



01076001

วิศวกรรมคอมพิวเตอร์เบื้องต้น

Introduction to Computer Engineering

Arduino #7

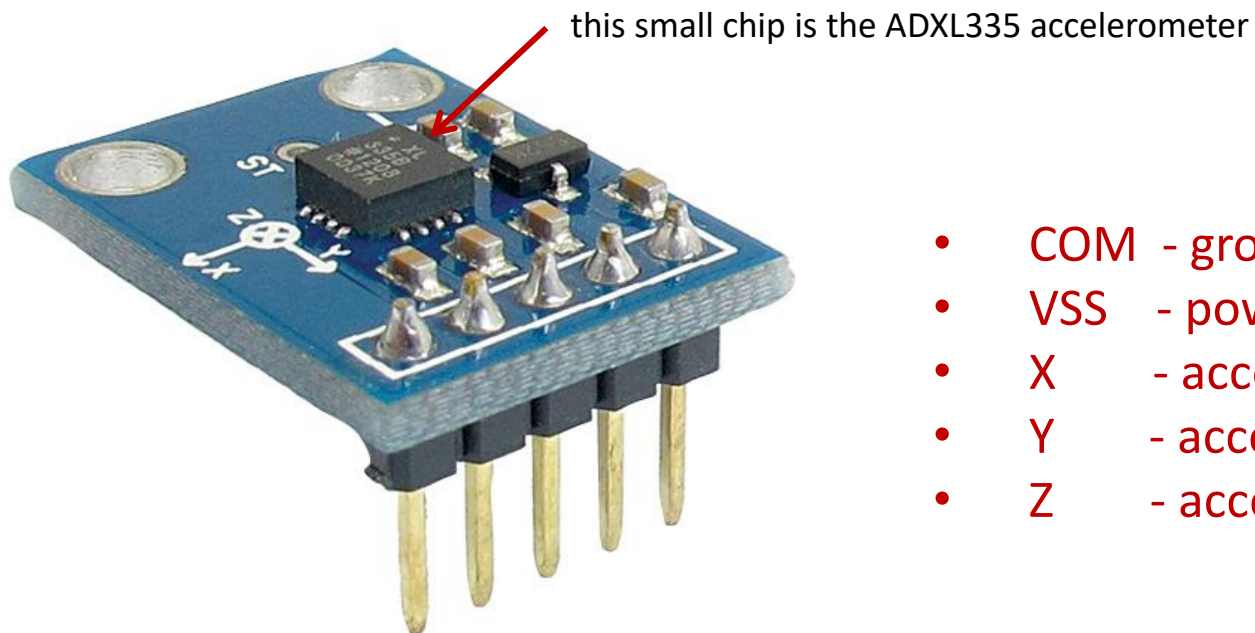
Timer Interrupt, Low Level I/O

Accelerometer

# ADXL 335

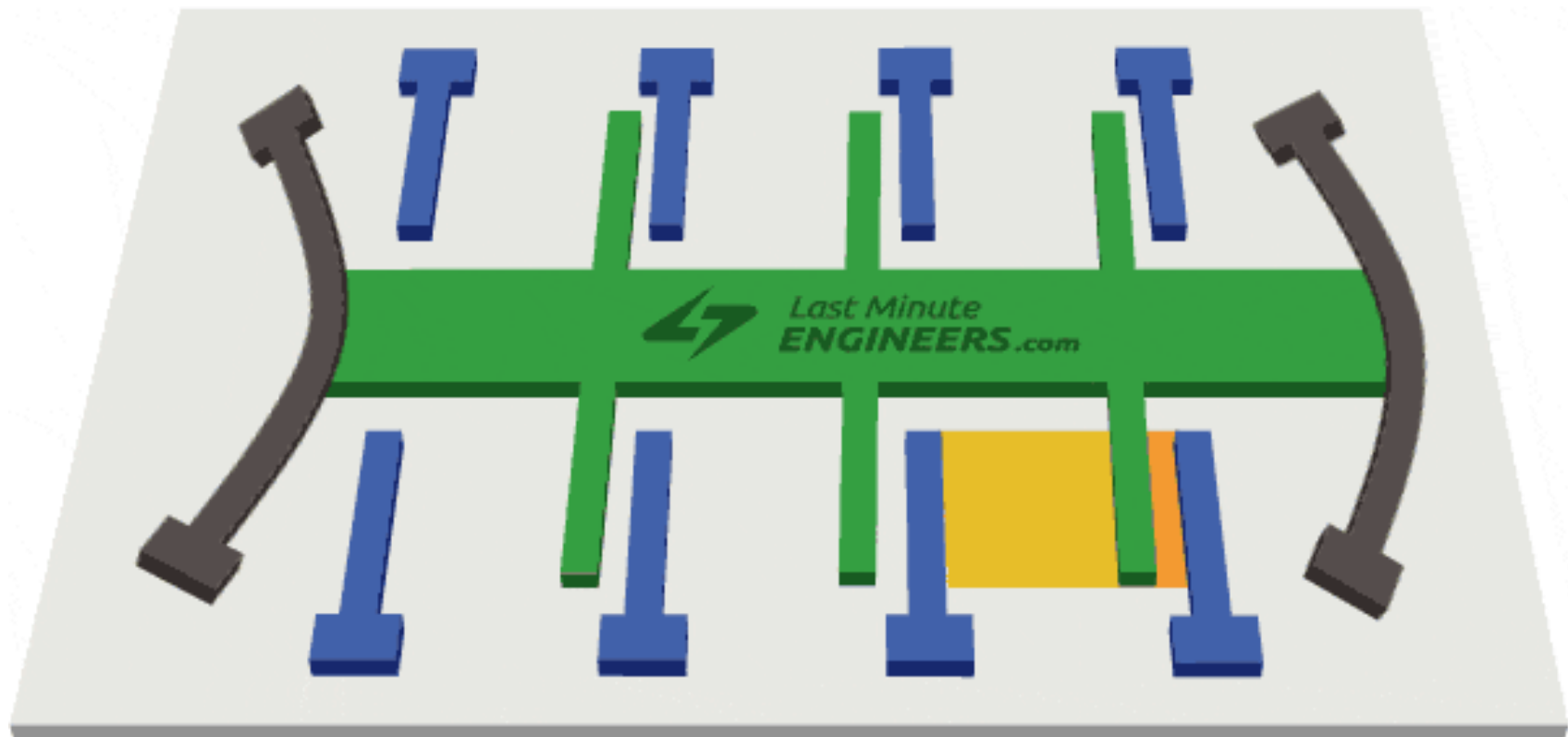


- ใช้วัดความเร่ง เบอร์ ADXL335 ซึ่งเป็นอุปกรณ์ประเภท MEMS
- มีรูสำหรับยึดกับอุปกรณ์ที่ต้องการ
- สามารถวัดความเร่งได้ 3 แกน



- COM - ground
- VSS - power (we will provide 5V)
- X - acceleration in x-direction
- Y - acceleration in y-direction
- Z - acceleration in z-direction

# ADXL 335



# การใช้งาน Accelerometer

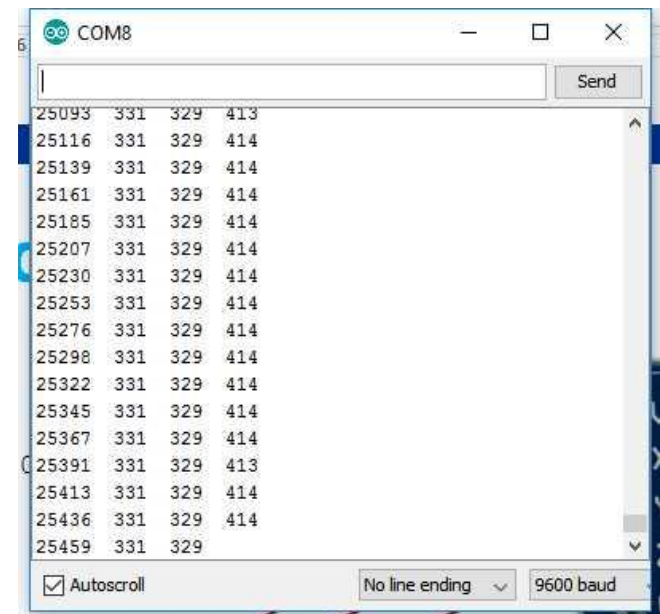
