Introduction

Cost reduction has always been a factor considered when designing a control system, especially for motor control applications. In DC motor control, sensors such as optical encoders, Hall effect sensors and current sense transformers are used to identify speed and position, which increases the total number of components, thus the overall system cost. To eliminate the use of such sensors, a drive technique such as sensorless motor control that uses motor output voltage and current feedback is commonly used in motor control drive.

Aside from the economic factor, the advantages of sensorless techniques include decreased maintenance, a smaller number of fragile connections and easier miniaturization of the system. However, one of the challenges with the implementation of sensorless techniques is software complexity, which can be lessened by the MCU peripherals by handling other system processes.

This project aims to provide an alternative solution to the traditional and expensive methods of using sensors to obtain and identify the motor position in precision control applications. Examples of such applications can be found in car seat adjustments, mirror control, power windows and car flaps. Using PIC® MCUs and Core Independent Peripherals (CIPs), the ripple counting technique based on the current drawn by the motor, can be used. These CIPs are utilized to drive the motor with minimum CPU intervention.

This application note describes three different Brushed DC (BDC) motor ripple counting solutions. Each solution has utilized a particular PIC MCU device family. The first solution is a low-complexity position control for a single BDC motor using a low-cost PIC18FXXQ10 device family, and external current filtering and conditioning components. The second solution is a low-cost, single-chip capable of controlling two motors nonsynchronously using the PIC16F177X device family. The third solution is a dual-simultaneous BDC motor control using the PIC16F188XX device family. The BDC motor used in testing these solutions is NIDEC COPAL MG16B-240-AB-00.

The three solutions described have the following key features:

- Full-Bridge Motor Control
- Bidirectional Motor Movement
- 5°-180° Configurable Angle Increment
- Motor Stall Detection and Recovery
- Overcurrent Detection for PIC16F177X
Figure 1 shows the block diagram of the ripple counting system based on PIC18FXXQ10. The PIC18FXXQ10 devices are inexpensive, robust and general purpose, making them suitable for this application. This solution uses CIPs to control the motor.

The CIPs used in the design are: Complementary Waveform Generator (CWG), Hardware Limit Timer (HLT) and Configurable Logic Cell (CLC). Integrating CIPs with other on-chip peripherals such as Pulse-Width Modulation (PWM), Analog-to-Digital Converter (ADC), Capture, Compare and PWM (CCP) and timers give the system the capacity to be fully functional.
# Table of Contents

Introduction.............................................................................................................................................. 1

1. Sensorless Method of BDC Motor Control.......................................................................................... 5
   1.1. Brushed DC Motor Current Ripple.................................................................................................. 6
   1.2. PWM Effects on the Ripple............................................................................................................. 9

2. Ripple to Pulse Conversion.................................................................................................................... 10
   2.1. Current Sensing Stage.................................................................................................................... 11
   2.2. Signal Conditioning Stage A.......................................................................................................... 12
   2.3. Signal Conditioning Stage B.......................................................................................................... 13
   2.4. Comparator Stage.......................................................................................................................... 16

3. Ripple Counting Solutions.................................................................................................................... 17
   3.1. PIC18FXXQ10 Solution.................................................................................................................... 17
   3.2. PIC16F177X Solution....................................................................................................................... 22
   3.3. PIC16F188XX Solution.................................................................................................................... 27

4. Motor Safety Features.......................................................................................................................... 31
   4.1. Motor Stall Detection..................................................................................................................... 31
   4.2. Overcurrent Detection................................................................................................................... 32

5. Ripple Counting Performance................................................................................................................ 33

6. Conclusion.......................................................................................................................................... 36

7. Appendix A: Circuit Schematic............................................................................................................ 37

8. Appendix B: MPLAB® Code Configuration (MCC) Peripheral Initialization................................. 40
   8.1. PIC18FXXQ10 Solution MCC Initialization.................................................................................... 40
   8.2. PIC16F177X Solution MCC Initialization....................................................................................... 42
   8.3. PIC16F188XX Solution MCC Initialization.................................................................................... 44

9. Appendix C: Source Code Listing........................................................................................................ 46

The Microchip Web Site............................................................................................................................ 47

Customer Change Notification Service.................................................................................................. 47

Customer Support.................................................................................................................................. 47

Microchip Devices Code Protection Feature.......................................................................................... 47

Legal Notice............................................................................................................................................ 48

Trademarks............................................................................................................................................. 48

Quality Management System Certified by DNV..................................................................................... 49
1. Sensorless Method of BDC Motor Control

The sensorless motor control is a type of drive technique that does not rely on sensors to determine the actual motor position or speed. It determines the position of the motor based on electrical parameters such as Back-EMF or motor current feedback, which is present during the motor run-time.

The current feedback is used for circuit protection, motor speed estimation or position control. The motor current can be monitored by placing a current sense resistor in a series with the motor. Using a current sense amplifier tapped on the resistor terminals, the current detected is converted to a measurable voltage that acts as a position or speed feedback signal to a closed-loop motor control. In this application, the periodic variations in current feedback, due to BDC motor electro-mechanical construction, are observed.

The displacement taken by the motor rotation can be identified by counting the number of those periodic variations during motor operation. The method of using motor current feedback to determine motor position and speed is referred to as the ripple counting technique and will be discussed and utilized in this application note. The periodic variations in a current feedback are shown in Figure 1-1.

Figure 1-1. Periodic Variation in a Current Feedback

Figure 1-2. Current Sense Configuration

(a) HIGH-SIDE CURRENT SENSE  (b) LOW-SIDE CURRENT SENSE  (c) BRIDGE-ORIENTED CURRENT SENSE
The ripple counting technique primarily depends on the motor current feedback to implement a sensorless drive. Different current sense configurations based on sensing resistor placement can be used to monitor current feedback and are shown in Figure 1-2.

High-side current sensing and bridge-oriented current sensing shown in Figure 1-2 (b) and Figure 1-2 (c) respectively, are commonly used for functional safety requirements. Though proven effective, these configurations require a high-end current sense amplifier with high Common-mode voltage and more complex circuitry, which is not ideal for this application.

The current sensing configuration where the current sense resistor is placed between the motor and the ground, as shown in Figure 1-2 (a), is referred to as low-side current sensing. It has the simplest and most cost-effective implementation of the sensing configurations based on amplifier selection and circuit complexity. The average voltage in the sensing resistor is near zero since it is ground referenced. It does not require high-input Common-mode voltage, making it easier to select devices for its design and implementation. It is also immune to voltage spikes or surges, which makes it suitable for high-voltage motor applications.

Low-side current sensing has its own limitations. The placement of the current sense resistor causes the load not to be ground referenced. The resistance introduced by the sensing elevates the low side of the load several millivolts above the ground. Without ground reference, the short circuit between load and ground cannot be detected. Despite this disadvantage, it is still a good choice for the ripple counting application because its simplicity and load are not required to be ground referenced. The current sensing method will be discussed later in the subsequent sections.

1.1 Brushed DC Motor Current Ripple

There are several types of ripple in a spinning motor. They can be classified as torque ripple, speed ripple and current ripple. The torque ripple is characterized by undesirable variation of torque production during shaft revolution, preventing smooth motor rotation and is observable mostly in Low-Frequency mode. The speed ripples, also known as speed fluctuations, are induced by parasitic torque pulsations that vary periodically with rotor position.

The current ripple is produced during motor rotation and defined as the low-amplitude current alternation, riding on the DC voltage supply. This current ripple is a periodic variation in current created by the rotor movement when it connects and disconnects rotor coils to the power supply via the brushes. The produced current ripple gives an insight about the commutation process happening in a BDC motor. Figure 1-3 shows how a three-commutator segment motor rotates and how the rotation produces current ripples or commutator spikes.
When the BDC motor is turning, the brushes short-out adjacent commutator segments that result in a current circulating between the segments. This causes the Back-EMF of the commutator segments to be shorted out, thus producing short duration high-voltage pulses on the motor terminals. The high-voltage pulses lower the impedance of the armature, thus causing the overall current to rise and generate a
current ripple. The periodic shifting of a brush from one coil to another produces a periodic current ripple. Also, it can be observed that the ripple waveforms are different from each other.

As shown in Figure 1-3, when the signal coming from a rotating motor is displayed on a measuring instrument, the DC component will be immediately observed. The AC component or current ripple will only be seen when this DC component is closely examined. The rotating direction of the DC motor in Figure 1-3 is assumed as counterclockwise as indicated by the black arrow inside Figure 1-3 (a). Brush 1 is connected to a positive supply and Brush 2 is connected to a negative supply. The branch circuit 1, which is composed of Coil 1 and Coil 2, has current flow in a clockwise direction from Brush 1 to Brush 2 as indicated by the red arrow. Meanwhile, the branch circuit 2, which is composed of Coil 3, has a counterclockwise current flow from Brush 1 to Brush 2. The flowing current occurred on Branch 1 is defined as $i_1$, while the flowing current on Branch 2 is defined as $i_2$. The total current of the DC motor follows the Kirchhoff's current law, $i = i_1 + i_2$.

