Electronic Bricks Cookbook vol. 1

V1.0a By Freezing Xie, Eric Pan

Preface

Should every child have played toy bricks; it is the most widely owned toy which help construct preliminary inspirations easily. Easy plugging and dissemble, fast and robust, then creativity are maximized by ignoring engineering difficulties.

Similar to electronics for elder inventors: it may not be a problem after you have enough experience and resource to make a prototyping from ground. But so much technical details usually distract the inspirations. How about constructing electronic projects just like Lego brick?

Yes! We can build electronics projects so easily. Arduino and its community have made the programming much easier than ever before. Surely we need some elixir on hardware part. By using electronic bricks, you may connect Arduino compatible boards easily with various digital, analog and I2C/Uart interfaces. These the breadboard-less firm connection are prepared to extensive modules like potentiometers, sensors, relays, servos...even buttons, just plug and play.

Intro

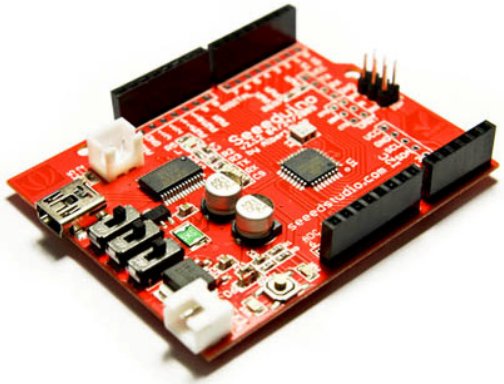
A basic setup of electronic bricks might include 5 parts: MCU boards, adapter shield, connectivity and functional modules.

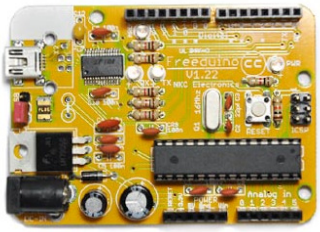
Arduino is based on flexible, easy-to-use hardware and software. These advantages make it became one of the best choices for the MCU part of Electronic bricks. Each functional module has buckled port with VCC, GND and Output, which has corresponding port on the adapter shields, with a plain 2.54mm dual-female cable you may start playing already. Buckled brick cables are like cement to make the connections easier, secure and more professional looking.

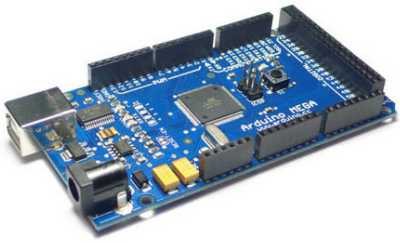
**** Below catalog is for demonstration only, please visit depot for complete list***

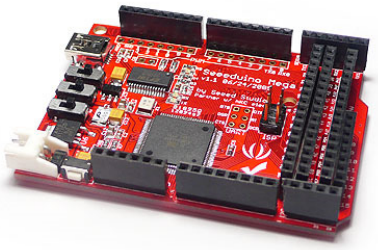
Micro controller units (MCU)

MCU is the director of the circuits, as the core module for electronic bricks. You may use any Arduino pin compatible boards, like Arduino Duemilanove, Seeeduino, Freeduino and etc.



