Chapter 6

In order to get effective use from our propellers chip we need to be able to interface any number of devices, that react with the real world, and provide us with information that we can use with microprocessor. Though there are a number of devices that you might start with but we going to start with the 3208 because it allows us to read potentiometers into the propellers chip. We will use the information that they provide to drive any number of devices from speakers to motors in our later experiments.

Take a close look at Figure 6-1 herein. Then refer it from time to time to see how the data transfer takes place. Also see page 16 of the data sheet Figure 5-1

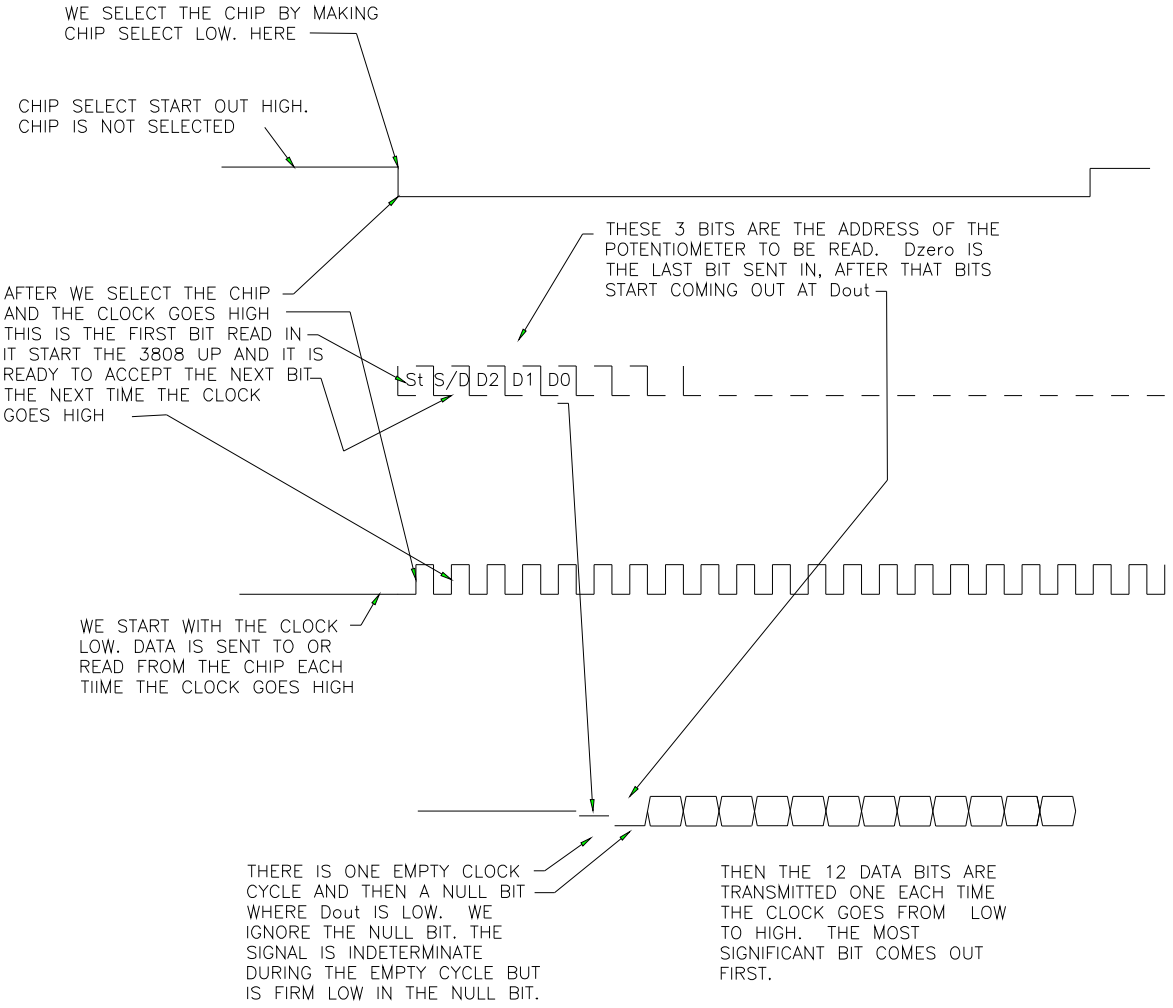

Figure 6-1

Segments of the program to read the 3208 are included in the following text. The entire program is listed at the end of the chapter and can be run on your

propeller. The program is also on line in the discussion forum if you want to downloaded it from there it will be easier. (Page 12 post #236)

The 3208 is capable of reading up to eight potentiometers one at a time at about 100,000 reading a second. We will place our potentiometers across 5 V and connect the wipers of the potentiometers to the eight input lines on the 3208. To start with, we will use only one potentiometer and it should be connected to Pin one.

