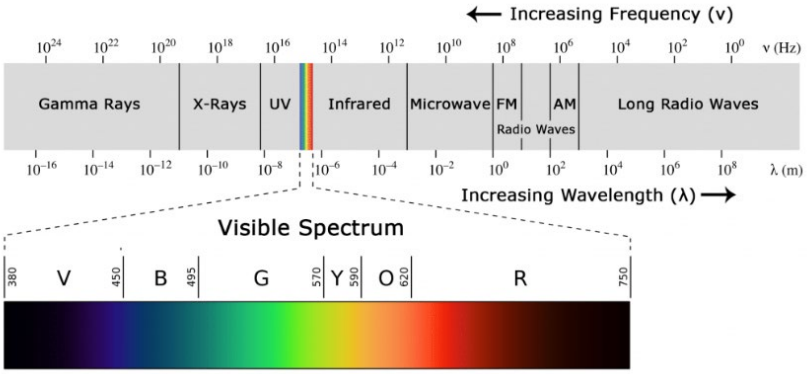
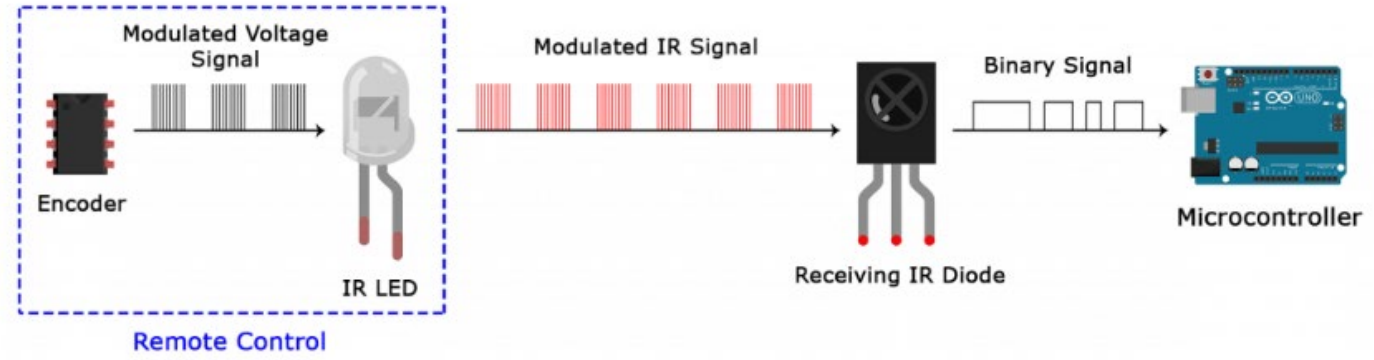
Telecomanda in Infrarosu

Introducere

Telecomenzile optice trimit un set de coduri binare folosind lumina infrarosie (cel mai adesea). Trimiterea codului binar se realizeaza cu o dioda ce emite lumina infrarosie.



Pentru a creste imunitatea transmisiei optice se foloseste o modulare a semnaului trimis, cel mai adesea cu o purtatoare de 38KHz.

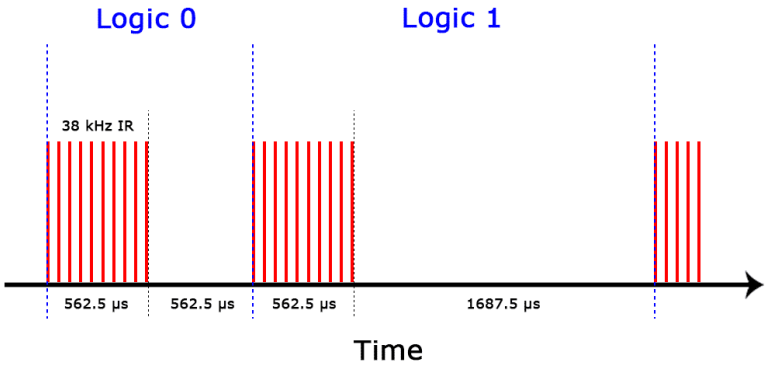


PROTOCOLE DE TRANSMISIE IR

Modelul în care semnalul IR modulat este convertit în binar este definit de un protocol de transmisie. Există multe protocoale de transmisie IR: Sony, Matsushita, NEC și RC5 sunt unele dintre cele mai comune protocoale. Protocolul NEC este, de asemenea, cel mai comun tip în proiectele Arduino, așa că îl vom folosi ca exemplu pentru a arăta cum transformă receptorul semnalul IR modulat într-unul binar.

