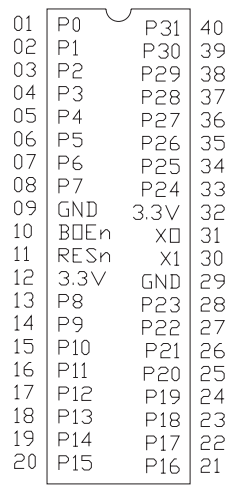


Figure 2-1 **The pin designations for the 40-pin Propeller chip**

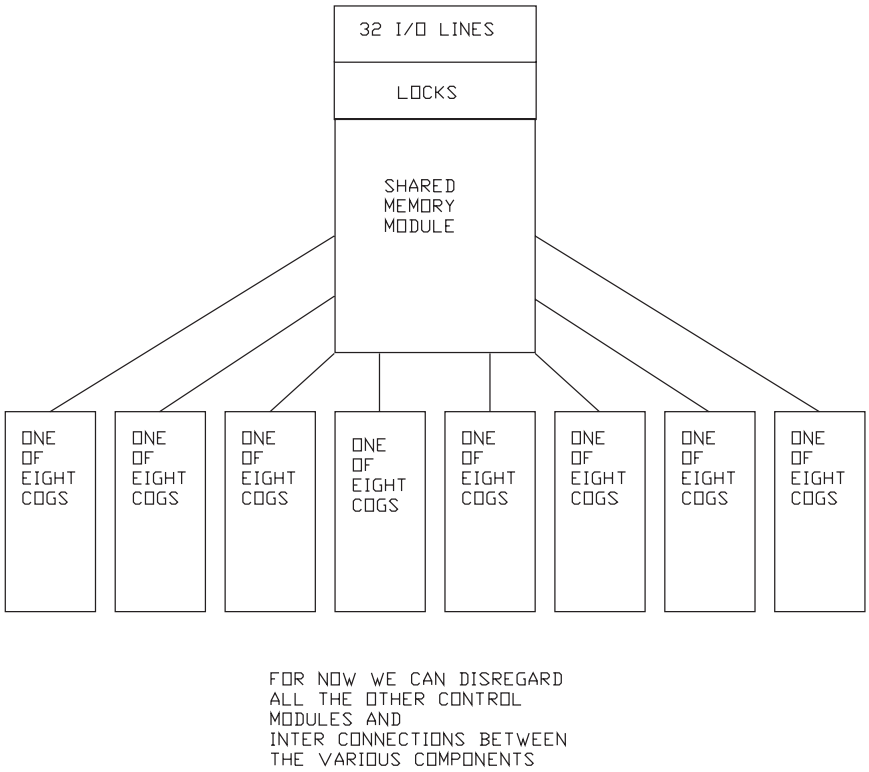


Figure 2-2 **The basic layout of the Propeller system**

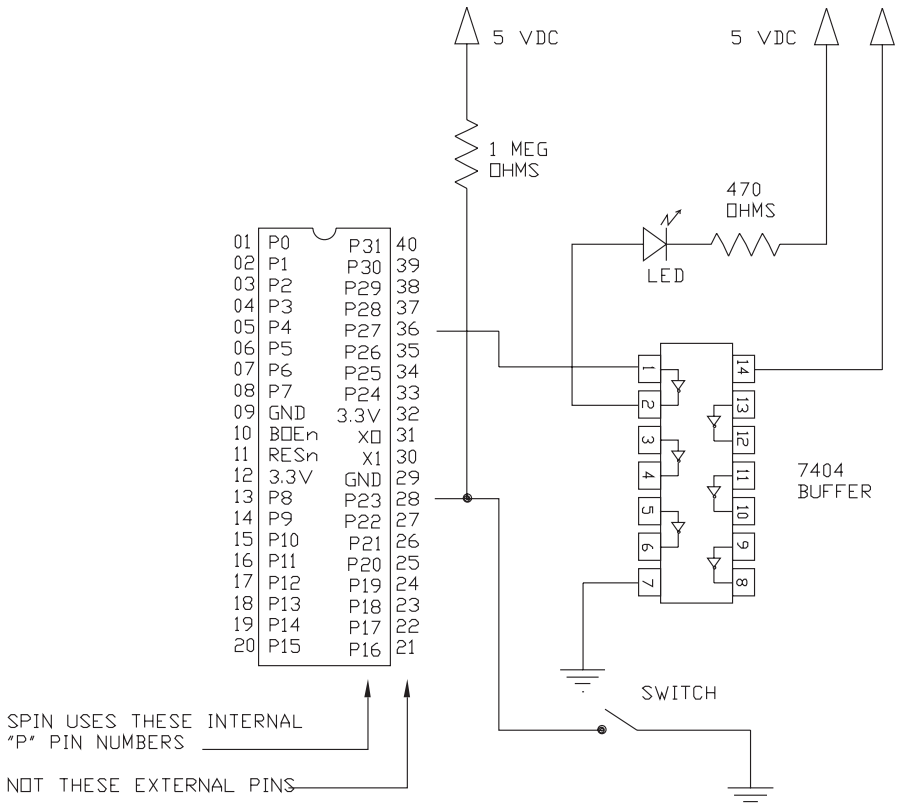


Figure 2-3 **Using an inverting buffer to connect to an LED and a dry contact switch with a pull-up resistor**

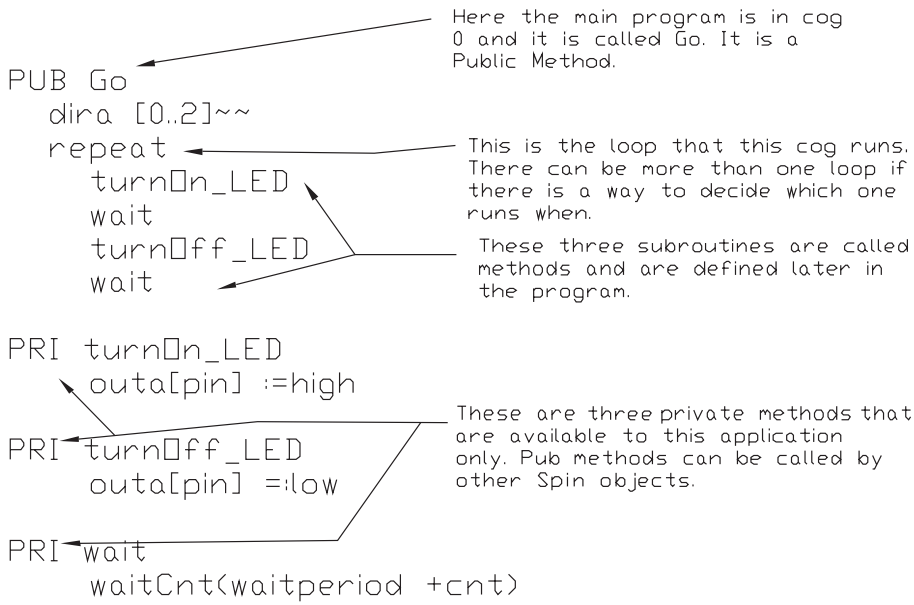


Figure 2-4 How methods are organized and called in a Spin application

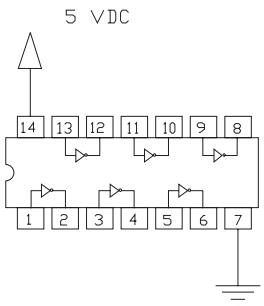


Figure 3-5 **Pinouts for the six inverters in a 7404 IC**

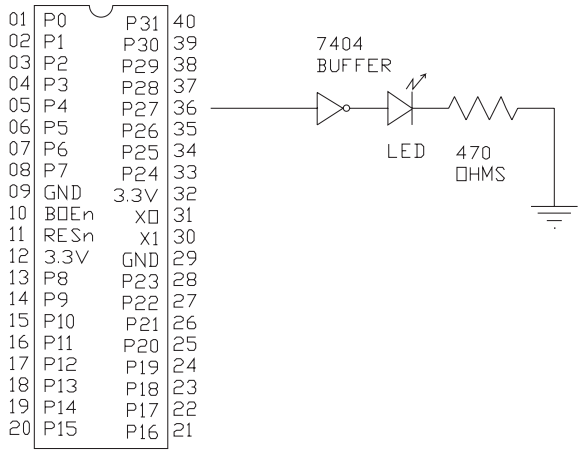


Figure 3-6 **Using one of the buffers in a 7404 to power an LED.**

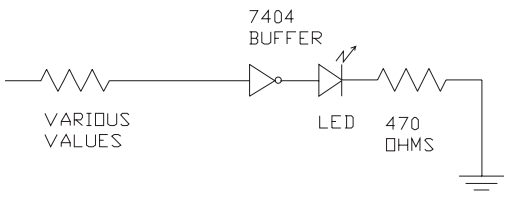


Figure 4-1 **Circuitry between the output line of the Propeller and LED**

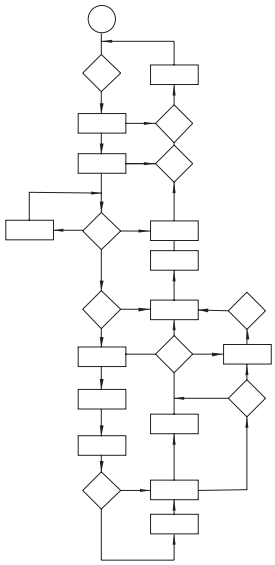


Figure 4-3 Linear program schematic

```
PUB Go
  dira [0..2]~~
  repeat
    turn□n_LED
    wait
    turn□ff_LED
    wait

PRI turn□n_LED
    outa[pin] :=high

PRI turn□ff_LED
    outa[pin] =:low

PRI wait
    waitCnt(waitperiod +cnt)
```

Here the main program is in cog 0 and it is called Go. It is a Public Method.

This is the loop that this C□G runs. There can be more than one loop if there is a way to decide which one runs when.

These three subroutines are called methods and are defined later in the program.

These are three private methods that are available to this application only. PUB methods can be called by other Spin objects.

Figure 4-5 Code descriptions for the program to blink an LED

Space assignment for a cog. Twenty-five longs are being assigned here.

This pin and direction declaration has nothing to do with the cog and method we are about to create on the next line.

This is the name of this new cog and public method. It uses two variables within it that do not have to be called out anywhere else. Only shared variables are called out.

```
VAR
  long Stack[25]
  long pulsWidth

PUB Go
  dira[6]~~
  cognew(MoveMotor(7),@Stack)
...
...
...
...
PUB MoveMotor(Pin)|CycleTime,period
  dira[Pin]~~
  ctra[30..26]:=%00100
  ctra[5..0]:=Pin
  frqa:=1
  PulsWidth:=0
  CycleTime:=clkfreq/100
  period:=cnt
  repeat
    phsa:=-PulsWidth
    period:=period+CycleTime
    waitcnt(period)
```

This is where you launch the second (next) cog to run MoveMotor. Its first executions takes place here in this space.

This constant is passed down to the method. It does not have to be declared in the constants.

These are the two variables that are used internally in this method. They are never used outside this method and they do not have to be declared. They will be treated as longs.

This is the pin that this cog uses. So this pin (Pin7) is set up here. Pin is not defined in the VAR or CON sections.

The two internal variables are used here and here.

This is the loop that this method runs. There can be more than one loop if there is a way to decide which one runs when.

