AN3353

3-Axis Stepper Motor Control Using an 8-Bit PIC® Microcontroller

Introduction

Author: Maria Loida Canada, Microchip Technology Inc.

Three-axis control applications, such as on a CNC machine, robotics and dispensing machines, are widely used in the industry. Most often, each motor has a dedicated controller that facilitates its speed control and sets its movement limitations. The use of multiple controllers in the development of the control system implies a higher cost.

This project is created to develop a solution that can control up to three motors simultaneously. Utilizing a single PIC® MCU with its Core Independent Peripherals (CIPs), driving the motors can be performed without additional burden to the core. The developed cost-effective solution provides an accurate linear motor movement. The PIC® device can be used solely with the positions data embedded in its firmware or can be used as a slave for the applications requiring more sophisticated control.

Table of Contents

1. Overview

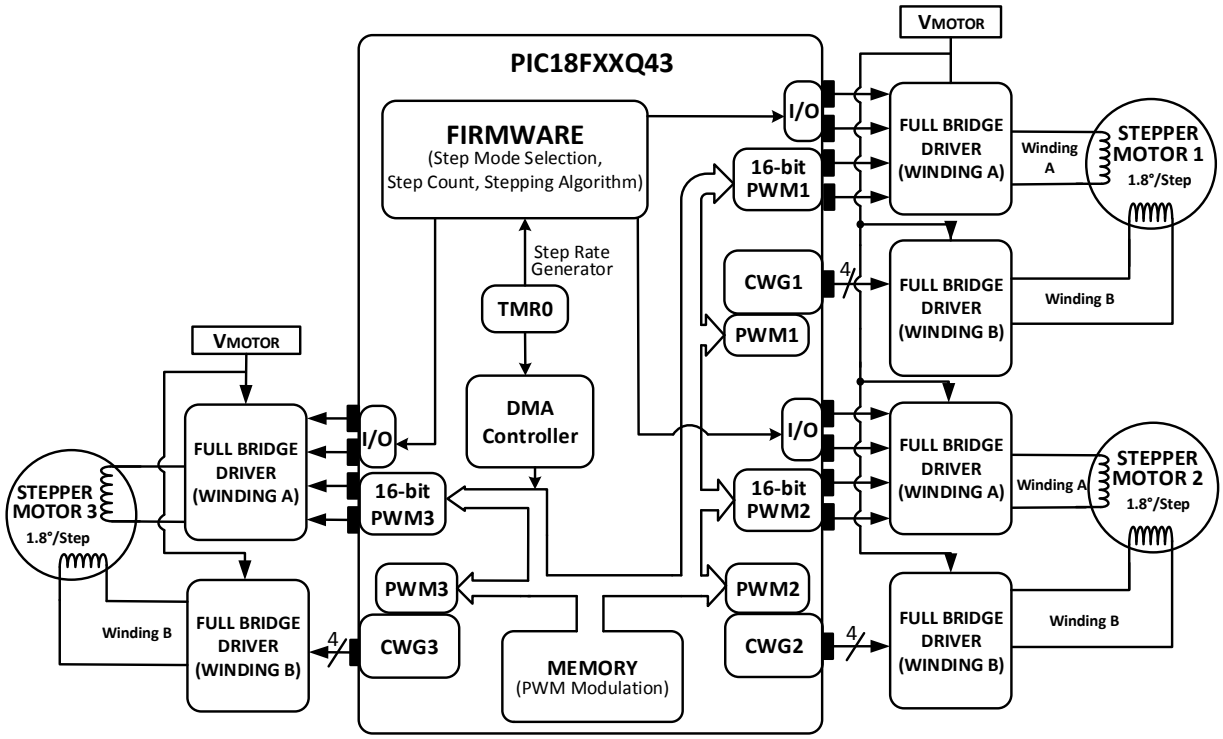
This application note describes a practical solution for controlling three motors independently. This application uses a single PIC18F-Q43 device to control the drive signal being fed to the driver of stepper motors in 3-axes. With the use of a single 8-bit microcontroller, the implementation cost is substantially reduced.

This application has the following key features:

- Full-step, half-step, and microstepping (1/4 and 1/16) modes
- Configurable steps/coordinate resolution
- Speed and direction control of each motor
- Up to three motors simultaneously controlled
- Motor control using Core Independent Peripherals (CIPs)

The interconnection of peripherals to control the signals used for driving the motors is shown in Figure 1-1. The CIPs used in this design are the new 16-bit high-resolution PWM, Complementary Waveform Generator (CWG) and Direct Memory Access (DMA). The full-bridge driver is used for bipolar stepper motor control. The integration of on-chip peripherals like TMR0 and a conventional PWM with the firmware, enables the system to reliably perform 3-axis control with minimum software overhead.

Figure 1-1. 3-Axis Motor Control Block Diagram



Note: For this application, Leadshine 42HS03 motors were used.

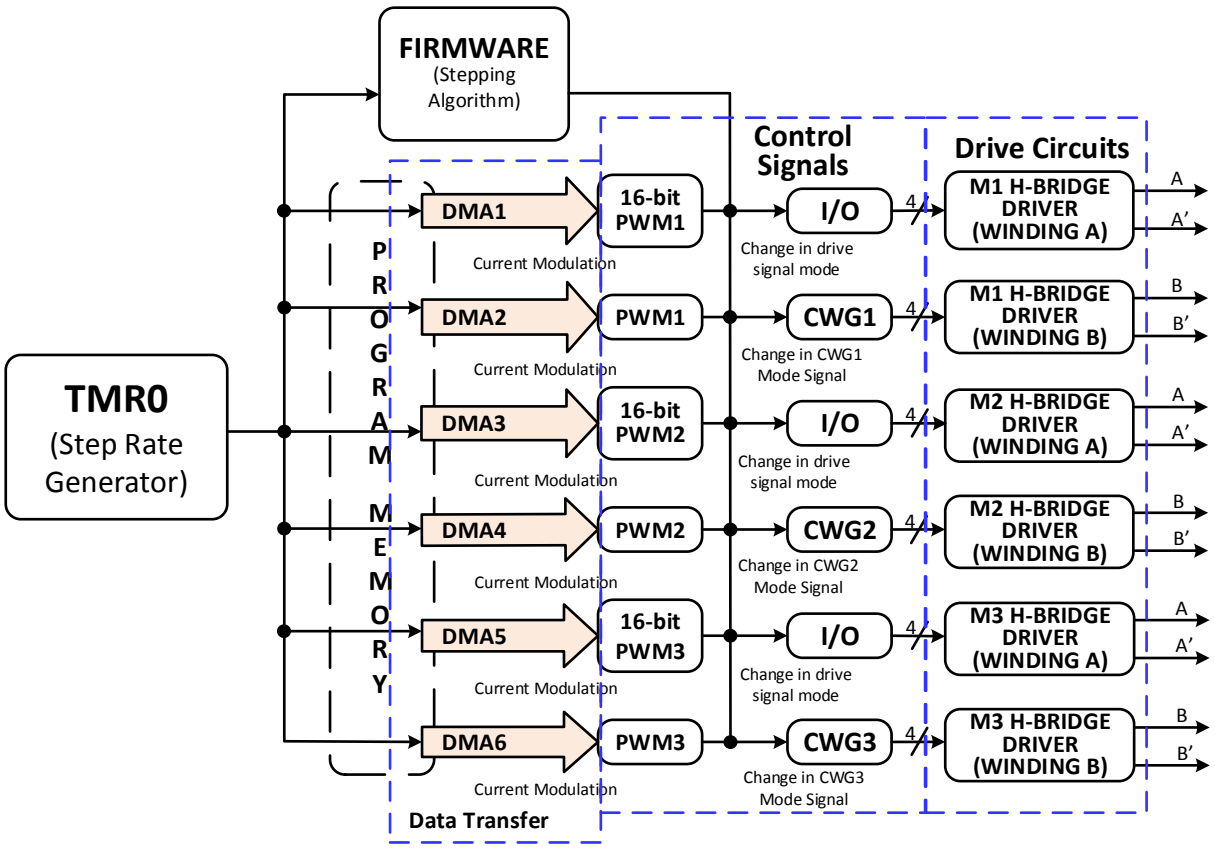
2. Stepper Motor Control

A stepper motor is a type of motor that rotates in discrete steps. It divides a full rotation into a number of equal steps and moves through it, one at a time. It converts the input digital pulses into mechanical shaft rotation. It can be driven to rotate a specific number of steps and stop precisely when triggered to stop. For an in-depth discussion about the fundamentals of stepper motors, refer to AN907: Stepping Motors Fundamentals.

2.1 Control Overview

Figure 2-1 shows a block diagram of the generic system used for controlling the three stepper motors. TMR0 acts as a step rate generator, which is primarily responsible for controlling the speed of the motors. Every time the TMR0 rolls over, the stepping sequence in the firmware is loaded to the CWG and GPIO registers, while loading the PWM values through the DMA. Bipolar motor control circuit, which is composed of two H-bridge drivers, is used for driving each motor in a clockwise or counterclockwise direction. The drive signal for each motor is a combination of CWG and GPIO signals along with the 16-bit PWM output. Lastly, the firmware dictates the limitation of movement, depending on the specified position.

