Preface to the Second Edition

A second edition of this book has given me the opportunity to respond to suggestions from both students and correspondents from around the world, from disparate regions ranging from Scotland to Hawaii. Since the time of the first edition written in the late 1990s, the Microchip PIC range has become the largest volume selling 8-bit MCU. The mid-range family used in the original edition has continued to expand vigorously, with some of the exemplars used becoming essentially obsolete. In addition, the enhanced-range 16-bit instruction line has been enlarged from virtually nothing to form a significant proportion of the family. At the same time, new introductions to the original low- (or base-) end architecture continue apace. Because of the close relationship between the low-, mid-, high- and enhanced-range lines, the focus of the new edition has stayed with the mid-range line up.

Virtually all diagrams have been modified, many extensively, and numerous additional new figures have been added. Throughout the text, special attention has been paid to clarify the basic concepts. In Part I, Chapter 3 has been extensively rewritten with this in mind and to better integrate with Chapters 4 and 5 in Part II, both of which bear only a superficial relation to the original text. Chapter 7, covering interrupt handling, has also been largely rewritten to elucidate a difficult topic. Part III not only has been revised to use current exemplars, but has been extended to cover additional peripherals such as the Analog Comparator and Voltage Reference modules. A new chapter covers the enhanced-range PIC18FXXX range.

With the exception of the first two and last chapters, all chapters have both fully worked examples and self-assessment questions. As an extension to this, an associated Web site at

`http://www.engj.ulst.ac.uk/sidk/quintessential`

has the following facilities:

- Solutions to self-assessment questions.
- Further self-assessment questions.
- Additional material.
- Source code for all examples and questions in the text.
- Pointers to development software and data sheets for devices used in the book.

- Errata.
- Feedback from readers.

The manuscript was typeset by the author on a variety of Microsoft® Windows™ PCs using a Y&Y implementation of LaTeX 2$_\varepsilon$ and the Lucida Bright font family. Line drawings were created or modified with Autocad R13 and incorporated as encapsulated PostScript files. Photographs were taken by the author using several Olympus digital cameras—which are absolutely full of microcontrollers!

Hopefully, any gremlins have been exorcised, but if you find any or have any other suggestions, I will be happy to acknowledge such communications via the Web site.

Sid Katzen
University of Ulster at Jordanstown
July 2005

Preface to the First Edition

Microprocessors and their microcontroller derivatives are a widespread, if rather invisible, part of the infrastructure of our twenty-first-century electronic and communications society. In 1998, it was estimated[1] that hidden in every home there were about 100 microcontrollers and microprocessors: in the singing birthday card, washing machine, microwave oven, television controller, telephone, personal computer and so on. About 20 more lurked in the average family car, for example, monitoring in-tire radio pressure sensors and displaying critical data through a control area network (CAN).

Around 4 billion such devices are sold each year to implement the intelligence of these "smart" electronic devices, ranging from smart egg-timers through to aircraft management systems. The evolution of the microprocessor from the first Intel device introduced in 1971 has revolutionised the structure of society, effectively creating the second Industrial Revolution at the beginning of the twenty-first century. Although the microprocessor is better known for its role in powering the ubiquitous PC, where raw computing power is the goal, sales of microprocessors such as the Intel Pentium represent only around 2% of the total volume. The vast majority of sales are of low-cost microcontrollers embedded into a dedicated-function digital electronic device, such as the smart card. Here the emphasis is on the integration of the core processor with memory and input/output resources in the one chip. This integrated computing system is known as a *microcontroller*.

In seeking to write a book in this area, the overall objective was to get the reader up-to-speed in designing small embedded microcontroller-based systems, rather than using microcontrollers as a vehicle to illustrate computer architecture in the traditional sense. This will hopefully give the reader confidence that, even at such an introductory level, he/she can design, construct, and program a complete working embedded system.

Given the practical nature of this material, real-world hardware and software products are used throughout to illustrate the material. The microcontroller market is dominated by devices that operate on 8-bit data (although 4- and 16-bit examples are available) like early microprocessors and unlike the 64-bit Intel Pentium and Motorola Power PC

[1] *New Scientist*, vol. 59, no. 2141, 4 July 1998, p. 139.

"heavy brigade". In contrast, the essence of the microcontroller lies in its high system-integration/low-cost profile. Power can be increased by distributing processors throughout the system. Thus, for example, a robot arm may have a microcontroller for each joint implementing simple local processes and communicating with a more powerful processor making overall executive decisions.

In choosing a target architecture, acceptance in the industrial market, easy availability, and low-cost development software have made the Microchip family one of the most popular choices as the pedagogic vehicle in learning microprocessor/microcontroller technology at all levels of electronic engineering from grade school to university. In particular, the reduced instruction set, together with the relatively simple innovative architecture, reduces the learning curve. In addition to their industrial and educational roles, the PIC® MCU families are also the mainstay of hobbyist projects, as a leaf through any electronics magazine will show.

Microchip, Inc., is a relatively recent entrant to the microcontroller market with its family of Harvard architecture PIC devices introduced in 1989. By 1999, Microchip was the second largest producer of 8-bit units—behind only Motorola.

This book is split into three parts. Part I covers sufficient digital, logic and computer architecture to act as a foundation for the microcontroller engineering topics presented in the rest of the text. Inclusion of this material makes the text suitable for stand-alone usage, as it does not require a prerequisite digital systems module.

Part II looks mainly at the software aspects of the mid-range PIC microcontroller family, its instruction set, how to program it at assembly and high-level **C** coding levels, and how the microcontroller handles subroutines and interrupts. Although the 14-bit PIC family is the exemplar, both architecture and software are comparable to both the 12- and 16-bit ranges.

Part III moves on to the hardware aspects of interfacing and interrupt handling, with the integration of the hardware and software being a constant theme throughout. Parallel and serial input/output, timing, analog, and EEPROM data-handling techniques are covered. A practical build and program case study integrates the previous material into a working system, as well as illustrating simple testing strategies.

Sid Katzen
University of Ulster at Jordanstown
December 2000

CHAPTER 14

Take the Rough with the Smooth

Given that digital microcontrollers are in the business of monitoring and controlling the real environment—which is commonly analog in nature—we need to consider the interaction between the analog and the digital worlds. In some cases all that is required is a **comparison** of two analog voltage levels. However, for more sophisticated situations, analog input signals need conversion to a digital equivalent; that is, **analog-to-digital conversion** (**ADC**). Thereafter the digital patterns can be processed in the normal way. Conversely, if the outcome is to be in the form of an analog signal, then a **digital-to-analog conversion** (**DAC**) stage will be necessary.

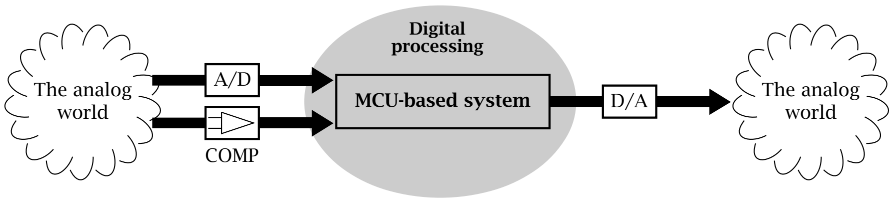


Fig. 14.1 Analog world—digital processing.

Of these various processes, illustrated in Fig. 14.1, A/D conversion is by far the more complex. Many PIC microcontrollers feature integral multi-channel A/D facilities. However, analog outputs frequently require external circuitry to implement the D/A process.

In this chapter we will look at the properties of analog and digital signals and the conversion between them as relevant to the PIC MCU. After completion you will:

- Understand the quantization relationship between analog and digital signals.
- Be aware of the need to sample an analog signal at least twice the highest frequency component.
- Appreciate how the successive approximation technique can convert an analog voltage to a binary equivalent.

- Understand the operation and be able to configure the Analog Comparator, Voltage Reference and ADC modules.
- Know how to configure I/O pins as either analog or digital.
- Be able to write assembly-level programs to acquire analog data using polling, interrupt-driven, and Sleep techniques, and to interrogate the state of the analog comparators.
- Be able to code high-level **C** programs to set-up and interact with the various analog modules.
- Know how to parallel interface to a proprietary DAC.

The information content of an **analog signal** lies in the continuously changeable worth of some constituent parameters, such as amplitude, frequency, or phase. Although this definition implies that an analog variable is a continuum in the range $\pm\infty$, in practice its range is restrained to an upper and lower limit. Thus a mercury thermometer may have a continuous range between, say, -10°C and $+180^\circ$C. Below the bottom number mercury disappears into the bulb. Above the highest number and the top of the tube is blown off!

Theoretically the quantum nature of matter sets a lower bound to the smooth continuous nature of things. However, in practice noise levels and the limited accuracy of the device generating the signal sets an upper limit to the resolution that processing needs to take account of.

Digital signals represent their information content in the form of arrangements of discrete characters. Depending on the number and type of symbols making up the patterns, only a finite totality of value portrayals are possible. Thus in a binary system, an n-digit pattern can at the most represent 2^n levels. Although this *rough* grainy view of the world seems inferior to the infinity of levels that can be *smoothly* represented by an analog equivalent, the quantizing grid can be tailored to fit the accuracy of the task to be undertaken. For instance, a telephone speech circuit will tolerate a resolution of around 1%. This can use an 8-bit depiction, which gives up to 256 discrete values with a corresponding $\approx 0.5\%$ resolution. A music compact disk uses a 16-bit scheme, giving a one part in 65,636 grid—around 0.0015% resolution.

From this discussion it can be seen that any process involving interconversion between the analog and digital domains will involve transition through the **quantization** state. Therefore we need to look at how this affects the information content of the associated signals.

As an example, consider the situation shown in Fig. 14.2, where an input range is represented as a 3-bit code. In essence the process of quantizing a signal is the comparison of the analog value with a fixed number of levels—eight in this case. The nearest level is then taken as expressing the original as its digital equivalent. Thus in Fig. 14.2 an input voltage of 0.4285 of full scale is 0.0536 above quantum level 3. Its quantized value will then be taken as level 3 and coded as b'011' in our 3-bit scheme of things.

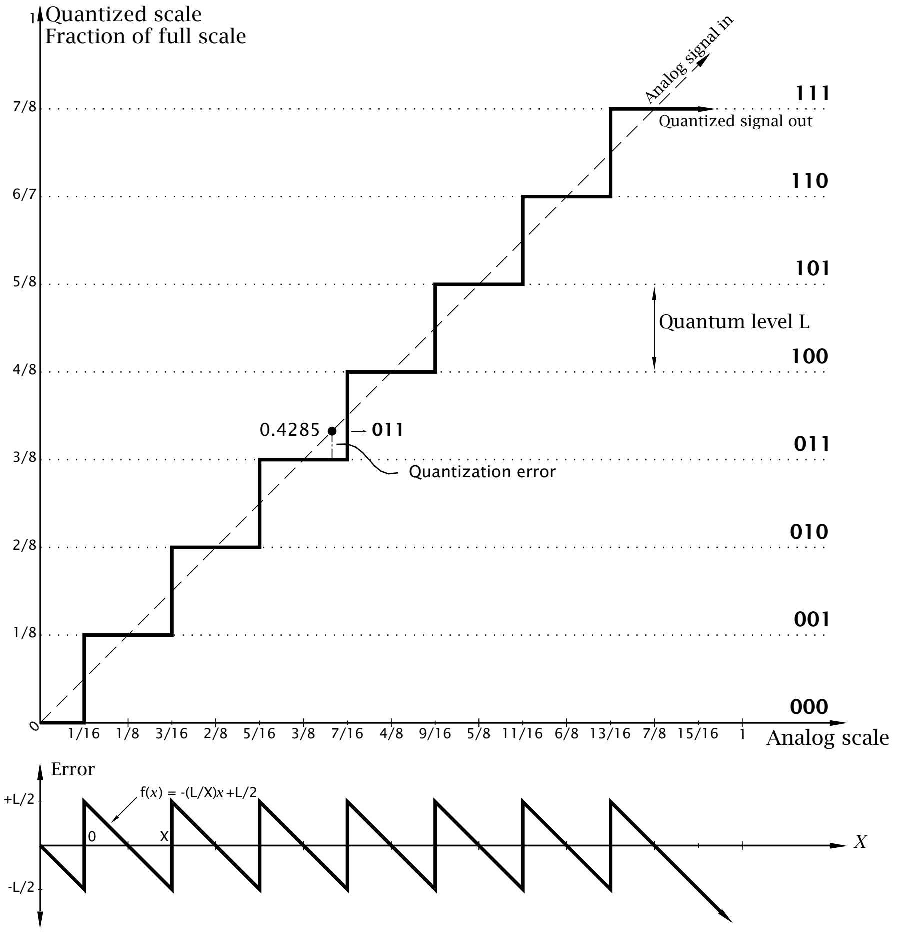


Fig. 14.2 The quantizing process.

The residual error of −0.0536 will remain as quantizing noise, and can never be eradicated; see also Fig. 14.3(d). The distribution of quantization error is given at the bottom of Fig. 14.2, and is affected only by the number of levels. This can simply be calculated by evaluating the average of the error function squared. The square root of this is then the root mean square (rms) of the noise.

$$\mathcal{F}(x) = -\frac{L}{X}x + \frac{L}{2}$$

The mean square is:

$$\begin{aligned}\frac{1}{X}\int_0^X \mathcal{F}(x)^2\,dx &= \frac{1}{X}\int_0^X\left[\frac{L^2}{X^2}x^2-\frac{L^2}{X}x+\frac{L^2}{4}\right]dx\\ &= \frac{1}{X}\left|\frac{L^2}{3X^2}x^3-\frac{L^2}{2X}x^2+\frac{L^2}{4}x\right|_0^X=\frac{L^2}{12}\end{aligned}$$

Thus the rms noise value of $\frac{L}{\sqrt{12}}=\frac{L}{2\sqrt{3}}$, where L is the quantum level.

A fundamental measure of a system's merit is the signal-to-noise ratio. Taking the signal to be a sinusoidal wave of peak to peak amplitude 2^nL, we have an rms signal of $\frac{\left(\frac{2^nL}{2}\right)}{\sqrt{2}}$; that is, $\frac{\text{peak}}{\sqrt{2}}$. Thus for a binary system with n binary bits, we have a signal-to-noise ratio of:

$$\frac{\left(\frac{2^nL}{2\sqrt{2}}\right)}{\left(\frac{L}{\sqrt{12}}\right)}=\frac{2^n\sqrt{12}}{2\sqrt{2}}=1.22\times 2^n$$

In decibels we have:

$$\text{S/N} = 20\log 1.22\times 2^n = (6.02n+1.77)\,\text{dB}$$

The dynamic range of a quantized system is given by the ratio of its full scale (2^nL) to its resolution, L. This is just 2^n, or in dB, $20\log 2^n = 20n\log 2 = 6.02n$. The percentage resolution given in Table 14.1 is of course just another way of expressing the same thing.

Table 14.1: Quantization parameters.

Binary bits n	**Quantum levels** (2^n)	**% resolution**	**Resolution dynamic range**	**S/N ratio (dB)**
4	16	16.25	24.1 dB	26.9 dB
8	256	0.391	48.2 dB	49.9 dB
10	1024	0.097	60.2 dB	61.9 dB
12	4096	0.024	72.2 dB	74.0 dB
16	65,536	0.0015	96.3 dB	98.1 dB
20	1,048,576	0.00009	120.4 dB	122.2 dB

The exponential nature of these quality parameters with respect to the number of binary-word bits is clearly seen in Table 14.1. However, the implementation complexity and thus price also follows this relationship. For example, a 20-bit conversion of 1 V full scale would have to deal with quantum levels less than 1 μV apart. Pulse-code modulated telephonic links use eight bits, but the quantum levels are unequally spaced, being closer at the lower amplitude levels. This reduces quantization hiss where conversations are held in hushed tones! Linear 8-bit conversions are suitable for most general purposes, having a resolution of better than $\pm\frac{1}{4}\%$. Actually video looks quite acceptable at a 4-bit resolution, and music can just about be heard using a single bit—i.e., positive or negative!

S/N ratios presented in Table 14.1 are theoretical upper limits, as errors in the electronic circuitry converting between representations and aliasing (discussed below) will add distortion to the transformation.

The analog world treats time as a continuum, whereas digital systems sample signals at discrete intervals. Shannon's sampling theorem[1] states that provided this interval does not exceed half that of the highest signal frequency, then no information is lost. The reason for this theoretical twice highest frequency sampling limit, called the Nyquist rate, can be seen by examining the spectrum of a train of amplitude modulated pulses. Ideal impulses (pulses with zero width and unit area) are characterized in the frequency domain as a series of equal-amplitude harmonics at the repetition rate, extending to infinity. Real pulses have a similar spectrum but the harmonic amplitudes fall with increasing frequency.

If we modulate this pulse train by a baseband signal $A\sin\omega_f t$, then in the frequency domain this is equivalent to multiplying the harmonic spectrum (the pulse) by $A\sin\omega_f t$, giving sum and different components thus:

$$A\sin\omega_f t \times B\sin\omega_h t = \frac{AB}{2}(\sin(\omega_h + \omega_f)t + \sin(\omega_h - \omega_f)t)$$

for each of the harmonic frequencies ω_h.

