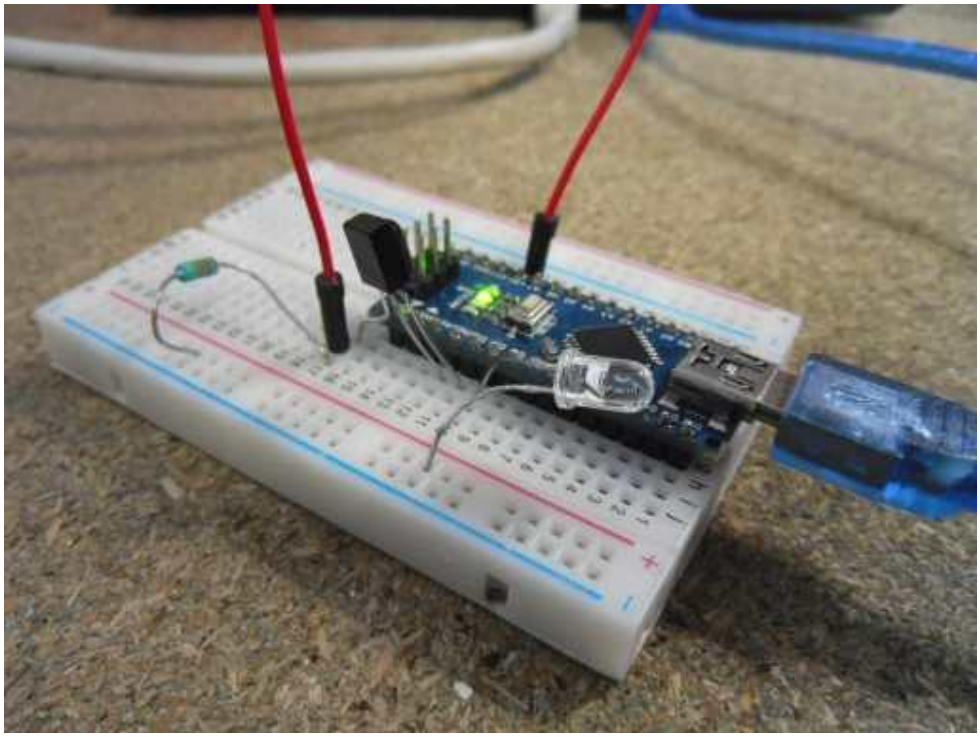
Arduino - Télécommande Infra-rouge

Fonctionnement de base

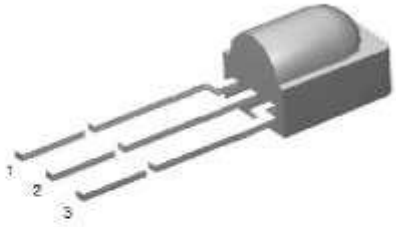
Avec 3 composants seulement, on peut faire joujou en lisant et en simulant une télécommande Infra-Rouge standard.

On va utiliser un Arduino nano (ou équivalent), un récepteur décodeur (dont la sortie est sur D2) et une LED infra-rouge branchée sur D3.

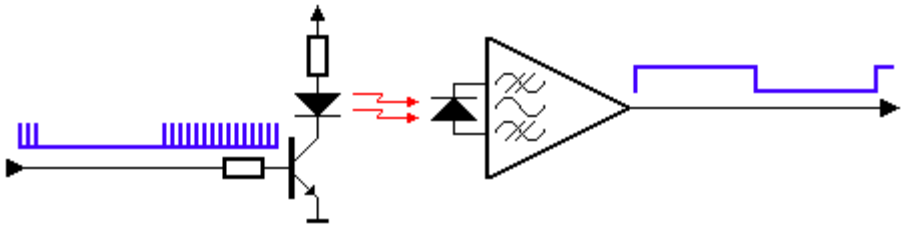


Les télécommandes infra rouge utilisent en général un signal constitué d'une suite de « rafales » d'oscillations de fréquence déterminée (38Khz le plus souvent), l'existence ou pas du signal et/ou les transitions de signal constituant des « zéro » et « un » du code émis.

Le récepteur infra rouge TSOP38238 reçoit le 38Khz et sort des 1 ou 0 selon la réception ou pas du signal (voir l'excellente doc en anglais sb projects ir). A noter que si la fréquence n'est pas 38Khz exactement, mais pas trop loin, ça pourra fonctionner quand même, mais avec moins de sensibilité.



