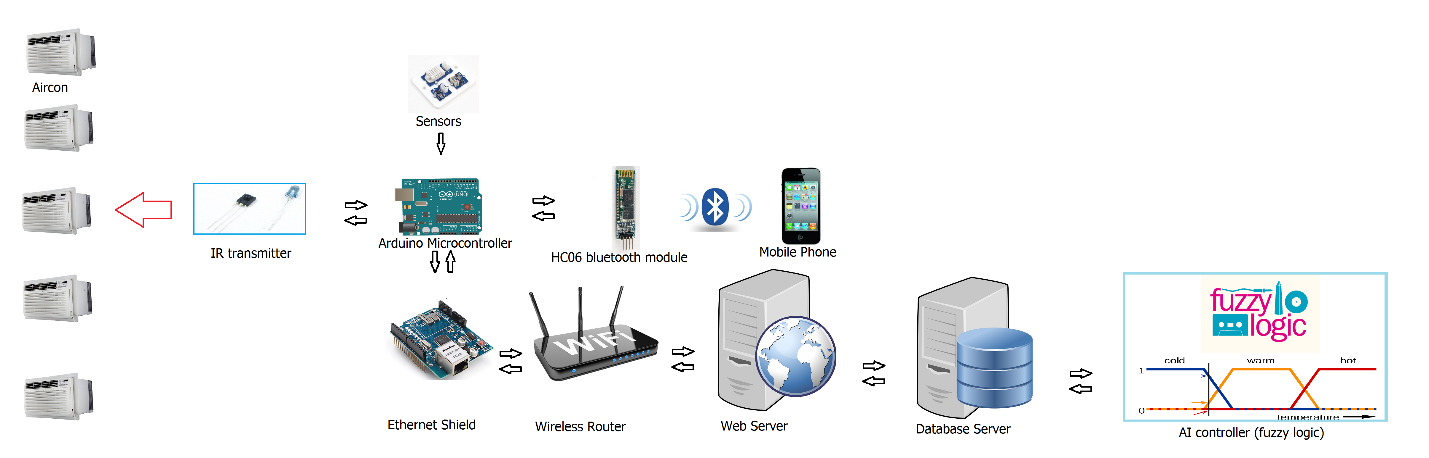
Chapter 3

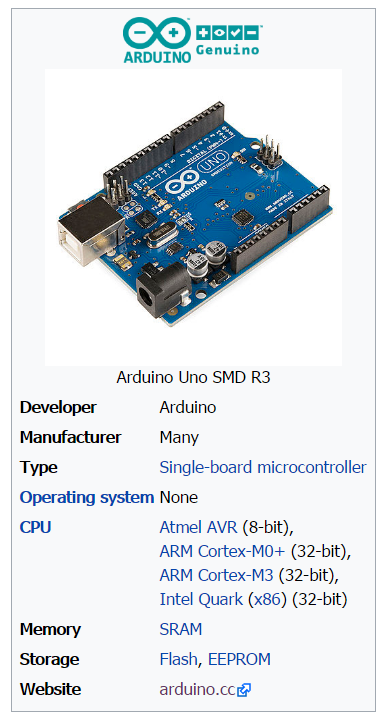
Design and Implementation of Building Ventilation System

Conceptual Design



Hardware Design

Arduino



Arduino is an open source, computer hardware and software company, project, and user community that designs and manufactures microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical world. The project's products are distributed as open-source hardware and software, which are licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the manufacture of Arduino boards and software distribution by anyone. Arduino boards are available commercially in preassembled form, or as do-it-yourself kits.

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project.

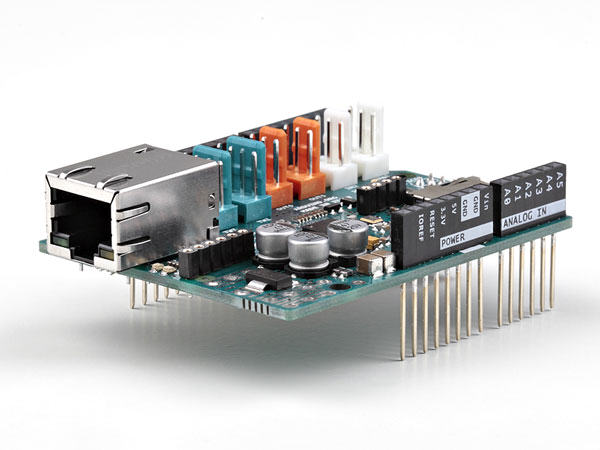
The Arduino project started in 2003 as a program for students at the Interaction Design Institute Ivrea in Ivrea, Italy, aiming to provide a low-cost and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots, thermostats, and motion detectors.

The name Arduino comes from a bar in Ivrea, Italy, where some of the founders of the project used to meet. The bar was named after Arduin of Ivrea, who was the margrave of the March of Ivrea and King of Italy from 1002 to 1014.

Arduino microcontrollers are pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory. The default bootloader of the Aduino UNO is the optiboot bootloader. Boards are loaded with program code via a serial connection to another computer. Current Arduino boards are programmed via Universal Serial Bus (USB), implemented using USB-to-serial adapter chips such as the FTDI FT232. Some boards, such as later-model Uno boards, substitute the FTDI chip with a separate AVR chip containing USB-to-serial firmware, which is reprogrammable via its own ICSP header.

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The current Uno provide 14 digital I/O pins, six of which can produce pulse-width modulated signals, and six analog inputs, which can also be used as six digital I/O pins. These pins are on the top of the board, via female 0.1-inch (2.54 mm) headers. Several plug-in application shields are also commercially available. The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards may provide male header pins on the underside of the board that can plug into solderless breadboards.

Ethernet Shield



The Arduino Ethernet shield 2 allows an Arduino board to connect to the internet using the Ethernet library and to read and write an SD card using the SD library. This shield is fully compatible with the former version, but relies on the newer W5500 chip. Depending on the shield version you have, you need to use the proper library, as documented in the Ethernet library page.

To use the shield, mount it on top of an Arduino board (e.g. the Uno). To upload sketches to the board, connect it to your computer with a USB cable as you normally would. Once the sketch has been uploaded, you can disconnect the board from your computer and power it with an external power supply.

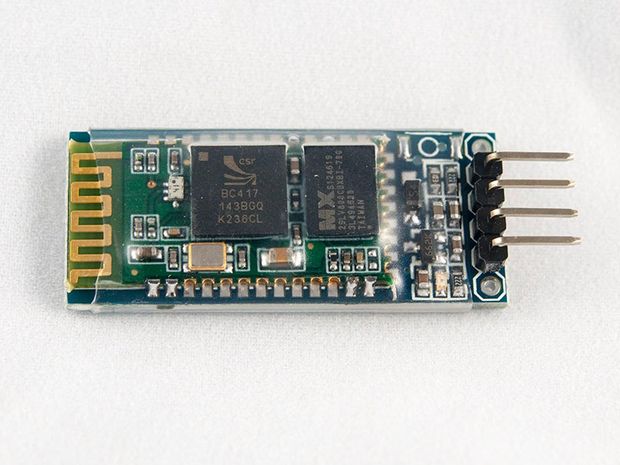
Connect the shield to your computer or a network hub or router using a standard ethernet cable (CAT5 or CAT6 with RJ45 connectors). Connecting to a computer may require the use of a cross-over cable (although many computers, including [all recent Macs](http://support.apple.com/kb/HT2274) can do the cross-over internally).

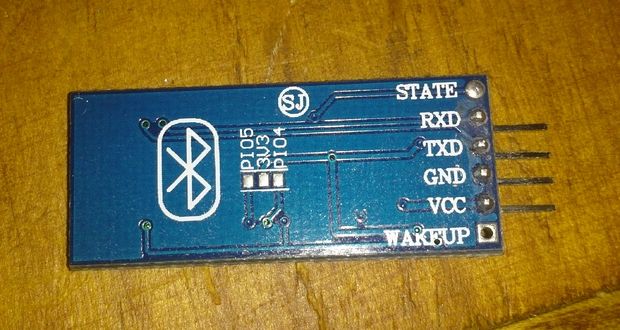
Ethernet Shield can be used for many purposes, for e.g.

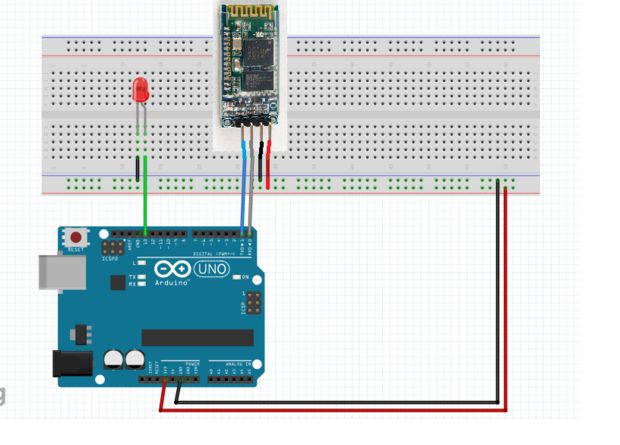
* ChatServer: set up a simple chat server.
* WebClient: make a HTTP request.
* WebClientRepeating: Make repeated HTTP requests.
* WebServer: host a simple HTML page that displays analog sensor values.
* BarometricPressureWebServer: outputs the values from a barometric pressure sensor as a web page.
* UDPSendReceiveString: Send and receive text strings via UDP.
* UdpNtpClient: Query a Network Time Protocol (NTP) server using UDP.
* DnsWebClient: DNS and DHCP-based Web client.
* DhcpChatServer: A simple DHCP Chat Server
* DhcpAddressPrinter: Get an IP address via DHCP and print it out
* TelnetClient: A simple Telnet client

HC-06 Bluetooth

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). Range is approximately 10 Meters (30 feet).







These modules are based on the Cambridge Silicon Radio BC417 2.4 GHz BlueTooth Radio chip. This is a complex chip which uses an external 8 Mbit flash memory. If you like LOTS of details see the: [Data Sheet].

These low-cost Bluetooth Sub-modules work well with Arduino and other Microcomputers.

HC-05 is a more capable module that can be set to be either Master or Slave. HC-06 is a Slave only device. These small ( 3 cm long) modules run on 3.3V power with 3.3V signal levels, They have no pins and usually solder to a larger board. The module has two modes of operation, Command Mode where we can send AT commands to it and Data Mode where it transmits and receives data to another bluetooth module.

"Breakout" Boards that make these easy to use are available and recommended. These mount the sub-module like that shown on the right on a slightly larger board. NOTE: Sellers often label them "HC-05" or "HC-06", but they have some other model number on the reverse side. Most of these boards support operation at 5V power and interface to 5V Arduino signal levels with some technique of level shifting.

Setting up the HC-06 is so easy . All we need to know is the pin configuration. The HC-06 has 6 pins: wakeup, VCC, GND, TXD, RXD and State. Right now we will only deal with 4 pins, which are VCC, GND, TXD and RXD.

Here is how we should connect the Bluetooth module to wer Arduino.

HC-06>>>Arduino

VCC>>>>3.3v

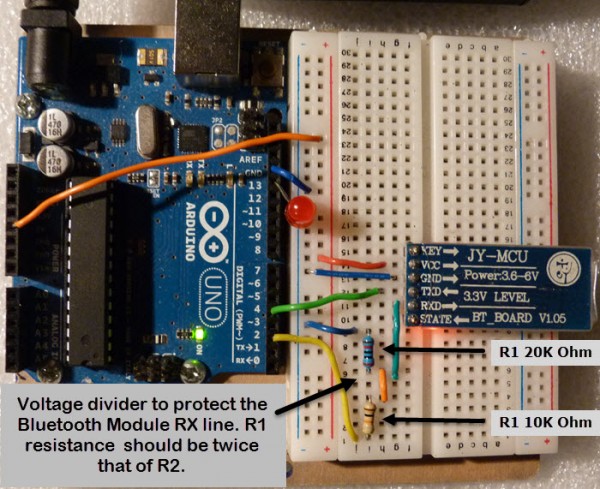
GND>>>>GND

TXD>>>>RXD

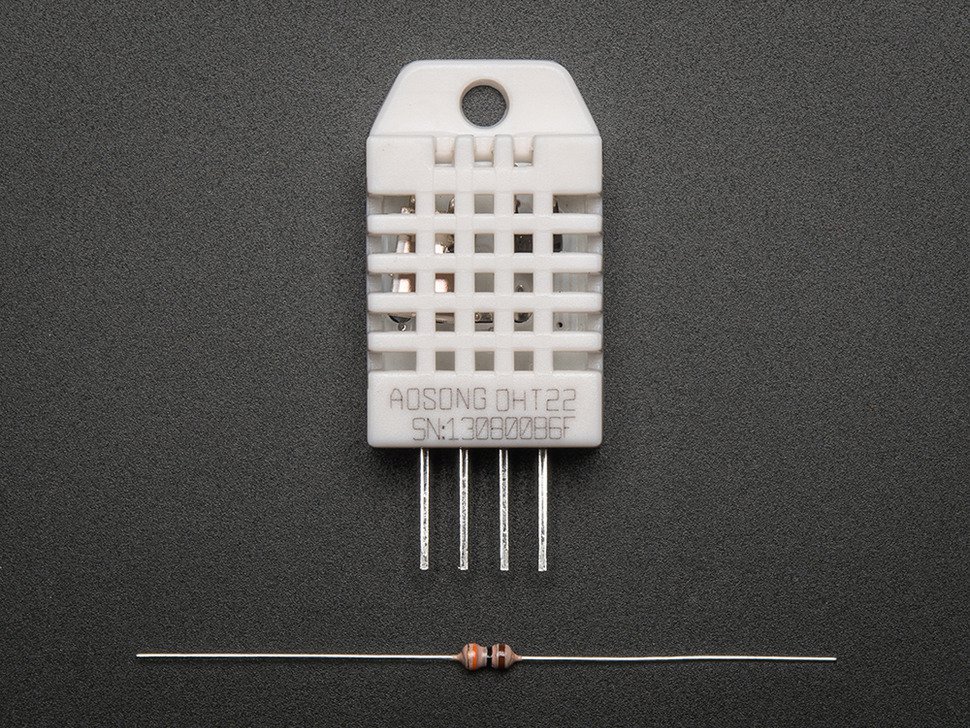
RXD>>>>TXD

The HC-06 acts as a serial port through which we can send and receive data. So using a serial terminal or a Bluetooth customized application on wer computer or phone, we can control and monitor wer project. We used Putty as the serial terminal.

Before, uploading the code to the Arduino, disconnect the HC-06 module, since it shares the tx/rx pins and will interfere with the upload. Connect it back once the code has been uploaded successfully.



DHT22



The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.  
  
Simply connect the first pin on the left to 3-5V power, the second pin to your data input pin and the right most pin to ground. Although it uses a single-wire to send data it is not Dallas One Wire compatible! If you want multiple sensors, each one must have its own data pin!

Compared to the DHT11, this sensor is more precise, more accurate and works in a bigger range of temperature/humidity, but its larger and more expensive

Comes with a 4.7K - 10K resistor, which you will want to use as a pullup from the data pin to VCC.

TECHNICAL DETAILS

Low cost

3 to 5V power and I/O

2.5mA max current use during conversion (while requesting data)

Good for 0-100% humidity readings with 2-5% accuracy

Good for -40 to 80°C temperature readings ±0.5°C accuracy

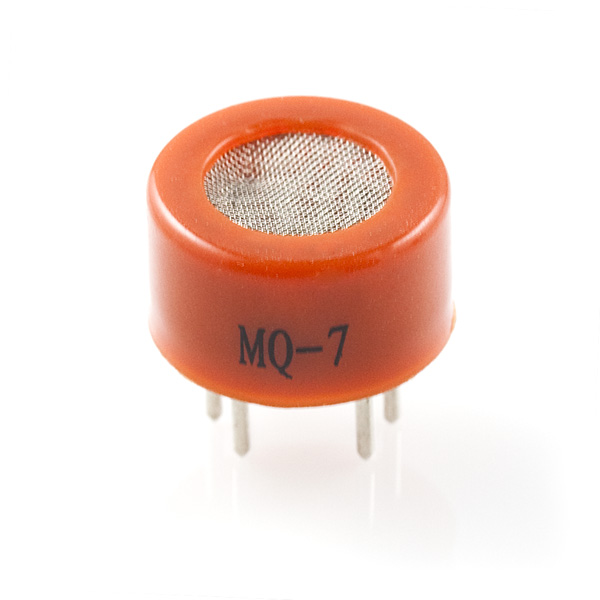
No more than 0.5 Hz sampling rate (once every 2 seconds)

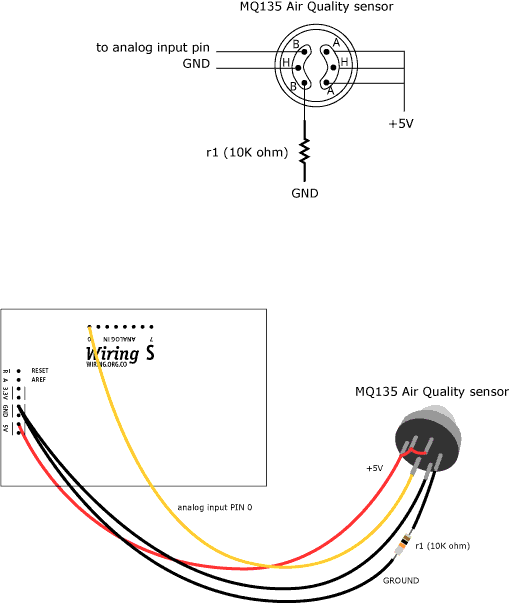
Body size 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")

4 pins, 0.1" spacing

Weight (just the DHT22): 2.4g

MQ7



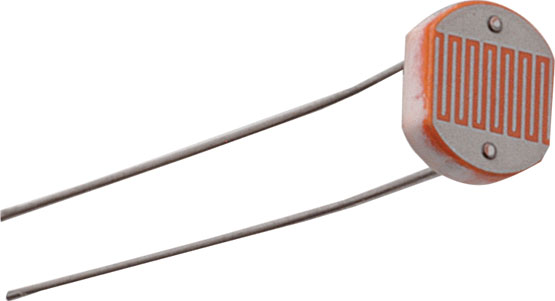


Description: This is a simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air. The MQ-7 can detect CO-gas concentrations anywhere from 20 to 2000ppm.

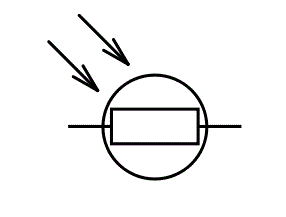
This sensor has a high sensitivity and fast response time. The sensor’s output is an analog resistance. The drive circuit is very simple; all you need to do is power the heater coil with 5V, add a load resistance, and connect the output to an ADC.

This sensor comes in a package similar to our MQ-3 alcohol sensor, and can be used with the breakout board below.

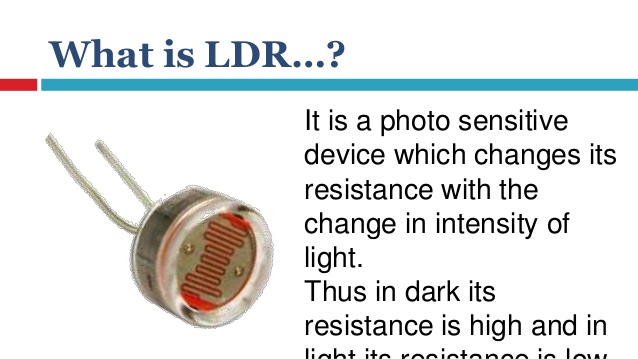
LDR

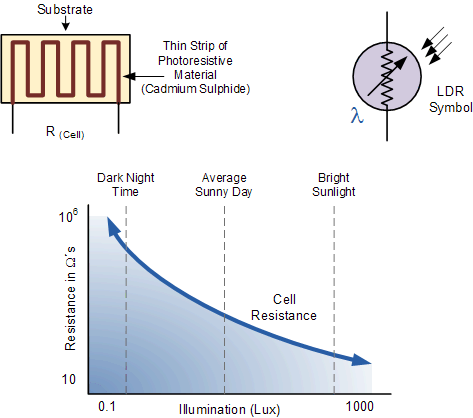


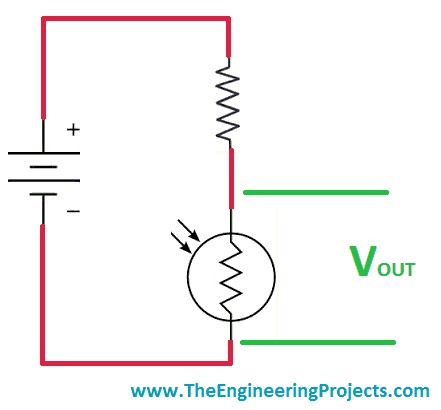
A Light Dependent Resistor (LDR) or a photo resistor is a device whose resistivity is a function of the incident electromagnetic radiation. Hence, they are light sensitive devices. They are also called as photo conductors, photo conductive cells or simply photocells. They are made up of semiconductor materials having high resistance. There are many different symbols used to indicate a LDR, one of the most commonly used symbol is shown in the figure below. The arrow indicates light falling on it.



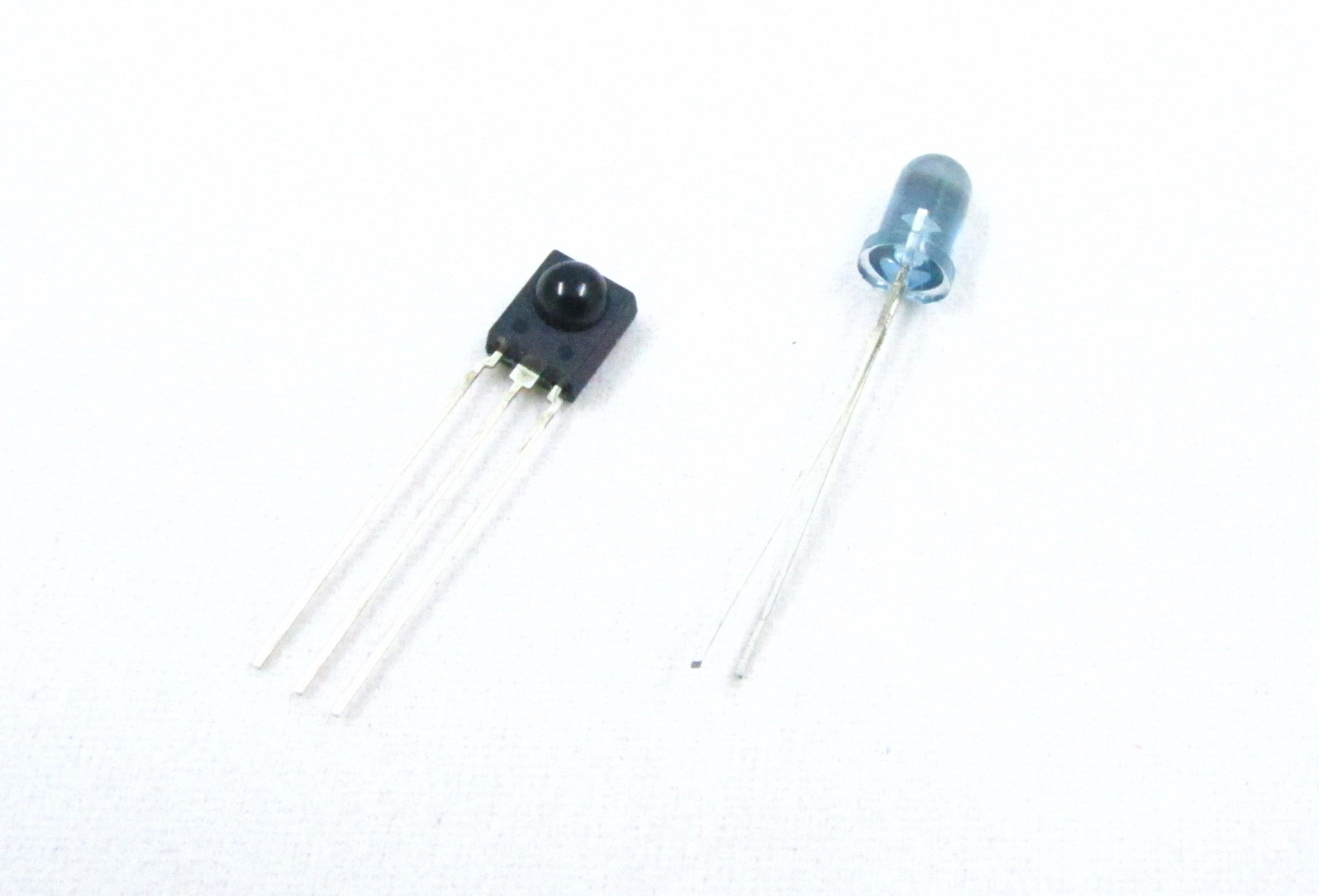








IR transmitter and Receiver



IR sensor tuned to 38KHz, perfect for receiving commands from a TV remote control.

Runs at 5V (although it seems to be OK down to 3.3V)To use, connect pin 3 (all the way to the right) to 5V power, pin 2 (middle) to ground and listen on pin 1. It doesn’t do any decoding of the signal, just passes the ‘raw data’ along.

* Size: square, 7mm by 8mm detector area
* Price: $2.00 at the Electrodragon shop
* Output: 0V (low) on detection of 38KHz carrier, 5V (high) otherwise
* Sensitivity range: 800nm to 1100nm with peak response at 940nm. Frequency range is 35KHz to 41KHz with peak detection at 38KHz
* Power supply: 5V DC 3mA
* [HS0038](http://s1.electrodragon.com/wp-content/uploads/2011/12/GP1UX31QS.pdf)Datasheet

**How to hack into the TV remote control and understand the IR code**

Posted on [December 11, 2011](http://www.electrodragon.com/how-to-hack-into-the-tv-remote-control-and-understand-the-ir-code/) by [admin](http://www.electrodragon.com/author/admin/) in [Network](http://www.electrodragon.com/category/bluetooth/) with [1 Comment](http://www.electrodragon.com/how-to-hack-into-the-tv-remote-control-and-understand-the-ir-code/#comments)

What you will need all is the IR receiver and sender, you can find it [here](http://electrodragon.com/?product=kit-universal-ir-receiver-gp1ux311qs-and-ir-led-transmitter).

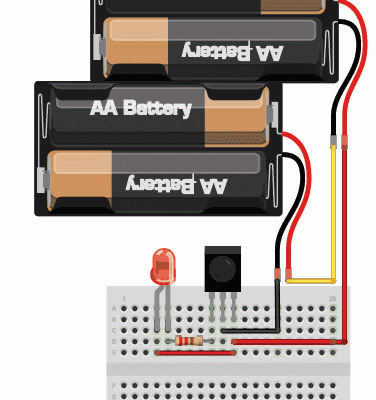
The Receiver should connect like this,

* Pin 1 is the output so we wire this to a visible LED and resistor
* Pin 2 is ground
* Pin 3 is VCC, connect to 5V

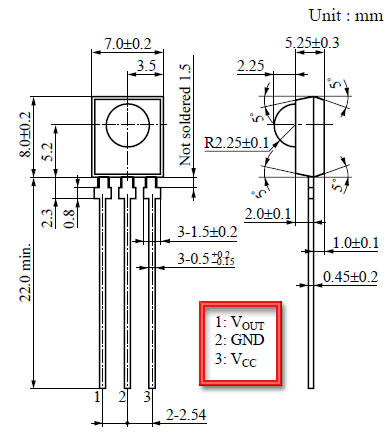
And now you can start to test it with a normal LED and 4 AA battery.

**IR remote signals**

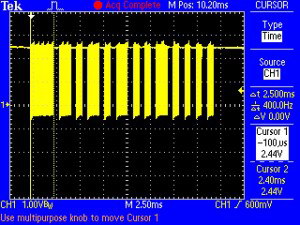
Now we know that the sensor works, we want to figure out whats being sent right? But before we do that let’s first examine exactly how data is being sent from the IR remote (in your hand) to the IR receiving sensor (on the breadboard).



For this example we will use the Sony power on/off IR code from a Sony TV remote. Its very simple and commonly documented!



Lets pretend we have a Sony remote, and we can look at exactly what light is being blasted out of the IR LED. We’ll hookup a basic light sensor (like a basic photocell!) and listen in. We won’t use a decoder like a PNA4602 (just yet) because we want to see the undecoded signal. What we see is the following:



Basically we see pulses or IR signal. the yellow ‘blocks’ are when the IR LED is transmitting and when there is only a line, the IR LED is off. (Note that the voltage being at 3VDC is just because of the way I hooked up the sensor, if I had swapped the pullup for a pulldown it would be at ground.)

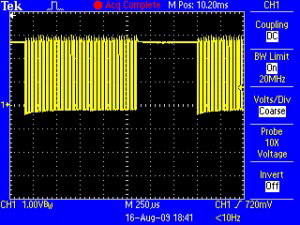
The first ‘block’ is about 2.5ms long

(see the cursors and the measurement on the side)

If you zoom into one of those blocks…

You see that they’re not really ‘blocks’ but actually very fast pulses!

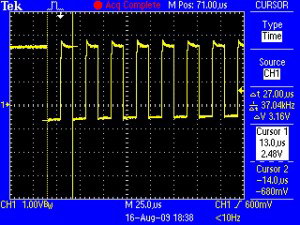
If you zoom in all the way…



You can measure the frequency of the IR pulses. As you can tell by the cursors and the measurements on the side, the frequency is about 37.04KHz

OK so now we can understand how IR codes are sent. The IR transmitter LED is quickly pulsed (PWM – pulse width modulated) at a high frequency of 38KHz and then that PWM is likewise pulsed on and off much slower, at times that are about 1-3 ms long.

Why not have the LED just on and off? Why have PWM ‘carrier’ pulsing? Many reasons!



One reason is that this lets the LED cool off. IR LEDs can take up to 1 Amp (1000 milliamps!) of current. Most LEDs only take 20mA or so. This means IR LEDs are designed for high-power blasting BUT they can only take it for a few microseconds. By PWM’ing it, you let the LED cool off half the time

Another reason is that the TV will only listen to certain frequencies of PWM. So a Sony remote at 37KHz wont be able to work with a JVC DVD player that only wants say 50KHz.

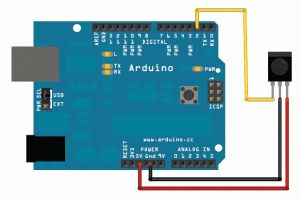
Finally, the most important reason is that by pulsing a carrier wave, you reduce the affects of ambient lighting. The TV only looks for changes in light levels that clock in around 37KHz. Just like its easier for us to tell differences between audio tones than to pin down the precsise pitch of a tone (well, for most people at least)

OK so now we know the carrier frequency. Its 37KHz. Next lets find the pulse widths!

**Reading out IR codes from an Arduino**

The good news is that it is very easy to hook up this sensor. Just connect the output to a digital pin. The bad news is that the Arduino’s friendly digitalRead() procedure is a tad too slow to reliably read the fast signal as its coming in. Thus we use the hardware pin reading function directly from pin D2, thats what the line IRpin\_PIN & (1 « IRpin)does.

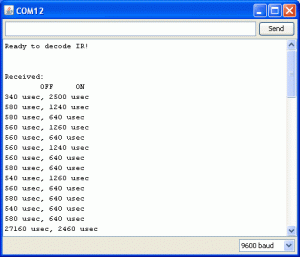
And now put the code into arduino, you can download it from here.

[](http://s3.electrodragon.com/wp-content/uploads/2011/12/arduinopna46021.gif)

<https://github.com/adafruit/Raw-IR-decoder-for-Arduino>

// you can also use codes from <https://github.com/shirriff/Arduino-IRremote> , the IRrecvdump could detect the IR code as well

Point your remote to the IR receiver, and just click the button once, and you will see the following display:



You have to ingore the first one as useless, and start with the second one, for example, my IR can modify like this with a pair:

19828 usec, // this is the useless one that you can ignore

7860 usec // and this is the IR pulse which goes higher and lower for 7860 microsecond certain time  
3940 usec, // and this is the delay, the IR pulse will start again

500 usec // IR pluse  
480 usec, // delay

500 usec  
480 usec,

500 usec  
1460 usec,

540 usec  
460 usec,

480 usec  
480 usec,

500 usec  
480 usec,

520 usec  
1440 usec,

520 usec  
1460 usec,

480 usec  
3940 usec,

520 usec  
1440 usec,

520 usec  
480 usec,

500 usec  
1460 usec,

500 usec  
480 usec,

500 usec  
480 usec,

500 usec  
480 usec,

500 usec  
480 usec,

500 usec  
480 usec,

480 usec // last one IR pulse

**Send signal to your TV**

now let’s wire the anode side of IR LED to digital pin 13, and cathode side of LED to GND.

First let’s take a look at following code:

[?](http://www.electrodragon.com/how-to-hack-into-the-tv-remote-control-and-understand-the-ir-code/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | // This sketch will send out a Nikon D50 trigger signal (probably works with most Nikons)  // See the full tutorial at <http://www.ladyada.net/learn/sensors/ir.html>  // this code is public domain, please enjoy!    int IRledPin =  13;    // LED connected to digital pin 13    // The setup() method runs once, when the sketch starts    void setup()   {    // initialize the IR digital pin as an output:    pinMode(IRledPin, OUTPUT);      Serial.begin(9600);  }    void loop()  {    Serial.println("Sending IR signal");      SendNikonCode();      delay(60\*1000);  // wait one minute (60 seconds \* 1000 milliseconds)  }    // This procedure sends a 38KHz pulse to the IRledPin  // for a certain # of microseconds. We'll use this whenever we need to send codes  void pulseIR(long microsecs) {    // we'll count down from the number of microseconds we are told to wait      cli();  // this turns off any background interrupts      while (microsecs > 0) {      // 38 kHz is about 13 microseconds high and 13 microseconds low     digitalWrite(IRledPin, HIGH);  // this takes about 3 microseconds to happen     delayMicroseconds(10);         // hang out for 10 microseconds     digitalWrite(IRledPin, LOW);   // this also takes about 3 microseconds     delayMicroseconds(10);         // hang out for 10 microseconds       // so 26 microseconds altogether     microsecs -= 26;    }      sei();  // this turns them back on  }    void SendNikonCode() {    // This is the code for my particular Nikon, for others use the tutorial    // to 'grab' the proper code from the remote      pulseIR(2080);    delayMicroseconds(27);    pulseIR(440);    delayMicroseconds(1500);    pulseIR(460);    delayMicroseconds(3440);    pulseIR(480);      delay(65); // wait 65 milliseconds before sending it again      pulseIR(2000);    delayMicroseconds(27);    pulseIR(440);    delayMicroseconds(1500);    pulseIR(460);    delayMicroseconds(3440);    pulseIR(480);  } |

In the above codes, void pulseIR go higher and lower for certain time, for example 7860 microseconds. now we can put all the timing we get from serial montior into pulseIR() and delaymicroseconds(); for example, it should be like:

[?](http://www.electrodragon.com/how-to-hack-into-the-tv-remote-control-and-understand-the-ir-code/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | pulseIR(7860);    delayMicroseconds(3940);      pulseIR(500);    delayMicroseconds(480);      pulseIR(500);    delayMicroseconds(3440);  ....    pulseIR(500);    delayMicroseconds(480);      pulseIR(500);    delayMicroseconds(480);      pulseIR(580); |

Do all the timing in the codes and put them into arduino finally, now you can control your TV by arduino!

Android application

Web Services

1. https://www.w3schools.com/tags/ref\_httpmethods.asp

PHP

Database

MySQL

Fuzzy Logic

Design and Implementation of Proposed System (Flow Chart)

Summary