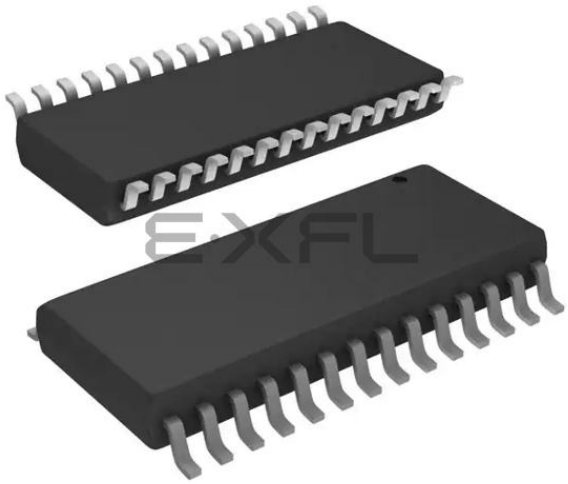
Microchip Technology - PIC16C63AT-04I/SO Datasheet



Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

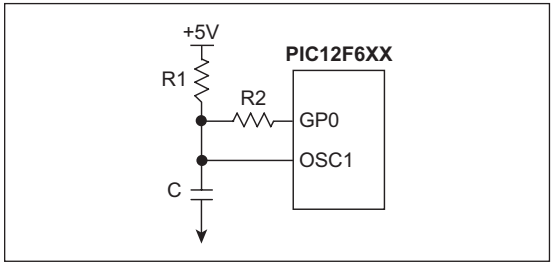
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" 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	4MHz
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/pic16c63at-04i-so

TIP #1 Dual Speed RC Oscillator

Figure 1-1



1. After reset I/O pin is High-Z
2. Output ‘1’ on I/O pin
3. R1, R2 and C determine OSC frequency
4. Also works with additional capacitors

Frequency of PIC MCU in external RC oscillator mode depends on resistance and capacitance on OSC1 pin. Resistance is changed by the output voltage on GP0. GP0 output ‘1’ puts R2 in parallel with R1 reduces OSC1 resistance and increases OSC1 frequency. GP0 as an input increases the OSC1 resistance by minimizing current flow through R2, and decreases frequency and power consumption.

Summary:

GP0 = Input: Slow speed for low current

GP0 = Output high: High speed for fast processing

TIP #2 Input/Output Multiplexing

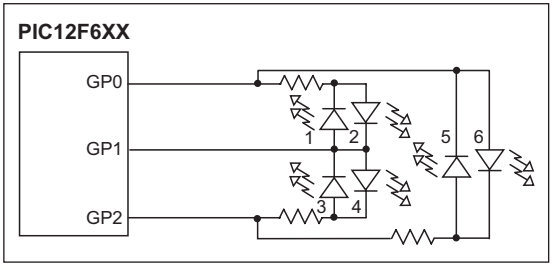
Individual diodes and some combination of diodes can be enabled by driving I/Os high and low or switching to inputs (Z). The number of diodes (D) that can be controlled depends on the number of I/Os (GP) used.

The equation is: D = GP x (GP - 1).

Example 2-1: Six LEDs on Three I/O Pins

GPx			LEDs					
0	1	2	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
0	1	Z	1	0	0	0	0	0
1	0	Z	0	1	0	0	0	0
Z	0	1	0	0	1	0	0	0
Z	1	0	0	0	0	1	0	0
0	Z	1	0	0	0	0	1	0
1	Z	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	0	1	0
1	0	0	0	1	0	0	0	1
1	0	1	0	1	1	0	0	0
1	1	0	0	0	0	1	0	1
1	1	1	0	0	0	0	0	0

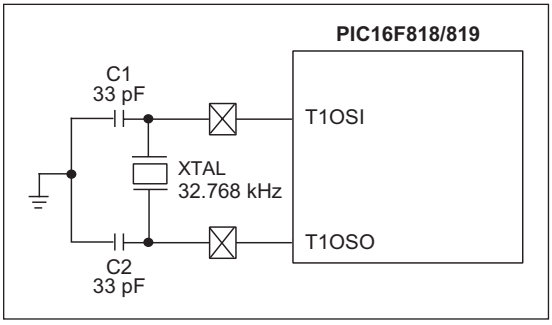
Figure 2-1



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.

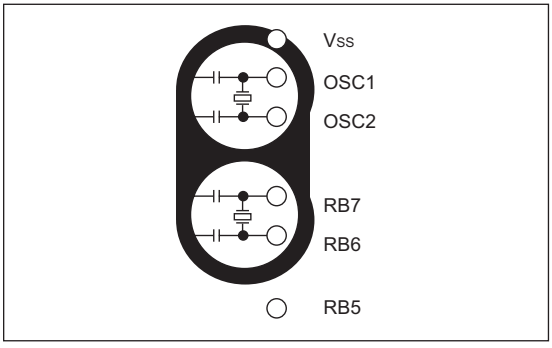
Doze mode is another low power mode available in PIC24, dsPIC DSCs, and PIC32 devices. In Doze mode, the system clock to the CPU is postscaled so that the CPU runs at a lower speed than the peripherals. If the CPU is not tasked heavily and peripherals need to run at high speed, then Doze mode can be 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.