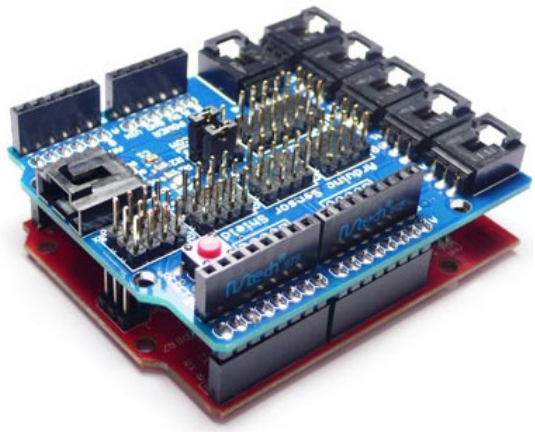


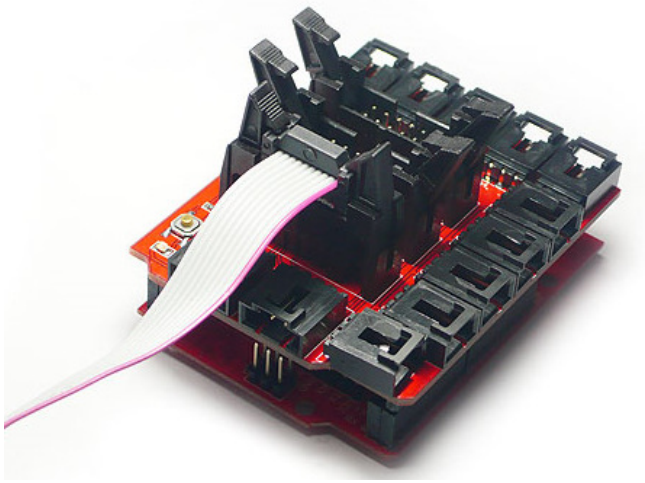




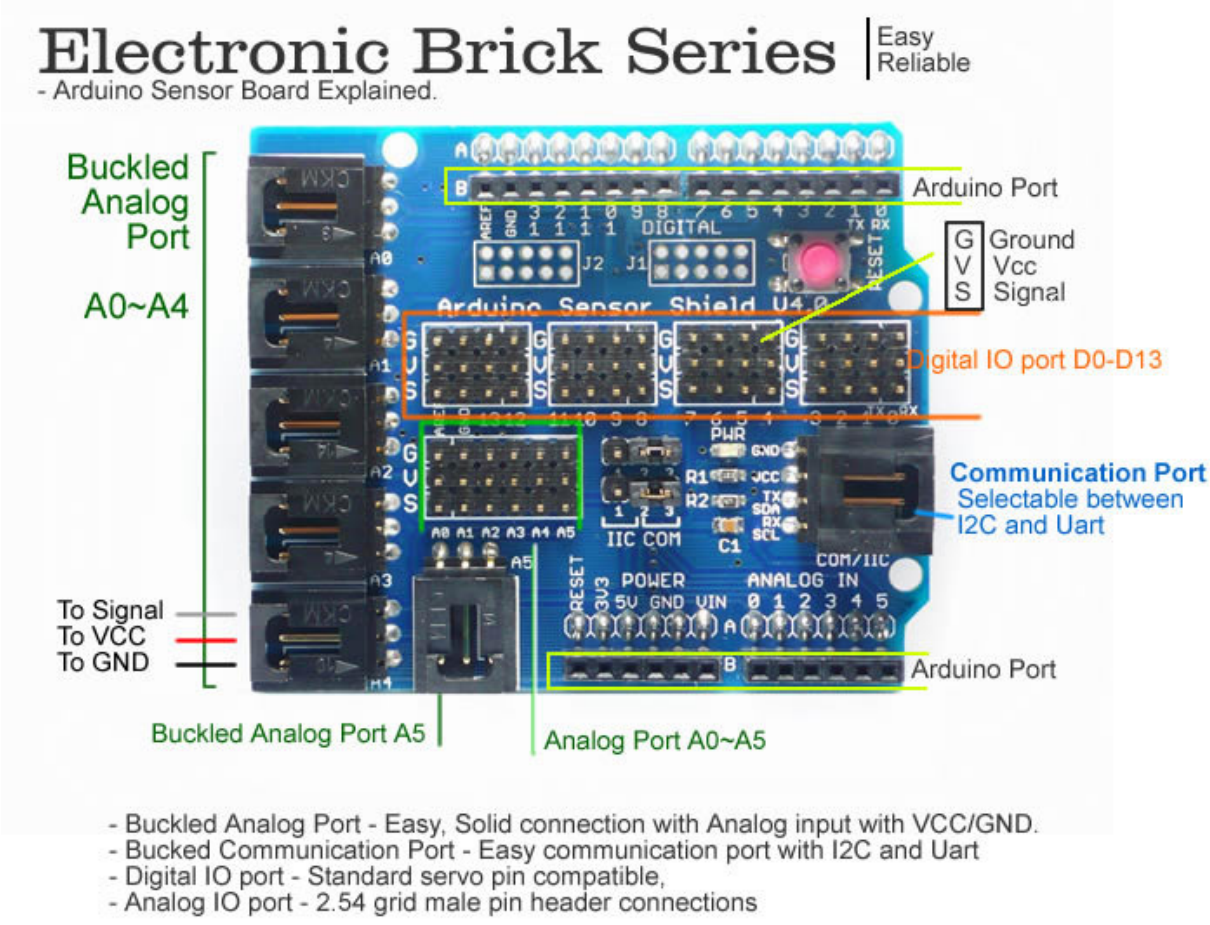
Electronic Brick shield

We provide two kind of Electronic Brick shields: Arduino Sensor Shield and Electronic Brick Chasis

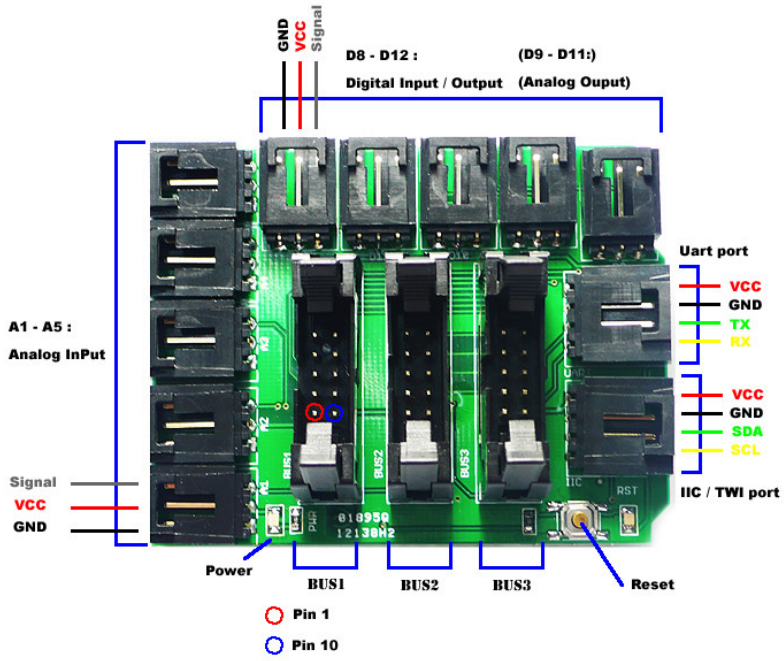




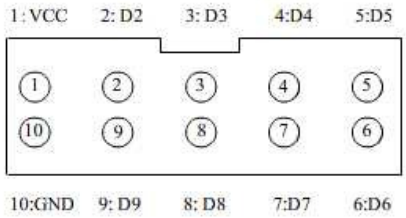
Arduino Sensor shield V4 is a standard footprint Arduino shield, provided stackable pin headers and 5 channel electronic brick connectivity, all digital and Analog IO are prepared with servo compatibility.



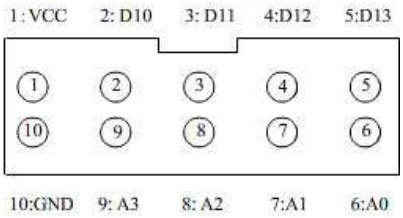
Electronic Brick Chassis is dedicated for electronic brick setup, carrying much more electronic brick connectors, even for data buses. It is more uniformed interface for beginners, on the other hand provide super strong connection.



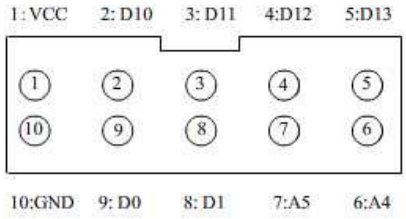
BUS 1



BUS 2



BUS 3



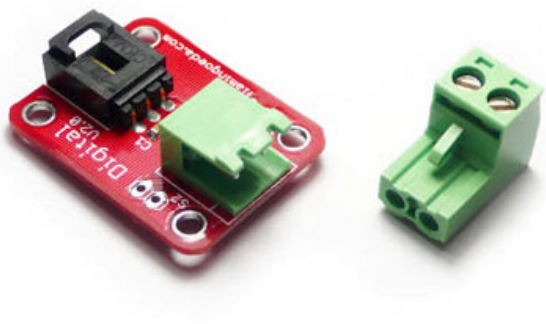
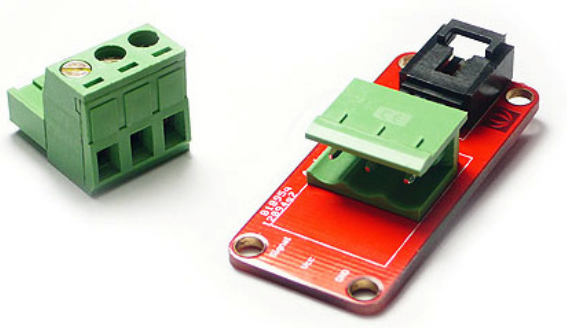
SPI : D10 : SS
D11 : MOSI
D12 : MISO
D13 : SCK

Chassis provides 5 Digital IO, 5 analog input, 1 IIC port and Uart serial port. The other 3 data buses are prepared for more complex devices. These buses could be reprogrammed for other usages too.