Figure 4-6 **Assigning space for a new cog**

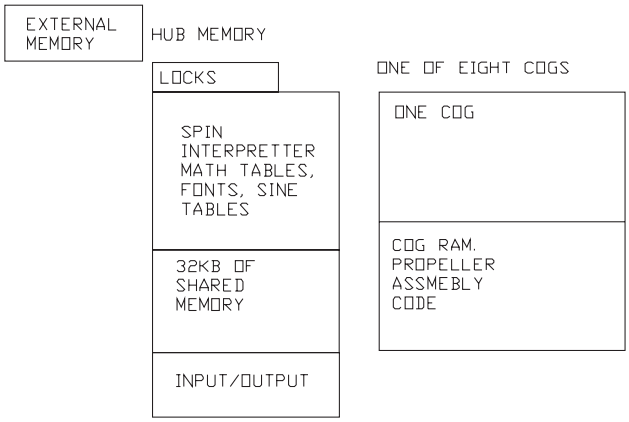


Figure 5-1 **An overview of the Propeller memory banks**

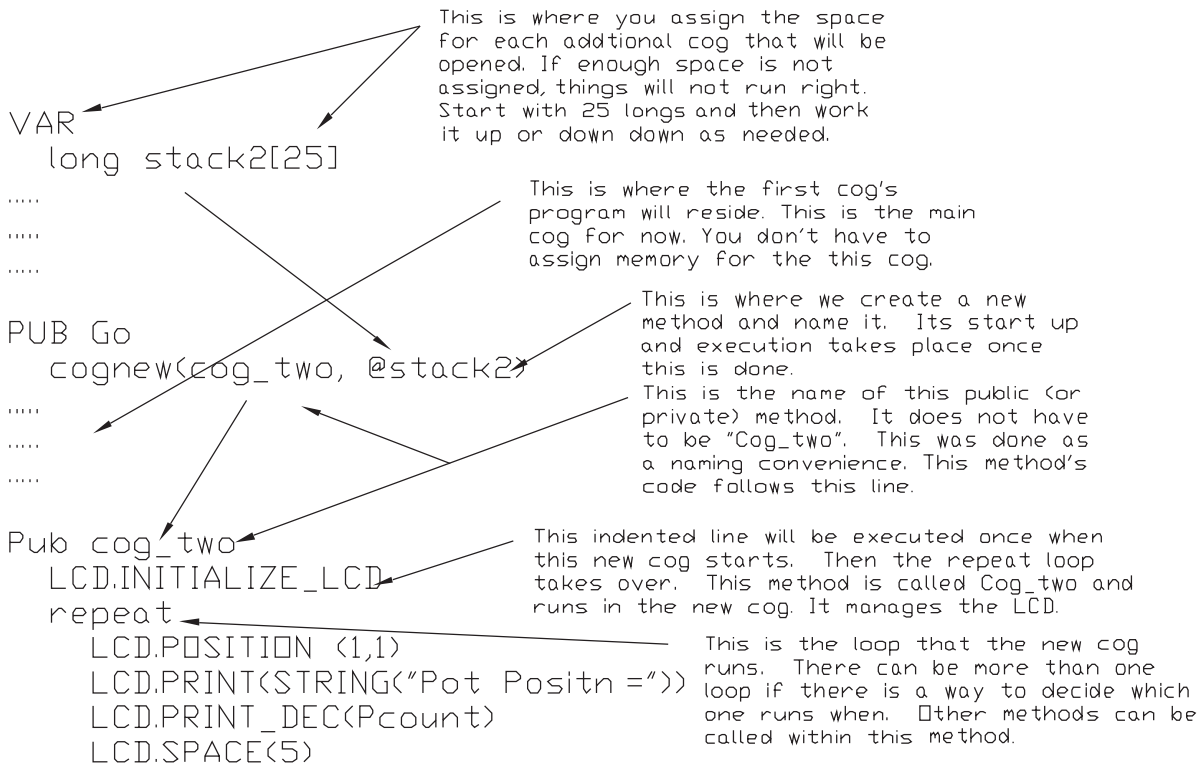


Figure 5-2 Simple cog launch explanation, with no variables

Space assignment for a cog. Twenty-five longs are being assigned here.

This pin and direction declaration has nothing to do with the cog and method we are about to create on the next line.

This is the name of this new cog and public method. It uses two variables within it that do not have to be called out anywhere else. Only shared variables are called out.

```
VAR
  long Stack[25]
  long pulsWidth

PUB Go
  dira[6]~~
  cognew(MoveMotor(7),@Stack)
'''
'''
'''
'''

PUB MoveMotor(Pin)|CycleTime,period
  dira[Pin]~~
  ctra[30..26]:=%00100
  ctra[5..0]:=Pin
  frqa:=1
  PulsWidth:=0
  CycleTime:=clkfreq/100
  period:=cnt
  repeat
    phsa:=-PulsWidth
    period:=period+CycleTime
    waitcnt(period)
```

This is where you launch the second (next) cog to run MoveMotor. Its first executions take place here in this space.

This constant is passed down to the method. It does not have to be declared in the constants.

These are the two variables that are used internally in this method. They are never used outside this method and they do not have to be declared. They will be treated as longs.

This is the pin that this cog uses. So this pin (Pin7) is set up here. Pin is not defined in the VAR or CON sections.

The two internal variables are used here and here.

This is the loop that this method runs. There can be more than one loop if there is a way to decide which one runs when.

Figure 5-3 **Advanced cog creation explanation, with variables.**

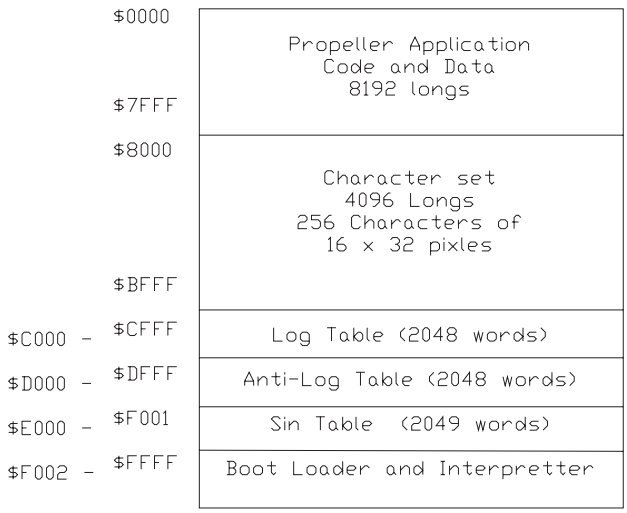


Figure 6-1 **Propeller hub main memory map (from page 31 of Propeller Manual [Ver. 1.1])**

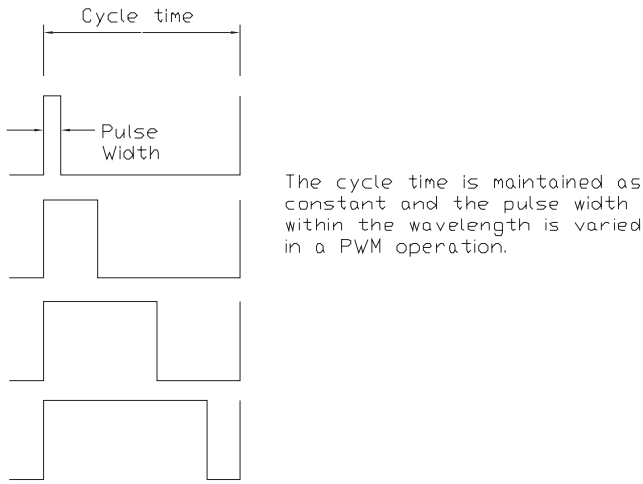


Figure 7-2 **Illustration of a PWM signal**

```
PUB Go
  dira [0..2]~~
  repeat
    turnOn_LED
    wait
    turnOff_LED
    wait

PRI turnOn_LED
    outa[pin] :=high

PRI turnOff_LED
    outa[pin] =:low

PRI wait
    waitCnt(waitperiod +cnt)
```

Here the main program is in Cog 0 and it is called Go. It is a public method.

This is the loop that this cog runs. There can be more than one loop if there is a way to decide which one runs when.

These three subroutines are called methods and are defined later in the program.

These are three private methods that are available to this application only. PUB methods can be called by other Spin objects.

