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-SSOP (0.209", 5.30mm Width)
Supplier Device Package	28-SSOP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic16c63a-20i-ss

TIP #13 Reading a Sensor With Higher Accuracy

Sensors can be read directly with the A/D but in some applications, factors such as temperature, external component accuracy, sensor non-linearity and/or decreasing battery voltage need to be considered. In other applications, more than 10 bits of accuracy are needed and a slower sensor read is acceptable. The following tips deal with these factors and show how to get the most out of a PIC MCU.

13.1. RC Timing Method (with reference resistor)

13.2. Charge Balancing Method

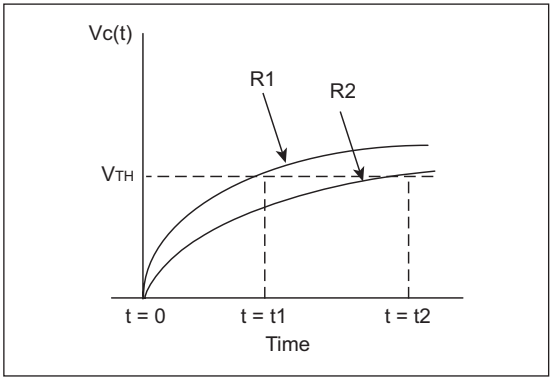
13.3. A/D Method

Tip #13.1 Reading a Sensor With Higher Accuracy – RC Timing Method

RC Timing Method:

Simple RC step response
$Vc(t) = V_{DD} * (1 - e^{-t/(RC)})$
$t = -RC \ln(1 - V_{TH}/V_{DD})$
V_{TH}/V_{DD} is constant
R2 = (t2/t1) * R1

Figure 13-1



A reference resistor can be used to improve the accuracy of an analog sensor reading. In this diagram, the charge time of a resistor/capacitor combination is measured using a timer and a port input or comparator input switches from a ‘0’ to ‘1’. The R1 curve uses a reference resistor and the R2 curve uses the sensor. The charge time of the R1 curve is known and can be used to calibrate the unknown sensor reading, R2. This reduces the affects of temperature, component tolerance and noise while reading the sensor.

Tip #13.3 Reading a Sensor With Higher Accuracy – A/D Method

NTC (Negative Temperature Coefficient) sensors have a non-linear response to temperature changes. As the temperature drops, the amount the resistance changes becomes less and less. Such sensors have a limited useful range because the resolution becomes smaller than the A/D resolution as the temperature drops. By changing the voltage divider of the RSEN, the temperature range can be expanded.

To select the higher temperature range, GP1 outputs '1' and GP2 is set as an input. For the lower range, GP2 outputs '1' and GP1 is configured as an input. The lower range will increase the amount the sensor voltage changes as the temperature drops to allow a larger usable sensor range.

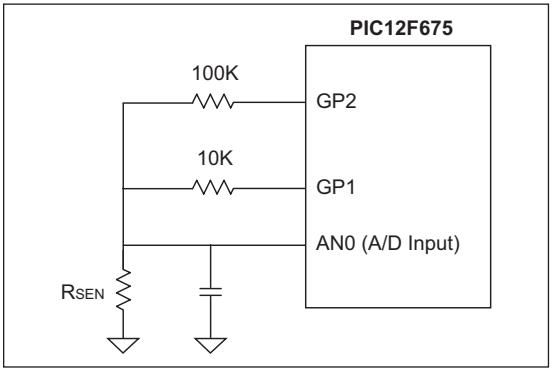
Summary:

High range: GP1 output '1' and GP2 input

Low range: GP1 input and GP2 output '1'

1. 10K and 100K resistors are used to set the range
2. VREF for A/D = VDD
3. Rth calculation is independent of VDD
4. Count = RSEN/(RSEN+RREF) x 255
5. Don't forget to allow acquisition time for the A/D

Figure 13-4

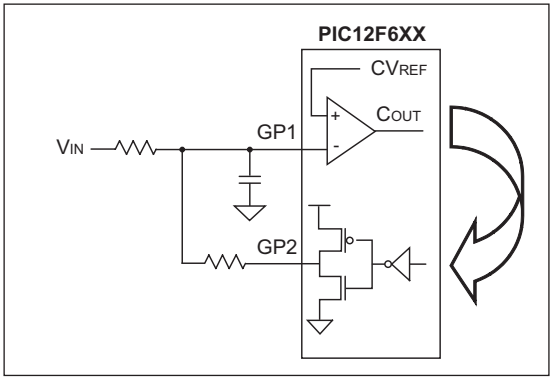


TIP #14 Delta-Sigma Converter

The charge on the capacitor on GP1 is maintained about equal to the CVREF by the MCU monitoring COUT and switching GP2 from Input mode or output low appropriately. A timer is used to sample the COUT bit on a periodic basis. Each time GP2 is driven low, a counter is incremented. This counter value corresponds to the input voltage.

To minimize the affects of external component tolerances, temperature, etc., the circuit can be calibrated. Apply a known voltage to the input and allow the microcontroller to count samples until the expected result is calculated. By taking the same number of samples for subsequent measurements, they become calibrated measurements.

Figure 14-1



1. GP1 average voltage = CVREF
2. Time base as sampling rate
3. At the end of each time base period:
   - If GP1 > CVREF, then GP2 Output Low
   - If GP1 < CVREF, then GP2 Output High
4. Accumulate the GP2 lows over many samples
5. Number of samples determines resolution

TIPS ‘N TRICKS WITH SOFTWARE

To reduce costs, designers need to make the most of the available program memory in MCUs. Program memory is typically a large portion of the MCU cost. Optimizing the code helps to avoid buying more memory than needed. Here are some ideas that can help reduce code size.

TIP #15 Delay Techniques

- Use GOTO “next instruction” instead of two NOPs.
- Use CALL Rtrn as quad, 1 instruction NOP (where “Rtrn” is the exit label from existing subroutine).

Example 15-1

```
        NOP
        NOP             ;2 instructions, 2 cycles

        GOTO    $+1     ;1 instruction, 2 cycles

        CALL Rtrn       ;1 instruction, 4 cycles
        . . .
Rtrn    RETURN
```

MCUs are commonly used to interface with the “outside world” by means of a data bus, LEDs, buttons, latches, etc. Because the MCU runs at a fixed frequency, it will often need delay routines to meet setup/hold times of other devices, pause for a handshake or decrease the data rate for a shared bus.

Longer delays are well-suited for the DECFSZ and INCFSZ instructions where a variable is decremented or incremented until it reaches zero when a conditional jump is executed. For shorter delays of a few cycles, here a few ideas to decrease code size.

For a two-cycle delay, it is common to use two NOP instructions which uses two program memory locations. The same result can be achieved by using “goto $+1”. The “$” represents the current program counter value in MPASM™ Assembler. When this instruction is encountered, the MCU will jump to the next memory location. This is what it would have done if two NOP’s were used but since the GOTO instruction uses two instruction cycles to execute, a two-cycle delay was created. This created a two-cycle delay using only one location of program memory.

To create a four-cycle delay, add a label to an existing RETURN instruction in the code. In this example, the label “Rtrn” was added to the RETURN of subroutine that already existed somewhere in the code. When executing “CALL Rtrn”, the MCU delays two instruction cycles to execute the CALL and two more to execute the RETURN. Instead of using four NOP instructions to create a four-cycle delay, the same result was achieved by adding a single CALL instruction.

TIP #16 Optimizing Destinations

- Destination bit determines W for F for result
- Look at data movement and restructure

Example 16-1

Example: A + B → A

```
MOVF    A,W
ADDWF   B,W
MOVWF   A
```

3 instructions

```
MOVF    B,W
ADDWF   A,F
```

2 instructions

Careful use of the destination bits in instructions can save program memory. Here, register A and register B are summed and the result is put into the A register. A destination option is available for logic and arithmetic operations. In the first example, the result of the ADDWF instruction is placed in the working register. A MOVWF instruction is used to move the result from the working register to register A. In the second example, the ADDWF instruction uses the destination bit to place the result into the A register, saving an instruction.

TIP #17 Conditional Bit Set/Clear

- To move single bit of data from REGA to REGB
- Precondition REGB bit
- Test REGA bit and fix REGB if necessary

Example 17-1

```
BTFSS   REGA,2
BCF     REGB,5
BTFSC   REGA,2
BSF     REGB,5
```

4 instructions

```
BCF     REGB,5
BTFSC   REGA,2
BSF     REGB,5
```

3 instructions

One technique for moving one bit from the REGA register to REGB is to perform bit tests. In the first example, the bit in REGA is tested using a BTFSS instruction. If the bit is clear, the BCF instruction is executed and clears the REGB bit, and if the bit is set, the instruction is skipped.The second bit test determines if the bit is set, and if so, will execute the BSF and set the REGB bit, otherwise the instruction is skipped. This sequence requires four instructions.

A more efficient technique is to assume the bit in REGA is clear, and clear the REGB bit, and test if the REGA bit is clear. If so, the assumption was correct and the BSF instruction is skipped, otherwise the REGB bit is set. The sequence in the second example uses three instructions because one bit test was not needed.

One important point is that the second example will create a two-cycle glitch if REGB is a port outputting a high. This is caused by the BCF and BTFSC instructions that will be executed regardless of the bit value in REGA.

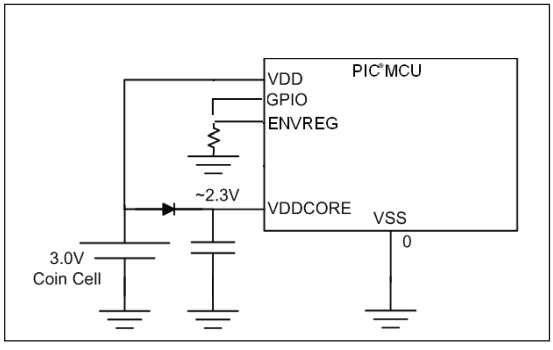
TIP #6 Use an External Source for CPU Core Voltage

Some PIC MCUs such as “J” type devices (ex. PIC18F87J90 or PIC24FJ64GA004) use separate power for CPU core. These devices have an internal voltage regulator that can be used to provide the core voltage. Alternatively, the core voltage can be provided externally by disabling the internal regulator. In some cases, it is more power efficient to use an external source for the core. This is because the internal regulator powers the core at the nominal voltage that allows full speed operation. However, if an application doesn’t require full speed, it is beneficial to use lower voltage to power the core. Disabling the internal regulator also turns off the BOR and LVD circuits, which saves power as well. The following examples show two different battery powered applications where it can be beneficial to disable the internal regulator.

Example 1: Constant Voltage Source

When using a regulated power source or a battery with a flat discharge curve, such as a lithium coin cell, the regulator can be disabled and the core powered directly from the battery through a diode. The diode provides the voltage drop necessary to power the core at the correct voltage. It may be necessary to use a zener diode with a higher forward voltage for applications using sleep mode, as the current consumed in sleep is too low to cause the full forward voltage drop which can result in applying a voltage too high for the core.

