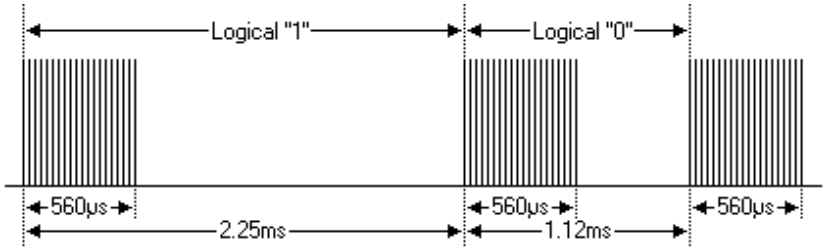
Protocolo NEC

Controlar la recepción de un mando a distancia.

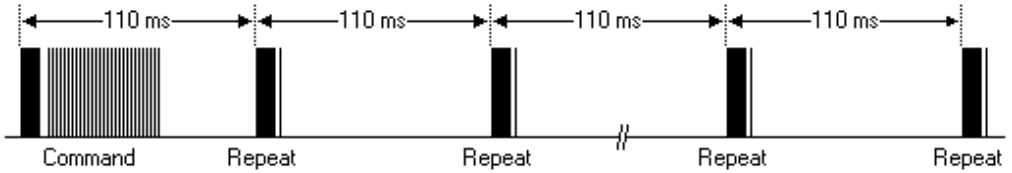
Menos teoría y más práctica.