At the moment when Coil 1 aligns with Brush 1 as shown in Figure 1-3 (b), it will make the voltage across Branch 1 decrease because the branch is only composed of Coil 2. Since the supply voltage across the two brushes is always kept constant, the shorting out of Coil 1 will suddenly increase the current value in Branch 1, while the current in Branch 2 stays the same. The effect will be the total current flowing into the DC motor, which will rapidly increase.

After the sudden shorting out, the next event is shown in Figure 1-3-c). Branch 1 will only be composed of Coil 2 while Branch 2 will be composed of Coil 1 and Coil 3. The current flow direction in Coil 2 stays the same as well as the voltage across it. Meanwhile, the current value in Branch 2 abruptly decreases when the process of commutation ends. In this manner, the current ripple is produced as shown in the lower left corner of Figure 1-3, with a single ripple waveform subdivided into sections to reflect what is happening during the commutation process from Figure 1-3 (a) to (c).

The succeeding ripples are created by the same process of coil and brush contact. Note that each coil must touch the two brushes to complete a rotation. Furthermore, every coil and brush contact produces a distinct ripple waveform. The reason behind this is that coils are constructed differently, and the brushes of the motor are also different from each other.

The number of brushes and commutator poles are fixed in a specific motor. This is why the number of ripples in a complete rotation is constant. The ripple repeats in every rotation and the number of ripples per revolution can be calculated using the equation below.

**Equation 1-1. Number of Ripples per Revolution**

$$No.\text{of ripples}_{Rev} = \text{brush} \times \text{commutator segment}$$

When dealing with a geared motor, the equation above will be multiplied by the gear ratio to get the number of ripples per revolution. The equation below shows the calculation of ripple in a geared motor.

**Equation 1-2. Number of Ripples per Revolution Using Geared Motor**

$$No.\text{of ripples}_{Rev} = \text{brush} \times \text{commutator segment} \times \text{gear ratio}$$

Knowing the number of ripples in a single rotation is important to determine the total number of ripples needed to rotate a certain degree of an angle. Applying Equation 1-1 in the example in Figure 1-3 can be used to calculate that six ripples are produced in a single revolution of three-commutator poles with two brushes. The current ripples need to be filtered, conditioned and converted to logic level series of rising and falling edges in order to be countable by an MCU.
### 1.2 PWM Effects on the Ripple

The simplest way to control the speed of a BDC motor is by controlling its supply voltage. The higher the voltage, the higher the resulting speed of the motor. The PWM on the low-side MOSFET of the full-bridge driver is used in many DC motor speed control applications. The ratio of ‘ON’ and ‘OFF’ time determines the motor speed.

Since the current ripple is characterized by a low amplitude, high-frequency periodic signal, any modulation that periodically changes the driving voltage causes the system not to clearly recognize the ripple or lower its signal integrity. Figure 1-4 shows the ripple produced by different PWM speed modulations. The PWM speed modulation affects the visibility of the ripple depending on the degree of modulation. Since this solution focuses on precision control applications, the speed modulation is set at 100% modulation.

**Figure 1-4. Ripples Produced at Different PWM Duty Cycles**

![Ripples Produced at Different PWM Duty Cycles](https://via.placeholder.com/150)

- Ripples produced at 80% PWM
- Ripples produced at 90% PWM
- Ripples produced at 100% PWM
2. **Ripple to Pulse Conversion**

To effectively count and convert current ripple into PIC MCU readable pulses, the current flowing through the BDC motor undergoes various signal conditioning and filtering stages. Chronologically, the stages are: current sensing stage, band-pass filter stage, differential amplifier stage and the comparator stage.

Figure 2-1 illustrates the output signals of each ripple conversion stage from the raw current signal up to the digital pulses that will be used as an input to PIC MCUs.

**Figure 2-1. Ripple Conversion Stages Output Signals**

![Diagram of ripple conversion stages]

Signal amplification is needed in the first three stages since the current signal is too small to be detected by the comparator. Depending on the solution, the signal conditioning stages and comparator stage can be external or internal with respect to the PIC MCU family used.

All passive components and amplifiers are powered by a 5V supply. For the external-based filtering solutions (PIC16F188XX and PIC18FXXQ10), MCP6024, a quad in the package operational amplifier (op amp) integrated circuit, is used. It features a wide bandwidth (10 MHz) and low noise feature that are suitable for small signal processes. For the comparator stage, an MCP6541, a push-pull output sub-microamp comparator is used. While PIC16F177X promotes a cheap single-chip solution, PIC18FXXQ10 promotes a simple and flexible solution where the signal conditioning and comparator stage are external.

The output of the current sense is fed to the microcontroller for the signal conditioning. The internal op amp peripheral has a 2 MHz gain bandwidth product and an open loop voltage gain of 90 db. It also operates from 0V to $V_{DD}$, which is 5V in this application. By utilizing these peripherals, the Bill of Materials (BOM) costs for hardware designs, size and system noise can be reduced. The external current sensing stage was designed using MCP6022. For a more detailed schematic diagram, refer to 7. Appendix A: Circuit Schematic.
2.1 Current Sensing Stage

Since the application requires a simple and cost-effective implementation, the low-side current sense configuration is used in this application. The current through the bridge driver is detected by tapping a differential amplifier across the terminals of the current sense resistor R1, which is placed before the ground as shown in Figure 2-2.

The current sense resistor at the low-side of the driver allows ripple conversion to operate bidirectionally since the current flow direction will not change when triggered. If the current sense resistor is placed within the load terminals of a bridge configuration, a more complex and more expensive circuit configuration is required.

Figure 2-2. Current Sense Amplifier Configuration

The value of R1 depends on the maximum current that could flow throughout the bridge. In this design, the current is not expected to exceed 5A. The value of the current sense resistor can be calculated using Equation 2-1. Another consideration when picking the right resistor is its power rating, which also determines the package size. For this application, a 2W resistor with a 2512 package size is chosen.

Equation 2-1. Current Sense Resistor Value

\[ R_{\text{SENSE}} = \frac{V_{\text{DD}} - V_{\text{REF}}}{I_{\text{Max}} \cdot A_v} \]
Aside from the current sense amplifier, the configuration for the current sense voltage biasing is needed. Since all stages use a single supply operational amplifier, this is more susceptible to saturation during the negative signal swing. It might cause clipping that will affect the original signal. Figure 2-3 shows a stable voltage bias, with \( V_{\text{REF}} \) established by using a unity gain configuration op amp fed by a voltage divider. To obtain the maximum voltage signal swing allowed before saturation, the \( V_{\text{REF}} \) is centered between the voltage source and the ground. \( V_{\text{REF}} \) is set at 2.5V on all succeeding stages.

The current that flows through the current sense resistor is so small that it requires high-gain amplification. Because of this, the gain is set to 110V/V. The \( V_{\text{OUT}} \) can be solved using the equation below.

**Equation 2-2. Current Sense Resistor Value**

\[
V_{\text{OUT}} = V_{\text{REF}} + (I \times R_{\text{SENSE}} \times A_v)
\]

### 2.2 Signal Conditioning Stage A

After the current sensing stage, the signal will undergo in a two-stage signal conditioning. The first signal conditioning stage is implemented using an active band-pass filter. Figure 2-4 shows the band-pass filter configuration used in this application. This stage is used to eliminate the DC component of the signal and the added external noise from the previous stage. It is also designed to allow frequency within approximately 100 Hz to 1500 Hz, which is based on the ripple frequency range. The ripple frequency can be identified by the number of commutator poles, gear ratio and the rated speed of the motor as shown in Equation 2-3. Different rated speeds of different motors affect the frequency range to be set.

**Equation 2-4** shows the gain and cut-off frequencies calculation of the band-pass filter. The desired gain is calculated by \( R_2 \) and \( R_3 \) depending on the maximum voltage swing before the op amp saturation. \( C_2 \) and \( C_3 \) capacitors can be calculated using the resistor values selected and frequency range. An inverting amplifier topology is used to retain the biasing from the previous stage to the next stage of ripple conversion.
Equation 2-3. Ripple Frequency

\[ f_{\text{ripple}} = \frac{\text{RPM} \times \text{GearRatio} \times \text{no.of.poles}}{30} \]

Ex: 960 Hz = \( \frac{40 \times \text{rpm} \times 240 \times 3}{30} \)

Equation 2-4. Band-Pass High and Low Cut-Off Frequency and Gain

\[ f_{c1} = \frac{1}{2\pi R_2 C_2}; \quad f_{c2} = \frac{1}{2\pi R_3 C_3}; \quad G_1 = \frac{R_3}{R_2} \]

Ex: 106.39 Hz = \( \frac{1}{2\pi (220k\Omega) (6.8nF)} \); 1560.34 Hz = \( \frac{1}{2\pi (1.5M\Omega) (68pF)} \); 6.8 = \( \frac{1.5M\Omega}{220k\Omega} \)

2.3 Signal Conditioning Stage B

The output of the active band-pass filter provides a clean amplified AC signal that is biased along the reference voltage. This allows the comparator to detect ripples and convert it to pulse with a frequency equal to the current ripple frequency. Although the output is a clean amplified AC signal, an initial current spike that is further discussed in this section has not been filtered out during this process. This condition leads to the addition of another signal conditioning stage while maintaining the proper biasing for the comparator.
Inrush Current

During motor start-up, motor windings behave as a low-resistance closed circuit causing the system to draw a large amount of current. This large amount of current during motor start-up is referred as the inrush current. It refers to the electrical current usually flowing through the motor in fractions of a second after the motor’s power is turned on. In this brief time interval, the current rapidly increases and decreases to its steady state as the motor begins to rotate.

