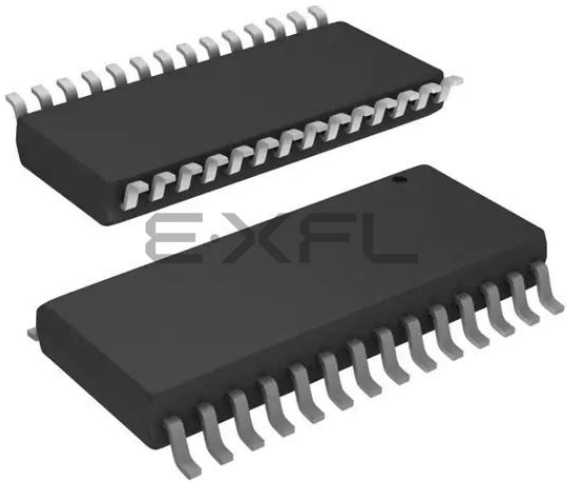


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	20MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	22
Program Memory Size	7KB (4K x 14)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	192 x 8
Voltage - Supply (Vcc/Vdd)	4V ~ 5.5V
Data Converters	-
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic16c63a-20i-so

TIP #3 Read Three States From One Pin

To check state Z:

- Drive output pin high
- Set to Input
- Read 1
- Drive output pin low
- Set to Input
- Read 0

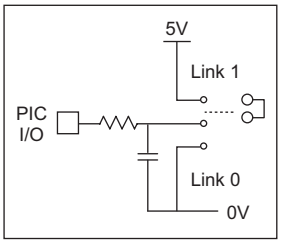
To check state 0:

- Read 0 on pin

To check state 1:

- Read 1 on pin

Figure 3-1



State	Link 0	Link 1
0	closed	open
1	open	closed
NC	open	open

Jumper has three possible states: not connected, Link 1 and Link 0. The capacitor will charge and discharge depending on the I/O output voltage allowing the "not connected" state. Software should check the "not connected" state first by driving I/O high, reading 1 and driving I/O low and reading 0. The "Link 1" and "Link 0" states are read directly.

TIP #4 Reading DIP Switches

The input of a timer can be used to test which switch(s) is closed. The input of Timer1 is held high with a pull-up resistor. Sequentially, each switch I/O is set to input and Timer1 is checked for an increment indicating the switch is closed.

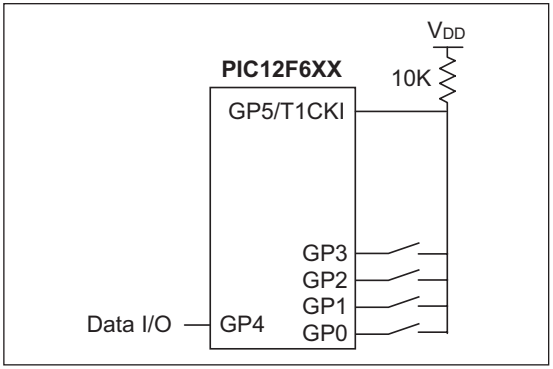
Example 4-1

```
        movlw   b'11111111'
        movwf   TRISIO
        movwf   DIP
        movlw   b'00000111'
        movwf   T1CON
        movlw   b'11111110'
        movwf   Mask
        clrf    GPIO
LOOP
        clrf    TMR1L
        movf    Mask,W
        movwf   TRISIO
        btfsc   TMR1L,0
        andwf   DIP,F
        bsf     STATUS,C
        rlf     Mask,F
        btfsc   Mask,4
        goto    Loop
        retlw   0
```

Each bit in the DP register represents its corresponding switch position. By setting Timer1 to FFFFh and enabling its interrupt, an increment will cause a rollover and generate an interrupt. This will simplify the software by eliminating the bit test on the TMR1L register.

Sequentially set each GPIO to an input and test for TMR1 increment (or 0 if standard I/O pin is used).

Figure 4-1



TIP #5 Scanning Many Keys With One Input

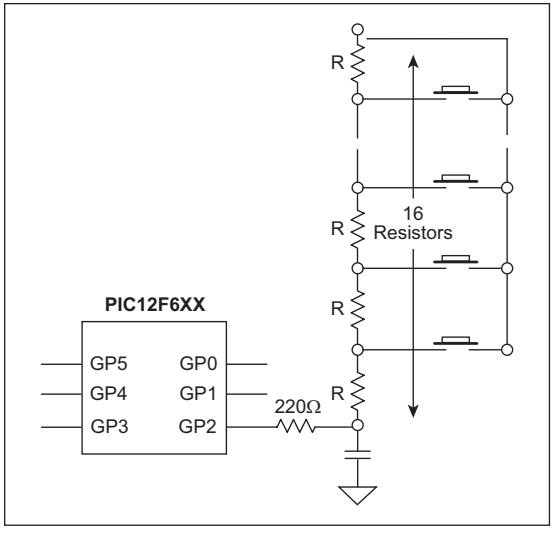
The time required to charge a capacitor depends on resistance between VDD and capacitor. When a button is pressed, VDD is supplied to a different point in the resistor ladder. The resistance between VDD and the capacitor is reduced, which reduces the charge time of the capacitor. A timer is used with a comparator or changing digital input to measure the capacitor charge time. The charge time is used to determine which button is pressed.

Software sequence:

1. Configure GP2 to output a low voltage to discharge capacitor through I/O resistor.
2. Configure GP2 as one comparator input and CVREF as the other.
3. Use a timer to measure when the comparator trips. If the time measured is greater than the maximum allowed time, then repeat; otherwise determine which button is pressed.

When a key is pressed, the voltage divider network changes the RC ramp rate.

Figure 5-1



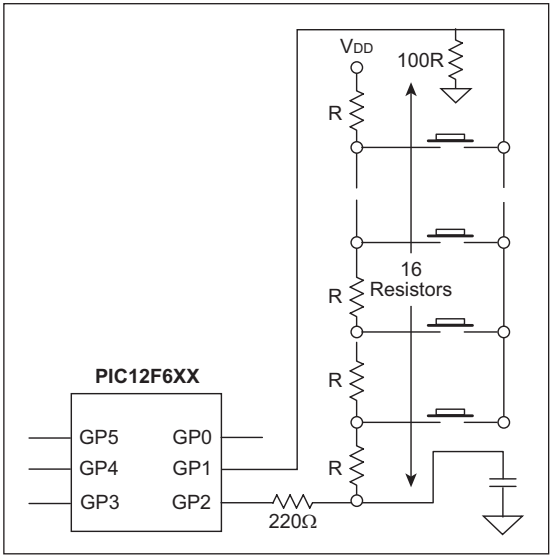
See AN512, "*Implementing Ohmmeter/ Temperature Sensor*" for code ideas.