TIP #19 Low Power Timer1 Oscillator Layout

Applications requiring very low power Timer1/SOSC oscillators on nanoWatt and nanoWatt XLP devices must take PCB layout into consideration. The very low power Timer1/SOSC oscillators on nanoWatt and nanoWatt XLP devices consume very little current, and this sometimes makes the oscillator circuit sensitive to neighboring circuits. The oscillator circuit (crystal and capacitors) should be located as close as possible to the microcontroller.

No circuits should be passing through the oscillator circuit boundaries. If it is unavoidable to have high-speed circuits near the oscillator circuit, a guard ring should be placed around the oscillator circuit and microcontroller pins similar to the figure below. Placing a ground plane under the oscillator components also helps to prevent interaction with high speed circuits.

Figure 19-1: Guard Ring Around Oscillator Circuit and MCU Pins



TIP #20 Use LVD to Detect Low Battery

The Low Voltage Detect (LVD) interrupt present in many PIC MCUs is critical in battery based systems. It is necessary for two reasons. First, many devices cannot run full speed at the minimum operating voltage. In this case, the LVD interrupt indicates when the battery voltage is dropping so that the CPU clock can be slowed down to an appropriate speed, preventing code misexecution. Second, it allows the MCU to detect when the battery is nearing the end of its life, so that a low battery indication can be provided and a lower power state can be entered to maximize battery lifetime. The LVD allows these functions to be implemented without requiring the use of extra analog channels to measure the battery level.

TIP #21 Use Peripheral FIFO and DMA

Some devices have peripherals with DMA or FIFO buffers. These features are not just useful to improve performance; they can also be used to reduce power. Peripherals with just one buffer register require the CPU to stay operating in order to read from the buffer so it doesn’t overflow. However, with a FIFO or DMA, the CPU can go to sleep or idle until the FIFO fills or DMA transfer completes. This allows the device to consume a lot less average current over the life of the application.

CHAPTER 3
PIC® Microcontroller CCP and ECCP Tips 'n Tricks

Table Of Contents

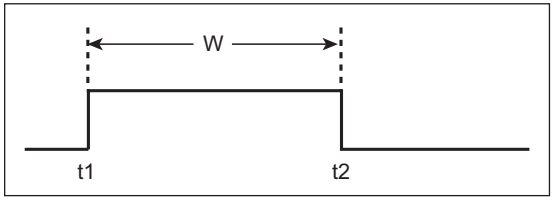
TIPS 'N TRICKS INTRODUCTION

Microchip continues to provide innovative products that are smaller, faster, easier-to-use and more reliable. PIC® microcontrollers (MCUs) are used in a wide range of everyday products, from washing machines, garage door openers and television remotes to industrial, automotive and medical products.

The Capture, Compare and PWM (CCP) modules that are found on many of Microchip's microcontrollers are used primarily for the measurement and control of time-based pulse signals. The Enhanced CCP (ECCP), available on some of Microchip's devices, differs from the regular CCP module in that it provides enhanced PWM functionality – namely, full-bridge and half-bridge support, programmable dead-band delay and enhanced PWM auto-shutdown. The ECCP and CCP modules are capable of performing a wide variety of tasks. This document will describe some of the basic guidelines to follow when using these modules in each mode, as well as give suggestions for practical applications.

TIP #3 Measuring Pulse Width

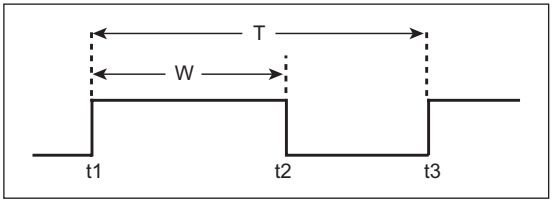
Figure 3-1: Pulse Width



1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every rising edge of the waveform.
2. Configure Timer1 prescaler so that Timer1 will run WMAX without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When CCP interrupt occurs, save the captured timer value (t1) and reconfigure control bits to capture every falling edge.
5. When CCP interrupt occurs again, subtract saved value (t1) from current captured value (t2) – this result is the pulse width (W).
6. Reconfigure control bits to capture the next rising edge and start process all over again (repeat steps 3 through 6).

TIP #4 Measuring Duty Cycle

Figure 4-1: Duty Cycle



The duty cycle of a waveform is the ratio between the width of a pulse (W) and the period (T). Acceleration sensors, for example, vary the duty cycle of their outputs based on the acceleration acting on a system. The CCP module, configured in Capture mode, can be used to measure the duty cycle of these types of sensors. Here's how:

1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every rising edge of the waveform.
2. Configure Timer1 prescaler so that Timer1 will run TMAX(1) without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When CCP interrupt occurs, save the captured timer value (t1) and reconfigure control bits to capture every falling edge.

Note 1: TMAX is the maximum pulse period that will occur.

5. When the CCP interrupt occurs again, subtract saved value (t1) from current captured value (t2) – this result is the pulse width (W).
6. Reconfigure control bits to capture the next rising edge.
7. When the CCP interrupt occurs, subtract saved value (t1) from the current captured value (t3) – this is the period (T) of the waveform.
8. Divide T by W – this result is the Duty Cycle.
9. Repeat steps 4 through 8.