Figure 10-1 **Simple program with three method calls**

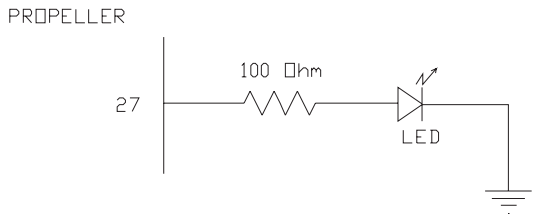


Figure 10-3 **Wiring schematic for blinking an LED in Program 10-1**

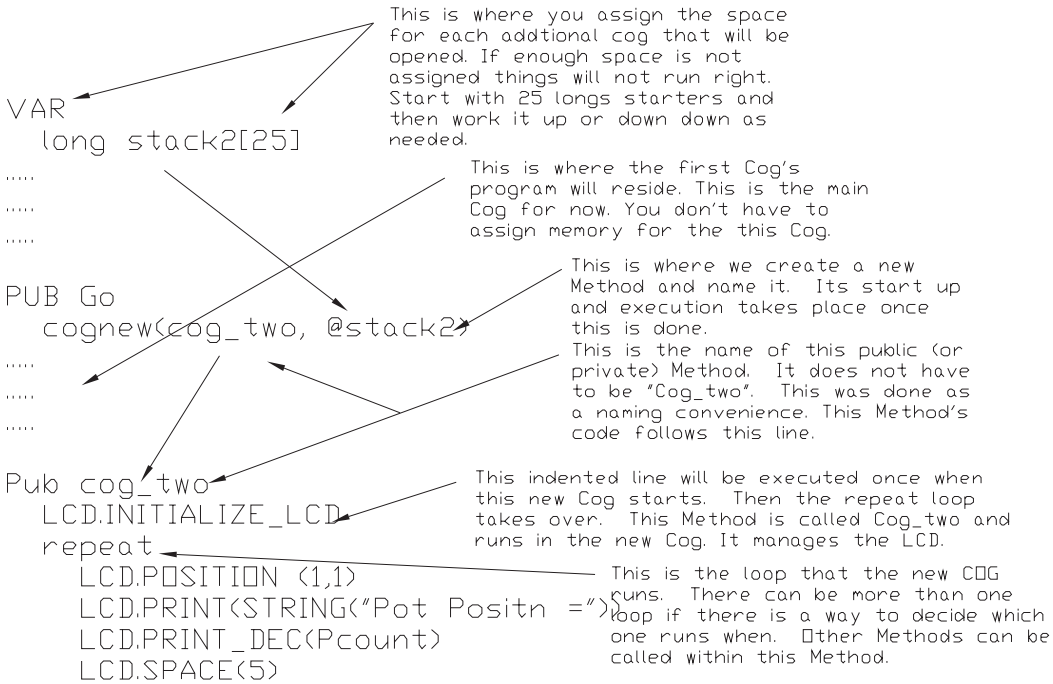


Figure 10-4 **Wiring for one LED as programmed in Program 10-1**

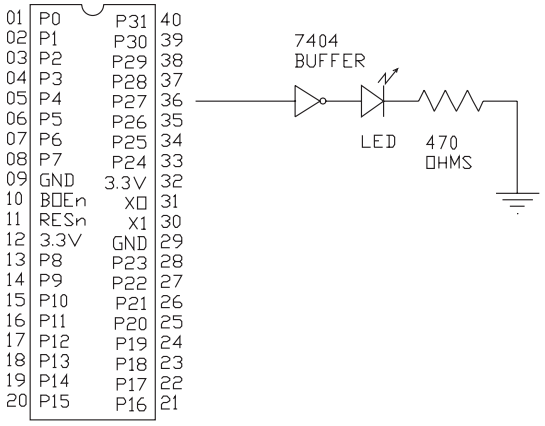


Figure 13-1 Wiring layout for blinking an LED

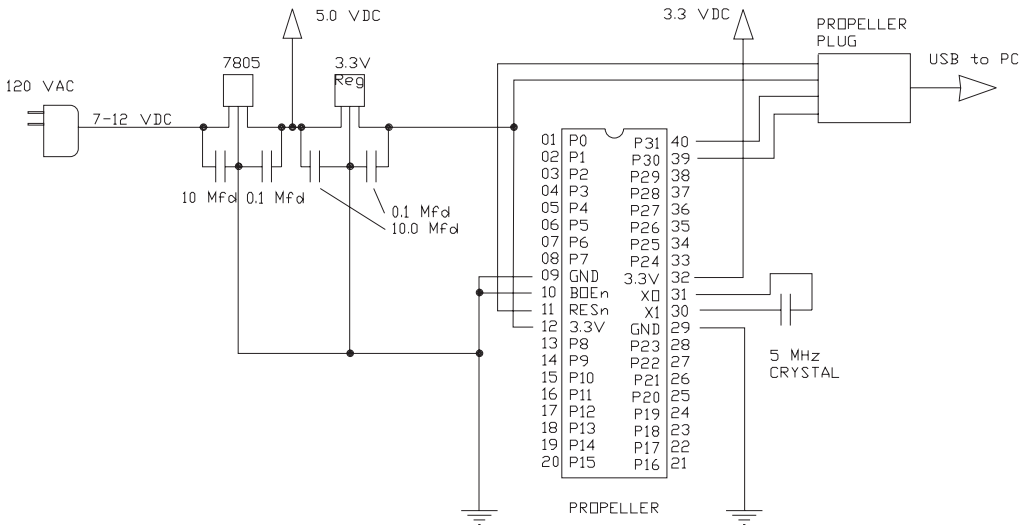


Figure 13-2 **Basic power-up layout and USB connection for a Propeller chip**

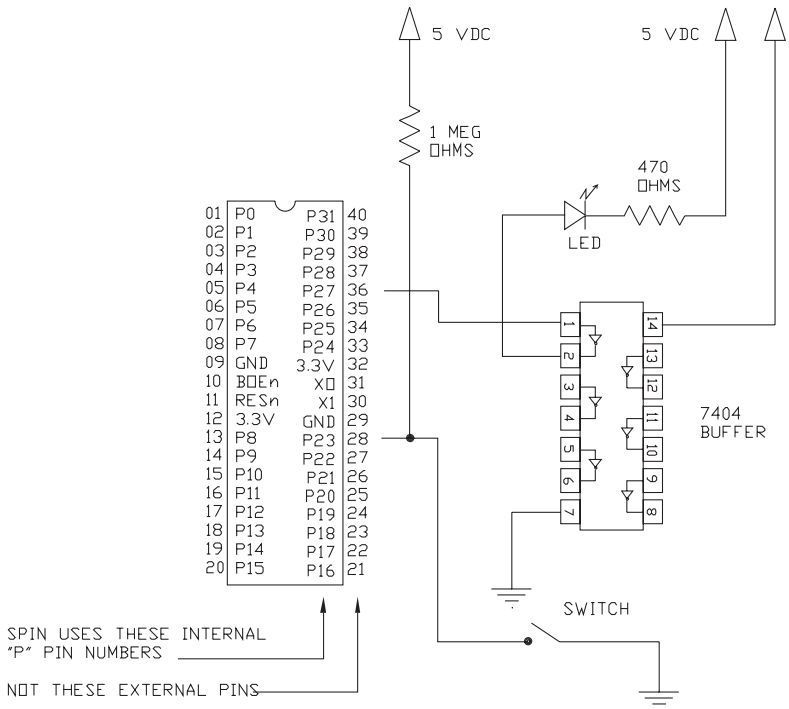


Figure 15-1 **Switch controls LED on line P27 by pulling line P23 low.**

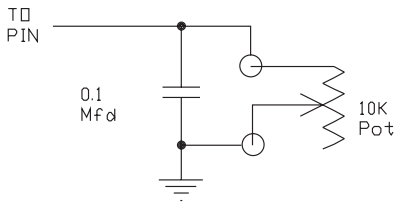


Figure 16-1 **Circuitry for reading a potentiometer with a Propeller chip**

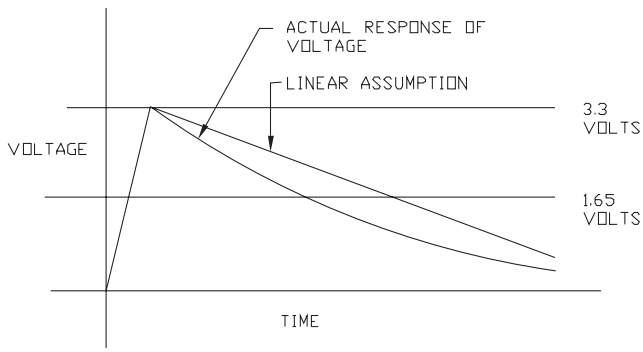


Figure 16-2 **Graphic of resistance vs. time to discharge**

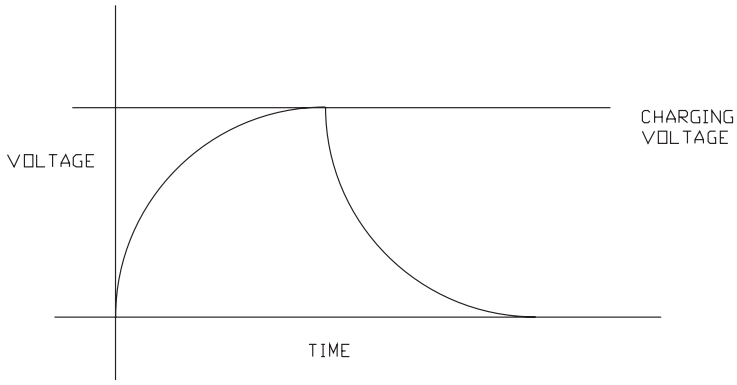


Figure 16-3 **Theoretical charging and discharging of a capacitor**

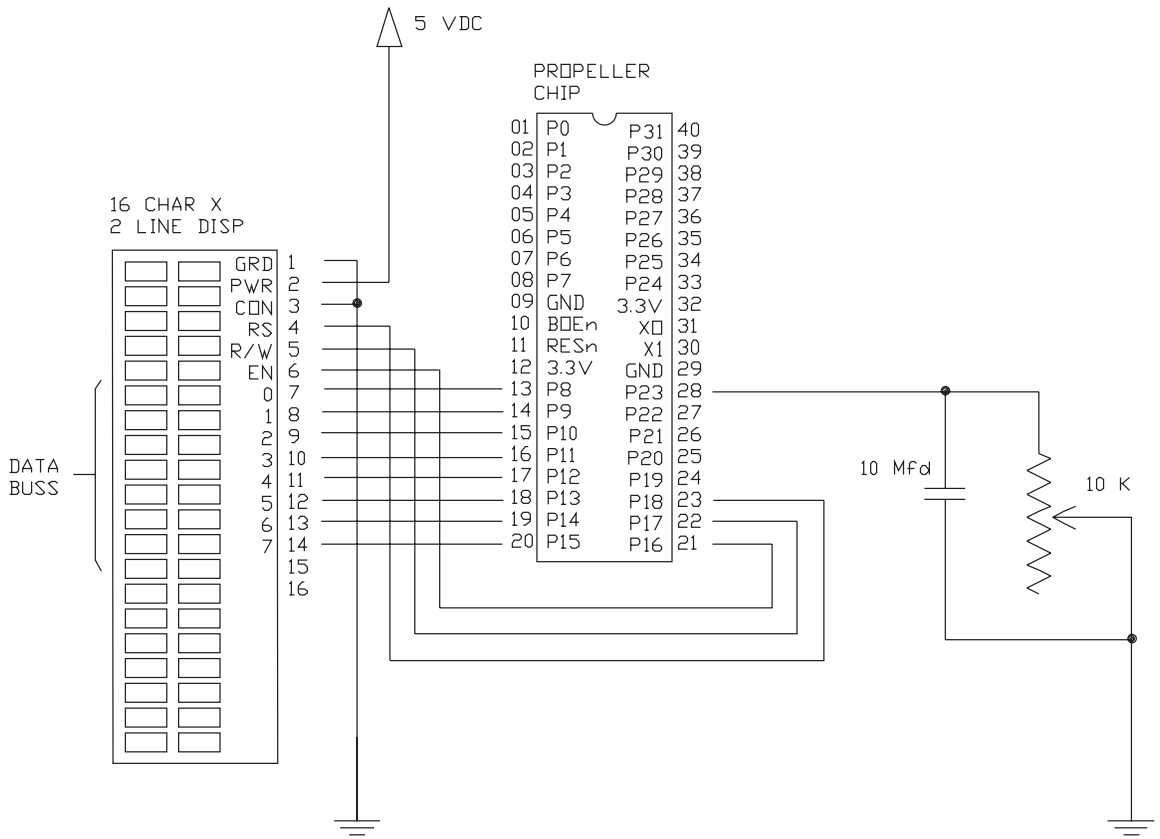
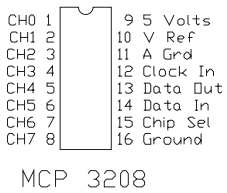


