Control of a Servo using PIC 16F84 and an Infrared Sensor

One basic application of PIC microcontrollers is their use to control motion based on input from a sensor. This is applicable to many different fields, from manufacturing to aeronautics to robotics. This tutorial will demonstrate the control of a Futaba servo motor using a PIC 16F84 microcontroller and input from a Sharp GP2D02 IR sensor.

MOTIVATION AND AUDIENCE

The focus of this tutorial is to demonstrate a method for receiving input from an GP2D02 IR sensor and translating it into a control signal for a servo motor. This tutorial will teach you:

- ***What a PWM signal is.***
- ***How to write code to control and receive input from a GP2D02 IR sensor.***
- ***How to write code to control a Futaba servo motor.***

To do this, it is assumed that you already:

- ***Have completed "A Fast Track to PIC Programming".***

The rest of the tutorial is presented as follows:

- **Parts List and Sources**
- **Construction**
- **Programming**
- **Final Words**

PARTS LIST AND SOURCES

In order to complete this tutorial you must have the circuit from the tutorial **"A Fast Track to PIC Programming"** (minus the dip switches and resistor LED circuits). The only additional parts you will require are:

TABLE 1

PART DESCRIPTION	VENDOR	PART	PRICE (2003)	QTY
GP2D02 IR Sensor	**Acroname**	R19-IR02	21.00	1
Futaba Servo Motor	**RC Hobby Center**	FUTM0031	21.99	1

This sensor was chosen because of its compactness and the wide range over which it can measure. It is also easily interfaceable with microcontrollers and has good control over ambient noise. The servo chosen is a standard servo, however, any servo that operates off of PWM input will do (timing may vary).

To construct the circuit, you will also need:

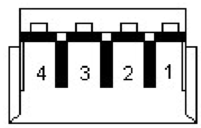
- ***a soldering iron with a fine point***
- ***materials for soldering (solder, flux, etc.)***
- ***small gauge wire***

- ***wire strippers***
- ***multimeter***
- ***DC power supply***

The items listed above can all be purchased from an electronics store such as Radio Shack. Some hardware such as Home Depot carry tools like wire strippers and multimeters.

CONSTRUCTION

The first part of the construction involves preparing the sensor to be hooked up to the PIC. The sensor comes with a connector and four different colored wires. The connector has small numbers on it, and the wires should be placed as follows:



	Pin 1	Pin 2	Pin 3	Pin 4
Sensor Pin	GND	Vin	Vcc	Vout
Wire Color	Black	Green	Red	Yellow

The circuit used to used to communicate with the PIC is the same circuit used from the afore mentioned tutorial with different inputs and outputs. This time input will be coming from the sensor, and output will be going to the sensor to control it and to the servo. To achieve this, the devices should be wired as follows:

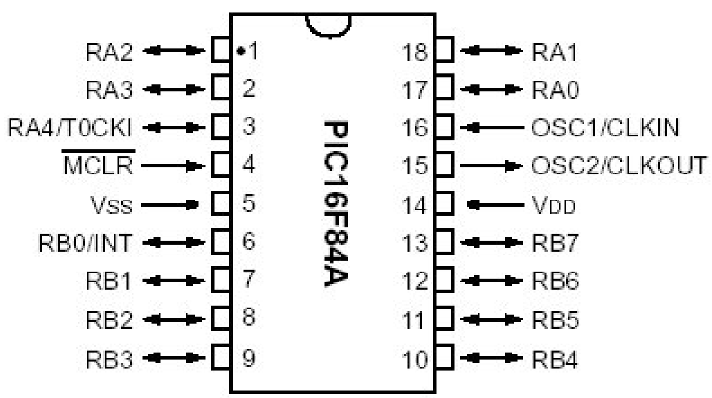


Figure 1

```
Port A1 (Fig 1 - Pin 18) <=WIRED TO=> Vout on sensor    (yellow wire on sensor)
Port B4 (Fig 1 - Pin 10) <=WIRED TO=> Vin on sensor     (green wire on sensor)
Port B0 (Fig 1 - Pin  6) <=WIRED TO=> Command to servo  (white wire on servo)
```

It should be noted that it is necessary to connect a diode (1N4148 or equivalent) inline between the PIC output line and the sensor Vin line, with the cathode (marked with a line) facing the PIC. This diode is provided in the package from Acroname.