Modulación a 38kHz

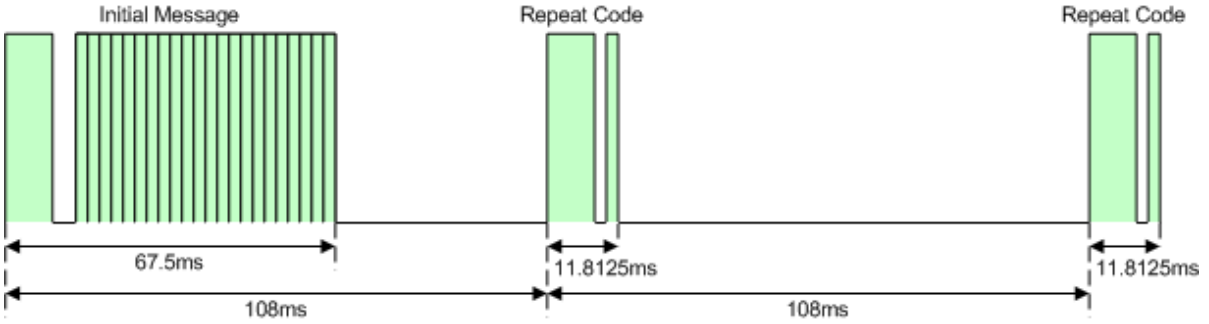


www.sbprojects.com/knowledge/ir/nec.php

Trama completa con repeticiones

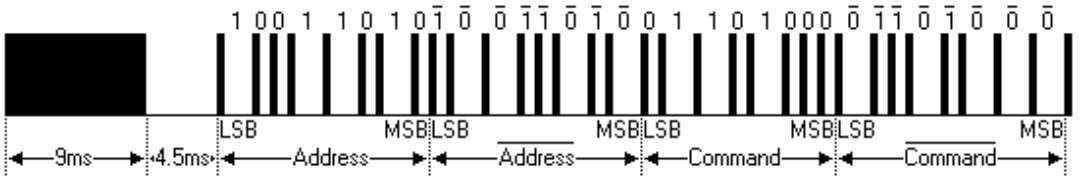


www.sbprojects.com/knowledge/ir/nec.php

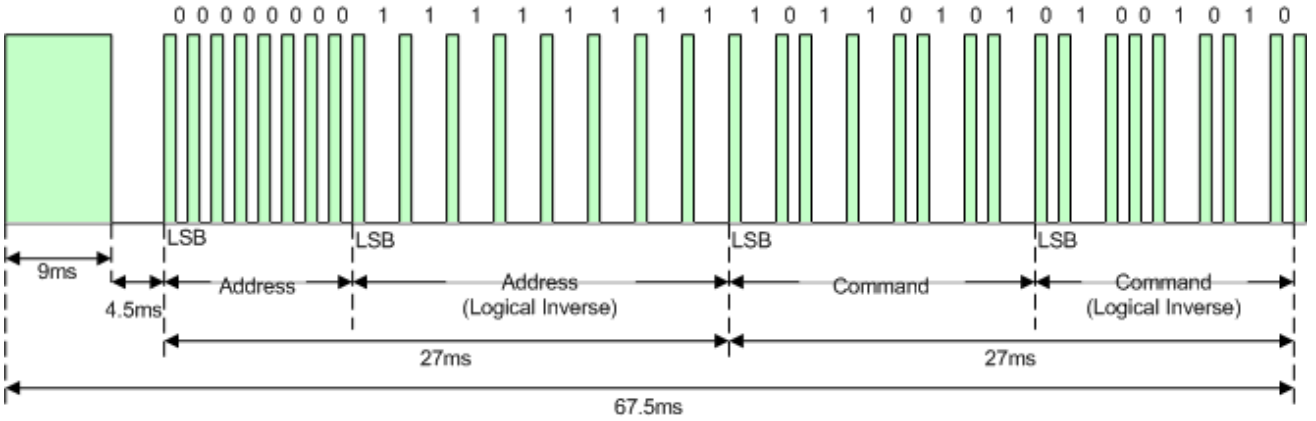


techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol

Inicio de transmisión y datos