Figure 16-4 Complete circuitry for reading a potentiometer



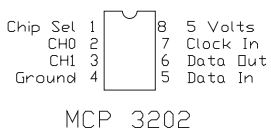


Figure 16-5 **The MCP3202 and MCP 3208 pinouts**

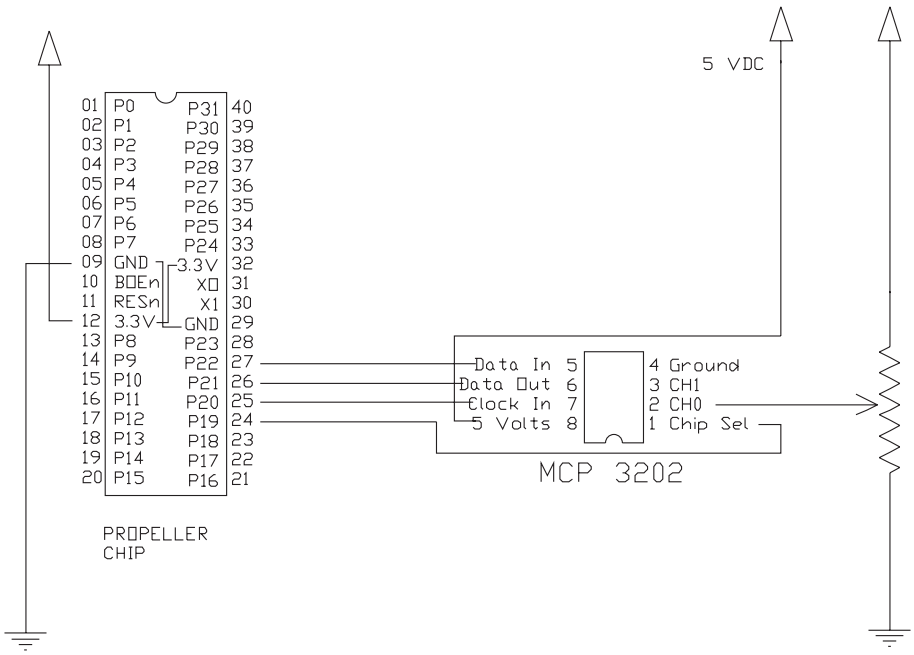


Figure 16-7 Wiring diagram for the code shown in Program 16-3

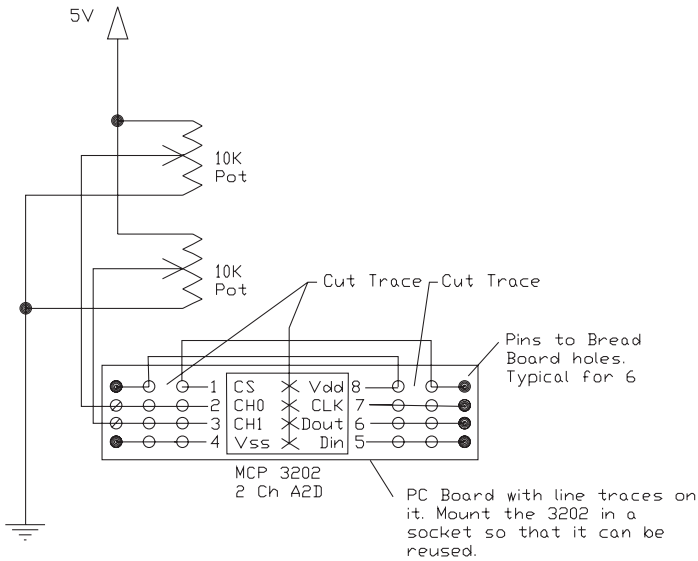


Figure 16-8 **Wiring diagram for a two-channel MCP3202 A2D module**

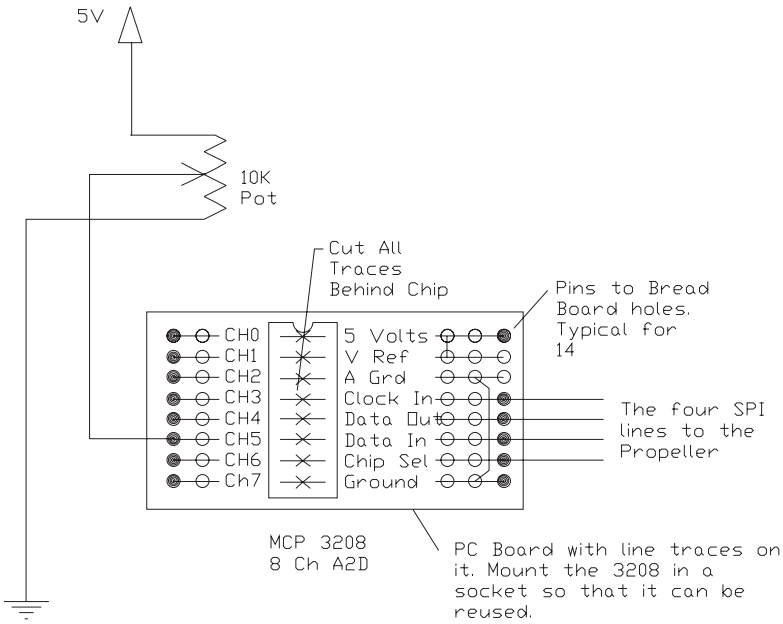


Figure 16-9 **Wiring diagram for an eight-channel 3208 A2D module**

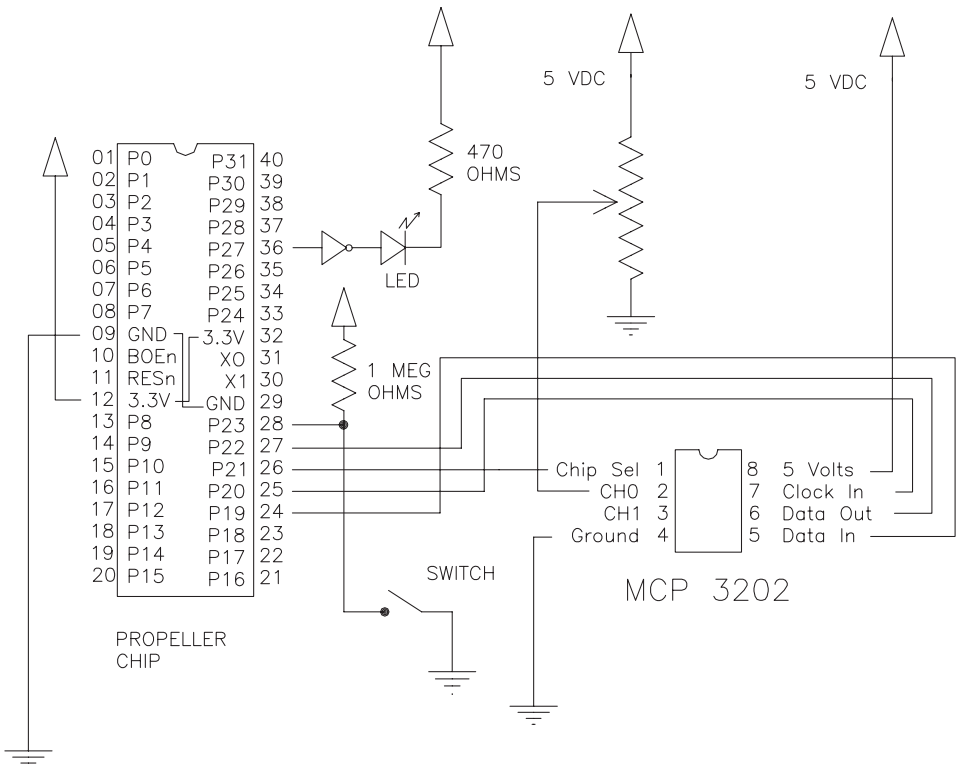


Figure 16-10 Wiring schematic for blinking an LED and reading a potentiometer

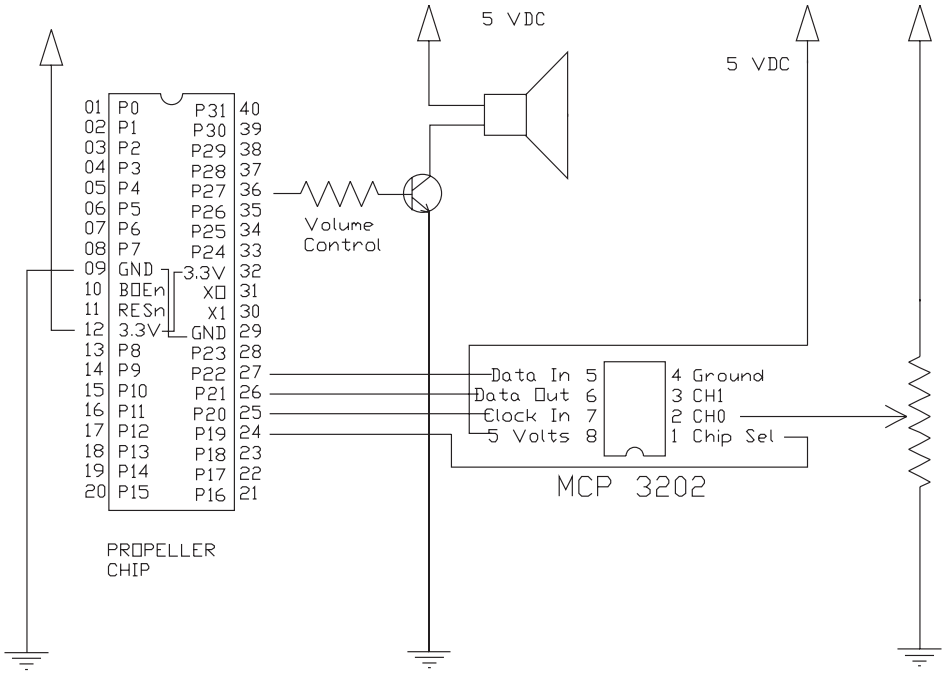


Figure 17-1 Wiring schematic for tone generator

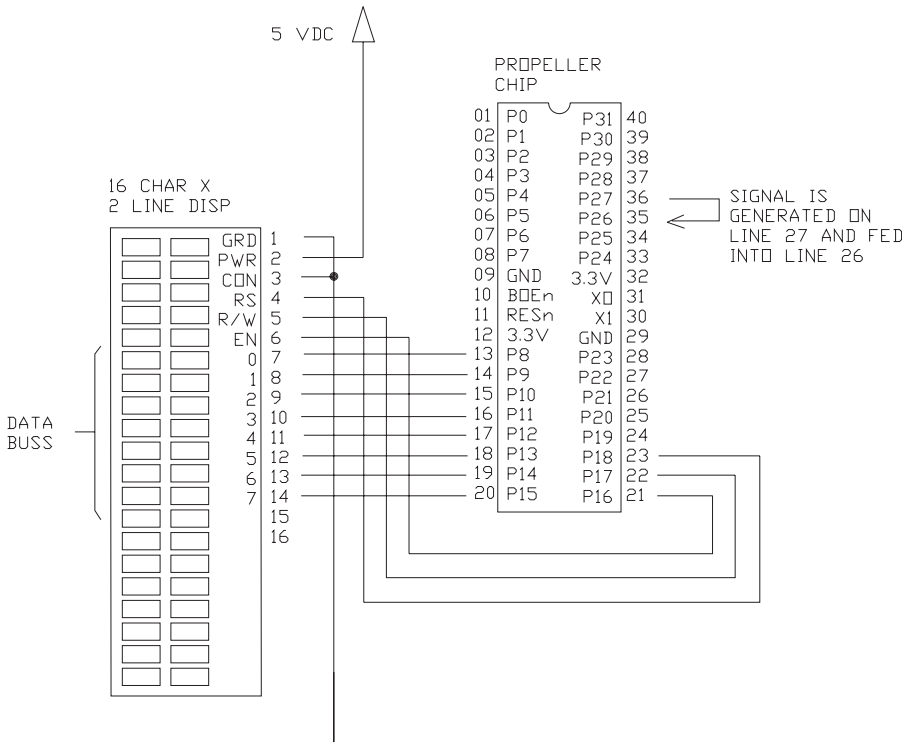


Figure 17-3 Setup for reading a fixed frequency

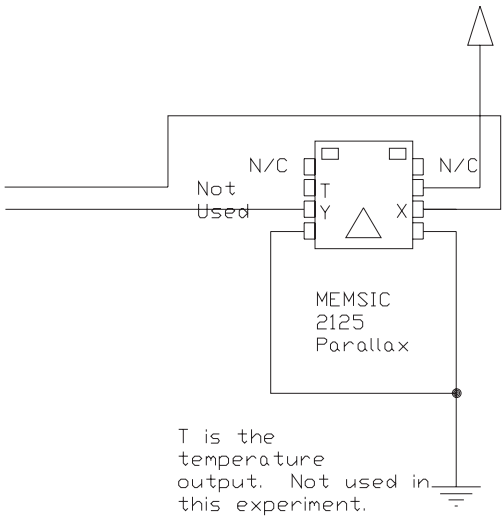


Figure 18-1 **Memsic 2125 gravity sensor connections**

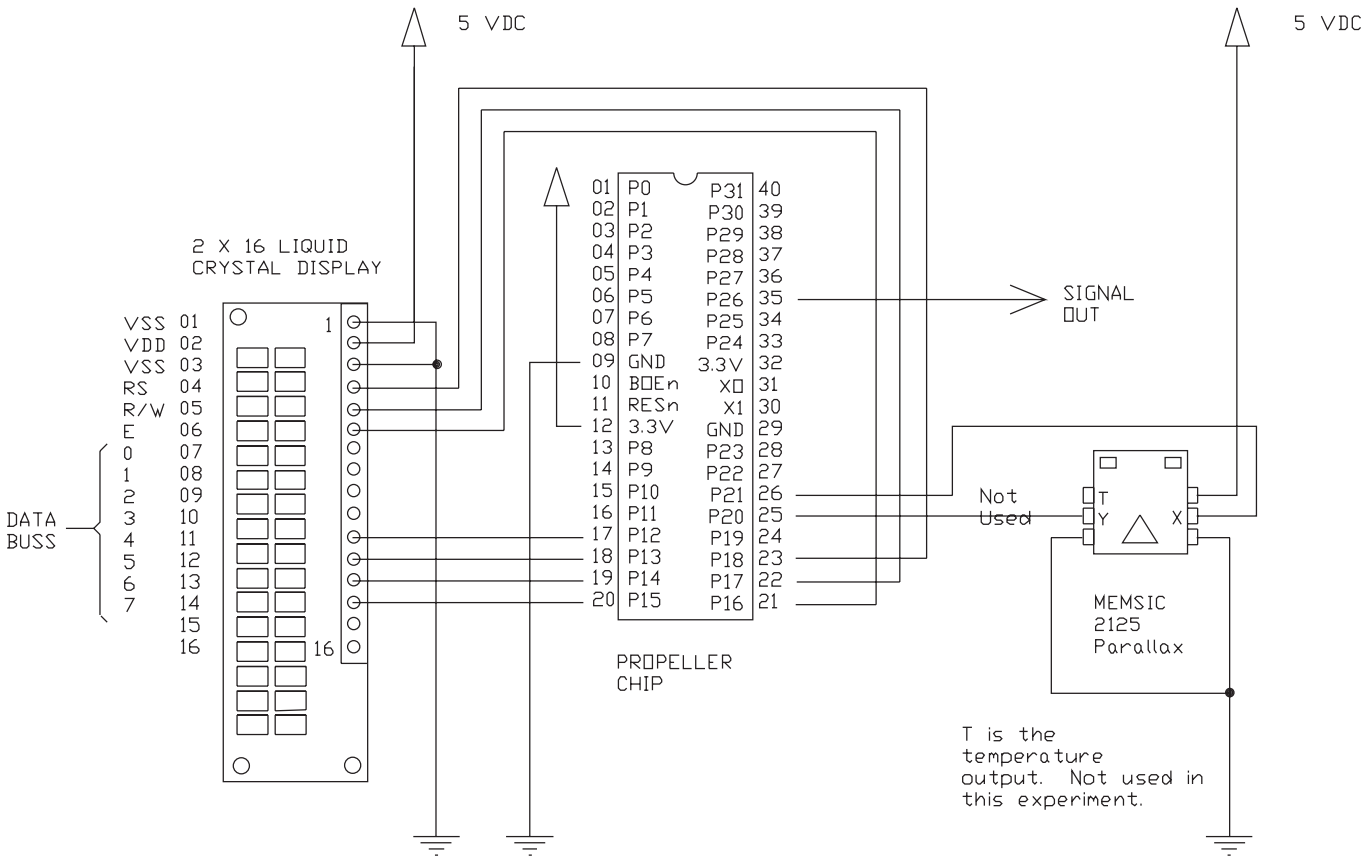


Figure 18-3 Wiring the Memsic 2125 to a Propeller chip and an LCD

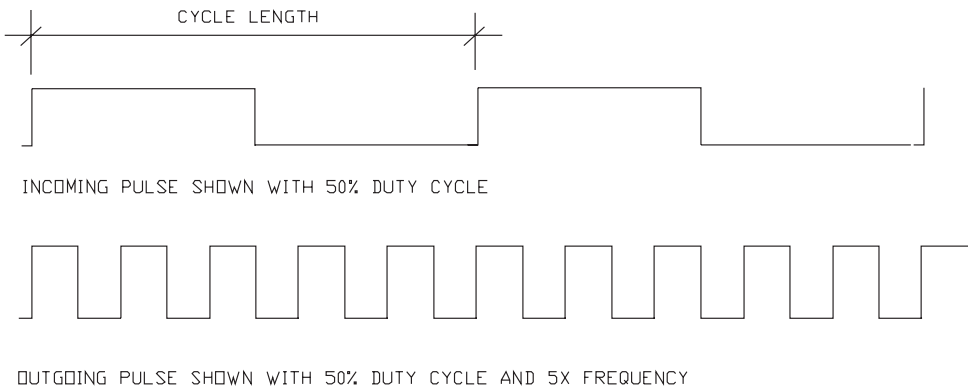
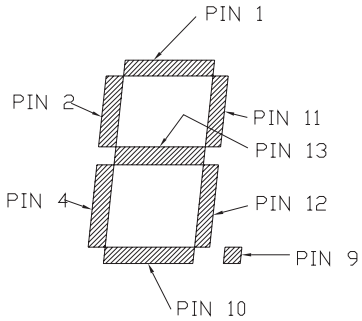


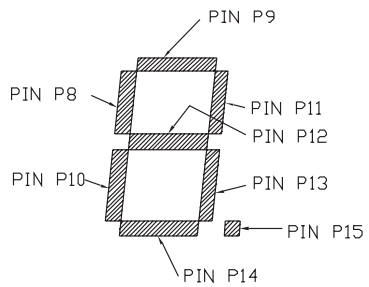
Figure 18-4 **Waveforms of read and created pulse widths**



PIN 3 IS THE COMMON ANODE.

PINS 5,7,8,11,15 AND 16 ARE EITHER NOT CONNECTED OR ARE MISSING. PIN 6 CONTROLS THE LEFT DECIMAL POINT.

Figure 19-2 **Pin assignment for a common anode seven-segment display (face view, as seen from above). These are the pins as assigned on the 16-pin device.**



CONNECTIONS TO THE PROPELLER PIN3. THE COMMON ANODE IS CONNECTED TO 5 VOLTS THROUGH A 220 OHM RESISTOR.

