APPS

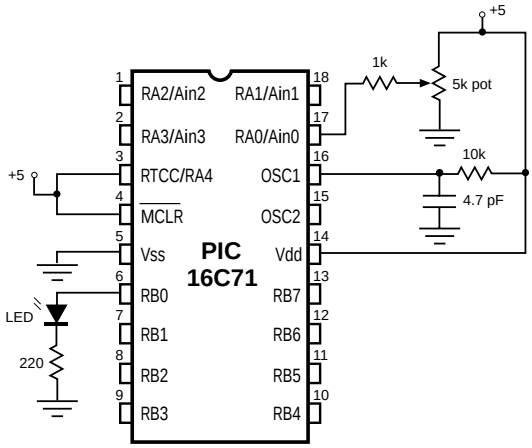
The PIC16C71 A/D Convertor

Introduction. This application note presents a program in TechTools assembly language that uses the PIC16C71's built-in analog-to-digital converter (ADC) to measure an input voltage and flash an LED at a proportional rate.

Background. One of the most popular enhancements offered by the new PIC16C71 is its eight-bit ADC, which features:

- 20-microsecond (µs) conversion time (nearly 50,000 samples per second, depending on additional processing time).
- Four multiplexed inputs.
- Built-in sample-and-hold.
- ±1 least-significant-bit accuracy (better than 20 millivolts with a 5-volt reference).
- Selectable voltage reference (Vdd or RA.3).

While using the ADC is fairly straightforward, it does require a series of decisions much like those required to select and use a separate ADC. The first consideration is hardware.



Input Characteristics. The ADC produces a digital output that is proportional to an analog input. A voltage reference determines the input voltage that will produce a full-scale (255) digital output. The voltage reference can be the +5-volt power-supply rail, or some other voltage source between 3 volts and the power supply voltage + 0.3 volts. The ADC is most accurate with a reference voltage of 5.12 volts, according to the manufacturer's specifications.

The specifications recommend that the analog voltage source being measured have an impedance of no more than 10ký. Above this value, accuracy suffers. They also suggest that the source have not less than 500ý impedance. This limits current through the PIC in the event that your program reconfigures the analog input pin as an output, or some other circuit trauma occurs.

Clock Source. The PIC's ADC, like the PIC itself, requires a clock signal. The ADC performs a conversion in 10 of its clock cycles, which must be no shorter than 2µs. Clock signals for the ADC can come from two sources, the PIC's own clock or an on-chip resistor-capacitor (RC) oscillator exclusive to the ADC.

When the PIC's clock is the source, it is divided by 2, 8 or 32, depending on the status of the ADC clock source bits (see figure 2). In order to have an ADC clock signal of 2µs or longer, the PIC clock speed must not exceed 1, 4, or 16 MHz, respectively. If you plan to run the PIC faster than 16 MHz, or you want the ADC conversion rate to be independent of the PIC clock, you must use the ADC's RC oscillator.

The tradeoff in using the RC oscillator is that its period can vary from 2 to 6µs, depending on temperature and manufacturing tolerances.

Interrupt Enable. The ADC is relatively slow—at 20 MHz the PIC can execute 100 instructions in the 20µs the ADC takes to make a conversion. In some cases, it makes sense not to force a PIC program to wait in a loop for a conversion to finish. The alternative is to configure the ADC to announce "conversion complete" through an interrupt. To keep things as simple as possible, the example program does not take advantage of interrupt capability.

Pin Configuration and Voltage Reference. Pins RA.0 through RA.3 can serve as inputs to the ADC. One of the choices you must make when setting up the ADC is which pins to configure as analog inputs, which (if any) as digital inputs, and what to use as a voltage reference. Figure 2 shows the range of available choices.

Note that the control register containing the configuration and voltage reference bits is in register page 1. To access it, you must first set bit *RP0*. The program listing shows how.

Input Selection. Only one of the pins configured for analog input can

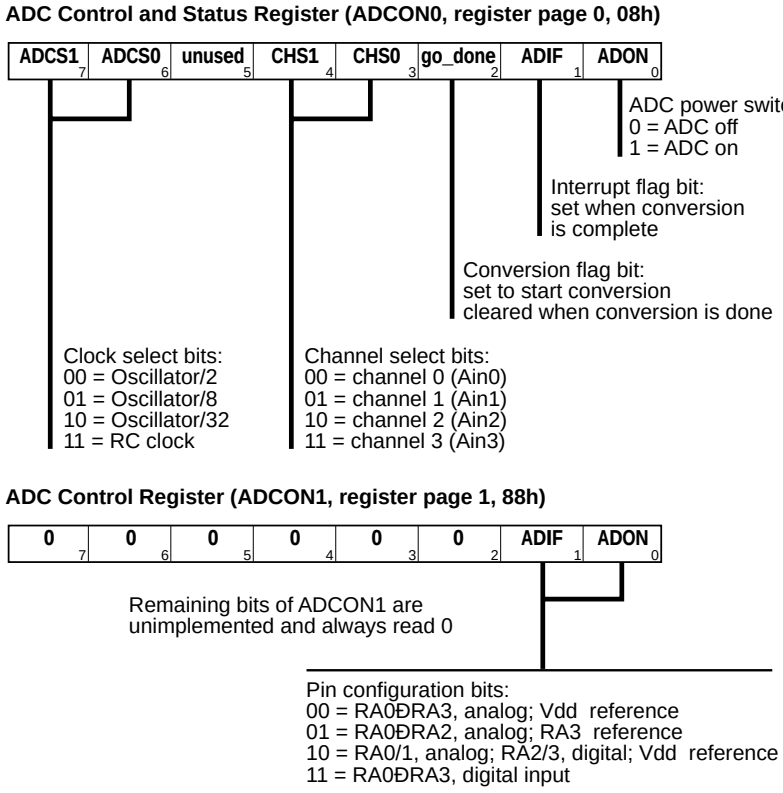


Figure 2. ADC control registers.

actually sample at any one time. In other words, if you want ADC input from two or more channels, your program must select a channel, wait long enough for the sample-and-hold circuit to charge up, command a conversion, get the result, and select the next channel...

