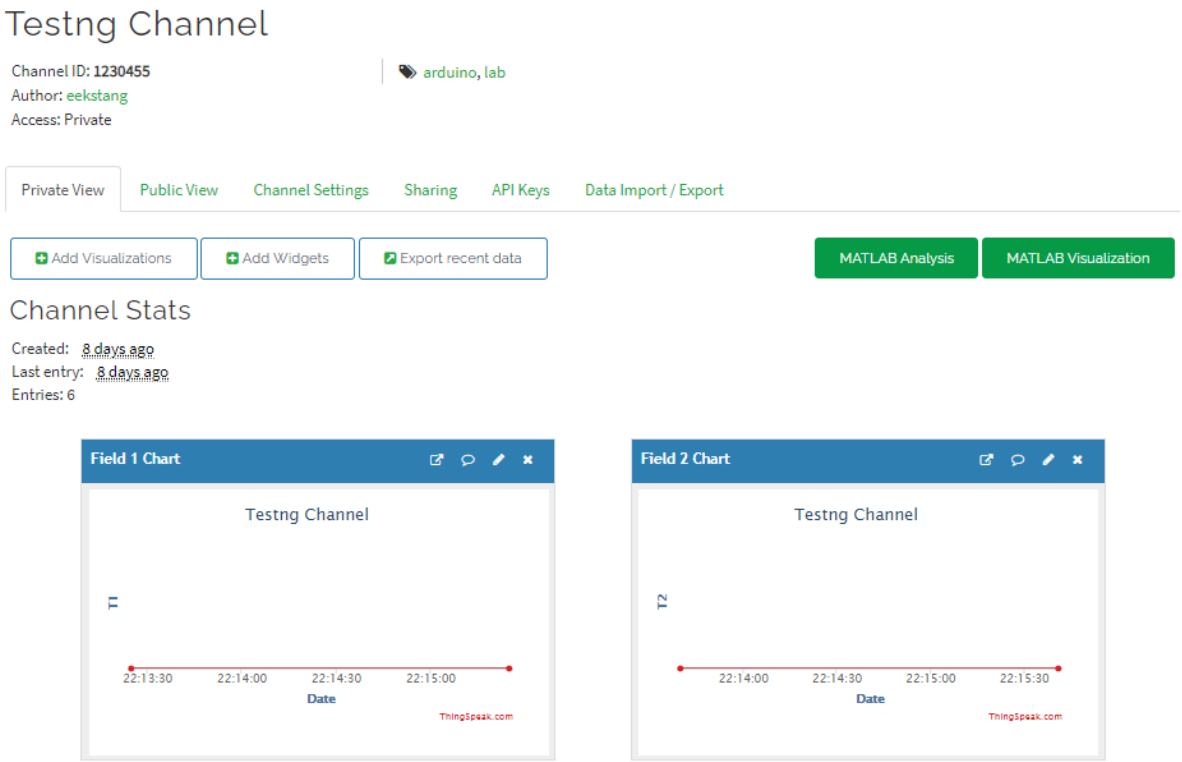


Fig.5.3: Channel statistics in ThingSpeak website

A.2 Limitation of Free Account

Read the license FAQ: https://thingspeak.com/pages/license_faq

Users with free account will be limited to sending no more than 3 million messages each year to the ThingSpeak service. They will also be limited to 4 channels, and the message update interval limit remains limited at 15 seconds.

B. Introduction on Wireless Communications

In order to access the ThingSpeak, we'll need to access the Internet. That is done via the wifi. The basic idea is as follows.

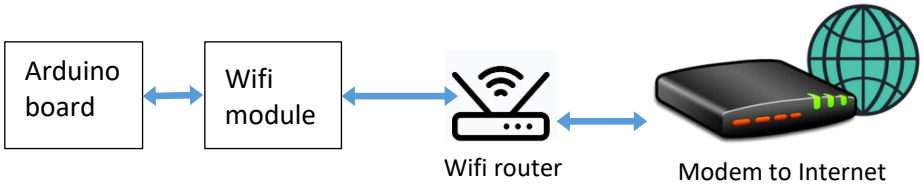


Fig. 5.4. Data flow for Internet access using Arduino

There is a Wifi router in each lab, which can access the Internet. The router is the access point (AP). It has a unique ID or the SSID, so that you can differentiate different AP and connect to a particular one. [Ask technician for SSID and password]

Once the Arduino accesses the Internet, it can read/write fields in the channel in ThingSpeak using the API-Key (with some functions in ThingSpeak.h library)

B.1. ESP8266 Wifi module

To have the wifi connection capability, a wifi module (part number ESP8266) is needed. Fig. shows the phot of ESP8266 and its pinout.

Top view

GND			RX
TX	CP	RST	VCC

Fig. 5.5. Wifi module ESP8266 and its pin layout

B.2 Hardware Connection

Arduino boards have built in support for serial communication on pins 0 (RX) and 1 (TX). However, it is reserved for the communication between Arduino and the COM port of your laptop. Therefore, if we want to use the Serial monitor, other Serial pins are to be used to connect the wifi module.

An example is given below.