Figure 19-3 **Actual segment connections to the Propeller**

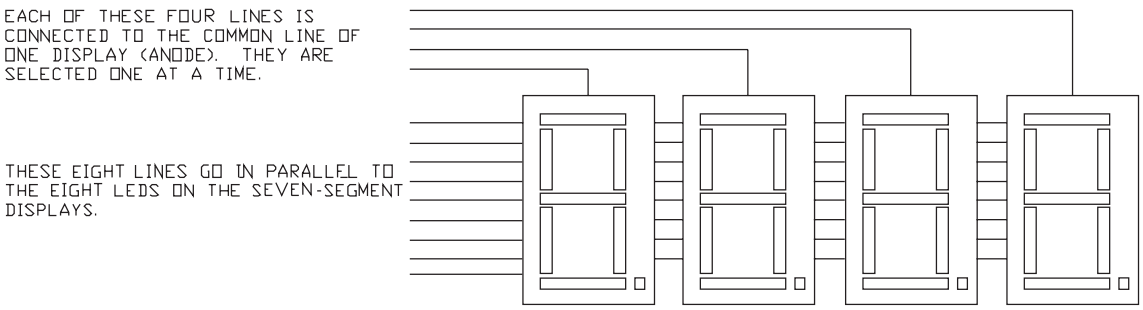


Figure 19-5 **Using four displays**

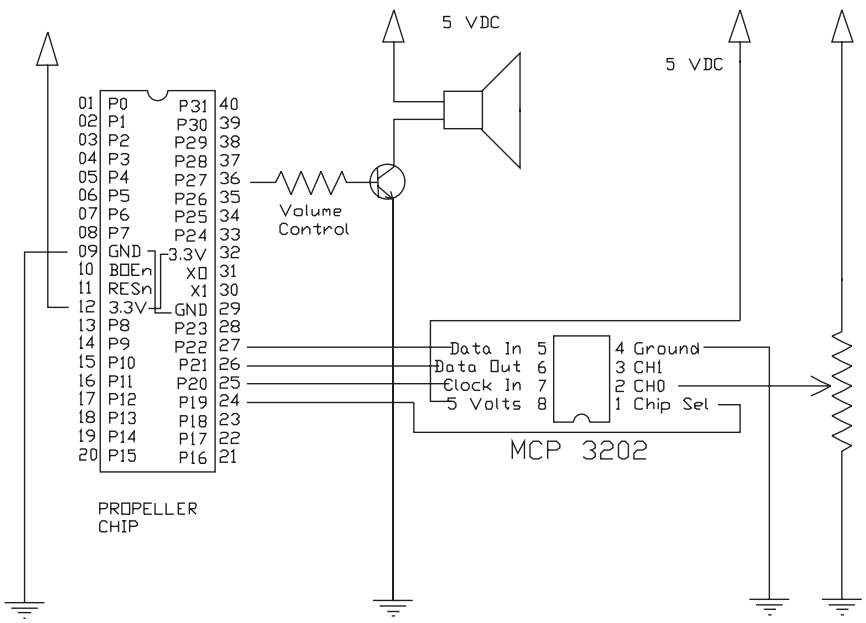


Figure 20-1 Wiring for the metronome

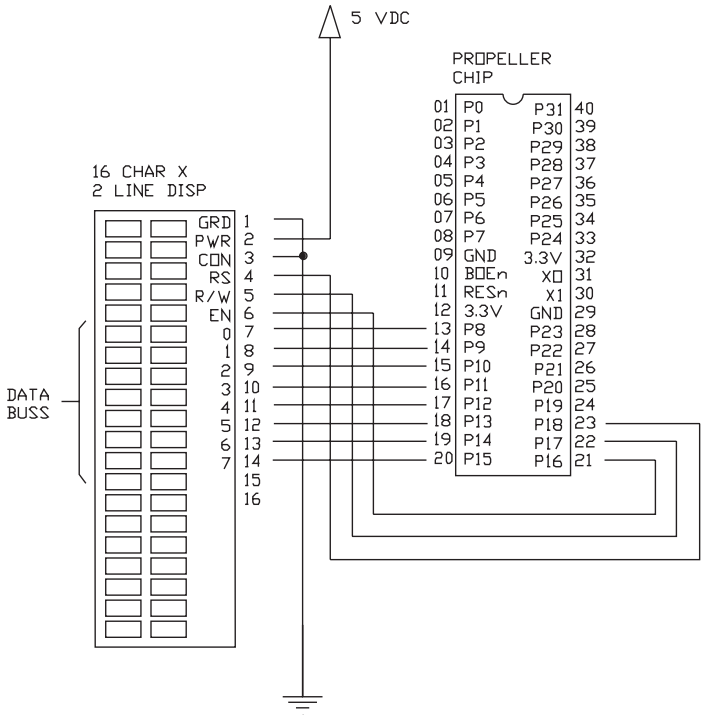


Figure 21-2 **Connecting the LCD to the Propeller chip (8-bit mode)**

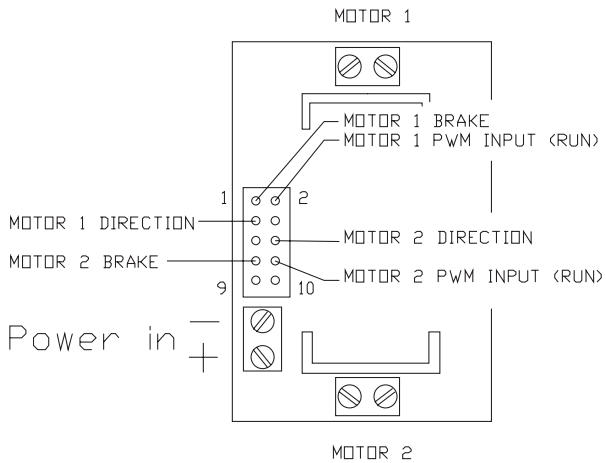


Figure 23-3 **Connection schematic for Xavien two-axis amplifier**

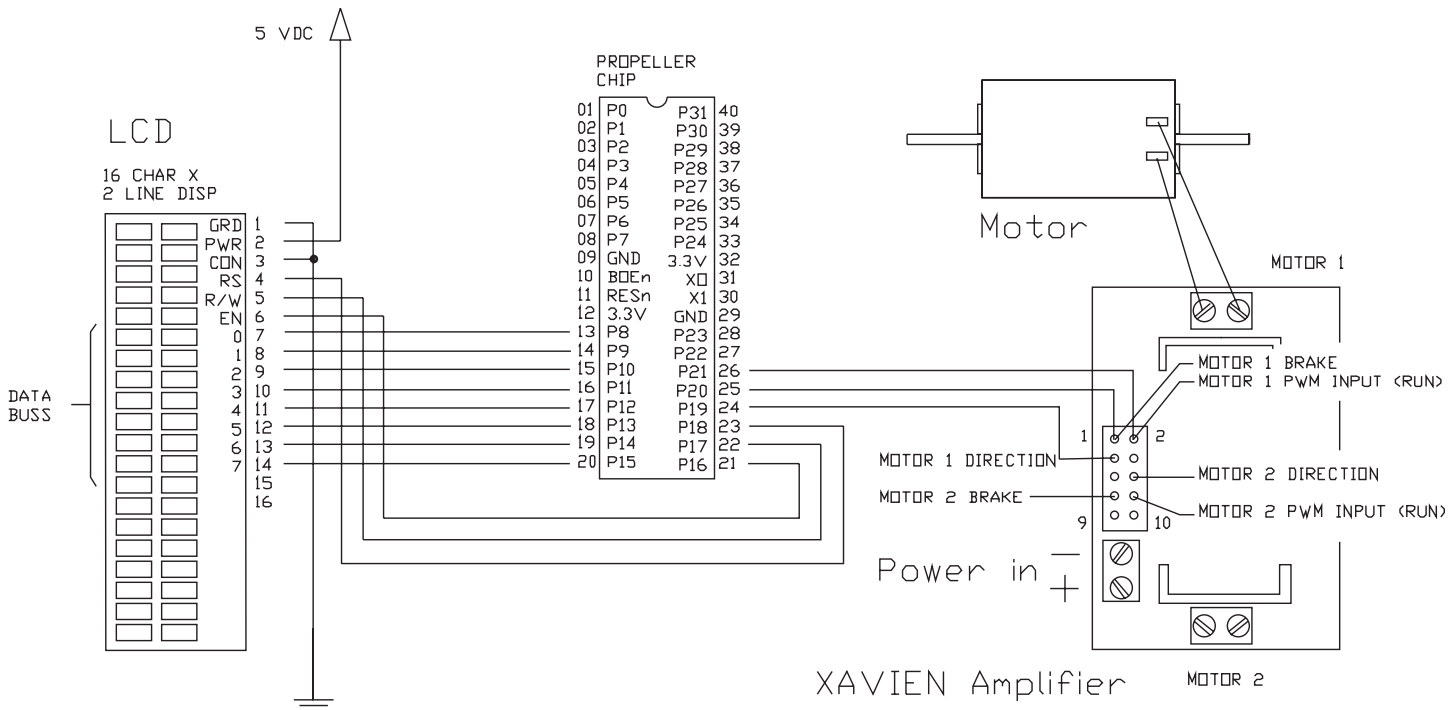


Figure 23-4 Using the Xavien two-axis amplifier with a motor