More complex baseband signals can be considered to be a band-limited (f_m) collection of individual sinusoids, and on the basis of this analysis, each of these pulse harmonics will sport an upper (sum) and lower (difference) sideband. We can see from the geometry of Fig. 14.3(b) that the harmonics (multiples of the sampling rate) must be spaced at least $2\times f_m$ apart, if the sidebands are not to overlap.

A low-pass filter can be used, as shown in Fig. 14.3(d), to recover the baseband from the pulse train. Realizable filters will pass some of the harmonic bands, albeit in an attenuated form. A close examination of the frequency domain of Fig. 14.3(d) shows a vestige of the first lower sideband appearing in the pass band. However, most of the distortion in the reconstituted analog signal is due to the quantizing error resulting from the crude 3-bit digitization. Such a system will have a S/N ratio of around 20 dB.

In order to reduce the demands of the recovery filter, a sampling frequency somewhat above the Nyquist limit is normally used. This introduces a guard band between sidebands. For instance, the pulse code telephone network has an analog input band limited to 3.4 kHz, but is sampled at 8 kHz. Similarly the audio compact disk uses a sampling rate of 44.1 kHz, for an upper music frequency of 20 kHz.

[1]Shannon, C.E.; *Communication in the Presence of Noise*, Proc. IRE, vol. 37, Jan. 1949, pp. 10–21.

A more graphic illustration of the effects of sampling at below the Nyquist rate is shown in Fig. 14.4. Here the sampling rate is only 0.75 of the baseband frequency. When the samples are reconstituted by filtering, the resulting pulse train, the outcome—shown in Fig. 14.4(b)—bears no simple relationship to the original. This spurious signal is known as an **alias.** Where an input analog signal has frequency components *above* half the sampling rate, maybe due to noise, then this will appear as distortion in the reconstituted signal. For this reason analog signals are usually

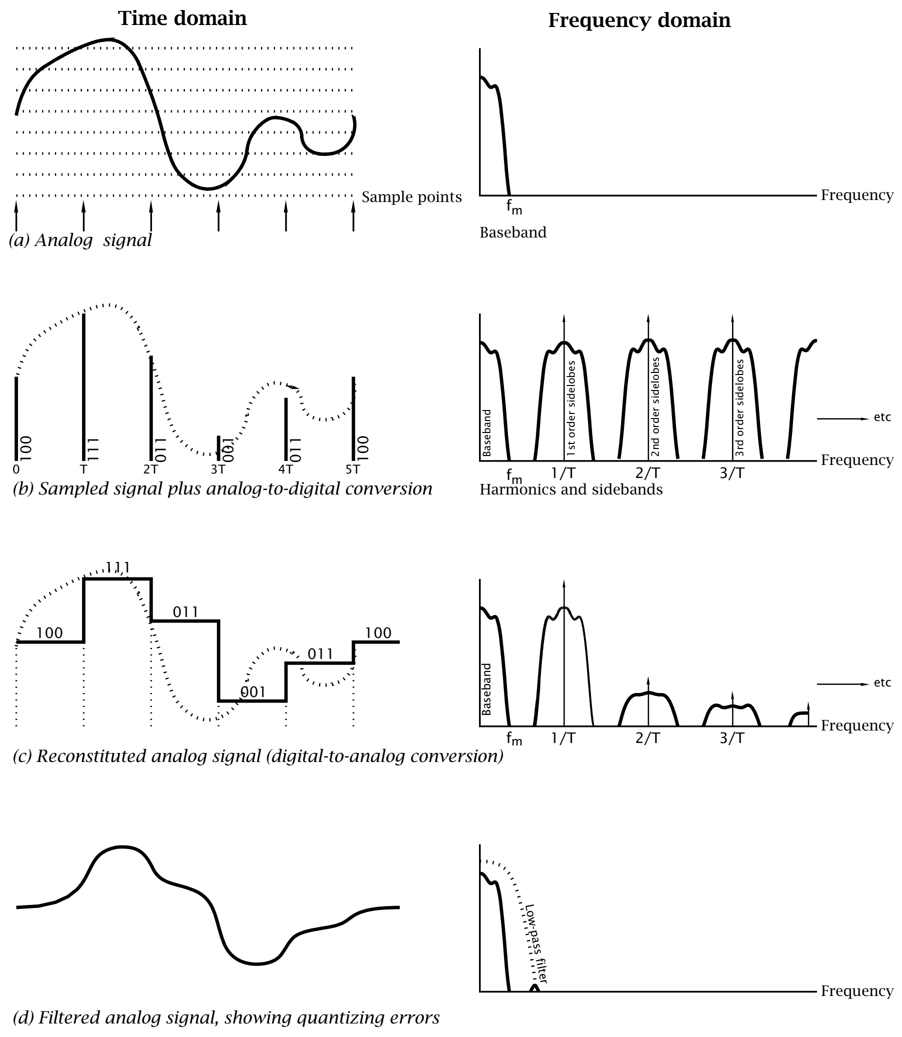


Fig. 14.3 The analog–digital process.

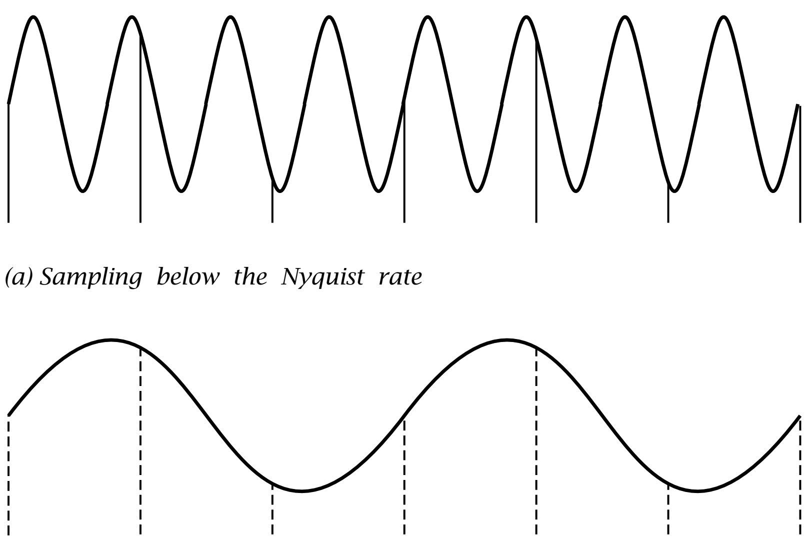

(a) Sampling below the Nyquist rate
(b) Resulting filtered signal

Fig. 14.4 Illustrating aliasing.

low-pass filtered at the input of an A/D converter. This process is known as anti-aliasing filtering.

In dealing with analog inputs many situations simply need to make a true:false decision on whether a voltage is above or below a reference value V_{ref}. For instance, the signal shown in Fig. 14.5 (see also Fig. 14.20) represents the current during the discharge of an EKG diphasic defibrillator, as generated using a Hall effect current to voltage sensor. When nothing is happening the baseline voltage is 2.6 V. When the defibrillator begins its discharge, this voltage rapidly rises to a peak of 3.6 V over a few tens of microseconds. If the MCU is to sample the voltage over the next several tens of milliseconds, say, to calculate the total shock energy, then to begin this process it needs to know when this voltage rises above a threshold. In the diagram this is shown as 3.4 V. It could of course rapidly sample the analog signal using its integral Analog-to-Digital module (if it has one), as described later on page 454, but this continuous sample-and-check routine would use most of the processing capability of the processor. It would be much more software efficient to be able to automatically generate an interrupt in hardware when the input voltage V_{defb} rises above this threshold. The resulting ISR could then begin sampling the signal and performing the real-time analysis.

In Fig. 14.5 the analog signal V_{defb} is used as the input to the non-inverting (+) terminal of an **analog comparator**. The inverting termi-

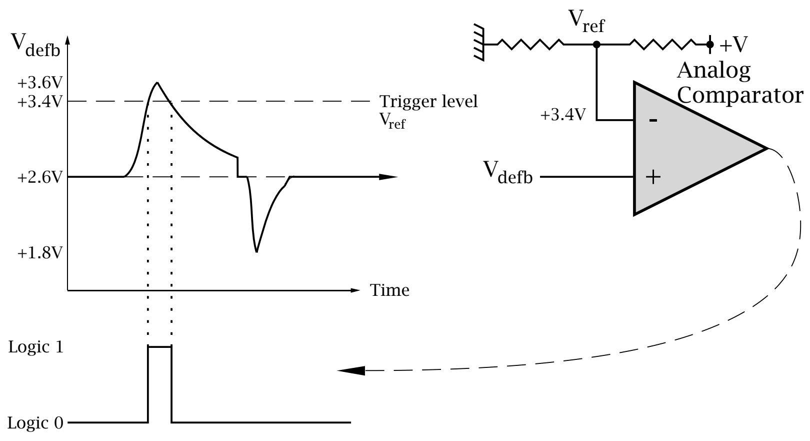


Fig. 14.5 Using an analog comparator to determine the start of the EKG defibrillator discharge.

nal (−) is connected to a fixed reference V_{ref} of 3.4 V. Whenever V_{defib} rises above V_{ref}, the comparator's output voltage changes from logic 0 to logic 1, and conversely when $V_{defb} < V_{ref}$ the output goes back to logic 0.

An analog comparator is basically a high-gain analog differential amplifier with no negative feedback. With a very large open-loop gain the amplifier will saturate at either near its positive or negative power supply if the difference between inputs is more than an exceedingly small value. An ordinary operational amplifier can act as an analog comparator, but circuits specifically designed for this purpose give standard logic levels at their output and have a snap action whenever slowly changing noisy inputs cross the differential threshold.

All three of our exemplar PIC microcontrollers feature a **Comparator module**. The PIC12F675 8-pin device has one analog Comparator. However, the dual configuration available in the PIC16F87XA group, shown in Fig. 14.6, is more typical for larger footprint devices.

The **COMparator CONtrol CMCON** register, usually at File h'9C', is used to set-up one of the eight configurations listed in the diagram, as commanded by the three **CM[2:0] Comparator Mode** bits in CMCON[2:0]. In the specific case of the PIC16F87XA devices these bits reset to mode b'111', which effectively completely removes the Comparator module from view. Most other devices reset to mode b'000' which, while also disabling the Comparators, rather subtly configures the associated pins as analog inputs.

It is a universal rule in PIC microcontrollers with analog modules, that all potential analog inputs pins (usually Port A, E or GP) *always come out of a Power-up reset as analog inputs.* This Power-on reset requirement is to prevent physical damage to the input digital buffers (see Fig. 11.7 on

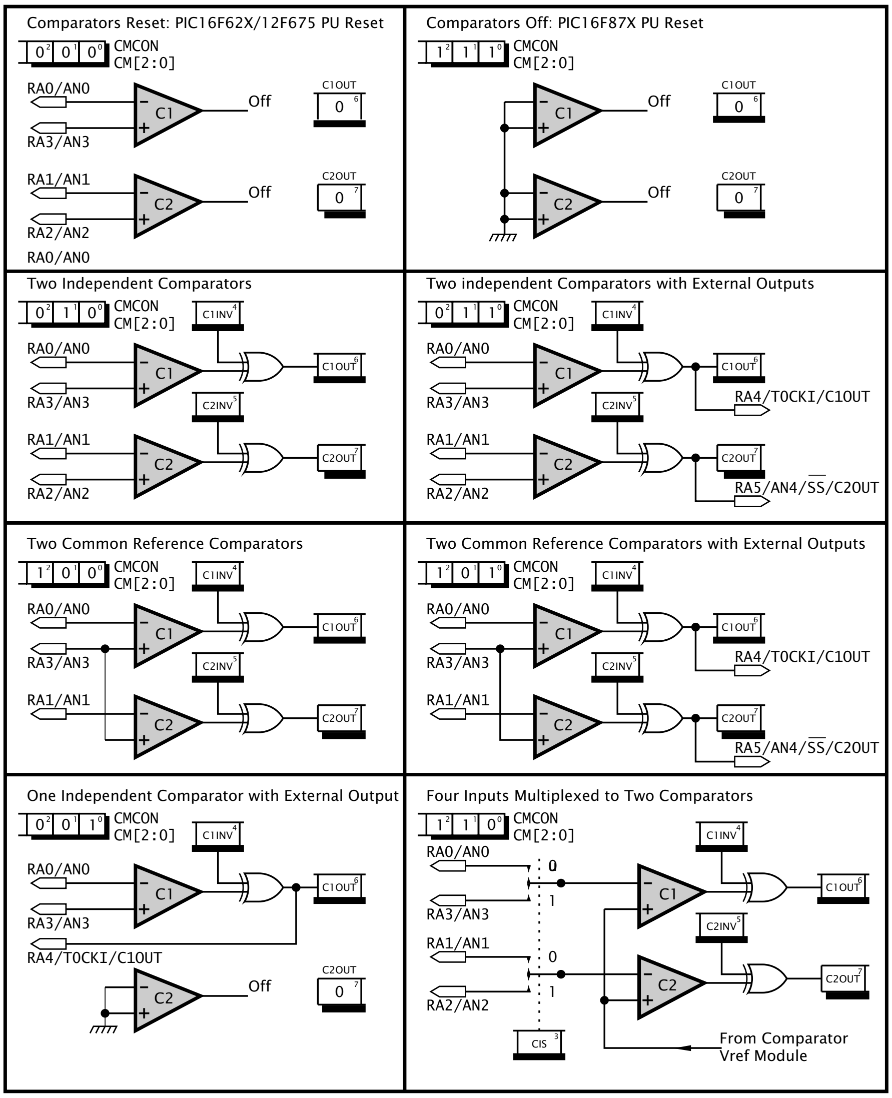

Comparators Reset: PIC16F62X/12F675 PU Reset
Comparators Off: PIC16F87X PU Reset
Two Independent Comparators
Two independent Comparators with External Outputs
Two Common Reference Comparators
Two Common Reference Comparators with External Outputs
One Independent Comparator with External Output
Four Inputs Multiplexed to Two Comparators
CMCON
CM[2:0]
RA0/AN0
RA1/AN1
RA2/AN2
RA3/AN3
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
C1
C2
Off
C1OUT
C2OUT
C1INV
C2INV
CIS
From Comparator Vref Module

Fig. 14.6 The Comparator module as implemented for the PIC16F87X device group.

page 306) if an analog input voltage, say 2.6 V, were present at a pin on powering up. If that pin was set to be a digital input, expecting a voltage around 0 V or V_{DD}, then an intermediate voltage could cause several transistors to conduct at the same time, possibly causing thermal damage. As analog voltages are not well defined, even where a pin is configured as analog, an external resistor is often used to limit current flow if the analog voltage exceeds V_{DD} or goes negative, as shown in Fig. 14.20.

In the specific case of the PIC16F87XA group, to preserve compatibility with the slightly older PIC16F87X which didn't have a Comparator module, the default is to completely remove this facility. However, all devices in this group have an Analog-to-Digital module, which on reset configures all associated pins as analog and so complies with this rule. Mode b'111' has the lowest power consumption, so it should be chosen if there are no analog input signals and the module is not to be used, especially when in the Sleep state.

The six active modes basically allow either completely independent operation for one or two Comparators, or both non-inverting inputs can be combined to be used as a common reference input. Outputs of any active Comparator may be read at any time from bit 6 **C1OUT** and bit 7 **C2OUT** of CMCON. Each output has an associated programmable invertor control in CMCON[4] and CMCON[5], respectively, labeled **C1INV** and **C2INV**. When $V_{in+} > V_{in-}$ and the associated INVersion bit is 0 then the Comparator output will read as 1, otherwise as 0.[2] As described on page 415, in some versions of Timer 1 the output of Comparator 2 may be used to gate the counting pulse train and thus measure the duration of time an analog signal is above a threshold voltage. In modes b'011' and b'101' the Comparator outputs can also be read externally via pins RA4/C1OUT and RA5/C2OUT—or equivalent shared pins in other devices. In this case pins RA4 and RA5 should be set-up as outputs using TRISA[5:4] in the usual way. Any parallel port pins to be used as analog inputs should similarly be set-up via the appropriate TRIS resister as inputs.

When there is a *change* in an active Comparator output, the **CoMparator Interrupt Flag CMIF** (in PIR2[6] for the PIC16F87XA) will be set and will generate an interrupt if the associated **CoMparator Interrupt Enable mask CMIE** (in PIE2[6] for the PIC16F87XA) and global mask bits GIE and PEIE have been set to 1. As each Comparator does not have its own interrupt flag, the software needs to maintain information regarding the status of the output bits C1OUT and C2OUT to determine which Comparator actually changed. This can be updated as part of the ISR. The act of reading CMCON will end the Change mismatch—in the same manner as the Port B Change interrupt described on page 313. Only then can the CMIF flag can be cleared, in the normal manner. If the Comparator mode is to be changed "on the fly" then interrupts should be disabled beforehand. After a delay of not less than 10 μs after the mode change, to allow voltage levels to stabilize, CMCON should be read again to clear

[2]There is a small uncertainty range in this difference signal of ±10 mV maximum (±5 mV typical) due to Comparator offset voltages.

any Change mismatch and CMIF cleared afterwards before re-enabling the interrupt system.