BUS1 (D2-D8): for 4*4 keyboard by default.
BUS2 (D10-D17): D10-D13 are SPI, A0-A3 could be used as D14-D17. For 16*2 LCD by default.
BUS3 (D10-D13,D0,D1,A4,A5): D10-D13 are SPI, D0 and D1 are RX/TX. For SD card module.

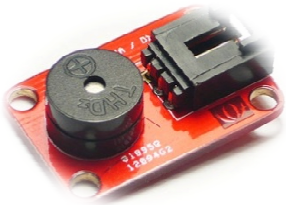
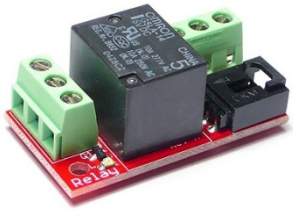
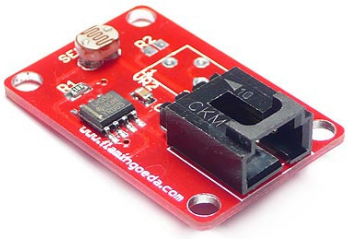
Connectivities

While we recommend buckled cable for easy and firm connection, most 2.54 pitch cable could be used on electronic brick connectors,

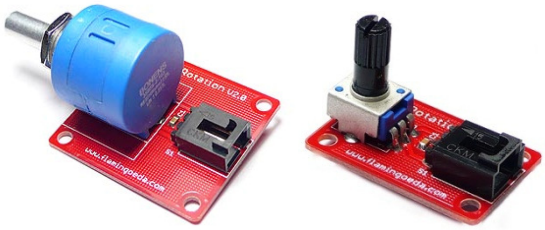
 **Half buckled 3 wire cable:** Compatible with V4 digital port, basic digital module connections.	 **Fully buckled 3 wire cable** Firm connectivity on both ends, basic Analog/Digital module connections.
 **Fully buckled 4 wire cable** Firm connectivity on both ends, for I2C and Uart connections.	 **Fully buckled 10 wire bus cable** Reverse proof, For ISP and data bus connections.
 **Quick plug terminal 2 wire** Easy adaption of various circuits to electronic bricks interface.	 **Quick plug terminal 3 wire** Easy adaption of various circuits to electronic bricks interface.

Functional modules

Electronic bricks could be roughly classes by Input/output/Bus on usage. Or Digital/Analog/mixed on signal type.

 **LED module：** Used as digital IO to produce light, analog IO to produce dimming lights.	 **Buzzer module:** Connect to PWM output for generating sound, or Analog input as piezo touch sensor.
 **Relay module：** Control high voltage circuit by 5V logic digital output.	 **Button module:** Simple basic momentary button for input.
 **PIR module:** Detect human existence, output HIGH when activated.	 **Gas module:** Detect gas, output HIGH when activated.
 **Temperature sensing module:** Output voltage changes according to temperature.	 **Light sensing module:** Output voltage changes according to environmental light strength.

Tilt sensor module: Connect when tilting, disconnect while horizontal.



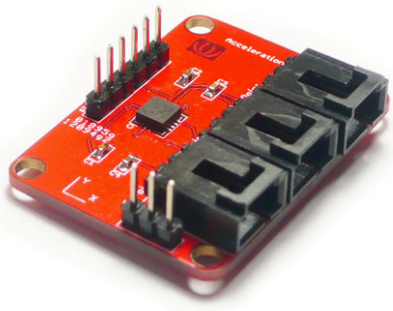
Rotary angle module: Output voltage changes according to rotated angle.

16*2 Character LCD: Display numeric and characters, max 32 in a display.

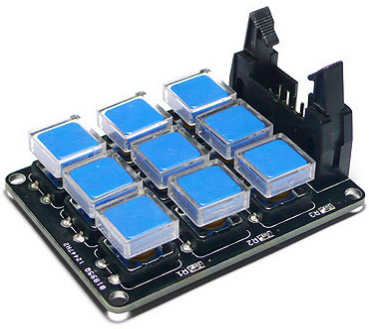
Joystick module: Joystick output 2 axis by analog signal, press down as momentary button to output digital pulse.

Sound sensing module: Output voltage changes according to environmental sound strength.

Distance measurement module: Output voltage changes according to distance detected.



Accelerometer module： Output 3 axis acceleration force by voltage.



Keypad Module： 3×3 scan keypad for input.

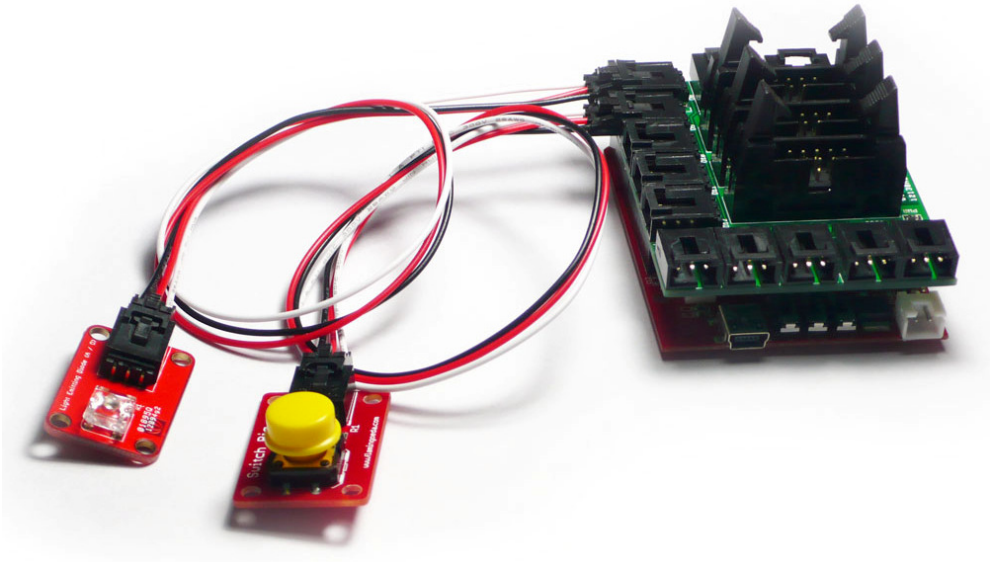
Course 0 - Analog and Digital

Analog signals are continuous while digital signals are discrete. Analog signals are continuously varying where digital signals are based on 0's and 1's (or as often said------- on's and off's). As an analogy, consider a light switch that is either on or off (digital) and a dimmer switch (analog) that allows you to adjust brightness. You may check more details about Analog and digital signals at Wikipedia: http://en.wikipedia.org/wiki/Analog_signal

[Illustration 0.0]

Let's try Digital signal first. The button or the digital sensor can be a digital input to Arduino. Arduino can read these digital signals by the digital Pin (almost all the pin is suppose digital input).And also can write the digital signal out by these digital Pins.