Figure 2-5 illustrates the current produced during motor start-up. This current spike is too large and too slow not to be filtered out by the previous stage. In addition, the high pass part of the band-pass filter has been set to 100 Hz at the previous stage. This configuration produces a fairly large time constant that prohibits the DC signal components to stabilize quickly.

Since the motor is already running and generating ripple at this point, this causes the comparator to miss those start-up ripples. The blue and green signals in Figure 2-5 represent the current sense stage and band-pass filter stage, respectively. Failure to detect and convert the start-up ripple current into comparator pulses will result in a missed pulse and inaccuracy on the ripple count. The problem about the start-up current ripple is addressed by using a differential amplifier stage for the second signal conditioning stage.
At this stage, the output of the band-pass filter is fed to both the positive and negative inputs of a differential amplifier with a capacitor (C9) and tapped in the positive input as shown in Figure 2-6. This capacitor acts as a low-pass filter that introduces a small amount of delay with respect to the original signal. This delay highly affects the output of the differential amplifier since the common voltage between inputs are being negated. The effect of the high-current spike is eliminated while retaining the ripple frequency component. The signal is amplified with a differential gain of approximately 68V/V for wide signal swing and rebiased at 2.5V for the comparator stage.

Figure 2-7 illustrates the difference between the comparator outputs when the differential amplifier output is used or not. If the band-pass filter is used as input for the comparator, the initial ripples might not be converted to pulses because of the inrush current. The greater and faster the change in the amplitude during the start-up, the greater the possibility of a missed pulse. The differential amplifier solves this issue.
2.4 Comparator Stage

This stage follows an inverting hysteresis topology that generates an output of either 0 or 5V signal pulses. These output pulses are used as an input to the counting algorithm of the firmware. The comparator hysteresis function moves the threshold level up or down with respect to the input signal. This prevents the output pulse from oscillating and avoids switching noise during level transition caused by small amounts of parasitic feedback.

The reference voltage used on the comparator depends on the behavior of the signal. Since different motors have different current rating and brush construction, the reference voltage signal varies depending on the motor used. Typically, the reference is set near the center, between the voltage source and the ground. The voltage reference can be configured using a voltage divider tapped to a feedback resistor. This determines the high and low voltage threshold the comparator will accept. Figure 2-8 shows the comparator circuit with hysteresis using MCP6541. For the PIC16F177X solutions, the internal comparator peripheral is utilized.

Figure 2-8. Comparator Stage Schematic

The value of the voltage thresholds depends on the values of the resistor applied. For the calculation of $V_{TL}$ and $V_{TH}$, refer to Equation 2-5. R18 and R19 are set similarly since $V_{TL}$ and $V_{TH}$ are centered at $V_{REF}$ as well.

Equation 2-5. Voltage Threshold Range

$$V_{TL} = \left( \frac{R_{19}R_{20}}{R_{19} + R_{20}} \right) \left( \frac{V_{CC}}{R_{19}R_{20} + R_{18}} \right) + \left( \frac{R_{19}R_{18}}{R_{19} + R_{18}} \right) \left( \frac{V_{OL}}{R_{19}R_{18} + R_{20}} \right)$$

$$V_{TH} = \left( \frac{R_{19}R_{20}}{R_{19} + R_{20}} \right) \left( \frac{V_{CC}}{R_{19}R_{20} + R_{18}} \right) + \left( \frac{R_{19}R_{20}}{R_{19} + R_{18}} \right) \left( \frac{V_{HL}}{R_{19}R_{18} + R_{20}} \right)$$
### Ripple Counting Solutions

The previous section introduced the method of converting current ripple into readable pulses by either the external comparator or the PIC MCU’s comparator module. The next step is to count these converted ripples and to learn how these ripples are processed using the PIC MCU. Three ripple counting solutions using PIC MCU families will be discussed thoroughly in this section. Each of these solutions offer different pros and cons; the user will decide which solution best fits their design.

#### 3.1 PIC18FXXQ10 Solution

This solution gives flexibility to the user to choose the desired current filtering and conditioning components. The value of the external components can be easily adjusted to suit the motor specifications for the accurate conversion of ripples to digital pulses. The PIC MCU is used to count the digital pulses and translate the number of pulses into equivalent motor positions. Furthermore, the use of a PIC MCU in a solution gives better performance due to added intelligence in the system.

#### 3.1.1 PIC18FXXQ10 Ripple Counting Implementation

The counting implementation is achieved through the integration of firmware and peripherals. The firmware commands to initiate the drive and to start the counting of pulses. It also facilitates the recording and storage of the motor’s position in the memory. On the other hand, the peripherals carry out the counting of pulses and their comparison to the expected number of ripples that are initialized in the beginning.

**Figure 3-1. PIC18FXXQ10 Control Diagram**

#### 3.1.2 Input References

*Figure 3-1* shows the complete signal flow in the system. The solution is designed with an analog potentiometer for selecting the desired angle and four buttons for the motor direction control as an input to the system. Two buttons were assigned per direction. In the forward direction, one button is for a normal incremental movement and another button is for moving to the endpoint, while in the reverse direction, there is a button for normal movement and another one for going back to home position. The
range of the incremental angle that the system can handle effectively is from 5° to 180° with 5° increment per potentiometer adjustment. The input angle is processed by the firmware and undergoes a series of calculations using equations.

Equations were developed for calculating the needed parameters in the firmware depending on the given motor specification. Motors may differ in the number of commutator poles and gear ratio. Equation 3-1 is developed to determine the number of ripples per degree in the movement of a geared motor. Meanwhile, Equation 3-2 is used to calculate the expected number of ripples to rotate a certain degree of an angle. Lastly, the compare load value in Equation 3-3 computes the value to be loaded into the CCP peripheral where the TIMER1 register data will be continuously compared to determine if the target position has been reached.

Equation 3-1. Ripple Count per Angle

\[ \text{rippleCountPerAngle} = \frac{\text{Gear Ratio} \times \text{Commutator Segment}}{180°} \]

Equation 3-2. Expected Ripple Count

\[ \text{expectedRippleCount} = \text{angleDesired} \times \text{rippleCountPerAngle} \]

Equation 3-3. Compare Load Value

\[ \text{compareLoadValue} = ((\text{expectedRippleCount} - \text{CALIBRATION VALUE}) >> 1) + \text{INITIAL TIMER VALUE} \]

3.1.3 Motor Drive Signal

The PWM generates a full-bridge signal with 100% duty cycle to supply the CWG. The PWM is set to 100% duty cycle, since this application does not require the modulation of the motor speed. The PWM signal is directed by the CWG peripheral to its specific channels depending on the chosen CWG mode. The CWG has the capacity to drive the motor in forward and reverse directions, which enables the motor to be driven bidirectionally. The signal coming from the CWG controls the full-bridge driver circuit, which accomplishes the actual driving of the motor. For the complete discussion of this peripheral operation, refer to TB3118: Complementary Waveform Generator. Aside from driving using CWG, if the device has no CWG, the user can also use the CLC peripheral for creating a combination of signals from the PWM output to drive the motor bidirectionally. When the motor is already spinning, it is the right time to control its position and dictate when to stop it. This will be discussed further in the next section.

3.1.4 Counting the Current Ripple

The counting implementation is achieved through the integration of firmware and peripherals. The firmware commands to initiate the drive and to start the counting of pulses. The conditioned ripple enters the PIC MCU and is fed to the TIMER1 peripheral.

3.1.5 Timer1 Configuration

The TIMER1 peripheral is a 16-bit timer/counter that can be configured in different modes. To learn more about the function of the TIMER1 peripheral refer to the discussion of TIMER1 in the device data sheet. This application used the TIMER1 module in Gate Toggle mode. The TIMER1 module is set to capture data every high pulse of the signal. The diagram in Figure 3-2 illustrates the operation of TIMER1 in this application.
The input to the TMR1 gate and to the TIMER1 clock source is the same because the pulses counted will come from the clock source and the indicator that needs to be captured is rendered by the pulses coming from the gate. In the first full-cycle of gate signal, the TxGVAL value goes high and low to the succeeding full-cycle of the ripple signal. If this happens alternately, the two ripples of the original signal will be equivalent only to a full cycle of TxGVAL.