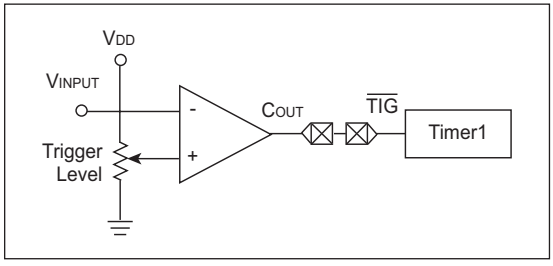
TIP #4 Pulse Width Measurement

To measure the high or low pulse width of an incoming analog signal, the comparator can be combined with Timer1 and the Timer1 Gate input option (see Figure 4-1). Timer1 Gate acts as a count enable for Timer1. If the input is low, Timer1 will count. If the $\overline{T1G}$ input is high, Timer1 does not count. Combining $\overline{T1G}$ with the comparator allows the designer to measure the time between a high-to-low output change and a low-to-high output change.

To make a measurement between a low-to-high and a high-to-low transition, the only change required is to set the CINV bit in the comparator CMCON register which inverts the comparator output.

Because the output of the comparator can change asynchronously with the Timer1 clock, only comparators with the ability to synchronize their output with the Timer1 clock should be used and their C2SYNC bits should be set.

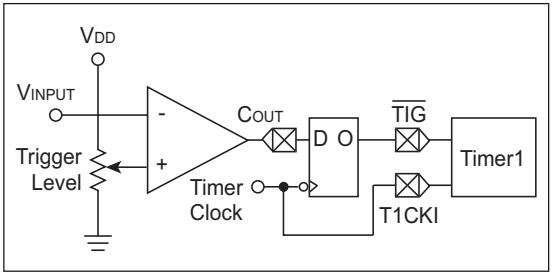
Figure 4-1: Comparator with Timer1 and $\overline{T1G}$



If the on-chip comparator does not have the ability to synchronize its output to the Timer1 clock, the output can be synchronized externally using a discrete D flip-flop (see Figure 4-2).

Note: The flip-flop must be falling edge triggered to prevent a race condition.

Figure 4-2: Externally Synchronized Comparator



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.

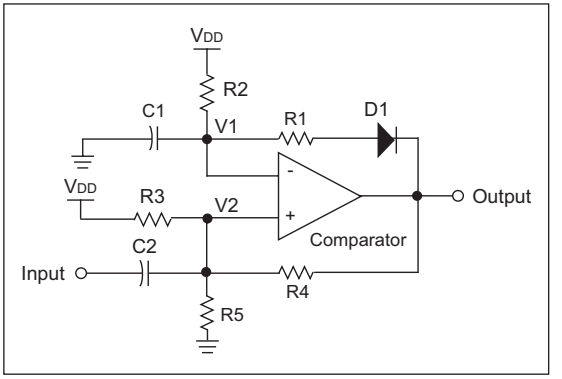
The one-shot circuit presented in Figure 7-1 is triggered by a low-high transition on its input and generates a high output pulse. Using the component values from the example, the circuit’s operation is as follows.

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:

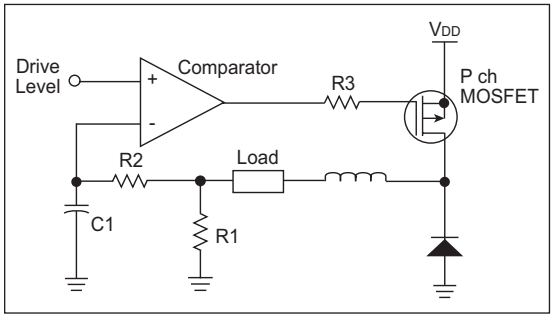
- V_{DD} = 5V, V_{TH} = 3.0V, V_{TL} = 2.5V
- From Tip #3, R4 = 1k, R5 = 1.5k and R3 = 12k
- T_{PULSE} = 1ms, C1 = .1 μF and R2 = 15k
- D1 is a 1N4148, R1 = 220 and C2 = 150 pF

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.

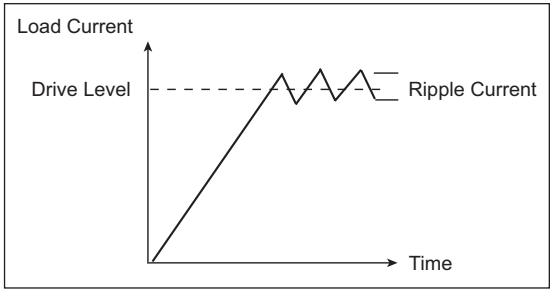
Figure 13-1: High Current Driver



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 #17 Logic: AND/NAND Gate

This tip shows the use of the comparator to implement an AND gate and its complement the NAND gate (see Figure 17-2). Resistors R1 and R2 drive the non-inverting input with 2/3 the supply voltage. Resistors R3 and R4 average the voltage of input A and B at the inverting input. If either A or B is low, the average voltage will be one half V_{DD} and the output of the comparator remains low. The output will go high only if both inputs A and B are high, which raises the input to the inverting input above 2/3 V_{DD}.