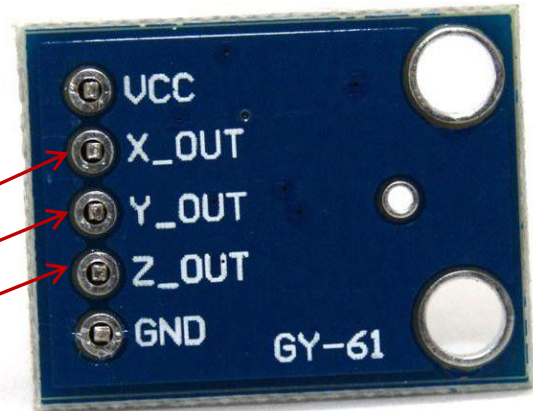


```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int xaccel = analogRead(A0);
  int yaccel = analogRead(A1);
  int zaccel = analogRead(A2);
  unsigned long timevar = millis();

  Serial.print(timevar);
  Serial.print(" ");
  Serial.print(xaccel);
  Serial.print(" ");
  Serial.print(yaccel);
  Serial.print(" ");
  Serial.println(zaccel);
}
```

associates a time with  
each set of accelerations



## Activity ทดสอบ ADXL335

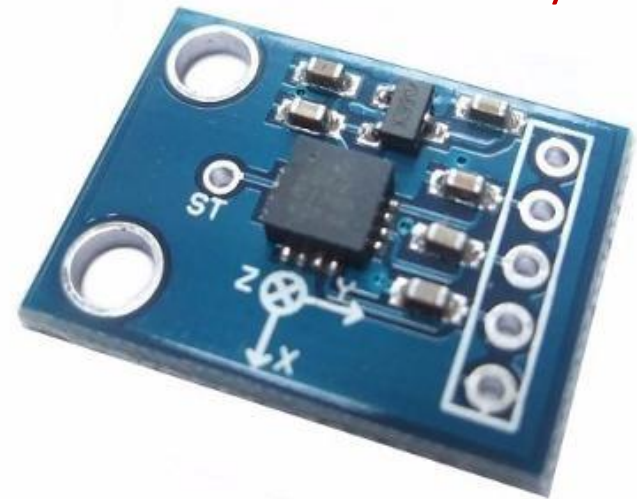
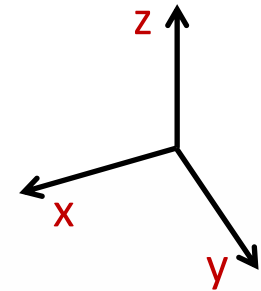


