INTRODUCTION

This device allows for the control of small appliances, even garage doors from the convenience of your easy chair, using an existing TV remote control.

APPLICATION OPERATION

A Sony TV remote control signal is coupled to a PIC12C508 through an infrared detector module. Decoding of the waveform is performed in software to provide an on/off signal for controlling small lamps and appliances and a pulsed output for controlling momentary-type interfaces, such as electric garage door controls, etc.

This particular application decodes a Sony data stream, but could just have easily been designed to respond to other remote commands using similar techniques. Most universal remote controls default to the Sony mode on initial power up however, so it seemed a good choice.

In addition to providing one continuous output and a 0.25 sec pulsed output, my design provides a (visual feedback) 'valid signal received' output for driving an LED. I found it convenient to have this feedback, although it is not essential to the design. One output pin on the PICmicro™ remains uncommitted and can be easily programmed for an additional function.
The listing that follows contains a description of the connections to the I.C.

**BILL OF MATERIALS (BOM)**

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Manufacture</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C508</td>
<td>Microchip</td>
</tr>
<tr>
<td>IR Detector Module (~40KHz)</td>
<td>Various</td>
</tr>
<tr>
<td>78L05 regulator</td>
<td>Various</td>
</tr>
<tr>
<td>12 VDC relay</td>
<td>Various</td>
</tr>
<tr>
<td>2N3904 transistor</td>
<td></td>
</tr>
<tr>
<td>1N4001 diode, LED, 2 - 0.1 μF capacitors, 2.2K, 4.7K, and 680Ω resistors, 12VDC supply</td>
<td>---</td>
</tr>
</tbody>
</table>
SOFTWARE LISTING:

; IR Remote
; ----------
;
; An infra-red receiver circuit for controlling various appliances.
; This circuit continuously monitors the output of a standard infra-red
; receiver module for Sony command strings. Upon receipt of the correct
; command, outputs are either turned on, off, or pulsed.
;
; As programmed, this circuit responds to the codes:
; 3 - 3 - 0 turns pin 7 (GP0) off
; 3 - 3 - 1 turns pin 7 (GP0) on
; 3 - 3 - 3 toggles the output on pin 7 (GP0)
; 3 - 3 - 5 pulses pin 2 (GP5) on for 0.25sec
;
; Circuit connections are as follows:
; - +5V is connected to pin 1, gnd to pin 8
; - An active high (continuous) output is on GP0 (pin 7)
; - An active high (pulsed) output is available at GP5 (pin 2)
; - An active high 'valid code received' signal is on GP1 (pin 6)
; - GP3 (pin 4) is configured as an active low MCLR, with internal pullup
; In the present configuration, the output from GP4 (pin 3) is not used.
;
***************************************************************************
Program Equates
ZeroKey EQU 9 ; Sony key codes
OneKey  EQU 0
ThreeKey EQU 2
FiveKey  EQU 4

LED   EQU 1 ; GPIO bit#

; Standard Equates
W     EQU 0
F     EQU 1

GPWUF  EQU 7
PA0    EQU 5
TO     EQU 4
PD     EQU 3
Z      EQU 2
Zero   EQU 2
DC     EQU 1
C      EQU 0
Carry  EQU 0

; fuses
MCLRDisabled EQU 0
MCLREnabled EQU H'10'
CodeProtect  EQU 0
NoCodeProtect EQU H'08'
WDTDisabled  EQU 0
WDTEnabled   EQU H'04'
IntRCOsc     EQU H'02'
ExtRCOsc     EQU H'03'
XTOsc       EQU H'01'
LPOsc       EQU 0

; '508 Registers
INDF   EQU H'00'
TMRO   EQU H'01'
Wireless and Remote Controlled Personal Appliance

PCL EQU H'02'
STATUS EQU H'03'
PSR EQU H'04'
OSCCAL EQU H'05'
GPIO EQU H'06'

; Program variables
Flags EQU H'07' ; standard signals
TFlag EQU 0 ; bit 0 of Flags