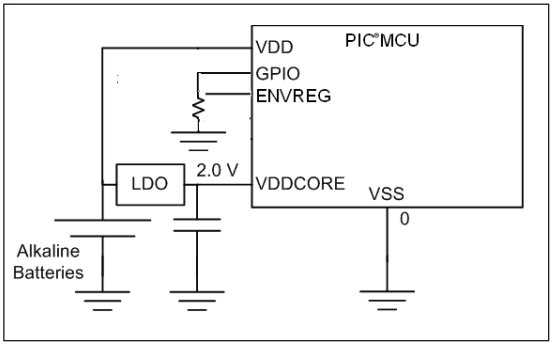
Figure 6-1:



Example 2: Non-Constant Voltage Source

If the source for V_{DD} is not constant, a regulator will be required. It can be beneficial to use an external low quiescent current regulator, which can be selected to provide lower voltage to the core than the internal regulator. Additionally, devices such as the MCP1700, which consumes 1 uA quiescent current while asleep, require less power than the internal regulator.

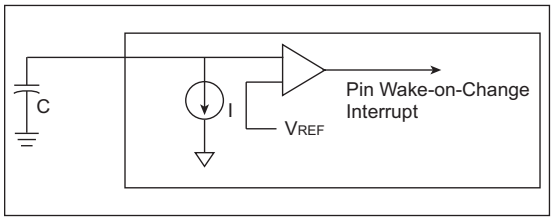
Figure 6-2:



TIP #22 Ultra Low-Power Wake-Up Peripheral

Newer devices have a modification to PORTA that creates an Ultra Low-Power Wake-Up (ULPWU) peripheral. A small current sink and a comparator have been added that allows an external capacitor to be used as a wake-up timer. This feature provides a low-power periodic wake-up source which is dependent on the discharge time of the external RC circuit.

Figure 22-1: Ultra Low-Power Wake-Up Peripheral



If the accuracy of the Watchdog Timer is not required, this peripheral can save a lot of current.

Visit the low power design center at: www.microchip.com/lowpower for additional design resources.

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

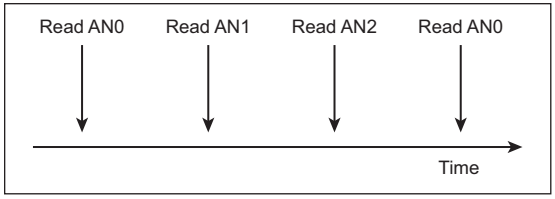
Step #3: Configuring CCP1CON

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 #11 Sequential ADC Reader

Figure 11-1: Timeline



Trigger Special Event mode (a sub-mode in Compare mode) generates a periodic interrupt in addition to automatically starting an A/D conversion when Timer1 matches CCPRxL and CCPRxH. The following example problem demonstrates how to sequentially read the A/D channels at a periodic interval.

Example

Given the PIC16F684 running on its 8 MHz internal oscillator, configure the microcontroller to sequentially read analog pins AN0, AN1 and AN2 at 30 ms intervals.

Step #1: Determine Timer1 Prescaler

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

b) For a prescaler of 1:1, the Timer1 overflow occurs in 32.8 ms.

c) This is greater than 30 ms, so a prescaler of 1 is adequate.

Step #2: Calculate CCPR1 (CCPR1L and CCPR1H)

a) CCPR1 = Interval Time/(Tosc*4*prescaler) = 0.030/(125 ns*4*1) = 6000 = 0xEA60

b) Therefore, CCPR1L = 0x60, and CCPR1H = 0xEA

Step #3: Configuring CCP1CON

The ECCP module should be configured in Trigger Special Event mode. This mode generates an interrupt when Timer1 equals the value specified in CCPR1. Timer1 is automatically cleared and the GO bit in ADCON0 is automatically set. For this mode, CCP1CON = '`b00001011`'.

Step #4: Add Interrupt Service Routine Logic

When the ECCP interrupt is generated, select the next A/D pin for reading by altering the ADCON0 register.

TIP #13 Deciding on PWM Frequency

In general, PWM frequency is application dependent although two general rules-of-thumb hold regarding frequency in all applications. They are:

1. As frequency increases, so does current requirement due to switching losses.
2. Capacitance and inductance of the load tend to limit the frequency response of a circuit.

In low-power applications, it is a good idea to use the minimum frequency possible to accomplish a task in order to limit switching losses. In circuits where capacitance and/or inductance are a factor, the PWM frequency should be chosen based on an analysis of the circuit.

Motor Control

PWM is used extensively in motor control due to the efficiency of switched drive systems as opposed to linear drives. An important consideration when choosing PWM frequency for a motor control application is the responsiveness of the motor to changes in PWM duty cycle. A motor will have a faster response to changes in duty cycle at higher frequencies. Another important consideration is the sound generated by the motor. Brushed DC motors will make an annoying whine when driven at frequencies within the audible frequency range (20 Hz-4 kHz.) In order to eliminate this whine, drive brushed DC motors at frequencies greater than 4 kHz. (Humans can hear frequencies at upwards of 20 kHz, however, the mechanics of the motor winding will typically attenuate motor whine above 4 kHz).

LED and Light Bulbs

PWM is also used in LED and light dimmer applications. Flicker may be noticeable with rates below 50 Hz. Therefore, it is generally a good rule to pulse-width modulate LEDs and light bulbs at 100 Hz or higher.

TIP #14 Unidirectional Brushed DC Motor Control Using CCP

Figure 14-1: Brushed DC (BDC) Motor Control Circuit

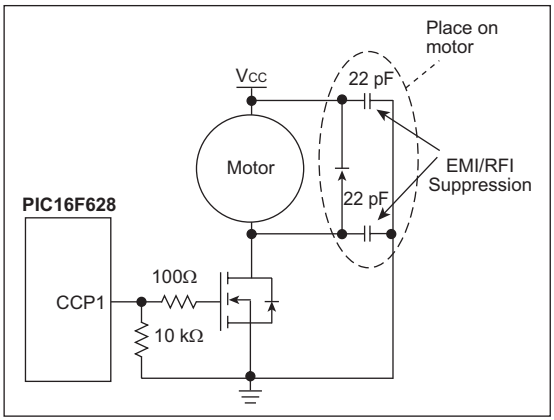


Figure 14-1 shows a unidirectional speed controller circuit for a brushed DC motor. Motor speed is proportional to the duty cycle of the PWM output on the CCP1 pin. The following steps show how to configure the PIC16F628 to generate a 20 kHz PWM with 50% duty cycle. The microcontroller is running on a 20 MHz crystal.