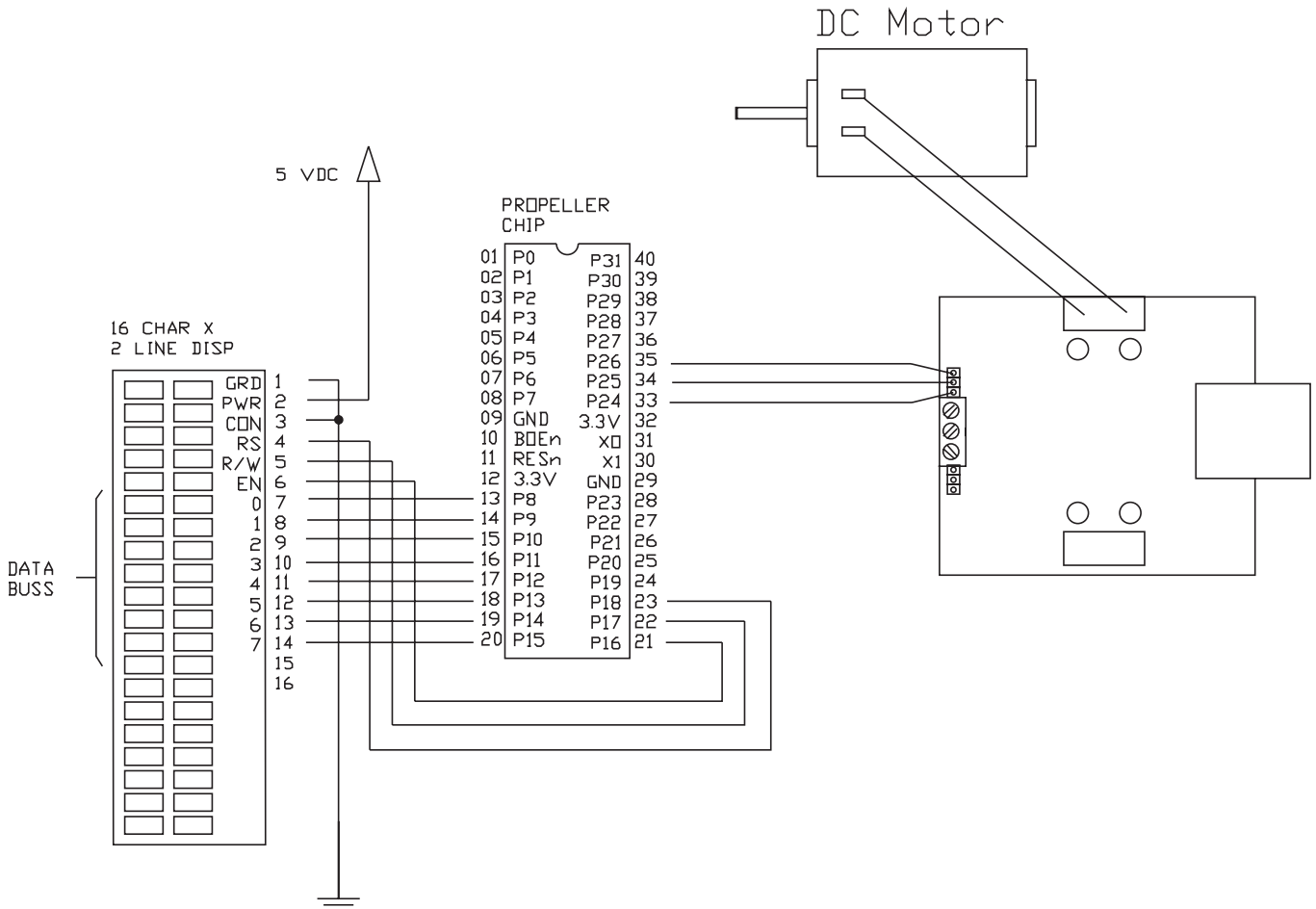


Figure 23-7 Using the Solarbotics two-axis amplifier

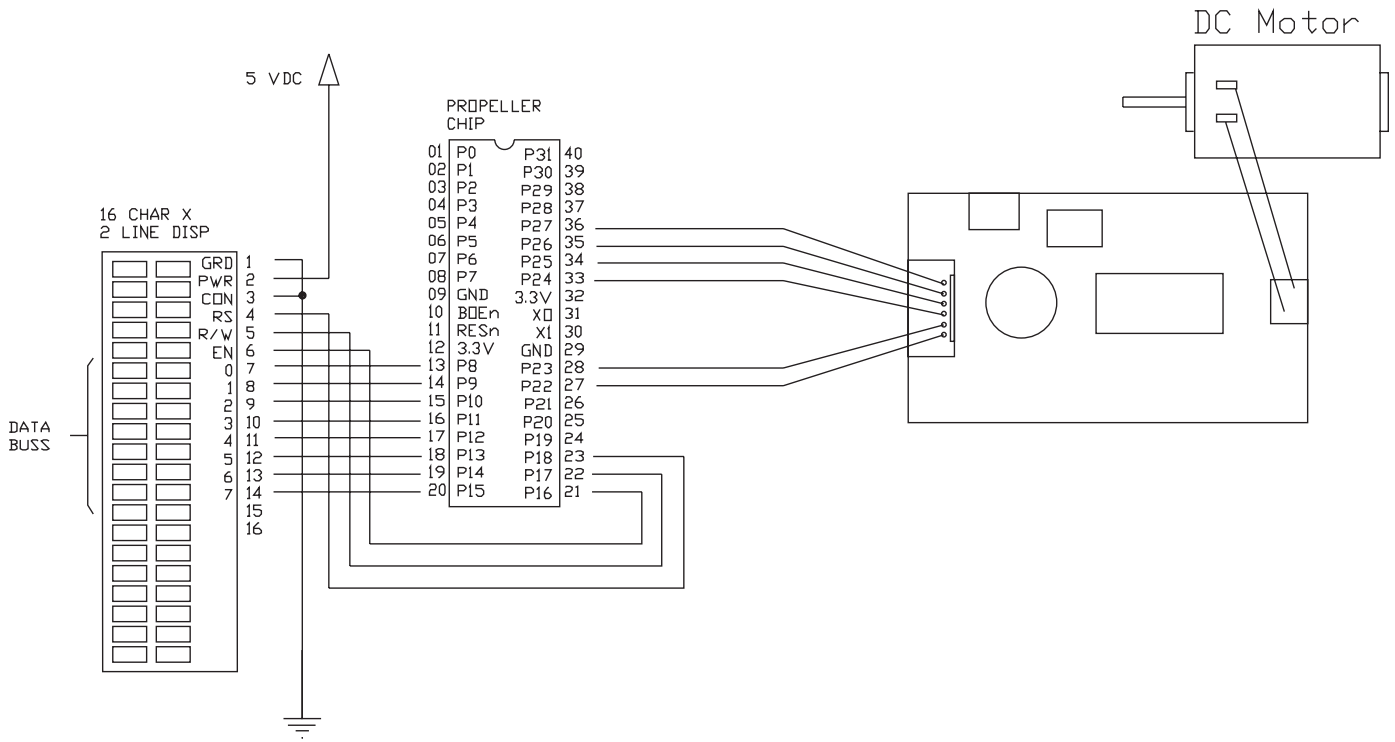


Figure 23-9 Wiring connections for Xavien single-axis amplifier

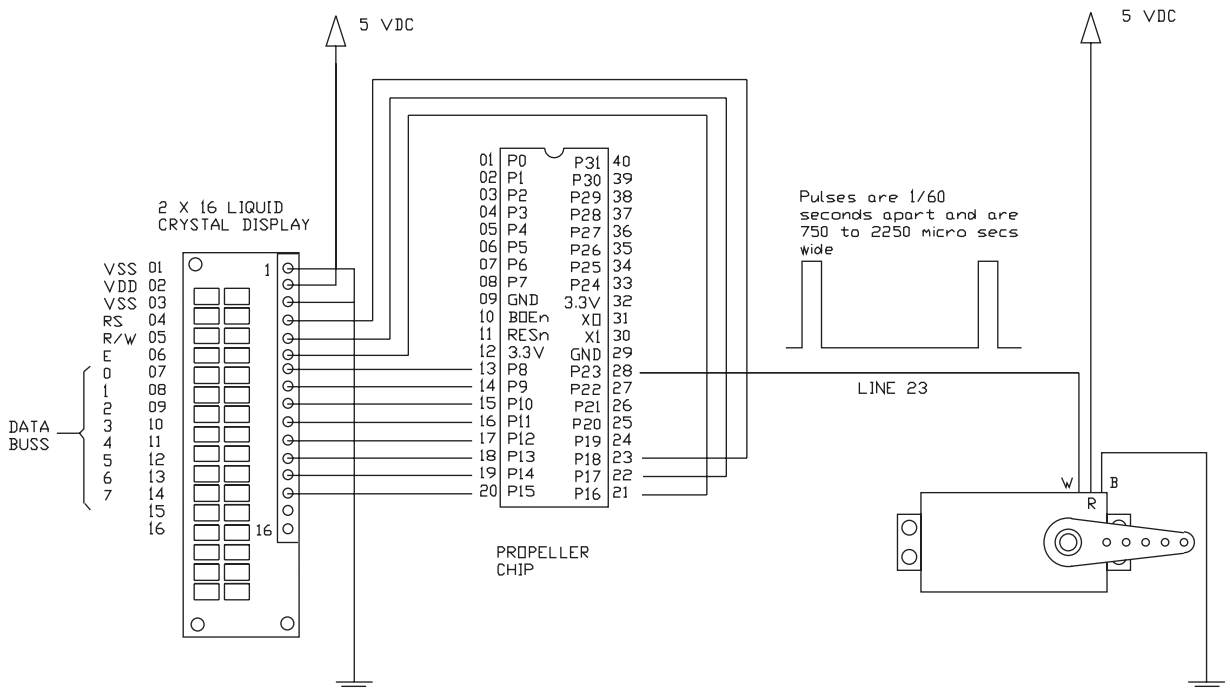


Figure 24-2 Wiring for running an R/C hobby servo from a Propeller

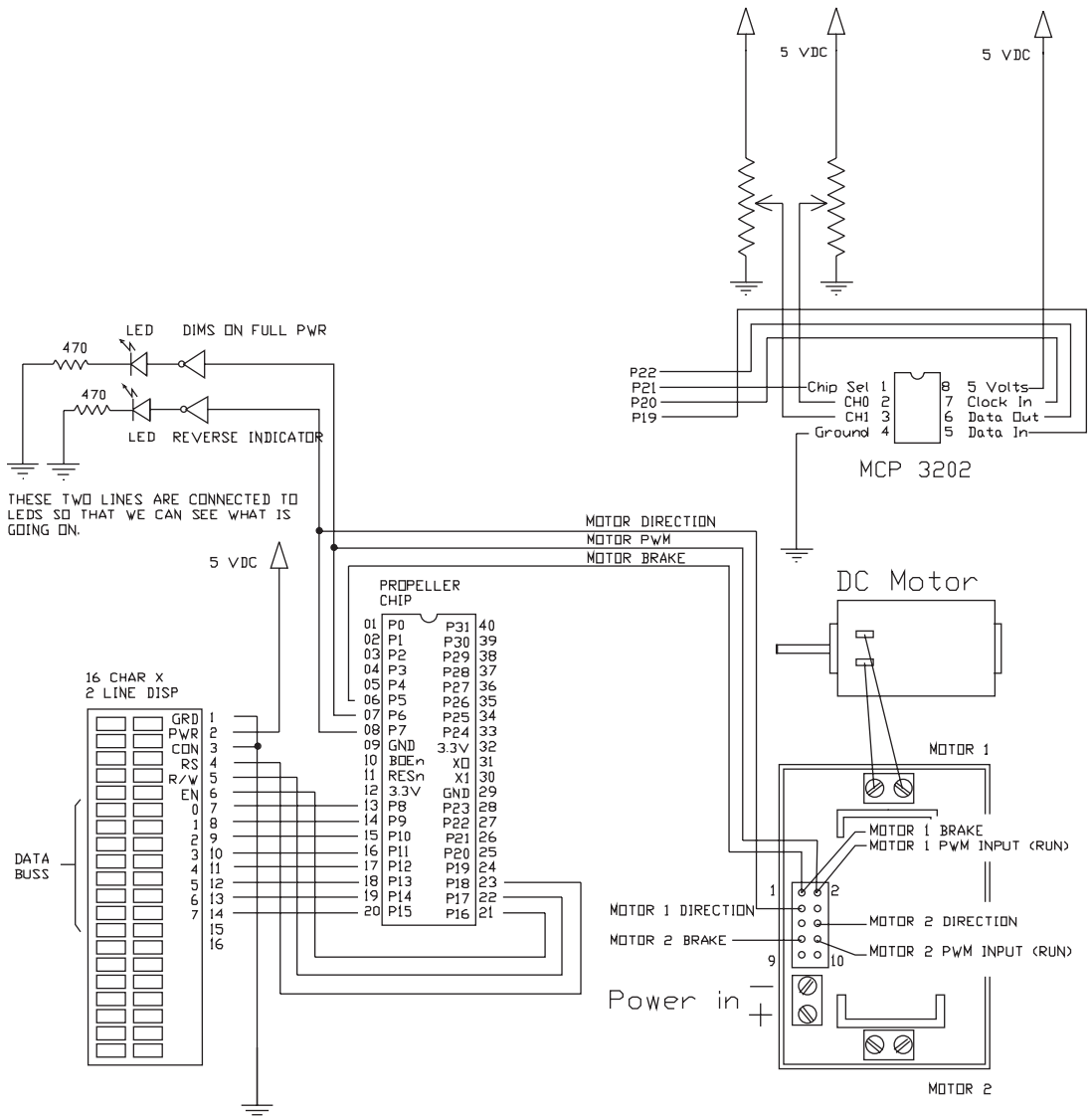


Figure 25-2 Wiring diagram for Propeller, potentiometer, and motor amplifier

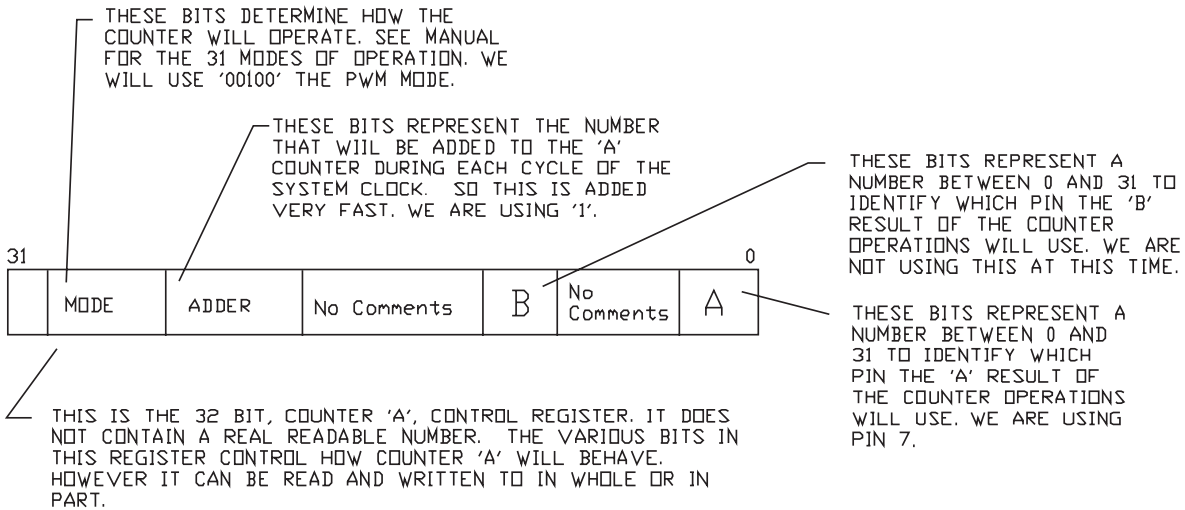


Figure 25-3 Diagram of the control register for CTRA bit assignments

