Figure 2-1 The pin designations for the 40-pin Propeller chip
For now we can disregard all the other control modules and interconnections between the various components.

**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

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 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
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 4-5** Code descriptions for the program to blink an LED
Figure 4-6 Assigning space for a new cog

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.

VAR
  long Stack[25]
  long pulseWidth

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

PUB MoveMotor(Pin)\CycleTime,period
  dira[Pin]~
  ctra[30:26]=\%0100
  ctra[5:0]=Pin
  freq=1
  PulsWidth=0
  CycleTime=\$\*freq/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 CDN 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 5-1 An overview of the Propeller memory banks
This is where you assign the space for each additional cog that will be opened. If enough space is not assigned, things will not run right. Start with 25 langs and then work it up or down down as needed.

This is where the first cog's program will reside. This is the main cog for now. You don't have to assign memory for the this cog.

This is where we create a new method and name it. Its start up and execution takes place once this is done. This is the name of this public (or private) method. It does not have to be "Cog_two". This was done as a naming convenience. This method's code follows this line.

This indented line will be executed once when this new cog starts. Then the repeat loop takes over. This method is called Cog_two and runs in the new cog. It manages the LCD.

This is the loop that the new cog runs. There can be more than one loop if there is a way to decide which one runs when. Other methods can be called within this method.

**Figure 5-2** Simple cog launch explanation, with no variables
**Figure 5-3** Advanced cog creation explanation, with variables.
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Propeller Application Code and Data 8192 longs</td>
</tr>
<tr>
<td>$7FFF</td>
<td></td>
</tr>
<tr>
<td>$8000</td>
<td>Character set 4096 longs 256 characters of 16 x 32 pixels</td>
</tr>
<tr>
<td>$BFFF</td>
<td></td>
</tr>
<tr>
<td>$C000 - $CFFF</td>
<td>Log Table (2048 words)</td>
</tr>
<tr>
<td>$D000 - $DFFF</td>
<td>Anti-Log Table (2048 words)</td>
</tr>
<tr>
<td>$E000 - $F001</td>
<td>Sin Table (2049 words)</td>
</tr>
<tr>
<td>$F002 - $FFFF</td>
<td>Boot Loader and Interpreter</td>
</tr>
</tbody>
</table>

**Figure 6-1** Propeller hub main memory map (from page 31 of Propeller Manual [Ver. 1.1])
The cycle time is maintained as constant and the pulse width within the wavelength is varied in a PWM operation.

Figure 7-2  Illustration of a PWM signal
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
This is where you assign the space for each additional cog that will be opened. If enough space is not assigned, things will not run right. Start with 25 longs starters and then work it up or down down as needed.

This is where the first cog’s program will reside. This is the main cog for now. You don’t have to assign memory for the this cog.

This is where we create a new Method and name it. Its start up and execution takes place once this is done.

This is the name of this public (or private) Method. It does not have to be ‘Cog_two’. This was done as a naming convenience. This Method’s code follows this line.

This indented line will be executed once when this new cog starts. Then the repeat loop takes over. This Method is called Cog_two and runs in the new cog. It manages the LCD.

This is the loop that the new cog runs. There can be more than one loop if there is a way to decide which one runs when. Other Methods can be called within this Method.

**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

T is the temperature output. Not used in this experiment.
Figure 18-3  Wiring the Memsic 2125 to a Propeller chip and an LCD
Figure 18-4  Waveforms of read and created pulse widths
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.
Figure 19-3  Actual segment connections to the Propeller
Each of these four lines is connected to the common line of one display (anode). They are selected one at a time.

These eight lines go in parallel to the eight LEDs on the seven-segment displays.

**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

These bits determine how the counter will operate. See manual for the 31 modes of operation. We will use '00100' the PWM mode.

These bits represent the number that will be added to the 'A' counter during each cycle of the system clock. So this is added very fast. We are using '1'.

These bits represent a number between 0 and 31 to identify which pin the 'B' result of the counter operations will use. We are not using this at this time.

These bits represent a number between 0 and 31 to identify which pin the 'A' result of the counter operations will use. We are using pin 7.

This is the 32 bit, counter 'A', control register. It does not contain a real readable number. The various bits in this register control how counter 'A' will behave. However it can be read and written to in whole or in part.
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.)