Only when TxGVAL is high the TIMER1 increments its register value, and whenever TxGVAL is low the TIMER1 register holds the value accumulated previously. In this way, two ripples are reduced into a single pulse in which the TIMER1 register value counts. That is the reason why during the setting of compareLoadValue in Equation 3-3 it is shifted once to the right to resemble that the expected number of ripples is divided into half. The initial value loaded to the TMR1 register is equivalent to half of its maximum for a wider range of the desired angle that the TIMER1 can accommodate. The TIMER1 counts the number of ripples, but without any position indicator it will not be known if the desired position has been reached. The CCP peripheral serves as an indicator when the position is reached.

### 3.1.6 CCP Operation

The Capture/Compare/PWM module is a peripheral that allows the user to time and control different events, and to generate PWM signals. This application uses this peripheral in Compare mode, which allows the user to trigger an event after a predetermined amount of time has expired. TIMER1 is the chosen timer to adjust the Compare trip point. This mode has a Compare register where the user can put the value to which the TIMER1 register data will be compared. For the TIMER1 and CCP to function properly, the TIMER1 register and CCP register must have the same initial value. In CCP, this initial value will be added to the expected ripple count to establish the correct reference count.

While retrieving the actual ripple count, the period timer value equal to the initial value is added to the content of the TIMER1 register to get the actual number of ripples that the motor rotated. In Equation 3-3, the variable compareLoadValue has an added constant INITIAL_TIMER_VALUE to indicate that the TIMER1 register has an initial value from where it will start counting. The CALIBRATION_VALUE is a constant that is being deducted from the total count of expected ripples to compensate for the excess movement of the motor due to inertia. It is dependent on the type of motor used and is obtained by counting the excess ripples of the motor running in its rated voltage. These excess ripples can no longer

---

**Figure 3-2. Timer1 Diagram**

<table>
<thead>
<tr>
<th>TMR1GE</th>
<th>T1GPOL</th>
<th>T1GTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR1 Gate Input/ Ripple Pulse</td>
<td>T1CKI/ Ripple Pulse</td>
<td>T1GVAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMR1 Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

- **TMR1GE**: TMR1 Gate Enable
- **T1GPol**: T1 Gate Polarity
- **T1GTm**: T1 Gate Trigger
- **TMR1 Register Value**: N, N + 1, N + 2, N + 3

© 2019 Microchip Technology Inc.
be converted into digital pulses because of their very small amplitude. When the TIMER1 value and CCP value became equal, the CCP generates an interrupt that triggers the motor to stop and retrieve the ripple count.

3.1.7 Getting the Ripple Count

The actual ripple count in Equation 3-4 comes from the TIMER1 register added to the period timer value; subsequently, the combined value is shifted once to the left. This equation reverses the previous operation made in the setting of the expected number of ripples in Equation 3-3. This operation is needed in order to properly count the ripples and compensate for the limitation of the Gate Toggle mode.

Equation 3-4. Actual Ripple Count

\[
\text{actualRippleCount} = ((\text{TMR1\_ReadTimer()} + \text{PERIOD\_TIMER1\_VALUE}) << 1)
\]

Equation 3-5. Angle Turned

\[
\text{angleTurned} = (\text{actualRippleCount} / \text{rippleCountPerAngle})
\]

The actual angle that the motor turned is calculated using Equation 3-5. All computations are performed by the firmware. After getting all the values, EUSART is used to display the actual number of ripples and actual angle. It is important that the position is displayed for the user to know if the desired angle has been rotated.

3.1.8 Braking Mechanism

Just as starting the motor is important, stopping the motor in this application has been very critical, too. The H-Bridge driver must be considered to avoid current shoot-through, which may take place if the two switches on the same side turned on simultaneously. The braking mechanism of the motor is accomplished through the firmware. The CWG is configured to Steer mode and a specific value is written to CWG1STR to turn on the full-bridge driver low-side switches.

In this manner, the motor is shorted to ground by turning off the two high-side switches and turning on the two low-side switches. When the motor terminals are shorted to ground, the BEMF across the motor will cause the current to flow to the lower potential which is the ground that immediately drains the energy across the motor. This braking mechanism is applicable to both forward and reverse motor drive.

3.1.9 Motor Positioning

There are two functions created for processing the angle: one for Forward mode and another for Reverse mode of the motor. This is necessary for establishing the home and endpoint of the motor’s position. From home, the motor can only move using the Forward mode. The motor may turn several times before reaching the endpoint depending on the incremental angle chosen or the user can press the endpoint button to immediately reach the endpoint. Upon reaching the endpoint, an LED will light up to indicate that the motor has reached the endpoint.

If the user insists on pressing the button that causes forward motion, the motor will not rotate and will stay still. This time the user can use the reverse button to turn backward depending on the desired angle set or home button to go back home. Similar with Forward mode, upon reaching home an indicator LED will light up to signify that the home is already reached and the button that triggers reverse movement will be disabled. After every motor movement, the position will be recorded and stored in the EEPROM address. This EEPROM address is read at the beginning of the next motor movement of the motor’s position in order to be kept updated.
Figure 3-3 shows the firmware flowchart of the solution. The firmware is developed through the aid of the MPLAB® Code Configurator (MCC). The instructions for setting up the peripheral configurations used in this application can be found in 8. Appendix B: MPLAB® Code Configuration (MCC) Peripheral Initialization.

The firmware flow starts with a `SYSTEM_Initialize()` routine. This routine initializes pin configuration, oscillator and all the peripherals used in the application. After system initialization, it is necessary to enable interrupts to address the functions requiring interrupts such as `Stall_ISR()` and `Compare_ISR()`.

The `Compare_ISR()` routine initializes the `Compare_ISR()` as the Compare interrupt handler. `Compare_ISR()` is the function that will be executed whenever compare peripheral generates an interrupt. `StallDetection_Initialize()` routine sets up the protection feature of this application.
When all the initialization is done, the firmware will be in a continuous loop executing the tasks as follows: 

The `ReadMotorPositionFromEEPROM()` routine is responsible for reading the motor’s position which is necessary for the firmware to be kept updated on the motor’s current position before another rotation begins. The next routine is `ReadInput()`, which gets the data from the potentiometer to select the desired angle that the motor will turn and computes the equivalent number of expected ripples for a specific desired angle. Check buttons are composed of four subroutines responsible for monitoring if a particular button is pressed and it initiates the movement in a specific direction. After pressing a button, the motor rotation and counting of pulses will start at the same time and continue rotation until a specific rotational position is reached.

When the position is reached, the `getCountDone` value will be set to ‘1’ (which will be tested in the main loop) and if the condition is satisfied, the program will proceed to `GetAngleTurned()` and `MotorPosition()`. The `GetAngleTurned()` routine obtains the actual ripple count and calculates the corresponding actual angle turned. While `MotorPosition()` routine adds the angle turned to the previous position and stores the combined latest and previous position to the EEPROM. The condition with a `motorStalled` test variable will only be satisfied if the motor is stalled (see the 4.1 Motor Stall Detection section). The position of the motor will be stored through `GetAngleTurned()` and `MotorPosition()` and will proceed to the position recovery through `ResumeMotor()` after a 5s delay. The `ResumeMotor()` routine is responsible for the motor to return to its home position. The loop will be continuously running, executing the tasks mentioned above. Refer to 9. Appendix C: Source Code Listing for the complete source code.

### 3.2 PIC16F177X Solution

This solution uses the integration of analog and digital peripherals of the PIC16F177X device family to create a one-chip solution. The PIC16F177X has a higher cost than all the PIC MCUs used in this application, but it has built-in analog peripherals capable of internally converting the ripple signal to digital pulses. Due to the elimination of the external op amp and comparators, the overall cost of the system decreased. However, by using an internal comparator, the capacity to convert ripple signals into digital pulses is limited by the internal comparator hysteresis. Furthermore, one constraint of this solution is represented by the fixed I/O pins of the analog modules, which means that rerouting the analog signal to other pins is not possible.
The block diagram of the solution based on PIC16F177X is shown in Figure 3-4. The PIC16F177X device family is created to support highly integrated applications. Through the integration of analog and digital peripherals, a single device can drive a single motor at a time. Similar to the PIC18FXXQ10 solution, it also uses CIPs in the design of the system such as: Complementary Output Generator (COG) and Hardware Limit Timer (HLT).

In contrast with the previous solution, the signal filtering and conditioning is done using on-board op amp and comparator peripherals. The use of on-board peripherals decreased the number of external components, hence lowering the cost of the overall system. Furthermore, the use of a single chip to reliably control the position of two motors reduced the system cost. Though this solution can address the control of two motors, it is required that those motors must not operate simultaneously.

### 3.2.1 Internal Op Amp for Signal Filtering and Conditioning

The PIC16F177X family of devices have built-in op amp peripherals. In this application, the internal op amp modules are used on an inverting configuration. The first internal op amp has the same functionality as the first external op amp in the PIC18FXXQ10 solution. It is used for eliminating the signal's DC component and external noises. The second op amp is used in the differential amplifier configuration to ensure that the signal will be detected by the comparator to prevent missed pulses during the motor start-up. After these two stages, the signal is fed to the comparator for digital conversion.
3.2.3 Input References

*Figure 3-5* shows the complete signal flow of the solution based on the PIC16F177X devices. The manner of setting the desired angle and calculation of expected ripples are similar to the first solution. The range of the angle that this system can handle is also from 5-180° with a 5° configurable angle increment. For the direction control, there are four buttons used as an input to the system (two buttons per motor for bidirectional control). This solution utilized all the equations used in PIC18FXXQ10 for getting the ripple count per angle (Equation 3-1), expected ripple count (Equation 3-2) and compare load value (Equation 3-3) that will be loaded into the CCP register. After processing all the inputs, the motor is setup to be driven.

3.2.4 Motor Drive Signal

The drive signal of the motor is provided by the COG peripheral. COG is built in the functionality of CWG with improved performance and additional features. For more information on the operation of the COG peripheral refer to TB3119: Complementary Output Generator. Each motor is driven by a dedicated COGx. The signal coming from the COGx controls the full-bridge driver circuit, which accomplishes the actual driving of the motors. After successfully driving the motor, the BEMF signal is fed to the series of internal operational amplifiers for filtering and conditioning. The processed signal is internally connected to the comparator for ripple conversion.

3.2.5 Counting the Current Ripple

When the ripples are transformed to rectangular pulses, they can already be counted by TIMER1 but there is no comparator option in the TIMER1 clock input source and TIMER1 gate input. The comparator output cannot be connected internally to the TIMER1 input. To address this issue, the comparator output is channeled externally and then fed back to the TIMER1 clock input and TIMER1 gate input through another input pin. The counting of the pulses followed the same principle implemented in the Q10 solution. The TIMER1 and CCP modules have the same configuration with the first solution.

Only a pair of TIMER1 and CCP is used for counting the ripple because the motors will only be driven one at a time. The expected ripple count is initially loaded to CCP, then the TIMER1 started its counting at...
the same time as the motor is driven. When the TIMER1 register value equates the CCP register value, the actual ripple count is retrieved in the firmware using Equation 3-4. Afterwards, the angle turned is computed from the actual ripple count value using Equation 3-5. The values are displayed through EUSART and managed for the home and endpoint establishment. Since motors are treated independently, they also have separate memory addresses for storing the motor’s position. After every measurement, the value is stored in the motor memory address and retrieve at the beginning of every motor movement. The High Endurance Flash (HEF) memory, which is available in the PIC16F177X family, is used for storing the position data.

3.2.6 Braking Mechanism

The braking mechanism used in this solution is similar to the one in the first solution. The COGx is configured to Steer mode and a specific value is written to COGxSTR to turn on the full-bridge driver low-side switches. In this manner, the motor is shorted to ground by turning off the two high-side switches and turning on the two low-side switches. When the motor terminals are shorted to ground, the BEMF across the motor will cause the current to flow to the lower potential, which is the ground that immediately drains the energy across the motor. This braking mechanism is applicable to both forward and reverse motor drive.
Figure 3-6 shows the firmware flow of the solution based on the PIC16F177X devices. The functions used are almost similar to the routines used in the PIC18FXXQ10 solution. But there are some additional routines like\texttt{OvercurrentDetection\_Initialize()}, which initializes the Overcurrent\_ISR as the comparator interrupt handler and will be triggered if an overcurrent occurred in the system. Two separate routines were created for motor positioning\texttt{Motor01Position()} and \texttt{Motor02Position()} to handle the position of two motors individually and save them to their specific HEF address. The test variable \texttt{faultDetected} will only be satisfied if stall or overcurrent occurred. It has similar functionality
with the `motorStall` in the PIC18FXXQ10 solution, but it addresses two Faults. Similar to the first solution, after the Fault detection took place the actual ripple count is retrieved through `GetActualRippleCount()`, stored position depending on what motor rotated through `Motor01Position()` or `Motor02Position()` and returned the motor to home through `ResumeMotor()` after a 5s recovery period. Refer to 9. Appendix C: Source Code Listing for the complete source code.

### 3.3 PIC16F188XX Solution

This solution is created for applications requiring simultaneous motor drives such as car mirrors or mud flaps. This solution is also developed to control the position of a single motor or two motors concurrently. The highlight of this solution is its capacity to drive two motors and control their positions simultaneously.

**Figure 3-7. PIC16F188XX Block Diagram**

The block diagram of the solution based on PIC16F188XX is illustrated in Figure 3-7. The PIC16F188XX family is composed of full-featured devices for general purpose and low-power applications. Similar to the previous solutions, the drive is accomplished through CIP–CWG and the functionality of the whole solution is achieved through the integration of on-chip peripherals and CIPs. The conversion of the ripple signal to digital pulses is accomplished externally similar to the PIC18FXXQ10 implementation.

The pulses from two motors are fed to a dedicated SMTx for counting. The **Signal Measurement Timer (SMT)** peripheral is used for counting the ripple pulses. The use of SMT made the simultaneous position control possible because there will be a dedicated hardware counter for each motor. SMT is a 24-bit counter/timer with advanced clock and gating logic. Refer to 3.3.4 Counting the Current Ripple section to learn how this peripheral is implemented in this application. The succeeding section will discuss the counting implementation of the solution.
3.3.1 PIC16F188XX Ripple Counting Implementation

Figure 3-8. PIC16F188XX Control Diagram

3.3.2 Input References

Figure 3-8 shows the complete signal flow of the solution based on the PIC16F188X devices. The manner of setting the desired angle and calculation of expected ripples are similar to the previous solutions. The range of angle that this system can handle is also from 5-180° with a 5° configurable angle increment. There are four buttons used as input to the system: two buttons were used for direction control (forward and reverse), one of them is used as a motor selector, and the other one is used for setting the angle desired for a specific motor in dual motor drive.

This solution used all the equations in PIC18FXXQ10 for getting the ripple count per angle (Equation 3-1) and the expected ripple count (Equation 3-2). The expected ripple count is initially loaded to the SMTx Period register in which the SMTx Timer register value will be continually compared. The SMT peripheral has an advantage of performing the counting functionality of both CCP and TIMER. It is a core independent peripheral that reduces the burden on the main core. After completing the parameter initialization, the motor is setup to be driven.

3.3.3 Motor Drive Signal

This solution is created to drive motors individually or simultaneously. The integration of the firmware and hardware made it possible to select a specific motor or motors to rotate. Similar to the first solution, the PWM generates a full-bridge signal with 100% duty cycle to supply the specific CWG. The PWM is set to 100% duty cycle, since this application does not require the modulation of motor speed. The PWM signal is directed by the CWGx peripheral to its specific channels depending on the chosen CWGx mode. Each motor has an independent CGWx driver that provides full-bridge drive signal to the driver circuit. The signal coming from the CWGx controls the full-bridge driver circuit, which accomplishes the actual driving of the motor. The feedback signal from the motor is processed to obtain a series of digital pulses that are supplied to the SMTx signal input for counting. The next segment will show how the ripple pulses are handled by SMT peripheral.
3.3.4 Counting the Current Ripple

The main goal of this counting implementation is to count all the ripples until the number of expected ripples is reached. To accomplish this, the SMTx is configured to Counter mode because this mode enables the ripple pulses to be continuously counted without requiring a window signal. The input polarity of the signal is set to rising edge, which means that in every rising edge of the signal the SMTx Timer register value increases.

**Figure 3-9. SMTx Counting Diagram**

![Diagram showing ripple counting](image)

Figure 3-9 shows the diagram that describes how counting is performed by SMT. The ripples are fed to the SMTx signal input. The SMTx Timer register value increments with every rising edge of the signal. When the SMTx period equates the value of the SMTx timer, an interrupt will occur. This match interrupt will trigger the capture of the SMTx timer value that contains the actual ripple count. The interrupt will also trigger the stopping of the motor, which is accomplished using the braking mechanism discussed in the first solution. The actual ripple count will then be converted to the corresponding angle turned using **Equation 3-5**.