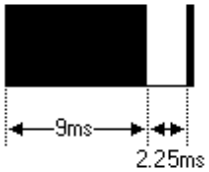


www.sbprojects.com/knowledge/ir/nec.php



techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol

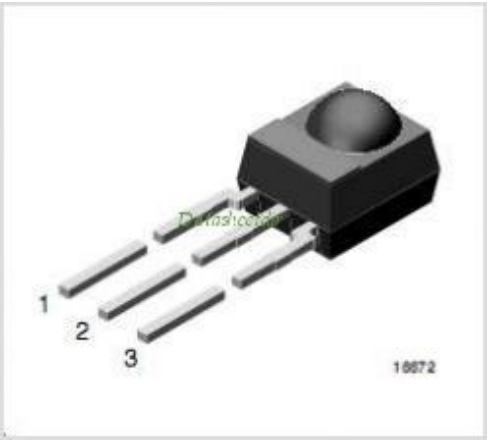
Repetición



560µs
562.5µs

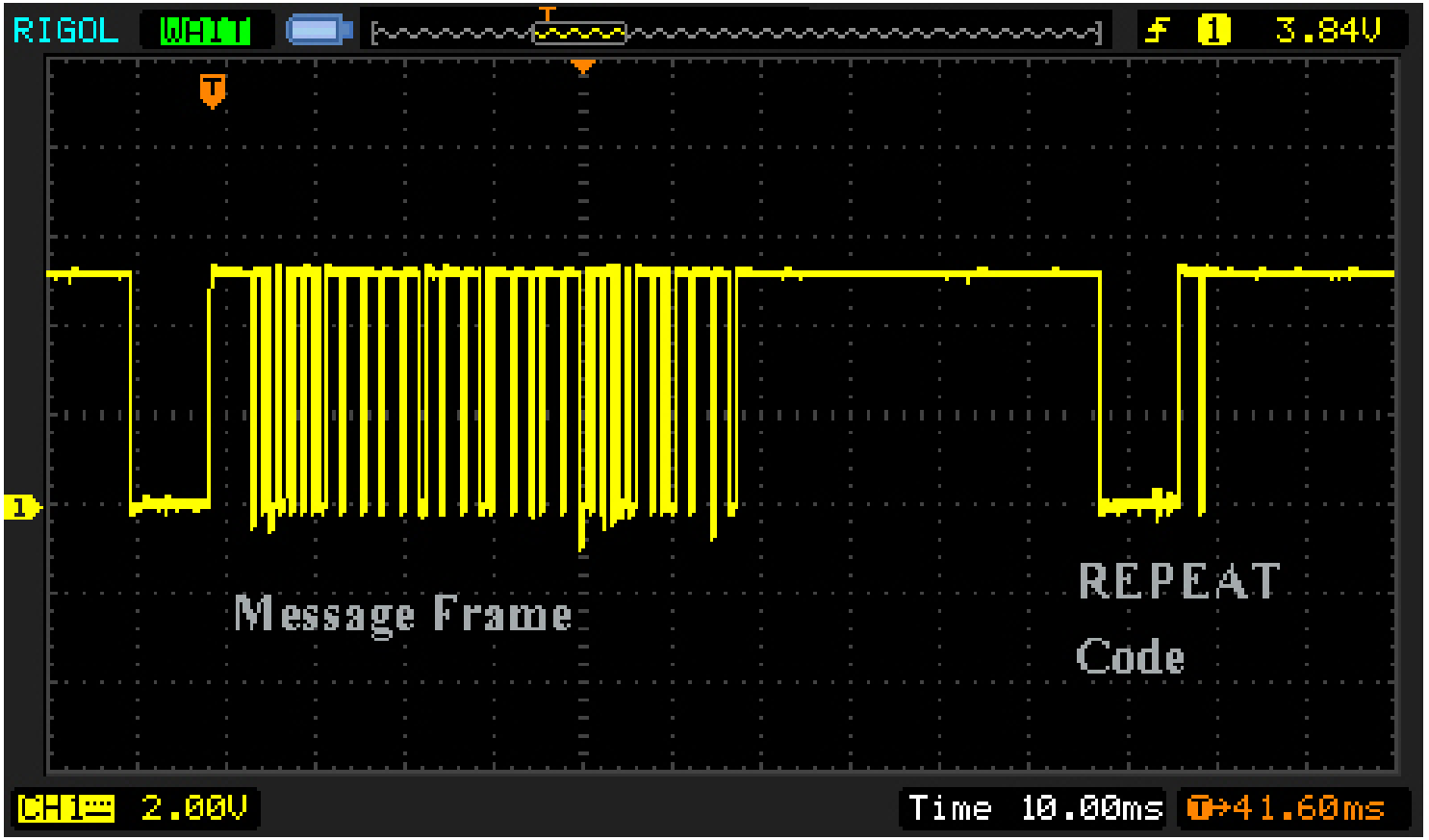
www.sbprojects.com/knowledge/ir/nec.php

Nuestro amigo el TSOP4138 (y similares)