As the Comparator module does not depend on the system oscillator, an active Comparator can be used to waken a sleeping PIC MCU when an external voltage crosses a V_{ref} threshold and sets CMIF. After wakening, the PIC MCU should cancel the Change mismatch and clear CMIF following the sleep instruction or in the ISR if the Comparator interrupt is enabled.

It should be noted that an active Comparator uses considerably more current than the base Sleep value. For instance, the PIC12F629/75 group has a typical quiescent current at 5 V of 2.9 nA (995 nA maximum). The Comparator module uses a current of typically 11.5 μA (16 μA maximum). Thus if Comparators are not being used during the Sleep duration, they should then be disabled.

Mode b'110' allows one of two voltages to be sampled for each of the comparators individually, as switched via the **Comparator Input Switch CIS** bit in CMCON[3]—which is zeroed on a Power-on reset. In addition, both non-inverting inputs are commoned to an internal voltage reference source, generated from the **Comparator Voltage Reference** module.

All family members with a Comparator module have a separate but related **Comparator Voltage Reference CVR** module. As can be seen from Fig. 14.7, this is essentially a resistor chain with an analog multiplexer gating through one of 16 different voltages, as selected with the **Comparator Voltage Reference** bits **CVR[3:0]** in the **Comparator Voltage Reference CONtrol** register **CVRCON[3:0]**. The CVR module is enabled when the **Comparator Voltage ENable CVREN** in CVRCON[7] is set to 1. This also connects the chain of typically 2 kΩ resistors to the V_{DD} supply voltage.

Two voltage ranges are available, as set with the **Comparator Voltage Reference Range CVRR** bit in CVRCON[5], which switches in or out an extra 8R resistor at the bottom of the chain. Denoting the 4-bit value of CVR[3:0] as n, these are:

CVRR	Value	Minimum	in 16 steps of	Maximum
0 (reset)	$V_{DD} \times (0.25 + n/32)$	$0.25 \times V_{DD}$	$V_{DD}/32$	$0.71875 \times V_{DD}$
1	$V_{DD} \times n/24$	0 V	$V_{DD}/24$	$0.625 \times V_{DD}$

where n ranges from 0 through 15.

The accuracy of this module is given as $\frac{1}{2}$ of a single step, but in reality the absolute value is directly proportional to the supply voltage; a quantity that is not normally tightly regulated. In addition, V_{DD} can change as the power supply or battery drifts with temperature or load current. Any noise on this line will also be coupled into this reference voltage, although this can be reduced somewhat by capacitive decoupling at the V_{DD} pin and judicious routing of the power supply lines. With these points in mind, where an accurate voltage level is required, an external precision voltage

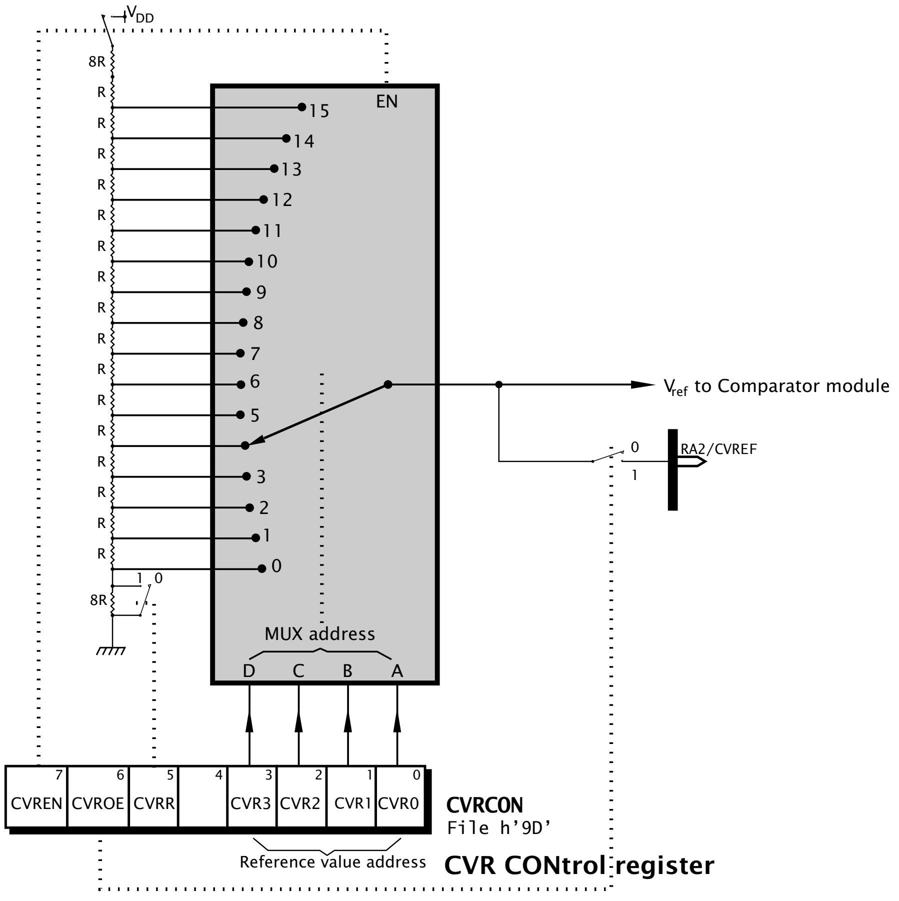


Fig. 14.7 The Comparator Voltage Reference module.

reference device is often used; for instance, with Comparator module mode b'100' connected to pin RA3; see Fig. 14.20.

As our example, assume that V_{DD} is 5 V and we are going to generate our threshold voltage of 3.4 V for Fig. 14.5. We will need to use the high range; that is, CVRR = 0, and calculate a value for CVR[3:0]:

$$\begin{aligned} 5 \times (0.25 + n/32) &= 3.4 \\ 0.25 + n/32 &= 3.4/5 \\ n &= (3.4/5 - 0.25) \times 32 = 13.76 \end{aligned}$$

giving $n = 14$ as our closest approximation. Making CVR[3:0] = b'1110' gives an actual V_{ref} of 3.4375 V.

Some family members have an additional control bit to switch the reference voltage to a port pin, so that it can be accessed externally. When the **CVR Output Enable bit CVROE** in CVRCON[6] is set to 1, the analog voltage V_{ref} is gated through to the appropriate pin. Due to the relatively high value of resistance, which also depends on the selected tap, Microchip recommend that this external reference voltage be buffered—typically with an operational amplifier. If necessary, the amplifier gain can be altered to give finer control over the V_{ref} value and filtering can be added to reduce high-frequency noise. Used in this way, the Voltage Reference module can be used as a simple 4-bit digital-to-analog converter.

The code to set-up the Comparator and CVREF modules for our defibrillator example using Comparator 1 with RA3 as the analog input is then:

```
include  "p16f877a.inc"
bsf    STATUS,RP0  ; Change to Bank 1
movlw  b'00001110' ; Comparator mode 110
movwf  CMCON       ; Switch to RA3 (CIS = 1)

movlw  b'10001110' ; CVREF module on (1), not external (0)
movwf  CVRCON      ; Hi range (0), CVR[3:0] = 1110

bsf    PIE2,CMIE   ; Enable Comparator interrupts

call   DELAY_10US  ; Allow 10us for voltages to settle
movf   CMCON,f     ; Read CMCON to clear any Change state

bcf    STATUS,RP0  ; Back to Bank 0
bcf    PIR2,CMIF   ; Zero the Comparator interrupt flag
bsf    INTCON,PEIE ; Enable Peripheral interrupt group
bsf    INTCON,GIE  ; & Globally enable interrupt system
```

Notice especially that before enabling the interrupt system a delay of 10 μs is executed to allow internal analog voltages to attain equilibrium. Reading the CMCON register then clears any Change situation, after which the Comparator Interrupt Flag CMIF is cleared. The general interrupt system can then be enabled by setting the PEIE and GIE mask bits in the usual way in the INTCON register.

Some device data sheets, for instance, the PIC12F675, label this module the **Voltage Reference** module. Here the associated Control register is labeled **VRCON** and the various Control bits similarly have their C prefix removed; e.g., **VREN** instead of CVREN.

In many situations more information on the analog signal is needed than a bang-bang comparison with a reference voltage. For instance, in the waveform shown in Fig. 14.5 the deviation of the voltage squared from the baseline, integrated with time, would be required to measure power. In such a situation the incoming signal would have to be sampled and converted from an analog amplitude to a digitized equivalent.

The mapping function from an analog input quantity to its digital equivalent can be expressed as:

$$V_{in} \mapsto V_{ref} \sum_{i=1}^{n} k_i \times 2^{-i}$$

where k_i is the ith binary coefficient having a Boolean value of 0 or 1 and $V_{in} \leq V_{ref}$, where V_{ref} is a fixed analog reference voltage. Thus V_{in} is expressed as a binary fraction of V_{ref} and the Boolean coefficients k_{-i} are the required binary digits.

To see how we might implement this in practice, consider the following mechanical successive approximation analogy. Suppose we have an unknown weight W (analogous to V_{in}), a balance scale (equivalent to an analog comparator) and a set of precision known weights 1, 2, 4, and 8 gm (analogous to a V_{ref} of 15 gm). A systematic technique based on the task list might be:

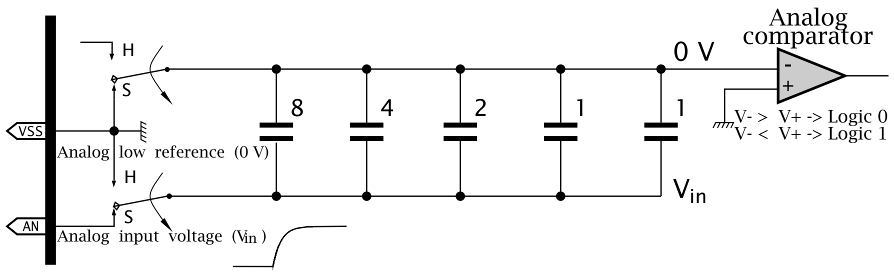
1. Place the 8 g weight on the pan. IF too heavy THEN remove ($k_1 = 0$) ELSE leave ($k_1 = 1$).
2. Place the 4 g weight on the pan. IF too heavy THEN remove ($k_2 = 0$) ELSE leave ($k_2 = 1$).
3. Place the 2 g weight on the pan. IF too heavy THEN remove ($k_3 = 0$) ELSE leave ($k_3 = 1$).
4. Place the 1 g weight on the pan. IF too heavy THEN remove ($k_4 = 0$) ELSE leave ($k_4 = 1$).

This technique will yield the nearest lower value as the sum of the weights left on the pan. For instance, if W were 6.2 g then we would have a weight assemblage of 4 + 2 g or b'0110' for a 4-bit system.

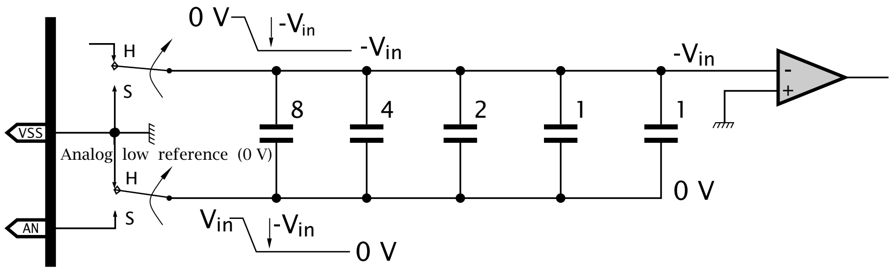
The electronic equivalent to this **successive approximation** technique uses a network of precision resistors or capacitors configured to allow consecutive halving of a fixed voltage V_{ref} to be switched in to an analog comparator, which acts as the balance scale.

Most MCUs use a network of capacitors valued in powers of two to subdivide the analog reference voltage, such as shown in Fig. 14.8. Small capacitance values are easily fabricated on a silicon integrated circuit and although the exact value will vary somewhat between different batches of ICs, within the one device all capacitor values will closely match and track with changes in temperature and supply voltage. Multiples of this base value can be fabricated by paralleling unit devices—typically FET gate-source capacitance.

Before the conversion process gets underway, the network has to be primed with the unknown analog input voltage V_{in}, as shown in Fig. 14.8(a). The dynamics of this **sampling** acquisition process involves charging up this capacitance network through both internal and external resistance allowing for the settling time of the internal analog switches. If we take the 10-bit ADC module of Fig. 14.11 as an example, then the parallel capacitor network with nominal unit value of 0.12 pF appears at the AN pin



(a) The sample process



(b) The hold process

Fig. 14.8 Initializing the 8-4-2-1 capacitor network for a 4-bit convertor.

as a 120 pF (120×2^{-12}F) capacitor. Internal resistance is of the order of 7.5 kΩ, but is rather temperature and supply voltage dependent. Externally the maximum recommended value is 2.5 kΩ in order to keep any ohmic voltage offset due to the pin leakage current of $\pm\frac{1}{2}\,\mu$A less than one quantum level (least-significant bit).

The time constant τ (CR) with the values given here is $120 \times 10^{-12} \times$ $\times 10^4 = 1.2\,\tau$s for a total resistance of 7.5 + 2.5 = 10 kΩ. In order to get within 0.05% of the final voltage; that is, $\frac{1}{2}$ of a 10-bit quantum level, takes approximately $8 \times \tau \approx 10\,\mu s$. The data sheet gives the maximum switch settling time of 10 μs, although examples use a value of 2 μs. Taking a worse-case scenario, a sampling time of 20 μs should ensure stability before the conversion.

Our analysis assumed that the input voltage on the capacitor network needed to be changed by the full range since the last sample. This is likely to be the case if sampling one of several different analog channels or a long time has elapsed since the last sample, allowing charge to leak away. A smaller external source resistance will reduce the time constant. Of course, to evaluate the maximum rate of samples that can be taken, the actual conversion time must be added to this acquisition time.

During the sample (S) period the top capacitor electrodes are held to 0 V and bottom electrodes are charged to V_{in}. The change-over to the hold (H) position, shown in Fig. 14.8(b), grounds the bottom electrodes and allows the top electrodes to float. The voltage across a capacitor can only change if charge is transferred across electrodes, $\Delta Q = C\Delta V$. Thus the change in voltage $\Delta V = -V_{in}$ at the bottom electrodes is matched at the top floating electrodes, which now become $0 - V_{in}$, as charge cannot flow in or out of the floating top electrodes. Thus at the start of the conversion process the inverting input of the analog comparator is $-V_{in}$.

A 4-bit version of the successive approximation network at the heart of the ADC module is shown in a simplified form in Fig. 14.9. The step-by-step process is sequenced by a shift register (SRG, see Fig. 2.22 on page 37) when the programmer sets the GO/$\overline{\text{DONE}}$ bit in the ADC Control register. As the Control shift register is clocked, a single 1 moves down to activate each step in the sequence:

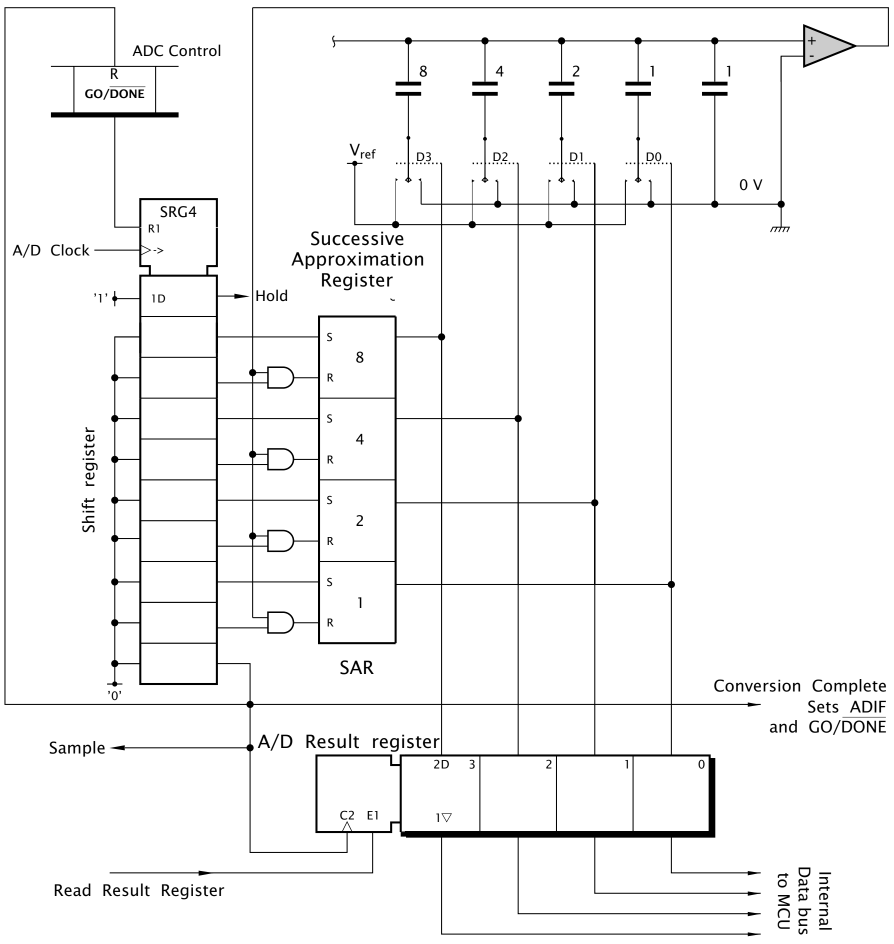


Fig. 14.9 Simplified view of a 4-bit successive approximation A/D converter.