TIP #6 Scanning Many Keys and Wake-up From Sleep

An additional I/O can be added to wake the part when a button is pressed. Prior to Sleep, configure GP1 as an input with interrupt-on-change enabled and GP2 to output high. The pull-down resistor holds GP1 low until a button is pressed. GP1 is then pulled high via GP2 and VDD generating an interrupt. After wake-up, GP2 is configured to output low to discharge the capacitor through the 220Ω resistor. GP1 is set to output high and GP2 is set to an input to measure the capacitor charge time.

- GP1 pin connected to key common
- Enable wake-up on port change
- Set GP1 as input and GP2 high prior to Sleep
- If key is pressed the PIC MCU wakes up, GP2 must be set low to discharge capacitor
- Set GP1 high upon wake-up to scan keystroke

Figure 6-1



Tip #13.2 Reading a Sensor With Higher Accuracy – Charge Balancing Method

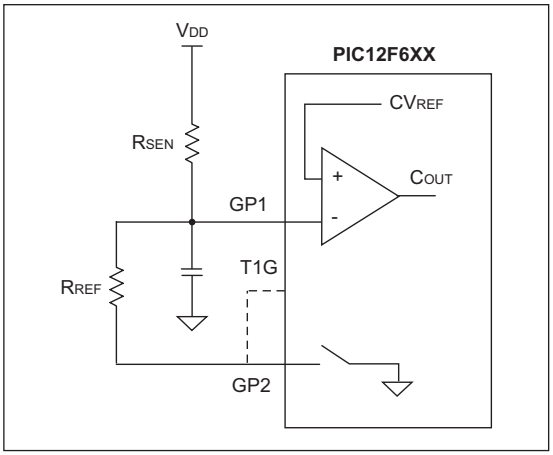
1. Sensor charges a capacitor
2. Reference resistor discharges the capacitor
3. Modulate reference resistor to maintain constant average charge in the capacitor
4. Use comparator to determine modulation

To improve resolution beyond 10 or 12 bits, a technique called “Charge Balancing” can be used. The basic concept is for the MCU to maintain a constant voltage on a capacitor by either allowing the charge to build through a sensor or discharge through a reference resistor. A timer is used to sample the capacitor voltage on regular intervals until a predetermined number of samples are counted. By counting the number of times the capacitor voltage is over an arbitrary threshold, the sensor voltage is determined.

The comparator and comparator voltage reference (CV_{REF}) on the PIC12F629/675 are ideal for this application.

1. GP1 average voltage = CV_{REF}
2. Time base as sampling rate
3. At the end of each time base period:
   - If GP1 > CV_{REF}, then GP2 Output Low
   - If GP1 < CV_{REF}, then GP2 Input mode
4. Accumulate the GP2 lows over many samples
5. Number of samples determines resolution
6. Number of GP2 lows determine effective duty cycle of R_{REF}

Figure 13-3



TIP #18 Swap File Register with W

Example 18-1

```
SWAPWF      MACRO   REG
            XORWF   REG,F
            XORWF   REG,W
            XORWF   REG,F
            ENDM
```

The following macro swaps the contents of W and REG without using a second register.

Needs: 0 TEMP registers
3 Instructions
3 TCY

An efficient way of swapping the contents of a register with the working register is to use three XORWF instructions. It requires no temporary registers and three instructions. Here’s an example:

W	REG	Instruction
10101100	01011100	XORWF REG,F
10101100	11110000	XORWF REG,W
01011100	11110000	XORWF REG,F
01011100	10101100	Result

TIP #19 Bit Shifting Using Carry Bit

Rotate a byte through carry without using RAM variable for loop count:

- Easily adapted to serial interface transmit routines.
- Carry bit is cleared (except last cycle) and the cycle repeats until the zero bit sets indicating the end.

Example 19-1

```
        LIST P=PIC12f629
        INCLUDE P12f629.INC
        buffer      equ   0x20

bsf     STATUS,C    ;Set 'end of loop' flag
rlf     buffer,f    ;Place first bit into C
bcf     GPIO,Dout   ;precondition output
btfsc   STATUS,C    ;Check data 0 or 1 ?
bsf     GPIO,Dout
bcf     STATUS,C    ;Clear data in C
rlf     buffer,f    ;Place next bit into C
movf    buffer,f    ;Force Z bit
btfss   STATUS,Z    ;Exit?
goto    Send_Loop
```

CHAPTER 2
PIC® Microcontroller Low Power Tips ‘n Tricks

Table Of Contents

TIPS ‘N TRICKS INTRODUCTION

Microchip continues to provide innovative products that are smaller, faster, easier to use and more reliable. The Flash-based PIC® microcontrollers (MCUs) are used in an wide range of everyday products, from smoke detectors, hospital ID tags and pet containment systems, to industrial, automotive and medical products.

PIC MCUs featuring nanoWatt technology implement a variety of important features which have become standard in PIC microcontrollers. Since the release of nanoWatt technology, changes in MCU process technology and improvements in performance have resulted in new requirements for lower power. PIC MCUs with nanoWatt eXtreme Low Power (nanoWatt XLP™) improve upon the original nanoWatt technology by dramatically reducing static power consumption and providing new flexibility for dynamic power management.

The following series of Tips n’ Tricks can be applied to many applications to make the most of PIC MCU nanoWatt and nanoWatt XLP devices.