Figure 2-1. 3-Axis Motor Control Diagram

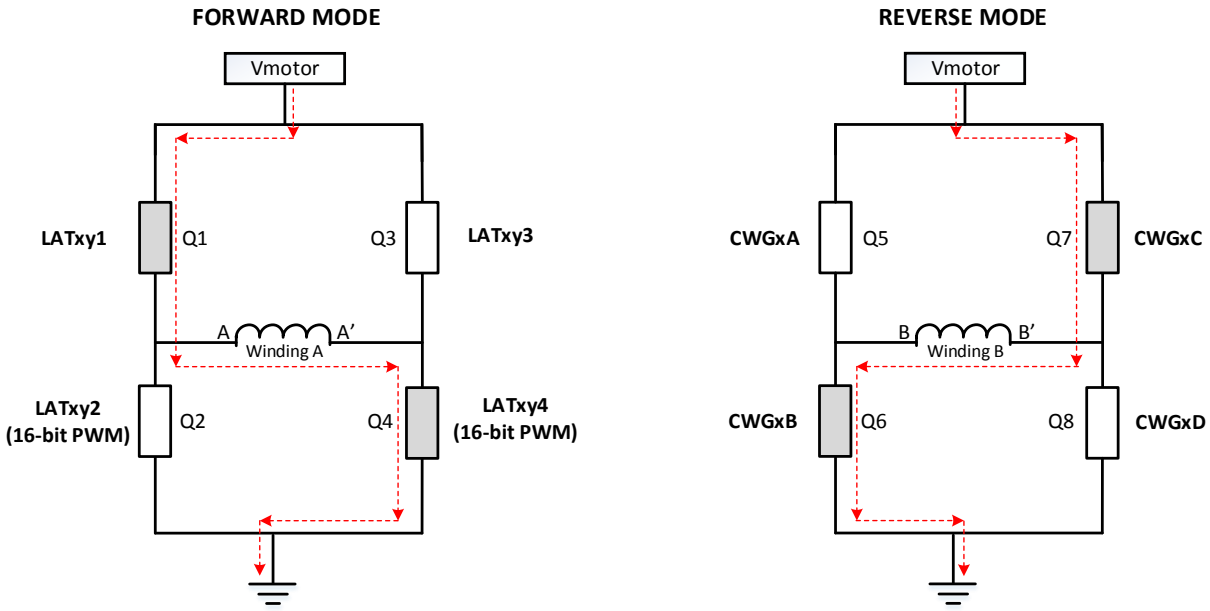


2.2 Drive Circuit and Control Process

The rotating magnetic field and varying magnetic pole polarity (North/South) on the stator causes the rotor to spin. The magnetic field and magnetic polarity variation are produced by electrically energizing the two stepper motor windings (Winding A and B). The energization is controlled by the CWG and PWM signals through the use of an H-bridge circuit. Figure 2-2 depicts the current flow through the half-bridge circuit during Forward and Reverse mode. The naming convention of Forward and Reverse mode are adopted from the modes of CWG and will be used throughout the document to indicate the state of Winding mode.

When Winding A is in Forward mode, the current is flowing through MOSFETs Q1 and Q4 down to ground, while ensuring that Q2 and Q3 are OFF. Otherwise, when it operates in Reverse mode, the current flows through Q2 and Q3, while keeping Q1 and Q4 OFF. The same principle applies to the MOSFETs Q5, Q6, Q7 and Q8 on Winding B. The switching of the MOSFETs in Winding B is implemented through the use of CWG Forward and Reverse-Full Bridge modes, while the MOSFETs in Winding A are controlled by GPIO output and 16-bit PWM. Thus, the drive method for each motor is a combination of firmware and peripheral drive, to assure that all the motors are exposed to similar control factors. Refer to TB3118: Complementary Waveform Generator, for more details about the CWG peripheral. For the drive circuits used, refer to section Appendix A: Circuit Schematics.

Figure 2-2. Drive Circuits and Control Signals



	FORWARD MODE	REVERSE MODE
Q1/Q5	ON	OFF
Q2/Q6	OFF	MODULATED
Q3/Q7	OFF	ON
Q4/Q8	MODULATED	OFF

2.3 16-Bit High Resolution PWM for Control Signal

The 16-bit Pulse-Width Modulator can produce a high-resolution modulation at low frequency. In this application, the drive resolution will be mainly determined by the step mode implementation. The switching frequency must be high enough to operate beyond the audio frequency. The arrangement was made by choosing the PWM to operate in 10-bit, having a 62.5 kHz switching frequency, which is within the range of typical driver switching frequency.

The 16-bit PWM is used for driving the low-side MOSFETs of Winding A for all motors. It is connected to both low-side MOSFETs through Peripheral Pin Select (PPS), but it is never intended to turn on both sides simultaneously. The 16-bit PWM has an independent 16-bit period timer, which is chosen to be HFINTOSC or equivalent to 64 MHz in this application. The PWMOUT is in Left-Aligned mode. For proper PWM operation, the registers PWMxPR and PWMxSaP1 must be properly configured. The requested frequency can be attained by setting up the PWMxPR register. The value of this register is equivalent to the number of PWM clock periods in the PWM period or it can be expressed as shown in Equation 2-1.

The PWMxSaP1 register determines the active period of slice “a”, parameter 1 output. The duty cycle shown in Equation 2-2 can then be calculated by getting the ratio of PWMxSaP1 to the PWMxPR value.

The automatic loading of the PWMxSaP1 register is enabled through setting the respective DMAx as the auto-load trigger source in the PWMxLDS register. For more information about the 16-bit PWM, refer to the device data sheet.

Equation 2-1. PWM Period Register Value

$$PWMxPR = \frac{PWM\ Clock(Hz)}{Requested\ \ Freq\ (Hz)}$$

Equation 2-2. Duty Cycle Calculation

$$Duty\ \ \ Cycle = \frac{PWMSaP1}{PWMxPR} \times 100\ \%$$

2.4 Data Transfer

The Look-Up Table of modulation values is initially stored in Programmable Flash Memory. The DMA is utilized for transferring the modulation values to PWM registers without the CPU intervention, freeing up the core for doing other tasks. DMA1 is used for transferring the values from the program memory to the Slice 1 Output of 16-bit PWM1 defined by the PWM1S1P1L and PWM1S1P1H registers, illustrated in Figure 2-1, while DMA2 is used for passing on the modulation data from the program memory to the conventional PWM1 registers: CCPR1L and CCPR1H. Each DMA channel is tied to its specific PWM peripheral to ensure that the transfer will simultaneously take place whenever a transfer trigger is received.

All the DMA channels are configured to start the data transfer when TMR0 interrupt is triggered. The TMR0 is selected as a trigger for starting the data transfer through the DMAnSIRQ (DMA Start Interrupt Request Source Selection) register. But for the interrupt source to take effect, the SIRQEN (Start of Transfer Interrupt Request Enable) register of each DMA must be enabled.

This application provides a drive implementation in 1/4 and 1/16 microstepping. It is important to note that the array size of PWM modulation values for 1/4 and 1/16 microstepping are different and the respective DMA source size register must be correctly initialized to transfer all the data necessary to complete a full step. Since the DMA is operating in 8-bit, the DMA source size must be equivalent to twice the number of array elements. For 1/4 microstepping, the step resolution is 16, which means that the DMA source size register must be equivalent to 32 or 0x20; and for 1/16 microstepping with step resolution of 64 the DMA source size must be 128 or 0x80. To learn more information about DMA, refer to TB3164: Direct Memory Access on 8-bit PIC® Microcontrollers and refer to the device data sheet for proper peripheral configuration.

3. Stepper Motor Control Characteristics

For proper operation, stepper motor characteristics must be carefully considered. Characteristics to consider are torque, speed and stepping rate. This section discusses about how these characteristics affect the stepping implementation.

3.1 Torque Consideration

Stepper motor maintains torque at relatively low speed. The torque required to move the load must be met by the stepper motor specifications. Stepping motor manufacturers will specify several torques in their data sheets for their motors. The torque magnitude depends on the driving technique, stepping rate, and winding current.

3.2 Stepping Rate

Step rate refers to the speed at which the hardware-firmware combination can send step pulses to the stepper motor driver. It is expressed in PPS (Pulse Per Second) and dictates the speed of the motor. To know more about how it is implemented using a PIC microcontroller, refer to the Stepping Rate section of AN2326: High-Torque/High Power Bipolar Stepper Motor using 8-bit PIC® MCUs.

4. Step Mode Implementation

Step mode is a drive technique used to rotate a stepper motor. It indicates the magnitude of a step angle taken in every energization of the stator windings. Each mode has a corresponding stepping resolution and output torque. The step modes that can be implemented in 3-axis control using a PIC18F-Q43 microcontroller are:

- Full-step Drive
- Half-step Drive
- Microstepping Drive (1/4 and 1/16)

The step mode implementation used in this application is referenced in AN2326: High-Torque/High Power Bipolar Stepper Motor using 8-bit PIC® MCUs. The principle of the stepping mode used is the same, only the actual implementation using the PIC18FXXQ43 device will be shown here. Likewise, the implementation is applicable to all three motors in 3-axis.