Table 5.1 Example of wiring between ESP8266 and Serial 1 of Arduino Mega

ESP8266	Arduino Mega
GND	GND
GPIO 2	
GPIO 0	
RX	TX1
TX	RX1
CH_PD	3.3V
RST	
Vcc	3.3V

Note: The CH_PD has been connected to Vcc in the project platform board. TX1 and RX1 are the transmitter and receiver pins for Serial1.

- Note: Serial, Serial1, Serial2, Serial3 are predefined for Arduino Mega

Self-Reading Materials

To understand more, read the following materials
https://create.arduino.cc/projecthub/user720003162/connecting-esp8266-01-to-arduino-uno-mega-and-blynk-194f17

B.3 Software Design

B.3.1 Library

You'll need to install the following libraries to program the ESP8266 and access Cloud Service in ThingSpeak:

1. ThingSpeak.h : for upload/read data from the cloud platform ThingSpeak
2. WiFiEsp.h : allows the Arduino board to connect to the Internet via ESP8266 WIFI module

Note: See Laboratory Manual (Arduino IDE and Programming) to learn how to add a library

Self-Reading Materials

To understand more, read the following materials
https://github.com/mathworks/thingspeak-arduino

https://github.com/bportaluri/WiFiEsp

B.3.2 Brief descriptions on functions, enum and structures

Some key functions for programming are given below (Note: The libraries in Sec. B.2.1 must be included.)

- Initialization the ESP8266

```
#define ESP_BAUDRATE 19200

// initialize Serial1 for ESP module. The default baud rate of the ESP is set as 19200 bps
Serial1.begin(ESP_BAUDRATE);

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

- Initialization the client for ThingSpeak

```
WiFiEspClient  client;     //create a wifi client
ThingSpeak.begin(client);   // Initialize ThingSpeak
```

- Reading/writing a field
  - If it is public, no API key is needed.
    ```
    ThingSpeak.readFloatField(weatherStationChannelNumber, temperatureFieldNumber);
    ```
  - If it is private, API key (read key or write key) is needed.
    ```
    ThingSpeak.readFloatField(myChannelNumber, i, myReadAPIKey);
    ThingSpeak.writeField(myChannelNumber, i+1, upload_number[i], myWriteAPIKey);
    ```
  - To check whether the read/write success. We can get the status value and, 200 implies success.
    ```
    statusCode = ThingSpeak.getLastReadStatus();   // 200 is success
    ```

B.3.3 Program Example

The following is an example program to interface with a wifi module

```
#include "WiFiEsp.h"

#define ESP_BAUDRATE 19200
char ssid[] = "Twim"; // your network SSID (name)
char pass[] = "12345678"; // your network password
int status = WL_IDLE_STATUS; // the Wifi radio's status

void setup() {
    // initialize serial for debugging
    Serial.begin(115200);
    Serial1.begin(ESP_BAUDRATE);


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

    // Connect or reconnect to WiFi
    if(WiFi.status() != WL_CONNECTED){
         Serial.print("Attempting to connect to SSID: ");
         Serial.println(ssid);
         while(WiFi.status() != WL_CONNECTED){
           WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network
           Serial.print(".");
           delay(5000);
    }
     Serial.println("\nConnected");
}

void loop()
{
  // check the network connection once every 10 seconds
  Serial.println();
  printWifiData();
  delay(10000);
}

void printWifiData()
{
  // print your WiFi shield's IP address
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

}
```

Basic Design:

Before you start, you need to get familiar with the channel definition in ThingSpeak. Then, a private channel containing two fields, and work on the following design tasks.

a) Design Task 1:

Write a program to download a value from a public channel (Note: You may refer to examples in ThingSpeak library, eg. ReadField)

b) Design Task 2:

Write a program to upload two random values to the fields in your private channel in every minute. In total, upload 5 sets of values in 5 minutes.

c) Design Task 3:

Write a program to download all the records of first field for the last three minutes, according to the level of an I/O pin (eg. It becomes high), then output the average to Serial monitor.

CHECKPOINT: SHOW THE RESULT AND EXPLAIN TO TUTOR FOR EACH TASK

Questions to be answered in report:

Q1. If Serial2 is used, which pin (give pin number) of an Arduino Mega is to connect to RX of ESP8266? Which pin of Arduino Mega is to connect to TX of ESP8266?

Q2. Can we update two data to two different fields of a channel with a single upload? If yes, describe how.

Q3. What data type that we can store in the field of a channel?

Advanced Design: You work on this only after finished tasks in Basic Design

a) Advanced Task 1

It is also possible to use HTML to get data from ThingSpeak. You may refer to a simple example (refer to the below link).

http://matternett.blogspot.com/2016/09/fetching-data-from-thingspeak-iot-cloud.html

To use it, please notice

a) The latest version of jquery is 3.6.0. You have to change the code in HTML accordingly

b) The url should be updated. If it is a public channel, no password is needed. For example, we may have

```
url =
'https://api.thingspeak.com/channels/1293177/feeds.json?results=1';
```

Modify the html and get the results from your private channel

b) Advanced Task 2

You can also design a user interface (UI) in MATLAB using App Designer. You may refer to https://www.mathworks.com/products/matlab/app-designer.html

Design a simple interface such that

- Button A: send a random data to the fields of your private channel
- Button B: get results from your private channel and display it.

An example of interface is shown below: