

Home automation using Arduino UNO:

Components:

There are main three components required in this project:

- ✓ Arduino UNO
- ✓ TSOP 1738
- ✓ RC-5 Remote control

iclub

1. ARDUINO UNO

1.1 Introduction to Arduino UNO

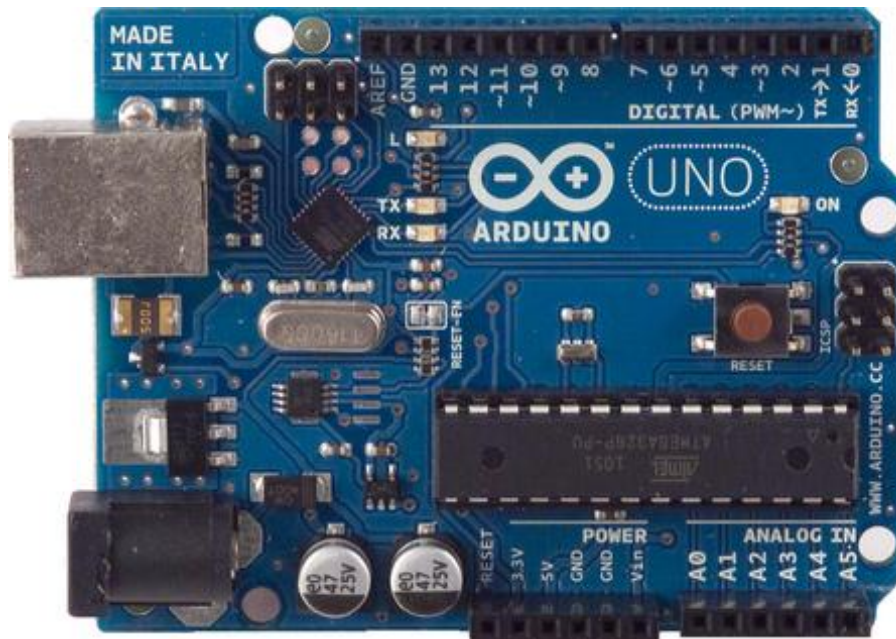


Fig.1.1 Arduino UNO Board

Arduino is a popular [open-source single-board microcontroller](#), descendant of the open-source [Wiring platform](#), designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an [Atmel AVR](#) processor and on-board [input/output](#) support. The software consists of a standard programming language compiler and the [boot loader](#) that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax and libraries), similar to [C++](#) with some slight simplifications and modifications, and a [Processing-based integrated development environment](#).

Current versions can be purchased pre-assembled;. Additionally, variations of the Italian-made Arduino—with varying levels of compatibility—have been released by third parties; some of them are programmed using the Arduino software. The Arduino project received an honorary mention in the Digital Communities category at the 2006 [Prix Ars Electronica](#).

An Arduino board consists of an 8-bit Atmel AVR [microcontroller](#) with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules known as **shields**. Official Arduinos have used the [megaAVR](#) series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt [linear regulator](#) and a 16 MHz [crystal oscillator](#) (or [ceramic resonator](#) in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip [flash memory](#), compared with other devices that typically need an external [programmer](#).

At a conceptual level, when using the Arduino software stack, all boards are programmed over an [RS-232](#) serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and [TTL](#)-level signals. Current Arduino boards are programmed via [USB](#), implemented using USB-to-serial adapter chips such as the [FTDI FT232](#). Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, [Bluetooth](#) or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, now superseded by the Duemilanove, for example, provides 14 digital I/O pins, six of which can produce [pulse-width modulated](#) signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application shields are also commercially available.

The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards provide male header pins on the underside of the board to be plugged into [solderless breadboards](#).

1.2 History of Arduino UNO

The project began in [Ivrea, Italy](#) (the site of the computer company [Olivetti](#)), in 2005 to make a device for controlling student-built interaction design projects less expensive than other prototyping systems available at the time. As of May 2011, more than 300,000 Arduino units are "in the wild." Founders Massimo Banzi and David Cuartielles named the project after [Arduin of Ivrea](#), the main historical character of the town. "Arduino" is an Italian masculine [first name](#), meaning "strong friend". The English version of the name is "Hardwin".

The Arduino project is a [fork](#) of the open-source [Wiring Platform](#). Wiring was created by Colombian artist and programmer [Hernando Barragán](#) as a master's thesis at the [Interaction Design Institute Ivrea](#) under the supervision of Massimo Banzi and [Casey Reas](#). Furthermore, Wiring is based on [Processing](#) and its [integrated development environment](#) created by Casey Reas and [Ben Fry](#).

“Arduino was built around the Wiring project of Hernando Barragan. Wiring was Hernando's thesis project at the Interaction Design Institute Ivrea. It was intended to be an electronics version of Processing that used our programming environment and was patterned after the Processing syntax. It was supervised by myself and Massimo Banzi, an Arduino founder. I don't think Arduino would exist without Wiring and I don't think Wiring would exist without Processing. And I know Processing would certainly not exist without [Design By Numbers](#) and [John Maeda](#).”

1.3 Basic Functions of Arduino Uno

The **setup()** function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

After creating a **setup()** function, the **loop()** function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond as it runs. Code in the **loop()** section of your sketch is used to actively control the Arduino board.

The code below won't actually do anything, but its structure is useful for copying and pasting to get you started on any sketch of your own. It also shows you how to make comments in your code.

Any line that starts with two slashes (*//*) will not be read by the compiler, so you can write anything you want after it. Commenting your code like this can be particularly helpful in explaining, both to yourself and others, how your program functions step by step.

```
void setup() {  
    // put your setup code here, to run once:  
}
```

```
void loop() {  
    // put your main code here, to run repeatedly:
```



- **setup()**

The **setup()** function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

- **loop()**

After creating a **setup()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

1.4 Software

Arduino Software



A screenshot of the Arduino IDE showing the "Blink" program, a simple beginner program.

Developer(s)	Arduino Software
Stable release	1.0 / November 30, 2011; 4 months ago ^[14]
Written in	Java
Operating system	Cross-platform
Type	Integrated development environment
License	LGPL or GPL license
Website	http://arduino.cc

The Arduino IDE is a cross-platform application written in [Java](#), and is derived from the IDE for the [Processing programming language](#) and the *Wiring* project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as [syntax highlighting](#), [brace matching](#), and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit [makefiles](#) or run programs on a [command-line interface](#). Although building on command-line is possible if required with some third-party tools such as [Ino](#).

The Arduino IDE comes with a C/C++ library called "Wiring" (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need define two functions to make a runnable program:

- `setup()` – a function run once at the start of a program that can initialize settings
- `loop()` – a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks a [LED](#) on and off. In the Arduino environment, the user might write a program like this:^[15]

```
#define LED_PIN 13

void setup () {
  pinMode (LED_PIN, OUTPUT);  // enable pin 13 for digital output
}

void loop () {
  digitalWrite (LED_PIN, HIGH); // turn on the LED
  delay (1000);                 // wait one second (1000 milliseconds)
  digitalWrite (LED_PIN, LOW);  // turn off the LED
  delay (1000);                 // wait one second
}
```

For the above code to work correctly, the positive side of the LED must be connected to pin 13 and the negative side of the LED must be connected to ground. The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple `main()` function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the [GNU toolchain](#) and [AVR Libc](#) to compile programs, and uses [avrdude](#) to upload programs to the board.

For educational purposes there is third party graphical development environment called [Minibloq](#) available under a different open source license.

1.5 How do I Connect an Arduino Uno to my PC?

Primary Software: LabVIEW Development Systems>>LabVIEW Professional Development System

Primary Software Version: 1.0

Primary Software Fixed Version: N/A

Secondary Software: N/A

Problem:

I want to use the LabVIEW Interface for Arduino Uno. How do I connect my Arduino Uno to my PC?

Solution:

Complete the following steps to connect your Arduino Uno to your PC for use with the LabVIEW Interface for Arduino:

Download the latest version of the Arduino IDE for your operating system from the following link: <http://arduino.cc/en/Main/Software>

- ✓ Extract the downloaded files to C:\Program Files.
- ✓ Attach the Arduino Uno to the PC using a USB Cable.
- ✓ Windows will attempt to install drivers for the Arduino but will not be able to find the correct drivers.

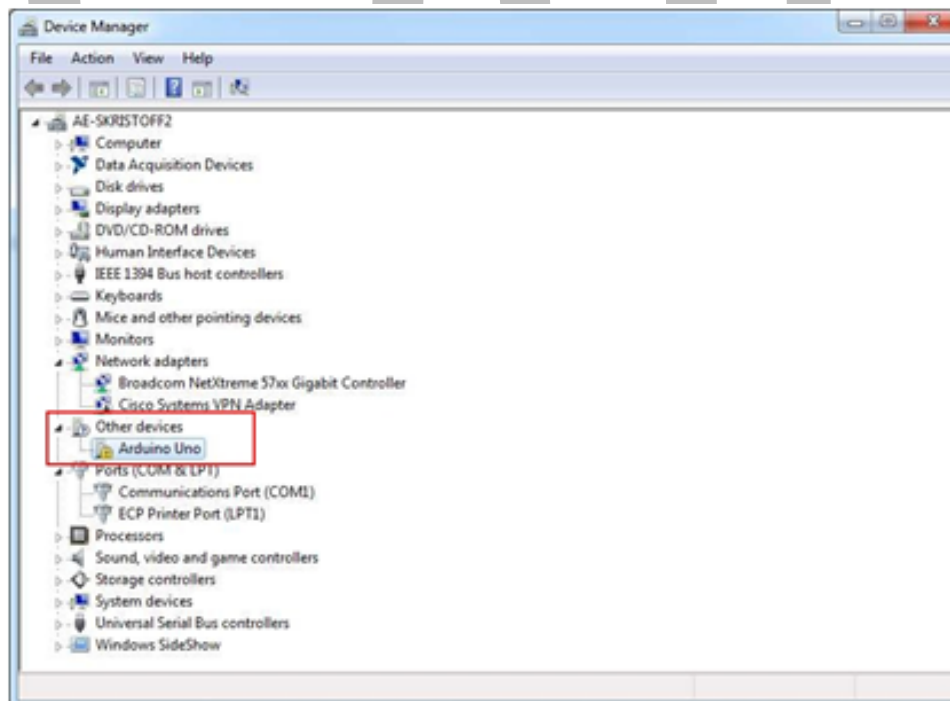
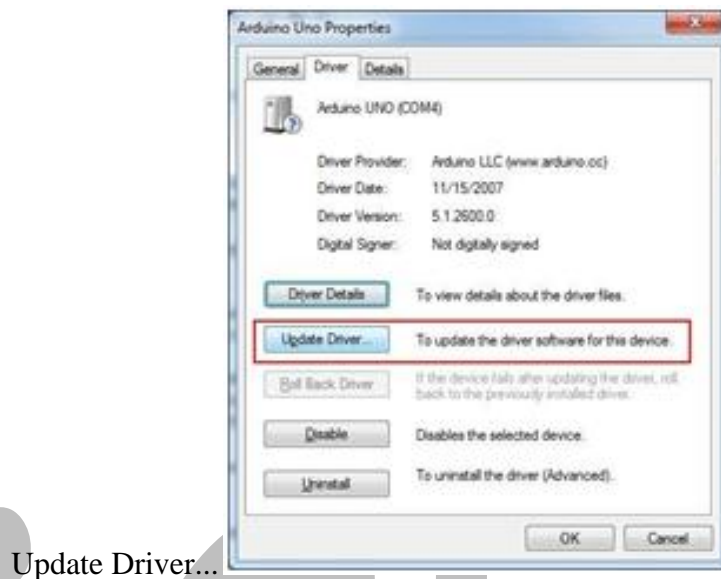
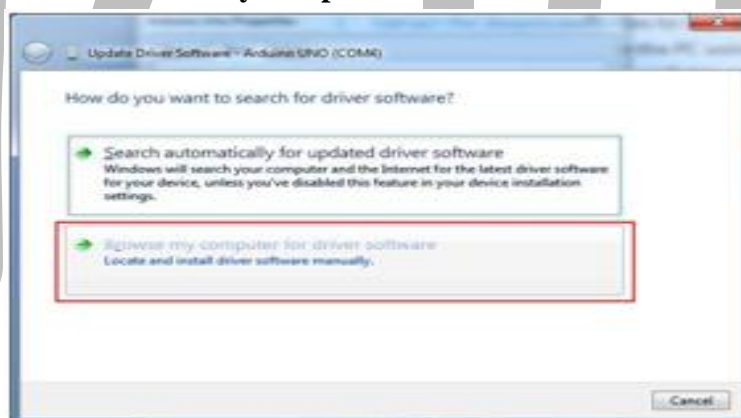


Fig.1.2 How do I Connect an Arduino Uno to my PC?

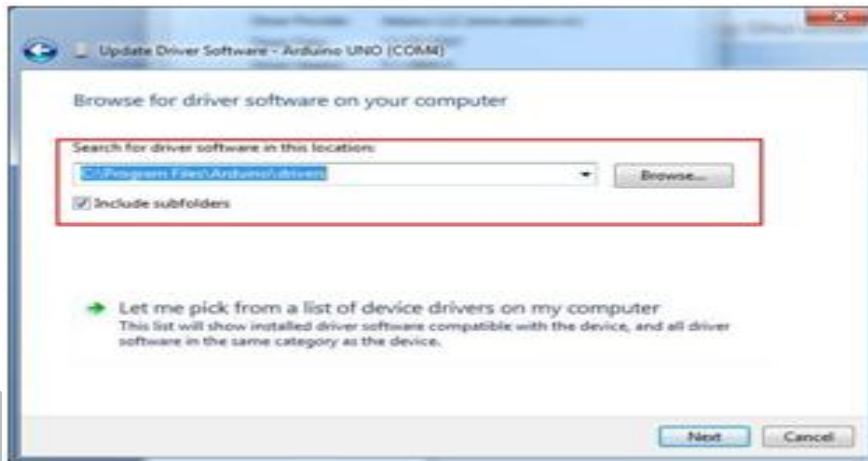
1. Click the **Driver** tab on the Arduino Uno properties window, then click



2. Choose **Browse my computer for driver software**.



3. Browse to C:\Program Files\Arduino-xxxx\drivers.

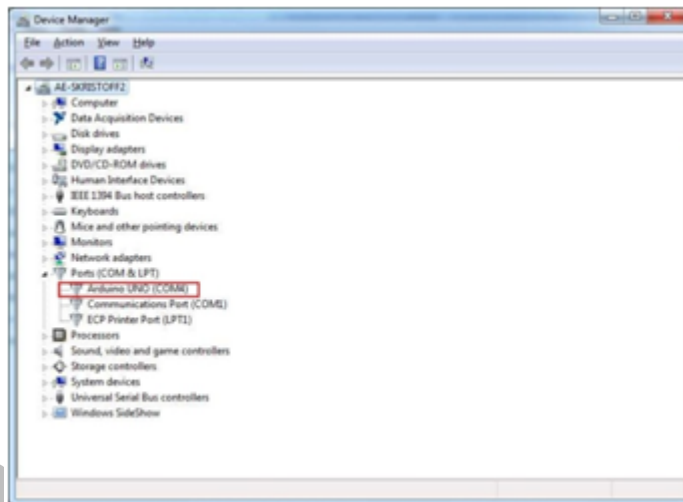


4. Click **Next**.

5. Click **Close**.



6. The Arduino Uno should now be listed under Ports(COM&LPT) in the Windows Device Manager.



1.6 IRremote Library

IRremote, by [Ken Shirriff](#), allows you to receive or transmit Infrared Remote Control codes. You can make your projects controlled by a remote, or make them control other devices like televisions and stereo components.

Basic Usage

IRremote acts like 2 libraries, one for sending and one for receiving. Usually it's easiest to find the codes to transmit by first using the receiver.

Receiving

IRrecv irrecv(**receivePin**)

Create the receiver object, using a name of your choice.

irrecv.enableIRIn()

Begin the receiving process. This will enable the timer interrupt which consumes a small amount of CPU every 50 μ s.

irrecv.decode(&results)

Attempt to receive a IR code. Returns true if a code was received, or false if nothing received yet. When a code is received, information is stored into "results".

results.decode_type: Will be one of the following: [NEC](#), [SONY](#), [RC5](#), [RC6](#), or [UNKNOWN](#).

results.value: The actual IR code (0 if type is UNKNOWN)

results.bits: The number of bits used by this code

results.rawbuf: An array of IR pulse times

results.rawlen: The number of items stored in the array

irrecv.resume()

After receiving, this must be called to reset the receiver and prepare it to receive another code.

irrecv.blink13(true)

Enable blinking the LED when during reception. Because you can't see infrared light, blinking the LED can be useful while troubleshooting, or just to give visual feedback.

2. Description of reciver TSOP17--

The TSOP17.. – series are miniaturized receivers forinfrared remote control systems. PIN diode and preamplifier are assembled on lead frame, the epoxy package is designed as IR filter. The demodulated output signal can directly be decoded by a microprocessor. TSOP17.. is the standard IR remote control receiver series, supporting all major transmission codes.

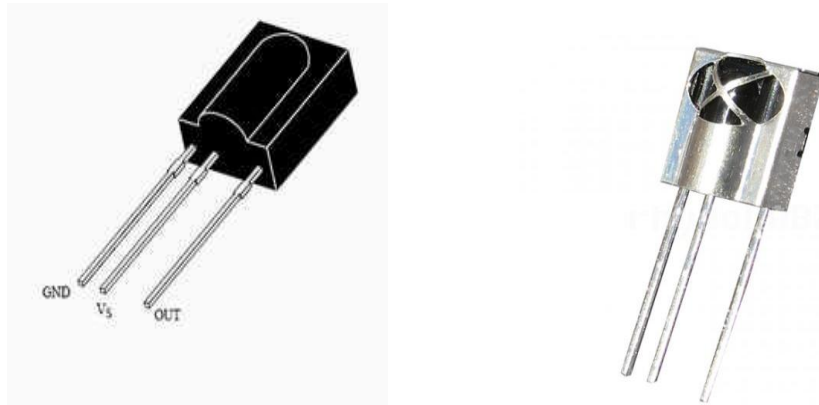


Fig.2.1 TSOP1738 Receiver

Features

- ✓ Photo detector and preamplifier in one package
- ✓ Internal filter for PCM frequency
- ✓ Improved shielding against electrical field disturbance
- ✓ TTL and CMOS compatibility
- ✓ Output active low
- ✓ Low power consumption
- ✓ High immunity against ambient light
- ✓ Continuous data transmission possible (up to 2400 bps)
- ✓ Suitable burst length ≥ 10 cycles/burst

Block Diagram

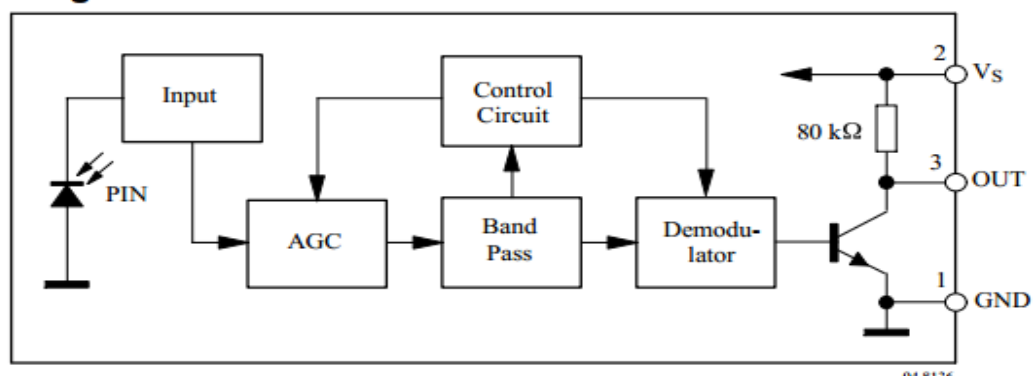


Fig 2.2 Block Diagram of TSOP 1738


3. RC-5 Remote Control

The advantage of the RC-5 protocol is that (when properly followed) any CD handset (for example) may be used to control any brand of CD player using the RC-5 protocol.

Protocol Details:

The basics of the protocol are well known. The handset contains a keypad and a transmitter [integrated circuit](#) (IC) driving an IR [LED](#). The command data is a [Manchester coded](#) bitstream modulating a 36 kHz carrier. (Often the carrier used is 38 kHz or 40 kHz, apparently due to misinformation about the actual protocol.) The IR signal from the transmitter is detected by a specialized IC with an integral photo-diode, and is amplified, filtered, and demodulated so that the receiving device can act upon the received command. RC-5 only provides a one-way link, with information traveling from the handset to the receiving unit.

The command comprises 14 bits:

- 
- A start bit, which is always logic 1 and allows the receiving IC to set the proper gain.
 - A field bit, which denotes whether the command sent is in the lower field (logic 1 = 0 to 63 decimal) or the upper field (logic 0 = 64 to 127 decimal). The field bit was added later by [Philips](#) when it was realized that 64 commands per device were insufficient. Previously, the field bit was combined with the start bit. Many devices still use this original system.
 - A control bit, which toggles with each button press. This allows the receiving device to distinguish between two successive button presses (such as "1", "1" for "11") as opposed to the user simply holding down the button and the repeating commands being interrupted by a person walking by, for example.
 - A five-bit system address, that selects one of 32 possible systems.
 - A six-bit command, that (in conjunction with the field bit) represents one of the 128 possible RC-5 commands.

The 36 kHz carrier frequency was chosen to render the system immune to interference from TV scan lines. Since the repetition of the 36 kHz carrier is 27.778 μ s and the duty factor is 25%, the carrier pulse duration is 6.944 μ s. Since the high half of each symbol (bit) of the RC-5 code word contains 32 carrier pulses, the symbol period is $64 \times 27.778 \mu\text{s} = 1.778 \text{ ms}$, and the 14 symbols (bits) of a complete RC-5 code word takes 24.889 ms to transmit. The code word is repeated every 113.778 ms ($4096 / 36 \text{ kHz}$) as long as a key remains pressed. (Again, please note that these timings are not strictly followed by all manufacturers, due to a lack of widespread distribution of accurate information on the RC-5 protocol.)

System and Command Codes

While the RC-5 protocol is well known and understood, what is not so well documented are the system number allocations and the actual RC-5 commands used for each system. The information provided below is the most complete and accurate information available at this time. It is from a printed document from Philips dated December 1992 that is unfortunately not available in electronic format (e.g., PDF), nor is an updated version available. This information is provided so that companies that wish to use the RC-5 protocol can use it properly, and avoid conflicts with other equipment that may or may not be using the correct system numbers and commands.