Hardware setup: Now we use a button to control the LED on or off. What we need is an Arduino, an Electronic Brick chassis, a button brick, LED brick and two signal cables. Hook up the button Brick to the D9 connector of chassis, then the button is connected with the 9th digital pin of Arduino. And hook up the LED Brick to the D8 connector of chassis, then the LED is connected with the 8th digital pin of Arduino.



The hardware is set, and now we can open the Arduino IDE to prepare the sketch .

Software setup:

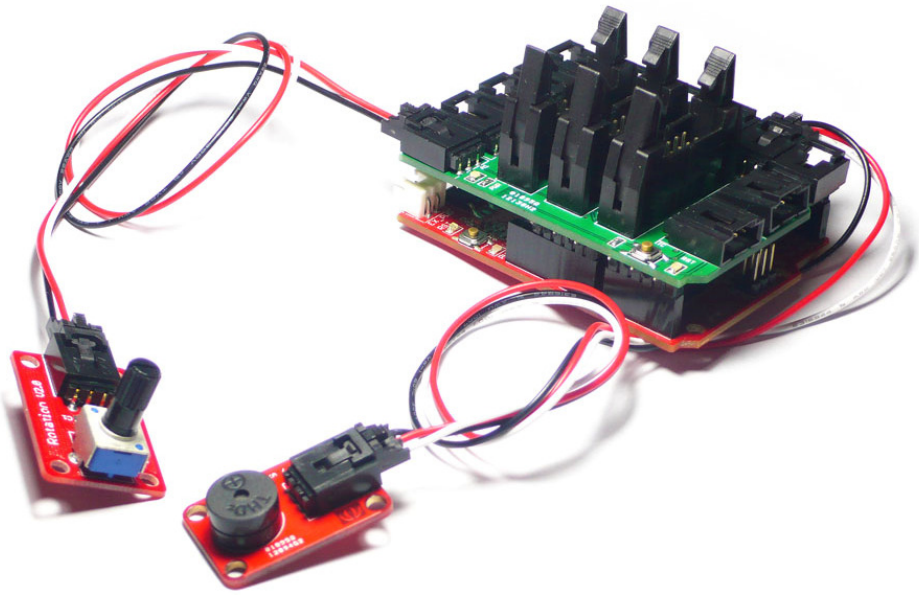
```
int Button = 9;    //define the 9th digital pin for button brick
int LED = 8;         //define the 8th digital pin for LED brick

void setup()
{
   pinMode(LED,OUTPUT); //set the LED pin for digital output
   pinMode(Button,INPUT);    //set the Button pin for digital input
}

void loop()
{
   if (digitalRead(Button))           // if button press
   digitalWrite(LED,HIGH);          // light the LED
   else                                 // if not press
   digitalWrite(LED,LOW);          // turn off the LED
}
```

Result: The LED will light whenever you press the button.

[Illustration 0.1]

The Arduino can Read the Analog signal via the AD by the analog pin, and put out the analog signal by PWM pins. In Arduino there are 6 analog input pin from A0-A1, and there are 6 analog output pins: D3 D5 D6 D9 D10 D11.

Hardware Setup: We use a buzzer brick and a Rotary brick to demonstrate the analog input and output .Connetct the buzzer to D9 connector and the rotary brick to A1 connector of chassis.



Software setup:

```
int Rotary    = 1;        // define the Rotary for 1th Analog pin
int Buzzer = 9;           // define the Buzzer for 9th Digital pin which is Analog out pin
also
int val = 0;

void setup()
{

}

void loop()
{
    val = analogRead(Rotary); // read the Rotary analog value
    analogWrite(Buzzer,val);    // Write the analog value out to Buzzer
}
```

Result: when you rotate the rotary brick, the buzzer sound will change.

Course 1 – Sensing the world

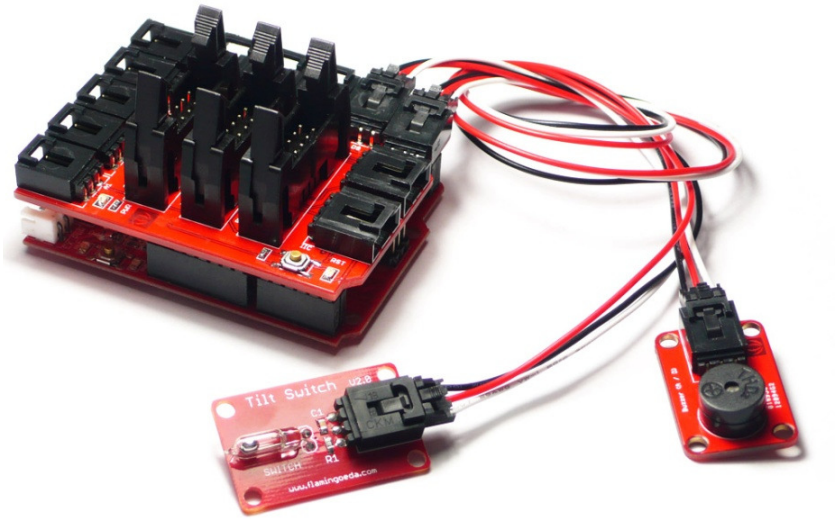
Sensors can be basically categorized as analog and digital. Some Digital sensor acts as a switch, whenever triggered it output a 'on' signal and the other environment it output a 'off' signal.

[Illustration 1.0]

We take a Mercury tilt switch as an example. When the Mercury tilt switch brick is tilted over a certain angle, it will be activated and send out a high level signal, else it sent out a low level signal.

Hardware Setup: Connect the Mercury tilt switch brick to D9 connector of chassis, and connect the LED brick to D8 connector. Then By tilting to different end, it works like a switch.

Software setup:




```
int Mercury_tilt = 9;    //define the 9th digital pin for Mercury tilt switch brick
int Buzzer = 8;          //define the 8th digital pin for LED brick
void setup()
{
   pinMode(Buzzer,OUTPUT); //set the LED pin for digital output
   pinMode(Mercury_tilt,INPUT);    //set the tilt sensor pin for digital input
}

void loop()
{
   if (digitalRead(Mercury_tilt)) // if up tilt
   digitalWrite(Buzzer,HIGH);    // ring the Buzzer
   else                                  // if not press
   digitalWrite(Buzzer,LOW);      // turn off the Buzzer
}
```

Result: when the brick up tilt, the buzzer will alarm.