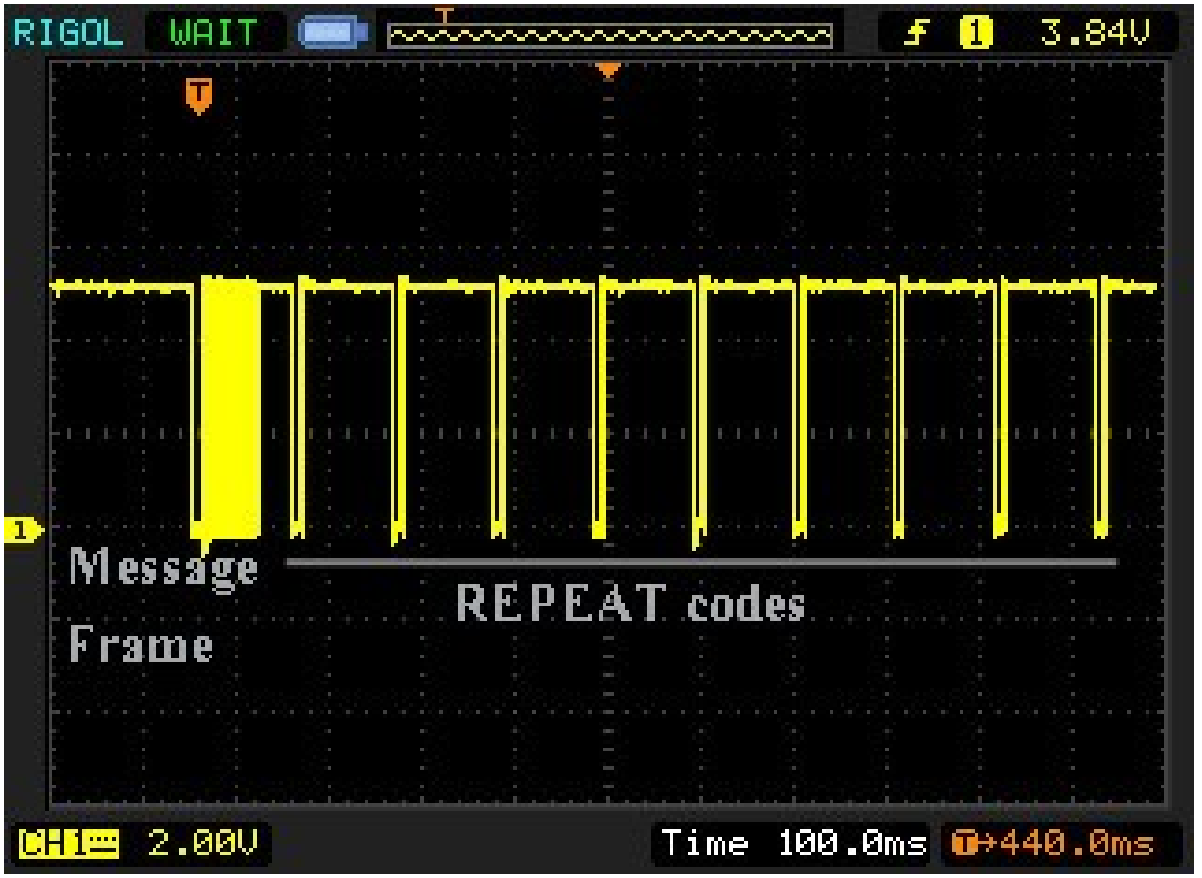
1 = OUT, 2 = GND, 3 = V_S

El TSOP4138, o similar, trabajando



www.circuitvalley.com/2013/09/nec-protocol-ir-infrared-remote-control.html

El TSOP4138, o similar, trabajando



www.circuitvalley.com/2013/09/nec-protocol-ir-infrared-remote-control.html

Software

Muestreo
vs
Interrupción externa.

Muestreo

- El muestreo precisa del uso y configuración muy específica de un temporizador.
- Ejecución de código y consumo de CPU cuando no es necesario.

Interrupción externa

- No «gasta» ningún timer de forma exclusiva ya que no exige una «alta precisión».
- Aunque se necesita de un timer para controlar el tiempo transcurrido (microsegundos), para ello se puede aprovechar uno cualquiera que se esté usando para otra cosa.
- La interrupción asociada al pin de entrada se ejecuta sólo si hay un cambio, haciéndolo más eficiente.

Implementando interrupción externa

- Para la implementación se ha usado el timer «original» del entorno de Arduino, mediante la función micros().
- El control se realiza usando una «máquina de estados» con la que se controlan las transiciones entre la ausencia y existencia de portadora, así como el tiempo transcurrido entre cada transición.