- ให้ใช้โปรแกรมจากหน้าที่แล้ว จากนั้นทดลองหมุนตามแกนต่างๆ ว่า ข้อมูลที่ส่งกลับมาใน Serial Monitor มีการเปลี่ยนแปลงหรือไม่

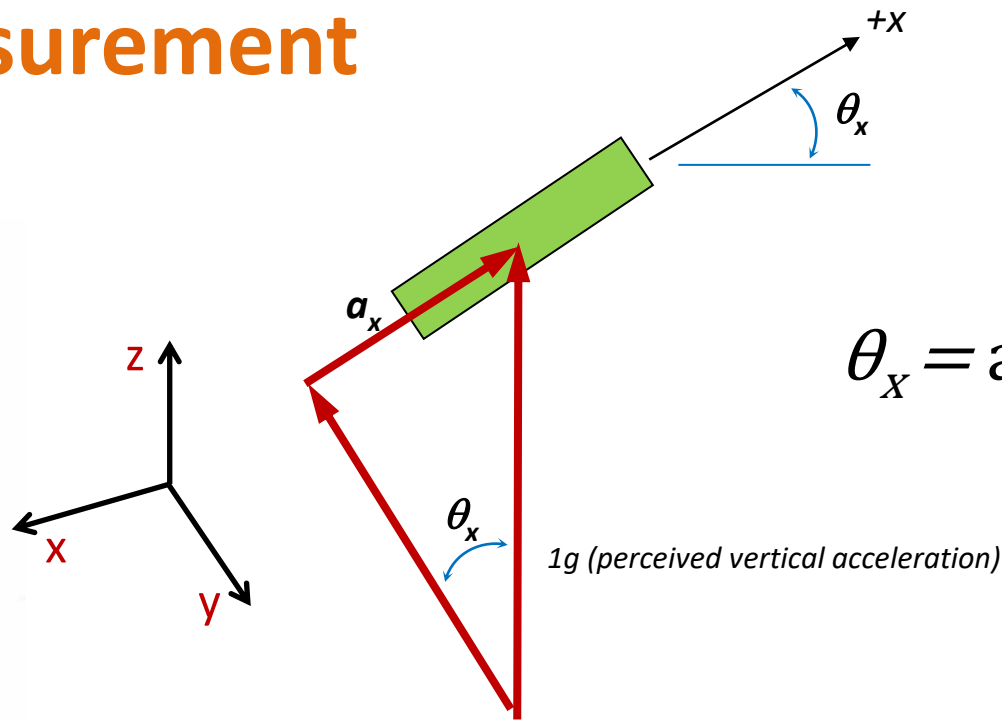
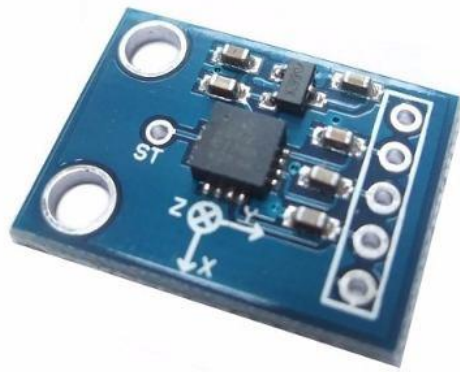


# ADXL335 Calibration

- เนื่องจากภายในเป็นกลไก ซึ่งการอ่านค่าแต่ละตัวอาจจะไม่เท่ากัน ดังนั้นจึงต้อง Calibrate ก่อน เพื่อหาช่วงข้อมูล มิฉะนั้นเมื่อนำไปใช้จะอ่านข้อมูลผิด
- ค่า X เมื่อหันขึ้นด้านบนบน = 266
- ค่า X เมื่อหันด้านล่าง = 399
- ค่าความแตกต่าง =  $399 - 266 = 133$
- ค่าความเร่งเปลี่ยน =  $2g$
- ดังนั้นค่าความเร่งต่อ  $g = 133 / 2 = 66$
- ค่าเมื่อวางในแนวราบ =  $399 - 66 = 333$



# Angle Measurement



- สมมติว่า Arduino อ่านค่าได้ 333 เมื่อวาง accelerator ในแนวราบ และเปลี่ยนแปลง 66 ต่อ 1 g
- **คำถาม** : Arduino จะให้ output เท่าไร เมื่อวางบอร์ดเอียง 45 องศา?
  - $a_x = \sin(45).g = 0.707g$   
 $= 333 + 0.707g * 66/g = 379$