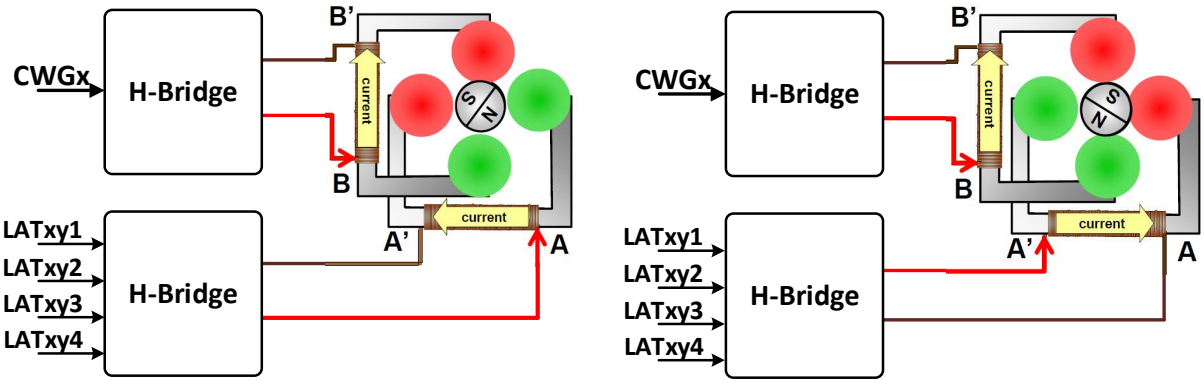
4.1 Full-Step Drive

On this drive, two phases are energized at the same time. The drive circuit implementation is shown in Figure 4-1. The CWG controls the Winding B while the Winding A is controlled by toggling the output of GPIO depending on stepping algorithm. Meanwhile, Figure 4-2 illustrates the algorithm used for stepping the motor. The individual output state of each pin is shown to properly move the shaft's position into its equivalent full-step angle.

Figure 4-3 shows the firmware execution used in controlling the signals. Every time the TMR0 interrupt is triggered, `stepCounter` variables are incremented. `stepCounter` signifies that the specific mode used is full-step and resets to 0 whenever it reaches 4. `stepCounter2Mx` are compared to their corresponding `MxdesiredStep`. If the counter exceeds the desired step, the motor will be stopped and *`axis`*`_movementDone` will be set. The definition of the *`axis`* can be `x, y` or `z`, depending on the specific motor movement it resembles. After all the `axis_movementDone` are set, the TMR0 is disabled to make sure that it will not count and to avoid an interrupt to be improperly triggered. All the stepCounter2Mx are cleared to ensure that the counting for the succeeding positions begins at zero. The `movementDone` variable is used in the main program to determine if all the motor movements are finished and gives the signal to prepare the system for another movement.

The switch case determined by the `motorDir` variable decides on which direction the motors will rotate. The motors are designed to be driven simultaneously, so at every Interrupt event, the `motorDir` will command which case to execute. Each case contains the direction of the motors and all the motors that used the drive table for clockwise and counterclockwise direction. The drive table requires a specific set of IO and CWG for each motor.

Figure 4-1. Full-Step Drive



Step	1	2	3	4
Winding A	A	A'	A'	A
Winding B	B	B	B'	B'

Step	1	2	3	4
Winding A	A	A'	A'	A
Winding B	B	B	B'	B'

Figure 4-2. Full-Step Drive Stepping Algorithm in Clockwise Direction

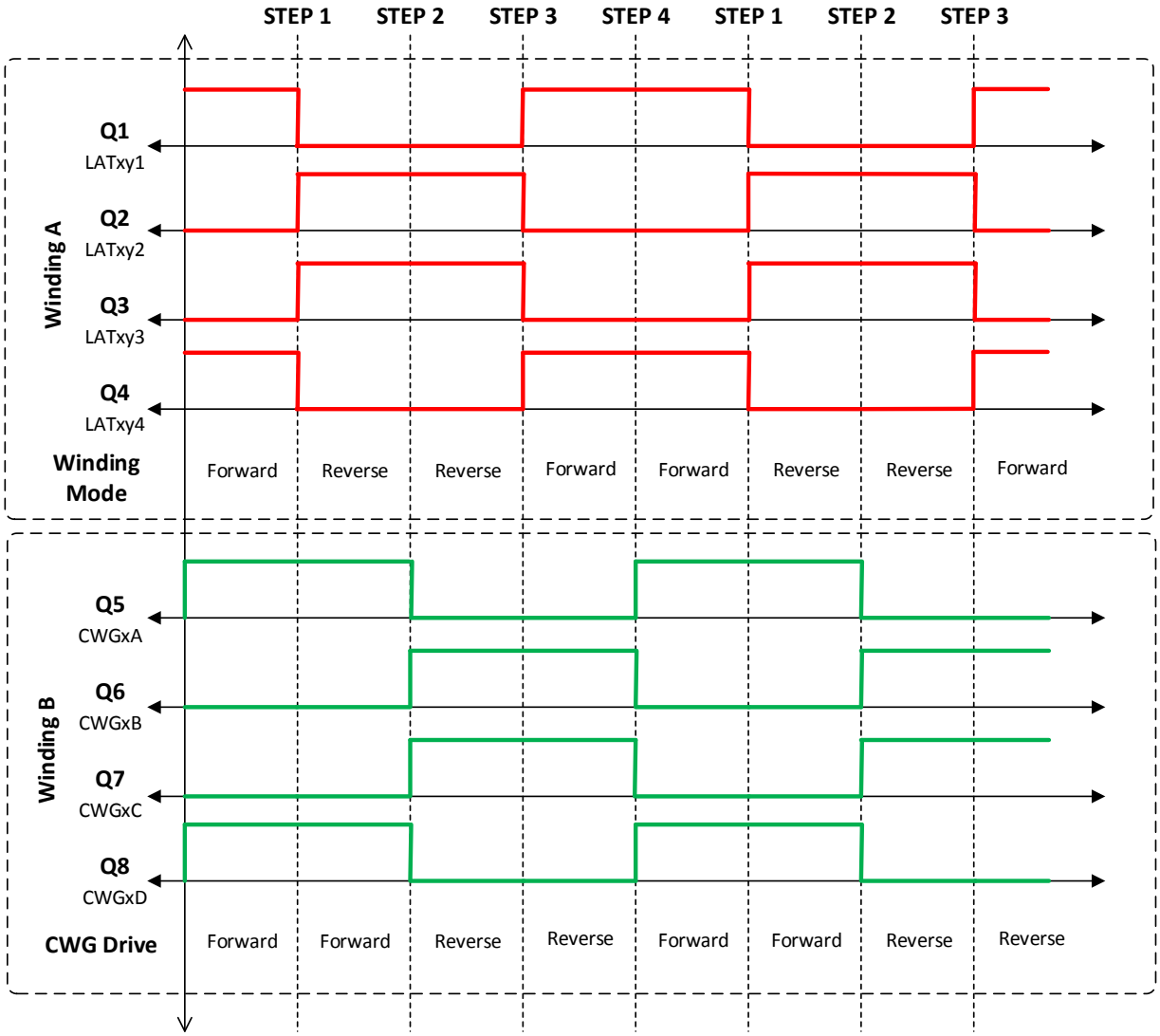
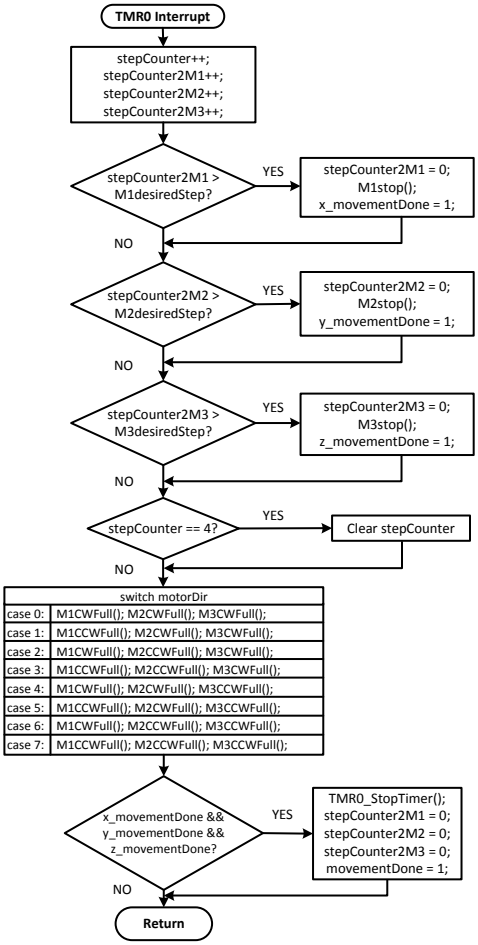