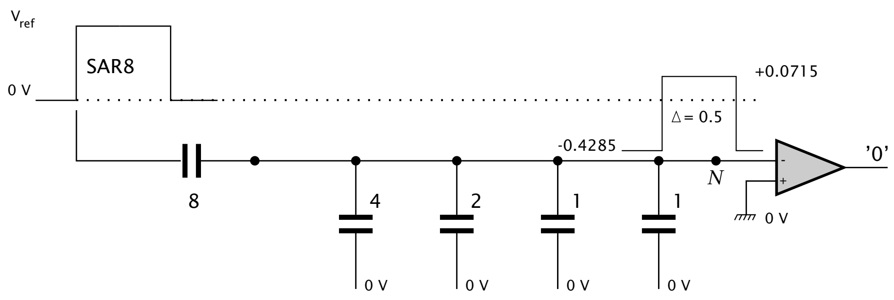
Hold	bit 3	bit 2	bit 1	bit 0	Complete/Sample

The capacitor network is switched to Hold and each capacitor, beginning with the largest value, is switched to V_{ref} in turn. The outcome of the comparator then determines the state of the corresponding bit in the Successive Approximation Register (SAR). The process is detailed in Fig. 14.10. After four set–try–reset actions, the outcome in the SAR is transferred to the Analog-to-Digital RESult register. The $GO/\overline{DONE}$ flag is now cleared to indicate the End Of Conversion and the Analog/Digital Interrupt Flag **ADIF** set. Finally, the analog input is again switched back into the capacitor network (Sample) which then charges up ready for the next conversion after a suitable period.

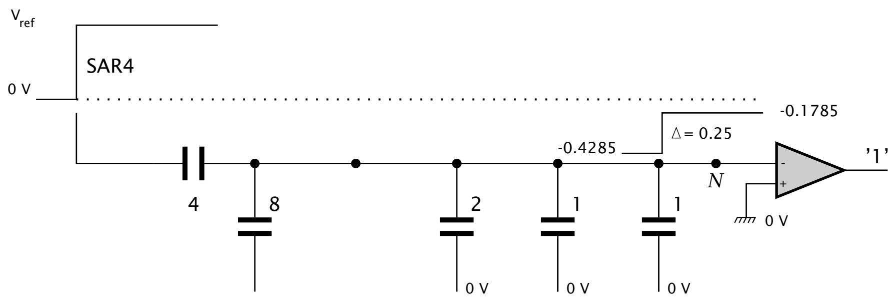
The total conversion time is approximately six times the clock period t_{AD} of the sequencer shift register—one period for each bit plus one each for the Hold and Ready/sample slots. In the case of a 10-bit module, this will be approximately 12 times the clock period. For the PIC MCU modules, the minimum clocking period is $1.6\,\mu$s ($\approx$ 600 kHz) for all but the older $2\,\mu$s PIC16C71/711 devices. There is no specified lower clocking frequency, but as charge slowly leaks away from the network capacitors, a t_{AD} of more than nominally $20\,\mu$s (50 kHz) should be avoided. From Fig. 14.11 we see that the ADC clock can be derived from one of four sources. The first three of these are fractions of the system clock rate and the fourth is a stand-alone CR oscillator with a nominal t_{AD} of $4\,\mu$s.

The conversion process, where each successive half-fraction of V_{ref} is added to and conditionally taken away from the initial value is illustrated in Fig. 14.10. As we have seen in Fig. 14.8, at the end of the acquisition period the top plates of the capacitor array are at $-V_{in}$. As an example, let us assume that V_{in} is $0.4285V_{ref}$.

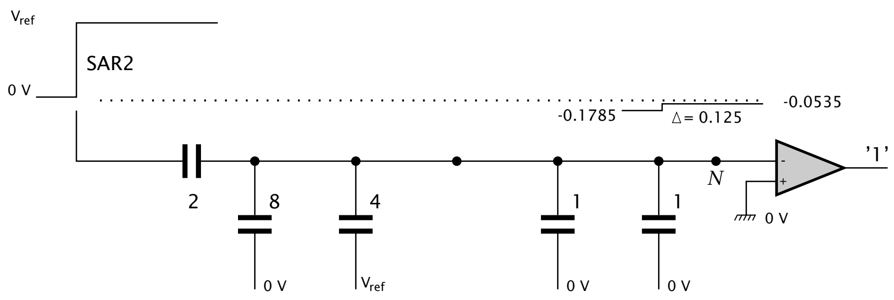
1. The process begins by switching in V_{ref} into the lower plate of the largest capacitor, as controlled by the SAR_8 latch in Fig. 14.9. This causes an injection of charge $\Delta Q = C_{total}V_{ref}$, which is identical across both the 8-unit capacitor C_1 and the rest of the capacitors which also have a parallel value of 8 units in Fig. 14.10. Thus the voltage at node N rises by $V_{ref}/2$ to $-0.485 + 0.5 = +0.07125V_{ref}$. In general $\Delta V_N = V_{ref}C_k/C_{total}$. The comparator output is now logic 0 and thus the SAQ_8 latch is consequently cleared, reversing the $V_{ref}/2$ step.
2. SAQ_4 switches V_{ref} into the next highest capacitor giving a $V_{ref}/4$ step at N ($\frac{4}{16}$). The resulting voltage of $-0.485 + 0.25 = -0.178V_{ref}$ giving a comparator output of logic 1 and SAR_4 remains set with the node voltage staying at $-0.1785V_{ref}$.
3. SAQ_2 switches V_{ref} into the second lowest capacitor giving a $V_{ref}/8$ step at N ($\frac{2}{16}$). The resulting voltage of $-0.1785+0.125 = -0.0535V_{ref}$ giving a comparator output of logic 1 and SAR_2 remains set with the node voltage staying at $-0.0535V_{ref}$.



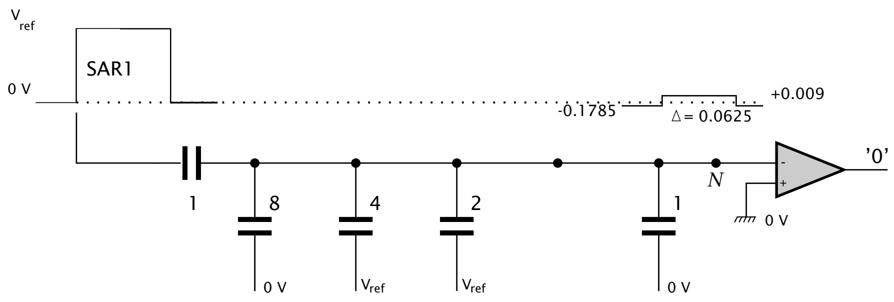
(a) The most significant bit



(b) The second most significant bit



(c) The third most significant bit



(d) The least significant bit

Fig. 14.10 The successive approximation process.

4. $\mathsf{SAQ_1}$ switches V_{ref} into the lowest capacitor giving a $V_{ref}/16$ step at N ($\frac{1}{16}$). The resulting voltage of $-0.0535 + 0.0625 = +0.009V_{ref}$ giving a comparator output of logic 0 and $\mathsf{SAR_1}$ is cleared reversing the $V_{ref}/16$ step.

The state of the SAR of b'0110' or $0.375V_{ref}$, represents the best 4-bit fit to $V_{in} = 0.4285V_{ref}$. The residue $0.0535V_{ref}$ is the quantizing error.

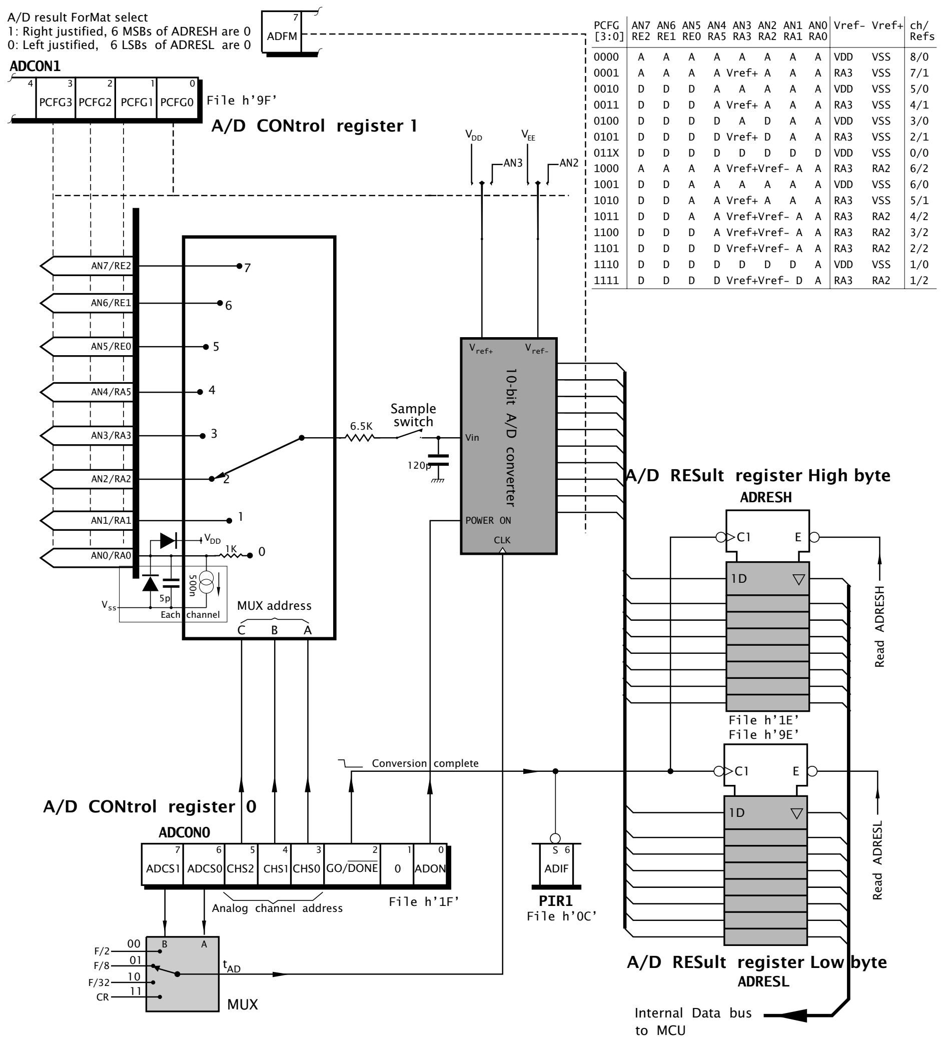
Most MCUs use an 8- or 10-bit capacitor array. In principle the technique can readily be extended to higher resolutions, but in practice the difficulty in matching ever greater capacitors and internal logic noise means the majority of processors are limited to 12-bit resolution. External high-speed successive-approximation devices with 12+ bit resolution, usually using a resistor ladder network, are readily available, but are relatively expensive.

Matching of the array capacitors, offsets, and resistance of internal switches, leakage currents, and analog comparator non-linearities all contribute to errors in the conversion process. It is beyond the scope of this text to analyze the various measures of error but the device data sheet lists sources and values of these component errors in terms of the least significant bit. For instance, in the PIC12F675 data sheet the 10-bit ADC module is listed as having a total absolute error of ±1 LSB. This guarantees that the transfer is monotonic; that is, the binary code will never move in the reverse direction for any change ΔV_{in} of input voltage. This error figure is for $V_{ref} = V_{DD}$; if V_{ref} is lower than V_{DD} then accuracy deteriorates, although values down to 2 V will give acceptable results in many cases.

Both the PIC12F675 and the PIC16F87X exemplar group have an integral 10-bit ADC module. Earlier devices, such as the PIC16F73, use an 8-bit version of this module, which is very similar in architecture and operating process to the 10-bit module shown in Fig. 14.11 and so will not be discussed here. All PIC MCU ADC modules use a capacitor network with characteristics previously described. From the user's perspective the details of the conversion process are less important than the system aspects in integrating this module into software.

In all relevant family members the single analog-to-digital converter is fronted with an analog multiplexer. This allows the software to select up to eight separate analog voltages *one at a time.* Two Control registers allow the program to select any one channel for sampling and to determine the source of the sequence clock. In addition the appropriate pins may be set-up as either analog (by default on a Power-on reset) or digital, with some control over the source of reference voltage. The conversion is initiated via the $\mathsf{GO}/\overline{\mathsf{DONE}}$ bit, which also indicates when the process is complete and the 10-bit outcome can then be read from the two Result registers.[3]

[3] 8-bit modules only require one AD Result register.



PCFG [3:0]	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	Vref-	Vref+	ch/ Refs
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	Vref+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	Vref+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	Vref+	D	A	A	RA3	VSS	2/1
011X	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	Vref+	Vref-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	Vref+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	Vref+	Vref-	A	A	RA3	RA2	4/2
1100	D	D	D	A	Vref+	Vref-	A	A	RA3	RA2	3/2
1101	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	Vref+	Vref-	D	A	RA3	RA2	1/2

Fig. 14.11 The PIC16F87X 10-bit 8-channel analog-to-digital conversion module.

Our description of the ADC module can be split into the initial set-up and the conversion process.

Initialization

In setting up your module you need to consider the following points:

1. How do I enable the module?
2. How am I going to clock the module?
3. Which channels am I going to use?
4. Do I only need an 8-bit outcome?

All these options are set up using **AD Control register 0**, **ADCON0** and **AD CONtrol register 1**, **ADCON1**.[4]

ADON
On a Power-on reset the ADC module is disabled. Setting ADCON[0] to 1 turns the module on. An enabled module typically uses 220 μA (PIC16F87X) even when idling, so it should be disabled when not in use where power consumption is a consideration. The GO/$\overline{\text{DONE}}$ switch bit should not be set to 1 in the same instruction as the module is enabled to avoid starting a conversion at the same time as the module is being started up.

ADCS[1:0] A/D Clock Select
The module needs a clock signal in order to time the set and test sequence of Fig. 14.10. If the clock rate is too fast, changes in switching voltages will not have time to settle. The data sheet specifies the upper frequency in terms of the A/D clock period t_{AD}, as 1.6 μs (3 μs for low-voltage situations) or approximately 600 kHz. For instance, a 5 MHz crystal with ADCS[1:0] = 01 gives a t_{AD} of 1.6 μs ($\frac{5}{8}$ MHz) using a $\div$8 ratio. Table 14.2 shows suggested settings for five typical crystal values.

Table 14.2: ADC clocking frequency versus device crystal frequency.

ADC clock source t_{AD}		PIC MCU crystal frequency				
	ADSC1:0	20 MHz	8 MHz	4 MHz	1 MHz	333 kHz
$f_{OSC}/2$	00	—	—	—	2 μs	6 μs
$f_{OSC}/8$	01	—	—	2 μs	8 μs	—
$f_{OSC}/32$	10	1.6 μs	4 μs	8 μs	—	—
CR[1]	11	2-6 μs	2-6 μs	2-6 μs	2-6 μs	2-6 μs
CR[2]	11	3-9 μs	3-9 μs	3-9 μs	3-9 μs	3-9 μs

Note 1: Standard devices; average 4 μs.
Note 2: Extended-range and low-voltage devices; average 6 μs.

To allow operation in a low-speed system clock environment; for instance, when a 32.768 kHz watch crystal is used, a separate internal Capacitor-Resistor (CR) oscillator is provided. As this stand-alone oscillator is separate from the system clock, a conversion can be completed while the PIC MCU is in its Sleep state. In this situation, the End Of Conversion interrupt can be used to waken the processor. Doing a conversion with the system clock turned off makes sense,

[4]The PIC12F675 has a slightly different arrangement of Control registers with an ANSEL register replacing ADCON1 and a different distribution of Control bits, but the principles are the same.

as this gives a quiet environment with little digital noise. If the separate CR clock is used with a system clock of greater than 1 MHz, then Microchip recommends using a Sleep conversion, as the lack of synchronization between the two clock rates increases noise induced into the analog circuitry.

Unusually, the PIC12F675 has three AD Clock Select bits, giving a frequency division option of ÷64. This is useful to cope with a 20 MHz system clock to give the 3 μs minimum specified for the larger voltage range for this device.