„1” logic începe cu un impuls HIGH de 562,5 µs de IR de 38 kHz urmat de un impuls LOW de 1.687,5 µs.

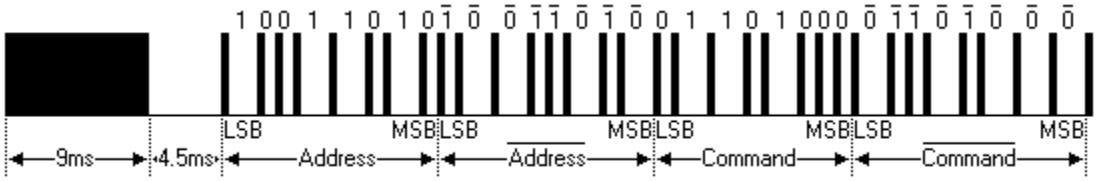
„0” logic este transmis cu un impuls HIGH de 562,5 µs urmat de un impuls LOW de 562,5 µs.



CODURI IR

De fiecare dată când se apasa un buton de pe telecomandă, este generat un cod hexazecimal unic. Aceasta este informația care este modulată și trimisă prin IR către receptor. Pentru a descifra ce tastă este apăsată, microcontrolerul receptor trebuie să știe ce cod corespunde fiecărei taste de pe telecomandă. Telecomenzi diferite trimit coduri diferite pentru apăsarea tastelor in functie de protocolul de transmisie utilizat.

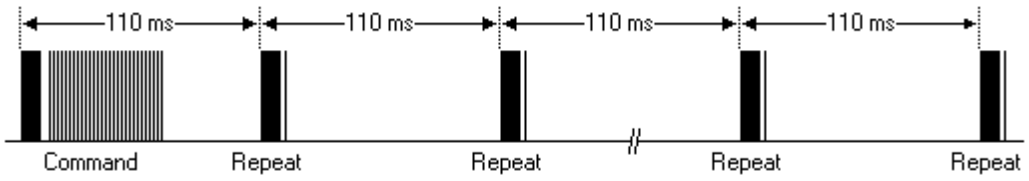
In cele ce urmeaza este prezentat un exemplu de modulare folosind protocolul NEC



Imaginea de mai sus arată o secvență tipică de impulsuri a protocolului NEC.

Notă: observati ca se trimite LSB (cel mai nesemnificativ bit) la inceput. Conform exemplului de mai sus, este trimisa adresa 0x59 si comanda 0x16. Transmisia incepe cu un “tren de impulsuri” de o durata aproximativa de 9ms, apoi urmeaza un nivel logic LOW de aprox. 4,5ms, apoi codul de adresa si de comanda. De obicei, atat valoarea de adresa cat si cea de comanda sunt trimise de emitator si in format negat (imediat dupa transmisia corecta) pentru validarea transmisiei si a corectitudinii informatiei (transmisiile mai de incredere folosesc protocoale CRC ptr. validarea unei receptii corecte).

Daca se tine apasata o perioada mai mare de timp tasta de la telecomanda, se trimite urmatorul mesaj:

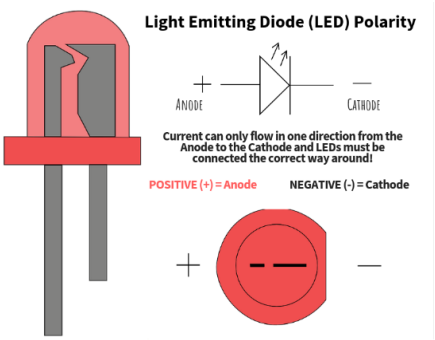


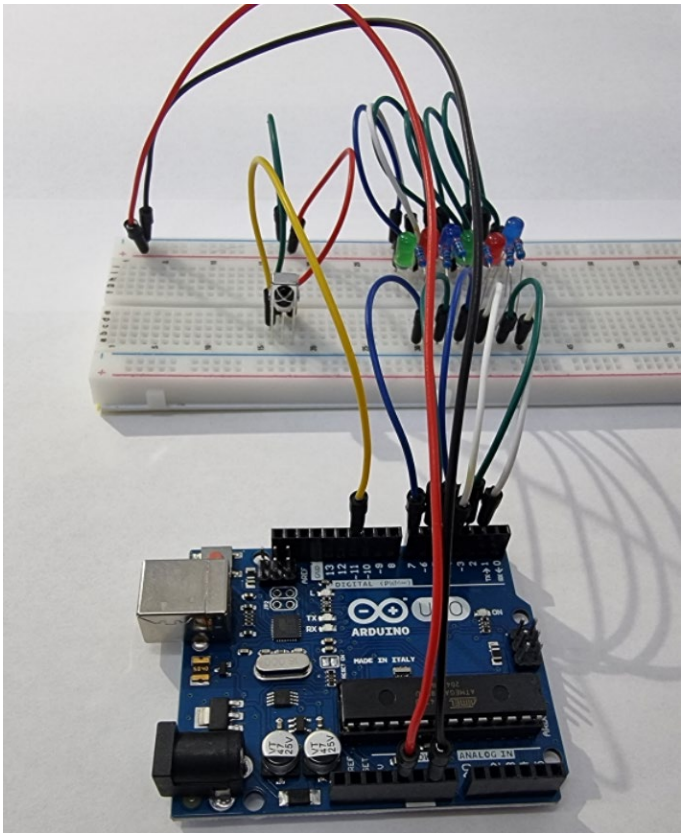
In cele ce urmeaza se va realiza montajul electronic.

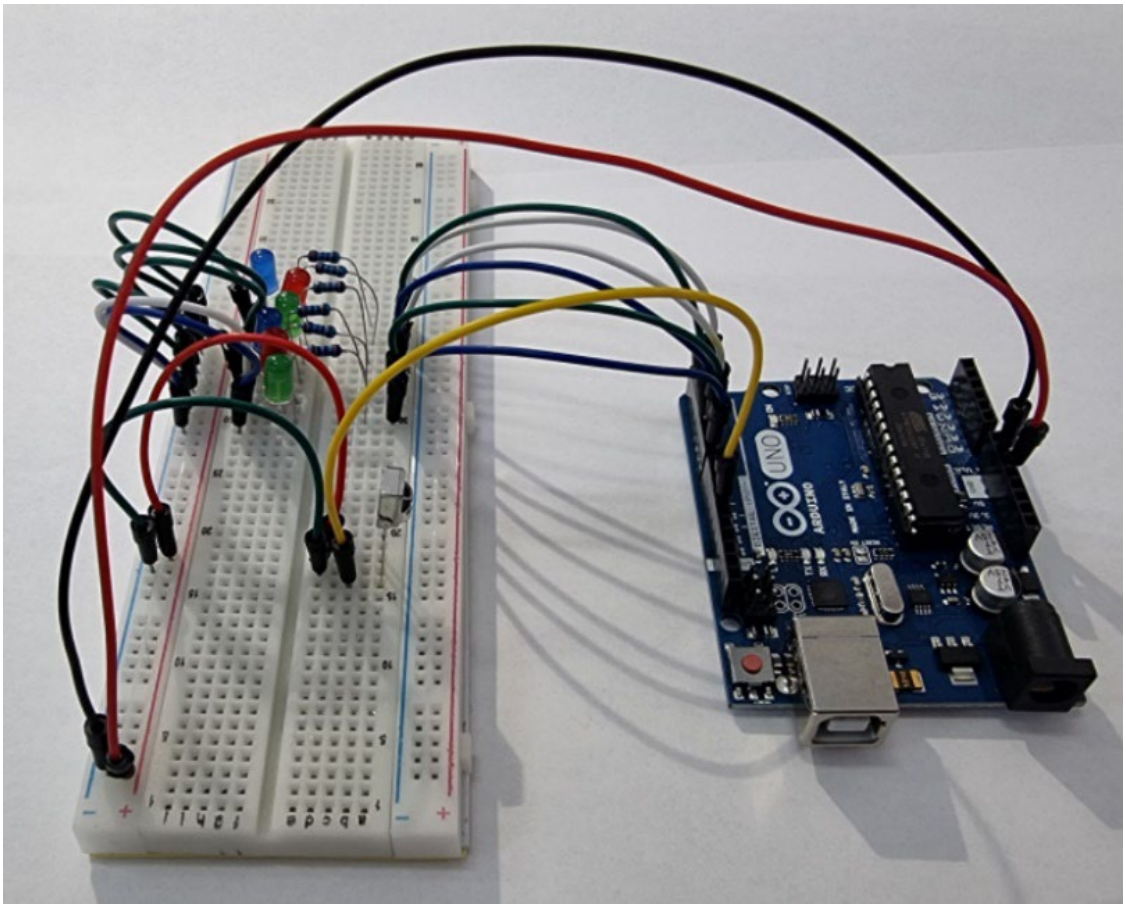
Receptorul IR se conecteaza cu pinul Vout la placa Arduino la pinul 11 si se alimenteaza cu +5V la Vcc (nu uitati sa conectati Vss(GND)).