Pinning:
1 = OUT, 2 = GND, 3 = V_S



Avec un logiciel adapté, on peut lire les durées des codes selon les touches de la télécommande actionnées.

```
/* Raw IR decoder sketch!

 This sketch/program uses the Arduno and a PNA4602 to
 decode IR received. This can be used to make a IR receiver
 (by looking for a particular code)
 or transmitter (by pulsing an IR LED at ~38KHz for the
 durations detected

 Code is public domain, check out www.ladyada.net and adafruit.com
 for more tutorials!
 */

// We need to use the 'raw' pin reading methods
// because timing is very important here and the digitalRead()
// procedure is slower!
//uint8_t IRpin = 2;
// Digital pin #2 is the same as Pin D2 see
// http://arduino.cc/en/Hacking/PinMapping168 for the 'raw' pin mapping
#define IRpin_PIN      PIND
#define IRpin          2

// the maximum pulse we'll listen for - 65 milliseconds is a long time
```

```
#define MAXPULSE 65000

// what our timing resolution should be, larger is better
// as its more 'precise' - but too large and you wont get
// accurate timing
#define RESOLUTION 20

// we will store up to 100 pulse pairs (this is -a lot-)
uint16_t pulses[100][2];  // pair is high and low pulse
uint8_t currentpulse = 0; // index for pulses we're storing

void setup(void) {
  Serial.begin(9600);
  Serial.println("Ready to decode IR!");
}

void loop(void) {
  uint16_t highpulse, lowpulse;  // temporary storage timing
  highpulse = lowpulse = 0; // start out with no pulse length


//  while (digitalRead(IRpin)) { // this is too slow!
    while (IRpin_PIN & (1 << IRpin)) {
     // pin is still HIGH

     // count off another few microseconds
     highpulse++;
     delayMicroseconds(RESOLUTION);

     // If the pulse is too long, we 'timed out' - either nothing
     // was received or the code is finished, so print what
     // we've grabbed so far, and then reset
     if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
       printpulses();
       currentpulse=0;
       return;
     }
  }
  // we didn't time out so lets stash the reading
  pulses[currentpulse][0] = highpulse;

  // same as above
  while (! (IRpin_PIN & _BV(IRpin))) {
     // pin is still LOW
     lowpulse++;
     delayMicroseconds(RESOLUTION);
     if ((lowpulse >= MAXPULSE)  && (currentpulse != 0)) {
       printpulses();
       currentpulse=0;
       return;
```

```
    }
  }
  pulses[currentpulse][1] = lowpulse;

  // we read one high-low pulse successfully, continue!
  currentpulse++;
}

void printpulses(void) {
  Serial.println("\n\r\n\rReceived: \n\rOFF \tON");
  for (uint8_t i = 0; i < currentpulse; i++) {
    Serial.print(pulses[i][0] * RESOLUTION, DEC);
    Serial.print(" usec, ");
    Serial.print(pulses[i][1] * RESOLUTION, DEC);
    Serial.println(" usec");
  }
}
```

Mais il y a mieux, on peut utiliser une bibliothèque va se charger de cela et qui, en plus, reconnait la plupart des codes de télécommandes; pour cela il faut télécharger le zip à Bibliothèque Arduino IRremote. Installer la par la commande Croquis / Include Library / Add ZIP Library. Il peut être nécessaire de redémarrer le programme Arduino pour prendre en compte la nouvelle bibliothèque. Vous devez voir les exemples de la bibliothèque dans le menu Fichier / Exemples /IRremote.

Parmi ces exemples, le programme IRrcvdump permet de lister sur le moniteur série la marque et le numéro de la touche de la télécommande (si elle est reconnue), ainsi que les valeurs brutes des temps de 0 et 1. Si la télécommande est reconnue, il suffit de noter le code et le nombre de bits de chaque touche pour pouvoir le « rejouer » par le programme IRsendDemo.

On peut utiliser n'importe quelle entrée/sortie digitale pour brancher le capteur: ici, on remplacera int RECV_PIN = 11 par int RECV_PIN = 2; dans IRrcvdump. Par contre, en ce qui concerne l'émission, la bibliothèque utilise de manière très approfondie les fonctions PWM et les timers du micro-chip et il n'est pas aisé (voir même pas possible dans certains cas) de changer la broche de sortie; il faut donc brancher la diode infra-rouge sur la sortie D3. De même le branchement de la diode est important: il faut faire en sorte qu'un niveau 1 « allume » la diode, donc brancher la résistance et la diode en série entre la broche D3 et le 0v (GND), en prenant bien garde de brancher l'anode (fil long) coté micro-contrôleur et la cathode (coté plat du boitier coté 0V.

A voir: les limites en courant des sorties des micro-contrôleurs et le besoin des diodes IR.

On remplacera l'instruction irsend.sendSony(0xa90, 12); par irsend.sendSony(« code » , « bits »), ou « code » et « bits » sont les codes et nombre de bits donnés par le programme de réception (le code donné est en hexadécimal, donc on écrira 0x10 pour 10 ou 0x2FD807F pour 2FD807F.