CHS[2:0] CHannel Select

Devices with ADC modules can select to digitize the voltage in one of several possible analog input pins. This ranges from four channels shared with Port GP for the diminutive 8-pin PIC12F675 through to eight shared with Port A and E for the 40-pin PIC16F874/7.

On a Power-on reset, all such shared port pins default to analog inputs; see page 442. As can be seen from Fig. 14.12, an I/O pin configured as an analog input simply disables the digital input buffer—compare with Fig. 11.3 on page 300. No other circuitry is affected. From this we can make the following deductions.

- A port pin configured as analog will read as logic 0 due to the disabled digital input buffer.

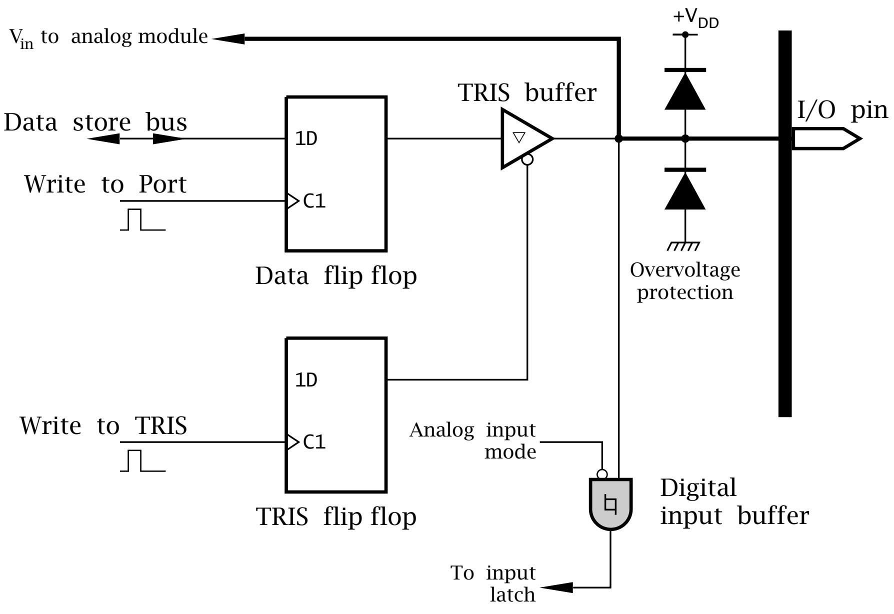


Fig. 14.12 Configuring the analog inputs for Port A and Port E.

- The TRIS buffer is not affected and thus the appropriate TRIS bits should be 1; that is, the direction of the port pins configured as analog should be set to input to prevent contention between the analog V_{in} and the digital state of the Data flip flop.
- The ADC can read an analog voltage at the pin even if that pin has not been configured as analog. However, the still active digital input buffer may consume an excessive current outside of the device's specification.

PCFG[3:0] Port ConFiGuration

If the number of analog channels required for a particular application is less than the maximum available, some unused channels can be reclaimed for use by the associated parallel port. This is accomplished using the appropriate bit pattern in ADCON1[3:0]. The actual choices, number, and location of these bits are device specific, but those patterns applicable to the PIC16F87X group are shown in Fig. 14.11. For instance, if you only require a single analog channel for your project, the pattern b'1110' will leave pin RA0/AN0 as analog and pins RA5, RA[3:1] and RE[2:0] are available for other purposes.

In the situation where no analog activity is required, the ADCON1 register still needs to be set-up; this time to pattern b'0110' or b'0111' to configure all pins as digital.[5] Failure to do this is one of the more common errors, as most newer devices have analog module(s) and, as described on page 442, all relevant pins always default to analog on a Power-on reset. Port pins set-up as analog always read through a parallel port as 0. As observed, any pins used for analog purposes should have their associated TRIS bits set to 1; that is, set-up as inputs.

As we have seen in Fig. 14.10, the successive approximation process essentially comprises a series of tries of ever-halving fractions of a fixed reference voltage. The precision of this process depends on the quality of this reference. One measure of this is the worth of the least significant bit (LSB); that is, the quatization increment. In the case of a 10-bit module this value is $V_{ref}/1024$, or better than 0.1% of this reference.

This reference voltage can be set-up to be internal, using the power supply voltage, say 5 V. For instance, the pattern b'1110' sets pin RA0 to be an analog input and V_{DD} will be the reference used. In this situation the digitization will essentially give the fraction of the supply this analog voltage is.

Using the supply voltage is rather noise prone and its value can vary somewhat. Where more precision or different voltages are required, analog pins can be used as the source of external reference

[5]In some devices (not the PIC16F87XA) the Analog Comparator module's Control register also needs to be changed from its default value.

voltages. All ADC modules will allow for at least one external voltage. In the case of the PIC16F87X, one or two external references may be used. For instance, pattern b'0101' would set-up pins RA[1:0] as two analog channels and use RA3 for an external precision voltage reference V_{ref+}; see Fig. 14.20.[6] V_{ref+} can be anywhere between $V_{DD} - -2.5$ V and $V_{DD} + 0.3$ V, with a 2 V minimum.

In some cases it can be an advantage to use a different lower bound than V_{SS} (0 V or ground). Some ADC modules, e.g., the PIC16F87X, allow for a separate lower reference voltage V_{ref-}. For our example, pattern b'1101' also specifies two analog channels with V_{ref+} on pin RA3. However, this time pin RA2 is used for V_{ref-}. This should not be higher than 2 V nor below −0.3 V. Overall, the full range $V_{ref+} - V_{ref-}$ should never be less than 2 V.

ADFM A/D ForMat
Our example ADC module needs two Files to hold the 10-bit outcome. As the total capacity of ADRESH:ADRESL is 16 bits, there are two ways of aligning these ten bits.

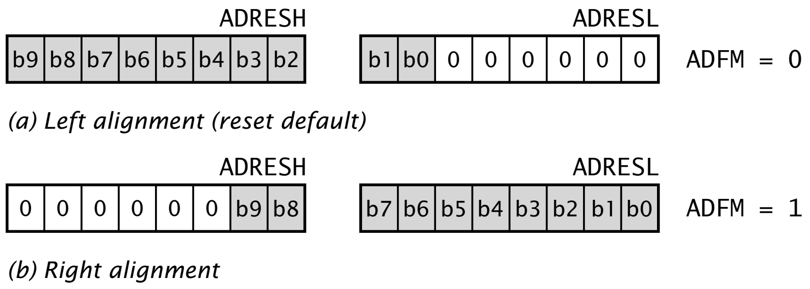


Fig. 14.13 Aligning the 10-bit digital outcome in a 16-bit field.

Many applications only require 8-bit resolution and processing. Where this is the case, the bottom two bits of the outcome word can be thrown away. From Fig. 14.13(a) we see that this is facilitated by left alignment. The content of ADRESL is simply ignored.

Where a full 10-bit word is necessary, setting ADCON1[7] to 1 will right align the datum. As can be seen from Fig. 14.13(b) the outcome is a 10-bit datum extended to a 16-bit format by padding with leading zeros. Normal 16-bit arithmetic and other processing algorithms can then be used.

[6]This could be derived from the Voltage Reference module if implemented (see Fig. 14.7) although this has many of the disadvantages of using the raw supply voltage.

Conversion Process

After the module has been configured, from the user's perspective digitizing a selected analog channel is relatively straightforward. Assuming first that interrupts are not being used, the following steps can be identified (including, for completeness, the initialization process) and is visualized with the timeline in Fig. 14.14.

1. Configure ADC module.
   - Set up port pins as analog/voltage reference (ADCON1).
   - Select ADC conversion clock source (ADCON0).
   - Select ADC input channel (ADCON0).
   - Turn on ADC module (ADCON0).
2. Wait for the required acquisition time, typically 20 μs.
3. Start conversion by setting the GO/$\overline{\text{DONE}}$ bit.
4. Wait for ADC conversion to complete by polling the GO/$\overline{\text{DONE}}$ bit for logic 0.
5. Read the ADRES registers.
6. For next conversion go to step 1 or step 2 as required.

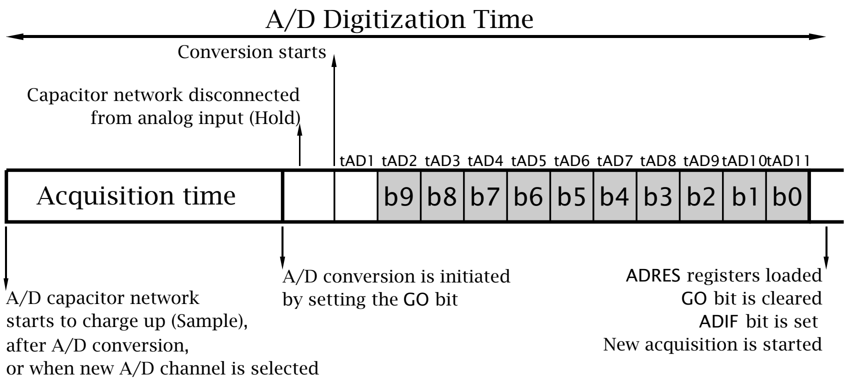

A/D Digitization Time
Conversion starts
Capacitor network disconnected from analog input (Hold)
tAD1 tAD2 tAD3 tAD4 tAD5 tAD6 tAD7 tAD8 tAD9 tAD10 tAD11
Acquisition time
b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
A/D capacitor network starts to charge up (Sample), after A/D conversion, or when new A/D channel is selected
A/D conversion is initiated by setting the GO bit
ADRES registers loaded
GO bit is cleared
ADIF bit is set
New acquisition is started

Fig. 14.14 Timeline for the conversion process.

As an example, consider that we wish to continually read each of the eight analog channels of a PIC16F874/7 in turn while outputting the most significant eight digitized bits to Port B and the channel number to the lower three bits of Port D. The main crystal is 20 MHz and the power supply is to be used as the reference voltage.

The listing of Program 14.1 assumes that the ADC module has been initialized at reset with start-up code of the form:

```
include "p16f877.inc"

bsf     STATUS,RP0 ; Change over to Bank 1

clrf    ADCON1     ; Shared port inputs are all analog
                   ; left aligned for 8-bit resolution
clrf    TRISB      ; Port B is all output
movlw   b'11111000'; Low 3 bits of Port D are Output
movwf   TRISD

bcf     STATUS,RP0 ; Back to Bank 0

movlw   b'10000001'; AD clock/32 (10), Ch0 (000)
movwf   ADCON0     ; No conversion (0), ADC turned on (1)
```

which configures the module to enable all eight analog channels with internal reference voltages and for a left-aligned outcome. ADCON1 is initialized to $\boxed{10}\boxed{000}\boxed{0}\boxed{0}\boxed{1}$ to select the ADC module clock source as $f_{OSC}/32$ (b'10'); i.e., $\frac{20}{32} = 625$ kHz or 1.6 μs, channel zero (the initial value is irrelevant) and with the module turned on. With an initial zero value of GO/$\overline{\text{DONE}}$ no conversion is actioned.

With the module initialized, the main software of Program 14.1 spends all its time in a loop reading the digitized equivalent of each channel in turn from ADRESH and copying it in turn to Port B. Before the digitization, the Channel counter is sent to Port D as a modulo-3 number.

The acquisition itself is implemented using the GET_ANALOG subroutine, to which is passed the desired channel number in the rightmost three bits of the Working register. This is copied into a temporary location TEMP, where it is logic shifted three places to the left to align the channel number with the CHSn bits in ADCON0[5:3]. After clearing the CHS[2:0] bits, the shifted Channel number can then be added into ADCON0 to set CHS[2:0] to the appropriate channel.

After the channel number has been set up, a delay subroutine is called to allow for switch delay and stabilization. As we require 8-bit resolution, only a $6\tau \approx 7\,\mu s$ delay is needed to charge up to within 0.25% of the final value, on top of the worst-case switch time of 10 μs; see page 449. Then the GO/$\overline{\text{DONE}}$ bit in ADCON0 is set to initiate a conversion.[7] The completion of the process can then be monitored by polling GO/$\overline{\text{DONE}}$ until this goes to 0. At this point the contents of ADRESH are the 8-bit outcome of the conversion.

Each actual conversion takes around $13 \times 1.6 \approx 21\,\mu$s, giving a total channel time of $17 + 21 = 38\,\mu$s. Thus a 8-channel scan takes around $38 \times 8 \approx 300\,\mu s$ to complete. That is, around 3300 scans per second.

Rather than polling for completion, the end of conversion can be used to generate an interrupt. In particular if a conversion is to be done in the

[7]A conversion may be aborted at any time by clearing GO/$\overline{\text{DONE}}$.

Program 14.1 Scanning an 8-channel data acquisition system.

```
MAIN  clrf   CHANNEL      ; Use a GPF to hold the Channel count
MAIN_LOOP
      movf   CHANNEL,w    ; Get the Channel number
      andlw  b'00000111'  ; Zero the top five bits
      movwf  PORTD        ; Copy to Port D

      call   GET_ANALOG   ; Digitize it; returned in W
      movwf  PORTB        ; and copy to Port B

      incf   CHANNEL,f    ; Advance to next channel
      goto   MAIN_LOOP    ; and DO forever

; ***************************************************************
; * FUNCTION: Analog/digital conversion at channel n           *
; * RESOURCE: Subroutine DELAY_17US, byte TEMP                 *
; * ENTRY   : Channel number in W                              *
; * EXIT    : Digitized analog value in W                      *
; ***************************************************************
GET_ANALOG
      movwf  TEMP         ; Copy of Channel number in TEMP
      bcf    STATUS,C     ; Shift channel number left 3 places
      rlf    TEMP,f       ; to align with ADCON0[5:3]
      rlf    TEMP,f       ; that is, the CHS[2:0] bits
      rlf    TEMP,w       ; and copy into W
      bcf    ADCON0,CHS0  ; Zero channel bits
      bcf    ADCON0,CHS1
      bcf    ADCON0,CHS2
      addwf  ADCON0,f     ; Moves Channel number to ADCON0[5:3]
      call   DELAY_17US   ; Wait 17us to stabilize
      bsf    ADCON0,GO    ; Start conversion
GET_ANALOG_LOOP           ; Takes around 20us to finish
      btfsc  ADCON0,GO    ; Check for End Of Conversion
       goto  GET_ANALOG_LOOP

      movf   ADRESH,w     ; Fetch byte when GO/NOT_DONE zero
      return

; *************************************************************
; * FUNCTION: Delays 17us at 20MHz (85 cycles)               *
; * ENTRY   : None                                           *
; * RESOURCE: None                                           *
; * EXIT    : W is zero                                      *
; *************************************************************
DELAY_17US
      movlw  d'20'         ; Delay constant
DELAY_17US_LOOP
      addlw  -1            ; Decrement
      btfss  STATUS,Z      ; Until zero
       goto  DELAY_17US_LOOP
      return
```

Sleep mode then this interrupt can be used to waken the device. The ADC module can operate when the PIC MCU is in its Sleep state as it has the option of its own private oscillator to sequence the conversion even if the system oscillator is disabled. The main advantage of a conversion while asleep is the electrically quiet environment when the system oscillator is off. Against this is the considerably longer conversion time, as when the PIC MCU is wakened, there will be the normal 1024-cycle delay to restart the system oscillator; see page 277.

This personal oscillator may be used even where the PIC MCU is not put to sleep. However, as there is no synchronization between the system and local oscillators, clock feedthrough noise becomes a problem, especially with system clock rates above 1 MHz.

The following task list outlines the Sleep state conversion process.

- The ADC clock source must be set to CR, ADCS1:0 = 11.
- The ADIF flag must be cleared to prevent an immediate interrupt.
- The ADIE and PEIE mask bits must be set to enable the ADC interrupt to awaken the processor.
- The GIE mask bit must be 0 unless the programmer wishes the processor to jump to an ISR when it awakens.
- The GO/$\overline{\text{DONE}}$ bit in the ADCON0 register must be cleared to initialize the conversion, followed immediately by the `sleep` instruction.
- On wakening, the ADRESH:L registers hold the digitized value.

For instance, consider a Sleep state version of the GET_ANALOG subroutine of Program 14.1. This time the initialization code must set up the interrupt system as specified in the task list, to ensure that when the AD Interrupt Flag ADIF is set at the end of the conversion (at the same time as the GO/$\overline{\text{DONE}}$ flag goes to 0) the PIC MCU is woken up.

```
include "p16f877.inc"

bsf     STATUS,RP0 ; Change over to Bank 1
clrf    ADCON1     ; Shared port inputs are analog, 8-bit res

clrf    TRISB      ; Port B is all output
movlw   b'11111000'; Low 3 bits of Port D are Output
movwf   TRISD

bsf     PIE1,ADIE  ; Enable AD interrupts
bcf     STATUS,RP0 ; Back to Bank 0

movlw   b'11000001'; CR AD clock (11), Ch0 (000)
movwf   ADCON0     ; No conversion (0), ADC turned on (1)

bcf     PIR1,ADIF  ; Zero the AD interrupt flag
bsf     INTCON,PEIE; & enable the Peripheral interrupt group
```

Apart from the initialization of the interrupt system, the only change is to the setting of ADCON0[7:6], which is made b'11' to select the internal ADC oscillator as the clock.

