Control of RFFS-100 Micro Servos

Thr RFFS-100 RF Flight System from Dynamics unlimited is a receiver package complete with voice coils used for servoing typically in small, low velocity aircraft. It is important to know how to control these servos with a microcontroller if you wish to achieve automated flight.

MOTIVATION AND AUDIENCE

The focus of this tutorial is to demonstrate a method for controlling the voice coils that come with the RFFS-100 package with a PIC 16F84. This tutorial will teach you:

- ***How the servos work.***
- ***What control signal is needed to drive the servos.***
- ***How to write code to for the PIC 16F84 to control the servos.***
- ***What hardware is required to drive the servos.***
- ***How to assemble the necessary hardware.***

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
Micro Servo	**Dynamics Unlimited**	-----	$20.00	1
2N4403 PNP Transistor	**Fairchild Semiconductor**	2N4403	$0.0453	2
2N3904 NPN Transistor	**Fairchild Semiconductor**	2N3904	$0.0413	2
1 kOhm Resistor				4

The first item listed is the servo we will be actuating. It can be purchased individually as shown or as part of the RFFS-100 package. The servo can not be directly driven by the PIC. It is therefore necessary to build an "H-Bridge". This will allow the PIC to drive the current in both

directions across the coil. The details for constructing this circuit are outlined later. You will also need to construct a test structure for the servo (the one shown in this tutorial is the elevator for the **CQAR**).

To construct the circuit, you will also need:

- ***alligator clips***
- ***DC power supply***
- ***Soldering Iron***
- ***Solder***

The items listed above can all be purchased from an electronics store such as Radio Shack.

BACKGROUND

This servo is simply a voicecoil. A voicecoil is a coil of wire that generates a magnetic field when current is passed through it and moves a permanent magnet with this generated field. In this case, the magnet is oreinted as shown in Figure 1.

Figure 1

When voltage is applied to the coil, a magnetic field is generated in the direction normal to the screen (along the axis of the coil). The magnet then tries to align its poles with this magnetic field. When voltage is applied in the opposite direction, the magnet again tries to align itself, this time in the opposite direction. It can be seen, then, that in order to get the full range of motion out of this servo, you must be able to apply +5 VDC to both leads of the coil (to achieve an effective switch in direction of current). You must also be able to achive certain voltage levels inbetween +/- 5 VDC. This can be done with a PWM signal.

HARDWARE

This portion of the tutorial outlines the construction of the H-Bridge necessary to control the actuator. The diagram for the circuit is shown below:

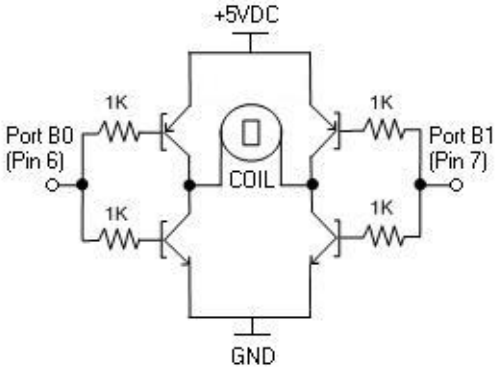

Figure 2

To deflect the actuator in one direction, a high signal is sent to one side of the H-Bridge and a low is sent to the other. This allows current to flow in one direction, as shown in Figure 3.

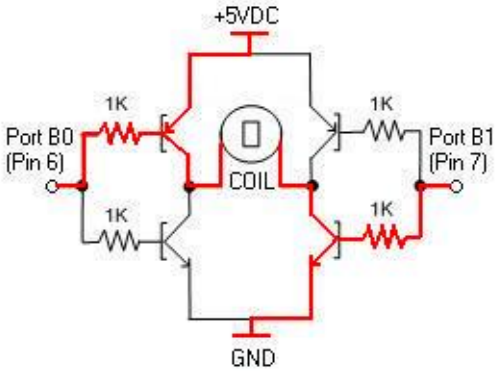

Figure 3

To deflect the actuator in the opposite direction, the signals are reversed. This allows current to flow in the opposite direction, as shown in Figure 4.

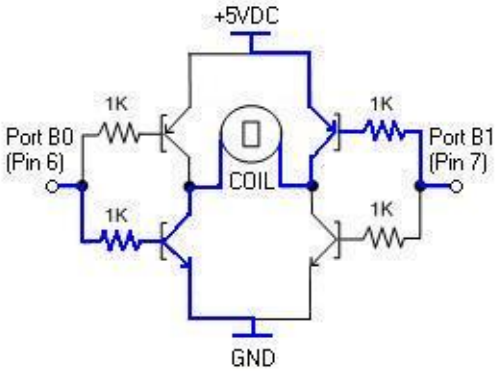

Figure 4

The actual construction of the circuit can be done on a PCB or by simply twisting the leads together and soldering. \care should be taken in dealing with the leads of the actuator, as the are

fragile and break off easily. Also, avoid soldering to the actuator leads as the plastic coating easily melts from the heat of the soldering iron.