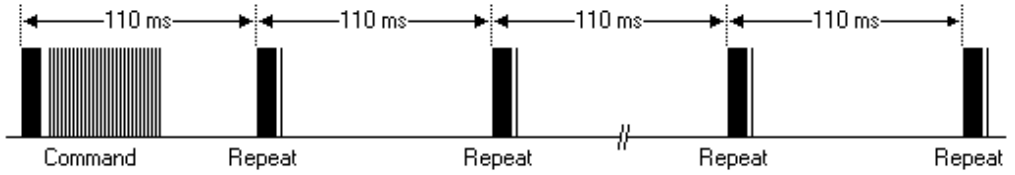
- La máquina de estado va verificando los tiempos y obteniendo los datos.
- Especial atención en controlar el inicio y final de la trama de datos y de repetición.
- Si hay algún error, verificamos si estamos ante un posible inicio de trama para tratarlo o se vuelve al estado de «reposo».
- Una vez completada la trama de datos, se verifica si los complementarios son correctos. (Existe una versión extendida del protocolo).
- Si se reciben correctamente los datos, pasa a esperar por las repeticiones.

Las repeticiones

- Las repeticiones se dan por terminadas si pasa demasiado tiempo entre una y otra (bloqueo de la señal) o si se detecta un inicio de trama (nueva tecla pulsada).
- La tecla pulsada, inicialmente siempre se guarda «sola» y con cero repeticiones.
- Las posteriores repeticiones se guardan aparte, como si de una nueva tecla se tratase, pero con al menos una repetición.

- Si no se consumieran las pulsaciones iniciales o las repeticiones, estas son almacenadas en un pequeño buffer de 6 elementos.
- El comportamiento del buffer se puede configurar para si se descartan las nuevas pulsaciones o si se descartan las antiguas.
- Las repeticiones, mientras las anteriores no han sido consumidas, no se añaden al buffer, sino que incrementa el número de repeticiones de la última tecla almacenada en el buffer.
- Siempre que se quieran usar las repeticiones, se ha de tener en cuenta si sumar su valor porque puede ser mayor que uno.

- Cosa importante si se va a tener en cuenta el número de repeticiones: la pulsación de cualquier tecla devolverá un primer resultado con las repeticiones a cero. Así que tal vez habría que «sumar uno» en este caso.
- La sensibilidad de la repetición inicial se puede ajustar en tiempo de ejecución. Por defecto es «no sensible».



www.sbprojects.com/knowledge/ir/nec.php

Experiencias

- No todos los mandos funcionan igual (diferentes estándares y «no estándares»).
- Los tiempos de las señales no siempre son lo que deberían de ser, así que el «ajuste fino» (a ojo) siempre es una posible opción.
- Interferencias en la transmisión.

Pulsador vs receptor IR

- Un pin por tecla (salvo ingenios sofisticados) vs un pin para muchas teclas.
- Inmeditatez (y posibles rebotes) vs retardo en el tiempo de reacción.
- El pulsador siempre está ahí vs el mando no se sabe dónde estará.
- Sin interferencias vs posibles interferencias.
- Levántate a pulsarlo vs lo hago desde el sofá.
- No cool vs lo que voy a fardar con el mando.

Pulsador

```
const int buttonPin = 12;
const int ledPin = 11;

int buttonState = 0;

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop(){
    buttonState = digitalRead(buttonPin);
    if (buttonState == HIGH) {
        // turn LED on:
        digitalWrite(ledPin, HIGH);
    }
    else {
        // turn LED off:
        digitalWrite(ledPin, LOW);
    }
}
```