Figure 4-3. Full-Step Drive Stepping Algorithm Flowchart



FULL STEP DRIVE TABLE CLOCKWISE DIRECTION

Step_One:	LATxbits.LATxy1 = 0
	TRISxbits.TRISxy2 = 0
	LATxbits.LATxy3 = 1
	TRISxbits.TRISxy4 = 1
	CWGxCON0bits.MODE0 = 1 (Reverse)
Step_Two:	LATxbits.LATxy1 = 1
	TRISxbits.TRISxy2 = 1
	LATxbits.LATxy3 = 0
	TRISxbits.TRISxy4 = 0
	CWGxCON0bits.MODE0 = 1 (Reverse)
Step_Three:	LATxbits.LATxy1 = 1
	TRISxbits.TRISxy2 = 1
	LATxbits.LATxy3 = 0
	TRISxbits.TRISxy4 = 0
	CWGxCON0bits.MODE0 = 0 (Forward)
Step_Four:	LATxbits.LATxy1 = 0
	TRISxbits.TRISxy2 = 0
	LATxbits.LATxy3 = 1
	TRISxbits.TRISxy4 = 1
	CWGxCON0bits.MODE0 = 0 (Forward)

FULL STEP DRIVE TABLE COUNTERCLOCKWISE DIRECTION

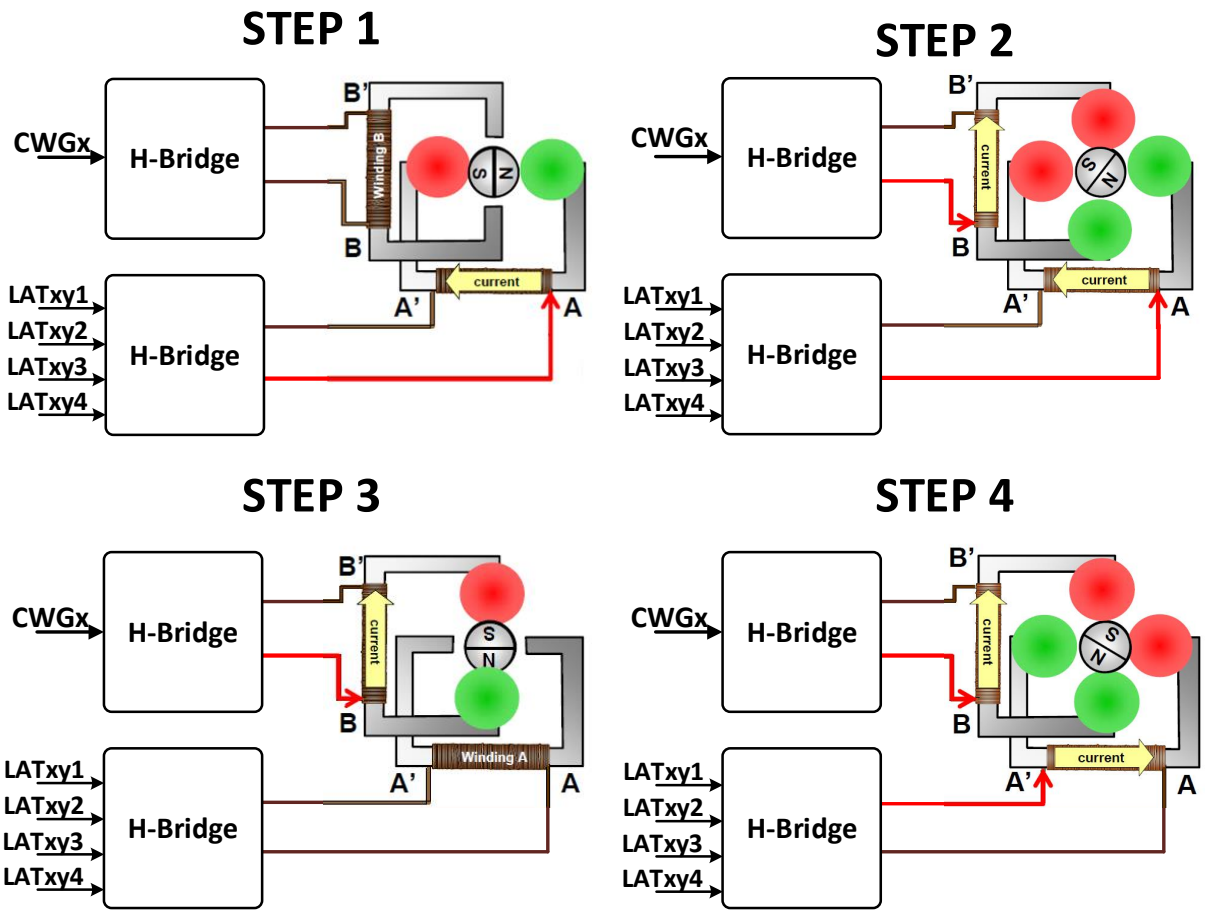
Step_One:	LATxbits.LATxy1 = 1
	TRISxbits.TRISxy2 = 1
	LATxbits.LATxy3 = 0
	TRISxbits.TRISxy4 = 0
	CWGxCON0bits.MODE0 = 1 (Reverse)
Step_Two:	LATxbits.LATxy1 = 0
	TRISxbits.TRISxy2 = 0
	LATxbits.LATxy3 = 1
	TRISxbits.TRISxy4 = 1
	CWGxCON0bits.MODE0 = 1 (Reverse)
Step_Three:	LATxbits.LATxy1 = 0
	TRISxbits.TRISxy2 = 0
	LATxbits.LATxy3 = 1
	TRISxbits.TRISxy4 = 1
	CWGxCON0bits.MODE0 = 0 (Forward)
Step_Four:	LATxbits.LATxy1 = 1
	TRISxbits.TRISxy2 = 1
	LATxbits.LATxy3 = 0
	TRISxbits.TRISxy4 = 0
	CWGxCON0bits.MODE0 = 0 (Forward)

4.2 Half-Step Drive

Half-step drive alternates between two phases on and a single phase on. It increases the resolution of the angle by halving the basic step angle, thus causing a smoother rotation than full-step. Figure 4-4 shows the drive circuit implementation with the control signals coming from CWG and GPIO output. While, Figure 4-5 illustrates the stepping algorithm used in this drive technique.

Figure 4-6 shows the firmware execution similar to the firmware flow in Figure 4-3. The value of the `stepCounter` doubled, which clearly states that the algorithm used is twice as long as the full-step algorithm. The drive table in clockwise and counterclockwise direction for all the steps are also shown.

Figure 4-4. Half-Step Drive Circuit



Step	1	2	3	4	5	6	7	8
Winding A	A	A	0	A'	A'	A'	0	A
Winding B	0	B	B	B	0	B'	B'	B'