Pinii 2,3,4,5,6,7 de la placa Arduino se conecteaza cu fire la placa breadboard, apoi se inseriaza rezistente de 220 ohm in serie cu LED-uri de diferite culori.

Terminalul mai lung al LED-ului este Anodul. Terminalul mai scurt al LED-ului (Catodul) se va conecta la VSS(GND).







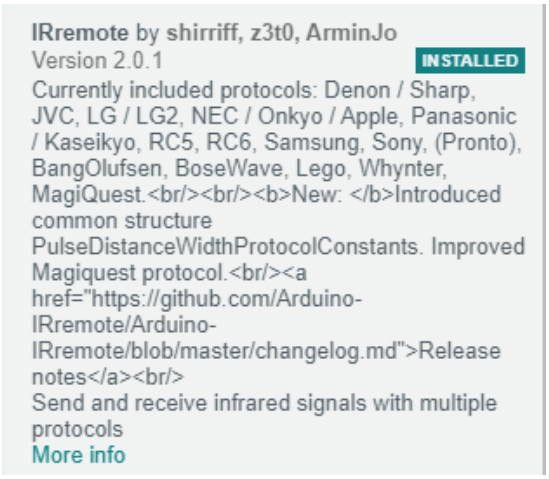
Se deschide programul Arduino si se copie soft-ul de mai jos:

```
#include <IRremote.h>
int RECV_PIN = 11;
int LED1 = 2;
int LED2 = 3;
int LED3 = 4;
int LED4 = 5;
int LED5 = 6;
int LED6 = 7;
long on1  = 0x00FFA25D;
long off1 = 0x00FFE01F;
long on2 = 0x00FF629D;
long off2 = 0x00FFA857;
long on3 = 0x00FFE21D;
long off3 = 0x00FF906F;
long on4 = 0x00FF22DD;
long off4 = 0x00FF6897;
long on5 = 0x00FF02FD;
long off5 = 0x00FF9867;
long on6 = 0x00FFC23D;
long off6 = 0x00FFB04F;
IRrecv irrecv(RECV_PIN);
decode_results results;
// Dumps out the decode_results structure.
// Call this after IRrecv::decode()
// void * to work around compiler issue
//void dump(void *v) {
//  decode_results *results = (decode_results *)v
void dump(decode_results *results) {
  int count = results->rawlen;
  if (results->decode_type == UNKNOWN)
    {
     Serial.println("Could not decode message");
    }
  else
   {
    if (results->decode_type == NEC)
      {
       Serial.print("Decoded NEC: ");
```

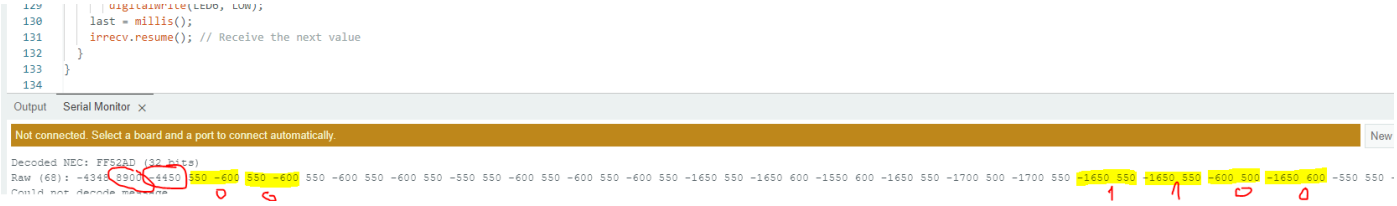
```
      }
    else if (results->decode_type == SONY)
      {
       Serial.print("Decoded SONY: ");
      }
    else if (results->decode_type == RC5)
      {
       Serial.print("Decoded RC5: ");
      }
    else if (results->decode_type == RC6)
      {
       Serial.print("Decoded RC6: ");
      }
     Serial.print(results->value, HEX);
     Serial.print(" (");
     Serial.print(results->bits, DEC);
     Serial.println(" bits)");
   }
     Serial.print("Raw (");
     Serial.print(count, DEC);
     Serial.print("): ");

  for (int i = 0; i < count; i++)
     {
      if ((i % 2) == 1) {
      Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
     }
    else
     {
      Serial.print(-(int)results->rawbuf[i]*USECPERTICK, DEC);
     }
    Serial.print(" ");
     }
      Serial.println("");
     }

void setup()
 {
  pinMode(RECV_PIN, INPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED5, OUTPUT);
  pinMode(LED6, OUTPUT);
  pinMode(13, OUTPUT);
  Serial.begin(9600);

  irrecv.enableIRIn(); // Start the receiver
 }

int on = 0;
unsigned long last = millis();

void loop()
{
  if (irrecv.decode(&results))
   {
    // If it's been at least 1/4 second since the last
    // IR received, toggle the relay
    if (millis() - last > 250)
      {
       on = !on;
//       digitalWrite(8, on ? HIGH : LOW);
       digitalWrite(13, on ? HIGH : LOW);
       dump(&results);
      }
    if (results.value == on1 )
       digitalWrite(LED1, HIGH);
    if (results.value == off1 )
       digitalWrite(LED1, LOW);
    if (results.value == on2 )
       digitalWrite(LED2, HIGH);
    if (results.value == off2 )
       digitalWrite(LED2, LOW);
    if (results.value == on3 )
       digitalWrite(LED3, HIGH);
    if (results.value == off3 )
       digitalWrite(LED3, LOW);
    if (results.value == on4 )
       digitalWrite(LED4, HIGH);
```

```
    if (results.value == off4 )
       digitalWrite(LED4, LOW);
    if (results.value == on5 )
       digitalWrite(LED5, HIGH);
    if (results.value == off5 )
       digitalWrite(LED5, LOW);
    if (results.value == on6 )
       digitalWrite(LED6, HIGH);
    if (results.value == off6 )
       digitalWrite(LED6, LOW);
    last = millis();
    irrecv.resume(); // Receive the next value
  }
}
```

Atentie: pentru compilare se foloseste libraria IRremote cu versiunea 2.0.1 (conform foto mai jos)



In fereastra Monitor din programul Arduino apare un mesaj ca in exemplul de mai jos:



Se observa bitii de 1 si 0 logic precum si timpii de 8900us respective 4450us. (acestea sunt valori reale masurate cu Timerul microcontrolerului).