After getting the angle turned the data will be handed to the functions managing the position of the motors. The position is stored in the EEPROM after every motor rotation. The stored position will be retrieved before another rotation begins for the motors’ position to be kept updated and monitored.
Figure 3-10 shows the firmware flow of the solution based on the PIC16F188XX devices. The process of initializing the device through the `SYSTEM_Initialize()` routine and enabling interrupts is similar to the PIC18FXXQ10 solution. In the continuous loop, the `CheckMotorSelectorButton()` routine is used to check the button to select the motor that will be driven in the switch-case statements. The `MotorDrive01()` routine drives motor 1, controls its position and saves it to EEPROM, while the `MotorDrive02()` routine drives motor 2, controls its position and saves it to EEPROM as well. The `DualMotorDrive()` routine performs the drive and position control of both motors and stores the position to the motors’ specific addresses. Those are the sequences of the routines that the program has continuously executed. Refer to 9. Appendix C: Source Code Listing for the complete source code.
4. Motor Safety Features

It is of prime importance that the system’s performance will not be adversely affected when it encounters a problem and it must recover after the Fault took place. Two safety features used to secure the motor in the Fault event are discussed below.

4.1 Motor Stall Detection

When the motor is spinning freely, the ripple pulses are continuously being fed back to the system. If the motor halts, the trail of digital pulses will also stop. The peripheral used for detecting the stall is the Hardware Limit Timer (HLT) mode in TIMER2/4/6. The primary function of HLT is to monitor any changes in the periodic pulses, and its controlled mode is set to roll over pulse.

When the signal is absent for resetting the value of HLT, it will generate an interrupt which will be used as a trigger to shut down the COG/CWG depending on the device family. The diagram in Figure 4-1 shows the HLT implementation in this application.

![Figure 4-1. HLT Implementation](image)

The comparator output, which produces the digital pulses serves as an external reset source to HLT. The desired period of HLT is set depending on the rated speed of the motor. In this application, the motor used has a rated speed of 40 RPM. This speed expressed in RPM must be converted to a value that will be loaded to HLT period register which is done using the equation below.

Equation 4-1. RPM to T4PR Conversion

\[
T4PR > \frac{RPM \times TMR4ClockSource}{60 \times \text{PrescalerValue} \times \text{PostscalerValue}} \\
= \frac{40 \times \text{MFINTOSC}}{60 \times 128 \times 1} > \frac{40 \times 31250}{60 \times 128} > 162 \text{ or } 0xA2
\]

Note: T4PR register value must be greater than 162 or 0xA2.
4.2 Overcurrent Detection

Overcurrent is a condition where there is an excessive current flow in the motor windings, hence exceeding the rated current in which the motor is designed to carry safely and efficiently. It can be caused by excessive torque loading, short-circuit conductors or an excessive voltage supply. To prevent motor damage, there must be a way to detect Fault when overcurrent occurs. The implementation of this protection feature using PIC16F177X is discussed below.

Figure 4-2. PIC16F177X Overcurrent Implementation

In Figure 4-2, the input is taken from the conditioned signal after the operational amplifier stages. The conditioned signal is fed to the inverting input of the comparator and compared to a reference voltage. This reference voltage results from the product of the current shunt resistance, the maximum allowable motor stall current and gain of the amplifiers used. The reference voltage is provided by the FVR. When overcurrent occurs, the conditioned signal voltage will exceed the reference voltage and the comparator will generate a signal to trigger the COG auto-shutdown feature.
5. **Ripple Counting Performance**

The results below show the waveforms, desired angle, calculated expected ripple count, actual ripple count, angle turned and percentage error at different angles tested using NIDEC COPAL MG16B-240-AB-00 with the gear ratio of 256. In the motor data sheet, it is indicated that the gear ratio is 240. But the results were inaccurate using this number, so it was carefully inspected and the real gear ratio came out to be 256.

Figure 5-1. 5° Angular Movement

\[
\text{angleDesired} = 5 \quad \text{Expected Ripple Count} = 20 \\
\text{actual Ripple Count} = 21 \quad \text{AngleTurned} = 5
\]

\[
\text{Percentage Error} = \left| \frac{\text{Actual Ripple Count} - \text{Expected Ripple Count}}{\text{Expected Ripple Count}} \right| \times 100 \% \\
= \left| \frac{21-20}{20} \right| \times 100 \% = 5\% \text{ error}
\]
Figure 5-2. 60° Angular Movement

\[
\text{Percentage Error} = \left| \frac{\text{Actual Ripple Count} - \text{Expected Ripple Count}}{\text{Expected Ripple Count}} \right| \times 100 \%
\]

\[
= \left| \frac{257 - 256}{256} \right| \times 100 \% = 0.3906\% \text{ error}
\]

Figure 5-3. 90° Angular Movement

\[
\text{Percentage Error} = \left| \frac{\text{Actual Ripple Count} - \text{Expected Ripple Count}}{\text{Expected Ripple Count}} \right| \times 100 \%
\]

\[
= \left| \frac{385 - 384}{384} \right| \times 100 \% = 0.2604\% \text{ error}
\]
The difference between the motor operation with and without load has also been observed. Figure 5-4 shows the signal difference between those conditions. At the same 12V of motor voltage, the ripple current is greater when there is a load causing a higher voltage swing as the ripple frequency decreases. The higher voltage swing makes the comparator easily recognize the high/low voltage transition.

Figure 5-4. Ripple Conversion with/without Loading Condition
6. Conclusion
This application note discusses how the ripples were successfully converted to pulses countable by the Microchip PIC devices. These pulses are counted and processed to estimate the position of the motor. The highly-flexible solution is achieved through the PIC18FXXQ10 family, the one chip solution is attained through the PIC16F177X family and the dual motor simultaneous BDC motor control is achieved through the PIC16F188XX family. The implementation of ripple counting in BDC positioning has considerably lowered the total system cost and the motor is protected by safety features in case of Fault event.
7. **Appendix A: Circuit Schematic**

Figure 7-1. PIC18FXXQ10 Schematic Diagram
Figure 7-2. PIC16F177X Schematic Diagram
8. Appendix B: MPLAB® Code Configuration (MCC) Peripheral Initialization

MPLAB® Code Configurator is an easy-to-use plugin tool for MPLAB® X IDE that generates codes for controlling the peripherals of Microchip microcontrollers, based on the settings made in its Graphical User Interface (GUI). MPLAB Code Configurator is utilized to easily configure the peripherals used in this motor control application. Refer to the MPLAB® Code Configurator User’s Guide (DS40001725) for further information on how to install and set up the MCC in MPLAB X IDE.

The step-by-step process of using MCC in each of the three solutions of this motor control positioning application is discussed in this section.

8.1 PIC18FXXQ10 Solution MCC Initialization

1. In the System Module, set the clock to HFINTOSC with 32 MHz frequency.
2. Set the ADCC module to Basic_mode, with FOSC/ADCLK clock source configured to FOSC/32.
3. For providing drive signal to the motor, three peripherals are needed to be configured: CCP2, TIMER2 and CWG. Prior to setting the CCP2 in PWM mode, the Timer 2 must be enabled with FOSC/4 clock source. In the CCP2 module, choose the PWM mode and select TIMER 2 as the assigned timer and also set the duty cycle to 100%. Enable the CCP2. The CWG must be configured with CCP2_OUT as an input source, with Output mode in Forward Full-Bridge mode and HFINTOSC as the selected clock source. Enable the CWG.
4. In counting the digital pulses, three peripherals are utilized: CLC1, CCP1 and TIMER1. The CLC1 is used for rerouting the input to TMR1 clock source and gate signal source. Configure CLC1 in 4-input AND mode, with one gate having the input pulses and the rest of the three AND gates inverted to produce an output 1 so that CLC1 acts as a buffer. Enable the CLC1. The CCP1 must be set to Compare mode with TIMER1 as the assigned timer for adjusting its trip point. Set COMPARE in Toggle mode. Enable the CCP interrupt and the CCP module. Choose the CLC1OUT as a TIMER1 clock source, enable the gate and enable Gate Toggle mode with CLC1OUT as the gate signal source and gate polarity as HIGH. Put an initial period count of 0x7FFF being half of the maximum count. Enable the Timer Gate interrupt, but do not enable the TIMER1 yet. The TIMER1 will be enabled in the program.
5. For displaying the data, EUSART1 is configured in Asynchronous mode with Transmit enabled, and a baud rate of 9600. Under its software settings, tick the box next to Redirect STDIO to USART.
6. The MEMORY is added to store the data. From the Device Resources, find the MEMORY and double click so that it will be added to the application.
7. TIMER4 is configured to its Hardware Limit Timer mode to be used for stall detection. The clock source used is MFINTOSC_31.25 kHz with a Prescaler value of 1:128 to extend the period range and Timer Period set to 100 ms. CLC1_OUT is used as an input to the external reset source, with Roll Over Pulse Control mode and TIMER4 resets its count every rising of signal on its reset source. Enable the Timer interrupt, but do not enable the TIMER4 yet, it will be enabled in the program.
8. In the PIN MANAGER configuration, set up the input/output pins of all the peripherals as shown in Figure 8-1.
9. After configuring all the peripherals, click the ‘Generate Code’ button next to the Project Resources tab name in the top left corner. This will generate a main.c file to the project automatically. It will also initialize the module and leave an empty while(1) loop for custom code entry.
Figure 8-1. PIC18FXXQ10 Pin Manager Configuration

![Pin Manager Configuration Diagram]

<table>
<thead>
<tr>
<th>Module</th>
<th>Function</th>
<th>Direction</th>
<th>Port A</th>
<th>Port B</th>
<th>Port C</th>
<th>Port D</th>
<th>Port E</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCC</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCC</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCC</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCC</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANx</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VREF+</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VREF-</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP1</td>
<td>CCP1</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP2</td>
<td>CCP2</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1OUT</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1DIN</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N0</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N1</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N2</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N3</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N4</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N5</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N6</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC1</td>
<td>CLC1N7</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2OUT</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2DIN</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N0</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N1</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N2</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N3</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N4</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N5</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N6</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC2</td>
<td>CLC2N7</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW5</td>
<td>CW5</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW5</td>
<td>CW5A</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW5</td>
<td>CW5B</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW5</td>
<td>CW5C</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CW5</td>
<td>CW5D</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUSART1</td>
<td>RXD</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUSART1</td>
<td>TXD</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSC</td>
<td>CKOUT</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>GROD</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>GROD</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>MCLR</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>TMR1</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>TMR1</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>TMR2</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>TMR2</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Module</td>
<td>TMR4</td>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.2 PIC16F177X Solution MCC Initialization

1. In the System Module, set the clock to INTOSC with 16 MHz frequency.
2. Set the ADC Clock to FOSC/16 clock source.
3. Three peripherals are needed to be configured for each motor drive. TIMER2 is common to both driving motors since it is used for adjusting the frequency of the PWMx. The TIMER2 clock source is set to FOSC/4 before it is enabled. For driving motor 1: PWM3, TIMER2 and COG1 are used. In the PWM3 module, select the TIMER2 as the assigned timer and set the duty cycle to 100%. Enable the PWM. The COG1 must be configured with PWM3 output as an input source to rising and falling event, with Output mode in Forward Full-Bridge mode and HFINTOSC is selected as the clock source. Enable the COG1. For driving motor 2: PWM9, TIMER2 and COG3 are used. In the PWM9 module, select the TIMER2 as the assigned timer and set the duty cycle to 100%. Enable the PWM. The COG3 must be configured with PWM9 output as an input source to the rising and falling event, with Output mode in Forward Full-Bridge mode and HFINTOSC is chosen as the clock source. Enable the COG3.
4. In converting ripples to digital pulses, analog peripherals are used. OPA1 is used for signal filtering, the positive channel is taken from OPA1IN0+ and the negative channel is supplied from OPA1IN1-. OPA4 is used for signal conditioning, the positive channel is taken from OPA4IN0+ and the negative channel is supplied from OPA4IN1-. Enable each op amp after setting up the module. CMP1 is used for converting the conditioned signal to digital pulses with inverted output polarity. The chosen channel for positive input is CIN1+ and for the negative input is CIN0-. Enable the module after choosing the desired configuration.
5. In counting the digital pulses, two peripherals are utilized: CCP1 and TIMER1. The CCP1 must be configured to Compare mode with TIMER1 as the assigned timer for adjusting its trip point. Select Toggle mode. Enable the CCP interrupt and enable the CCP module. Choose External as the TIMER1 clock source, enable the gate and enable Gate Toggle mode with T1G_ pin as the gate signal source and gate polarity as HIGHS. Put an initial period count of 0x7FFF being half of the maximum count. Enable the Timer Gate interrupt, but do not enable the TIMER1 yet. The TIMER1 will be enabled in the program.
6. For displaying the data, EUSART1 is configured in Asynchronous mode with Transmit enabled, and baud rate of 9600. Under its software settings, tick the box next to Redirect STDIO to USART. Another peripheral used for display is MSSP; this peripheral is used to communicate with an LCD. Choose the SPI Master mode, with Clock Polarity (Idle:Low, Active:High) and Clock Edge set to Active to Idle. The SPI clock source is FOSC/64.
7. The MEMORY is added to store the data. From the Device Resources, look for the MEMORY module and double click it so that it will be added to the application.
8. This application is equipped with safety features: Stall Detection and Overcurrent Detection. For Stall Detection, TIMER4 is configured to its Hardware Limit Timer mode. The clock source used is MFINTOSC_31.25 kHz with a Prescaler value of 1:128 to extend the period range and Timer Period set to 100 ms. C1OUT is used as an input to the external reset source, with Roll Over Pulse Control mode and TIMER4 resets its count every rising of signal on its reset source.
9. In the PIN MANAGER configuration, set up the input/output pins of all the peripherals as shown in Figure 8-2.
10. After configuring all the peripherals, click the Generate Code button next to the Project Resources tab name in the top left corner. This will generate a main.c file to the project automatically. It will also initialize the module and leave an empty while(1) loop for custom code entry.
Figure 8-2. PIC16F177X Pin Manager Configuration
8.3 PIC16F188XX Solution MCC Initialization

1. In the System Module, set the clock to HFINTOSC with 32 MHz frequency.
2. Set the ADCC module to Basic_mode, with FOSC/ADCLK clock source configured to FOSC/32.
3. Three peripherals are needed to be configured for each motor drive. TIMER2 is common to both motor drives since it is used for adjusting the frequency of the PWMx. The clock source is set to FOSC/4, afterwards TIMER2 is enabled. For driving motor 1: CCP1, TIMER2 and CWG1 are used. In the CCP1 module, choose the PWM mode and select TIMER 2 as the assigned timer and also set the duty cycle to 100%. Enable the CCP1. The CWG1 must be configured with CCP1_OUT as an input source, with Output mode in Forward Full-Bridge mode and HFINTOSC as the selected clock source. Enable the CWG1. For driving motor 2: CCP2, TIMER2 and CWG2 are used. In CCP2 module, choose the PWM mode and select TIMER 2 as the assigned timer and also set the duty cycle to 100%. Enable the CCP2. The CWG2 must be configured with CCP2_OUT as an input source, with Output mode in Forward Full-Bridge mode and HFINTOSC as the selected clock source. Enable the CWG2.
4. In counting the digital pulses, the signal measurement timer is used. Since this solution is created to drive two motors simultaneously, each motor has a dedicated SMT counter. SMT1 is used for counting the pulses from Motor 1. It is configured to Counter mode with Repeated Data Acquisition. The signal input is taken from SMT1SIGPPS with high/rising edge input polarity and the period count is initialized with the value of 0x7FFFFF. Tick the box next to Enable the Match Interrupt and Enable SMT, but do not tick the box next to Data Acquisition After Init. The Data Acquisition will be enabled in the program. While SMT2 is used for counting the pulses from Motor 2. It is configured to Counter mode with Repeated Data Acquisition. The Signal input is taken from SMT2SIGPPS with high/rising edge input polarity and the period count is initialized with the value of 0x7FFFFF. Tick the box next to Enable the Match Interrupt and Enable SMT, but do not tick the box next to Data Acquisition After Init. The Data Acquisition will be enabled in the program as well.
5. For displaying the data, EUSART1 is configured in Asynchronous mode with Transmit enabled, and baud rate of 9600. Under its software settings, tick the box next to Redirect STDIO to USART. Another peripheral used for display is MSSP; this peripheral is used to communicate with an LCD. Choose the SPI Master mode, with Clock Polarity(Idle:Low, Active:High) and Clock Edge set to Active to Idle. The SPI clock source is FOSC/64.
6. The MEMORY is added to store the data. From the Device Resources, find the MEMORY module and double click it so that it will be added to the application.
7. For safety features, each motor has a dedicated timer for stall detection. TIMER4 is configured to its Hardware Limit Timer mode and used for detecting stall in motor 1. The clock source used is MFINTOSC_31.25 kHz with a Prescaler value of 1:128 to extend the period range and Timer Period set to 100 ms. The T4CKIPPS pin is used as an input to the external reset source, with Roll Over Pulse Control mode and TIMER4 resets its count every rising of signal on its reset source. Enable the Timer interrupt, but do not enable the TIMER4 yet, it will be enabled in the program. While TIMER6 is configured to its Hardware Limit Timer mode and used for detecting stall in motor 2. The clock source used is the same with TIMER4. The T6CKIPPS pin is used as an input to the external reset source, with Roll Over Pulse Control mode and TIMER6 resets its count every rising of signal on its reset source. Enable the Timer interrupt, but do not enable the TIMER6 yet, it will be enabled in the program.
8. In the PIN MANAGER configuration, set up the input/output pins of all the peripherals as shown in Figure 8-3.
9. After configuring all the peripherals, click the ‘Generate Code’ button next to the Project Resources tab name in the top left corner. This will generate a `main.c` file to the project automatically. It will also initialize the module and leave an empty `while(1)` loop for custom code entry.