Step #1: Choose Timer2 Prescaler

a) F_{PWM} = Fosc/((PR2+1)*4*prescaler) = 19531 Hz for PR2 = 255 and prescaler of 1

b) This frequency is lower than 20 kHz, therefore a prescaler of 1 is adequate.

Step #2: Calculate PR2

PR2 = Fosc/(F_{PWM}*4*prescaler) – 1 = 249

Step #3: Determine CCPR1L and CCP1CON<5:4>

a) CCPR1L:CCP1CON<5:4> = DutyCycle*0x3FF = 0x1FF

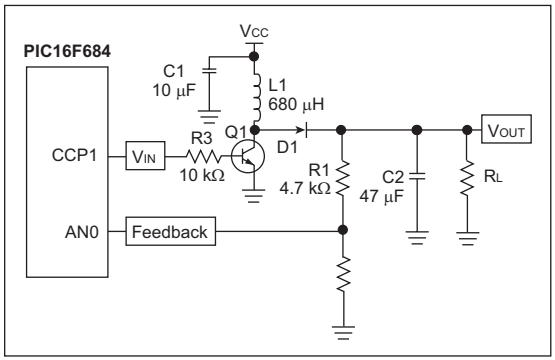
b) CCPR1L = 0x1FF >> 2 = 0x7F, CCP1CON<5:4> = 3

Step #4: Configure CCP1CON

The CCP module is configured in PWM mode with the Least Significant bits of the duty cycle set, therefore, CCP1CON = '`b001111000`'.

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.

Example 20-1: Transmit Routine

```
TxRountine
  MOVLW  8          ;preload bit counter
                    ;with 8
  MOVWF  counter
  BCF    TxLine     ;line initially high,
                    ;toggle low for START
                    ;bit
TxLoop
  CALL   DelayTb    ;wait Tb (bit period)
  RRF    RxByte,f   ;rotate LSB first into
                    ;the Carry flag
  BTFSS  STATUS,C   ;Tx line state equals
                    ;state of Carry flag
  BCF    TxLine
  BTFSC  STATUS,C
  BSF    TxLine
  DECFSZ Counter,f  ;Repeat 8 times
  GOTO   TxLoop
  CALL   Delay Tb   ;Delay Tb before
                    ;sending STOP bit
  BSF    TxLine     ;send STOP bit
```

Example 20-2: Receive Routine

```
RxRoutine
  BTFSC  RxLine       ;wait for receive
                      ;line to go low
  GOTO   RxRoutine
  MOVLW  8            ;initialize bit
                      ;counter to 8
  MOVWF  Counter
  CALL   Delay1HalfTb;delay 1/2 Tb here
                      ;plus Tb in RxLoop
                      ;in order to sample
                      ;at the right time
RxLoop
  CALL   DelayTb      ;wait Tb (bit
                      ;period)
  BTFSS  RxLine       ;Carry flag state
                      ;equals Rx line
                      ;state
  BCF    STATUS,C
  BTFSC  RxLine
  BSF    STATUS,C
  BTFSC  RxLine
  BSF    STATUS,C
  RRF    RxByte,f     ;Rotate LSB first
                      ;into receive type
  DECFSZ Counter,f    ;Repeat 8 times
  GOTO   RxLoop
```

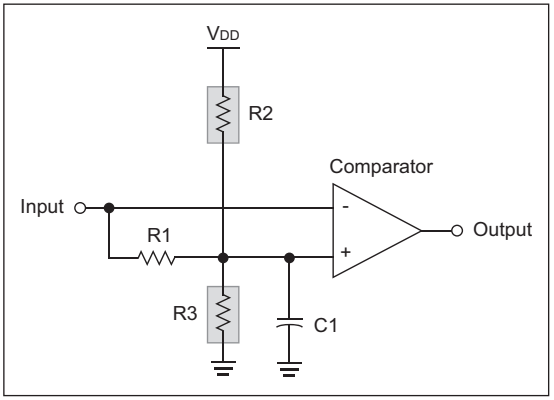
TIP #6 Data Slicer

In both wired and wireless data transmission, the data signal may be subject to DC offset shifts due to temperature shifts, ground currents or other factors in the system. When this happens, using a simple level comparison to recover the data is not possible because the DC offset may exceed the peak-to-peak amplitude of the signal. The circuit typically used to recover the signal in this situation is a data slicer.

The data slicer shown in Figure 6-1 operates by comparing the incoming signal with a sliding reference derived from the average DC value of the incoming signal. The DC average value is found using a simple RC low-pass filter (R1 and C1). The corner frequency of the RC filter should be high enough to ignore the shifts in the DC level while low enough to pass the data being transferred.

Resistors R2 and R3 are optional. They provide a slight bias to the reference, either high or low, to give a preference to the state of the output when no data is being received. R2 will bias the output low and R3 will bias the output high. Only one resistor should be used at a time, and its value should be at least 50 to 100 times larger than R1.

Figure 6-1: Data Slicer



Example:

Data rate of 10 kbits/second. A low pass filter frequency of 500 Hz: R1 = 10k, C1 = 33 μF. R2 or R3 should be 500k to 1 MB.

TIP #7 One-Shot

When dealing with short duration signals or glitches, it is often convenient to stretch out the event using a mono-stable, multi-vibrator or one-shot. Whenever the input pulses, the one-shot fires holding its output for a preset period of time. This stretches the short trigger input into a long output which the microcontroller can capture.

The circuit is designed with two feedback paths around a comparator. The first is a positive hysteresis feedback which sets a two level threshold, V_{HI} and V_{LO}, based on the state of the comparator output. The second feedback path is an RC time circuit.

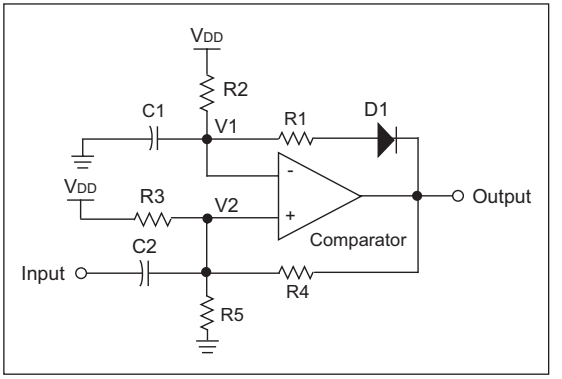
The one-shot circuit presented in Figure 7-1 is triggered by a low-high transition on its input and generates a high output pulse. Using the component values from the example, the circuit’s operation is as follows.

Prior to triggering, C1 will have charged to a voltage slightly above 0.7V due to resistor R2 and D1 (R1 << R2 and will have only a minimal effect on the voltage). The comparator output will be low, holding the non-inverting input slightly below 0.7V due to the hysteresis feedback through R3, R4 and R5 (the hysteresis lower limit is designed to be less than 0.7V). With the non-inverting input held low, C2 will charge up to the difference between the circuit input and the voltage present at the non-inverting input.

When the circuit input is pulsed high, the voltage present at the non-inverting input is pulled above 0.7V due to the charge in C2. This causes the output of the comparator to go high, the hysteresis voltage at the non-inverting input goes to the high threshold voltage, and C1 begins charging through R2.

When the voltage across C1 exceeds the high threshold voltage, the output of the comparator goes low, C1 is discharged to just above the 0.7V limit, the non-inverting input is pulled below 0.7V, and the circuit is reset for the next pulse input, waiting for the next trigger input.

Figure 7-1: One-Shot Circuit



To design the one-shot, first create the hysteresis feedback using the techniques from Tip #3. Remember to set the low threshold below 0.7V. Next, choose values for R2 and C1 using Equation 7-1.

Equation 7-1

$$T_{PULSE} = \frac{R2 * C1 * \ln(V_{TH}/V_{TL})}{4}$$

D1 can be any low voltage switching diode. R1 should be 1% to 2% of R2 and C2 should be between 100 and 220 pF.

Example:

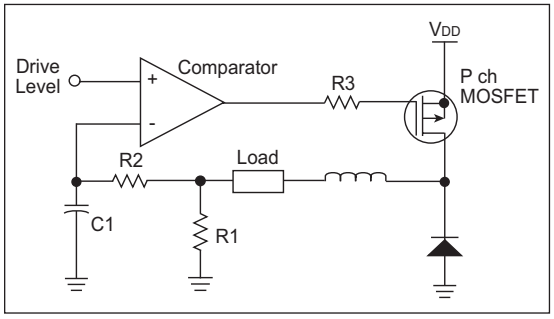
- V_{DD} = 5V, V_{TH} = 3.0V, V_{TL} = 2.5V
- From Tip #3, R4 = 1k, R5 = 1.5k and R3 = 12k
- T_{PULSE} = 1ms, C1 = .1 μF and R2 = 15k
- D1 is a 1N4148, R1 = 220 and C2 = 150 pF

TIP #13 PWM High-Current Driver

This tip combines a comparator with a MOSFET transistor and an inductor to create a switch mode high-current driver circuit. (See Figure 13-1).

The operation of the circuit begins with the MOSFET off and no current flowing in the inductor and load. With the sense voltage across R1 equal to zero and a DC voltage present at the drive level input, the output of the comparator goes low. The low output turns on the MOSFET and a ramping current builds through the MOSFET, inductor, load and R1.

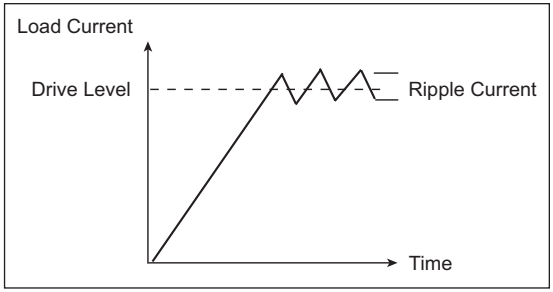
Figure 13-1: High Current Driver



When the current ramps high enough to generate a voltage across R1 equal to the drive level, the comparator output goes high turning off the MOSFET. The voltage at the junction of the MOSFET and the inductor then drops until D1 forward biases. The current continues ramping down from its peak level toward zero. When the voltage across the sense resistor R1 drops below the drive level, the comparator output goes low, the MOSFET turns on, and the cycle starts over.

R2 and C1 form a time delay network that limits the switching speed of the driver and causes it to slightly overshoot and undershoot the drive level when operating. The limit is necessary to keep the switching speed low, so the MOSFET switches efficiently. If R2 and C1 were not present, the system would run at a speed set by the comparator propagation delay and the switching speed of the MOSFET. At that speed, the switching time of the MOSFET would be a significant portion of the switching time and the switching efficiency of the MOSFET would be too low.