Program 14.2 Scanning an 8-channel data acquisition system.

```
; ***************************************************************
; * FUNCTION: A/D conversion at channel n while asleep          *
; * RESOURCE: Subroutine DELAY_17US, byte TEMP                  *
; * ENTRY   : Channel number in W                               *
; * EXIT    : Digitized 8-bit analog value in W                 *
; ***************************************************************
GET_ANALOG
      movwf  TEMP          ; Copy of Channel number in TEMP
      bcf    STATUS,C      ; Shift channel number left 3 places
      rlf    TEMP,f
      rlf    TEMP,f
      rlf    TEMP,w        ; and copy into W
      bcf    ADCON0,CHS0   ; Zero channel bits
      bcf    ADCON0,CHS1
      bcf    ADCON0,CHS2
      addwf  ADCON0,f      ; Moves Channel number to ADCON0[5:3]
      call   DELAY_17US    ; Wait 17us to stabilize

      bcf    INTCON,GIE    ; Disable all interrupts
      bcf    PIR1,ADIF     ; Ensure the AD Int flag is 0 before
      bsf    ADCON0,GO     ; starting the conversion

      sleep                ; Doze in quiet while converting

      bsf    INTCON,GIE    ; Re-enable interrupts (optional)
      movf   ADRESH,w      ; Fetch byte when awake
      return
```

The Sleep version of GET_ANALOG shown in Program 14.2 is virtually identical to the original version, with a the following changes.

1. GIE may need to be cleared if other devices can request an interrupt.
2. Before the conversion is started, the ADIF flag is cleared to ensure that the Sleep state is not prematurely terminated.
3. A `sleep` instruction follows the setting of the GO/$\overline{\text{DONE}}$ switch. Where the local clock option is selected, an extra t_{AD} period is automatically inserted to ensure that conversion only begins after the `sleep` instruction has been executed.
4. There is no need to poll the GO/$\overline{\text{DONE}}$ status flag, as the PIC MCU will only restart after the conversion has completed and will then execute the following instruction. In our example the GIE mask bit has been cleared, and it should then be set again to 1 if there is to be interrupt activity from other sources. If GIE is permanently left at 1 then the processor will automatically jump to an ISR after it awakens.

For our final example we are going to code a 20 MHz PIC16F874 in CCS **C** to act as a magnitude comparator in the manner of Example 11.2 on page 319. Here we want to measure up the parallel-input 8-bit word N at Port B against an analog input at Channel 1. Outputs at `RC[2:0]` are to represent Analog Lower Than N (b'001'), Equivalent (b'010') and Higher Than N (b'100') respectively. The comparator is to have a hysteresis of $\Delta = \pm 1$ bit; called `delta` in our program. That is, if a previous comparison showed Analog $< N$ then the new trigger level is $N + 1$. Similarly, on a downward trajectory the trigger level is decreased to $N - 1$.

The function `compare()` of Program 14.3 assumes that initialization code of the form:

```
#include <16f874.h>
#byte PORT_B = 0x06
#byte PORT_C = 0x07
#device ADC=8            /* Configure for an 8-bit outcome  */
/* Declare function to which is send delta (+1 or -1)
and which returns updated value +1 or -1                    */
unsigned int compare(unsigned int delta);

int main()
{
unsigned int hysteresis = 0;
set_tris_c(0xF8);
setup_adc(ADC_CLOCK_DIV_32);
setup_adc_ports(RA0_RA1_RA3_ANALOG);
set_adc_channel(1);
```

has already been executed.

The key internal functions used here are:

`setup_adc(ADC_CLOCK_DIV_32);`
This function configures bits ADCS[1:0] in ADCON0[7:6] to select the module's clock source; here the processor oscillator/32. The script ADC_CLOCK_INTERNAL may be used to select the internal CR oscillator.

`setup_adc_ports(RA0_RA1_RA3_ANALOG);`
This configures bits PCFG[3:0] in ADCON1[3:0] to select which port pins are analog, which are digital, and if external reference voltages are to be used. The script RA0_RA1_RA3_ANALOG indicates that port lines RA3 and `RA[1:0]` are to be analog with internal reference voltages, with the rest being digital — PCFG[3:0] = b'0100'; see Fig. 14.11. The equivalent script using an external V_{ref+} at RA3 is RA0_RA1_ANALOG_RA3_REF. Scripts appropriate to any particular device are stored in the corresponding header file; in this case `16f874.h`. All devices with an ADC module have scripts ALL_ANALOG and NO_ANALOGS.

`set_adc_channel(n);`
This is used to set up the channel number bits CHS[2:0] in ADCON0[5:3].

`read_adc();`
This activates GO/$\overline{\text{DONE}}$ in ADCON0[2] and returns with the digitized value from ADRESH:L when GO/$\overline{\text{DONE}}$ goes to 0.

`#device ADC=8`
This directive configures the ADC module to left align the 10-bit outcome (see Fig. 14.13) and is used by the function `read_adc()` to return an 8-bit `int`, which it gets from ADRESH. The directive `device ADC=10` returns a `long int` from ADRESH:L.

Program 14.3 A digital/analog comparator with hysteresis.

```
unsigned int compare(unsigned int delta)
{
unsigned int analog;
analog = read_adc();
if(analog > PORT_B + delta) {PORT_C = 0x04; delta = 0xff;}
if(analog == PORT_B) {PORT_C = 0x02;}
else {PORT_C = 0x01; delta = 1;}
return delta;
}
```

The function `compare()` in Program 14.3 expects the value of the hysteresis, called `delta`, which here is either +1 or −1 (`h'FF'`). After the ADC module is read, the digitized value `analog` is compared with the contents of Port B plus `delta` and the three Port C bits (`RC[2:0]`) set to their appropriate state.

At the same time as the comparison is resolved, `delta` will be updated to reflect the outcome (i.e., +1 if `analog` $<$ (`PORT_B` + `delta`), −1 if `analog` $>$ (`PORT_B` + `delta`)). The value `delta` is returned by the function to allow the caller function to update its variable; called, say, `hysteresis`. Thus to activate the comparator outputs and also update `hysteresis` at the same time, the caller might have a statement such as `hysteresis = compare(hysteresis);`. An alternative would be to define the variable `hysteresis` before the main function `main()` making it global; that is, known to all functions. In this situation its value need not be passed by the caller back and forth to any appropriate function.

Conversion from a digital quantity to an analog equivalent is somewhat simpler than the converse and not so commonly required. Perhaps for these reasons digital-to-analog converters (DACs) are not often found as an integral function in most MCU families.

We have already seen that one way of providing this mapping is to vary the mark:space ratio of a pulse train of constant repetitive duration, as shown in Fig. 13.9 on page 424. Here a small digital number gives a skinny pulse, which when smoothed out by a low-pass filter (which gives the average or d.c. value) translates to a low voltage. Conversely, a large digital number leads to a correspondingly large mark:space ratio, which in turn, after smoothing, yields a higher voltage.

PWM conversion can be very accurate and is simple to implement. However, extensive filtering is required to remove harmonics of the pulse rate and this makes the conversion slow to respond to changes in the digital input. Normally PWM is used to control heavy loads, such as motors or heaters, where the inertia of these devices inherently provides the smoothing action. Furthermore, the pulsed nature of the signal is ideally suited to power control, activating thyristor firing circuits.

Another way is to switch in a tapping on a chain of resistors, each adding one least significant bit increment to the grand total. This is the principle used in the Comparator Voltage Reference module of Fig. 14.7. However, rather a lot of resistors are needed; e.g., 1024 for 10-bit resolution.

Many commercial DAC devices are available which can be controlled externally. Two examples were given in Figs. 12.3 and 12.5 on pages 336 and 340, where the MCU transferred digital data in series. Here for completeness, we will look at an example where parallel data transfer is used.

The majority of proprietary devices are based on an R-2R ladder network, such as that shown in Fig. 14.15(a). Voltage appearing at any bit switch node emerges at the output node in an attenuated form. As our analysis will show, each move to the left attenuates this voltage $\mathsf{b_i}$ by 50%, which is the binary weighting relationship:

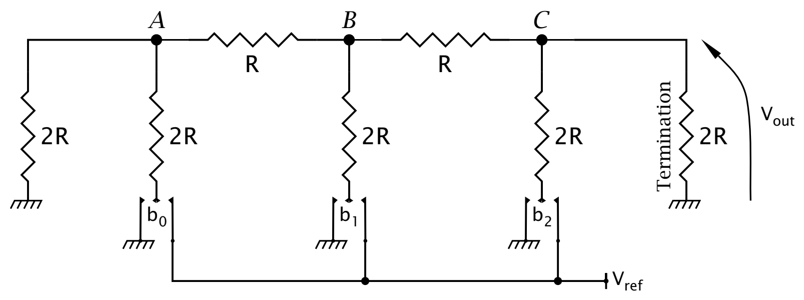
$$V = \sum_{i=0}^{N+1} \mathsf{b_i} \times 2^i$$

for an N-bit word.

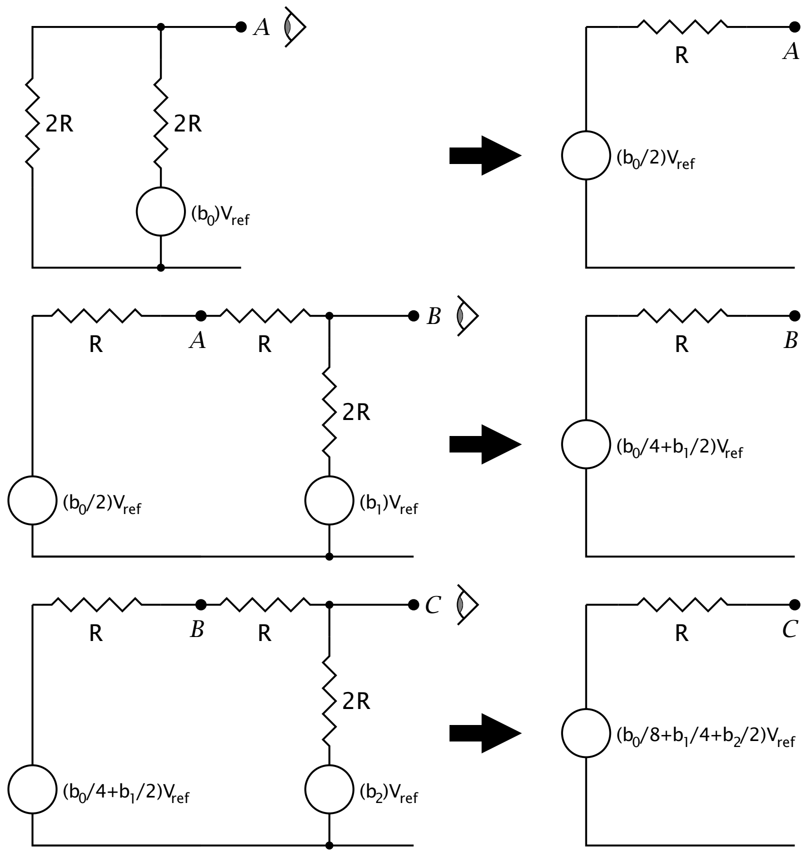
In Fig. 14.15(b), at node A looking to the left we see a resistance of R (2R//2R) and the voltage is attenuated by two. As we move to the right the process is repeated with each voltage divided by two. Thus, at node B the voltage $\mathsf{b_0}/2$ is further divided by two as is voltage $\mathsf{b_1}$, giving $\mathsf{V_B}$= $\mathsf{b_0}/4$ + $\mathsf{b_1}/2$. As the network is symmetrical the resistance looking right at any mode is also 2R. This means that as seen from *any* digital switch, the total resistance is 2R + 2R//2R = 3R. This is important as the characteristics of a transistor switch, such as resistance, are dependent on current, and keeping this the same reduces error.

For clarity our analysis has been for three bits. This can be extended by simply moving the leftmost terminating resistor over and inserting the requisite number of sections. This does not affect the resistance as seen left of the mode, and therefore does not change the conditions of the rightmost sections. An inspection of our analysis shows that nowhere does the absolute value of resistance appear. In fact the accuracy of the analysis depends only on the R:2R ratio. While it is relatively easy to fabricate accurate ratioed resistors on a silicon die, this is certainly not the case for absolute values. For this reason R:2R networks are the standard technique used for most integrated circuit DACs.

The Maxim MAX506 of Fig. 14.16 is an example of a commercial D/A converter (DAC). This 20-pin footprint device contains four separate DACs sharing a common external $\mathsf{V_{ref}}$. Digital data is presented to the `D[7:0]` pins and one of four latch registers selected with the `A[1:0]` address inputs. Once this is done, the datum byte is loaded into the selected register n and appears at the corresponding output VOUTn.



(a) A 3-bit R-2R ladder network



(b) Reducing the circuit

Fig. 14.15 R-2R digital-to-analog conversion.

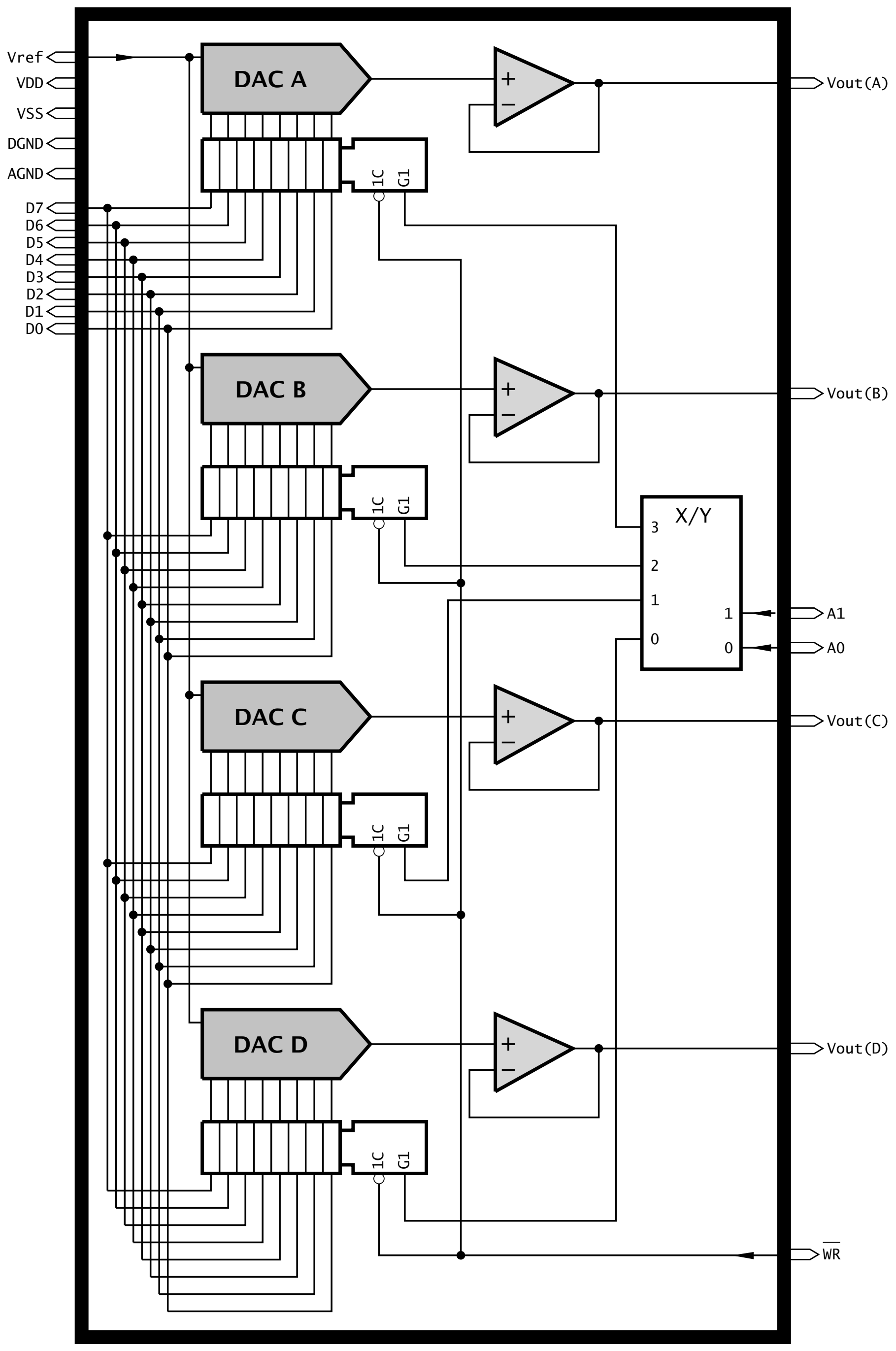


Fig. 14.16 The Maxim MAX506 quad 8-bit D/A converter.