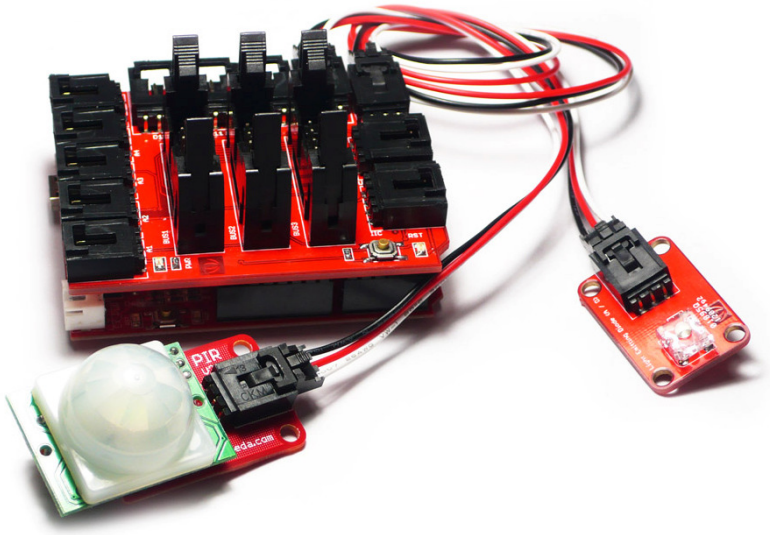
[Illustration 1.1]

Some Digital sensor are not like a switch just to put out high or low level signal in different situation. They put out pulse signal when activated.

For PIR sensor, when people move into the range, it will send a high level pulse out. The pulse duration can be change by the resistance on the brick. It's easier to check pulses which lasts longer.

Hardware Setup: Connect the PIR sensor brick to D9 connector of Chassis, and

connect the LED brick to D8.



Software setup:

```
int PIR = 9;    //define the 9th digital pin for PIR sensor brick
int LED = 8;          //define the 8th digital pin for LED brick
int time=0; // initial the time

void setup()
{
  pinMode(LED,OUTPUT); //set the LED pin for digital output
  pinMode(PIR,INPUT);    //set the tilt sensor pin for digital input
}

void loop()
{
  if (digitalRead(PIR)) // if check people move throw
  {
       time = 10000 ; // set a light time
  }
  if (time>0) // check if need to light the LED
  {
     digitalWrite(LED,HIGH);    // light the LED
     time--;                // decrease light time
  }
  else
  {
     digitalWrite(LED,LOW);    // turn off the LED
  }
}
```

Result: when somebody detected by PIR sensor, the LED will light for a moment.

More information about the PIR sensor is here:

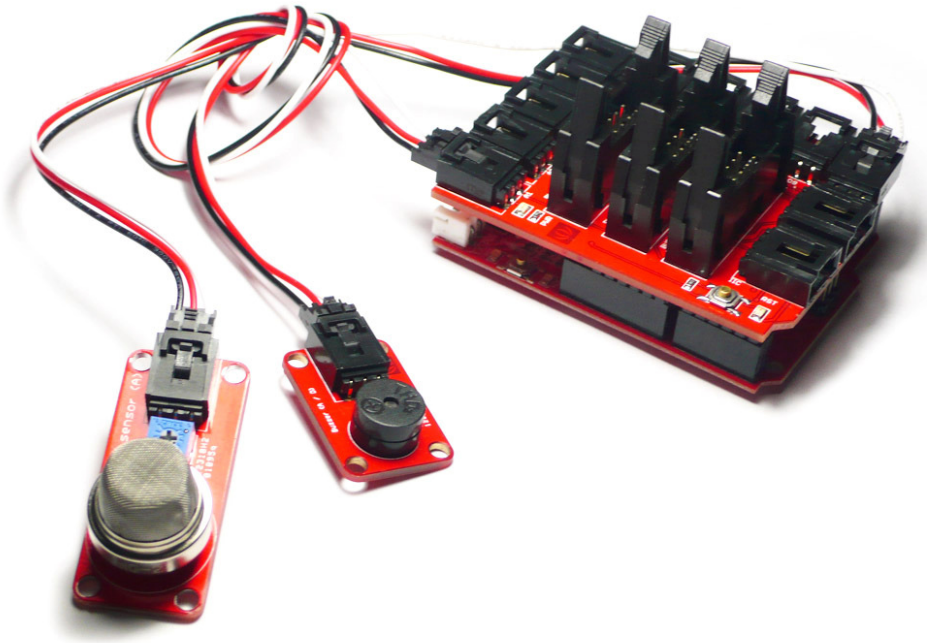
http://www.seeedstudio.com/depot/pir-motion-sensor-module-p-74.html

[Illustration 1.2]

The Analog sensor is different from Digital sensor, the level of analog signal continuously reflect the factor been measured, like light strength, gas density, temperature and so on.

Let's take a Gas sensor for example. First, adjust the Resistor of the Brick to correct the sensor output analog level. When power up the gas sensor will heat up, and then we can adjust the resistor to about 5K correct the output analog signal to about 1V.

Hardware Setup: Connect the Gas sensor brick to A1 connector of the chassis, and the buzzer to the D8 connector.



Software Setup:

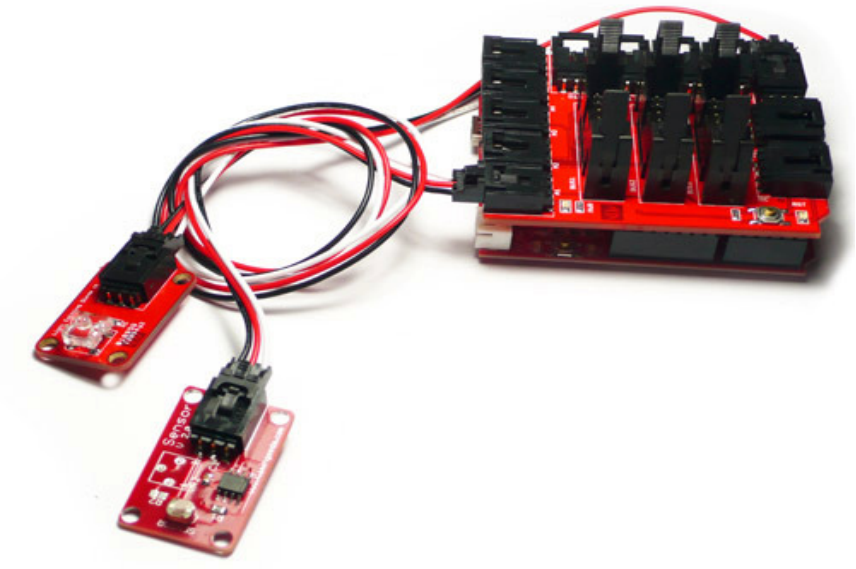
```
int Gassensor = 1;    //define the 1th digital pin for gss sensor brick
int Buzzer = 8;         //define the 8th digital pin for buzzer brick

void setup()
```

```
{
  pinMode(Buzzer,OUTPUT); //set the LED pin for digital output
}

void loop()
{
  int val=0;
  val=analogRead(Gassensor); //Read the gas sensor for gas density
  if (val>0x300) // if gas dense
  {
    digitalWrite(Buzzer,HIGH); //   ring the Buzzer for alarm
  }
  else
  {
    digitalWrite(Buzzer,LOW);   // turn off the buzzer
  }
}
```

Result: If environmental gas density exceeds the value that we setup, the buzzer will ring for alarm.