# ADXL335 Buffering



- ในการรับข้อมูลแต่ละครั้ง อาจมีการกระโดดของค่า (เรียกว่า jitter) ดังนั้นควรมีการเฉลี่ยค่า เพื่อให้ข้อมูลมีความนิ่งมากขึ้น

```
const unsigned int X_AXIS_PIN = 2;
const unsigned int Y_AXIS_PIN = 1;
const unsigned int Z_AXIS_PIN = 0;
const unsigned int NUM_AXES = 3;
const unsigned int PINS[NUM_AXES] = {
  X_AXIS_PIN, Y_AXIS_PIN, Z_AXIS_PIN
};
const unsigned int BUFFER_SIZE = 16;
const unsigned int BAUD_RATE = 9600;
int buffer[NUM_AXES][BUFFER_SIZE];
int buffer_pos[NUM_AXES] = { 0 };

void setup() {    Serial.begin(BAUD_RATE); }

int get_axis(const int axis) {
  delay(1);
  buffer[axis][buffer_pos[axis]] = analogRead(PINS[axis]);
  buffer_pos[axis] = (buffer_pos[axis] + 1) % BUFFER_SIZE;
  long sum = 0;
  for (unsigned int i = 0; i < BUFFER_SIZE; i++)
    sum += buffer[axis][i];
  return round(sum / BUFFER_SIZE);
}
```

```
int get_x() { return get_axis(0); }
int get_y() { return get_axis(1); }
int get_z() { return get_axis(2); }
void loop() {
  Serial.print(get_x());
  Serial.print(" ");
  Serial.print(get_y());
  Serial.print(" ");
  Serial.println(get_z());
}
```



## Activity ทดสอบ ADXL335



- ให้ใช้โปรแกรมจากหน้าที่แล้ว จากนั้นทดลองหมุนตามแกนต่างๆ ว่า ข้อมูลที่ส่งกลับมาใน Serial Monitor มีความนิ่งขึ้นหรือไม่

# Memory



- ปกติจะแบ่งหน่วยความจำของคอมพิวเตอร์ออกเป็น ROM และ RAM
  - ROM (Read Only Memory) บางครั้งเรียกว่า Non-Volatile Memory เนื่องจากความจำจะไม่หายไปแม้ไม่มีไฟเลี้ยง
    - PROM = Programmable Read Only Memory
    - EPROM = Erasable Programmable Read Only Memory สามารถลบได้โดยใช้แสง Ultraviolet การโปรแกรมใหม่ต้องใช้วงจรพิเศษ
    - EEPROM = Electrically Erasable Programmable Read Only Memory สามารถลบได้โดยใช้ไฟฟ้า การโปรแกรมจะใช้ Programmed In-circuit หน่วยความจำ Flash ก็อยู่ในกลุ่มนี้
  - RAM (Random Access Memory)
    - Static RAM มีความเร็วในการทำงานสูง แต่ความจุต่อพื้นที่ต่ำกว่า ทำให้มีราคาแพง
    - Dynamic RAM มีความเร็วในการทำงานต่ำกว่า แต่ความจุต่อพื้นที่สูงกว่า ทำให้มีราคาถูก



EPROM



# EEPROM

- ใน Arduino มี EEPROM ขนาด 1024 **Kbit** ซึ่งสามารถอ่านเขียนได้ใหม่ประมาณ 100,000 ครั้ง
- EEPROM จะอยู่คนละส่วนกับหน่วยความจำที่เป็น Bootloader และที่เก็บโปรแกรม
- คำสั่งที่ใช้
  - EEPROM.read(ADDR) อ่านค่า byte ที่เก็บไว้ใน address ที่ต้องการ โดย ADDR ใช้ได้ตั้งแต่ 0-1023
  - EEPROM.write(ADDR,value) เขียนค่า byte ที่เก็บไว้ใน address ที่ต้องการ โดย value มีค่าตั้งแต่ 0-255
  - EEPROM.update(ADDR,value) คล้ายกับ write แต่จะตรวจสอบก่อนว่าค่า value ต่างไปหรือไม่ ถ้าเหมือนเดิมก็จะไม่เขียน เพื่อเป็นการรักษารอบการใช้งานเอาไว้

# EEPROM



```
#include <EEPROM.h>

const int buttonPin = 6;
const int ledPin = 13;

int ledState;
int buttonState;
int lastButtonState = LOW;
long debounceTime = 0;
long debounceDelay = 50;

void setup() {
    Serial.begin(9600);
    pinMode(buttonPin, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    ledState = EEPROM.read(0);
    digitalWrite(ledPin, ledState);
}
```

```
void loop() {
    int reading = digitalRead(buttonPin);
    if(reading != lastButtonState){
        debounceTime = millis();
    }
    if((millis()- debounceTime) > debounceDelay){
        if(reading != buttonState){
            buttonState = reading;
            if(buttonState == LOW){
                ledState = !ledState;
            }
        }
    }
    digitalWrite(ledPin, ledState);
    EEPROM.update(0, ledState);
    lastButtonState = reading;
}
```



# Activity

- ต่อสวิตช์แบบ INPUT\_PULLUP ที่ขา 6
- โหลดโปรแกรมในสไลด์ก่อนหน้านี้
- สังเกตว่า LED on board ที่ขา 13 ติดหรือไม่
- กดสวิตช์ สังเกตว่า LED on board เปลี่ยนสถานะ
- ให้กด reset และสังเกตว่า LED on board คงสถานะเดิมหรือไม่



# EEPROM

- กรณีที่ต้องการเขียนหรืออ่านข้อมูลหลายไบต์ จะใช้คำสั่ง `put()` และ `get()`
- ดูตัวอย่างได้จาก File -> Examples -> EEPROM -> `EEPROM_write` และ File -> Examples -> EEPROM -> `EEPROM_read`