Temp EQU H'08' ; scratch pad register
CntrLo EQU H'09' ; Timing Counter (low byte)
CntrMid EQU H'0A' ; **(middle byte)
CntrHi EQU H'0B' ; **(hi byte)

BitCnt EQU H'0C' ; used when receiving a byte
Command EQU H'0D' ; the command byte last received

; *********************************************
; Set the ID words...
ORG H'0200'
ID0 Data.W H'0000'
ID1 Data.W H'0000'
ID2 Data.W H'0004'
ID3 Data.W H'0009'

; *********************************************
; and the Fuse bits...
ORG H'0FFF'
CONFIG Data.W MCLREnabled + NoCodeProtect + WDTDisabled + IntRCOsc

; *********************************************
; PIC starts here on power up...
; *********************************************
ORG H'00'

MOVWF OSCCAL ; store the factory osc. calibration value
GOTO Init ; and jump past the subroutines

; *********************************************
; Subroutines...
; *********************************************

; *********************************************
; Delay
; Waits approx W mSec then returns
Delay CLRF Temp
dl NOP ; 255 * 4 uS loop
DECFSZ Temp,F
GOTO dl
MOVWF Temp ; repeat the loop W times
DECFSZ Temp,W
GOTO Delay
RETLW 0

; *********************************************

© 1998 Microchip Technology Inc.
Wireless and Remote Controlled Personal Appliance

; GetaByte
; Wait up to 2 secs for a 12 bit word. Sony data is sent as 7 bits for command then 5 bits for device code (both LSB first). We only want a command byte, so pad out the 7 bits to 8, and ignore rest.

GetaByte

; reset the counters
CLR Lo

; (but preload the middle byte to get more accurate timing)
MOVWF Mid

; wait for a signal
INC Lo, F

; but no more than 2 sec
MOV W D'3'

; and loop 'til they're all here
DECFSZ Lo, F

; fake an 8th bit
BCF C, 7

; that's always 0

; call header check; see if it's a header
CALL HdrCheck

; and go on if it is
GOTO gb1

; keep waiting
GOTO gb1

; the agc header was OK, now get the command...
MOV W D'7'

; prepare to receive 7 bits
MOVWF BitCnt

; and loop 'til they're all here
GETAB

; fake an 8th bit
BCF C, 7

; as a final test, make sure there were only 12 bits...
CLR Lo

; reset the timers
CLR Mid

; check for no signal for rest of frame
GOTO Main

; if there is any signal, abort
INC Lo, F

; check how long we've waited
MOV Lo D'4'

; (4*256*10uS is approx 10mS)
GOTO gb4

; loop until the 10mS is up
RETLW 0

; *********************************************

; GetaBit
; -> returns with Carry=1 if bit=1, and Carry=0 if bit=0
; Determines the value of the bit currently being sent. Measurements seem to conflict with information obtained on the Sony format, and initial
Wireless and Remote Controlled Personal Appliance

%; attempts at adaptive routines were not reliable enough. I’ve used hard-
%; coded values instead to determine the bit values. Sony bits seem
%; to be sent as 400uS of no carrier followed by either 800uS (‘0’) or
%; 1400uS (‘1’) of carrier (although tolerances are wide).

GetaBit CLRF CntrLo ; reset the counter,
b1 INCF CntrLo,F ; determine no-carrier time
  MOVlw D’100’
  SUBWF CntrLo,W ; allow up to 800uS (100*8uS)
  BTFSC STATUS,Carry
  GOTO Main ; and abort if too long
  BTFSC GPIO,2 ; keep counting ‘til the carrier comes
  GOTO b1

; a space of less than 800uS was received...
  MOVlw D’25’ ; make sure it was greater than
  SUBWF CntrLo,W ; 25*8uS = 200uS
  BTFSS STATUS,Carry
  GOTO Main ; abort if it was too short