The operation of the NAND gate is identical to the AND gate, except that the output is inverted due to the swap of the inverting and non-inverting inputs.

Note: Typical propagation delay for the circuit is 250-350 ns using the typical on-chip comparator peripheral of a microcontroller. Delay measurements were made with 10k resistance values.

While the circuit is fairly simple, there are a few requirements for correct operation:

1. The inputs A and B must drive from ground to V_{DD} for the circuit to operate properly.
2. The combination of R1 and R2 will draw current constantly, so they must be kept large to minimize current draw.
3. All resistances on the inverting input react with the input capacitance of the comparator. So the speed of the gate will be affected by the source resistance of A and B, as well as, the size of resistors R3 and R4.
4. Resistor R2 must be 2 x R1.
5. Resistor R3 must be equal to R4.

Figure 17-1: AND Gate

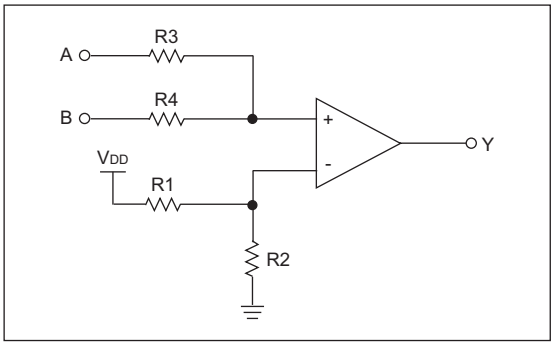
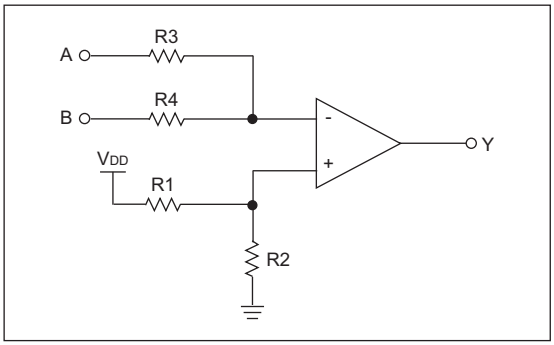


Figure 17-2: NAND Gate



Example:

- V_{DD} = 5V, R3 = R4 = 10k
- R1 = 5.1k, R2 = 10k

Application Note References

- AN532, “*Servo Control of a DC Brush Motor*” (DS00532)
- AN696, “*PIC18CXXX/PIC16CXXX DC Servomotor*” (DS00696)
- AN718, “*Brush-DC Servomotor Implementation using PIC17C756A*” (DS00718)
- AN822, “*Stepper Motor Microstepping with the PIC18C452*” (DS00822)
- AN843, “*Speed Control of 3-Phase Induction Motor Using PIC18 Microcontrollers*” (DS00843)
- AN847, “*RC Model Aircraft Motor Control*” (DS00847)
- AN857, “*Brushless DC Motor Control Made Easy*” (DS00857)
- AN885, “*Brushless DC (BLDC) Motor Fundamentals*” (DS00885)
- AN899, “*Brushless DC Motor Control Using the PIC18FXX31*” (DS00899)
- AN893, “*Low-cost Bidirectional Brushed DC Motor Control Using the PIC16F684*” (DS00893)
- AN894, “*Motor Control Sensor Feedback Circuits*” (DS00894)
- AN898, “*Determining MOSFET Driver Needs for Motor Drive Applications*” (DS00898)
- AN901, “*Using the dsPIC30F for Sensorless BLDC Control*” (DS00901)
- AN905, “*Brushed DC Motor Fundamentals*” (DS00905)
- AN906, “*Stepper Motor Control Using the PIC16F684*” (DS00906)
- AN907, “*Stepper Motor Fundamentals*” (DS00907)
- AN1017, “*Sinusoidal Control of PMSM Motors with dsPIC30F DSC*” (DS01017)
- GS001, “*Getting Started with BLDC Motors and dsPIC30F Devices*” (DS93001)

Application notes can be found on the Microchip web site at www.microchip.com.

Motor Control Development Tools

- PICDEM™ MC Development Board (DM183011)

  Used to evaluate the PIC18FXX31 8-bit microcontroller family.
- PICDEM™ MCLV Development Board (DM183021)
- dsPIC30F Motor Control Development System (DM300020)

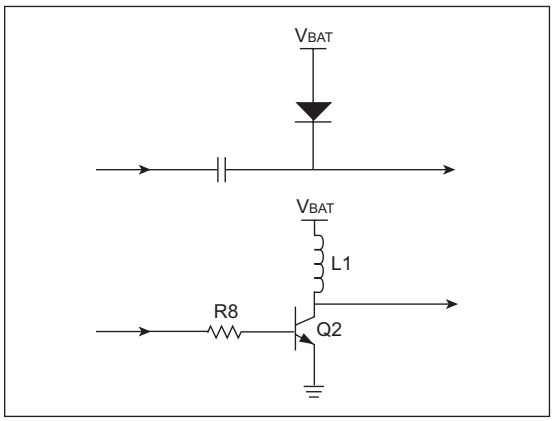
  Used to evaluate the dsPIC30F 16-bit Digital Signal Controller family.
- Motor Control (MC) Graphical User Interface (GUI)

  The MC-GUI allows user to configure the motor and a wide range of system parameters for a selected motor type.

  The MC-GUI is free an can be downloaded at www.microchip.com

  Visit the Motor Control Design Center at: www.microchip.com/motor for additional design resources.

Figure 5-2: Two Types of Boost Converter



The two methods of producing a boost converter are shown above. The first circuit is simply a switched capacitor type circuit. The second circuit is a standard inductor boost circuit. These circuits work by raising V_{DD}. This allows the voltage at V_{LCD} to exceed V_{DD}.

TIP #6: Software Controlled Contrast with PWM for LCD Contrast Control

In the previous contrast control circuits, the voltage output was set by a fixed reference. In some cases, the contrast must be variable to account for different operating conditions. The CCP module, available in the LCD controller devices, allows a PWM signal to be used for contrast control. In Figure 6-1, you see the buck contrast circuit modified by connecting the input to RA6 to a CCP pin. The resistor divider created by R4 and R5 in the previous design are no longer required. An input to the ADC is used to provide feedback but this can be considered optional. If the ADC feedback is used, notice that it is used to monitor the V_{DD} supply. The PWM will then be used to compensate for variations in the supply voltage.

Figure 6-1: Software Controlled Voltage Generator

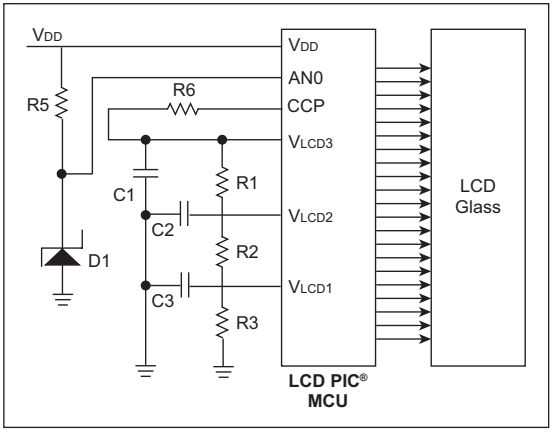
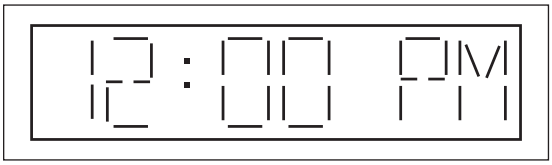
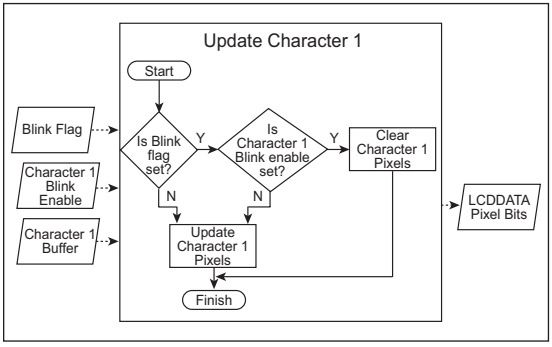


Figure 11-1: Common Clock Application



Fortunately, blinking is quite easy to implement. There are many ways to implement a blinking effect in software. Any regular event can be used to update a blink period counter. A blink flag can be toggled each time the blink period elapses. Each character or display element that you want to blink can be assigned a corresponding blink enable flag. The flowchart for updating the display would look like:

Figure 11-2: Updating Display Flowchart



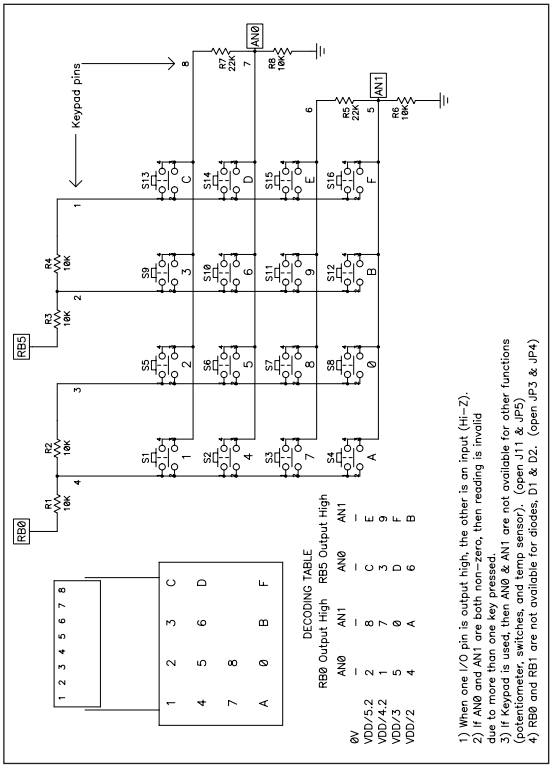
TIP #12 4 x 4 Keypad Interface that Conserves Pins for LCD Segment Drivers

A typical digital interface to a 4 x 4 keypad uses 8 digital I/O pins. But using eight pins as digital I/Os can take away from the number of segment driver pins available to interface to an LCD.

By using 2 digital I/O pins and 2 analog input pins, it is possible to add a 4 x 4 keypad to the PIC microcontroller without sacrificing any of its LCD segment driver pins.

The schematic for keypad hook-up is shown in Figure 12-1. This example uses the PIC18F8490, but the technique could be used on any of the LCD PIC MCUs.

Figure 12-1: Keypad Hook-up Schematic

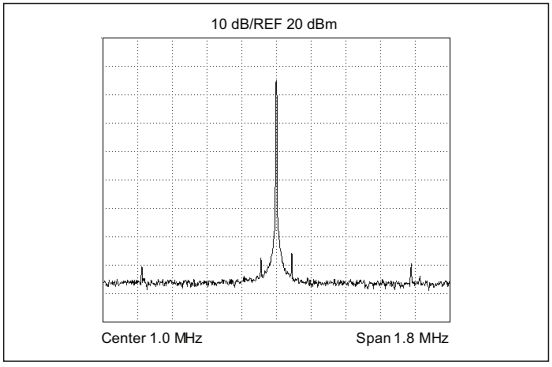


TIP #4 Creating a Dithered PWM Clock

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.

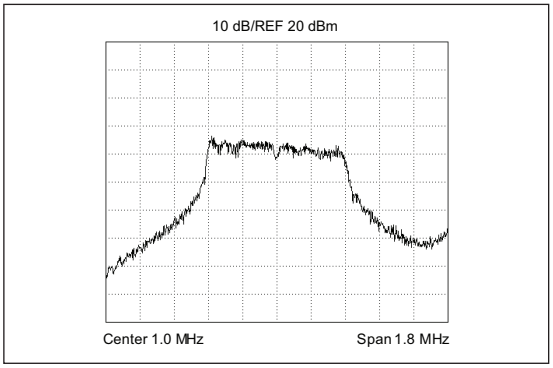
The PIC10F200 has an internal 4 MHz oscillator. A scaled version of oscillator can be output on a pin (Fosc/4). The scaled output is 1/4 of the oscillator frequency (1 MHz) and will always have a 50% duty cycle. Figure 4-1 shows a spectrum analyzer shot of the output of the Fosc/4 output.

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 #12 Using Auto-Shutdown CCP

PWM Auto-Shutdown

Several of Microchip’s PIC MCUs, such as the PIC16F684, PIC16F685 and PIC16F690, have a PWM auto-shutdown feature. When auto-shutdown is enabled, an event can terminate the current PWM pulse and prevent subsequent pulses unless the event is cleared. The ECCP can be setup to automatically start generating pulses again once the event clears.

Figure 12-1: PWM Auto-Shutdown Timing

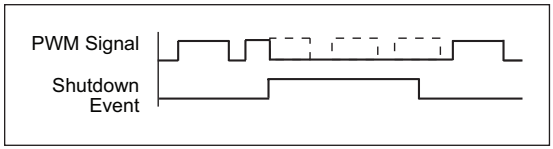
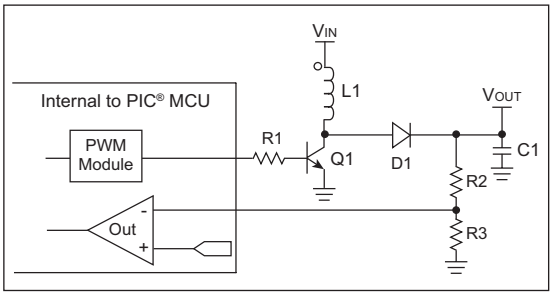


Figure 12-1 shows an example timing for the PWM auto-shutdown. When the shutdown event occurs, the current pulse is immediately terminated. In this example, the next two pulses are also terminated because the shutdown event had not been cleared by the beginning of the pulse period. After the event has cleared, pulses are allowed to resume, but only at the beginning of a pulse period.

Using Auto-Shutdown to Create a Boost Supply

By using the auto-shutdown feature, a very simple SMPS can be created. Figure 12-2 shows an example boost power supply.

Figure 12-2: Boost Power Supply



This power supply configuration has several unique features:

1. The switching frequency is determined by the PWM frequency and, therefore, can be changed at any time.
2. The maximum on-time is determined by the PWM duty cycle and, therefore, can be changed any time. This provides a very easy way to implement soft-start.
3. On PIC MCUs that have a programmable reference module, the output voltage can be configured and changed at any time.

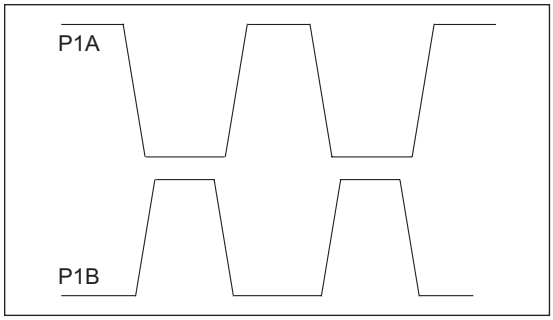
The topology can also be re-arranged to create other types of power supplies.

Example software is provided for the PIC16F685 (but can be adapted to any PIC MCU with the ECCP module). The software configures the PWM and comparator modules as shown in Figure 12-2.

TIP #13 Generating a Two-Phase Control Signal

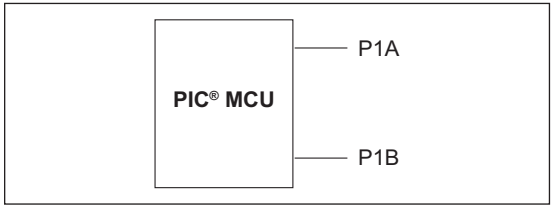
Power supplies using a push-pull topology or with multiple switching components require a two-phase control signal as shown in Figure 13-1.

Figure 13-1: Two-Phase Control Signal



It is possible to produce this type of control signal with two out-of-phase square waves using a PIC MCU with an ECCP module.

Figure 13-2: Two-Phase Control Signal Schematic



In order to configure the ECCP to produce this type of output:

1. Configure the ECCP in half H-bridge configuration PWM pulse with both outputs active-high.
2. Set the duty cycle register (CCPR1L) with the maximum duty cycle of 50%.
3. Change the programmable dead-time generator to reduce the pulse width to the desired value.

The programmable dead-time generator has a 7-bit resolution and, therefore, the resulting pulses will only have a 7-bit resolution. Each pulse will have a 50% duty cycle, less the dead time.

Using an internal 4 MHz clock produces 31 kHz output pulses, and using a 20 MHz crystal would produce 156 kHz output. The frequency of the output could be increased with a loss in resolution.

Example software is provided for the PIC16F684, but this tip is applicable to all PIC MCUs with ECCP modules.

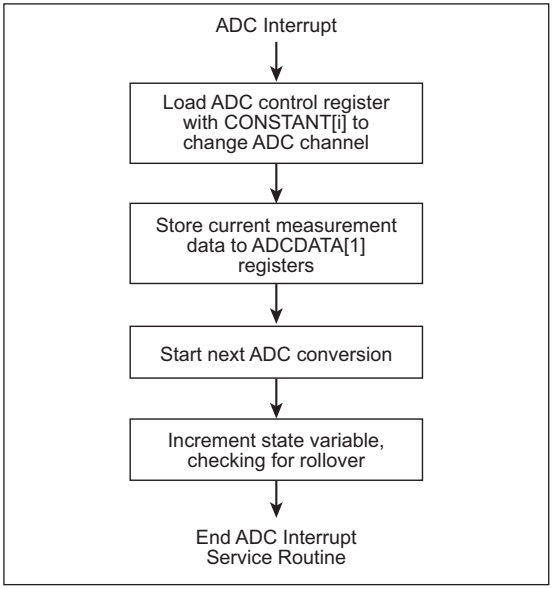
TIP #18 Data-Indexed Software State Machine

A state machine can be used to simplify a task by breaking the task up into smaller segments. Based on a state variable, the task performed or the data used by the state machine can be changed. There are three basic types of state machines: data-indexed, execution indexed and a hybrid of the two. This tip will focus on a data-indexed state machine.

The data-indexed state machine is ideal for monitoring multiple analog inputs with the Analog-to-Digital Converter (ADC). The state variable in these state machines determines which data is acted upon. In this case, the tasks of changing the ADC channel, storing the current result and starting a new conversion are always the same.

A very simple flow diagram for a data-indexed state machine is shown in Figure 18-1.

Figure 18-1: Data-Indexed State Machine Flowchart



As shown in Figure 18-1, a constant array (CONSTANT[i]) can be created to store the values to be loaded into the ADC control register to change the ADC channel. Furthermore, a data array (ADCDATA[i]) can be used to store the results of the ADC conversion. Finally, the next conversion is started and the logic required to increment and bind the state variable is executed.

This particular example used the ADC interrupt to signal when a conversion has completed, and will attempt to take measurements as quickly as possible. A subroutine could also be built to perform the same task, allowing the user to call the subroutine when needed.

Example software is provided using the PIC16F676 and RS-232 to monitor several ADC channels.

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.

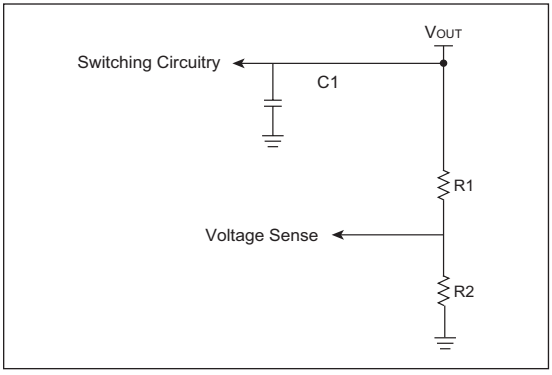
Means for safeguarding this problem include:

- 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.

TIP #21 Using Output Voltage Monitoring to Create a Self-Calibration Function

A PIC microcontroller can be used to create a switching power supply controlled by a PID loop (as described in Tip #16). This type of power supply senses its output voltage digitally, compares that voltage to the desired reference voltage and makes duty cycle changes accordingly. Without calibration, it is sensitive to component tolerances.

Figure 21-1: Typical Power Supply Output Stage



The output stage of many power supplies is similar to Figure 21-1. R1 and R2 are used to set the ratio of the voltage that is sensed and compared to the reference.

A simple means of calibrating this type of power supply is as follows:

1. Supply a known reference voltage to the output of the supply.
2. Place the supply in Calibration mode and allow it to sense that reference voltage.

By providing the supply with the output voltage that it is to produce, it can then sense the voltage across the resistor divider and store the sensed value. Regardless of resistor tolerances, the sensed value will always correspond to the proper output value for that particular supply.

Futhermore, this setup could be combined with Tip #20 to calibrate at several temperatures.

This setup could also be used to create a programmable power supply by changing the supplied reference and the resistor divider for voltage feedback.

TIP #7 3.3V → 5V Using a Diode Offset

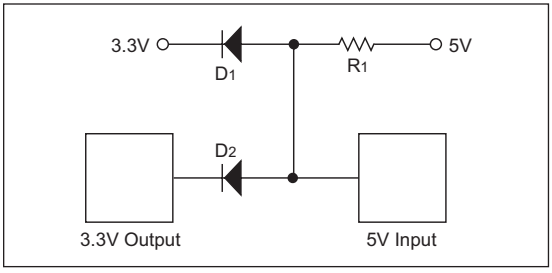
The inputs voltage thresholds for 5V CMOS and the output drive voltage for 3.3V LVTTL and LVCMOS are listed in Table 7-1.

Table 7-1: Input/Output Thresholds

	5V CMOS Input	3.3V LVTTL Output	3.3V LVCMOS Output
High Threshold	> 3.5V	> 2.4V	> 3.0V
Low Threshold	< 1.5V	< 0.4V	< 0.5V

Note that both the high and low threshold input voltages for the 5V CMOS inputs are about a volt higher than the 3.3V outputs. So, even if the output from the 3.3V system could be offset, there would be little or no margin for noise or component tolerance. What is needed is a circuit that offsets the outputs and increases the difference between the high and low output voltages.

Figure 7-1: Diode Offset



When output voltage specifications are determined, it is done assuming that the output is driving a load between the output and ground for the high output, and a load between 3.3V and the output for the low output. If the load for the high threshold is actually between the output and 3.3V, then the output voltage is actually much higher as the load resistor is the mechanism that is pulling the output up, instead of the output transistor.

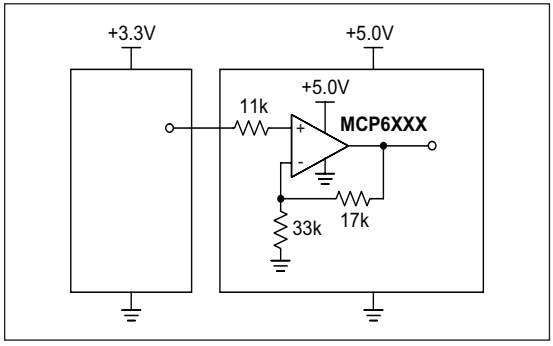
If we create a diode offset circuit (see Figure 7-1), the output low voltage is increased by the forward voltage of the diode D1, typically 0.7V, creating a low voltage at the 5V CMOS input of 1.1V to 1.2V. This is well within the low threshold input voltage for the 5V CMOS input. The output high voltage is set by the pull-up resistor and diode D2, tied to the 3.3V supply. This puts the output high voltage at approximately 0.7V above the 3.3V supply, or 4.0 to 4.1V, which is well above the 3.5V threshold for the 5V CMOS input.

Note: For the circuit to work properly, the pull-up resistor must be significantly smaller than the input resistance of the 5V CMOS input, to prevent a reduction in the output voltage due to a resistor divider effect at the input. The pull-up resistor must also be large enough to keep the output current loading on the 3.3V output within the specification of the device.

TIP #14 3.3V → 5V Analog Gain Block

To scale analog voltage up when going from 3.3V supply to 5V supply. The 33 kΩ and 17 kΩ set the op amp gain so that the full scale range is used in both sides. The 11 kΩ resistor limits current back to the 3.3V circuitry.

Figure 14-1: Analog Gain Block



TIP #15 3.3V → 5V Analog Offset Block

Offsetting an analog voltage for translation between 3.3V and 5V.

Shift an analog voltage from 3.3V supply to 5V supply. The 147 kΩ and 30.1 kΩ resistors on the top right and the +5V supply voltage are equivalent to a 0.85V voltage source in series with a 25 kΩ resistor. This equivalent 25 kΩ resistance, the three 25 kΩ resistors, and the op amp form a difference amplifier with a gain of 1 V/V. The 0.85V equivalent voltage source shifts any signal seen at the input up by the same amount; signals centered at 3.3V/2 = 1.65V will also be centered at 5.0V/2 = 2.50V. The top left resistor limits current from the 5V circuitry.

Figure 15-1: Analog Offset Block