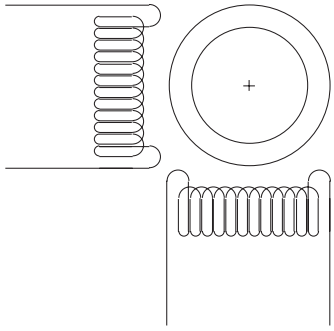


Figure 26-2 Wiring schematic for a stepper motor

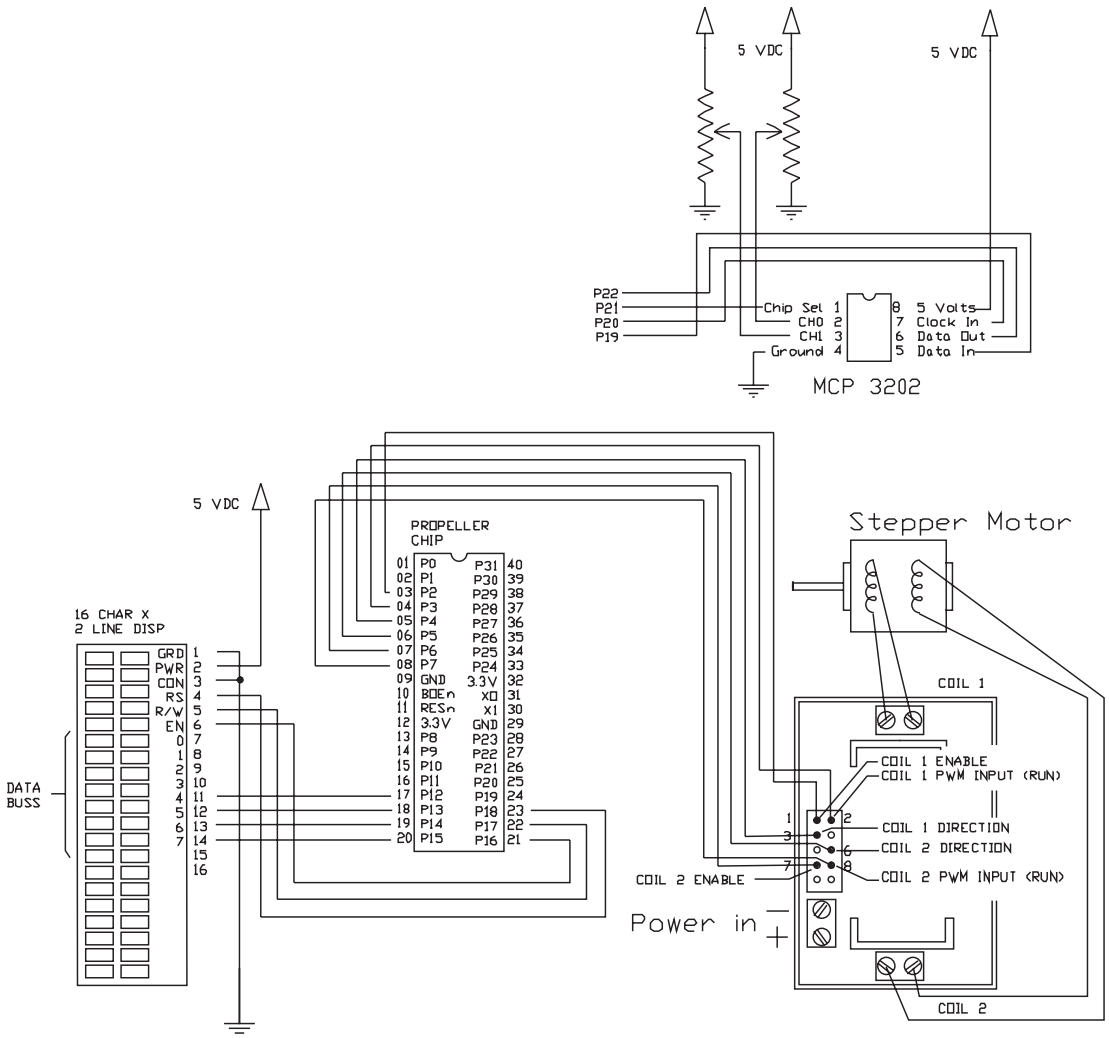


Figure 26-3 Wiring diagram for stepper motor control from potentiometers

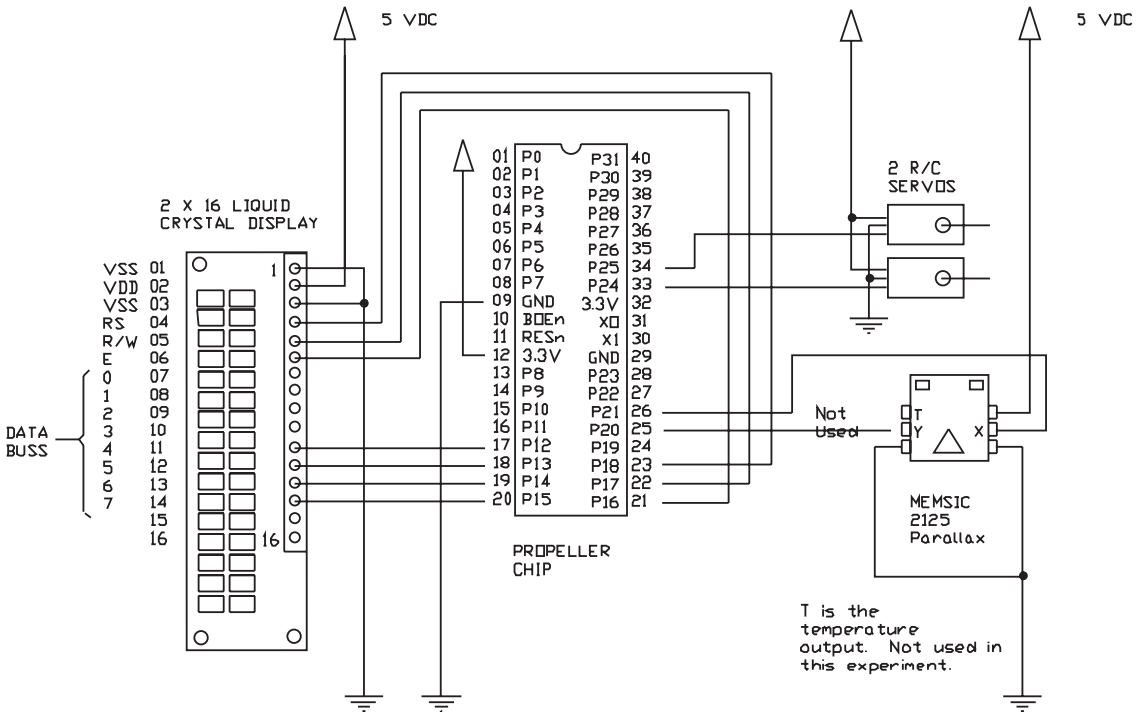


Figure 27-2 Wiring diagram for the table: connecting to the Memsic accelerometer to two servos

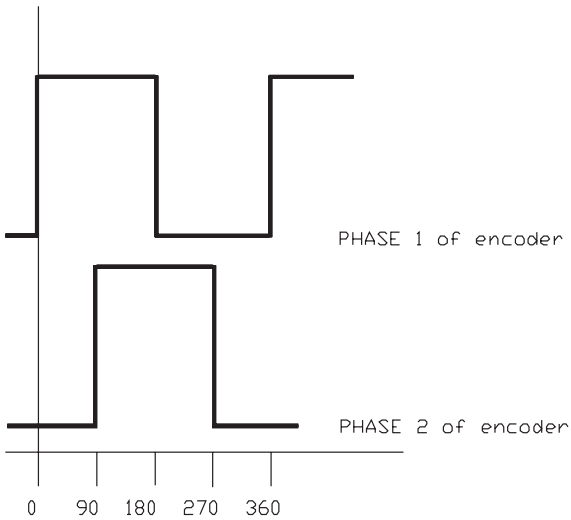


Figure 28-2 **Quadrature encoder signals. One signal leads the other by 90 degrees in a 360-degree cycle.**