How long should the program wait for the sample-and-hold? Microchip suggests a worst case of 4.67µs plus two ADC clock cycles (after a conversion, the sample-and-hold takes a two-cycle break before it begins sampling again).

How it works. The PIC in figure 1 accepts a voltage from a pot wired as variable voltage divider. The PIC's ADC, which is set up to use Vdd as a reference, outputs a one-byte value that's proportional to the input voltage. This value controls a timing routine that flashes an LED. When the input voltage is near 5 volts, the LED flashes about once a second. When it's near 0, the LED flashes very rapidly.

The program listing shows how it's done. Most of the code is devoted to setting up the ADC. Constants at the beginning of the program are assigned with values that, when loaded into the appropriate ADC control registers, turn the ADC's various features on and off. If you wish to change, for instance, the pin that the circuit uses for analog input, just comment out the line containing *AD_ch = 0* and uncomment the desired channel ("commenting" and "uncommenting" are handy techniques for temporarily removing and restoring instructions in source code. Putting a semicolon (;) in front of a line causes the assembler to ignore it, as though it were a comment).

If you accidentally leave two assignements for *AD_ch* uncommented, the assembler will catch the mistake, flag the "redefinition" and tell you the line number of the error.

The assembler combines values assigned to *ADC_ch* and *ADC_clk* into a single byte by performing a logical OR (|) on the values and putting the result into another constant, *ADC_ctl*. This technique makes the program easier to understand and modify, and doesn't cost a thing in PIC program memory or processing time. The assembler does all the work.

Program listing. This program may be downloaded from our Internet ftp

site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

; PROGRAM: Using the 16C71's analog-to-digital converter (ADC71.SRC)

```
; This program demonstrates use of the ADC in a simple circuit that samples a
; voltage and flashes an LED at a proportional rate. The header contains a number
; of constants representing setup constants for the ADC control registers.
; Uncomment the constant corresponding to the desired ADC setting.

                device  rc_osc,wdt_off,pwrt_off,protect_off
                id      'ADC1'

; The following constants set the ADC clock source and speed. Uncomment one.
;AD_clk         =       0               ; Oscillator x 2  (<= 1 MHz).
;AD_clk         =       64              ; Oscillator x 8  (<= 4 MHz).
;AD_clk         =       128             ; Oscillator x 32 (<= 16 MHz).
AD_clk          =       192             ; RC oscillator, 2–6 us.

; The following constants select a pin for ADC input. Uncomment one.
AD_ch           =       0               ; ADC channel 0 (Ain0, pin 17).
;AD_ch          =       8               ; ADC channel 1 (Ain1, pin 18).
;AD_ch          =       16              ; ADC channel 2 (Ain0, pin 1).
;AD_ch          =       24              ; ADC channel 3 (Ain0, pin 2).

AD_ctl          =       AD_clk | AD_ch  ; Logical OR.

; The following constants determine which pins will be usable by the ADC and
; whether Vdd or RA.3 will serve as the voltage reference. Uncomment one.
AD_ref          =       0               ; RA.0-3 usable, Vdd reference.
;AD_ref         =       1               ; RA.0-3 usable,RA.3 reference.
;AD_ref         =       2               ; RA.0/1 usable, Vdd reference.
;AD_ref         =       3               ; All unusable—digital inputs only.

                org     0Ch
counter1        ds      1
counter2        ds      1

; Set starting point in program ROM to zero. Jump past interrupt vector to beginning
; of program. (This program doesn't use interrupts, so this is really not needed, but
it
; will be easier to add interrupts later if required.)
                org     0
                jmp     5
                org     5
```

```
start       mov     !ra, #255           ; Set RA to input.
            mov     !rb, #0             ; Set RB to output.
            mov     intcon, #0          ; Turn interrupts off.
            mov     adcon0,#AD_ctl      ; Set AD clock and channel.
            setb    rp0                 ; Enable register page 1.
            mov     adcon1,#AD_ref      ; Set usable pins, Vref.
            clrb    rp0                 ; Back to register page 0.
            setb    adon                ; Apply power to ADC.

:loop       call    wait                ; Delay for time determined by
                                        ; ADC input.
            setb    rb.0                ; Turn LED on.
            call    wait                ; Delay for time determined by
                                        ; ADC input.
            clrb    rb.0                ; Turn LED off.
            goto    :loop               ; Endless loop.

wait        setb    go_done             ; Start conversion.
:not_done   snb     go_done             ; Poll for 0 (done).
            jmp     :not_done           ; If 1, poll again.
            mov     counter2,adres      ; Move ADC result into counter.

; The number of loops this delay routine makes is dependent on the result of the AD
; conversion. The higher the voltage, the longer the delay.
:loop       djnz    counter1, :loop
            djnz    counter2, :loop
            ret
```