# EEPROM

```
#include <EEPROM.h>

struct MyObject {
    float field1;
    byte field2;
    char name[10];
};

void setup() {

    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }
    float f = 123.456f; //Variable to store in EEPROM.
    int eeAddress = 0; //Location we want the data to be put.
    EEPROM.put(eeAddress, f);
    Serial.println("Written float data type!");

    //Data to store.
    MyObject customVar = {
        3.14f,
        65,
        "Working!"
    };
    eeAddress += sizeof(float); //Move address to the next byte after float 'f'.
    EEPROM.put(eeAddress, customVar);
    Serial.print("Written custom data type! \n");
}
```

```
void loop() {
    /* Empty loop */
}
```



# EEPROM

```
#include <EEPROM.h>

void setup() {

    float f = 0.00f;    //Variable to store data read from EEPROM.
    int eeAddress = 0; //EEPROM address to start reading from

    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }
    Serial.print("Read float from EEPROM: ");
    EEPROM.get(eeAddress, f);
    Serial.println(f, 3);
    secondTest(); //Run the next test.
}

struct MyObject {
    float field1;
    byte field2;
    char name[10];
};

void secondTest() {
    int eeAddress = sizeof(float); //Move address to the next byte after float 'f'.

    MyObject customVar; //Variable to store custom object read from EEPROM.
    EEPROM.get(eeAddress, customVar);
    Serial.println("Read custom object from EEPROM: ");
    Serial.println(customVar.field1);
    Serial.println(customVar.field2);
    Serial.println(customVar.name);
}
```



# Activity



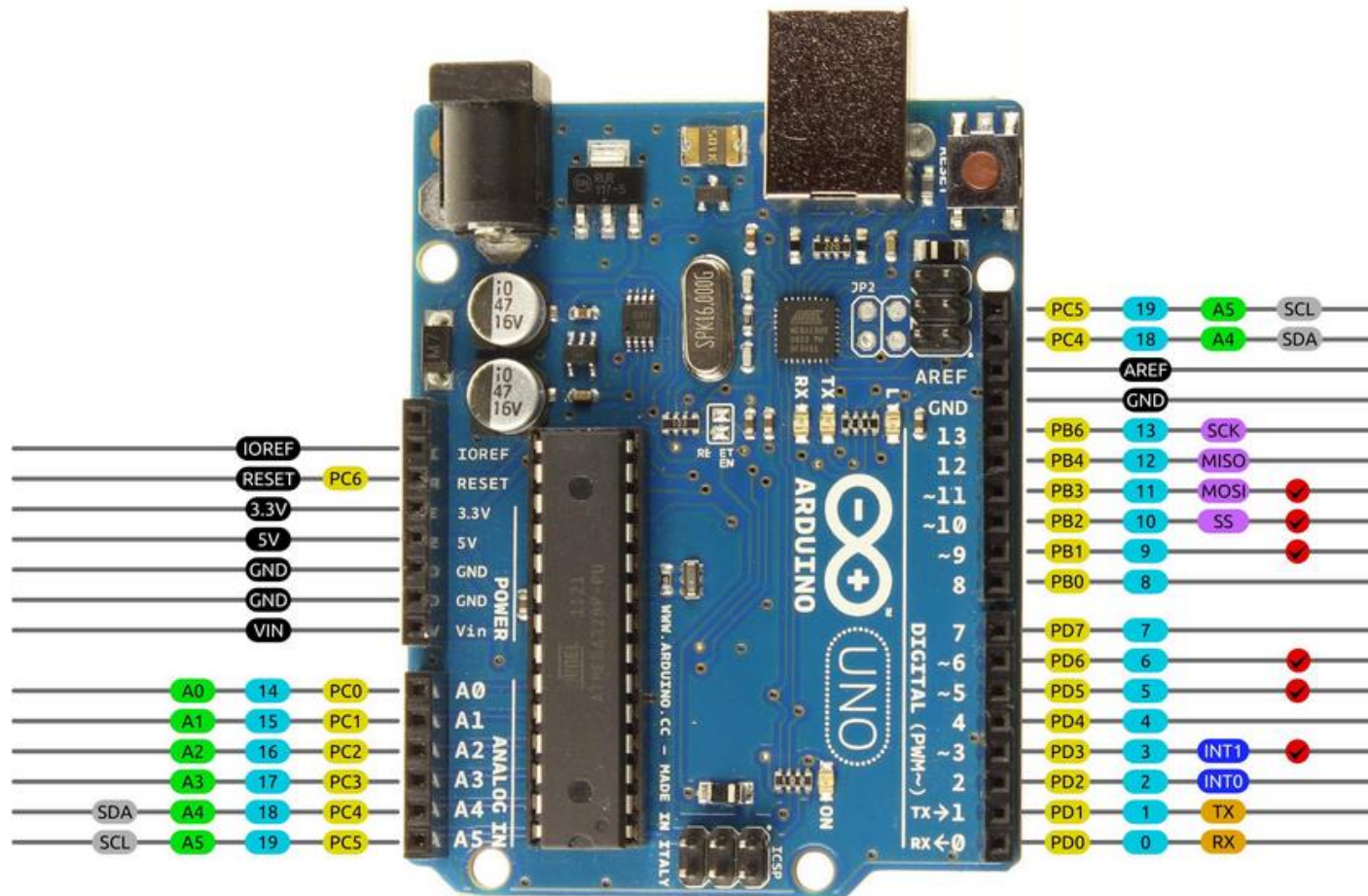
- โหลดโปรแกรมในสไลด์ที่ 14 ซึ่งเป็นโปรแกรมที่เขียนค่าลง EEPROM
- โหลดโปรแกรมในสไลด์ที่ 15 แล้วสังเกตว่าโหลดค่าที่อ่านไว้ก่อนหน้านี้ได้หรือไม่



# Low Level I/O

- นอกเหนือจากการใช้คำสั่ง digitalWrite และ digitalWrite แล้ว เรายังสามารถใช้การทำงานในระดับ Low Level ในการสั่งงานแต่ละ pin ของ Arduino ได้
- ก่อนอื่นต้องทำความเข้าใจ
  - Port หมายถึงช่องทางเข้าออกของข้อมูล ปกติ 1 Port จะมีขนาด 8 บิต
  - Pin หมายถึงแต่ละบิตที่อยู่ใน Port แต่นำมาเรียกใหม่เป็น Pin เพื่อให้การเรียกใช้งานทำได้สะดวกกว่า เป็นมิตรกับผู้ใช้งานมากกว่า
  - จากรูปใน Slide ถัดไป จะเห็นว่า
    - A0-A6 ตรงกับ PC0-PC5
    - D0-D7 ตรงกับ PD0-PD7
    - D8-D13 ตรงกับ PB0-PB6

# Low Level I/O



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT



# Low Level I/O

- ในการกำหนดให้ Pin ใด เป็น 0 หรือ 1 เราต้องใช้ Bit Level
- เช่นต้องการให้ Pin 13 มีค่าเป็น 1 จะใช้คำสั่ง

```
PORTB |= (1 << PORTB6);
```

- หรือต้องการให้ Pin 13 มีค่าเป็น 0 จะใช้คำสั่ง

```
PORTB &= ~(1 << PORTB6);
```

- นอกจากนั้นยังสามารถใช้ macro

```
PORTB |= _BV(PORTB6); // _BV(x) is the same as (1 << x)
```



# Low Level I/O

- ในแต่ละ Port จะมี Register ที่กำกับจำนวน 3 ตัว เช่น Port B จะมี PORTB เก็บค่าที่ต้องการให้ออกเป็น Output, DDRB เก็บทิศทางของ Port ว่าเป็น Input หรือ Output (0=Input, 1=Output) และ PINB เก็บค่าที่อ่านเข้ามา

## 13.4.2 PORTB – The Port B Data Register

| Bit           | 7      | 6      | 5      | 4      | 3      | 2      | 1      | 0      |       |
|---------------|--------|--------|--------|--------|--------|--------|--------|--------|-------|
| 0x05 (0x25)   | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | PORTB |
| Read/Write    | R/W    | R/W    | R/W    | R/W    | R/W    | R/W    | R/W    | R/W    |       |
| Initial Value | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |       |

## 13.4.3 DDRB – The Port B Data Direction Register

| Bit           | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |      |
|---------------|------|------|------|------|------|------|------|------|------|
| 0x04 (0x24)   | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | DDRB |
| Read/Write    | R/W  | R/W  | R/W  | R/W  | R/W  | R/W  | R/W  | R/W  |      |
| Initial Value | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |      |

## 13.4.4 PINB – The Port B Input Pins Address

| Bit           | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     |      |
|---------------|-------|-------|-------|-------|-------|-------|-------|-------|------|
| 0x03 (0x23)   | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | PINB |
| Read/Write    | R     | R     | R     | R     | R     | R     | R     | R     |      |
| Initial Value | N/A   | N/A   | N/A   | N/A   | N/A   | N/A   | N/A   | N/A   |      |



# Low Level I/O

- DDRx (Data Direction Register) เก็บทิศทางของ Port ว่าเป็น Input หรือ Output (0=Input, 1=Output)
- PORTx (Port Register)
  - ถ้า Pin เดียวกันใน DDR เป็น 0 : (Input) จะใช้กำหนดว่าจะใช้ Internal Pull-Up Register หรือไม่ (0 = ไม่ใช่, 1 = ใช่)
  - ถ้า Pin เดียวกันใน DDR เป็น 1 : (Output) : จะใช้ในการกำหนด Logic ของ Pin
- PINx เก็บค่าปัจจุบันของ Pin นั้นๆ เพื่อให้อ่านค่าออกไปได้



# Low Level I/O

- ดังนั้นถ้าเราต้องการให้ Pin 13 กระพริบ สามารถเขียนโปรแกรมได้ดังนี้ (ให้สังเกตขนาดของ code ที่สร้าง)

```
int main()
{
    DDRB = B00100000;

    while(1) {
        PORTB = B00100000; // Turn on LED
        _delay_ms(1000);
        PORTB = B00000000; // Turn off LED
        _delay_ms(1000);
    }
}
```