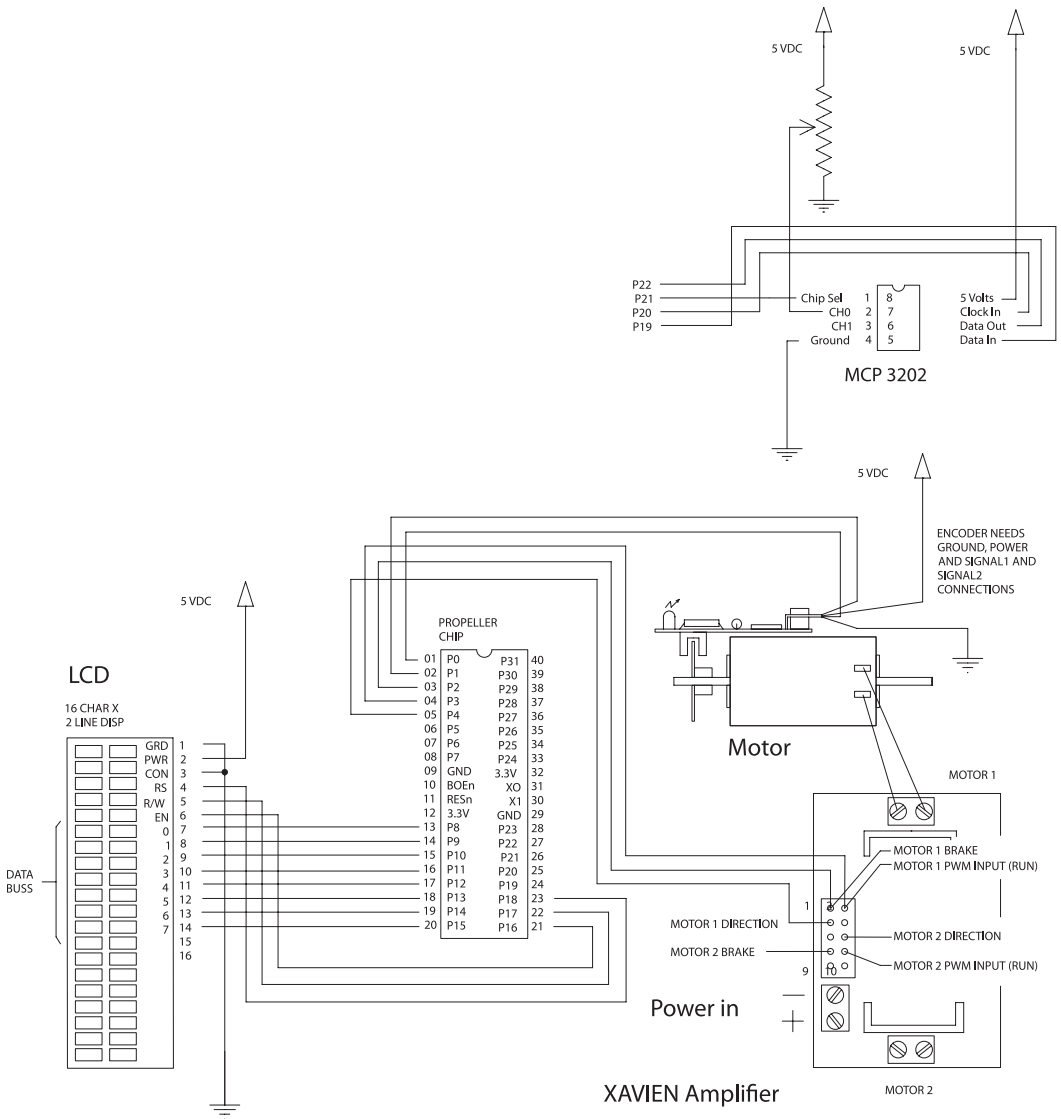


Figure 28-4 Wiring schematic for running a DC motor with an encoder

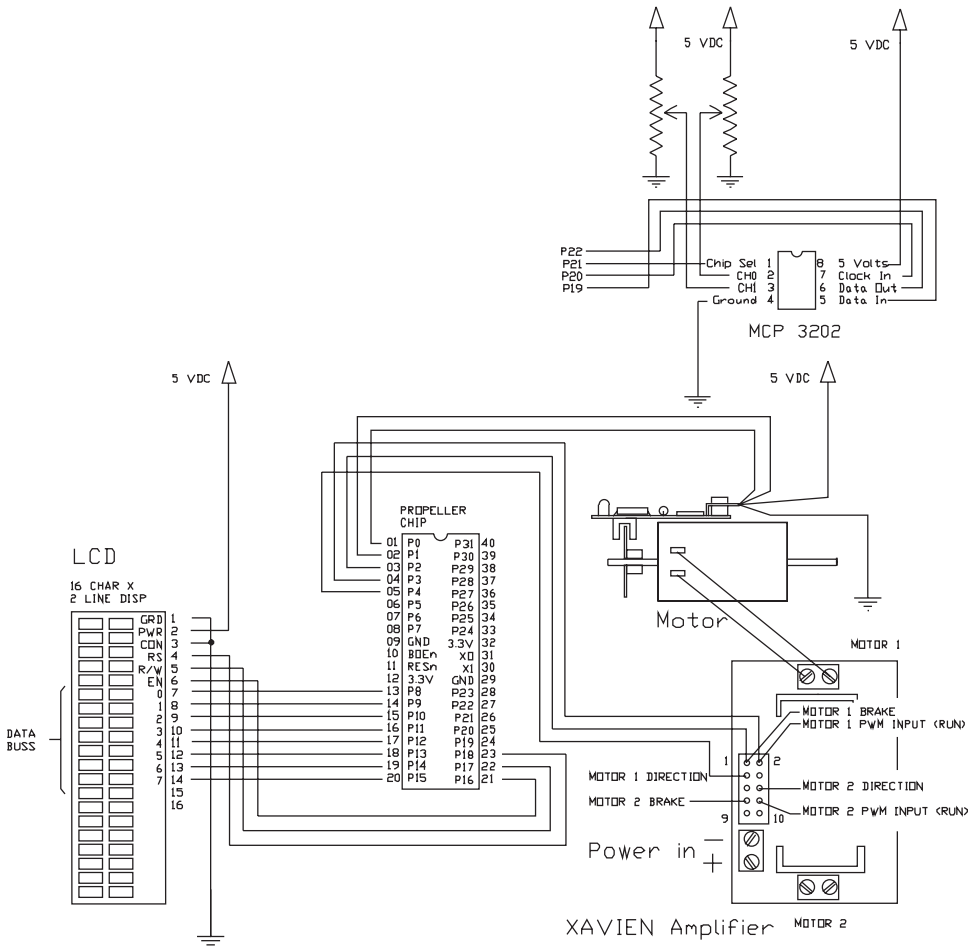


Figure 28-5 Two-potentiometer setup motor-control diagram

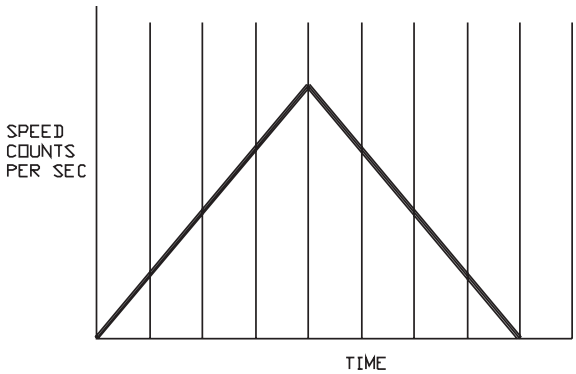


Figure 28-6 Simple ramp up/ramp down

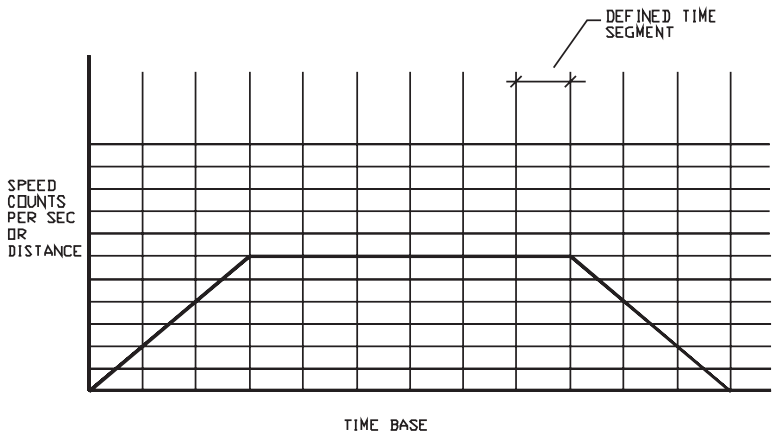


Figure 28-7 The speed/time path to be followed by the motor

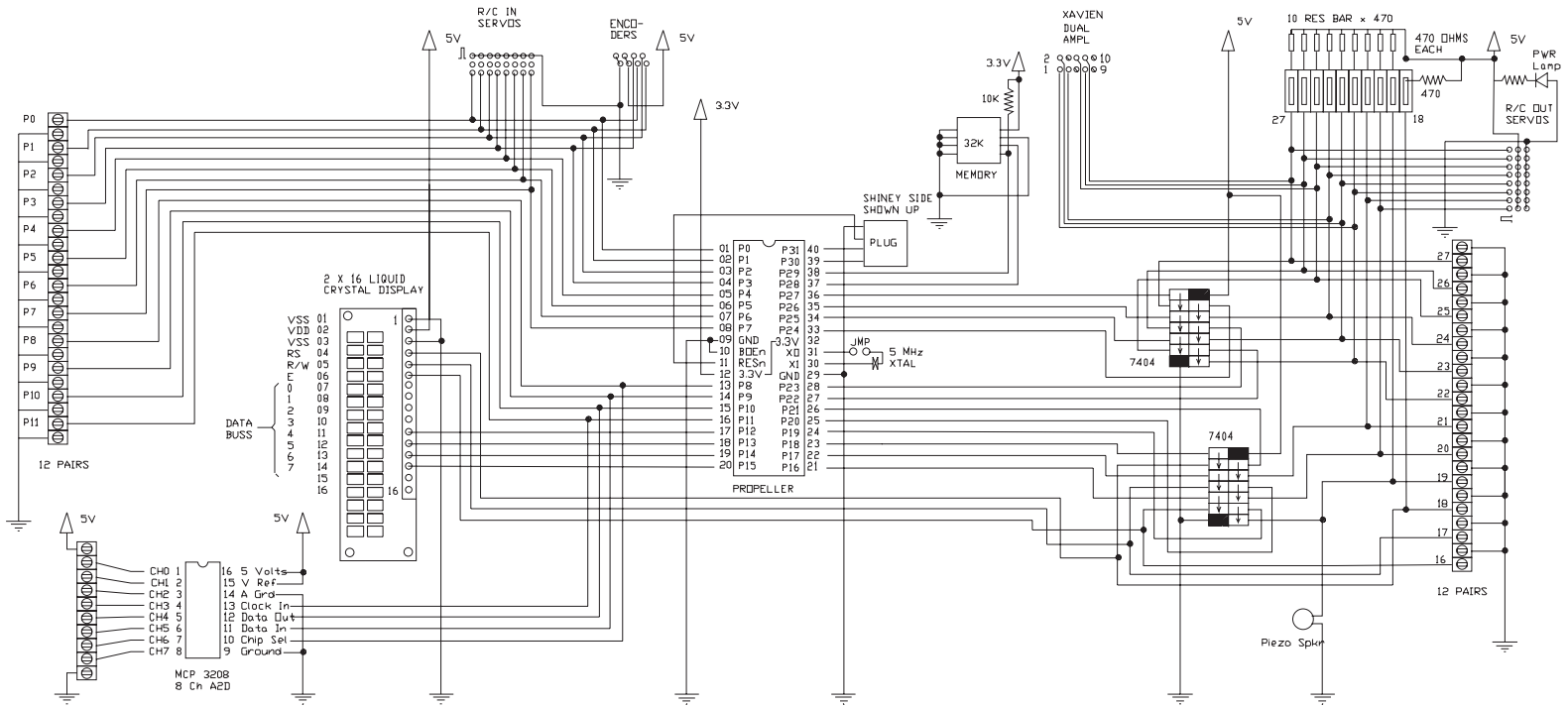


Figure D-1 Wiring schematic for experimental board. (Specifications are liable to change without notice.)