[Illustration 1.3]

The light sensor is similar to gas sensor, it also outputdifferent level according to the luminous intensity.



Hardware Setup: Connect the light sensor brick to A1 connector of chassis, and the LED brick to D8 connector as a light source.

Software Setup:

```
int LightSensor = 1;    //define the 1th digital pin for light sensor brick
int LED = 8;          //define the 8th digital pin for LED brick

void setup()
{
   pinMode(LED,OUTPUT); //set the LED pin for digital output
}

void loop()
{
   int val=0;
   val=analogRead(LightSensor); //Read analog level which match the luminous
intensity
   if (val<0x200)
   {
      digitalWrite(LED,HIGH); //    turn on the light
   }
   else
   {
      digitalWrite(LED,LOW);    // turn off the light
   }
}
```

Result: It will turn the light on in darkness and turn off if bright enough.

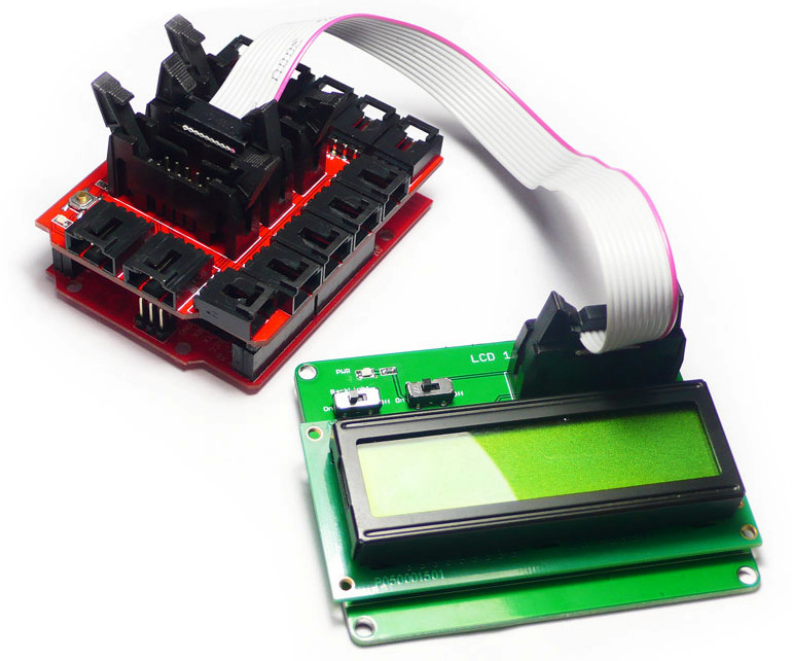
You can find more sensor brick here:

http://www.seeedstudio.com/depot/electronic-bricks-c-44.html

Course 2 – BUS

Most of the sensors and switches use a single wire for signal. There are modules need more than one signal line, so we need the bus connection. Like the 16*2 Character LCD module have 16 pins, there are 5 power pins and 11 signal pins. At least 10 pins are needed to control the 16*2 Character LCD modules, so we need a bus to connect it to Arduino.

Hardware Setup: Connect the 16*2 Character LCD Brick with the 10pins cable to the BUS2 connector of Chassis. And toggle the power switch to ‘On’.



We can use the library for Liquid Crystal to control the LCD easily. This library allows an Arduino board to control Liquid Crystal displays (LCDs) based on most popular Hitachi HD44780 (or a compatible) chipset. The library works in either 4- or 8-bit mode.

Software Setup:

LiquidCrystal() Creates a variable of type Liquid Crystal.

Syntax: LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)

LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

clear() Clears the LCD screen and positions the cursor in the upper-left corner.

home() Positions the cursor in the upper-left of the LCD. That is, use that location in outputting subsequent text to the display. To also clear the display, use the clear() function instead.

setCursor() Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.

Syntax: lcd.setCursor(col, row)

write() Write a character to the LCD.

Syntax: lcd.write(data)

print() Prints text to the LCD.

Syntax: lcd.print(data)

lcd.print(data, BASE) (BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).)

```
#include <LiquidCrystal.h> // include a library headfile
// LiquidCrystal display with:
// rs on pin 10
// rw on pin 11
// enable on pin 12
// d4, d5, d6, d7 on pins 13, 14, 15, 16
LiquidCrystal lcd(10, 11, 12, 13, 14, 15, 16);

void setup()
{
  lcd.clear();//clears the LCD and positions the cursor in the upper-left corner

  lcd.print("hello, world!");// Print a message to the LCD.

  lcd.setCursor(2,1); // set to the 3th column and 2nd row

  lcd.print("Seeedstudio");// Print a message to the LCD.
}

void loop()
{
}
```

More information about the LCE1602 and Library is here:

http://www.seeedstudio.com/depot/lcd-162-characters-green-yellow-back-light-p-62.html

http://arduino.cc/en/Reference/LiquidCrystal

Course 3 – IIC Interface:

If you want Arduino to communication with external module, there are some communication protocols optional: IIC, SPI and UART.

See IIC / TWI for an example:

NXP Semiconductors developed a simple bi-directional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. (Description from NXP I2C-bus specification and user manual v3).

It's not a must to know the specific protocol and time sequence, because there is a ready library allows you to communicate with I2C / TWI devices. On most Arduino boards, SDA (data line) is on analog input pin 4, and SCL (clock line) is on analog input pin 5. On the Arduino Mega, SDA is digital pin 20 and SCL is 21. Let's take a quick look at the functions of the library:

begin() / begin(address) Initiate the Wire library and join the I2C bus as a master or slave.

address :the 7-bit slave address (optional); if not specified, join the bus as a master. (note: There are both 7- and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8 bit address, you'll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127.)

Syntax: wire.begin(0x04); // set a Arduino as a slave which address is 0x04.

wire.begin(); //set a Arduino as a master.

requestFrom(address, count) Request bytes from another device. The bytes may then be retrieved with the available() and receive() functions. quantity: the number of bytes to request.

Syntax : wire.requestFrom(0x04,4); // A master request the slave which address is 0x04 to return 4 byte.

beginTransmission(address) Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the send() function and transmit them by calling endTransmission().

Syntax: wire.beginTransmissin(0x04);// The master start a transmission with the slave which address is 0x04

endTransmission() Ends a transmission to a slave device that was begun by beginTransmission() and actually transmits the bytes that were queued by send().

Syntax: wire.endTransmissin();// The master ends a transmission with the slave which connected.

send() Sends data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to beginTransmission() and endTransmission()).

Parameters: value: a byte to send (byte)

string: a string to send (char *)

data: an array of data to send (byte *)

quantity: the number of bytes of data to transmit (byte)

Syntax : wire.send("hallow word");//maser send up the data "hallow" to IIC bus.

byte available() Returns the number of bytes available for retrieval with receive(). This should be called on a master device after a call to requestFrom() or on a slave inside the onReceive() handler. return the number of bytes available for reading.

Syntax : if(wire.available()) // check if there is data receive.

byte receive() Retrieve a byte that was transmitted from a slave device to a master after a call to requestFrom or was transmitted from a master to a slave.

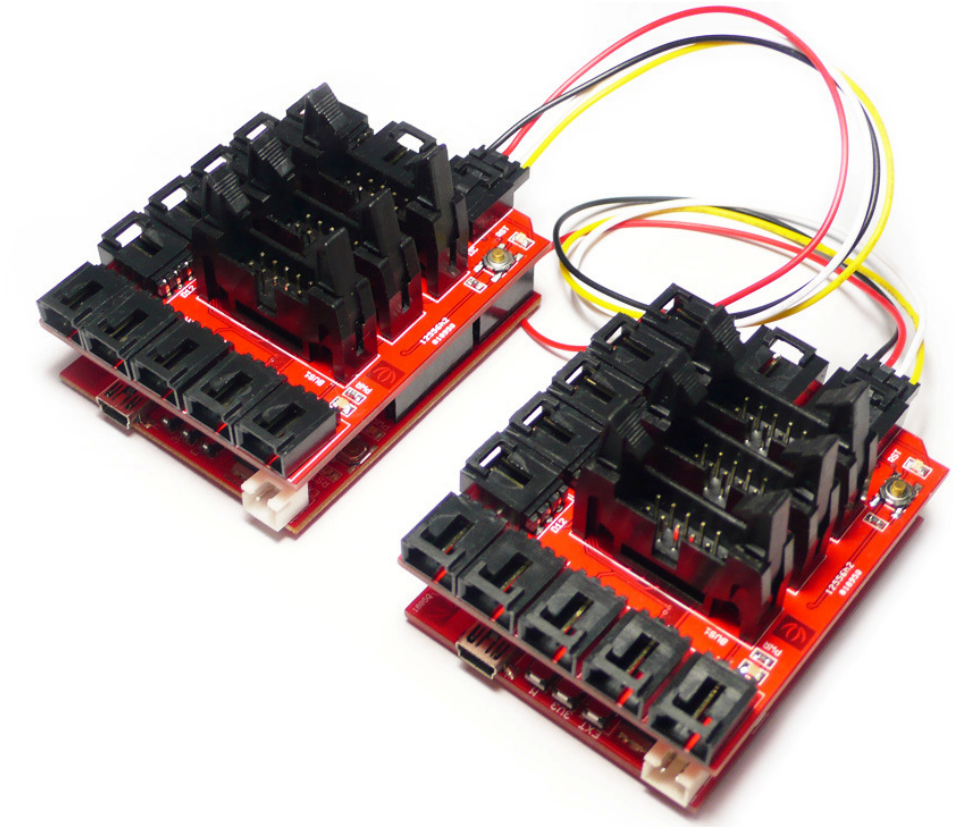
Syntax : data = receive(); // Retrieve the data that on the wire.

onReceive(handler) Registers a function to be called when a slave device receives a transmission from a master. (handler: the function to be called when the slave receives data; this should take a single int parameter (the number of bytes received from the master) and return nothing, e.g.: void)

Syntax: wire.onReceive(read);// when the slave receives a data from the master it will jump into the function read()

onRequest(handler) Register a function to be called when a master requests data from this slave device. (handler: the function to be called, takes no parameters and returns nothing, e.g.: void myHandler())

Syntax: wire.onRequest(read); // when the master receives a data from the slave it will jump into the function read()



Let's try connecting two Arduino together and communicate.

Connect two Arduino with 4 pins cable use the IIC connector on the Chassis. Then the hardware is set.

```
#include <Wire.h>

void setup()
{
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(4); // transmit to device #4
  Wire.send("x is ");              // sends five bytes
  Wire.send(x);                    // sends one byte
  Wire.endTransmission();       // stop transmitting

  x++;
  delay(500);
```

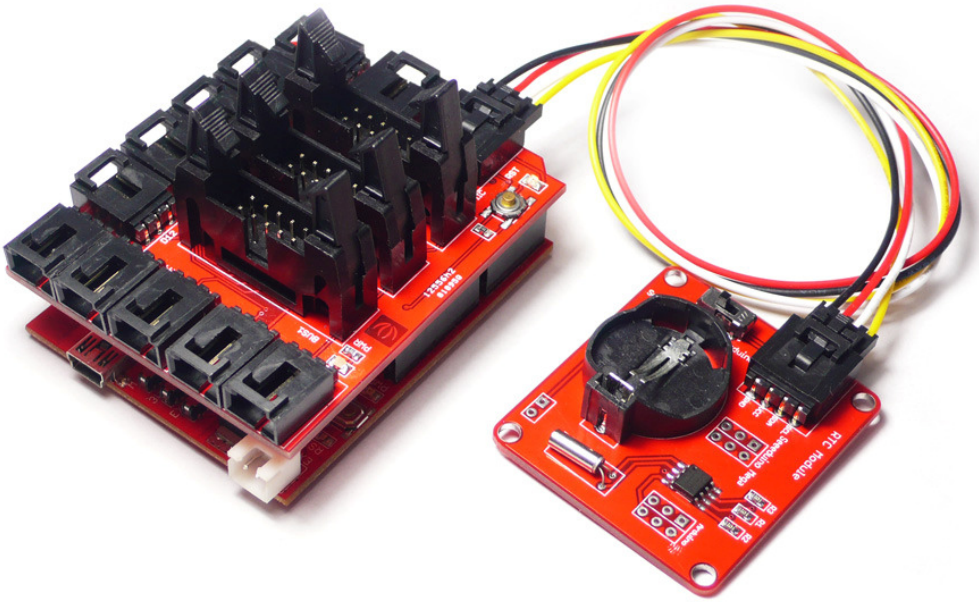
```
}
```

Program this code into the first Arduino , then this one is the master. And we use it to sent out data.

```
#include <Wire.h>
void setup()
{
  Wire.begin(4);                          // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);                     // start serial for output
}

void loop()
{
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  while(1 < Wire.available()) // loop through all but the last
  {
    char c = Wire.receive(); // receive byte as a character
    Serial.print(c);                // print the character
  }
  int x = Wire.receive();       // receive byte as an integer
  Serial.println(x);              // print the integer
}
```

Program this code into the other Arduino to turn it a receiver, then it become the slave which address is 4. And press “serial monitor” button on the Arduino IDE, you can see the data that slave received.

