

# Activación de Carga por Infrarrojo

Walter Hernández Angel

20 de agosto de 2003

## 1. Introducción

En esencia se construye un receptor de infrarrojos que capta y decodifica la señal enviada por un mando. El mando puede ser un control remoto de cualquier aparato (tv, video, sonido, etc.) y de cualquier marca, pero en este trabajo el hardware se diseña para ocupar un mando marca Sony de una televisión, sin que en ello exista una razón técnica, salvo el precio de éste.

La etapa decodificadora está compuesta por un microcontrolador, el cual toma la señal del receptor de infrarrojo. Dicha señal, una vez capturada e identificada por el micro, es tratada por éste para llevar a cabo una determinada función dependiendo del botón pulsado en el mando Sony.

El hardware aquí diseñado, es usado para controlar una ampollita incandescente. Específicamente, el dispositivo se construyó para variar la intensidad de luz, (al igual que un dimmer) encenderla y apagarla.

Es obvio (para los entendidos) que la carga puede ser desde un o unos simples led, hasta un motor, todo depende de la etapa de potencia que se agregue a la etapa de control.

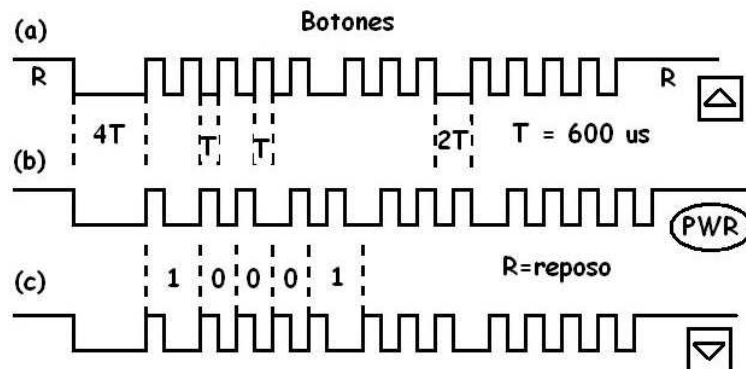


Figura 1: diagrama de tiempos

## 2. Emisor IR

Primeramente, debemos saber lo que tenemos a la salida del control remoto, que como dije es un Sony de tv. Por ello, lo mejor tanto para el lector como para mí, es que visites la dirección <http://www.fortunecity.es/arcoiris/tarot/572/index.html> donde encontrarás información general, pero valiosa, sobre como obtener la señal de salida de un control remoto. No obstante, la Figura 1 es un diagrama obtenido con un osciloscopio (generosamente prestado, muchas gracias) conectado al pin de salida del integrado IRM8601S. Mas adelante se encuentra la lista completa de componentes para este sistema embebido.

De la Figura 1 se tiene: (a) botón de canal ascendente, (b) botón de power PWR y en (c) el botón de canal descendente.

Y la función que cumplen estos botones en el sistema embebido:

- (a) aumentar intensidad lumínica
- (b) encender o apagar la ampollita
- (c) disminuir intensidad lumínica

A la salida del Receptor de Infrarrojos tenemos un estado ALTO representado por R=reposo, cuando recibe señal a través del mando cambia a estado BAJO con una duración de  $4T=2400 \mu s$ , después viene la trama de unos y ceros con un total de ocho bits, que es la que determina que tecla se pulsa en el mando. Esta trama es la que nos interesa programar en el microcontrolador.

Los últimos cuatro bits que son todos ceros no se toman en cuenta en la programación, esto se debe a que todos los botones pulsados en el mando terminan de igual forma. La ventaja o desventaja de este enfoque es que el control de la carga podrá ser activado tanto con un mando de tv como con uno de video mientras sea de SONY.

Por último, en cuanto al tramo de 4T al principio del diagrama, éste nos sirve para saber en que momento exacto se ha pulsado un botón en el mando, y así estar atento a capturar la trama que identifica la tecla pulsada.

### 3. Receptor IR

Para recibir la señal IR enviada por el control remoto, existen varios dispositivos, tales como el TSOP1736, IRM8601S, gp1u52x, etc. En particular aca usamos el IRM8601S.

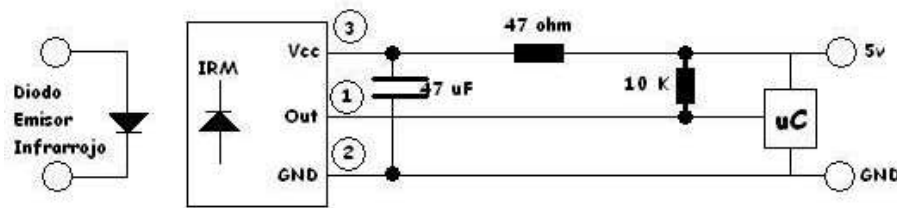


Figura 2: conexión típica

En la Figura 2 se muestra la forma de conectar el dispositivo al microcontrolador. Refiérase al Datasheet para más detalles.

### 4. Decodificador

Un microcontrolador, el PIC 16f84, es el encargado de identificar el botón pulsado en el mando Sony. Para esto, el pin PA1 del puerto A del micro se conectó al pin de salida del IRM8601S, es decir, la comunicación entre ambos dispositivos es serial "sólo por si acaso". El programa residente en el PIC es el siguiente:

```
LIST          p=16f84
```

```
#include    <p16f84a.inc>

TMR_OPT    EQU    0x01
PCL        EQU    0x02
EST        EQU    0x03
PA         EQU    0x05
PB         EQU    0x06
INT        EQU    0x0B
DATOINFRA  EQU    0x0C
MAX        EQU    0x0D
PWR        EQU    0x0E
RETARDO    EQU    0x0F
BYTE       EQU    0x10
AUX        EQU    0x11

ORG        0
goto      inicio
ORG        4
goto      inter
ORG        5

inicio     bsf     EST,5
           movlw  0xC3
           movwf  TMR_OPT
           movlw  0x02
           movwf  PA
           clrf   PB
           bcf   EST,5
           clrf   PA
           bsf   PA,0
           clrf   PB
           movlw  0x95
           movwf  PWR
           movlw  0x1F
           movwf  MAX
           movlw  0x55
           movwf  RETARDO
```

```
comienzo  movlw  0x90
          movwf  AUX
          movlw  0x08
          movwf  BYTE
          clrf   DATOINFRA
          clrf   INT

exp_sen   btfsc  PA,1
          goto  exp_sen
          clrf  INT
          movlw 0x58
          movwf TMR_OPT

inic_band btfss  INT,2
          goto  inic_band
          btfss PA,1
          goto  comienzo
          movlw 0xB5
          movwf TMR_OPT
          movlw 0xA0
          movwf INT
          movlw 0xB5

bucle     goto  bucle

inter     movwf  TMR_OPT
          btfss PA,1
          goto  es_cero
          bcf   INT,2
          bcf   EST,0
          rrf   DATOINFRA,1
          decfsz BYTE,1
          retfie
          goto  comparar

es_cero   movlw  0x90
          movwf  TMR_OPT
          bsf   EST,0
```

```
        rrf    DATOINFRA,1
        movlw  0xB5
        decfsz BYTE,1
        goto   vuelve

comparar  movf   DATOINFRA,0
          xorwf  PWR,0
          btfsc  EST,2
          goto   on_off
          movf   DATOINFRA,0
          xorwf  AUX,0
          btfsc  EST,2
          goto   up_pot
          incf   AUX,1
          movf   DATOINFRA,0
          xorwf  AUX,0
          btfsc  EST,2
          goto   down_pot
          goto   comienzo

vuelve   bcf    INT,2
          retfie

on_off   btfss  PA,0
          goto   encender
          bcf    PA,0
          call   ret350ms
          goto   comienzo

encender bsf    PA,0
          call   ret350ms
          goto   comienzo

up_pot   clrw
          xorwf  PB,0
          btfss  EST,2
          decf   PB,1
          call   ret350ms
```

```
                goto    comienzo

down_pot  movf    MAX,0
          xorwf  PB,0
          btfss  EST,2
          incf  PB,1
          call  ret350ms
          goto  comienzo

ret350ms  clrf    TMR_OPT
          bcf    INT,2
esp4ms   btfss  INT,2
          goto  esp4ms
          bcf    INT,2
          decfsz RETARDO,1
          goto  esp4ms
          movlw  0x55
          movwf  RETARDO
          return
          END
```

No se harán comentarios explicativos del código presentado; pues la Idea no es dar clases de programación, sí se destaca, que mediante el programa, este PIC no es el encargado de provocar el disparo para la etapa de potencia, puesto que para ello se usa un segundo PIC.

La necesidad de usar dos PIC 16f84 radica en que el primero, (el decodificador) usa el TMR0 para sincronizar e identificar el tren de pulsos provenientes del pin de salida del IRM8601S, y el segundo PIC (el disparador) usa el TMR0 para variar el ángulo de disparo del triac de potencia conectado a la carga.

Resumiendo; si sólo queremos encender o apagar la ampolleta, nos basta con un solo PIC, pero como la idea es también variar la intensidad lumínica, se necesitan dos PIC. Con esto no quiero decir que necesariamente deban usarse dos PIC para resolver el problema, pues talvés alguien que este leyendo esto, pueda hacerlo con uno solo.

## 5. Disparador

Esta compuesto por el segundo PIC 16f84 el cual recibe la información del decodificador en forma paralela y es el encargado de excitar al fototriac, el MOC3021, para que este sea finalmente el que envía la señal de disparo a la puerta o gate del triac BTA 08-600B conectado a la carga. El código del disparador a continuación.

```
LIST      P=16F84
#include  <p16f84a.inc>

TMR_OPT  EQU   0x01
EST       EQU   0x03
PA        EQU   0x05
PB        EQU   0x06
INT       EQU   0x0B
CONT      EQU   0x0C

ORG      0
goto     inicio
ORG      4
goto     inter
ORG      5

inicio    bsf     EST,5
          movlw   0xC2
          movwf   TMR_OPT
          movlw   0x09
          movwf   PB
          movlw   0xFF
          movwf   PA
          bcf     EST,5
          clrf    PA
          clrf    PB
off       movlw   0x90
          movwf   INT

bucle     goto    bucle
```

```
inter    btfss   PB,3
         goto    off
         nop
         movf   PA,0
         movwf  CONT
         incf   CONT,1
         call   disparo
         retfie

disparo  movlw   0xDD
         movwf  TMR_OPT
         bcf    INT,2
esp288us btfss   INT,2
         goto   esp288us
         bcf    INT,2
         decfsz CONT,1
         goto   disparo
         movlw  0xFD
         movwf  TMR_OPT
         bsf    PB,2
esp48us  btfss   INT,2
         goto   esp48us
         bcf    PB,2
         bcf    INT,1
         return
         END
```

## 6. Listado de Componentes

Cantidad	Descripción
6	Resistencia 10K 1/4w
2	Resistencia 330 1/4w
1	Resistencia 15K 1/4w
1	Resistencia 3,3K 1/4w
1	Resistencia 56K 1/4w
1	Resistencia 180 1/4w
1	Resistencia 39 1/4w
2	Diodos 1N4007
1	Condensador 4,7 uF 25v
1	Condensador 0,01 uF 1Kv
2	PIC16F84
1	LM324
1	MOC3021
1	BTA08-600B
1	trafo tap central 12v

## 7. Esquemáticos

En esta sección se muestran los circuitos esquemáticos por partes, el listado anterior dicta todos los componentes salvo la fuente de poder cc de 5v y los componentes asociados a la conexión típica de los PIC como por ejemplo los cristales de 4 MHz, condensadores de 22pF, reset, etc. El conexionado de trabajo del PIC se puede encontrar en los Datasheet o en Internet. En la Figura 4 de arriba hacia abajo el decodificador y el disparador respectivamente.