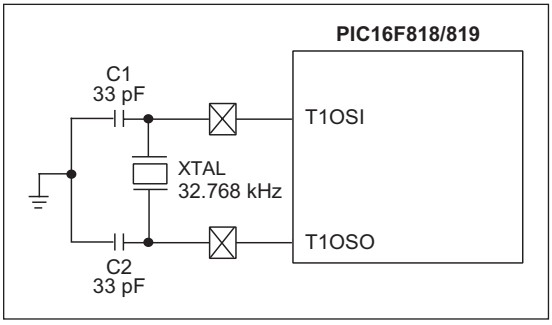
GENERAL LOW POWER TIPS ‘N TRICKS

The following tips can be used with all PIC MCUs to reduce the power consumption of almost any application.

TIP #12 Internal Oscillator Calibration

An internal RC oscillator calibrated from the factory may require further calibration as the temperature or V_{DD} change. Timer1/SOSC can be used to calibrate the internal oscillator by connecting a 32.768 kHz clock crystal. Refer to AN244, “*Internal RC Oscillator Calibration*” for the complete application details. Calibrating the internal oscillator can help save power by allowing for use of the internal RC oscillator in applications which normally require higher accuracy crystals

Figure 12-1: Timer1 Used to Calibrate an Internal Oscillator



The calibration is based on the measured frequency of the internal RC oscillator. For example, if the frequency selected is 4 MHz, we know that the instruction time is 1 µs (F_{OSC}/4) and Timer1 has a period of 30.5 µs (1/32.768 kHz). This means within one Timer1 period, the core can execute 30.5 instructions. If the Timer1 registers are preloaded with a known value, we can calculate the number of instructions that will be executed upon a Timer1 overflow.

This calculated number is then compared against the number of instructions executed by the core. With the result, we can determine if re-calibration is necessary, and if the frequency must be increased or decreased. Tuning uses the OSCTUNE register, which has a ±12% tuning range in 0.8% steps.

TIP #13 Idle and Doze Modes

nanoWatt and nanoWatt XLP devices have an Idle mode where the clock to the CPU is disconnected and only the peripherals are clocked. In PIC16 and PIC18 devices, Idle mode can be entered by setting the Idle bit in the OSCON register to ‘1’ and executing the `SLEEP` instruction. In PIC24, dsPIC® DSCs, and PIC32 devices, Idle mode can be entered by executing the instruction “`PWRSAV #1`”. Idle mode is best used whenever the CPU needs to wait for an event from a peripheral that cannot operate in Sleep mode. Idle mode can reduce power consumption by as much as 96% in many devices.

Doze mode is another low power mode available in PIC24, dsPIC DSCs, and PIC32 devices. In Doze mode, the system clock to the CPU is postscaled so that the CPU runs at a lower speed than the peripherals. If the CPU is not tasked heavily and peripherals need to run at high speed, then Doze mode can be used to scale down the CPU clock to a slower frequency. The CPU clock can be scaled down from 1:1 to 1:128. Doze mode is best used in similar situations to Idle mode, when peripheral operation is critical, but the CPU only requires minimal functionality.

Static Power Reduction Tips n' Tricks

The following tips and tricks will help reduce the power consumption of a device while it is asleep. These tips allow an application to stay asleep longer and to consume less current while sleeping.

TIP #16 Deep Sleep Mode

In Deep Sleep mode, the CPU and all peripherals except RTCC, DSWDT and LCD (on LCD devices) are not powered. Additionally, Deep Sleep powers down the Flash, SRAM, and voltage supervisory circuits. This allows Deep Sleep mode to have lower power consumption than any other operating mode. Typical Deep Sleep current is less than 50 nA on most devices. Four bytes of data are retained in the DSGPRx registers that can be used to save some critical data required for the application. While in Deep Sleep mode, the states of I/O pins and 32 kHz crystal oscillator (Timer1/SOSC) are maintained so that Deep Sleep mode does not interrupt the operation of the application. The RTCC interrupt, Ultra Low Power Wake-up, DSWDT time-out, External Interrupt 0 (INT0), MCLR or POR can wake-up the device from Deep Sleep. Upon wake-up the device resumes operation at the reset vector.

Deep Sleep allows for the lowest possible static power in a device. The trade-off is that the firmware must re-initialize after wake-up. Therefore, Deep Sleep is best used in applications that require long battery life and have long sleep times. Refer to the device datasheets and Family Reference Manuals for more information on Deep Sleep and how it is used.

TIP #17 Extended WDT and Deep Sleep WDT

A commonly used source to wake-up from Sleep or Deep Sleep is the Watchdog Timer (WDT) or Deep Sleep Watchdog Timer (DSWDT). The longer the PIC MCU stays in Sleep or Deep Sleep, the less power consumed. Therefore, it is appropriate to use as long a timeout period for the WDT as the application will allow.

The WDT runs in all modes except for Deep Sleep. In Deep Sleep, the DSWDT is used instead. The DSWDT uses less current and has a longer timeout period than the WDT. The timeout period for the WDT varies by device, but typically can vary from a few milliseconds to up to 2 minutes. The DSWDT time-out period can be programmed from 2.1ms to 25.7days

TIP #18 Low Power Timer1 Oscillator and RTCC

nanoWatt XLP microcontrollers all have a robust Timer1 oscillator (SOSC on PIC24) which draws less than 800 nA. nanoWatt technology devices offer a low power Timer1 oscillator which draws 2-3 uA. Some devices offer a selectable oscillator which can be used in either a low-power or high-drive strength mode to suit both low power or higher noise applications. The Timer1 counter and oscillator can be used to generate interrupts for periodic wakes from Sleep and other power managed modes, and can be used as the basis for a real-time clock. Timer1/SOSC wake-up options vary by device. Many nanoWatt XLP devices have a built-in hardware Real-Time Clock and Calendar (RTCC), which can be configured for wake-up periods from 1 second to many years.

Some nanoWatt devices and all nanoWatt XLP devices can also use the Timer1/SOSC oscillator as the system clock source in place of the main oscillator on the OSC1/OSC2 pins. By reducing execution speed, total current consumption can be reduced.

TIP #7 Periodic Interrupts

Generating interrupts at periodic intervals is a useful technique implemented in many applications. This technique allows the main loop code to run continuously, and then, at periodic intervals, jump to the interrupt service routine to execute specific tasks (i.e., read the ADC). Normally, a timer overflow interrupt is adequate for generating the periodic interrupt. However, sometimes it is necessary to interrupt at intervals that can not be achieved with a timer overflow interrupt. The CCP configured in Compare mode makes this possible by shortening the full 16-bit time period.