# Low Level I/O

- หรือหากจะเขียนให้ไม่กระทบกับ Pin อื่นๆ ก็เขียนได้ดังนี้

```
int main()
{
    DDRB |= (1 << 5);
    while(1) {
        PORTB |= (1 << 5);           // Turn on LED
        _delay_ms(1000);
        PORTB &= ~(1 << 5);         // Turn off LED
        _delay_ms(1000);
    }
}
```





# Low Level I/O

- โดย Code ที่ทำงานในระดับ Low Level จะทำงานเร็วกว่า ซึ่งทดสอบโดยโปรแกรมนี้

- ผลลัพธ์

deltaT\_libraries (uS) : 7548

deltaT\_registers (uS) : 1320

- พบว่า Low Level เร็วกว่า  
ประมาณ 5 เท่า

```
void loop(){
    time1 = micros();
    while(i<N) {
        digitalWrite(13,HIGH);
        digitalWrite(13,LOW);
        i++;
    }
    time2 = micros();
    i= 0;
    time3 = micros();
    while(i<N) {
        PORTB = B00100000;
        PORTB = B00000000;
        i++;
    }
    time4 = micros();
    deltaT_libraries = time2-time1;
    deltaT_registers = time4-time3;
    Serial.begin(9600);
    Serial.print("deltaT_libraries (uS) : ");
    Serial.println(deltaT_libraries);
    Serial.print("deltaT_registers (uS) : ");
    Serial.println(deltaT_registers);
    while(1){}
```



# Low Level I/O

- หากต้องการอ่านค่าจาก Pin 2

```
byte input;
void setup() {
    DDRD = DDRD & 0b11111011; // Input pin 2
    PORTB |= 0b00000100;      // Internal Pullup on pin 2

    Serial.begin(9600);
}

void loop() {
    input = PIND & 0b00000100; // if pin 2 == 0 input = 0b00000000
                                // else input = 0b00000100

    Serial.println(input);
    delay(500);
}
```

# Activity



- ทดลองเขียนโปรแกรม Low Level โดยต่อ Switch ที่ขา 2
- กด Switch ให้ LED ขา 13 ติด

# Timer/Counter



- ปกติใน ไมโครคอนโทรลเลอร์ มักจะมีการสร้าง Hardware สำหรับทำงานกับเรื่องเป็นเวลา เช่น ใช้นับเวลาในคำสั่ง millis() หรือ delay() จะเรียก Hardware นี้ว่า Timer หรือ Counter
- ใน ATmega328 ซึ่งเป็นไมโครโปรเซสเซอร์ของ Arduino จะมี Timer อยู่ 3 ตัว คือ Timer0, Timer1 และ Timer 2 โดยมีขนาด 8,16 และ 8 บิตตามลำดับ
- Timer0 มีขนาด 8 บิตจึงนับข้อมูลได้เพียง 255 ใน Arduino จะใช้ Timer0 กับคำสั่ง delay() และ millis()
- Timer2 มีขนาด 8 บิตเช่นกัน โดย Arduino จะใช้กับคำสั่ง Tone และ noTone
- Timer1 มีขนาด 16 บิต จึงสามารถนับค่าได้มากถึง 65,535



# Timer Interrupt

- นอกจาก Interrupt จะเกิดจาก Hardware แล้ว ในระบบคอมพิวเตอร์มักจะมีการสร้าง Interrupt ที่เป็น Software ขึ้นมาด้วย โดยอาศัยการทำงานของ Timer โดยสามารถกำหนดให้มี Interrupt เกิดขึ้นเมื่อ Counter ทำงานแบบ Count Down จนเหลือ 0
- อาจเรียกว่า Timer Interrupt
- Timer ที่เราจะใช้ คือ Timer1 ซึ่งมี 16 บิต ดังนั้นจะนับได้ 65535 โดย Input ของ Timer คือ สัญญาณ Clock ของระบบ (16 MHz) แต่เนื่องจากมีความถี่มากเกินไป ระบบจึงกำหนดให้มีตัวหาร (Prescaler) โดยเลือกได้ 5 ค่า คือ 1, 8, 64, 256, 1024 ซึ่งในระบบของเราจะเลือกตัวหาร 256



# Timer Interrupt

- จากความถี่ 16 MHz เมื่อนำไปหาร 256 จะได้  $16,000,000 / 256 = 62500$  แปลว่าใน 1 วินาทีจะมีสัญญาณมาที่ Timer = 62500 ครั้ง
- เมื่อได้รับ 1 clock ตัว Timer จะเพิ่มค่า 1 เมื่อค่าเพิ่มขึ้นเป็น 65535 และได้รับ clock ต่อไป Timer จะรีเซ็ตกลับเป็นค่า 0 และเกิด Interrupt ซึ่งเราสามารถนำเอา Interrupt ไปใช้งานได้
- นอกจากจะกำหนดตัวหารแล้ว Timer ยังอนุญาตให้เรากำหนดค่าเริ่มต้นสำหรับการนับได้อีกด้วย
- เนื่องจากเราต้องการให้ Interrupt ทุกๆ 1 วินาที เพื่อใช้เป็นตัวนับวินาที เราก็จะต้องกำหนดค่าเริ่มต้น โดยเอา  $65535 - 62500 + 1 = 3036$  ซึ่งจะใช้เป็นค่าเริ่มต้น เมื่อมี clock เข้ามาครบ 62500 ก็จะเป็นเวลา 1 วินาทีพอดี



```
#define ledPin 13
int timer1_counter;
int t1=0;
void setup()
{
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);

    // initialize timer1
    noInterrupts();           // disable all interrupts
    TCCR1A = 0;               // Set Mode of Timer : Normal Mode
    TCCR1B = 0;

    timer1_counter = 3036;    // preload timer 65536-16MHz/256/1Hz

    TCNT1 = timer1_counter;   // preload timer
    TCCR1B |= (1 << CS12);    // 256 prescaler
    TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
    interrupts();             // enable all interrupts
}

ISR(TIMER1_OVF_vect)         // interrupt service routine
{
    TCNT1 = timer1_counter;   // preload timer
    digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
    Serial.println(t1++);
}

void loop()
{
    // your program here...
}
```