Figure 4-5. Half-Step Drive Stepping Algorithm

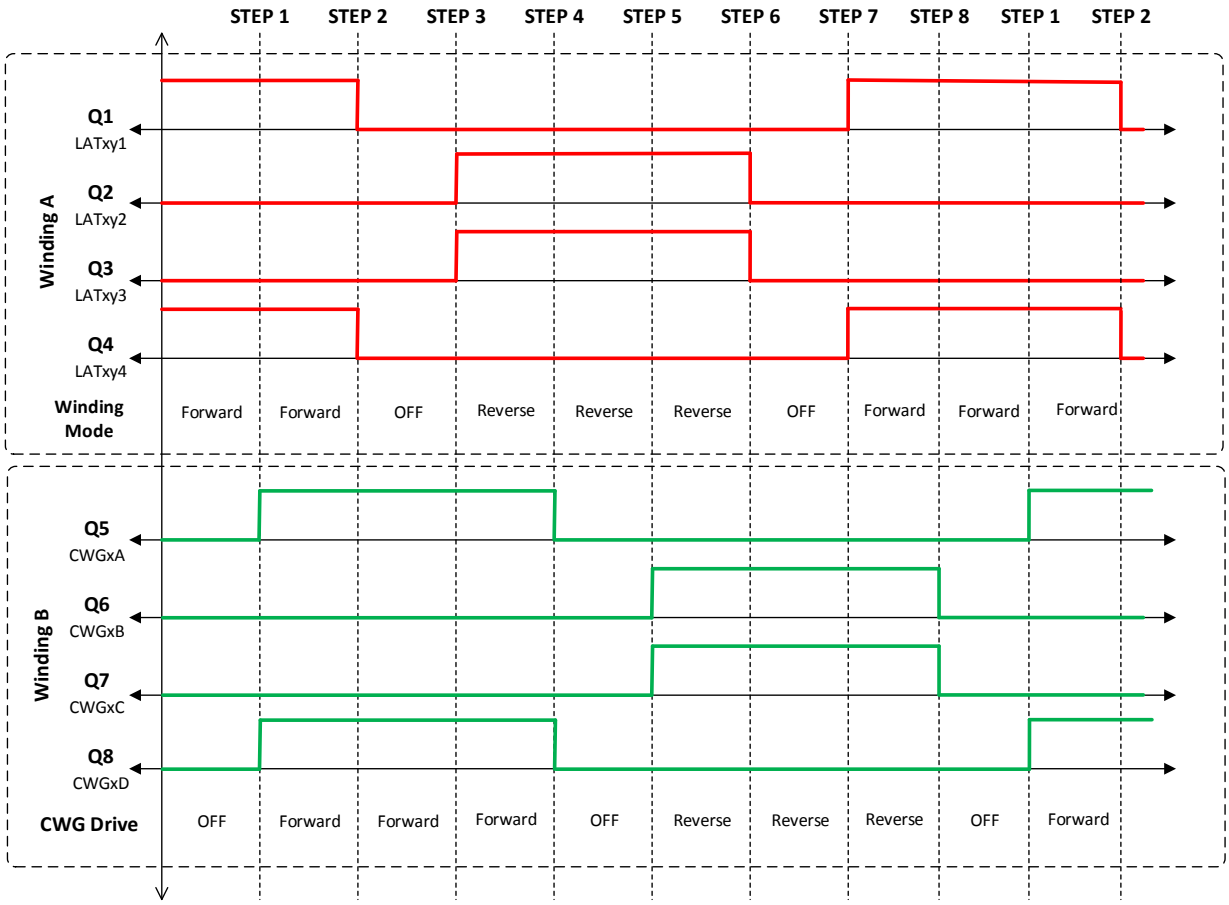
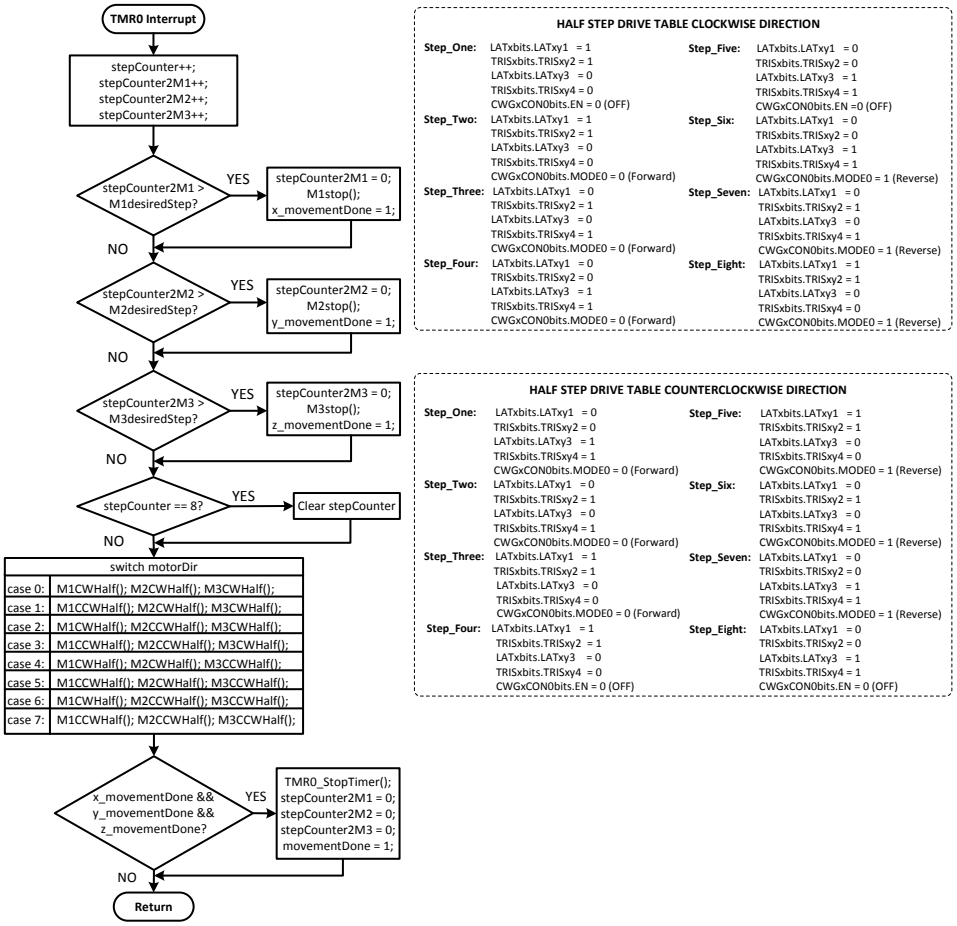


Figure 4-6. Half-Step Drive Firmware Execution



4.3 Microstepping

Microstepping is a manner of moving the stator flux of a stepper motor smoothly. One type of microstepping is constant-torque, which is used in this application. Constant-torque microstepping is achieved by simultaneously varying the current in both windings of a stepper motor.

The drive circuit used in this stepping mode is identical to the full-step and half-step circuits. However, the control signals are modulated instead of supplying a full-on or full-off signals. The currents can be varied by changing the PWM percent modulation in accordance with the Equation 4-1 and Equation 4-2.

Equation 4-1. Winding A Current Formula

$$I_A = I_{MAX} \times \sin\left(\frac{Step\ Number \times 360}{Step\ Resolution}\right)$$

Equation 4-2. Winding B Current Formula

$$I_B = I_{MAX} \times \cos\left(\frac{Step\ Number \times 360}{Step\ \ Resolution}\right)$$

Using the 1/16 microstepping, assume that I_{MAX} is equivalent to one and the drive is in step number one. The sin of 360° multiplied by the present step number divided by 64 (1/16 microstepping resolution) results to 0.098. This represents that the modulation of current in Winding A must only be 9.8% of the maximum current. The modulation of Winding A and Winding B for the remaining 63 steps is calculated and plotted in Figure 4-7. The step resolution of 64 makes the torque graph resemble a sinusoid. The stepping algorithm used in 1/16 microstepping is shown on Figure 4-8.

Figure 4-7. Phase Diagram and Torque Response of Constant Torque 1/16 Microstepping

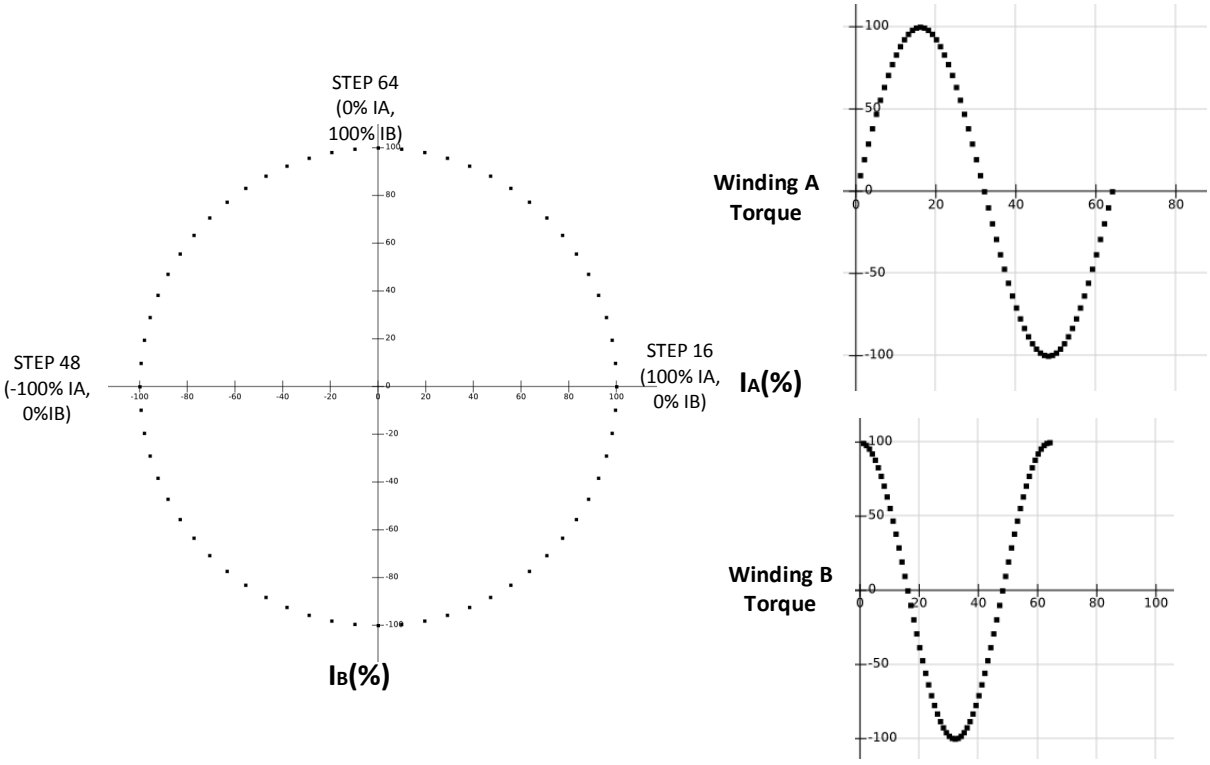
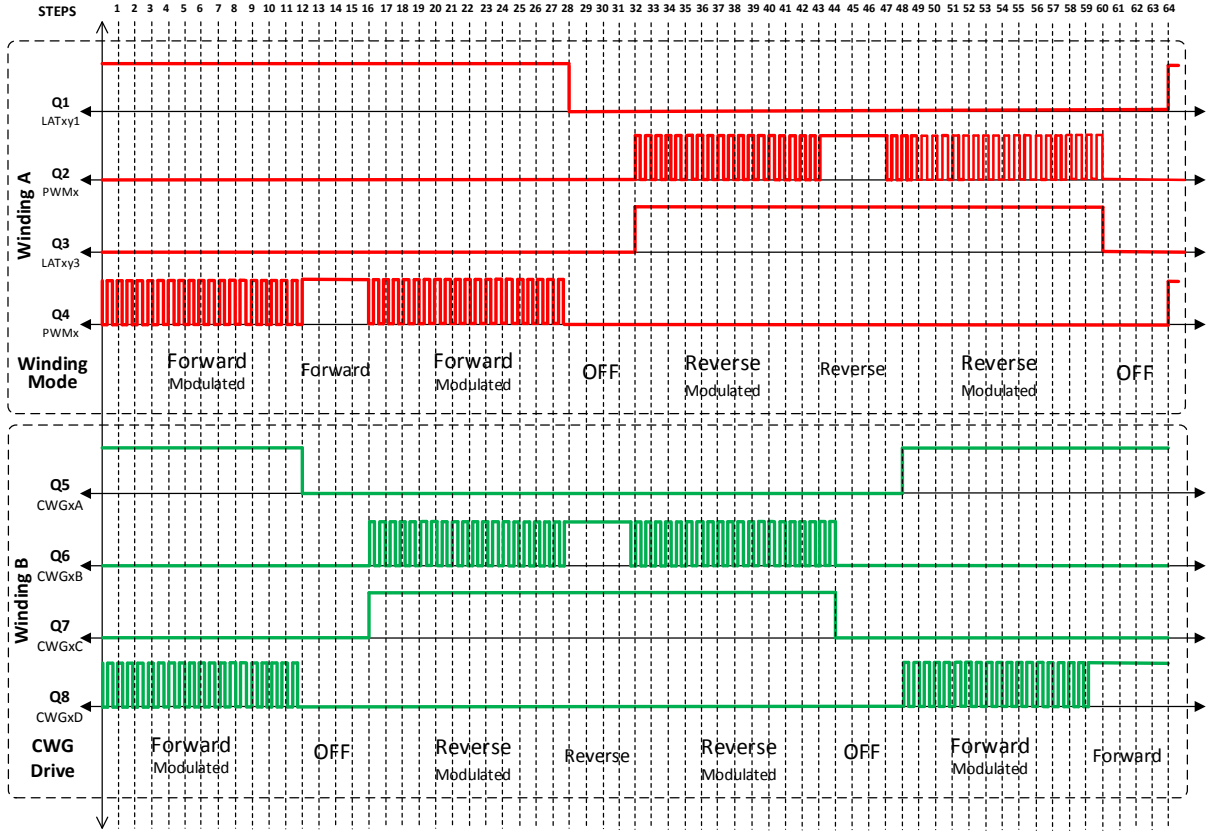