Example Problem:

A PIC16F684 running on its 8 MHz internal oscillator needs to be configured so that it updates a LCD exactly 5 times every second.

Step #1: Determine a Timer1 prescaler that allows an overflow at greater than 0.2 seconds

a) Timer1 overflows at: Tosc*4*65536* prescaler

b) For a prescaler of 1:1, Timer1 overflows in 32.8 ms.

c) A prescaler of 8 will cause an overflow at a time greater than 0.2 seconds.
8 x 32.8 ms = 0.25s

Step #2: Calculate CCPR1 (CCPR1L and CCPR1H) to shorten the time-out to exactly 0.2 seconds

a) CCPR1 = Interval Time/(Tosc*4*prescaler) = 0.2/(125 ns*4*8) = 5000 = 0xC350

b) Therefore, CCPR1L = 0x50, and CCPR1H = 0xC3

Step #3: Configuring CCP1CON

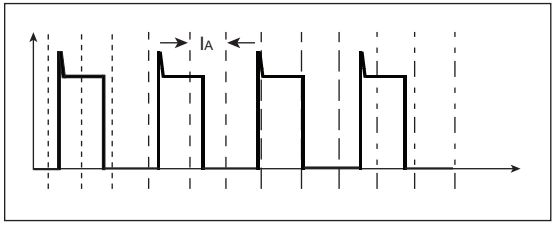
The CCP module should be configured in Trigger Special Event mode. This mode generates an interrupt when the Timer1 equals the value specified in CCPR1L and Timer1 is automatically cleared(1). For this mode, CCP1CON = 'b00001011'.

Note 1: Trigger Special Event mode also starts an A/D conversion if the A/D module is enabled. If this functionality is not desired, the CCP module should be configured in "generate software interrupt-on-match only" mode (i.e., CCP1CON = b'00001010'). Timer 1 must also be cleared manually during the CCP interrupt.

TIP #12 Repetitive Phase Shifted Sampling

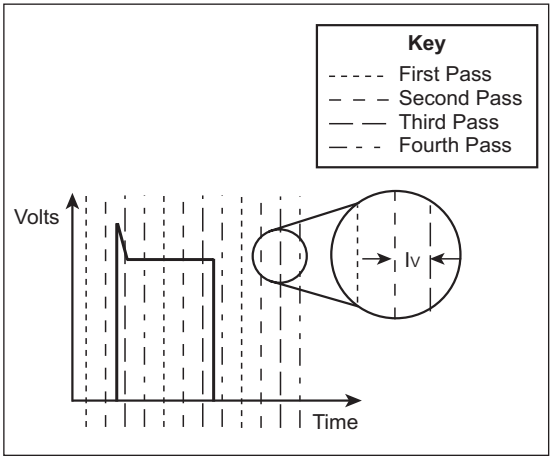
Repetitive phase shifted sampling is a technique to artificially increase the sampling rate of an A/D converter when sampling waveforms that are both periodic and constant from period to period. The technique works by capturing regularly spaced samples of the waveform from the start to finish of the waveform's period. Sampling of the next waveform is then performed in the same manner, except that the start of the sample sequence is delayed a percentage of the sampling period. Subsequent waveforms are also sampled, with each sample sequence slightly delayed from the last, until the delayed start of the sample sequence is equal to one sample period. Interleaving the sample sets then produces a sample set of the waveform at a higher sample rate. Figure 12-1 shows an example of a high frequency waveform.

Figure 12-1: High Frequency Periodic Waveform



As indicated in the key, the finely dotted lines show where the A/D readings are taken during the first period of the waveform. The medium sized dashed lines show when the A/D readings are taken during the second period, and so on. Figure 12-2 shows these readings transposed onto one period.

Figure 12-2: Transposed Waveform



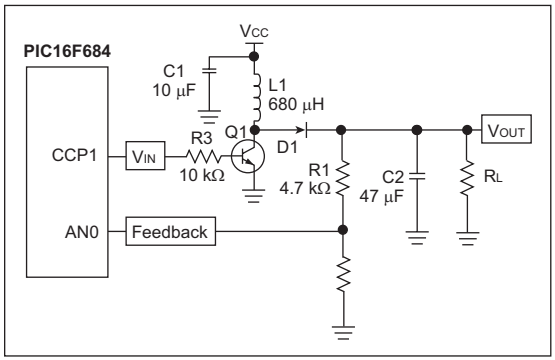
The CCP module is configured in Compare Special Event Trigger mode to accomplish this task. The phase shift is implemented by picking values of CCPRxL and CCPRxH that are not synchronous with the period of the sampling waveform. For instance, if the period of a waveform is 100 μs, then sampling at a rate of once every 22 μs will give the following set of sample times over 11 periods (all values in μs).

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
0	10	20	8	18	6	16	4	14	2	12
22	32	42	30	40	28	38	26	36	24	34
44	54	64	52	62	50	60	48	58	46	56
66	76	86	74	84	72	82	70	80	68	78
88	98		96		94		92		90	

When these numbers are placed in sequential order, they reveal a virtual sampling interval (I_V) of 2 μs from 0 μs to 100 μs, although the actual sampling interval (I_A) is 22 μs.

TIP #17 Boost Power Supply

Figure 17-1: Boost Power Supply Circuit



Hardware

Pulse-width modulation plays a key role in boost power supply design. Figure 17-1 shows a typical boost circuit. The circuit works by Q1 grounding the inductor (L1) during the high phase of the PWM signal generated by CCP1. This causes an increasing current to flow through L1 while V_{CC} is applied. During the low phase of the PWM signal, the energy stored in L1 flows through D1 to the storage capacitor (C2) and the load. V_{OUT} is related to V_{IN} by Equation 17-1.

Note: Technical Brief TB053 "*Generating High Voltage Using the PIC16C781/782*" provides details on boost power supply design.

The first parameter to determine is the duty cycle based upon the input and output voltages. See Equation 17-1.

Equation 17-1

$$\frac{V_{OUT}}{V_{IN}} = \frac{1}{1 - D}$$

Next, the value of the inductor is chosen based on the maximum current required by the load, the switching frequency and the duty cycle. A function for inductance in terms of load current is given by Equation 17-2, where T is the PWM period, D is the duty cycle, and I_{OUT} is the maximum load current.

Equation 17-2

$$L = \frac{V_{IN}\,(1 - D)\,DT}{2\,I_{OUT}}$$

The value for L is chosen arbitrarily to satisfy this equation given I_{OUT}, a maximum duty cycle of 75% and a PWM frequency in the 10 kHz to 100 kHz range.

Using the value chosen for L, the ripple current is calculated using Equation 17-3.

Equation 17-3

$$I_{RIPPLE} = \frac{V_{IN}\,DT}{L}$$

I_{RIPPLE} can not exceed the saturation current for the inductor. If the value for L does produce a ripple current greater than I_{SAT}, a bigger inductor is needed.

Note: All equations above assume a discontinuous current mode.

Firmware

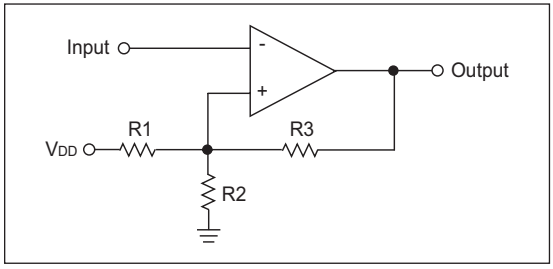
The PWM duty cycle is varied by the microcontroller in order to maintain a stable output voltage over fluctuating load conditions. A firmware implemented PID control loop is used to regulate the duty cycle. Feedback from the boost power supply circuit provides the input to the PID control.

Note: Application Note AN258 "*Low Cost USB Microcontroller Programmer*" provides details on firmware-based PID control.

TIP #3 Hysteresis

When the voltages on a comparator's input are nearly equal, external noise and switching noise from inside the microcontroller can cause the comparator output to oscillate or "chatter." To prevent chatter, some of the comparator output voltage is fed back to the non-inverting input of the comparator to form hysteresis (see Figure 3-1). Hysteresis moves the comparator threshold up when the input is below the threshold, and down when the input is above the threshold. The result is that the input must overshoot the threshold to cause a change in the comparator output. If the overshoot is greater than the noise present on the input, the comparator output will not chatter.

Figure 3-1: Comparator with Hysteresis



To calculate the resistor values required, first determine the high and low threshold values which will prevent chatter (V_{TH} and V_{TL}). Using V_{TH} and V_{TL}, the average threshold voltage can be calculated using the equation.

Equation 3-1

$$V_{AVG} = \frac{V_{DD} * V_{TL}}{V_{DD} - V_{TH} + V_{TL}}$$

Next, choose resistor values that satisfy Equation 3-2 and calculate the equivalent resistance using Equation 3-3.

Note: A continuous current will flow through R1 and R2. To limit the power dissipation in R1 and R2 the total resistance of R1 and R2 should be at least 1k. The total resistance of R1 and R2 should also be kept below 10K to keep the size of R3 small. Large values for R3, 100k-10 M , can produce voltage offsets at the non-inverting input due to the comparator's input bias current.

Equation 3-2

$$V_{AVG} = \frac{V_{DD} * R2}{R1 + R2}$$

Equation 3-3

$$R_{EQ} = \frac{R1 * R2}{R1 + R2}$$

Then, determine the feedback divider ratio D_R, using Equation 3-4.

Equation 3-4

$$D_R = \frac{(V_{TH} - V_{TL})}{V_{DD}}$$

Finally, calculate the feedback resistor R3 using Equation 3-5.

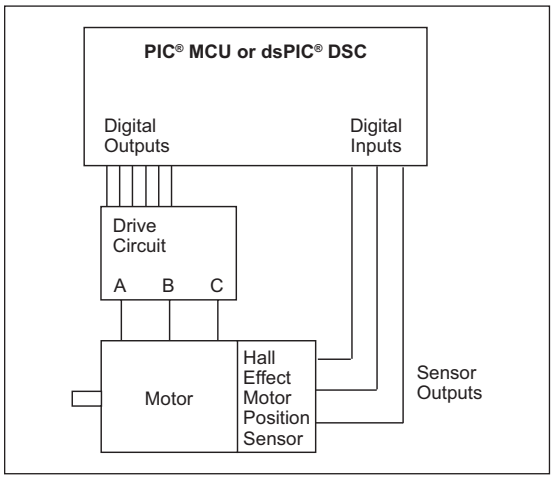
Equation 3-5

$$R3 = R_{EQ} \left[\left(\frac{1}{D_R} \right) - 1 \right]$$

Example:

- A V_{DD} = 5.0V, V_H = 3.0V and V_L = 2.5V
- V_{AVG} = 2.77V
- R = 8.2k and R2 = 10k, gives a V_{AVG} = 2.75V
- R_{EQ} = 4.5k
- D_R = .1
- R3 = 39k (40.5 calculated)
- V_{HACT} = 2.98V
- V_{LACT} = 2.46V

Figure 2-3: Quadrature Decoder (Sensor Motor)



Application notes describing Brushless DC Motor Control are listed below and can be found on the Microchip web site at: www.microchip.com.

- AN857, “*Brushless DC Motor Control Made Easy*” (DS00857)
- AN885, “*Brushless DC Motor Fundamentals*” (DS00885)
- AN899, “*Brushless DC Motor Control Using PIC18FXX31*” (DS00899)
- AN901, “*Using the dsPIC30F for Sensorless BLDC Control*” (DS00901)
- AN957, “*Sensored BLDC Motor Control Using dsPIC30F2010*” (DS00957)
- AN992, “*Sensorless BLDC Motor Control Using dsPIC30F2010*” (DS00992)
- AN1017, “*Sinusoidal Control of PMSM with dsPIC30F DSC*” (DS01017)
- GS005, “*Using the dsPIC30F Sensorless Motor Tuning Interface*” (DS93005)

TIP #3 Stepper Motor Drive Circuits

Stepper motors are similar to Brushless DC motors in that the control system must commutate the motor through the entire rotation cycle. Unlike the brushless motor, the position and speed of a stepping motor is predictable and does not require the use of sensors.

There are two basic types of stepper motors, although some motors are built to be used in either mode. The simplest stepper motor is the unipolar motor. This motor has four drive connections and one or two center tap wires that are tied to ground or Vsupply, depending on the implementation. Other motor types are the bipolar stepper and various combinations of unipolar and bipolar, as shown in Figure 3-1 and Figure 3-2. When each drive connection is energized, one coil is driven and the motor rotates one step. The process is repeated until all the windings have been energized. To increase the step rate, often the voltage is increased beyond the motors rated voltage. If the voltage is increased, some method of preventing an over current situation is required.

There are many ways to control the winding current, but the most popular is a chopper system that turns off current when it reaches an upper limit and enables the current flow a short time later. Current sensor systems are discussed in Tip #6. Some systems are built with a current chopper, but they do not detect the current, rather the system is designed to begin a fixed period chopping cycle after the motor has stepped to the next position. These are simpler systems to build, as they only require a change in the software.

Figure 3-1: 4 and 5 Wire Stepper Motors

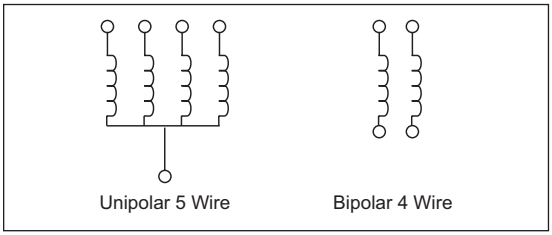


Figure 3-2: 6 and 8 Wire Stepper Motors

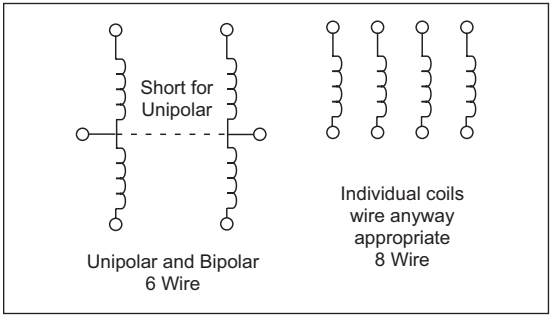


Figure 3-3: Unipolar Motor (4 Low Side Switches)

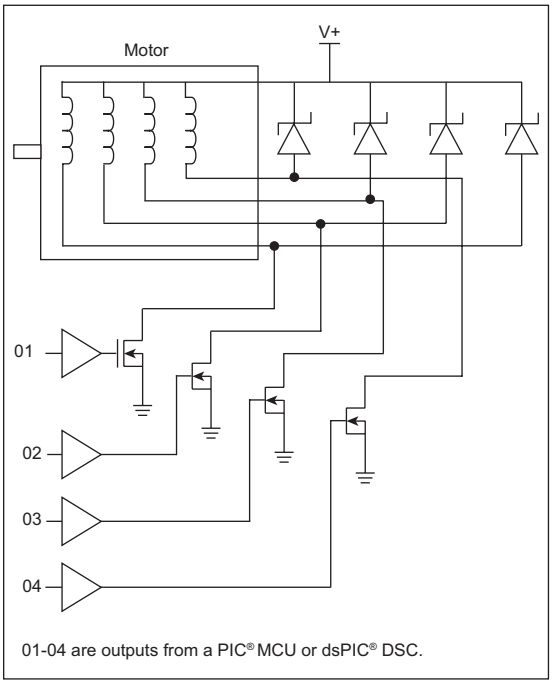
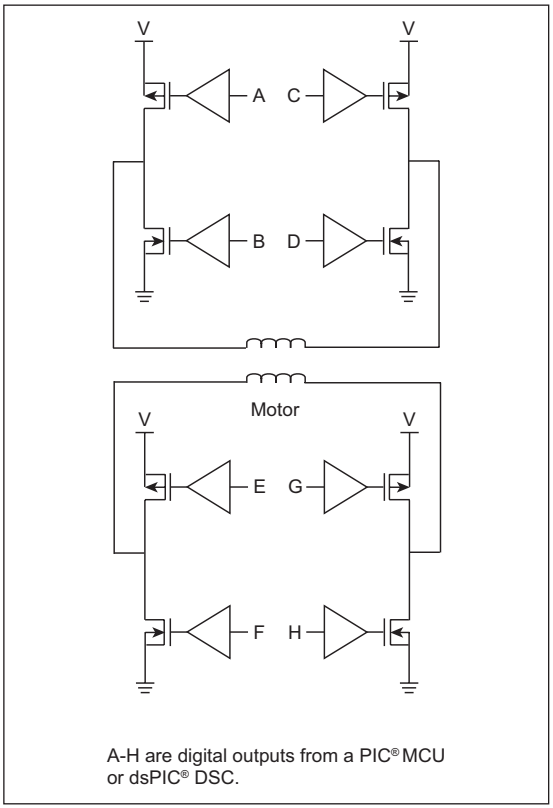


Figure 3-4: Bipolar Motor (4 Half-Bridges)



NOTES:

CHAPTER 6
LCD PIC® Microcontroller Tips ‘n Tricks

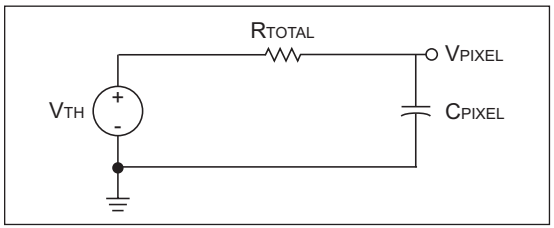
Table Of Contents

TIPS ‘N TRICKS INTRODUCTION

Using an LCD PIC® MCU for any embedded application can provide the benefits of system control and human interface via an LCD. Design practices for LCD applications can be further enhanced through the implementation of these suggested “Tips ‘n Tricks”.

This booklet describes many basic circuits and software building blocks commonly used for driving LCD displays. The booklet also provides references to Microchip application notes that describe many LCD concepts in more detail.

Figure 3-4: Voltage Change Across Pixel



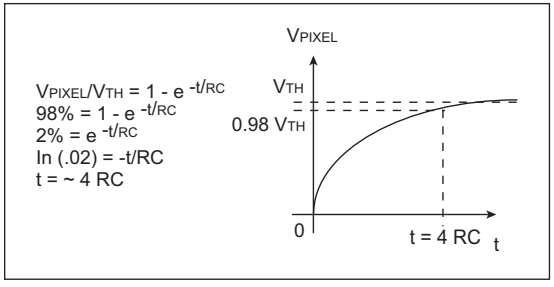
The step response of the voltage across a pixel is subject to the following equation:

Equation 3-1

$$V_{PIXEL} = V_{TH} (1 - e^{-t/RC})$$

By manipulating the equation, we can see that it will take a time equal to 4 time constants for the pixel voltage to reach 98% of the bias voltage.

Figure 3-5: Step Response Diagram



Now we need to estimate the capacitance. Capacitance is proportional to the area of a pixel. We can measure the area of a pixel and estimate the capacitance as shown. Obviously, a bigger display, such as a digital wall clock, will have bigger pixels and higher capacitance.

Equation 3-2

$$C_{PIXEL} = 1500 \text{ pF/cm}^2$$
$$AREA_{PIXEL} = 1 \text{ mm} * 3 \text{ mm} = .03 \text{ cm}^2$$
$$C_{PIXEL} = 45 \text{ pF}$$

We want the time constant to be much smaller than the period of the LCD waveform, so that rounding of the LCD waveform will be minimized. If we want the RC to be equal to 100 μS, then the total resistance can be calculated as shown:

Equation 3-3

$$R_{TOTAL} = 100 \text{ μS}/45 \text{ pF} = 2.22 \text{ mΩ}$$
$$R_{TH} = 2.2M - 5.1K = 2.2M$$

The resistance of the switching circuits within the LCD module is very small compared to this resistance, so the Thevenin resistance of the resistor ladder at $V_{LCD}2$ and $V_{LCD}1$ can be treated the same as R_{TOTAL}. We can then calculate the value for R that will give us the correct Thevenin resistance.

Equation 3-4

$$R = 3 R_{TH}/2 = 3.3M$$

Now we can calculate the current through the resistor ladder if we used 3.3 mΩ resistors.

Equation 3-5

$$R_{LADDER} = 9.9M,$$
$$I_{LADDER} = 5V/9.9M = 0.5 \text{ μA}$$

Use this process to estimate maximum resistor sizes for your resistor ladder and you will drastically reduce power consumption for your LCD application. Don't forget to observe the display over the operating conditions of your product (such as temperature, voltage and even, humidity) to ensure that contrast and display quality is good.

TIP #8 In-Circuit Debug (ICD)

There are two potential issues with using the ICD to debug LCD applications. First, the LCD controller can freeze while the device is Halted. Second, the ICD pins are shared with segments on the PIC16F946/917/916/914/913 MCUs.

When debugging, the device is Halted at breakpoints and by the user pressing the pause button. If the ICD is configured to Halt the peripherals with the device, the LCD controller will Halt and apply DC voltages to the LCD glass. Over time, these DC levels can cause damage to the glass; however, for most debugging situations, this will not be a consideration. The PIC18F LCD MCUs have a feature that allows the LCD module to continue operating while the device has been Halted during debugging. This is useful for checking the image of the display while the device is Halted and for preventing glass damage if the device will be Halted for a long period of time.

The PIC16F946/917/916/914/913 multiplex the ICSP™ and ICD pins onto pins shared with LCD segments 6 and 7. If an LCD is attached to these pins, the device can be debugged with ICD; however, all the segments driven by those two pins will flicker and be uncontrolled. As soon as debugging is finished and the device is programmed with Debug mode disabled, these segments will be controlled correctly.

TIP #9 LCD in Sleep Mode

If you have a power-sensitive application that must display data continuously, the LCD PIC microcontroller can be put to Sleep while the LCD driver module continues to drive the display.

To operate the LCD in Sleep, only two steps are required. First, a time source other than the main oscillator must be selected as the LCD clock source, because during Sleep, the main oscillator is Halted. Options are shown for the various LCD PIC MCUs.

Table 9-1: Options for LCD in Sleep Mode

Part	LCD Clock Source	Use in Sleep?
PIC16C925/926	Fosc/256	No
	T1OSC	Yes
	Internal RC Oscillator	Yes
PIC16F946/917/916/914/913	Fosc/8192	No
	T1OSC/32	Yes
	LFINTOSC/32	Yes
PIC18F6X90	(Fosc/4)/8192	No
PIC18F8X90	T1OSC	Yes
PIC18F6XJ90	INTRC/32	Yes
PIC18F8XJ90		

Second, the Sleep Enable bit (SLPEN) must be cleared. The LCD will then continue to display data while the part is in Sleep. It's that easy!

When should you select the internal RC oscillator (or LFINTOSC) over the Timer1 oscillator? It depends on whether your application is time-sensitive enough to require the accuracy of a crystal on the Timer1 oscillator or not. If you have a timekeeping application, then you will probably have a 32 kHz crystal oscillator connected to Timer1.

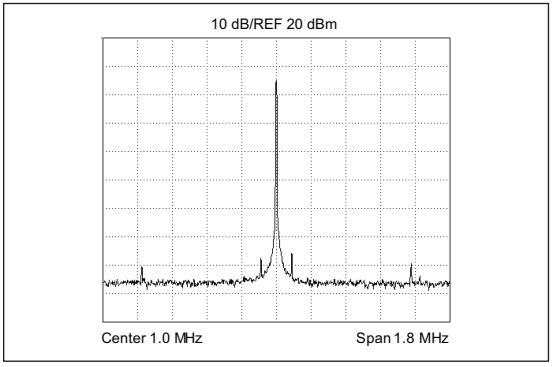
Since Timer1 continues to operate during Sleep, there is no penalty in using Timer1 as the LCD clock source. If you don't need to use an external oscillator on Timer1, then the internal RC oscillator (INTRC or LFINTOSC) is more than sufficient to use as the clock source for the LCD and it requires no external components.

TIP #4 Creating a Dithered PWM Clock

In order to meet emissions requirements as mandated by the FCC and other regulatory organizations, the switching frequency of a power supply can be varied. Switching at a fixed frequency produces energy at that frequency. By varying the switching frequency, the energy is spread out over a wider range and the resulting magnitude of the emitted energy at each individual frequency is lower.

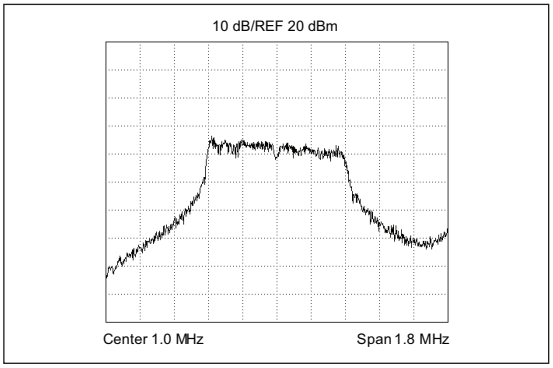
The PIC10F200 has an internal 4 MHz oscillator. A scaled version of oscillator can be output on a pin (Fosc/4). The scaled output is 1/4 of the oscillator frequency (1 MHz) and will always have a 50% duty cycle. Figure 4-1 shows a spectrum analyzer shot of the output of the Fosc/4 output.

Figure 4-1: Spectrum of Clock Output Before Dithering



The PIC10F200 provides an Oscillator Calibration (OSCCAL) register that is used to calibrate the frequency of the oscillator. By varying the value of the OSCCAL setting, the frequency of the clock output can be varied. A pseudo-random sequence was used to vary the OSCCAL setting, allowing frequencies from approximately 600 kHz to 1.2 MHz. The resulting spectrum is shown in Figure 4-2.

Figure 4-2: Spectrum of Clock Output After Dithering



By spreading the energy over a wider range of frequencies, a drop of more than 20 dB is achieved.

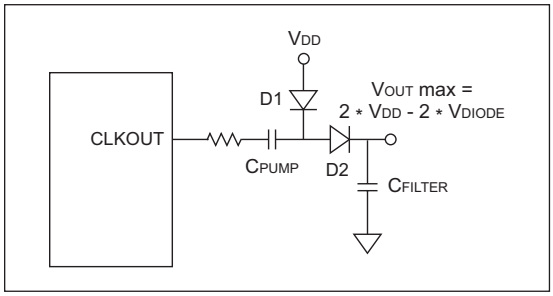
Example software is provided for the PIC10F200 that performs the pseudo-random sequence generation and loads the OSCCAL register.

TIP #10 Driving High Side FETs

In applications where high side N channel FETs are to be driven, there are several means for generating an elevated driving voltage. One very simple method is to use a voltage doubling charge pump as shown in Figure 10-1.

Method 1

Figure 10-1: Typical Change Pump

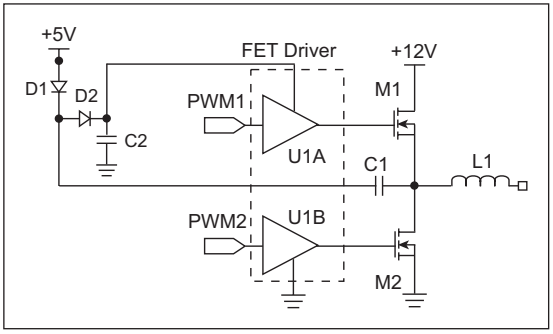


The PIC MCUs CLKOUT pin toggles at 1/4 of the oscillator frequency. When CLKOUT is low, D1 is forward biased and conducts current, thereby charging CPUMP. After CLKOUT is high, D2 is forward biased, moving the charge to CFILTER. The result is a voltage equal to twice the VDD minus two diode drops. This can be used with a PWM or any other I/O pin that toggles.

In Figure 10-2, a standard FET driver is used to drive both the high and low side FETs by using the diode and capacitor arrangement.

Method 2

Figure 10-2: Schematic



The +5V is used for powering the microcontroller. Using this arrangement, the FET driver would have approximately 12 + (5 - VDIODE) - VDIODE volts as a supply and is able to drive both the high and low side FETs.

The circuit above works by charging C1 through D1 to (5V - VDIODE) while M2 is on, effectively connecting C1 to ground. When M2 turns off and M1 turns on, one side of C1 is now at 12V and the other side is at 12V + (5V - VDIODE). The D2 turns on and the voltage supplied to the FET driver is 12V + (5V - VDIODE) - VDIODE.

TIP #19 Execution-Indexed Software State Machine

Another common type of state machine is the execution-indexed state machine. This type of state machine uses a state variable in order to determine what is executed. In C, this can be thought of as the switch statement structure as shown in Example 19-1.

Example 19-1: Example Using Switch Statement

```
SWITCH (State)
{
  CASE 0: IF (in_key()==5)  THEN state = 1;
          Break;
  CASE 1: IF (in_key()==8)  THEN State = 2;
                            Else State = 0;
          Break;
  CASE 2: IF (in_key()==3)  THEN State = 3;
                            Else State = 0;
          Break;
  CASE 3: IF (in_key()==2)  THEN UNLOCK();
                            Else State = 0;
          Break;
}
```

Each time the software runs through the loop, the action taken by the state machine changes with the value in the state variable. By allowing the state machine to control its own state variable, it adds memory, or history, because the current state will be based on previous states. The microcontroller is able to make current decisions based on previous inputs and data.

In assembly, an execution-indexed state machine can be implemented using a jump table.

Example 19-2: Example Using a Jump Table

```
  MOVFW   state      ;load state into w
  ADDWF   PCL,f      ;jump to state
                     ;number
  GOTO    state0     ;state 0
  GOTO    state1     ;state 1
  GOTO    state2     ;state 2
  GOTO    state3     ;state 3
  GOTO    state4     ;state 4
  GOTO    state5     ;state 5
```

In Example 19-2, the program will jump to a GOTO statement based on the state variable. The GOTO statement will send the program to the proper branch. Caution must be taken to ensure that the variable will never be larger than intended. For example, six states (000 to 101) require a three-bit state variable. Should the state variable be set to an undefined state (110 to 111), program behavior would become unpredictable.

Means for safeguarding this problem include:

- Mask off any unused bits of the variable. In the above example, ANDLW b’00000111’ will ensure that only the lower 3 bits of the number contain a value.
- Add extra cases to ensure that there will always be a known jump. For example in this case, two extra states must be added and used as error or Reset states.