This circuit will allow us to receive input from port A of the PIC and send output to port B. The ports were chosen to seperate inputs and outputs and to facilitate the insertion of other sensors. Different ports could be used, however, the code must be changed accordingly.

PROGRAMMING

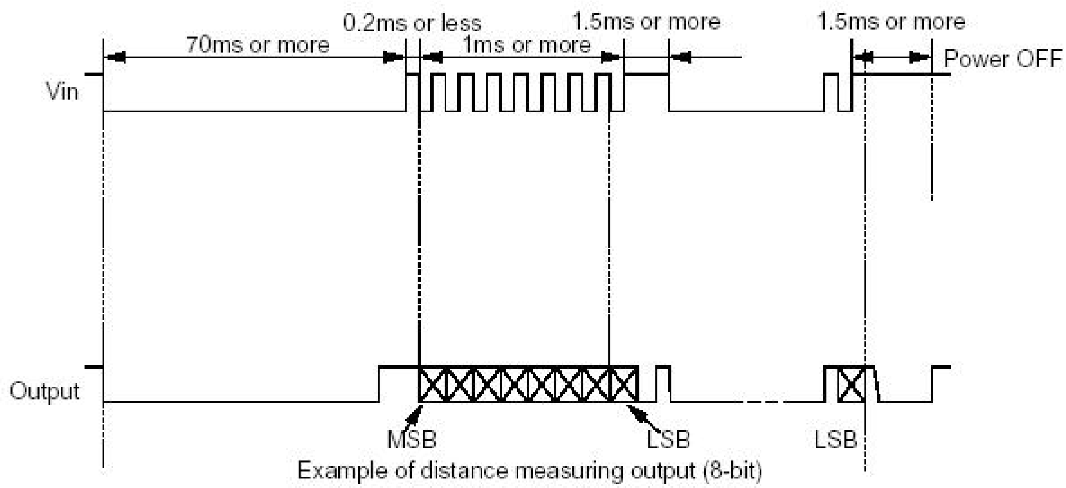


Figure 2

Figure 2 above shows the command signal that must be generated to begin a reading and the resulting output from the sensor. To initialize a reading, the command to the sensor must be held low for a minimum of 70 msec, and then brought high for a minimum of 2 msec. From this point, one of 8 bits is outputed at each falling command edge, starting with the most signifficant bit. After all 8 bits have been outputted, the command must be brought low for 70 msec again to initialize another reading.

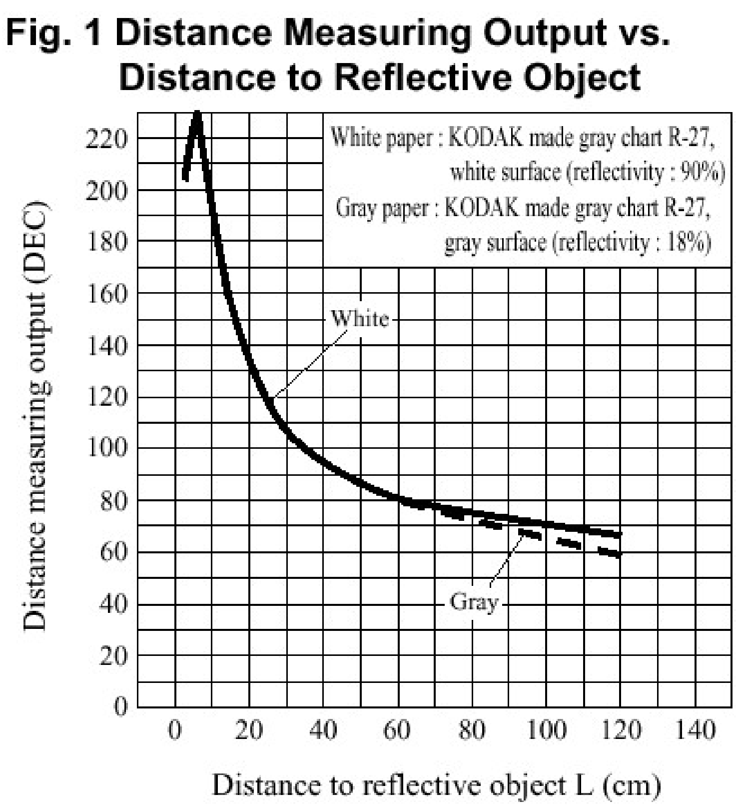


Figure 3

Figure 3 shows the output from the sensor as distance varies. As can be seen, the output is non-linear. If you wish to have a linear response, such as turning a servo motor to a specific point based on distance, you must employ a method to linearize the response. The method used in this tutorial was breaking up the response into 2 linear regions. This will be seen later in the programming.

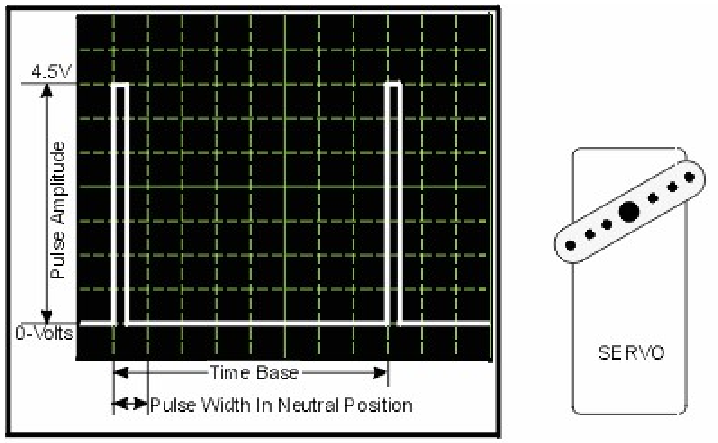


Figure 4

Control of the servo is achieved by generating a PWM signal. A PWM signal is simply a pulse of varying length that can be translated into a position requested of the servo. This is illustrated in Figure 4. Generally, the length of the pulse for a servo varies between 1 msec and 2 msec over a 20 msec period.

The following code requests a reading from the sensor, receives the reading, and transforms the reading into a signal that is outputted to the servo.

IRserv.asm

```
; FILE: IRserv.asm
; AUTH: Keith Sevcik
; DATE: 1/27/03
; DESC:
; NOTE: Tested on PIC16F84-04/P


;------------------------------------------------------------------------
;       cpu equates (memory map)

        list    p=16f84
        radix   hex

;------------------------------------------------------------------------

status  equ     0x03            ; status equate
porta   equ     0x05            ; port a equate
portb   equ     0x06            ; port b equate
IRout   equ     0x11            ; IR output
PWM     equ     0x0c            ; PWM signal length
count   equ     0x0d            ; general register
temp    equ     0x0e            ; general register
loop    equ     0x0f            ; general register
bits    equ     0x10            ; number of bits to read
addpwm  equ     0x12            ; general register

;-----------------------------------------------------------------------

c       equ     0               ; status bit to check after subtraction

;-----------------------------------------------------------------------
```

```
        org     0x000

start   movlw   0x00            ; load W with 0x00 make port B output
        tris    portb           ; port B is outputs
        movlw   0xFF            ; load W with 0xFF make port A input
        tris    porta           ; port A is inputs
        movlw   0x00            ; load W with 0x00 to set intial value of B
        movwf   portb           ; set port b outputs to low

;------------------------------------------------------------------------------------------
; Port b4 is control to sensor
; Port a1 is input from sensor
; Port b0 is output to servo

main
        bcf     portb, 4        ; turn on detector
        nop
wait_for_reading
        btfss   porta, 1        ; wait until done
        goto wait_for_reading   ; with measurement
        bsf     portb, 4        ; bring detector high
        clrf    IRout           ; clear old value
        movlw   8               ; set up to clock out data
        movwf   bits
        movlw   4               ; set up to clock out data
        movwf   count
        bcf     status, c       ; ensure carry bit clear for rotates
        nop                     ; clock delay
read_bit
        bcf     portb, 4        ; bring detector low
        nop                     ; clock delay
        call    delay
        nop
        rlf     IRout, f        ; roll out prev. bit
        btfsc   porta, 1        ; check bit on output
        bsf     IRout, 0        ; set if output 1
        bsf     portb, 4        ; bring detector high
        nop                     ; clock delay
        call    delay
        nop
        decfsz  bits, f         ; all 8 bits done?
        goto read_bit

;------------------------------------------------------------------------------------------
; output from the sensor is in IRout.  Now we need to scale it to PWM.
; We must seperate it into 2 linear ranges.

        movf    IRout,w         ; move the output to w
        sublw   d'110'          ; subtract 110 from the output
        btfsc   status,c        ; if the output was under 110, skip to lower
        goto    lower
        movlw   d'230'          ; else store 230 in temp
        movwf   temp
        movf    IRout,w         ; move the output to w
        subwf   temp,f          ; subtract the output from 230 and store the result in temp
        call    divide          ; call the divide routine
        goto    PWMstrt         ; skip to PWMstrt
lower   movlw   d'110'          ; move 110 into temp
        movwf   temp
        movf    IRout,w         ; move the output to w
        subwf   temp,f          ; subtract the output from 110
        call    mult            ; call the mult routine

;------------------------------------------------------------------------------------------
PWMstrt clrf    count
        movf    PWM,w           ; move PWM to w
        sublw   d'220'          ; subtract PWM cycle from 220
```

```
        btfsc   status,c        ; if PWM is greater than 220, skip next instruction
        goto    skip1
        movlw   d'220'          ; set 220 as the upper limit to PWM
        movwf   PWM
skip1   movf    PWM,w           ; move PWM to w
        sublw   d'20'           ; subtract PWM cycle from 20
        btfss   status,c        ; if PWM is greater than 20, skip next instruction
        goto    skip2
        movlw   d'20'           ; set 20 as the lower limit to PWM
        movwf   PWM
skip2   movlw   d'1'            ; set the delay for generating the PWM
        movwf   count
        bsf     portb,0         ; start the PWM pulse
LoopPWM movf    count,w
        movwf   temp
rep     decfsz  temp
        goto    rep
        nop
        nop
        nop
        decfsz  PWM             ; decrement the PWM length
        goto    LoopPWM         ; as long as PWM is greater than 0, loop
        bcf     portb,0         ; when done looping, stop the pulse
        movlw   d'15'           ; set the counter for generating the rest of the PWM signal
        movwf   loop
del15   movlw   d'255'          ; set the delay counter
        movwf   count
        call    delay
        decfsz  loop
        goto    del15
        goto    main

;-----------------------------------------------------------------------

delay   movf    count,w         ; delay loop
        movwf   temp
del     decfsz  temp            ; 3 clock cycles per delay loop
        goto    del
        return

;-----------------------------------------------------------------------

;-----------------------------------------------------------------------

divide  bcf     status, c       ; make sure the carry bit is clear
        movlw   0
        movwf   addpwm          ; initialize addpwm as 0
min     incf    addpwm,f        ; increment addpwm
        movlw   d'2'            ; move 2 to w
        subwf   temp,f          ; subtract 2 from temp
        btfsc   status,c        ; repeat as long as there is a carry
        goto    min
        movlw   d'210'
        movwf   PWM             ; store 210 in PWM
        movf    addpwm,w
        subwf   PWM,f           ; subtract addpwm from 210, inverting the input
        return

;-----------------------------------------------------------------------

;-----------------------------------------------------------------------

mult    bcf     status, c       ; make sure the carry bit is clear
        movlw   0
        movwf   addpwm          ; initialize addpwm as 0
add     movlw   d'3'
        addwf   addpwm,f        ; increment addpwm by 3
```

```
        decfsz  temp            ; decrement temp as long as it is > 0
        goto    add
        movlw   d'150'
        movwf   PWM             ; store 150 in PWM
        movf    addpwm,w
        subwf   PWM,f           ; subtract addpwm from 150, inverting the input
        return

;-----------------------------------------------------------------------


        end

;-----------------------------------------------------------------------
; at burn time, select:
;       memory uprotected
;       watchdog timer disabled
;       standard crystal (4 MHz)
;       power-up timer on
```