; determine the length of the IR pulse being received...
  CLRF CntrLo
  b2 INCF CntrLo,F ; count this pass
  MOVlw D’225’
  SUBWF CntrLo,W ; allow up to 1800uS (225*8uS)
  BTFSC STATUS,Carry
  GOTO Main ; and abort if too long
  BTFSS GPIO,2 ; keep counting ‘til the pulse ends
  GOTO b2

; signal is gone. Was it long enough?
  MOVlw D’50’ ; make sure it was greater than
  SUBWF CntrLo,W ; 50*8uS = 400uS
  BTFSS STATUS,Carry
  GOTO Main ; if not, abort

; and was it a 1 or a 0 ?
  MOVlw D’125’ ; compare the width to 1000uS
  SUBWF CntrLo,W ; which sets/resets the carry
  RETlw 0

; *********************************************
; HdrCheck
; -> returns with Carry=1 if it is a header, and 0 if it is not
; Called with the input low (carrier being received).
; Checks to see if this is a valid agc header.

HdrCheck
  CLRF Temp ; look for the ~2.5mS long header...
hc1 INCF Temp,F ; count each pass thru the loop
  BSF GPIO,2
  MOVlw B’00001000’ ; Pulse GP2 to reset the Schmitt trigger
  TRIS GPIO ; (input drifts due to ambient light)
  MOVlw B’00001100’ ;
  TRIS GPIO
  MOVlw D’250’ ; check for too long a burst
  SUBWF Temp,W ; (250*13uS ~ 3.25mS)
  BTFSC STATUS,Carry
  GOTO hc2 ; abort if the header is too long
  BTFSS GPIO,2 ; check if the signal is still there
  GOTO hc1 ; and if so, keep looping

; a pulse of less than 3mS duration has been received...
MOVLW D'138' ; make sure it was greater than
SUBWF Temp,W ; 138*13uS ~ 1.8mS
RETLW 0 ; C=1 if T>1.8mS

hc2
CLRW
ADDF W Temp,W ; adding 0 to any number clears C
RETLW 0

; ******************************************************
; KeyIsUp
; Waits for no header for at least 65mS. Auto-repeat causes signals
to be sent about every 45mS, so a 65mS pause should signify that
the key has been released (is up).

KeyIsUp CLRF TMR0 ; reset the timer
BCF Flags,TFlag ; and the timeout flag
ku1
CLR W
BTFSS GPIO,2 ; if there's a signal,
CALL HdrCheck ; see if it's a header
BTFSC STATUS,Carry
GOTO KeyIsUp ; loop if it is
BTFSC TMR0,7
BSF Flags,TFlag ; else flag that timeout is pending
BTFSS Flags,TFlag ; timeout pending?
GOTO ku1 ; no
BTFSC TMR0,7 ; yes - has it rolled over?
GOTO ku1 ; no - keep waiting
BCF GPIO,LED ; make sure the LED is off
RETLW 0

; ******************************************************
; Power On jumps to here...
; ******************************************************

Init CLRF GPIO ; initialize the output port
MOVLW B'00001100' ; GP2 and GP3 are inputs, and
TRIS GPIO ; the others are outputs
CLR WDT ; set up the timers...
MOVLW B'11000111' ; int clock to TMR0, uses /256 prescaler
OPTION ; no pullups, and no wakeup on change

Main CALL KeyIsUp ; make sure no key is pressed
CALL GetaByte ; then get the first control byte
MOVF Command,W
XORL W ThreeKey ; see if it's a 3
BTFSS STATUS,Zero
GOTO Main ; if not, start over
BSF GPIO,LED ; else turn the LED on, and
CALL KeyIsUp ; wait for key release
CALL GetaByte ; get the second control byte
MOVF Command,W
XORL W ThreeKey ; see if it's a 3
BTFSS STATUS,Zero
GOTO Main ; if not, start over
BSF GPIO,LED ; else turn the LED on, and
CALL KeyIsUp ; wait for key release
CALL GetaByte ; get the third control byte
MOVF Command,W
XORL W ZeroKey
BTFSC STATUS,Zero ; and see if it's a 0