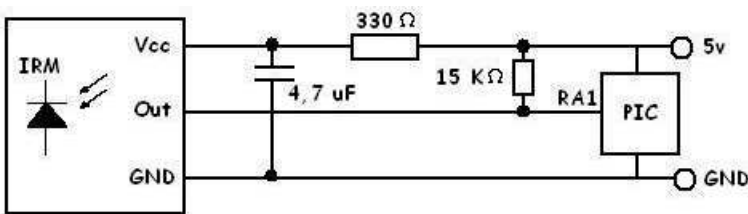


Figura 3: receptor de infrarrojos

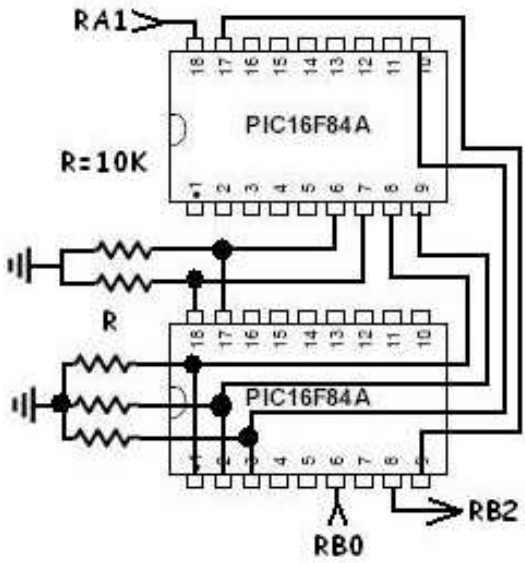


Figura 4: pic decodificador y pic disparador

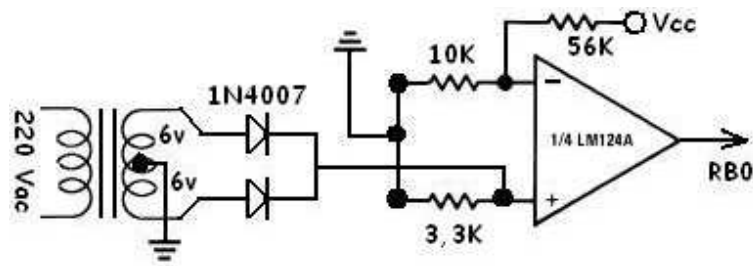


Figura 5: detector de cruce por cero con LM324

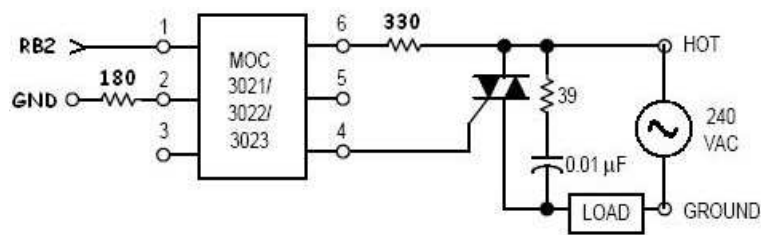


Figura 6: etapa de potencia para activar la carga

## 8. Conclusión

El lenguaje de programación usado es ensamblador. Fue elegido para tener un control exacto en los tiempos de sincronismo con la señal infrarroja. De igual manera puede abordarse la programación en C, teniendo claro que al momento de compilar el programa, puede resultar en una mayor cantidad de memoria de código y que se vea afectada la sincronía con la señal. La ventaja de programar en ensamblador además del sincronismo, es que si la aplicación demanda rapidez y eficiencia en memoria, éste lenguaje es el adecuado. A modo de información extra, el código fue escrito usando el entorno de programación MPLAB de Microchip y compilado con el MPASM, en todo caso se puede escribir en el notepad o block de notas de windows y después compilarlo usando el MPASM.

Para que el hardware funcione solamente con un tipo de control, es necesario que el programa decodificador sea capaz de capturar la trama completa, desde 4T hasta los últimos 4 bits que son todos ceros como se mostró en el diagrama de tiempos. Para ello, basta con agregar algunas subrutinas mas, pero nada complicado.

La conexión ac para la carga está dada por la red eléctrica chilena, es decir, 50Hz y 220v, datos que son básicos para el cálculo de potencia entregada a la ampolleta en función del tiempo de disparo en el triac. Ya que, como se mencionó anteriormente, se varía el ángulo de disparo para el triac. Si no quisiéramos utilizarlo como dimmer, entonces, se puede reemplazar el MOC3021 por un MOC3041 el cual tiene un detector de cruce por cero incorporado.

El hardware y el software abierto ya es una realidad apoyada por muchos. Reconociendo que la comunidad de programadores se adelantó por mucho a la comunidad de electrónicos, (pero de atrás pica el Indio). La Internet nos da la oportunidad de poder trabajar juntos para crear, reinventar, mejorar o simplemente desafiar, el poder está en nuestra mente.

La piratería solamente hace nada. . .