PROGRAMMING

As stated above, to control the servo we must be able to effectively send a positive and negative PWM signal to the servo. This is not inuitively feasible as the PIC is only able to output +5 VDC. However, by splitting the ouput between two pins on the PIC, we can apply voltage in either direction by alternating which pin is +5 and which is ground. The logic behind a program that does this is illustrated in Figure 2.

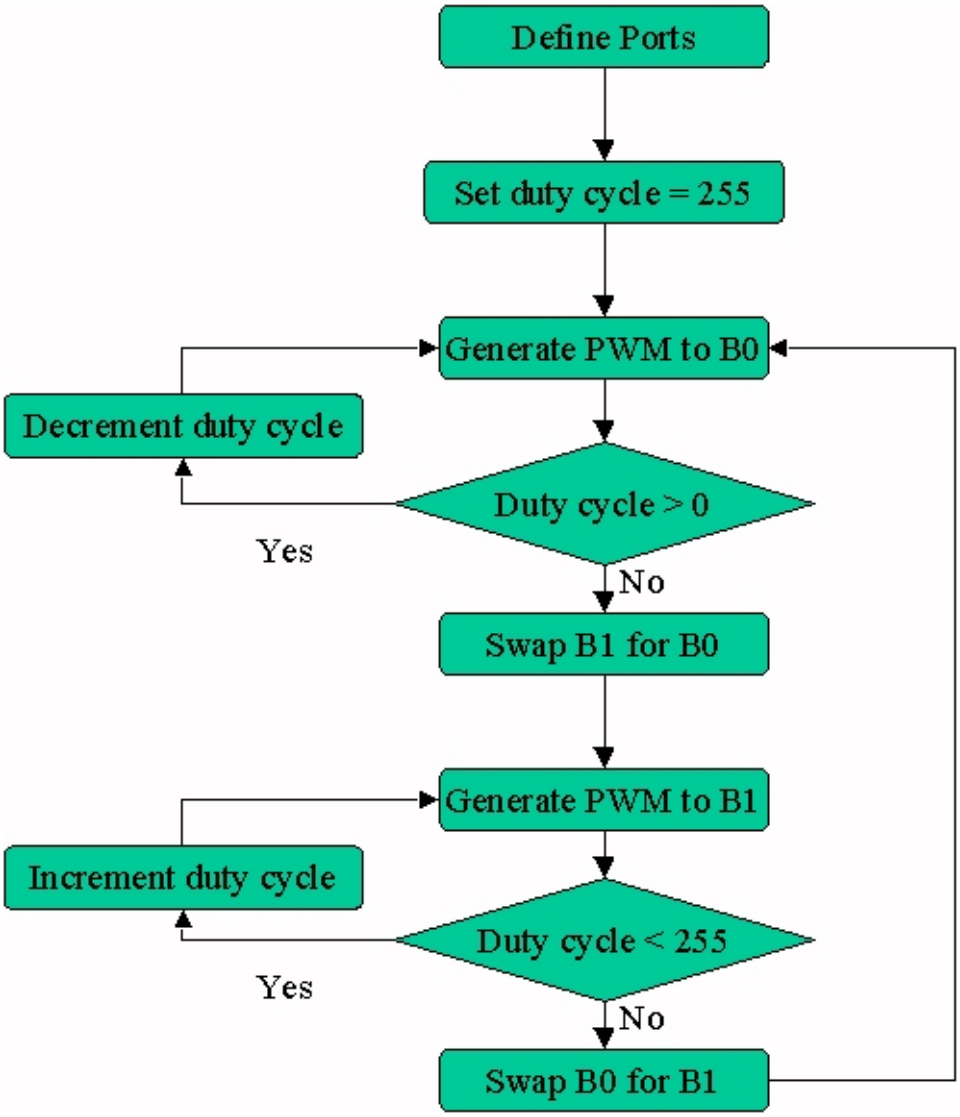

Figure 5

The duty cycle of the pwm signal is initiated to its highest value (equivalent to a +5 VDC source) and then decremented down to 0 (equivalent to 0 VDC). At this point, the pin outputting the PWM signal is switched, and the duty cycle is incremented back up to its maximum value (now equivalent to -5 VDC). Once it reaches max again, the process starts over. The code to implement this is shown below.