Though we are connecting the potentiometer across 5 V it is not necessary that the potentiometers read across 5 V. There is a ground line, and a reference voltage line dedicated to the use with the input devices. The limitation is that since there is only one reference voltage line for all the potentiometers they all have to play be placed across this same voltage.

We are interested in potentiometers because a potentiometer is a device that is easily manipulated to provide a variable input. When we build other devices and connect them to a propellers chip for whatever purpose we may have in mind we can make the connections through the 3208 to provide the interface. The importance of a variable signal is to be appreciated, because we want to be able to make sure that we are actually reading or manipulating a changing signal. The signal may be an input or may be an output but in either case, we will have to see the results change in some way to make sure that the device is actually working. If nothing changes, not much can be deduced.

The 3208 is particularly well suited to our purpose or reading our first device, because the device is fairly easy to connect to and to use. Here is the procedure for reading the device. Follow along with the diagrams provided so that you can see exactly what we are going to do and how the system will respond.

There are four lines that control the operation of the 30 28. They are.
The clock line.
The data input line.
The data output line. We read the signal that comes out of this line
The chip select line.

The chip is dormant when the chip select line is high. We select the chip by making the chip select line low. When we make the chip select line low the 3028 responds by seeing it as a start signal for the whole next clock cycle.

```
DAT          org      0                    'sets the starting point in Cog
generate     mov      dira,   set_dira     'sets direction of the prop pins
             call     #chip_sel_lo         'selects chip by pulling line low
             call     #Clk_lo              'START. Clock needs to be low to load data
             call     #Din_hi              'must start with Din high to set up 3208
             call     #Tog_clk             'clk hi-lo to read data
```

```
              call      #Din_Hi                 'SINGLE DIFF  Low to load
              call      #Tog_Clk                'toggle clock line hi then low to read in the data
```

The next bit, we send it is a bit that selects the mode in which we want the 3208 to respond. For our purposes, we are interested in a single response and this is selected by making the data input line, low and toggling the clock chip high and then low it.

We next send out three the more bits. These bits identify one of the eight lines that we are going to read. A three bit signal can select one of the eight lines on the 3208. We will select line 0 in the initial experiment so the address we transmit to make the selection will be 000. Each line is impressed on the Din line and each time the clock is toggled high and then low one bits is read the the 3208.

```
              call      #Din_Lo       'D2 Low to load input line selection sequence 000 for line 0
             call      #Tog_Clk      'toggle clock line hi then low to read in the data
             call      #Din_Lo       'D1 Low to load input line selection sequence 000 for line 0
             call      #Tog_Clk      'toggle clock line hi then low to read in the data
             call      #Din_Lo       'D0 Low to load input line selection sequence 000 for line 0
             call      #Tog_Clk      'toggle clock line hi then low to read in the data
             call      #Din_Lo       'blank bit needs a clock cycle, next
             call      #Tog_Clk      'toggle clock line hi then low to read in the data
                                     'next toggle is for the null bit, nothing read
             call      #Tog_Clk      'toggle clock line hi then low to read in the data
```

Once the chip has accepted the three bit signal that the Din line goes into a don't care state. And we have no interest in it for the rest of the reading cycle.

Once the 3208 chip now knows, which line to read. It starts sending us the information about that line as 14 bits released by 14 clock cycles that we sent to the 3208. The first cycle provides indeterminate information and is to be ignored. The next bit as a low bit, to make sure we initiate our reading cycle properly. This bit is to be considered a null bit but it does tells us that the cycle has started. The next 12 bits are the data that we are interested in, and they are transmitted one bit at a time. Each bit arriving when the clock goes from low to high.

```
              mov       dat_red,  #0        'Clear register we will read data into
              mov       count,    #12       'Counter for number of bits we will read
read_bit      mov       temp,     ina       'read in what is in all the input lines
              andn      temp,     inputmask wz   'mask off everything except Dout line. Set Z flag
              shl       Dat_red,  #1        'shift reg left 1 bit to get ready for next bit
        if_nz add       Dat_red,  #1        'if value is still positive add 1 to data register
```

```
            call      #Tog_Clk            'toggle clock to get next bit ready in Dout
            sub       count,    #1 wz     'decrement the "bits read" counter. Set Z flag
        if_nz jmp     #read_bit           'go up and do it again if counter not yet 0
```