# Activity

- ให้โหลดโปรแกรมข้างต้นแล้วทดลองทำงาน พร้อมทั้งทำความเข้าใจกับโปรแกรม
- ข้อเสนอแนะในการเขียน ISR
  - ให้โปรแกรม ISR สั้นเข้าไว้ เพื่อป้องกันไม่ให้เกิดการ Interrupt ซ้ำ
  - อย่าใช้ delay เพราะอาจเกิดการ Interrupt ซ้ำ





# TimerOne Library

- เป็นอีกวิธีในการใช้ Timer1 แต่เป็นการใช้ Library
- ให้ Download Library TimerOne
- ให้ไปที่ File -> Preferences
- Copy ตำแหน่งของ File Sketchbook Location
- เปิด Explorer และไปที่ Libraries แล้ว copy Library ลงไป



# TimerOne Library

```
#include <TimerOne.h>
String LEDStatus = "OFF";
int yellowLED = 8;
int redLED = 9;

void setup()
{
  // Pin 13 has an LED connected on most Arduino boards
  pinMode(yellowLED, OUTPUT);
  pinMode(redLED, OUTPUT);

  Timer1.initialize(500000); // 0.5 s
  Timer1.attachInterrupt( blinkYellow );
  Serial.begin(9600);
}

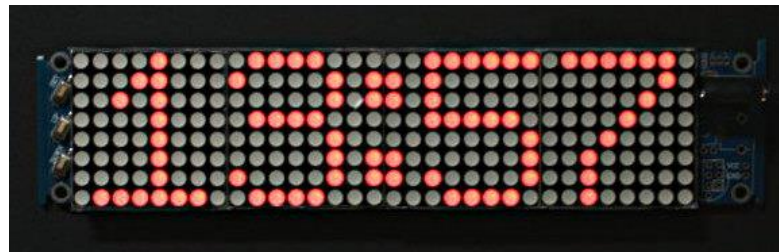
void loop()
{
  digitalWrite(redLED, HIGH);
  delay(1000);
  digitalWrite(redLED, LOW);
  delay(1000);
}

void blinkYellow()
{
  if (LEDStatus == "ON") {
    digitalWrite(yellowLED, LOW);
    LEDStatus = "OFF";
    return;
  }
  if (LEDStatus == "OFF") {
    digitalWrite(yellowLED, HIGH);
    LEDStatus = "ON";
    return;
  }
}
```



# Assignment #7 : mini clock

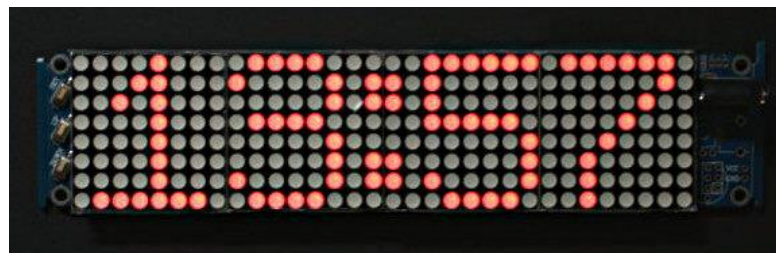
- ให้ใช้แผง LED Dot Matrix หรือ OLED ทำเป็นนาฬิกาดิจิตอล สำหรับความสามารถอื่นให้นักศึกษากำหนดเอง เช่น ตั้งปลุก โดยต้องใช้อุปกรณ์ประกอบ 1 ตัว เช่น ADXL335 (หากกลับข้างให้กลับการแสดงผล) หรือ LDR (หรือแสง)
- ต้องสามารถตั้งเวลาได้ โดยใช้ Volume หรือ Switch หรือ Serial ต้องสามารถ Save เวลาลง EEPROM เพื่อเปิดใหม่จะได้เวลาเดิม (ห้ามเขียนตลอดเวลา เพราะจะทำให้หน่วยความจำเสีย)





# Assignment #7 : mini clock

- การส่งงาน
  - ตรวจสอบโดย Staff
  - รายงาน ประกอบด้วย 1) แนวคิดการออกแบบ (Conceptual Design) 2) การใช้  
งานโดยย่อ 3) โปรแกรมและการอธิบายโปรแกรมโดยย่อ (อธิบายในระดับฟังก์ชัน)
  - คะแนน 10 คะแนน (โครงงานหลัก) เกณฑ์การให้คะแนน 1) ฟังก์ชันการใช้งาน  
2) ลูกเล่น 3) ความละเอียดครบถ้วนของรายงาน 4) Code





*For your attention*