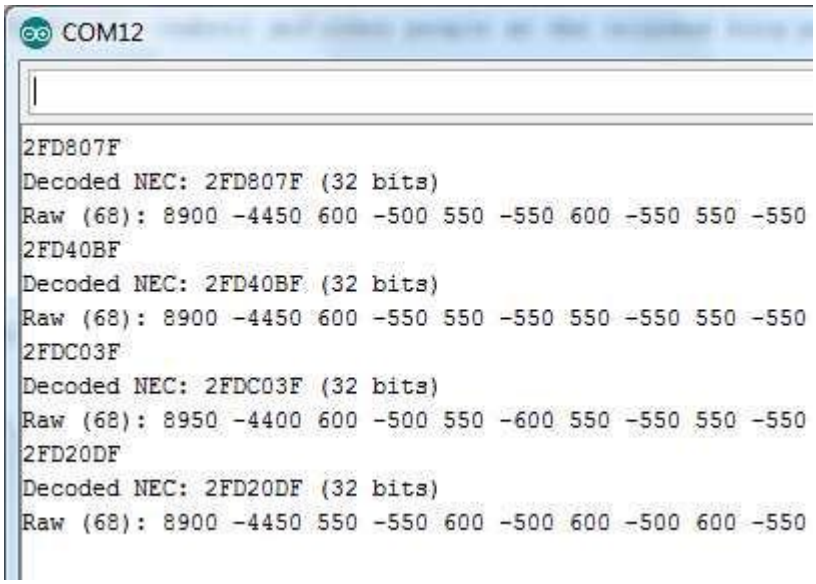
Exemples:

Le programme IRrecvDump:

```
/*
 * IRremote: IRrecvDump - dump details of IR codes with IRrecv
 * An IR detector/demodulator must be connected to the input RECV_PIN.
 * Version 0.1 July, 2009
 * Copyright 2009 Ken Shirriff
 * http://arcfn.com
 * JVC and Panasonic protocol added by Kristian Lauszus (Thanks to zenwheel and other
people at the original blog post)
 * LG added by Darryl Smith (based on the JVC protocol)
 */

#include <IRremote.h>

/*
*  Default is Arduino pin D11. <---------------------- ici D2
*  You can change this to another available Arduino Pin.
*  Your IR receiver should be connected to the pin defined here
*/
//int RECV_PIN = 11; // <----------------------------- mis en commentaire
int RECV_PIN = 2;    // <---------------------------- ajout

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}


void dump(decode_results *results) {
  // Dumps out the decode_results structure.
  // Call this after IRrecv::decode()
  int count = results->rawlen;
  if (results->decode_type == UNKNOWN) {
    Serial.print("Unknown encoding: ");
  }
  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: ");
  }
  else if (results->decode_type == PANASONIC) {
    Serial.print("Decoded PANASONIC - Address: ");
    Serial.print(results->address, HEX);
    Serial.print(" Value: ");
  }
  else if (results->decode_type == LG) {
    Serial.print("Decoded LG: ");
  }
  else if (results->decode_type == JVC) {
    Serial.print("Decoded JVC: ");
  }
  else if (results->decode_type == AIWA_RC_T501) {
    Serial.print("Decoded AIWA RC T501: ");
  }
  else if (results->decode_type == WHYNTER) {
    Serial.print("Decoded Whynter: ");
  }
  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 = 1; i < count; i++) {
    if (i & 1) {
      Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
    }
    else {
      Serial.write('-');
      Serial.print((unsigned long) results->rawbuf[i]*USECPERTICK, DEC);
    }
    Serial.print(" ");
  }
  Serial.println();
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    dump(&results);
    irrecv.resume(); // Receive the next value
  }
}
```

Résultats:



Dans l'image ci-dessus, on a appuyé successivement sur les touches 1,2,3,4 d'une télécommande Toshiba pour téléviseur; on voit que les touches sont décodées selon un protocole NEC avec comme code:

1. 2FD807F, l'instruction d'envoi sera donc irsend.sendSony(0x2FD807F, 32);
2. 2FD40BF, l'instruction d'envoi sera donc irsend.sendSony(0x2FD40BF, 32);
3. 2FDC03F, l'instruction d'envoi sera donc irsend.sendSony(0x2FDC03F, 32);
4. 2FD20DF, l'instruction d'envoi sera donc irsend.sendSony(0x2FD20DF, 32);