We read the bits by first clearing the register we are going to read into and then setting a counter to 12 to represent the 12 bits that we are going to read in. The bits are read by reading in the entire I/O register and then masking every bit except that the Dout bit from the 3208. If the masked answer is a one we add one to the register we are reading into and shift the whole register to the left one bit. If the red bit is a zero, we'd just shift all the bits left one bit. This makes the LSB in the register to zero. We do this 12 times and at the end of the 12 cycles. We have the reading from the potentiometer in our register.

```
            wrlong    dat_red,   par     'write it in PAR to share it as P.Val
            call      #Chip_Sel_Hi       'Put chip to sleep , for low power usage
            jmp       #generate          'go back to do it all again
```

We then write this information into the PAR register and it becomes available to the SPIN cog in our program, and we can use it for what ever we want. I have written in the code needed to send what is needed to the parallax serial terminal both as 12 bits binary and as a decimal quantity. As you manipulate the control knob of the potentiometer, the readings should go from 0 to 1111_11111111 on line 1 and from zero to 4095 on line 2.

```
B null | P_VAL
fds.start(31,30,0,115200)       'start console at 115200 for debug output
  cognew(@generate, @P_Val)     'start new cog at "generate" and read variable into P_Val
  cognew(oscope, @stack1)       'open cog to generate osc signals
  cognew(spkr, @stack2)         'open cog to generate speaker signals
  dira[0 ..11]~~                'sets 12 lines as outputs. 12 lines needed for 1.5 bytes
  repeat                        'loop
    global_value:=P_VAL         'endless loop to display data
    outa[0..11] := P_Val        'displays 1.5 bytes of data on the LEDs
    fds.bin(P_val,12)           'print value to the PST in binary to match LEDs
    fds.tx($d)                  'new line
    fds.dec(P_val)              'print value as decimal
    fds.tx("   ")               'spaces
    fds.tx($1)                  'home to 0,0
    waitcnt(clkfreq/60+cnt)     'flicker free wait
```