Figure 8-3. PIC16F188XX Pin Manager Configuration
Appendix C: Source Code Listing

The latest software version can be downloaded from the Microchip website (www.microchip.com). The user will find the source code appended to the electronic version of this application note. The latest version is v1.0.
The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip’s customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.


Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
• Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.


SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.
Quality Management System Certified by DNV

ISO/TS 16949
Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company’s quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.
<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Office</td>
<td>Australia - Sydney</td>
<td>India - Bangalore</td>
<td>Austria - Wels</td>
</tr>
<tr>
<td>2355 West Chandler Blvd.</td>
<td>Tel: 61-2-9868-6733</td>
<td>Tel: 91-80-3090-4444</td>
<td>Tel: 43-7242-2244-39</td>
</tr>
<tr>
<td>Chandler, AZ 85224-6199</td>
<td>China - Beijing</td>
<td>India - New Delhi</td>
<td>Fax: 43-7242-2244-393</td>
</tr>
<tr>
<td>Tel: 480-792-7200</td>
<td>Tel: 86-10-8569-7000</td>
<td>India - Pune</td>
<td>Denmark - Copenhagen</td>
</tr>
<tr>
<td>Fax: 480-792-7277</td>
<td>China - Chengdu</td>
<td>Tel: 91-11-4160-8631</td>
<td>Tel: 45-4450-2828</td>
</tr>
<tr>
<td>Technical Support:</td>
<td>China - Chongqing</td>
<td>Japan - Osaka</td>
<td>Finland - Espoo</td>
</tr>
<tr>
<td><a href="http://www.microchip.com/support">http://www.microchip.com/support</a></td>
<td>Tel: 86-23-8980-9588</td>
<td>Tel: 81-6-6152-7160</td>
<td>Tel: 358-9-4520-820</td>
</tr>
<tr>
<td>Web Address:</td>
<td>China - Dongguan</td>
<td>Japan - Tokyo</td>
<td>France - Paris</td>
</tr>
<tr>
<td><a href="http://www.microchip.com">www.microchip.com</a></td>
<td>Tel: 86-769-8702-9880</td>
<td>Tel: 81-3-6880-3770</td>
<td>Tel: 33-1-69-53-63-20</td>
</tr>
<tr>
<td>Atlanta</td>
<td>China - Guangzhou</td>
<td>Korea - Daegu</td>
<td>Fax: 33-1-69-30-90-79</td>
</tr>
<tr>
<td>Duluth, GA</td>
<td>Tel: 86-20-8755-8029</td>
<td>Korea - Seoul</td>
<td>Germany - Garching</td>
</tr>
<tr>
<td>Tel: 678-957-9614</td>
<td>China - Hangzhou</td>
<td>Tel: 82-53-744-4301</td>
<td>Tel: 49-6931-9700</td>
</tr>
<tr>
<td>Fax: 678-957-1455</td>
<td>China - Hong Kong SAR</td>
<td>China - Penang</td>
<td>Germany - Haan</td>
</tr>
<tr>
<td>Austin, TX</td>
<td>Tel: 852-2943-5100</td>
<td>Tel: 60-4-227-8870</td>
<td>Tel: 49-2129-3766400</td>
</tr>
<tr>
<td>Tel: 512-257-3370</td>
<td>China - Nanjing</td>
<td>Malaysia - Kuala Lumpur</td>
<td>Germany - Heilbronn</td>
</tr>
<tr>
<td>Boston</td>
<td>Tel: 86-25-8473-2460</td>
<td>Tel: 60-3-7651-7906</td>
<td>Tel: 49-7131-67-3636</td>
</tr>
<tr>
<td>Westborough, MA</td>
<td>China - Qingdao</td>
<td>Malaysia - Penang</td>
<td>Germany - Karlsruhe</td>
</tr>
<tr>
<td>Tel: 774-750-0087</td>
<td>Tel: 86-532-8502-7355</td>
<td>Tel: 60-2-634-9065</td>
<td>Tel: 49-72-625370</td>
</tr>
<tr>
<td>Fax: 774-760-0088</td>
<td>China - Shanghai</td>
<td>Singapore</td>
<td>Germany - Munich</td>
</tr>
<tr>
<td>Chicago</td>
<td>Tel: 86-21-3326-8000</td>
<td>Tel: 65-6334-8870</td>
<td>Tel: 49-89-627-144-0</td>
</tr>
<tr>
<td>Itasca, IL</td>
<td>China - Shenzhen</td>
<td>Taiwan - Hsin Chu</td>
<td>Fax: 49-89-627-144-44</td>
</tr>
<tr>
<td>Tel: 630-285-0071</td>
<td>Tel: 86-755-8864-2200</td>
<td>Taiwan - Kaohsiung</td>
<td>Germany - Rosenheim</td>
</tr>
<tr>
<td>Fax: 630-285-0075</td>
<td>China - Suzhou</td>
<td>Tel: 886-7-213-7830</td>
<td>Tel: 49-8031-354-560</td>
</tr>
<tr>
<td>Dallas</td>
<td>Tel: 86-24-2334-2829</td>
<td>Taiwan - Taipei</td>
<td>Israel - Ra’anana</td>
</tr>
<tr>
<td>Addison, TX</td>
<td>China - Wuhan</td>
<td>Tel: 886-2-2508-8600</td>
<td>Tel: 972-9-744-7705</td>
</tr>
<tr>
<td>Tel: 972-818-7423</td>
<td>Tel: 86-27-5980-5300</td>
<td>Thailand - Bangkok</td>
<td>Italy - Milan</td>
</tr>
<tr>
<td>Fax: 972-818-2924</td>
<td>China - Xian</td>
<td>Tel: 66-2-694-1351</td>
<td>Tel: 39-0331-742611</td>
</tr>
<tr>
<td>Detroit</td>
<td>Tel: 86-29-8833-7252</td>
<td>Vietnam - Ho Chi Minh</td>
<td>Fax: 39-0331-466781</td>
</tr>
<tr>
<td>Tel: 248-848-4000</td>
<td>China - Xiamen</td>
<td>Tel: 84-28-5448-2100</td>
<td>Italy - Padova</td>
</tr>
<tr>
<td>Houston, TX</td>
<td>Tel: 86-592-2388138</td>
<td></td>
<td>Tel: 39-049-7625286</td>
</tr>
<tr>
<td>Tel: 281-894-5983</td>
<td>China - Zhuhai</td>
<td></td>
<td>Netherlands - Drunen</td>
</tr>
<tr>
<td>Indianapolis</td>
<td>Tel: 86-756-3210040</td>
<td></td>
<td>Tel: 31-416-690399</td>
</tr>
<tr>
<td>Noblesville, IN</td>
<td></td>
<td></td>
<td>Fax: 31-416-690340</td>
</tr>
<tr>
<td>Tel: 317-773-8323</td>
<td></td>
<td></td>
<td>Norway - Trondheim</td>
</tr>
<tr>
<td>Fax: 317-773-5453</td>
<td></td>
<td></td>
<td>Tel: 47-7286438</td>
</tr>
<tr>
<td>Tel: 317-536-2380</td>
<td></td>
<td></td>
<td>Poland - Warsaw</td>
</tr>
<tr>
<td>Los Angeles</td>
<td></td>
<td></td>
<td>Tel: 48-22-3325737</td>
</tr>
<tr>
<td>Mission Viejo, CA</td>
<td></td>
<td></td>
<td>Romania - Bucharest</td>
</tr>
<tr>
<td>Tel: 949-462-9523</td>
<td></td>
<td></td>
<td>Tel: 40-21-407-87-50</td>
</tr>
<tr>
<td>Fax: 949-462-9608</td>
<td></td>
<td></td>
<td>Spain - Madrid</td>
</tr>
<tr>
<td>Tel: 951-273-7800</td>
<td></td>
<td></td>
<td>Tel: 34-91-708-08-90</td>
</tr>
<tr>
<td>Raleigh, NC</td>
<td></td>
<td></td>
<td>Fax: 34-91-708-08-91</td>
</tr>
<tr>
<td>Tel: 919-844-7510</td>
<td></td>
<td></td>
<td>Sweden - Gothenberg</td>
</tr>
<tr>
<td>New York, NY</td>
<td></td>
<td></td>
<td>Tel: 46-31-704-60-40</td>
</tr>
<tr>
<td>Tel: 631-435-6000</td>
<td></td>
<td></td>
<td>Sweden - Stockholm</td>
</tr>
<tr>
<td>San Jose, CA</td>
<td></td>
<td></td>
<td>Tel: 46-8-5090-4654</td>
</tr>
<tr>
<td>Tel: 408-735-9110</td>
<td></td>
<td></td>
<td>UK - Wokingham</td>
</tr>
<tr>
<td>Fax: 408-436-4270</td>
<td></td>
<td></td>
<td>Tel: 44-118-921-5800</td>
</tr>
<tr>
<td>Canada - Toronto</td>
<td></td>
<td></td>
<td>Fax: 44-118-921-5820</td>
</tr>
</tbody>
</table>