Receptor IR

```
#include "ReceptorIR.H"
#include "codigos_mandos.h"

const int receptorPin = 2;   // el número del pin al que está conectado el receptor
const int receptorIRQ = 0;   // La interrupción 0 del Arduino UNO está asociada al pin 2
const int ledPin =  11;

ReceptorIR::data_t teclaPulsada;    // valor de la tecla pulsada

void setup() {
    pinMode(ledPin, OUTPUT);
    ReceptorIR::begin(receptorIRQ, receptorPin); // Inicializamos el receptor de IR
}

void loop(){
    if (ReceptorIR::next()) { // XXX IMPORTANTE XXX Esta condición ha de estar sola. Hemos de asegurar que se llama sí o sí.
        if (!ReceptorIR::getRepeats()) {
            teclaPulsada = ReceptorIR::getData();

            if (teclaPulsada == HITACHI_RB6_REMOTE_KEY__NEXT) {
                // turn LED on:
                digitalWrite(ledPin, HIGH);
            }
            else if (teclaPulsada == HITACHI_RB6_REMOTE_KEY__PREV) {
                // turn LED off:
                digitalWrite(ledPin, LOW);
            }
        }
    }
}
```

Pulsador vs receptor IR

```
const int buttonPin = 12;

const int ledPin = 11;

int buttonState = 0;

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop(){


    buttonState = digitalRead(buttonPin);

    if (buttonState == HIGH) {
        // turn LED on:
        digitalWrite(ledPin, HIGH);
    }
    else {
        // turn LED off:
        digitalWrite(ledPin, LOW);
    }
}
```

```
#include "ReceptorIR.H"
#include "codigos_mandos.h"

const int receptorPin = 2;   // el número del pin al que está conectado el receptor
const int receptorIRQ = 0;   // La interrupción 0 del Arduino UNO está asociada al pin 2
const int ledPin =  11;

ReceptorIR::data_t teclaPulsada;    // valor de la tecla pulsada

void setup() {
    pinMode(ledPin, OUTPUT);
    ReceptorIR::begin(receptorIRQ, receptorPin); // Inicializamos el receptor de IR
}

void loop(){
    if (ReceptorIR::next()) { // XXX IMPORTANTE XXX Esta condición ha de estar sola.
        if (!ReceptorIR::getRepeats()) {
            teclaPulsada = ReceptorIR::getData();

            if (teclaPulsada == HITACHI_RB6_REMOTE_KEY__NEXT) {
                // turn LED on:
                digitalWrite(ledPin, HIGH);
            }
            else if (teclaPulsada == HITACHI_RB6_REMOTE_KEY__PREV) {
                // turn LED off:
                digitalWrite(ledPin, LOW);
            }
        }
    }
}
```

Nota

ReceptorIR está definida como una clase estática, con todos sus atributos y funciones-miembro estáticos, por lo que se han de referenciar usando doble «dos puntos»

ReceptorIR::

en lugar de un punto

ReceptorIR.

Tipos:

```
typedef unsigned long data_t;       // Tipo que retorna ReceptorIR::getData()
typedef unsigned int repeats_t;     // Tipo que retorna ReceptorIR::getRepeats()
```

Métodos:

```
static void begin(int8_t irqNumber, int8_t pinNumber); // Inicializa el receptor (indicar interrupción asociada y pin).

static bool next();                                    // Retorna true si hay una nueva tecla pulsada.

static ReceptorIR::data_t getData();                   // Obtiene el valor de la tecla pulsada.

static ReceptorIR::repeats_t getRepeats();             // Obtiene en número de veces que se ha repetido la pulsación.
                                                       // siempre es 0 para la primera pulsación.

static void enable();                                  // Habilita la recepción.

static void disable();                                 // Deshabilita la recepción.

static void setSensitive(bool sensitive);              // Hace que sea más o menos sensible a la primera repetición, descartándola o no.
                                                       // Por defecto es 'no sensible' y la descarta.

static void setAlwaysPush(bool alwaysPush = true);     // Si el buffer se llena, establece si lo sobreescribe (true) o no (false).
                                                       // Por defecto lo sobreescribe (true).
```

Interrupción asignada a cada pin

Según sea el modelo de Arduino,
la interrupción asignada a cada pin varía.

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	
Due			(see below)			

En el caso del Due el número del pin
y el número de la interrución coinciden.

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

Fin

Ahora sí, ahora a ponerlo en práctica.