Here is the listing of the entire program.

```
{{
Program to read a pot
August 05 2011
Sandhu
Works.
Using a speaker or the o'scope is optional.
```

```
LEDs and serial terminal both show what is being read
}}
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  spkr_line=22
  osc_line=23

VAR
  long global_value
  long stack1[25]         'space for oscope
  long stack2[25]         'space for speaker

OBJ
  fds : "FullDuplexSerial"

PUB null | P_VAL
fds.start(31,30,0,115200)       'start console at 115200 for debug output
  cognew(@generate, @P_Val)     'start new cog at "generate" and read variable into P_Val
  cognew(oscope, @stack1)       'open cog to generate osc signals
  cognew(spkr, @stack2)         'open cog to generate speaker signals
  dira[0 ..11]~~                'sets 12 lines as outputs. 12 lines needed for 1.5 bytes
  repeat                        'loop
    global_value:=P_VAL         'endless loop to display data
    outa[0..11] := P_Val        'displays 1.5 bytes of data on the LEDs
    fds.bin(P_val,12)           'print value to the PST in binary to match LEDs
    fds.tx($d)                  'new line
    fds.dec(P_val)              'print value as decimal
    fds.tx("   ")               'spaces
    fds.tx($1)                  'home to 0,0
    waitcnt(clkfreq/60+cnt)     'flicker free wait

PRI oscope                      'oscilloscope output cog
dira [osc_line]~~               'set pin direcion as output
  repeat                        'loop
     !outa[osc_line]            'invert line
     waitcnt(clkfreq/(global_value+20)+cnt) 'wait suitable for osc view

PRI spkr                        'speaker oputput cog
dira [spkr_line]~~              'set pin direcion as output
  repeat                        'loop
    !outa[spkr_line]            'invert line
     waitcnt(clkfreq/(global_value+20)+cnt)  'wait suitable for speaker

DAT           org       0                     'sets the starting point in Cog
generate      mov       dira,   set_dira      'sets direction of the prop pins
              call      #chip_sel_lo          'selects chip by pulling line low
              call      #Clk_lo               'START. Clock needs to be low to load data
              call      #Din_hi               'must start with Din high to set up 3208
              call      #Tog_clk              'clk hi-lo to read data
```

```
              call      #Din_Hi                 'SINGLE DIFF  Low to load
              call      #Tog_Clk                'clock line hi then low to read in the data
              call      #Din_Lo                 'D2 Low to load input sel 000 for line 0
              call      #Tog_Clk                'clock line hi then low to read in the data
              call      #Din_Lo                 'D1 Low to load input seq 000 for line 0
              call      #Tog_Clk                'clock line hi then low to read in the data
              call      #Din_Lo                 'D0 Low to load line seq 000 for line 0
              call      #Tog_Clk                'clock line hi then low to read in the data
              call      #Din_Lo                 'blank bit needs a clock cycle, next
              call      #Tog_Clk                'lock line hi then low to read in the data
                                                'next toggle is for the null bit
              call      #Tog_Clk                'tlock line hi then low to read in the data
              mov       dat_red,  #0            'Clear register we will read data into
              mov       count,    #12           'Counter for number of bits we will read
read_bit      mov       temp,     ina           'read in what is in all the input lines
              andn      temp,     inputmask wz  'mask off except Dout line. Set Z flag
              shl       Dat_red,  #1            'shift reg left 1 bit, ready for next bit
        if_nz add       Dat_red,  #1            'if still positive add 1 to data register
              call      #Tog_Clk                'toggle clock to get next bit ready in Dout
              sub       count,    #1 wz         'decr the "bits read" counter. Set Z flag
        if_nz jmp       #read_bit               'go up and do it again if counter not yet 0
              wrlong    dat_red,   par          'write it in PAR to share it as P.Val
              call      #Chip_Sel_Hi            'Put chip to sleep by de selecting
              jmp       #generate               'go back to do it all again

'Subroutines
Clk_Hi        or        outa,   clk_bit         'OR it with the Clock Bit to make high
Clk_Hi_ret                ret                   'return from this subroutine

Clk_Lo        andn      outa ,   clk_bit        'ANDN it with the Clock Bi to make low
Clk_Lo_ret                ret                   'return from this subroutine

Tog_Clk       call      #Clk_hi                 'make clock bit high
              call      #clk_lo                 'make clock bit low
Tog_Clk_ret               ret                   'return from this subroutine

Din_Hi        or        outa ,   din_Bit        'Makes the Din high
Din_Hi_ret                ret                   'return from this subroutine

Din_Lo        andn      outa ,   din_Bit        'makes Din low
Din_Lo_ret                ret                   'return from this subroutine

Chip_Sel_Hi   or        outa ,   chs_Bit        'Makes Chip select high
Chip_Sel_Hi_ret           ret                   'return from this subroutine

Chip_Sel_Lo   andn      outa,   chs_Bit         'makes chip select low
Chip_Sel_Lo_ret           ret                   'return from this subroutine

Read_Next_Bit mov       temp,  ina              'Get the INA register
              or        temp,  inputmask        'mask all but Din bit
```

```
Read_Next_Bit_ret       ret                     'return from this subroutine

'Constants. This section is similar to the CON block in SPIN
Set_dira      long      %00001011_11000000_00001111_11111111   'Set dira register
Chs_Bit       long      %00000001_00000000_00000000_00000000   'Chip select bit 24
Din_Bit       long      %00000010_00000000_00000000_00000000   'Data in bit         25
Dout_Bit      long      %00000100_00000000_00000000_00000000   'Data out bit        26
Clk_Bit       long      %00001000_00000000_00000000_00000000   'Clock bit           27
inputmask     long      %11111011_11111111_11111111_11111111   'Mask for Dout bit only
Pin_23        long      %00000000_10000000_00000000_00000000   'osc line
Pin_22        long      %00000000_01000000_00000000_00000000   'Speaker line

'Variables. This section is similar to the VAR block in SPIN
temp          res       1       'temporary storage variable, misc
count         res       1       'temporary storage variable, read bit counter
Dat_Red       res       1       'temporary storage variable, data being read


My Comments

Everywhere 3208 is refereed to as chip in the PASM comments I think should read 3208.
```