iclub

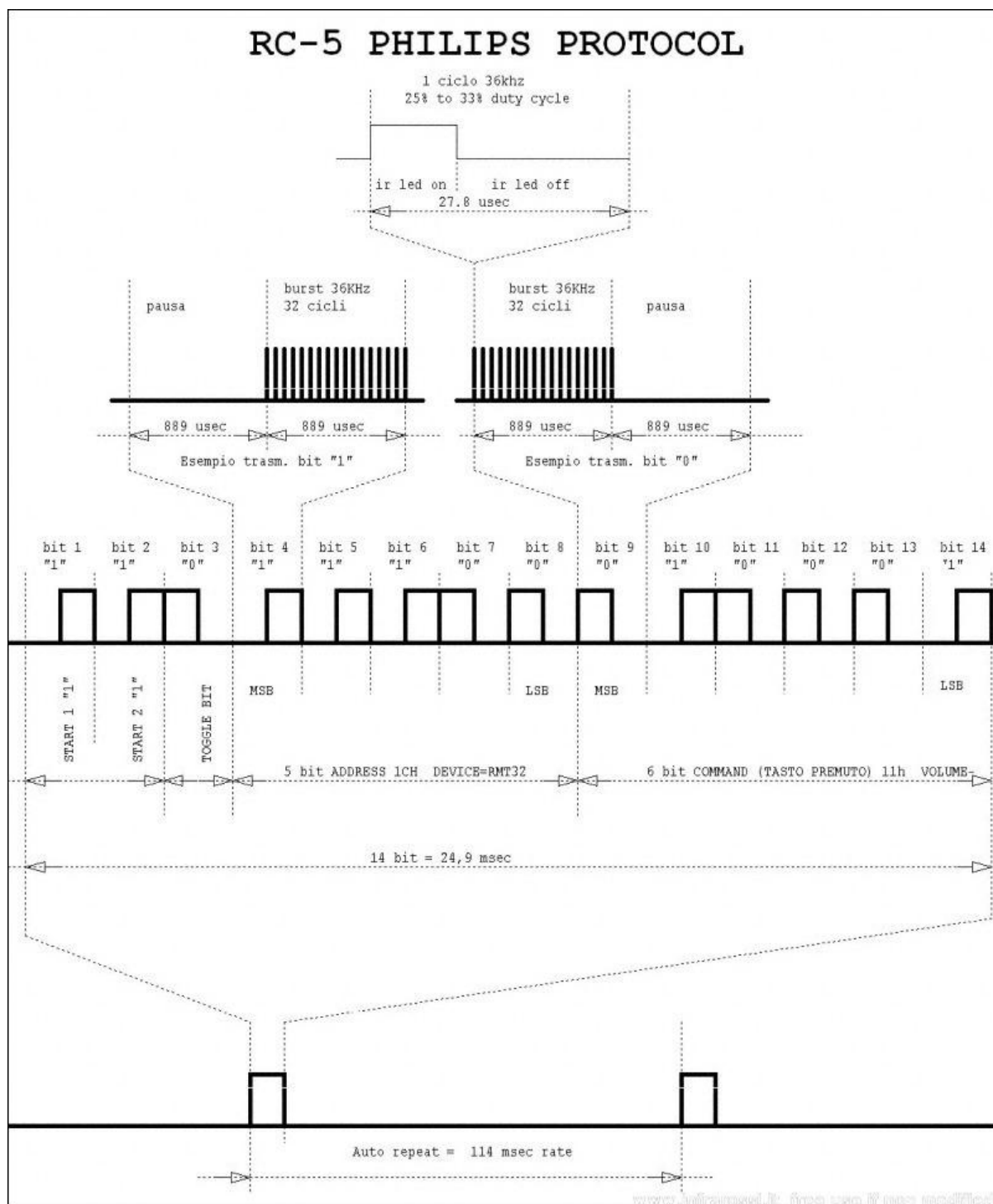


Fig.3.1 RC-5 remote control bit pattern

4) Circuit diagram :-

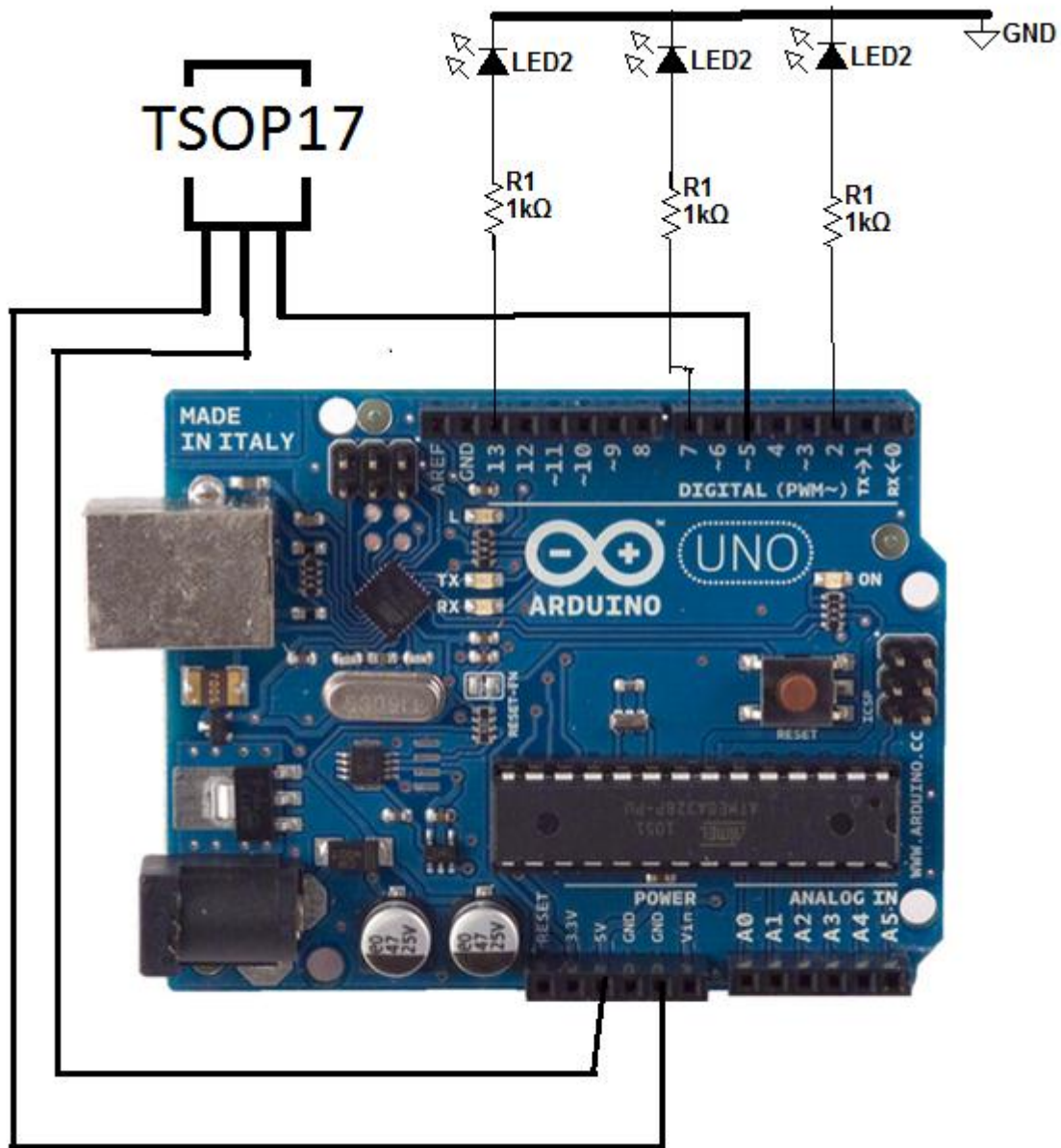


Fig.4.1 Circuit Diagram of Home automation Project

Programme:-

```
#include <IRremote.h>           //required library for IR receiver

#include <IRremoteInt.h>        //required library for IR receiver


void onoff(int p);              //function for switching on/off the device

int FAN = 7;                   //Declaration: Device-1 is connected to the 7th pin

int LIGHT1 = 2;                //Declaration: Device-1 is connected to the 2nd pin

int AC =13                     //Declaration: Device-1 is connected to the 13th pin

int RECV_PIN =5;               //Declaration: Receiver is connected to the 3rd pin

IRrecv irrecv(RECV_PIN);       //Declaration of Receiving object

decode_results results;         //received code stored in results will be decoded

long a,e;                      //variables for decoding

int p;                          //variable for device-choice

void setup()                   //Initialize the settings and runs once time
{
    long a=0,b=0,c=0,d=0;

    pinMode(1,OUTPUT);          //Declaration:Pin-1 act as output pin

    pinMode(7,OUTPUT);          //Declaration:Pin-7 act as output pin

    pinMode(2,OUTPUT);          //Declaration:Pin-2 act as output pin

    pinMode(13,OUTPUT);         //Declaration:Pin-13 act as output pin

    Serial.begin(9600);          //Serially receives at 9600 baud rate

    irrecv.enableIRIn();         // Start the receiver

    irrecv.blink13(true);        //LED at 13th pin will blink, if IR receiver receives
}
```

```

void loop()                                //Run programme continuously
{
    {
        if (irrecv.decode(&results))      //Returns to true when receiver receives IR code