Figure 13-1: Current Through the Load



To design a PWM high current driver, first determine a switching speed (F_{SWX}) that is appropriate for the system. Next, choose a MOSFET and D1 capable of handling the load current requirements. Then choose values for R2 and C1 using Equation 13-1.

Equation 13-1

$$F_{SWX} = \frac{2}{R2 * C1}$$

Next determine the maximum ripple current that the load will tolerate, and calculate the required inductance value for L1 using Equation 13-2.

Equation 13-2

$$L = \frac{V_{DD} - V_{LOAD}}{I_{RIPPLE} * F_{SWX} * 2}$$

Finally, choose a value for R1 that will produce a feedback ripple voltage of 100 mV for the maximum ripple current I_{RIPPLE}.

Example:

- F_{SWX} = 10 kHz, R2 = 22k, C1 = .01 µF
- I_{RIPPLE} = 100 mA, V_{DD} = 12V, V_L = 3.5V
- L = 4.25 mH

TIP #10 How to Update LCD Data Through Firmware

To update the LCD, the content of the LCDDATA registers is modified to turn on, or off, each pixel on the LCD display. The application firmware will usually modify buffer variables that are created to correspond with elements on the display, such as character positions, bar graph, battery display, etc.

When the application calls for a display update, the values stored in the buffer variables must be converted to the correct setting of the pixel bits, located in the LCDDATA registers.

For Type-A waveforms, the LCD Data registers may be written any time without ill effect. However, for Type-B waveforms, the LCD Data registers can only be written every other LCD frame in order to ensure that the two frames of the Type-B waveform are compliments of one another. Otherwise, a DC bias can be presented to the LCD.

The LCD Data registers should only be written when a write is allowed, which is indicated by the WA bit in the LCDCON register being set.

On the PIC16C926 parts, there is no WA bit. The writing of the pixel data can be coordinated on an LCD interrupt. The LCD interrupt is only generated when a multiplexed (not static) Type-B waveform is selected.

TIP #11 Blinking LCD

Information can be displayed in more than one way with an LCD panel. For example, how can the user's attention be drawn to a particular portion of the LCD panel? One way that does not require any additional segments is to create a blinking effect.

Look at a common clock application. The ":" between the hours and minutes is commonly made to blink once a second (on for half a second, off for half a second). This shows that the clock is counting in absence of the ticking sound or second hand that accompanies the usual analog face clock. It serves an important purpose of letting the user know that the clock is operating.

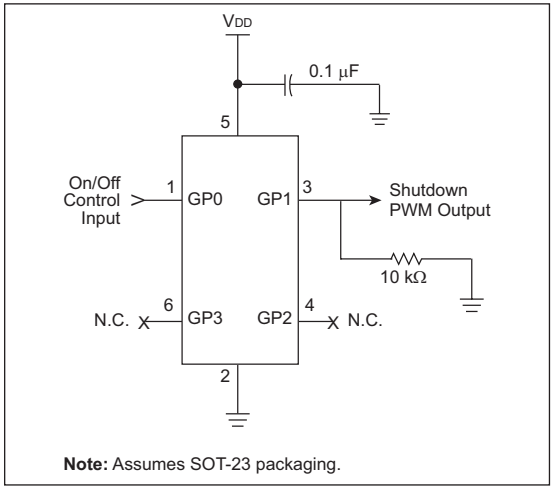
If there is a power outage, then it is common for the entire clock display to blink. This gives the user of the clock an immediate indication that the clock is no longer showing the correct time.

When the user sets the time, then blinking is commonly used to show that a new mode has been entered, such as blinking the hours to identify that the hours are being set, or blinking the minutes to show that the minutes are being set. In a simple clock, blinking is used for several different purposes. Without blinking effects, the common digital clock would not be nearly as user friendly.

TIP #1 Soft-Start Using a PIC10F200

Almost all power supply controllers are equipped with shutdown inputs that can be used to disable the MOSFET driver outputs. Using Pulse-Width Modulation (PWM), the amount of time the power supply is allowed to operate can be slowly incremented to allow the output voltage to slowly rise from 0% to 100%.

Figure 1-1: Soft-Start Circuit Schematic



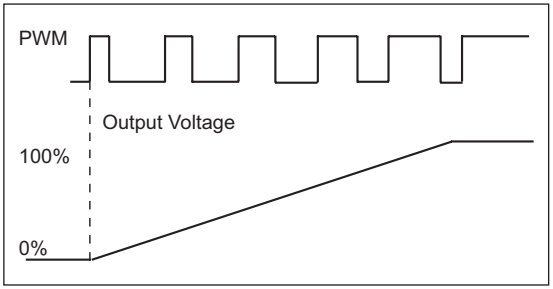
Note: Assumes SOT-23 packaging.

This technique is called soft-start and is used to prevent the large inrush currents that are associated with the start-up of a switching power supply.

GP0 on the PIC MCU is used to enable or disable the soft-start. Once enabled, the on-time of the PWM signal driving the shutdown output will increase each cycle until the power supply is fully on.

During the PIC MCU Power-on Reset, the PWM output (GP1) is initially in a high-impedance state. A pull-down resistor on the PWM output ensures the power supply will not unexpectedly begin operating.

Figure 1-2: Timing Diagram



It is important to note that this type of soft-start controller can only be used for switching regulators that respond very quickly to changes on their shutdown pins (such as those that do cycle-by-cycle limiting). Some linear regulators have active-low shutdown inputs, however, these regulators do not respond fast enough to changes on their shutdown pins in order to perform soft-start.

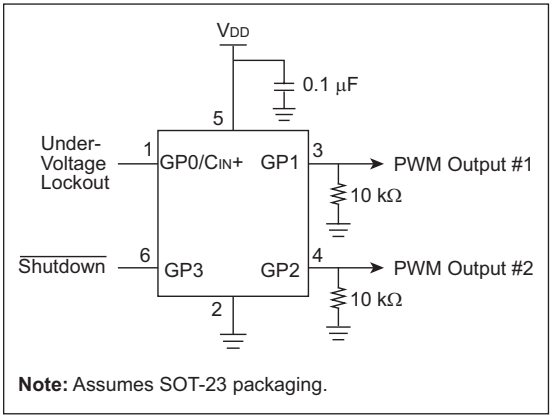
Example software is provided for the PIC10F200 which was taken from TB081. Please refer to TB081, “*Soft-Start Controller For Switching Power Supplies*” (DS91081) for more information.

TIP #2 A Start-Up Sequencer

Some new devices have multiple voltage requirements (e.g., core voltages, I/O voltages, etc.). The sequence in which these voltages rise and fall may be important.

By expanding on the previous tip, a start-up sequencer can be created to control two output voltages. Two PWM outputs are generated to control the shutdown pins of two SMPS controllers. Again, this type of control only works on controllers that respond quickly to changes on the shutdown pin (such as those that do cycle-by-cycle limiting).

Figure 2-1: Multiple PWM Output Soft-Start Controller



Note: Assumes SOT-23 packaging.

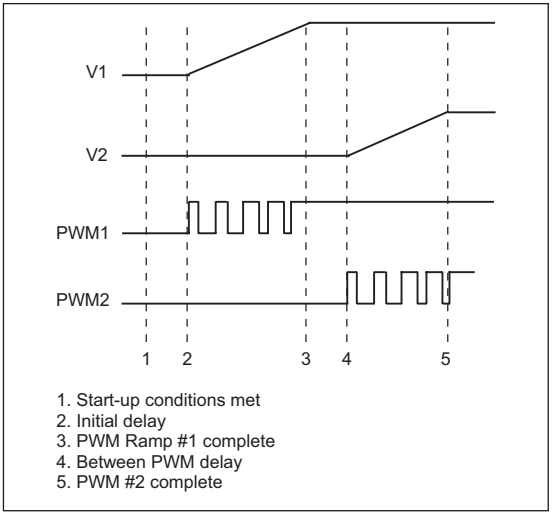
This design uses the PIC MCU comparator to implement an under-voltage lockout. The input on the GP0/CIN+ pin must be above the internal 0.6V reference for soft-start to begin, as shown in Figure 2-2.

Two conditions must be met in order for the soft-start sequence to begin:

1. The shutdown pin must be held at VDD (logic high).
2. The voltage on GP0 must be above 0.6V.

Once both start-up conditions are met, the sequences will delay and PWM #1 will ramp from 0% to 100%. A second delay allows the first voltage to stabilize before the sequencer ramps PWM #2 from 0% to 100%. All delays and ramp times are under software control and can be customized for specific applications. If either soft-start condition becomes invalid, the circuit will shutdown the SMPS controllers.

Figure 2-2: Timing Diagram



Example software is provided for the PIC10F200 which was taken from TB093, “*Multiple PWM Output Soft-Start Controller for Switching Power Supplies*” (DS91093).

TIP #3 A Tracking and Proportional Soft-Start of Two Power Supplies

Expanding on the previous tip, we can also use a PIC MCU to ensure that two voltages in a system rise together or rise proportionally to one another, as shown in Figure 3-1. This type of start-up is often used in applications with devices that require multiple voltages (such as I/O and core voltages).

Like the previous two, this tip is designed to control the shutdown pin of the SMPS controller and will only work with controllers that respond quickly to changes on the shutdown pin.

Figure 3-1: Timing Diagram

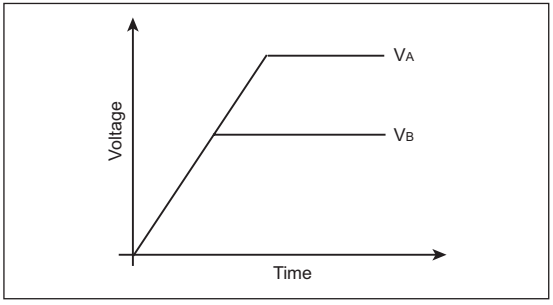
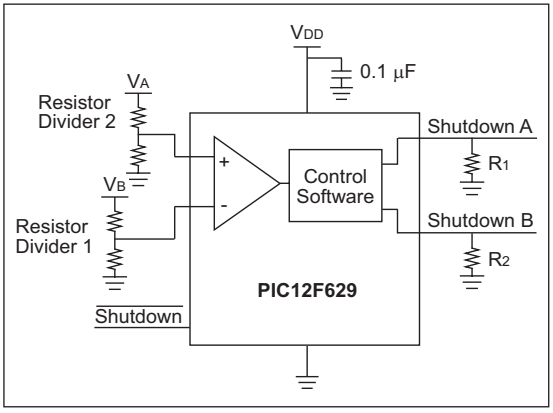


Figure 3-2: Example Schematic



The comparator of the PIC MCU is used to determine which voltage is higher and increases the on-time of the other output accordingly. The logic for the shutdown pins is as shown in Table 3-1.

Table 3-1: Shutdown Pin Logic

Case	Shutdown A	Shutdown B
$V_A > V_B$	Low	High
$V_B > V_A$	High	Low
V_B > Internal Reference	High	High

To determine if it has reached full voltage, V_B is compared to the internal voltage reference. If V_B is higher, both shutdown outputs are held high.

Resistor Divider 1 should be designed so that the potentiometer output is slightly higher than the comparator voltage reference when V_B is at full voltage.

The ratio of resistors in Resistor Divider 2 can be varied to change the slope at which V_A rises.

Pull-down resistors ensure the power supplies will not operate unexpectedly when the PIC MCU is being reset.