Exemple avec une TV Sony, on simule alternativement les touches 1 et 4 toutes les 10 secondes (les codes sont émis 3 fois de suite chacun)

```
/*
 * IRremote: IRsendDemo - demonstrates sending IR codes with IRsend
 * An IR LED must be connected to Arduino PWM pin 3.
 * Version 0.1 July, 2009
 * Copyright 2009 Ken Shirriff
 * http://arcfn.com
 */


#include <IRremote.h>

IRsend irsend;

void setup()
{
}
```

```
void loop() {
	for (int i = 0; i < 3; i++) {
		irsend.sendSony(0x10,12); // touche 1 pour une TV Vaio sony
		delay(40);
	}
	delay(10000); //5 second delay between each signal burst
	for (int i = 0; i < 3; i++) {
		irsend.sendSony(0xc10,12); //// touche 4 pour une TV Vaio sony
		delay(40);
	delay(10000); //5 second delay between each signal burst
	}
}
```

Vers un prototype fonctionnel

Bon, c'est bien joli de lire et d'envoyer des codes de touches, mais il faudrait faire ça de manière un peu plus intéressante; 'idée est donc de faire un petit boitier qui permettrait d'enregistrer et de rejouer des codes, éventuellement multiples (Une application, par exemple serait de commander l'allumage simultané d'un téléviseur et du décodeur TNT associé).

Pour cela, on va essayer de monter le module nano, la diode IR et le décodeur dans un boitier, en ajoutant quatre boutons et une LED témoin. On ne va pas s'embêter à essayer de trouver un boitier, on est dans un fablab, non? Donc on s'imprime un boitier en 3d, avec un trou pour le connecteur mini-usb, on verra pour les autres trous après.

On fabrique avec une plaque de CI trouée et pastillée un support pour les 4 boutons, sur une autre on met la LED et le récepteur IR, un programme pour tester les boutons et la LED témoin; un programme tout bête qui va consister chaque seconde à allumer la LED lorsqu'un des boutons sera actif (on appuie dessus), et d'écrire sur le moniteur série l'état de chaque bouton (Actif, Inactif).

Quelques explications:

- la LED témoin est connectée entre la sortie du nano (cathode) et le +5V à travers une résistance (470 ohm convient pour une tension de 5.5v max, et environ 10mA, ça suffit pour l'allumer); il faut donc mettre à 0 la sortie du nano pour l'allumer.
- pour éviter une entrée flottante, donc une lecture aléatoire quand les boutons sont inactifs, il faut fixer le potentiel de l'entrée. Les sorties du nano peuvent présenter des résistances de « pull up » au +5V. Pour activer ces résistances, il faut initialiser les entrées au mode INPUT_PULLUP.

```
/*


 The circuit:
 * infra red LED to digital 3 (needed to use PWM)
 * infra red decoder output attached to digital 2
 * pushbuttons attached to digital 4,5,6,7 other side to ground : so when pressed -state
LOW
 * signal LED cathode attached to digital 8. Anode attached to one side of 470 ohm
resistor, other side to ground : so  ouput LOW to liht LED

 */

// constants won't change. They're used here to
// set pin numbers:
const int button1Pin = 4;     // the number of the pushbutton pin
const int button2Pin = 5;     // the number of the pushbutton pin
const int button3Pin = 6;     // the number of the pushbutton pin
const int button4Pin = 7;     // the number of the pushbutton pin
const int ledPin     = 8;     // the number of the LED pin

String mess;

// variables will change:
int button1State = 0;         // variable for reading the pushbutton status
int button2State = 0;         // variable for reading the pushbutton status
int button3State = 0;         // variable for reading the pushbutton status
int button4State = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input with pull_up resistor, to prevent floating
  pinMode(button1Pin, INPUT_PULLUP);
  pinMode(button2Pin, INPUT_PULLUP);
  pinMode(button3Pin, INPUT_PULLUP);
  pinMode(button4Pin, INPUT_PULLUP);

  Serial.begin(9600);
}

void loop() {
  // read the state of the pushbutton value:
  button1State = digitalRead(button1Pin);
  mess=(button1State)? "INactif" : "ACTIF";
  Serial.println("Bouton 1: " + mess);
  button2State = digitalRead(button2Pin);
  mess=(button2State)? "INactif" : "ACTIF";
  Serial.println("Bouton 2: " + mess);
  button3State = digitalRead(button3Pin);
```

```
  mess=(button3State)? "INactif" : "ACTIF";
  Serial.println("Bouton 3: " + mess);
  button4State = digitalRead(button4Pin);
  mess=(button4State)? "INactif" : "ACTIF";
  Serial.println("Bouton 4: " + mess);
  Serial.println("----------------------");

  // check if the pushbutton is pressed.
  // if it is, the buttonState is LOW:
  if (button1State == LOW || button2State == LOW || button3State == LOW || button4State
== LOW) {
    // turn LED on:
    digitalWrite(ledPin, LOW);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, HIGH);
  }
  delay(1000);
}
```

Encours de montage: