Control of a Servo using PIC 16F84 and an Ultrasonic 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 Devantech SRF04 ultrasonic sensor.

MOTIVATION AND AUDIENCE

The focus of this tutorial is to demonstrate a method for receiving input from an SRF04 ultrasonic 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 SRF04 ultrasonic 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

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
SRF04 Ultrasonic Sensor	**Acroname**	R93-SRF04	33.00	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. 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 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 wire as follows:

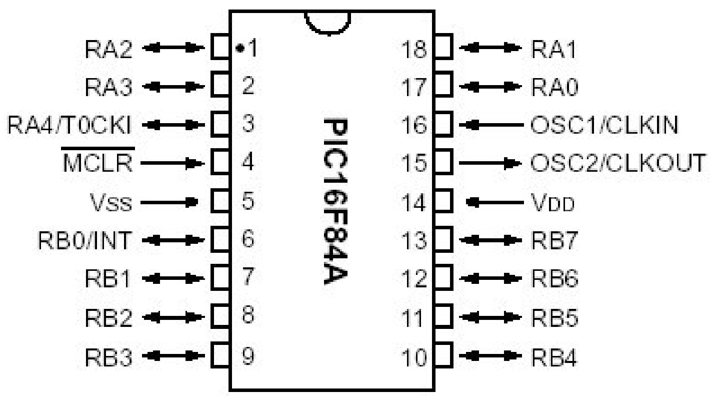

Figure 1

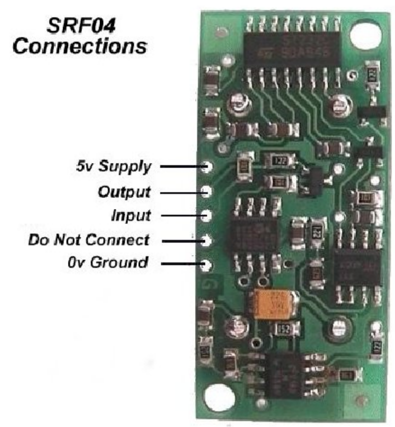

Figure 2

```
Port A0 (Fig 1 - Pin 17) <=WIRED TO=> Output from sensor (Fig 2 - Output)
Port B3 (Fig 1 - Pin  9) <=WIRED TO=> Input to sensor    (Fig 2 - Input)
Port B0 (Fig 1 - Pin  6) <=WIRED TO=> Command to servo   (white wire)
```

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

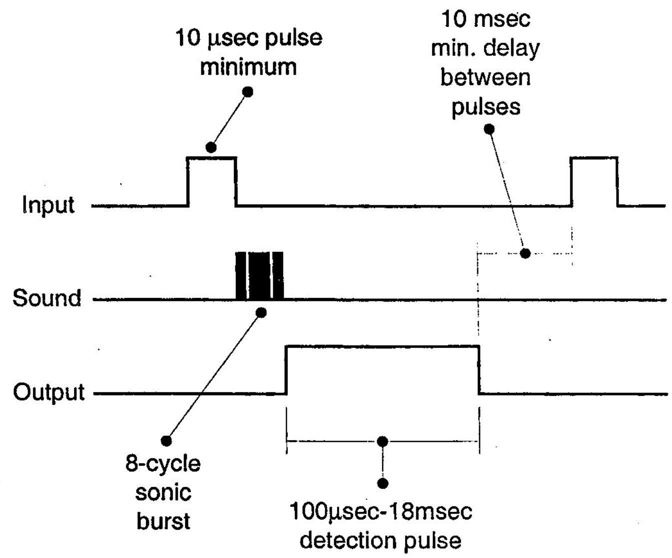

Figure 3

Figure 3 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, and then brought high for a minimum of 10 microsec. The pulse is generated on the falling edge of the command signal. After the command is given, the microcontroller must wait until it sees the output from the sensor go high. As soon as the output goes high, the microcontroller begins recording the lenth of the output signal until it sees the output go low again.

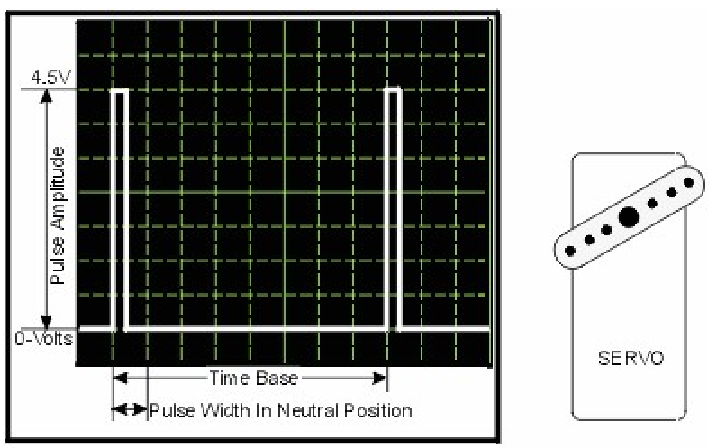

Figure 4

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.

sncserv.asm

```
; FILE: sncserv.asm
```

```
; AUTH: Keith Sevcik
; DATE: 1/24/03
; 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
PWM     equ     0x0c            ; PWM signal length
count   equ     0x0d            ; general register
temp    equ     0x0e            ; general register
loop    equ     0x0f            ; general register

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

c       equ     0               ; status bit to check after subtraction

;----------------------------------------------------------------------
; porta0 = input from sensor
; portb3 = command to sensor
; portb0 = command to servo

        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
main    clrf    count           ; clear count
        clrf    PWM             ; clear PWM
        bsf     portb,3         ; tell sensor to make reading
        movlw   d'4'            ; 4*3=12 clock delay
        movwf   count
        call    delay           ; delay
        bcf     portb,3         ; end sensor command signal
        movlw   d'5'            ; set the delay for measuring the output
        movwf   count
A0LOW   btfss   porta,0         ; if the output has gone high, skip the next instruction
        goto    A0LOW           ; else check again
A0HIGH  incf    PWM             ; increment the length of the PWM signal
        call    delay           ; delay for the count assigned above
        nop
        btfsc   porta,0         ; check to see if the output is still high
        goto    A0HIGH          ; if it is, repeat
        movf    PWM,w           ; move PWM to w
        sublw   d'200'          ; 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
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
        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

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

        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
PWM     equ     0x0c            ; PWM signal length
count   equ     0x0d            ; general register
temp    equ     0x0e            ; general register
loop    equ     0x0f            ; general register

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

c       equ     0               ; status bit to check after subtraction

;---------------------------------------------------------------------
; porta0 = input from sensor
; portb3 = command to sensor
; portb0 = command to servo
```

```
        org     0x000
```

The only equate of signifficance here is PWM. This register will be used to store the length of the PWM signal to be generated.

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    clrf    count           ; clear count
        clrf    PWM             ; clear PWM
        bsf     portb,3         ; tell sensor to make reading
        movlw   d'4'            ; 4*3=12 clock delay
        movwf   count
        call    delay           ; delay
        bcf     portb,3         ; end sensor command signal
```

After setting up the ports, the main loop is begun. At the beginning of the main loop, the count and PWM registers are cleared. The command pulse is then sent to the sensor.

```
        movlw   d'5'            ; set the delay for measuring the output
        movwf   count
A0LOW   btfss   porta,0         ; if the output has gone high, skip the next instruction
        goto    A0LOW           ; else check again
```

The next bit of code sets up the counter for the next operation. This counter will add a delay to the process of measuring the output from the sensor. This delay will scale down the output from the sensor. The following two lines of code detect when the output from the sensor goes high.

```
A0HIGH  incf    PWM             ; increment the length of the PWM signal
        call    delay           ; delay for the count assigned above
        nop
        btfsc   porta,0         ; check to see if the output is still high
        goto    A0HIGH          ; if it is, repeat
```

This loop increments the PWM register, delays, and then loops again as long as the output from the sensor is high.

```
        movf    PWM,w           ; move PWM to w
        sublw   d'200'          ; 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 SRF04 ultrasonic 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 ultrasonic sensor.

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