TIP #14 Brushless DC Fan Speed Control

There are several methods to control the speed of a DC brushless fan. The type of fan, allowable power consumption and the type of control desired are all factors in choosing the appropriate type.

Figure 14-1: Low-Side PWM Drive

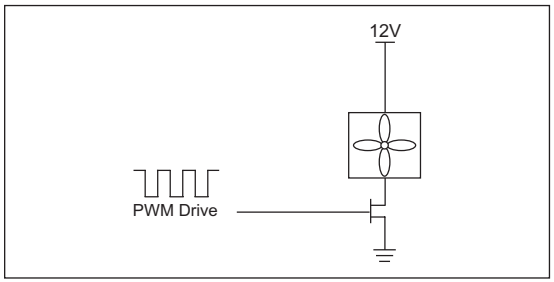
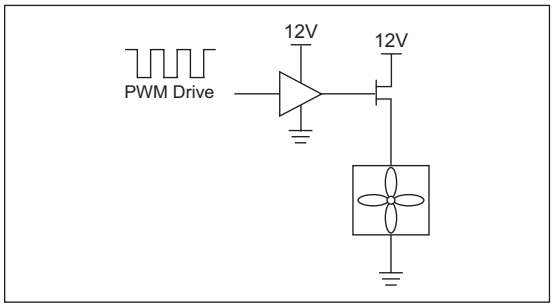


Figure 14-2: High-Side PWM Drive



Method 1 – Pulse-Width Modulation

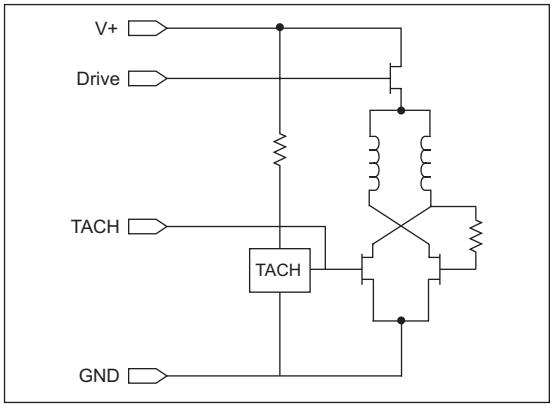
As shown in Figure 14-1 and Figure 14-2, a simple PWM drive may be used to switch a two-wire fan on and off. While it is possible to use the circuit in Figure 14-1 without a high-side MOSFET driver, some manufacturers state that switching on the low side of the fan will void the warranty.

Because of this, it is necessary to switch the high side of the fan in order to control the speed. The simplest type of speed control is ‘on’ or ‘off’. However, if a higher degree of control is desired, PWM can be used to vary the speed of the fan.

For 3-wire fans, the tachometer output will not be accurate if PWM is used. The sensor providing the tachometer output on 3-wire fans is powered from the same supply as the fan coils, thus using a PWM to control fan speed will render the fan’s tachometer inaccurate.

One solution is to use a 4-wire fan which includes both the tachometer output and a drive input. Figure 14-3 shows a diagram of a 4-wire fan.

Figure 14-3: Typical 4-Wire Fan



A 4-wire fan allows speed to be controlled using PWM via the Drive line. Since power to the tachometer sensor is not interrupted, it will continue to output the correct speed.

TIP #20 Compensating Sensors Digitally

Many sensors and references tend to drift with temperature. For example, the MCP9700 specification states that its typical is ±0.5°C and its max error is ±4°C.

Figure 20-1: MCP9700 Accuracy

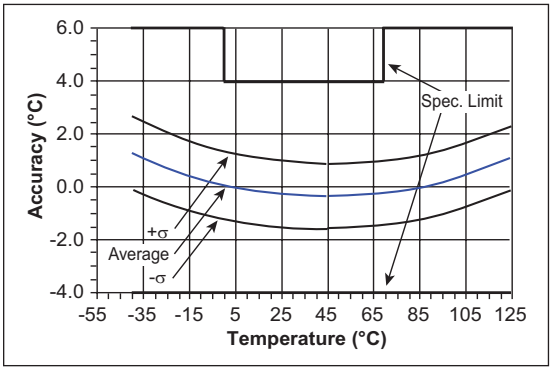


Figure 20-1 shows the accuracy of a 100 sample lot of MCP9700 temperature sensors. Despite the fact that the sensor’s error is nonlinear, a PIC microcontroller (MCU) can be used to compensate the sensor’s reading.

Polynomials can be fitted to the average error of the sensor. Each time a temperature reading is received, the PIC MCU can use the measured result and the error compensation polynomials to determine what the true temperature is.

Figure 20-2: MCP9700 Average Accuracy After Compensation

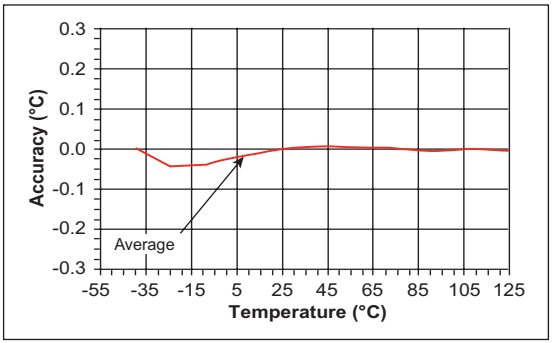


Figure 20-2 shows the average accuracy for the 100 sample lot of MCP9700 temperature sensors after compensation. The average error has been decreased over the full temperature range.

It is also possible to compensate for error from voltage references using this method.

For more information on compensating a temperature sensor digitally, refer to AN1001, “*IC Temperature Sensor Accuracy Compensation with a PIC Microcontroller*” (DS01001).