        {
            a = results.value,DEC;         //value store in decimal in variable a
            Serial.println(a,DEC);         //Serially print value in serial shif monitor
            delay(1000);
            irrecv.resume();               //reset the receiver to receive second time code
        }
    }
    if (irrecv.decode(&results))           //Returns to true when receiver receives IR code
    {
        e = results.value,DEC;             //value store in decimal in variable e
        switch (e)                         //for checking a code for device
        {
            case 12582919:                 //code of 7th button of remote
                p=7;                       //selecting of 7th device
                delay(1000);
                onoff(p);                   // Syntax to CALL a Function
                break;
            case 12582914:                 //code of 2nd button of remote
                p=2;                       //selecting of 2nd device
                delay(1000);
                onoff(p);                   // Syntax to CALL a Function
                break;
        }
    }
}

```

```

    default:

loop();

    }

}

}

void onoff(int p)

{

    label: long m=0;

    irrecv.resume();           //reset the receiver for receiving next time
    delay(700);
    irrecv.decode(&results);
    m= results.value,DEC;       //code of on-off button will store in 'm' in decimal
    if(m == 12583021)           //checking code for on button
    {
        digitalWrite(p,HIGH);   //to switch on the device
    }
    if(m == 12583022)           //checking code for on button
    {
        digitalWrite(p,LOW);    //to switch on the device
    }
}

    if(m == 0)

    {

        goto label;             //continuous loop until m≠0
    }

return;

}

```