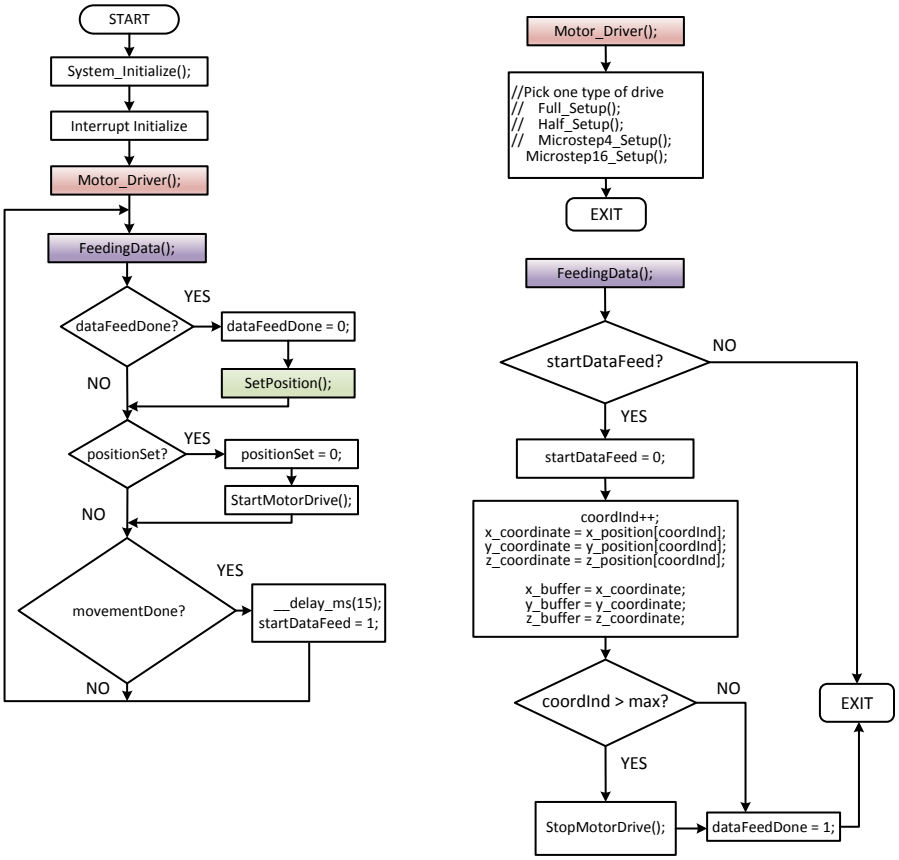
Figure 4-8. Constant-Torque Microstepping Drive Signal



5. Firmware Flow Diagram

This section explains the firmware design implemented for successfully driving the three motors in 3-axis.

Figure 5-1. Firmware Flowchart



The firmware flow starts with the `SYSTEM_Initialize()` routine. This routine initializes pin configuration, oscillator and all the peripherals used in the application. It is followed by enabling interrupts to address the functions requiring interrupts. The `Motor_Driver()` function contains the different drive implementation. Pick one type of drive-by uncommenting the function of the chosen drive. The selected drive will assign the value of `stepping_mode_constant`, which will be used for calculating the number of steps required to rotate in a distinct position. Notice that the type of drive will only be setup once at the beginning of the program, and the type of drive that will be used will depend on the application requirement and user's discretion.

After all the initialization is done, the program will undergo a continuous loop, executing the functions necessary to successfully move the motors in predefined positions. The `FeedingData()` function contains the commands for acquiring and processing the desired position. The positions initialized at the beginning of the program are placed in the `axis_coordinate` parameters for later processing. The condition that tests `coordInd` against `max` variable tells if the end position has been reached and signifies that the motors will be stopped if the condition is satisfied. Otherwise, the `dataFeedDone` variable will be set to be used as a test variable for the next instruction.

When the `dataFeedDone` test variable is satisfied, the `SetPosition()` function will be executed. This function serves a vital role in the 3-axis motion control. This function consists of checking if the individual motor movements are completed through testing the `axis_movementDone` variable. Provided that the previous movement is finished, the current axes coordinate will be tested against the previous axes coordinate. If the current and previous coordinates are not equal, two conditional statements will be tested consequently. The first condition tests if the current coordinate is less than the previous coordinate; if so, the resulting current coordinate will be the previous

coordinate less the current coordinate shown in Equation 5-1. The `Mxdirection` is set to the counterclockwise direction and the DMA is initialized for transferring the modulation data necessary for moving in the counterclockwise direction. However, if the current coordinate is greater than the previous coordinate, the resulting current coordinate will be the current coordinate less the previous coordinate, as shown in Equation 5-2. Furthermore, the `Mxdirection` will be set to the clockwise direction and the DMA source is initialized to transfer the modulation data for clockwise direction.

The resulting current coordinate will be used to compute for the motor's desired step, which is shown in Equation 5-3. The constant variable *STEPS_PER_COORDINATE* may take any value, depending on the tool used for converting rotational to linear motion. This variable is introduced for flexibility in the application. The meaning of `axis` in `axis_coordinate` can be `x`, `y` or `z`, depending on the coordinate, and the `x` in `MxDesiredStep` can be `1`, `2` or `3`, depending on the motor number. The pairs created in this control scheme are as follows: `x_coordinate` to M1, `y_coordinate` to M2 and `z_coordinate` to M3.

Equation 5-1. axis_coordinate < axis_prevCoordinate

$$axis_{coordinate} = axis_{prevCoordinate} - axis_{coordinate}$$

Equation 5-2. axis_coordinate > axis_prevCoordinate

$$axis_{coordinate} = axis_{coordinate} - axis_{prevCoordinate}$$

Equation 5-3. Motor Desired Step

$$MxDesiredStep = axis\_coordinate \times STEPS\_PER\_COORDINATE \times stepping\_mode\_constant$$

Subsequently, the previous coordinate equates to the current coordinate to be the reference for successive positions. Then, the motor drive is enabled by clearing the auto-shutdown feature of CWG and setting the drive pins as output. The `positionSet` variable is set, which primarily indicates that the setting of position is done.

Consequently, the function `StartMotorDrive()` will be implemented, which literally starts the drive of the three motors. The testing of `movementDone` variable implies that all the motors must be moved to their destination before feeding the next sets of data for the subsequent movements. The loop will be continually running, executing the tasks mentioned above until coordinate movements were completed. Refer to section Appendix C: Source Code Listing for the complete source code.

6. 3-Axis Control Performance

In order to show that the microcontroller can provide drive signals simultaneously, a logic analyzer is used to capture the drive signal when operating in different stepping modes.

Figure 6-1. Drive Signals for Three Motors from MCU Captured using Logic Analyzer

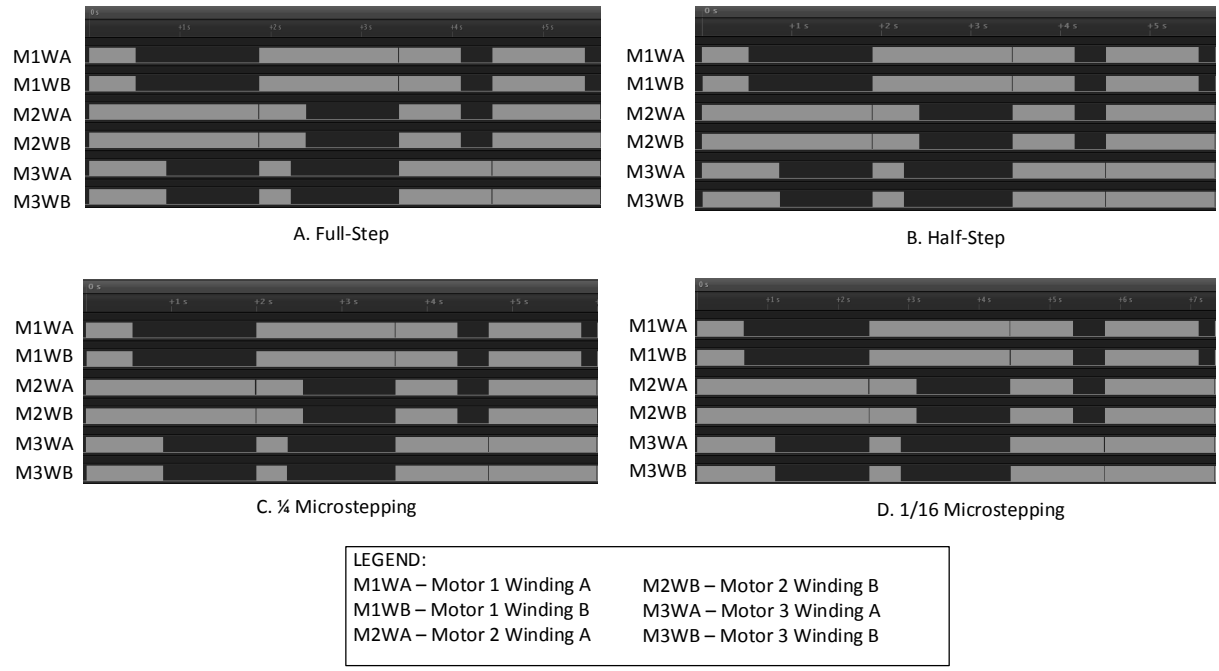


Figure 6-2. Drive Signal Step Mode Comparison

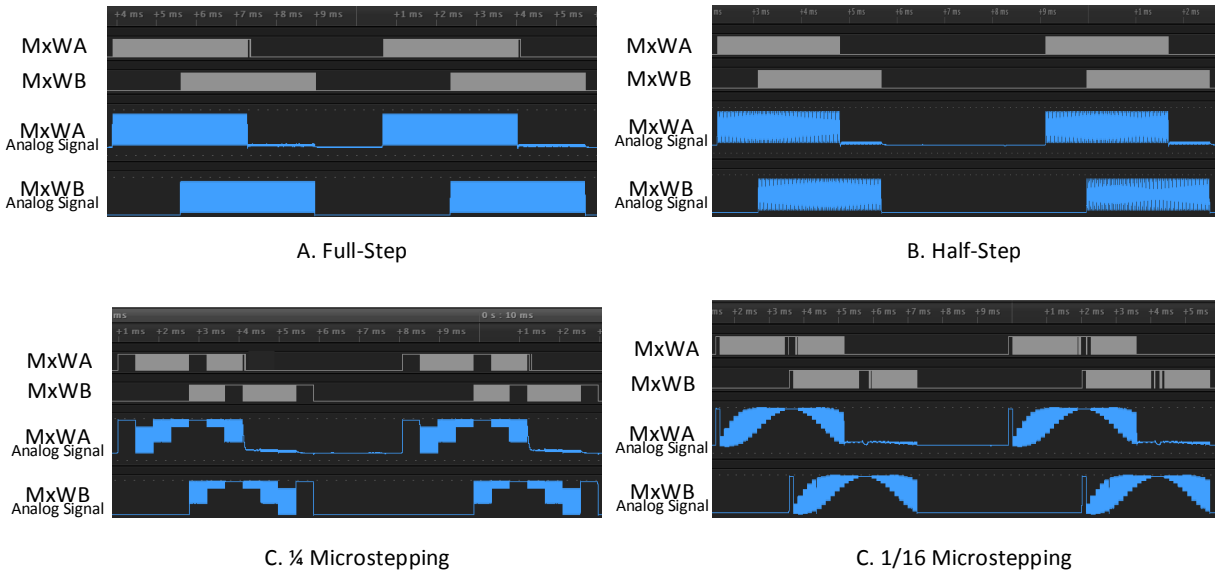


Figure 6-3. Data Coordinates Feed to the Motor

```
x_position []= {0,3,  12, 8,  2, 10,  2, 12,  9,  5, 12,  3, 9,  3, 12, 11, 2, 10,  2,  5,  11, 0};
y_position [] ={0,11, 8 ,12,  5, 9,  12,  8,  1,  4,  9,  4, 2, 11,  8, 12, 5,  9, 12,  8,   1, 0};
z_position [] ={0,5,  3 , 9,  2, 11,  8,  3,  8,  10, 6,  3, 7,  4, 11,  9, 6, 11,  8,  3,   8, 0};
```

Figure 6-1 shows the movement of motors in 3-axis in terms of number of steps driven at full-step, half-step and microstepping (1/4 and 1/16) with the speed of 180 RPM. It can be observed that the time elapsed for all step mode implementations is uniform because they all operate at 180 RPM. But, it can be seen in Figure 6-2 that the required PPS (Pulse per Second) for each drive is different to retain the speed of 180 RPM. The drive signals shown are from the control signals of low-side MOSFETs of bipolar drive circuits.

An example is shown in Figure 6-3, in which the positions are defined as an array of coordinates. At an instance, the positions having the same element order for all the axis will be processed by the firmware using Equations 5-1 or 5-2 depending on the preceding coordinate before passing onto Equation 5-3. Table 6-1 shows the data of how the desired steps are calculated for all the motors with *STEPS_PER_COORDINATE* of 100, based on the coordinate given in Figure 6-3.

Table 6-1. Step Calculation for Three Motors in Full-Step Mode

n	X_n	$\lvert X_{n+1} - X_n \rvert$	M1desiredStep	Y_n	$\lvert Y_{n+1} - Y_n \rvert$	M2desiredStep	Z_n	$\lvert Z_{n+1} - Z_n \rvert$	M3desiredStep
0	0			0			0		
1	3	3(CW)	300	11	11(CW)	1100	5	5(CW)	500
2	12	9(CW)	900	8	3(CCW)	300	3	2(CCW)	200
3	8	4(CCW)	400	12	4(CW)	400	9	6(CW)	600
4	2	6(CCW)	600	5	7(CCW)	700	2	7(CCW)	700

7. Conclusion

PIC18F-Q43 devices have the capacity to provide control signals for 3-axis stepper motor control. Aside from providing control signals, it is also capable of setting the movement limitation to accurately move the motors to the defined positions. Different modes of stepping drive such as full-step, half-step, 1/4 and 1/16 microstepping can be implemented to achieve the desired resolution. The CIPs such as CWG, 16-bit PWM and DMA significantly reduced the burden on the core, enabling the CPU to process data for motor control movements.