This output analog voltage ranges from zero (Analog GrouND) for a digital input of h'00' through to V_{ref} for a digital input of h'FF'. Where V_{SS} is connected to ground, then V_{ref} can be anything between 0 V and V_{DD} (+5 V). However, V_{SS} can be as low as −5 V and in this situation V_{ref} can be anywhere in the range ±5 V. If V_{ref} is negative for dual supplies then the output voltage will also be negative. In either case, effectively the output can be treated as the product $D \times V_{ref}$ where D is the digital input byte scaled to the range 0–1 (h'00–FF').

The MAX505 is a 24-pin variant which permits separate reference voltages to be used for each of the four DAC channels. In addition, the MAX505's DAC latches are isolated from the converter ladder circuits by a further layer of latches all clocked at the *same* time with a $\overline{\text{LDAC}}$ ($\overline{\text{Load DAC}}$) control signal. This double buffering permits the programmer to update all four DACs simultaneously after their individual latches have been set up.

As an example, consider that a MAX506 quad DAC has its Address selected via RA[1:0] and RA2 drives the $\overline{\text{WR}}$ input to latch in the addressed data from Port B. We need to generate the continuous staircase sawtooth waveform shown in Fig. 14.17 from DACD. A suitable software routine would be something like the following listing:

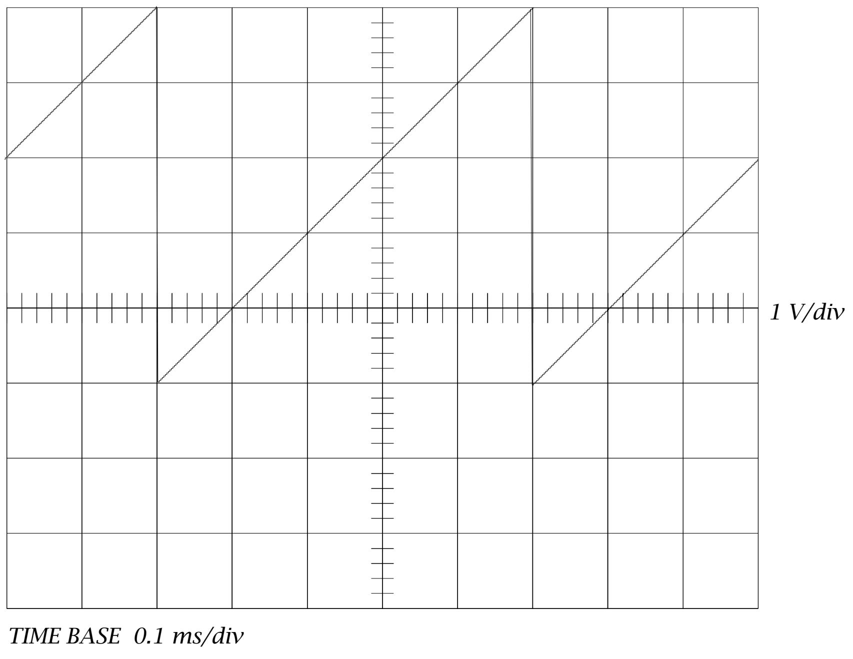


Fig. 14.17 Generating a continuous sawtooth using a MAX506 DAC.

```
      movlw   b'0111'     ; DACD is channel 3 (b'11'), WR = 1
      movwf   PORTA       ; To MAX506 WR, A1:0
LOOP  movwf   PORTB       ; Datum to MAX506's D7:0
      bcf     PORTA,2     ; WR = 0; Latch datum in
      bsf     PORTA,2     ; WR = 1; by pulsing WR
      addlw   1           ; Increment staircase count
      goto    LOOP        ; and repeat forever
```

where we are assuming that Port B and Port A[2:0] have been set up as outputs.

The typical DAC staircase output waveform shown in the oscillogram in Fig. 14.17 is based on a 12 MHz crystal clocked PIC MCU. With a loop cycle count of six cycles gives a sawtooth duration of $(256 \times 6)/3 \approx 0.5$ ms, at 2 μs per step.

Examples

Example 14.1

The analog input channel voltage range for most ADC modules[8] is limited to the positive range 0 – V_{ref+}, where V_{ref+} can either be the internal V_{DD} voltage or an external voltage at RA3 in the range 3 – V_{DD}. Many situations require a digitized mapping from bipolar analog signals. Design a simple resistive network to translate a bipolar voltage range of ±10 V to a unipolar range of 0 – 5 V, assuming V_{ref+} is +5 V. Extend the design to give an anti-aliasing filter, assuming a sampling rate of 5000 per second.

Solution

One possibility is shown in Fig. 14.18. The value of the three resistors must be such that the input voltage V_{in} range of ±10 V will be shifted so that the midpoint of 0 V gives half-scale ($V_{ref+}/2 = 2.5$ V) at the input pin AN. The range at this pin must also be attenuated by a factor of 4. A more general way of expressing this is given by the relationship $V_{in} = \pm G \times V_{ref+}$.

1. When V_{in} is zero, the voltage at the summing node is half-scale, which maps to b'10000000'. To do this, R_1 paralleled with R_2 must have the same resistance as R_3, i.e.,
   $$R_3 = R_1 // R_2$$
2. The attenuation of the network is a function of the potential divider between R_1 and $R_2 // R_3$. This gives us the value of G as:
   $$2G = (R_1 + (R_2 // R_3)) / (R_2 // R_3)$$

Where in our instance $G = 2$.

After some manipulation we have:

$$\begin{aligned} R_1 &= (G - 1) \times R_2 \\ R_2 &= G \times R_3 \end{aligned}$$

[8]The PIC16C77X devices can be configured to accept bipolar input analog voltages.

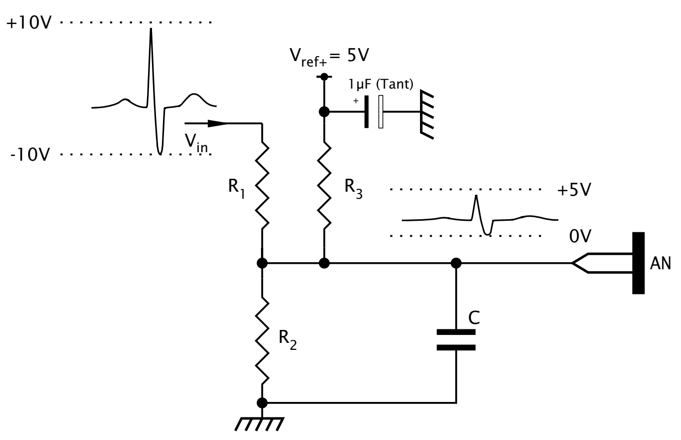


Fig. 14.18 A level-shifting resistor network.

Of course we have three unknowns and only two equations, so we have to start off by choosing a value for one of them. If we pick a value of 5 kΩ for R_3, then we have $R_2 = 2 \times 5 = 10\,k\Omega$ and $R_1 = 10\,k\Omega$.

The resistance looking out from the pin is all three resistors in parallel; which in our case is 2.4 kΩ. This meets the maximum to keep within a LSB leakage error for a 10-bit conversion. For 8-bit resolution, the resistor values could be increased by a factor of four.

A small capacitor at the summing node can be used to implement a simple first-order low-pass filter to attenuate high frequencies from external sources, such as the MCU's system clock, and act as an anti-aliasing filter, as described in Fig. 14.4. With a sampling rate of 5000 per second, then ideally the filter break frequency should be no more than 2.5 kHz—half the sampling frequency. As this filter has an attenuation of only 6 dB per octave, choosing a break frequency $\frac{1}{2\pi CR}$ of 1 kHz provides a generous margin. We then have:

$$\begin{aligned}\frac{1}{2\pi CR} &= 1000\\ C &= \frac{10^{-6}}{4.8\times\pi}\\ C &\approx 66\,\text{nF}\end{aligned}$$

To further reduce noise, the filter capacitor should have good high-frequency characteristics, e.g., polyester (capacitors become inductors at high frequencies) and together with the resistors, be physically as close as possible to the pin and not adjacent to any digital lines. It is always good practice to decouple the reference voltage and power supply with a low-value Tantalum electrolytical capacitor or/and a 0.1 μF ceramic capacitor to reduce switching noise from the MCU and other devices taking power from the same source. Using a separate supply and ground connection to the power supply to the PIC MCU should also reduce noise from this source.

Example 14.2
As part of a smart biomedical monitor, the peak analog value of an electrocardiogram (EKG) signal is to be determined anew for each cycle. This R-point (see Fig. 7.1 on page 186) maximum value is to be output from Port B and RA5 is to be pulsed High whenever this value is being updated. Assuming that a PIC16F87X is used to implement the intelligence, and the EKG signal (conditioned as shown in Fig. 14.18) is connected to Channel 1 RA1, devise a possible strategy. Timer 0 is being used to interrupt the processor at nominally 2000 times per second; see Program 13.2 on page 411. Design a suitable ISR to implement your strategy.

Solution
As in any biomedical parameter the EKG signal will vary from cycle to cycle in gain, shape, and period. Even if this were not so, imperfections in the data acquisition system, notably the skin electrodes, can cause slow baseline (d.c.) drift. Thus the threshold at which the signal is to be tracked to its peak R-value must be reset at some sensible fraction of its previous peak during the period following the last update.

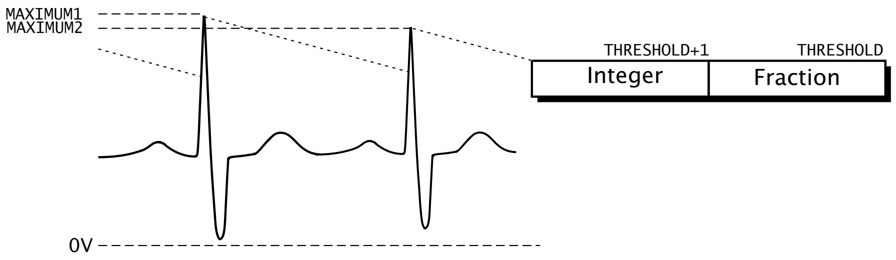


Fig. 14.19 EKG detection strategy.

One possibility is shown in Fig. 14.19. Here the threshold is slowly decremented after the peak to ensure that a following peak of lower amplitude is not missed. On the basis of a lowest EKG rate of 40 beats per minute (period 1.5 s), if we reduce the threshold by $\frac{1}{64}$ of a bit every sample then the maximum reduction would be a count of ≈ 47 at a sample rate of 2000 per second. To do this the threshold value THRESHOLD in Program 14.4 is stored as a double-byte number of form integer:fraction and $\frac{1}{64}$ of an integer (i.e., fraction = b'00000100') subtracted in each sample where the peak value MAXIMUM is not updated. This droop rate can be altered by changing the subtracted fraction.

The task list implemented by this listing is:

1. DO a conversion to get ANALOG.
2. IF (ANALOG > THRESHOLD)
    - MAXIMUM = ANALOG.
    - THRESHOLD = ANALOG.
    - PORTB = ANALOG.
    - RA5 = 1.
3. ELSE
    - Reduce THRESHOLD by $\frac{1}{64}$.
    - RA5 = 0.

In updating THRESHOLD (where ANALOG $>$ THRESHOLD) the integer byte takes the new value of MAXIMUM whilst the fractional byte is zeroed. Treating this byte pair as a 16-bit word, this effectively equates the threshold as MAXIMUM $\times$ 256 or THRESHOLD $=$ MAXIMUM $<<$ 8, where MAXIMUM has been shifted left eight places. We are assuming that THRESHOLD has been zeroed in the background program during the initialization phase and that we are doing an 8-bit conversion.

If the digitized analog sample is less than the threshold trip value then h′04′ = b′00000100′ is subtracted from the lower byte at THRESHOLD+1 and if this produces a borrow, then the upper byte at THRESHOLD is decremented. This subtract $\frac{1}{64}$ routine is skipped if the threshold has reached zero, thus preventing underflow.

Program 14.4 uses the subroutine GET_ANALOG of Program 14.1 and its associated 17 μs delay subroutine. However, as there is a considerable period between calls, the 17 μs delay can be reduced to 10 μs if required.

Program 14.5 gives the **C** coded version implementing our task list. The #int_rtcc directive tells the compiler to treat the following function as a Real-Time Counter Clock (Timer 0) ISR. In function ecg_isr(), the variables threshold and maximum are declared static. This means that their value will be retained after the function has exited and will be available next time on entry. The default way of treating **C** function variables is to hold their value only for the duration of the function. An alternative way of dealing with this problem is to declare such variables outside any function, in which case they will be global and retain their value indefinitely.

The threshold variable is defined as a long int and the CCS compiler will then treat this datum as a 16-bit variable as required. The definition in equating threshold to zero will only initialize it once when the program begins its run, as the variable is static. Again this is not the normal behavior of the default auto variable.

In equating threshold to the new maximum value, the latter is multiplied by 256 by shifting left eight times. A good compiler will automatically change a N*256 to N<<8; or even better just take the upper byte of

Program 14.4 EKG peak picking.

```
; ***************************************************************
; * FUNCTION: ISR to update the EKG parameters                  *
; * ENTRY   : On a Timer0 interrupt                             *
; * EXIT    : Update MAXIMUM and THRESHOLD:THRESHOLD+1          *
; * RESOURCE: GET_ANALOG subroutine gets 8-bit digitized data *
; ****************************************************************
; First save context
EKG_ISR  movwf   _work         ; Put away W
         swapf   STATUS,w      ; and the Status register
         movwf   _status

; ===============================================================
         btfss   INTCON,T0IF   ; Was this a Timer0 interrupt?
          goto   EKG_EXIT      ; IF not THEN exit

         bcf     INTCON,T0IF   ; ELSE clear flag
         movlw   1             ; Initiate a conversion of
         call    GET_ANALOG    ; Channel 1

         movwf   TEMP          ; Save digitized byte
         subwf   THRESHOLD,w   ; THRESHOLD - ANALOG
         btfsc   STATUS,C      ; IF no Borrow THEN
          goto   BELOW         ; don't update MAXIMUM
         movf    TEMP,w        ; ELSE get digitized value
         movwf   MAXIMUM       ; which is the new MAXIMUM
         movwf   PORTB         ; made visible to outside
         bsf     PORTA,5       ; which is signaled
         movwf   THRESHOLD     ; Now update double-byte
         clrf    THRESHOLD+1   ; threshold
         goto    EKG_EXIT      ; and finish

; Land here if the input is below the threshold
BELOW    bcf     PORTA,5       ; Signal no update
; Now reduce the threshold by 0.5 unless it is zero
         movf    THRESHOLD,f   ; Is integer threshold zero?
         btfsc   STATUS,Z      ; Skip if not
          goto   EKG_EXIT      ; IF it is THEN leave alone

         movlw   h'04'         ; 1/64 = b'00000100'
         subwf   THRESHOLD+1,f ; Take away from fraction byte
         btfss   STATUS,C      ; Skip if no borrow
          decf   THRESHOLD,f   ; ELSE decrement integer thresh
; ===============================================================

EKG_EXIT swapf   _status,w     ; Untwist the original Status reg
         movwf   STATUS
         swapf   _work,f       ; Get the original W reg back
         swapf   _work,w       ; leaving STATUS unchanged
         retfie                ; and return from interrupt
```

Program 14.5 An implementation of the EKG peak picker in **C**.

```
#bit  RA5    = 5.5              /* Pin RA5 is bit 5 of Port A */
#byte PORT_B = 6                /* Port B is File 6           */

#int_rtcc
ecg_isr()
{
unsigned int analog;
static unsigned long int threshold = 0;
static unsigned int maximum;
analog = read_adc();

if(analog > (threshold>>8))
    {
    maximum = analog;           /* New maximum value          */
    PORT_B  = analog;           /* Show the outside world     */
    threshold = maximum << 8;   /* New 2-byte threshold       */
    RA5 = 1;                    /* Tell outside world         */
    }
else
    {
    threshold = threshold - 0x0004; /* Reduce by 1/64         */
    RA5 = 0;                    /* Signal no update           */
    }
}
```

the pair as the outcome. This double-byte form allows for the reduction by $\frac{1}{64}$ of a bit h'0004' to give the specified falling trip level.

Example 14.3

A microcontroller is to be used to calculate a measure of power discharged by the diphasic defibrillator of Fig. 14.5. When the MCU detects the beginning of the discharge, 256 samples are to be taken at a nominal rate of 20,000 per second, with the sum of the squares of the deviation from the baseline voltage being an analog measure of the power—assuming that the resistance of the patient's chest/electrodes remaining constant whilst all this is going on!

A 4.096 V voltage reference device is to act as an external reference voltage for a ADC module, giving a 16 mV resolution for an 8-bit conversion. After the process begins, pin RA4 is to be pulsed as a trigger for a storage oscilloscope, which allows the waveform to be captured for archiving purposes. When the process has been completed, the top byte of the power summation is to be output via Port B for display.

Show how you might use a 20 MHz PIC16F87XA device to implement the logic of the measurement system. You can assume that the voltage reference device can be biased as for a Zener diode. In practice an optional potentiometer can be used to trim the voltage slightly for more accurate results.