coillp1.asm

```
; FILE: coillpV1.asm
; AUTH: Keith Sevcik
; DATE: 5/2/03
; DESC: This program generates a PWM waveform to a voice coil.  The leads of the
;       coil should be hooked up to output pins B0 and B1.  At any one point
;       one pwm is generating a PWM signal while the other is kept at ground.  By
;       alternating which pin is ground and which is signal, a positive and negative
;       PWM signal cna be achieved, thus getting the maximum deflection from the coil.
; NOTE: Tested on PIC16F84-04/P

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

        list    p=16f84
        radix   hex

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

portb   equ     0x06            ; port b equate
duty    equ     0x0c            ; length of duty cycle
temp    equ     0x0d            ; length of duty cycle

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

c       equ     0               ; status bit to check after subtraction

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

        org     0x000

start   movlw   0x00            ; load W with 0x00 make port B output
        tris    portb           ; copy W tristate to port B outputs
        movlw   0x00            ; fill w with zeroes
        movwf   portb           ; set port b outputs to low
rstrt   movlw   d'0'
        movwf   portb
        movlw   d'255'
        movwf   duty
b0loop  movf    duty,w
        movwf   temp
        bsf     portb,0
pwm1a   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm1a
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,0
pwm1b   nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm1b
        decfsz  duty
        goto    b0loop
        movlw   d'0'
        movwf   portb
        movlw   d'0'
        movwf   duty
b1loop  movf    duty,w
        movwf   temp
        bsf     portb,1
pwm2a   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm2a
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,1
pwm2b   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm2b
        incfsz  duty
        goto    b1loop
        goto    rstrt

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

        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

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

portb   equ     0x06            ; port b equate
duty    equ     0x0c            ; length of duty cycle
temp    equ     0x0d            ; length of duty cycle

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

c       equ     0               ; status bit to check after subtraction

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

        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           ; copy W tristate to port B outputs
        movlw   0x00            ; fill w with zeroes
        movwf   portb           ; set port b outputs to low
```

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

```
rstrt   movlw   d'0'
        movwf   portb
        movlw   d'255'
        movwf   duty
```

After setting up the ports, the main loop is begun. At the beginning of the main loop, all port b pins are set to low just incase they are high when they shouldn't be. The duty cycle is then set to 255.

```
b0loop  movf    duty,w
        movwf   temp
        bsf     portb,0
```

```
pwm1a   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm1a
```

The next bit of code is the loop for the PWM signal generated at pin B0. The pwm1a loop generates the high portion of the PWM signal. The duty cycle is stored in temp and then the pin is set high. after a pause, temp is decremented and so long as it doesnt reach zero the pause is repeated and temp is decremented again. After temp reaches zero, the code continues.

```
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,0
pwm1b   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm1b
        decfsz  duty
        goto    b0loop
```

The next portion of code generates the low part of the PWM signal. The value 255 is stored in temp, and the duty cycle is subtracted from this. This gives the remaining length of signal to be generated. Temp is then decremented in the same manner as above, this time with B0 set to low. Once the entire PWM signal has been generated, the duty cycle is decremented and the PWM signal is generated again. This continues until the duty cycle reaches zero, at which point B1 becomes the output for the PWM signal.

```
        movlw   d'0'
        movwf   portb
        movlw   d'0'
        movwf   duty
b1loop  movf    duty,w
        movwf   temp
        bsf     portb,1
pwm2a   nop
        nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm2a
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,1
pwm2b   nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwm2b
        incfsz  duty
        goto    b1loop
        goto    rstrt
```

The port b outputs are set to low to ensure that b0 is low and that b1 starts at low. The duty cycle is then intialized at 0. From this point, the PWM signal is generated in the same fashion, except this time incrementing the duty cycle to its maximum value of 255 before going back to the beginning.

This code causes the actuator to start at one extreme, move slowly down to the opposite extreme, and then jerk back up to the extreme where it began. A more elegant display of motion of the servo is to have the servo move smoothly from one extreme to the next and then move back. The code to achieve this is similar to that outlined above. The difference is that where this code ends with B! at its maximum duty cycle, the elegant code would then decrement the duty cycle for B1 back down to 0, switch outputs to B0, increment the duty cycle back up to 255 for B0 and then repeat the code all over again. Both versions are included below accompanied by an avi file that demonstrates how they work.

Coil Loop V1 - Assembly code for slow move down then reset
Coil Movie 1 - avi file for slow move down then reset
Coil Loop V2 - Assembly code for slow move up and down
Coil Movie 2 - avi file for slow move up and down

FINAL WORDS

After completing this tutorial you should be familiar with the servos that accompany the RFFS-100 flight system and how to program a PIC 16F84 to control them.

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