8. Appendix A: Schematics

Figure 8-1. Circuit Schematics

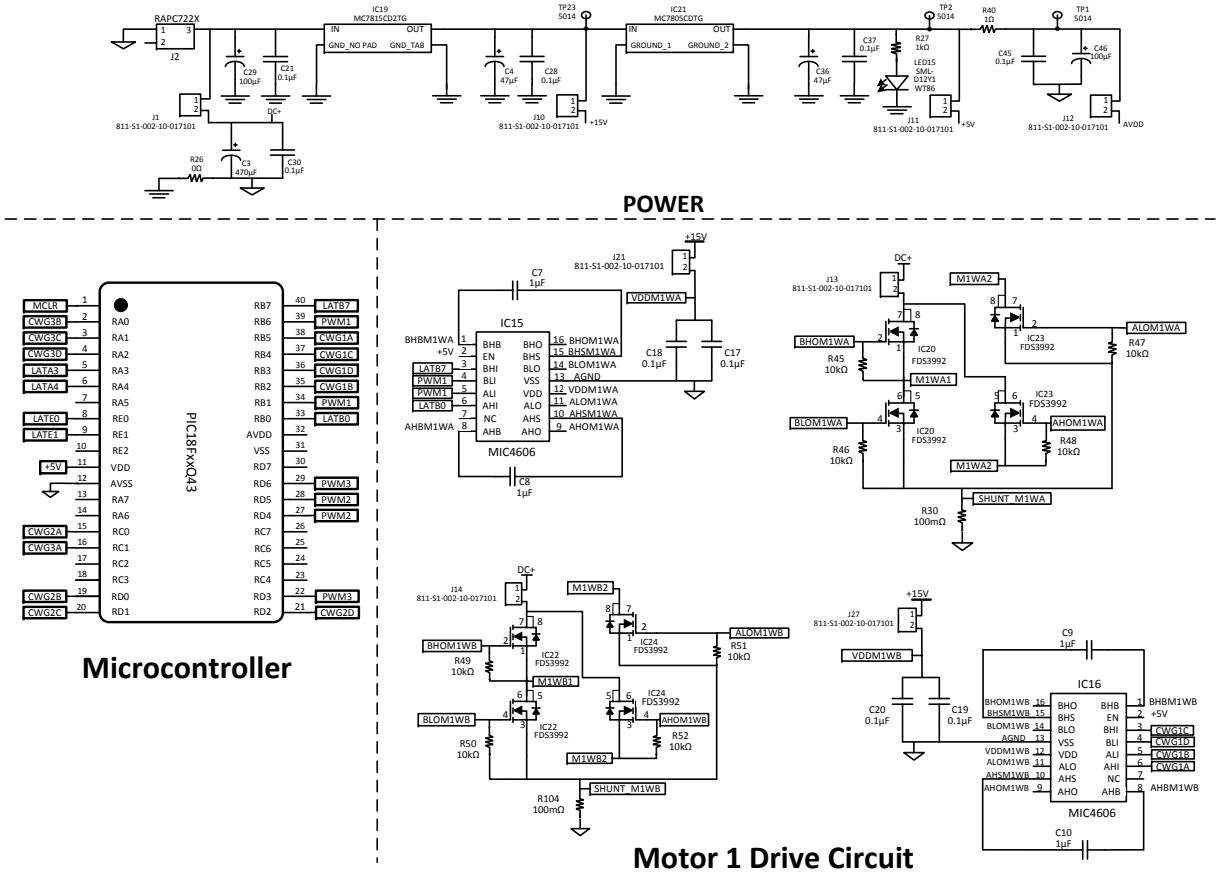
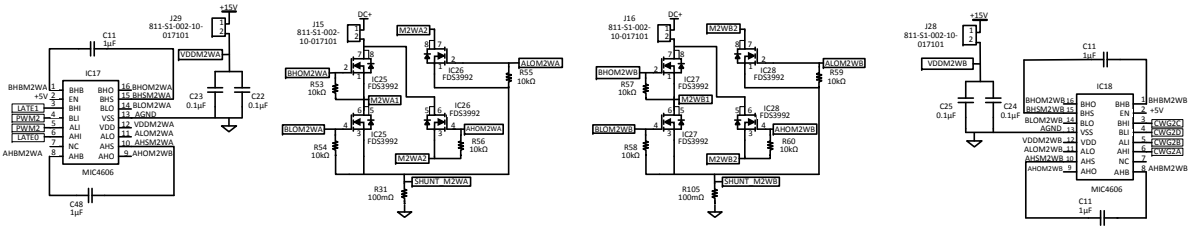


Figure 8-2. Motor 2 and 3 Drive Circuit Schematics



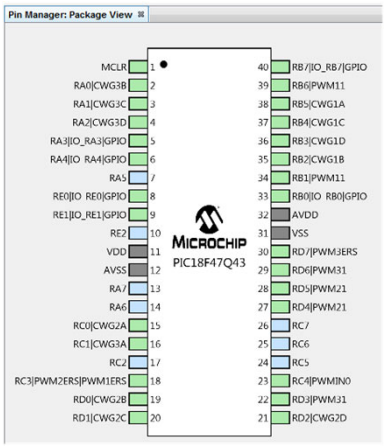
9. Appendix B: MPLAB® Code Configurator (MCC) Peripheral Initialization

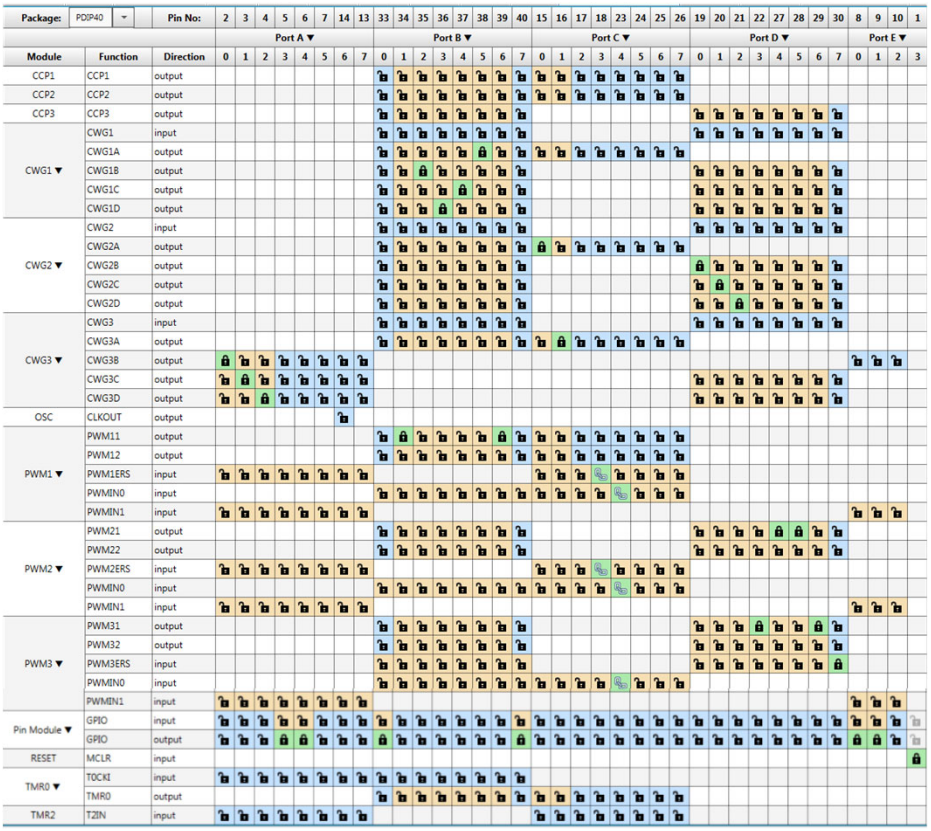
MPLAB® Code Configurator (MCC) is an easy-to-use plugin tool for MPLAB® X IDE that generates codes for controlling the peripherals of Microchip microcontrollers, based on the settings made in its Graphical User Interface (GUI). MCC is utilized to easily configure the peripherals used in this motor control 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 this application is listed below.

1. In the system module, set the clock to HFINTOSC with the highest available frequency of 64 MHz.
2. Timer2 is used as a clock source for CCP1/2/3. For the CCP to produce PWM, the Timer2 clock source must be set to $F_{OSC}/4$. Enable the Timer.
3. Configure the Timer0 in 16-bit Timer mode with MFINTOSC clock source, having a requested period of 1.5 ms. Enable the Timer interrupt and let the Timer0 be initially disabled.
4. For Motor 1 drive, configure PWM1, CCP1, CWG1, DMA Channel 1 and DMA Channel 2.
   - 4.1. Set up the PWM1 with HFINTOSC clock source, having a requested frequency of 62.5 kHz and duty cycle of 50%. Enable the PWM module.
   - 4.2. CCP1 module must be in PWM mode with Timer2 as the selected Timer, having a duty cycle of 50%.
   - 4.3. 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 CWG1.
5. For Motor 2 drive, configure PWM2, CCP2, CWG2, DMA Channel 3 and DMA Channel 4.
   - 5.1. Set up the PWM2 with HFINTOSC clock source, having a requested frequency of 62.5 kHz and duty cycle of 50%. Enable the PWM module.
   - 5.2. CCP2 module must be in PWM mode with Timer2 as the selected Timer, having a duty cycle of 50%.
   - 5.3. 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 CWG2.
6. For Motor 3 drive, configure PWM3, CCP3, CWG3, DMA Channel 5 and DMA Channel 6.
   - 6.1. Set up the PWM3 with HFINTOSC clock source, having a requested frequency of 62.5 kHz and duty cycle of 50%. Enable the PWM module.
   - 6.2. CCP3 module must be in PWM mode with Timer2 as the selected Timer, having a duty cycle of 50%.
   - 6.3. CWG3 must be configured with CCP3_OUT as an input source, with Output mode in Forward Full-Bridge mode and HFINTOSC as the selected clock source. Enable CWG3.
7. The DMA channels are configured by following these series of steps:
   - 7.1. In the DMAxCON0 register, enable the SIRQEN bit of DMA Channel 1 and clear the EN bit (disable).
   - 7.2. Define the DMAxDSZL size to 0x02, which means that the destination is 2 bytes wide.
   - 7.3. Configure the DMAxSIRQ to TMR0. Depending on the microstepping resolution, DMAxSSZL will be 0x80 if the mode will be 1/16 microstepping and 0x20 if it will operate in 1/4 microstepping. The source and destination addresses are defined in the firmware, depending on the direction of the motor and motor number. Leave the configuration of other registers not mentioned here, as is.
8. In the Pin Manager configuration, set up the input/output pins of all the peripherals as shown in Figure B-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 9-1. PIC18FXXQ43 Pin Manager Configuration





10. 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 attached to the electronic version of this application note. The latest version is v1.0.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

Product Change Notification Service

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

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website 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, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks 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.

ISBN: 978-1-5224-5479-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820