// Se foloseste Intreruperea Externa 0, pin 2 placa Arduino

```
boolean nec_ok = 0;
byte  i, nec_state = 0, command, inv_command;
unsigned int address;
unsigned long nec_code;

void setup() {
  Serial.begin(9600);
  // Timer1 module configuration
  TCCR1A = 0;
  TCCR1B = 0;                                   // Disable Timer1 module
  TCNT1  = 0;                                   // Set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;                                   // enable Timer1 overflow interrupt
  attachInterrupt(0, remote_read, CHANGE);      // Enable external interrupt (INT0)
}

void remote_read() {
unsigned int timer_value;
  if(nec_state != 0){
    timer_value = TCNT1;                        // Store Timer1 value
    TCNT1 = 0;                                  // Reset Timer1
  }
  switch(nec_state){
   case 0 :                                     // Start receiving IR data (we're at the beginning of 9ms
pulse)
    TCNT1  = 0;                                 // Reset Timer1
    TCCR1B = 2;                                 // Enable Timer1 module with 1/8 prescaler ( 2 ticks every
1 us)
    nec_state = 1;                              // Next state: end of 9ms pulse (start of 4.5ms space)
    i = 0;
    return;
   case 1 :                                     // End of 9ms pulse
    if((timer_value > 19000) || (timer_value < 17000)){         // Invalid interval ==> stop decoding and
reset
      nec_state = 0;                            // Reset decoding process
      TCCR1B = 0;                               // Disable Timer1 module
    }
    else
      nec_state = 2;                            // Next state: end of 4.5ms space (start of 562µs pulse)
    return;
   case 2 :                                     // End of 4.5ms space
    if((timer_value > 10000) || (timer_value < 8000)){
      nec_state = 0;                            // Reset decoding process
      TCCR1B = 0;                               // Disable Timer1 module
    }
    else
      nec_state = 3;                            // Next state: end of 562µs pulse (start of 562µs or 1687µs
space)
    return;
   case 3 :                                     // End of 562µs pulse
    if((timer_value > 1400) || (timer_value < 800)){           // Invalid interval ==> stop decoding and
reset
      TCCR1B = 0;                               // Disable Timer1 module
      nec_state = 0;                            // Reset decoding process
    }
    else
      nec_state = 4;                            // Next state: end of 562µs or 1687µs space
    return;
   case 4 :                                     // End of 562µs or 1687µs space
    if((timer_value > 3600) || (timer_value < 800)){           // Time interval invalid ==> stop decoding
      TCCR1B = 0;                               // Disable Timer1 module
      nec_state = 0;                            // Reset decoding process
      return;
    }
    if( timer_value > 2000)                     // If space width > 1ms (short space)
      bitSet(nec_code, (31 - i));               // Write 1 to bit (31 - i)
    else                                        // If space width < 1ms (long space)
      bitClear(nec_code, (31 - i));             // Write 0 to bit (31 - i)
    i++;
    if(i > 31){                                 // If all bits are received
      nec_ok = 1;                               // Decoding process OK
      detachInterrupt(0);                       // Disable external interrupt (INT0)
      return;
    }
    nec_state = 3;                              // Next state: end of 562µs pulse (start of 562µs or 1687µs
space)
  }
}
```

```
ISR(TIMER1_OVF_vect) {                          // Timer1 interrupt service routine (ISR)
  nec_state = 0;                                // Reset decoding process
  TCCR1B = 0;                                   // Disable Timer1 module
}

void loop() {
  if(nec_ok){                                   // If the mcu receives NEC message with successful
    nec_ok = 0;                                 // Reset decoding process
    nec_state = 0;
    TCCR1B = 0;                                 // Disable Timer1 module
    address = nec_code >> 16;
    command = nec_code >> 8;
    inv_command = nec_code;

    Serial.print("adresa este: ");
    Serial.println(address,HEX);
    Serial.print("comanda este: ");
    Serial.println(command,HEX);
    Serial.print("inv_command este: ");
    Serial.println(inv_command,HEX);
    attachInterrupt(0, remote_read, CHANGE);     // Enable external interrupt (INT0)
  }
}
/*
https://simple-circuit.com/arduino-nec-remote-control-decoder/
I used Timer1 module to measure pulses and spaces widths, it is configured so that is increments by 2 every
1 us (1/8 prescaler).
The interval [ 9500µs, 8500µs ] is used for the 9ms pulse and for the 4.5ms space the interval
[ 5000µs, 4000µs ] is used.
The 562.5µs pulse is checked with the interval [ 700µs, 400µs ] .
For the 562.5µs or 1687.5µs space I used the interval [ 1800µs, 400µs ], and to know if its a short or long
space I used a length of 1000µs.
In Timer ticks:
[ 9500µs, 8500µs ] = [ 19000 , 17000 ] ticks
[ 5000µs, 4000µs ] = [ 10000 , 8000 ] ticks
[ 700µs, 400µs ]    = [ 1400 , 800 ] ticks
[ 1800µs, 400µs ]  = [3600 , 800 ] ticks
*/
```