HEADER AND EQUATES

The first portion of code is the header and register equates. For more information about the meaning of the header see the previous tutorial.

```
        list    p=16f84
        radix   hex

;-----------------------------------------------------------------------

status  equ     0x03            ; status equate
porta   equ     0x05            ; port a equate
portb   equ     0x06            ; port b equate
IRout   equ     0x11            ; IR output
PWM     equ     0x0c            ; PWM signal length
count   equ     0x0d            ; general register
temp    equ     0x0e            ; general register
loop    equ     0x0f            ; general register
bits    equ     0x10            ; number of bits to read
addpwm  equ     0x12            ; general register

;----------------------------------------------------------------------

c       equ     0               ; status bit to check after subtraction

;----------------------------------------------------------------------
; portb4 is control to sensor
; porta1 is input from sensor
; portb0 is output to servo

        org     0x000
```

The equates of signifficance here are IRout, PWM and bits. The IRout register will be used to store the output from the sensor. The PWM register will be used to store the length of the PWM signal to be generated. The bits register stores the number of bits to be received from the sensor.

INSTRUCTIONS

The next portion of code contains the actual instructions that tell the PIC what to do.

```
start   movlw   0x00            ; load W with 0x00 make port B output
        tris    portb           ; port B is outputs
        movlw   0xFF            ; load W with 0xFF make port A input
```

```
        tris    porta           ; port A is inputs
        movlw   0x00            ; load W with 0x00 to set intial value of B
        movwf   portb           ; set port b outputs to low
```

These lines set up port A as inputs and port B as outputs. All outputs are then set to low.

```
main
        bcf     portb, 4        ; turn on detector
        nop
wait_for_reading
        btfss   porta, 1        ; wait until done
        goto wait_for_reading   ; with measurement
```

The main loop begins by setting the command signal to the sensor low, thereby intializing a reading. The PIC then waits for the sensor to signal that it is done taking a reading by setting the output high.

```
        bsf     portb, 4        ; bring detector high
        clrf    IRout           ; clear old value
        movlw   8               ; set up to clock out data
        movwf   bits
        movlw   4               ; set up to clock out data
        movwf   count
        bcf     status, c       ; ensure carry bit clear for rotates
        nop                     ; clock delay
```

The next bit of code prepares the PIC to receive input from the sensor. The command signal is brought high. The bits register is set to 8 to set the number of bits to read from the sensor. A clock delay of 4 is set and the carry bit of the status register is cleared.

```
read_bit
        bcf     portb, 4        ; bring detector low
        nop                     ; clock delay
        call    delay
        nop
        rlf     IRout, f        ; roll out prev. bit
        btfsc   porta, 1        ; check bit on output
        bsf     IRout, 0        ; set if output 1
        bsf     portb, 4        ; bring detector high
        nop                     ; clock delay
        call    delay
        nop
        decfsz  bits, f         ; all 8 bits done?
        goto read_bit
```

This portion of code actually reads the input from the sensor. The clock is brought low and the previous bit is moved to the left clearing the way for the next bit. A bit is then read from port A, stored in the register IRout and the command is brought high again for a short delay. The number of bits read is decremented, and so long as the bits register is greater than 0, the program loops back to read the next bit.

```
        movf    IRout,w         ; move the output to w
        sublw   d'110'          ; subtract 110 from the output
        btfsc   status,c        ; if the output was under 110, skip to lower
        goto    lower
        movlw   d'230'          ; else store 230 in temp
        movwf   temp
        movf    IRout,w         ; move the output to w
        subwf   temp,f          ; subtract the output from 230 and store the result in temp
        call    divide          ; call the divide routine
        goto    PWMstrt         ; skip to PWMstrt
```

```
lower   movlw   d'110'          ; move 110 into temp
        movwf   temp
        movf    IRout,w         ; move the output to w
        subwf   temp,f          ; subtract the output from 110
        call    mult            ; call the mult routine
```

This code divides the output into 2 linear regions, the cutoff point being a value of 110 from the sensor. If the value outputted is below 110, the program skips to a portion of code that processes lower regions. The value is stored in temp, and the program then proceeds to a subroutine that effectively divides the output. If the value is above 110, the value again is stored in temp, but the program continues to a subroutine that multiplies the input. We'll now go out of order to see what happens in these various subroutines.

```
divide  bcf     status, c       ; make sure the carry bit is clear
        movlw   0
        movwf   addpwm          ; initialize addpwm as 0
min     incf    addpwm,f        ; increment addpwm
        movlw   d'2'            ; move 2 to w
        subwf   temp,f          ; subtract 2 from temp
        btfsc   status,c        ; repeat as long as there is a carry
        goto    min
        movlw   d'210'
        movwf   PWM             ; store 210 in PWM
        movf    addpwm,w
        subwf   PWM,f           ; subtract addpwm from 210, inverting the input
        return
```

The divide subroutine scales down the input for inputed values greater than 110. This portion of the input has a high gain. It is therefore necessary to scale down the input to get it to match the relatively low gain for values less than 110. This is achieved by successively subtracting 2 from the input and subsequently adding 1 to a temporary value. If you noticed, the input is inversely proportional to the output (i.e. the input gets greater the closer the object is). To invert this, the input is subtracted from 210. Therefore, the highest input (the closest object) will be translated into a low PWM value.

```
mult    bcf     status, c       ; make sure the carry bit is clear
        movlw   0
        movwf   addpwm          ; initialize addpwm as 0
add     movlw   d'3'
        addwf   addpwm,f        ; increment addpwm by 3
        decfsz  temp            ; decrement temp as long as it is > 0
        goto    add
        movlw   d'150'
        movwf   PWM             ; store 150 in PWM
        movf    addpwm,w
        subwf   PWM,f           ; subtract addpwm from 150, inverting the input
        return
```

The mult subroutine scales up the input for inputted values less than 110. This portion of the input has a low gain. It must be scaled up to match the high gain of the other region. This is achieved by successively subtracting 1 from the input and subsequently adding 3 to a temporary value. In effect, this multiplies the input by 3. Again, the value must be inverted as it is inversely proportional to distance. Both of these subroutines output a PWM signal length. We now jump back to where we left the code.

```
        movf    PWM,w           ; move PWM to w
        sublw   d'225'          ; subtract PWM cycle from 200 (2 msec)
        btfsc   status,c        ; if PWM is greater than 200 (2 msec), skip next instruction
        goto    skip1
        movlw   d'200'          ; else set the max PWM length to 200
```

```
        movwf   PWM
skip1   movf    PWM,w               ; move PWM to w
        sublw   d'20'               ; subtract PWM cycle from 200 (2 msec)
        btfss   status,c            ; if PWM is greater than 200 (2 msec), skip next instruction
        goto    skip2
        movlw   d'20'               ; else set the min PWM length to 20
        movwf   PWM
```

These lines set a max and min value for the PWM signal to prevent it from damaging the servo. It subtracts a value of 200 and 20 from the PWM signal and tests to see if there wasnt or was a carry, respectively. If the PWM length fails either test, it is set to either the max or min and the program continues.

```
skip2   movlw   d'1'                ; set the delay for generating the PWM
        movwf   count
        bsf     portb,0             ; start the PWM pulse
LoopPWM call    delay
        nop
        nop
        nop
        decfsz  PWM                 ; decrement the PWM length
        goto    LoopPWM             ; as long as PWM is greater than 0, loop
        bcf     portb,0             ; when done looping, stop the pulse
```

This code actually generates the PWM pulse. A delay length is stored in the count register. The output to the sensor is then set high. This brins the program into a loop that decrements the PWM register, delays, and then continues to loop so long as the value of the PWM register is greater than 0. After completing the loop, the output to the servo is brought low again.

```
        movlw   d'15'               ; set the counter for generating the rest of the PWM signal
        movwf   loop
del15   movlw   d'255'              ; set the delay counter
        movwf   count
        call    delay
        decfsz  loop
        goto    del15
        goto    main
```

This final bit of code generates the remainder of the PWM signal. It consists of a delay nested inside a loop to complete the 20 msec period. When the loop has finished, the entire program is repeated.

FINAL WORDS

After completing this tutorial you should be familiar with the GP2D02 infrared sensor, PWM control of a servo and be able to write code for a PIC 16F84 to control a servo based on input from an infrared sensor.

If you have questions about this tutorial you can email me at **Keithicus@drexel.edu**.