There are many module have the IIC interface, you can use the Arduino to control them easily too. Let’s take the DS1307 RTC module for example: DS1307 is a real time clock IC, it can be set and sent out the clock data by IIC.



Connect the RTC module to the IIC connector on chassis and program the code below, then you will get a clock.

```
#include <WProgram.h>
#include <Wire.h>
#include <DS1307.h>
// written by mattt on the Arduino forum and modified by D. Sjunnesson

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

  RTC.stop();
  RTC.set(DS1307_SEC,1);          //set the seconds
  RTC.set(DS1307_MIN,23);         //set the minutes
  RTC.set(DS1307_HR,12);          //set the hours
  RTC.set(DS1307_DOW,4);          //set the day of the week
  RTC.set(DS1307_DATE,5);         //set the date
  RTC.set(DS1307_MTH,3);          //set the month
  RTC.set(DS1307_YR,9);           //set the year
  RTC.start();
```

```
}

void loop()
{

  Serial.print(RTC.get(DS1307_HR,true)); //read the hour and also update all the values by pushing in true
  Serial.print(":");
  Serial.print(RTC.get(DS1307_MIN,false));//read minutes without update (false)
  Serial.print(":");
  Serial.print(RTC.get(DS1307_SEC,false));//read seconds
  Serial.print("          "); // some space for a more happy life
  Serial.print(RTC.get(DS1307_DATE,false));//read date
  Serial.print("/");
  Serial.print(RTC.get(DS1307_MTH,false));//read month
  Serial.print("/");
  Serial.print(RTC.get(DS1307_YR,false)); //read year
  Serial.println();

  delay(1000);

}
```

Press “serial monitor” button on the Arduino IDE, you can see the data returned by RTC module.

More information about IIC/TWI library and RTC module is here:

http://www.arduino.cc/playground/Learning/I2C

http://www.arduino.cc/en/Reference/Wire

http://www.seeedstudio.com/depot/electronic-brick-real-time-clock-moduleds1307-p-491.html

Course 4 – Interrupt

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. A good task for using an interrupt might be reading a rotary encoder, monitoring user input.

For an example, if you want to check the PIR sensor when displaying LCD. It may be

lost the PIR pulse signal when update the screen. The best way it's use the Interrupt to catch the small pulse. In the situations like this , using an interrupt can free the microcontroller to get some other work done while not missing the input signal.

Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). The Arduino Mega has an additional four: numbers 2 (pin 21), 3 (pin 20), 4 (pin 19), and 5 (pin 18).

And you can use the interrupt function to control the interrupt:

attachInterrupt(interrupt, function, mode): Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt.

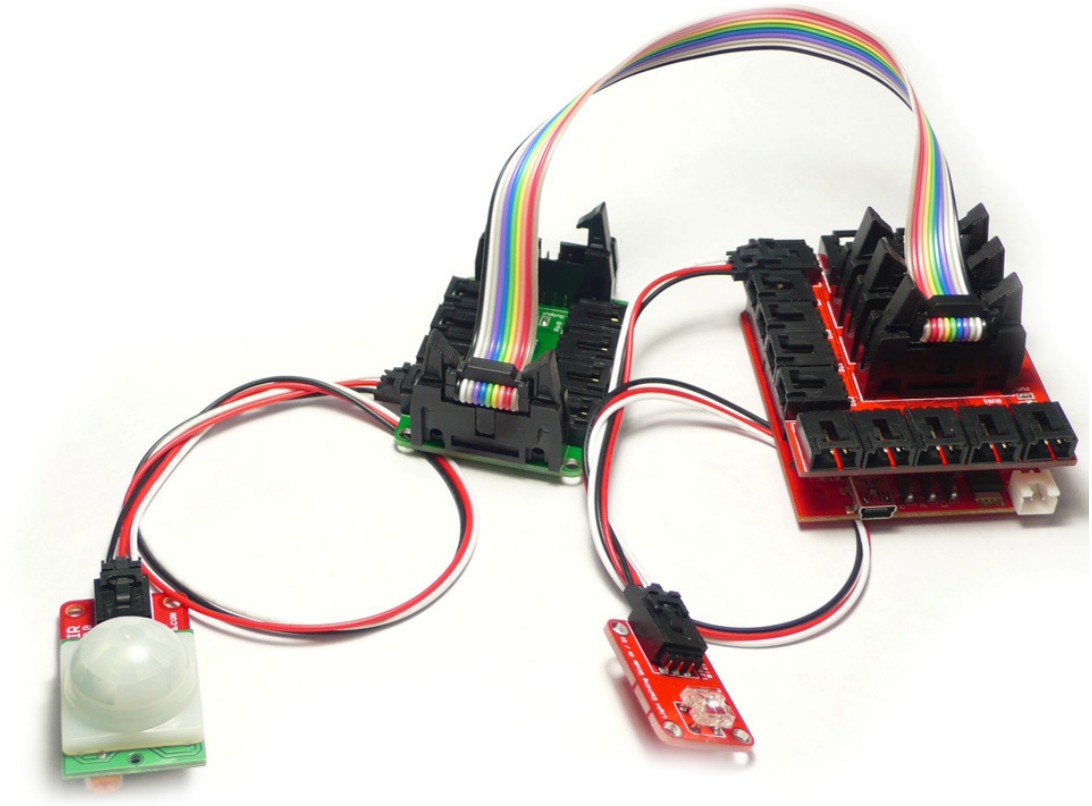
Parameters "Interrupt" is used to choose the external interrupt channel. "Function" is the function to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine. "Mode" defines when the interrupt should be triggered. Four contestants are predefined as valid values:

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.



In chassis the D2 and D3 are not prepared individually, but they are include in the Bus 1. So we need a BUS HUB module to break them out of BUS1. Connect the “Input” connector on the bus hub to “BUS1” connector on chassis by the 10 pins cable.

Hardware Setup: Connector the PIR sensor to the “Pin2” connector on the bus hub, and connector the LED to the D8 connector on the chassis.

Modify the code for interrupt:

```
int PIR = 0;            //define the 2th digital pin for PIR sensor brick interrupt
int LED = 8;          //define the 8th digital pin for LED brick
int time=0;          // initial the time

void setup()
{
   pinMode(LED,OUTPUT); //set the LED pin for digital output
attachInterrupt(PIR, blink, RISING); // enable the interrupt , and when D2 rising jump into the
blink() function.

}

void loop()
{
```

```
  if (time>0) // check if need to light the LED
  {
     digitalWrite(LED,HIGH);    // light the LED
     time--;                    // decrease light time
  }
  else
  {
     digitalWrite(LED,LOW);    // turn off the LED
  }
}

void blink()
{
 time=1000; // catch the PIR sensor pulse
}
```

Program the code into the Arduino. When somebody detected by PIR sensor, the LED will light for a moment.

Revision History

21st Aug – V1.0a revision published