Solution

Figure 14.20 shows a suitable hardware configuration, from which we can estimate the peripheral budget. The signal itself ranging between +1.8 and +3.6 V (see Fig. 14.5) is connected to Analog channel 0 at pin RA0/AN0. The 10 kΩ resistor protects the analog input against overvoltage as well as implementing an anti-aliasing filter, with the 3.3 nF capacitor giving a 450 kHz nominal breakpoint. As the actual defibrillator uses very large voltages (of the order of 25 kV) the two 1N4004 diodes act as additional protection against high-voltage spikes, supplementing the internal diodes shown in Fig. 14.12.

The external 4.096 V reference voltage is directly connected to pin RA3, which for the ADC module configuration mode b'0101' is used as the

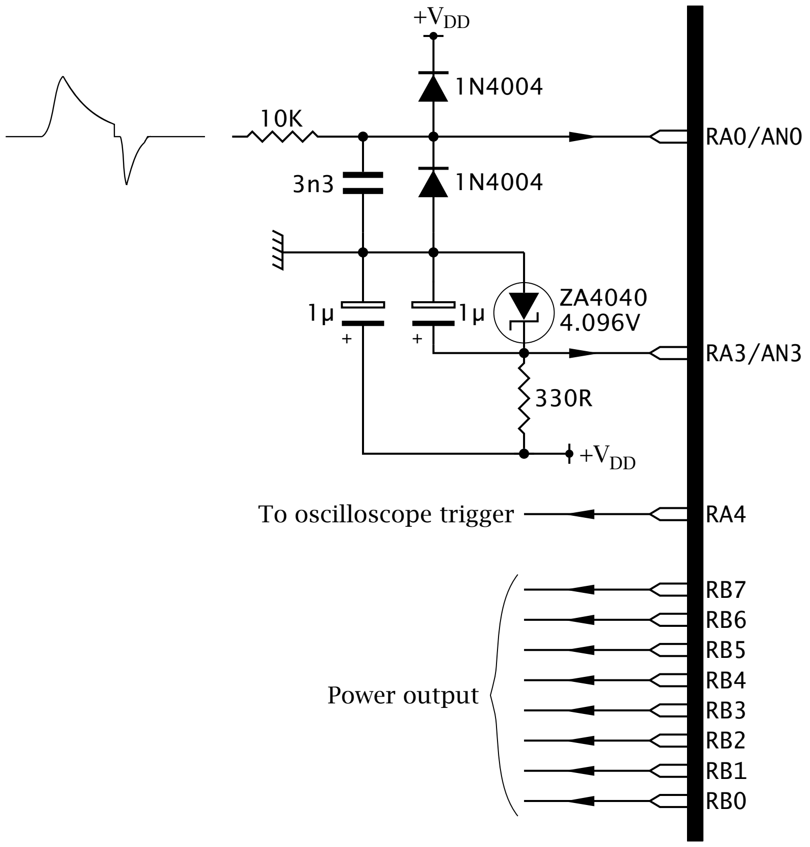


Fig. 14.20 Measuring the discharge power for an EKG defibrillator.

external reference. Both the V_{DD} and V_{ref+} voltages are decoupled with 1 μF tantalum capacitors to reduce noise at this point.

An internal analog comparator is used to detect the initial rise of the discharge voltage as described in Fig. 14.5. If Comparator module mode b'1110' is used then the CVREF module can be used to generate an internal reference voltage as described on page 447. With the CIS bit cleared (see Fig. 14.6) Comparator 1 can share Analog channel 0 with the ADC module.

Finally, both RA4 and all of Port B must be configured as digital outputs. The former is going to be used to generate a synchronization pulse, and the latter to output the end result of the analysis.

```
       include "p16f877a.inc"

       org   0             ; Reset vector
       goto  SET_UP        ; On reset go to background routine
       org   4             ; Interrupt vector
       goto  EKG_ISR       ; On Comparator change go to ISR

SET_UP bsf    STATUS,RP0   ; Change to Bank 1
       movlw  b'00000110'  ; Comparator mode 110 CIS = 0
       movwf  CMCON
       call   DELAY_17US   ; Allow 17us for voltages to settle
       movf   CMCON,f      ; Read CMCON to clear Change state
       bsf    PIE2,CMIE    ; Enable Comparator interrupts

       movlw  b'10001110'  ; CVREF module on (1), not extern (0
       movwf  CVRCON       ; Hi range (0), CVR[3:0] = 1110

       movlw  b'00000101'  ; RA0/1 analog inputs
       movwf  ADCON1       ; RA3 is Vref+ input

       movlw  b'101111'    ; Make RA4 an output
       movwf  TRISA
       clrf   TRISB        ; PortB is all output

       bcf    STATUS,RP0   ; Back to Bank 0

       movlw  b'10000001'  ; Clock /32 turn on ADC module
       movwf  ADCON0

       bcf    PIR2,CMIF    ; Zero the Comparator interrupt flag
       bsf    INTCON,PEIE  ; Enable Peripheral interrupt group
       bsf    INTCON,GIE   ; & Globally enable interrupt system
```

Based on our analysis, the initialization code is shown above. The modules are set-up as follows:

1. The Analog comparator module is turned on in Mode b'110' with CIS = 0. For convenience, the 17 μs delay subroutine DELAY_17US used by GET_ANALOG is also employed to allow the module to settle, rather than a separate 10 μs delay. After this, the CMCON register is read to clear any Change condition. This allows the CMIF flag to be subsequently zeroed and interrupts enabled.

2. The CVREF module is enabled and set to tapping b'1110' in the high range to give a 3.4375 V reference.
3. The ADC module is enabled and set to mode b'0101' to configure pin RA0 as an analog channel (and also RA1) and use RA3 for the external V_{ref+}. Alignment is set-up to facilitate an 8-bit outcome. The ADC clock is sourced as the system 20 MHz frequency divided by 32. This gives a 625 kHz rate, as shown in Table 14.2.
4. PORTA[4] is set-up as an output. Other Port A pins are left as inputs, as required for AN0, AN1 and AN3. All of Port B is configured as output.

The actual software itself is shown in Program 14.6. The Main routine itself simply sleeps until the Comparator module changes state and generates an interrupt. When control returns to the background routine, the top byte of the triple-byte Power accumulator is copied to Port B and the process repeated for the next run.

After saving the context in the normal way, the foreground routine first confirms the source of the interrupt and then clears a loop counter and the three bytes used to store the grand sum of the 256 squares of the sampled voltage. Pin RA4 is then pulsed to tell the outside world that the discharge is beginning.

The GET_ANALOG subroutine listed in Program 14.1 is used to acquire an 8-bit digitized sample. The difference between the baseline voltage of 2.6 V (see Fig. 14.5) is then determined. If this is negative; that is, a Borrow is generated after the subtraction showing that the input voltage is below 2.6 V, then the difference byte is 2's complement inverted (see page 9) to switch from negative to positive. This modulus voltage is then squared using the SQR routine of Program 8.3 on page 232. The two global return bytes SUM:SUM+1 are then added to the triple-byte total POWER:POWER+1:POWER+2 array.

This is repeated 256 times with an extra loop delay of 470 μs to give an approximate 500 μs total delay necessary to give the specified 2 kHz sampling rate. When this has been completed, taking a total time of around 128 ms, the Comparator module is read to clear the difference condition. This is done at the end of the process, rather than at the beginning, as the input voltage will fall back through the 3.4375 V Comparator threshold part way through the process and trigger another change! The CMIF flag is then cleared and the context, restored.

Of course this is rather rudimentary. For instance, the baseline voltage may vary with time, so a learning run prior to an analysis may be necessary. If fairly stable, this value can be burnt into non-volatile memory as described in the next chapter. The use of a fixed number of samples can be restrictive, and additional loops can be implemented until the voltage difference drops below a certain threshold.

Program 14.6 Gauging the defibrillator discharge power.

```
MAIN     sleep              ; Idle
         nop
         movf  POWER,w      ; Get top byte of power
         movwf PORTB        ; and output
         goto  MAIN
; ************************************************************
; * FUNCTION: ISR to begin the defibrillator analysis        *
; * ENTRY   : On a Comparator module interrupt               *
; * EXIT    : Update POWER:3                                 *
; * RESOURCE: GET_ANALOG subroutine gets 8-bit digitized data *
; * RESOURCE: SQUARE subroutine does 8x8 multiplication      *
; ************************************************************
; First save context
EKG_ISR   movwf  _work       ; Put away W
          swapf  STATUS,w    ; and the Status register
          movwf  _status
; ============================================================
          btfss  PIR2,CMIF   ; Was this a Comparator interrupt?
           goto  EKG_EXIT    ; IF not THEN exit
          bcf    PIR2,CMIF   ; ELSE clear the interrupt
          clrf   POWER       ; Zero the 3-byte grand total
          clrf   POWER+1
          clrf   POWER+2     ; LSB
          clrf   COUNT       ; Prepare to do loop 256 times
          bcf    PORTA,4     ; Pulse pin RA4
          bsf    PORTA,4     ; to generate a synch pulse
          bcf    PORTA,4
ACQUIRE   clrw               ; Analog channel 0 (W is h'00')
          call   GET_ANALOG  ; Do a conversion
          addlw  -BASELINE   ; Difference from baseline voltage
          btfsc  STATUS,C    ; IF Borrow (C==0) THEN skip
           goto  EKG_CONT    ; as difference is positive
          xorlw  b'11111111' ; ELSE invert
          addlw  1           ; plus one makes -ve diff positive
EKG_CONT  call   SQR         ; Square it
          movf   SUM+1,w     ; Get LSB of squared voltage
          addwf  POWER+2,f   ; Add it to the low byte of Power
          btfss  STATUS,C    ; Check for a Carry
           goto  NEXT_BYTE   ; IF not THEN add next byte
          movlw  1           ; Increment the high byte of Power
          addwf  POWER+1,f
          btfsc  STATUS,C    ; Check; did this generate a Carry?
           incf  POWER,f     ; IF yes THEN increment upper byte
NEXT_BYTE movf   SUM,w       ; Get MSB of squared voltage
          addwf  POWER+1,f   ; Add it to the high byte of Power
          btfsc  STATUS,C    ; Check for a Carry; IF yes
           incf  POWER,f     ; THEN increment the Upper byte
          call   DELAY_470US ; Hang around until the next sample
          incfsz COUNT,f     ; Increment the loop count and do
           goto   ACQUIRE    ; another acquisition if not zero
; ============================================================
EKG_EXIT  bsf     STATUS,RP0 ; First clear the Comparator
          movf    CMCON,f    ; Change situation
          bcf     STATUS,RP0 ; by reading it in Bank 1
          bcf     PIR2,CMIF  ; and clear the interrupt flag
          swapf   _status,w  ; Untwist the original Status reg
          movwf   STATUS
          swapf   _work,f    ; Get the original W reg back
          swapf   _work,w    ; leaving STATUS unchanged
          retfie             ; and return from interrupt
```

Example 14.4
Using **C** coding show how a 10-bit digitized reading from Channel 3 of a PIC16F874 can be acquired with the processor in its Sleep state.

Solution

The CCS compiler uses the `sleep()` function to put the MCU to sleep; this simply translates to the `sleep` instruction. A Sleep conversion cannot be implemented using the `read_adc()` function of Program 14.3 as no processing is done in the Sleep state. Instead we need to set and clear individual interrupt related bits before going to sleep in the manner outlined in the assembly-level Program 14.2. On wakening the state of ADRESH:L registers can then be read "manually" and combined to give the 10-bit outcome.

Coding for this specification is shown in Program 14.7. Here the PEIE, ADIF and GO/$\overline{\text{DONE}}$ bits are defined using the `#bit` directive. This time the script ADC_CLOCK_INTERNAL is used with the `setup_adc()` internal function to select the internal CR clock for the DAC module, as necessary for the Sleep conversion.

Program 14.7 Sleep conversion in **C**.

```
#include <16f874.h>
#device ADC=10       /* Configure for a 10-bit outcome       */
#use delay(clock=8000000) /* Tell compiler it's an 8MHz clock */
#bit ADIF = 0x0C.6   /* The A/D interrupt flag in PIR1[6]     */
#bit PEIE = 0x0B.6   /* The group interrupt flag in INTCON[6] */
#bit GO  =  0x1F.2   /* The Go/NOT_DONE bit in ADCON0[2]      */

#byte ADRESH = 0x1e  /* The Result registers                  */
#byte ADRESL = 0x9E

int main()
{
unsigned long int result; /* 16-bit digitized outcome         */
set_tris_a(0x0E);
setup_adc(ADC_CLOCK_INTERNAL);
setup_adc_ports(RA0_RA1_RA3_ANALOG);
set_adc_channel(3);
delay_us(17);             /* Allow voltages to stabilize      */
disable_interrupts(GLOBAL);/* Disable all ints (GIE & PEIE=1) */
ADIF = 0;
enable_interrupts(INT_AD);
PEIE = 1;            /* Enable the auxiliary group interrupts */
/*    Code                                                    */
GO = 1;
sleep();
/* When awake read each byte with high byte x 256             */
result = (long int)ADRESH * 256 + ADRESL;
}
```

The internal function `disable_interrupts(GLOBAL)` clears *both* GIE and PEIE mask bits. The complementary `enable_interrupts(GLOBAL)` sets both bits, but we need to enable the PEIE only and leave GIE cleared. This is implemented by the "bit-twiddling" statement `PEIE=1;`. Similarly, clearing the ADIF flag is directly actioned by `ADIF=0;`. Before calling `sleep()` the statement `GO=1;`manually starts the conversion. After `sleep()` the ADRESH register is read and cast to a `long int` to ensure that the compiler treats it as a 16-bit object. Multiplying by 256 tells an intelligent compiler to treats it as the top byte of a 16-bit object. Adding ADRESL puts this in the low byte of the 2-byte outcome.

Self-Assessment Questions

14.1 In Example 14.2 the decay of the threshold level was linear. Although this is fairly effective for situations where the nominal period is known a priori and does not vary greatly, an exponential decay would be better suited where this is not the case. To generate this type of relationship a fixed *percentage* of the value at each sample point should be subtracted to give the new outcome rather than a constant. Show how you could modify Programs 14.4 and 14.5 to decrement at a rate of approximately 0.025% ($\approx \frac{1}{4096}$) on each sample and determine the time constant in terms of the number of samples.

14.2 Real-world analog signals are noisy. In practice this means that some form of filtering or smoothing is frequently required. In any circumstance, noise coming in from outside should not have any appreciable frequency components above half the sampling rate since such noise will be frequency shifted back into the baseband, as shown in Fig. 14.4. Such low-pass filtering must be applied to the signal *before* the A/D conversion, as shown in Fig. 14.20.

Although this external **anti-alias** filter must by definition be implemented using hardware circuitry (such as a CR network), noise within the passband can be smoothed out using software filtering routines. One simple approach to digital filtering is to take multiple readings and average them to give a composite outcome. For example, 16 readings summed and shifted right four times (÷16) would reduce random noise by a factor of $\sqrt{16} = 4$.

Another approach well known to statisticians, is to take a moving average; for example, of a stock price over a month interval. A comparatively efficient algorithm of this type is a 3-point average:

$$\mathtt{Array[i]} = \frac{S_n}{4} + \frac{S_{n-1}}{2} + \frac{S_{n-2}}{4}$$

where S_n is the nth sample from the analog module.

Show how you could modify the GET_ANALOG subroutine to remember the last samples from the two previous calls and return the smoothed value.

14.3 It has been proposed that as part of the EKG monitor of Example 14.2 that a MAX506 DAC be used to introduce an automatic gain control (AVC) function preceding the PIC MCU's analog input. The aim of the AVC is to keep the peak of the analog input between $\frac{3}{4}$ and $\frac{7}{8}$ full scale. How might you go about implementing this subsystem? *Hint*: Recall that each channel of a MAX506 is the product of its digital input and V_{ref} and that the latter can vary between 0 V and V_{DD}.

14.4 An input analog sinusoid signal, conditioned as shown in Fig. 14.18, is to be full-wave rectified; that is, voltages that were originally negative are to have their sign changed. Design a routine to do this assuming that the 8-bit digitized input voltage is available at ADRESH and the processed output is to be presented via Port B to a DAC.

14.5 Figure 14.21 is based on Fig. 10 of Microchip's application note AN546 *Using the Analog-to-Digital (A/D) Converter* as a means of providing an external voltage reference source for power-sensitive applications. How do you think the circuit works and what factors govern the choice of current limiting resistor?

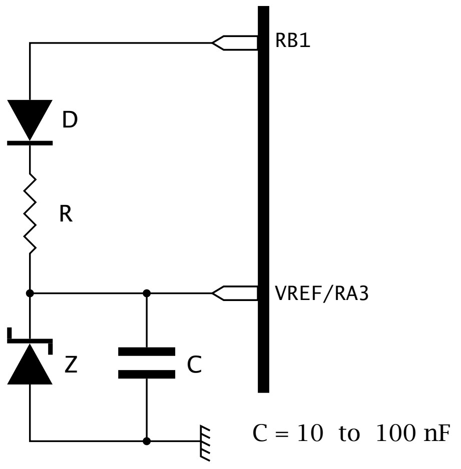


Fig. 14.21 A controllable external voltage circuit.