
Brushless DC Fan Speed Control Using Temperature Input and Tachometer Feedback

Introduction

Author: Christopher Best, Microchip Technology Inc.

Electronic circuits dissipate energy in the form of heat. As voltage forces free electrons to move, the electrons' kinetic energy increases. As the electrons pass through a material, the material may restrict the flow, causing electrons to crash into each other. When the electrons crash into each other, some of the kinetic energy is released in the form of heat. Complex electrical components, such as a microprocessor, may contain billions of transistors which are interconnected through resistive materials. Inevitably these devices get hot due to the transfer of energy. As the workload of a component increases, the resistive losses also increase, resulting in increased thermal temperatures. If the internal circuitry gets too hot, degradation of the materials may cause the device to fail. One simple way to combat this is to use a brushless DC (BLDC) fan to move the heat away from the device.

Many of today's BLDC fans are designed such that as the thermal temperature increases, the fan's speed also increases to provide more airflow to the hot component, which in turn removes more heat.

This application note discusses the use of the PIC18-Q43's Analog-to-Digital Converter with Computation (ADCC) module to monitor an object's temperature using a thermistor, individually adjust the speed of each of three fans based on the temperature input using the 16-bit Pulse-Width Modulator (PWM) module and ensure the fans are operating at the expected speed using the Compare/Contrast/PWM (CCP) module to connect to each fan's tachometer output. The temperature of the object and the speed of each fan is transmitted using the Universal Asynchronous Receiver Transmitter (UART) module to a serial terminal for viewing.

Table of Contents

Introduction.....	1
1. Project Description and Theory of Operation.....	3
1.1. Project Description.....	3
1.2. Theory of Operation.....	11
2. BLDC Basics.....	18
2.1. Rotor Assembly.....	18
2.2. Stator.....	19
2.3. Frame.....	20
2.4. Motor Drive Circuit.....	21
3. Conclusion.....	23
The Microchip Website.....	24
Product Change Notification Service.....	24
Customer Support.....	24
Microchip Devices Code Protection Feature.....	24
Legal Notice.....	24
Trademarks.....	25
Quality Management System.....	25
Worldwide Sales and Service.....	26

1. Project Description and Theory of Operation

1.1 Project Description

This project was designed to individually control the speed of three axial fans using a thermistor to detect temperature, three independent PWM outputs to set the speed of each fan respectively, and tachometer feedback to ensure each fan is operating at the desired speed.

1.1.1 Project Components

This project uses the following components:

- Microchip PIC18F57Q43 Curiosity Nano Development Board (Part No. DM164150)
- Microchip Curiosity Nano Base for Click Boards™ (Part No. AC164162)
- Microchip MCP2200 USB-to-UART Breakout Module (Part No. ADM00393)
- Sanyo Denki San Ace 40 4-Wire Axial Fan (Part No. 9GA0405P6F001)
- DROK 10k ohm B3950 NTC Temperature Sensor Probe (Part No. B01MR37GOQ)
- 10k ohm through-hole resister 1%
- 5V Power Supply
- Jumper wires

1.1.2 Peripherals Overview

This project utilizes four peripherals contained in the PIC18F57Q43 microcontroller:

- Analog-to-Digital Converter with Computation (ADCC)
- 16-bit Pulse-Width Modulator (PWM)
- Capture/Compare/PWM (CCP)
- Universal Asynchronous Receiver Transmitter (UART)

1.1.2.1 ADCC Overview

The PIC18F57Q43's ADCC module is a 12-bit Analog-to-Digital Converter that contains a computation block, which allows mathematical manipulation and threshold testing of the ADC conversion result. These computation modes include:

- Basic mode
- Accumulate mode
- Average mode
- Burst-Average mode
- Low-Pass Filter (LPF) mode

Basic mode can be considered as a legacy mode and operates similarly to a standard ADC. The ADC will perform a conversion, but no filtering or averaging is performed on the result. Threshold comparisons are performed after each conversion.

Accumulate mode allows multiple conversion results to be accumulated or added together. After each ADC trigger event occurs (the GO bit is set by either software or an external source), the conversion results are added to the ADC Accumulator (ADACC) register. Next, the ADC Counter (ADCNT) register increments by one and the new accumulator result is right-shifted (divided) by the value of the ADC Repeat Count (ADRPT) register. The right-shifted accumulator result is then stored in the ADC Filter (ADFLTR) register and a threshold comparison is performed on the ADFLTR result.

Average mode operates similarly to Accumulate mode. However, when the number of accumulated samples matches the value of ADRPT, a threshold comparison is performed on the ADFLTR value. Upon the next ADC trigger event, the accumulator is cleared.

Burst-Average mode operates much like Average mode, except that in Burst-Average mode, a single ADC trigger event causes the ADC to accumulate samples until the number of samples matches the ADRPT value. When the ADC trigger event occurs, the accumulator is cleared and the GO bit will remain set until the desired number of

samples have been accumulated. Once ADCNT matches ADRPT, hardware clears the GO bit, the accumulator is right-shifted by the value of ADRPT and a threshold comparison is performed on the new ADFLTR value.

Low-Pass Filter mode operates similar to Average mode, until ADCNT is equal to ADRPT and then hardware performs a threshold comparison on the ADFLTR value. Instead of clearing the accumulator at the next ADC trigger event, the new conversion result is added to the current accumulator value, the right-shift occurs and another threshold test is performed. The filtering process will continue as long as new trigger events occur or until a threshold violation is observed.

This project will use Burst-Average mode to monitor the thermistor. ADRPT is set such that an average of 32 samples is performed for each trigger event. [Example 1-1](#) shows the ADCC initialization routine used for this project.

Example 1-1. ADCC Initialization Code

```
void ADCC_Initialize(void)
{
    ADLTHL = 0x32;                      // Lower threshold set at 50 (0x0032)
    ADLTHH = 0x00;
    ADUTHL = 0xA0;                      // Upper threshold set at 4000 (0x0FA0)
    ADUTHH = 0x0F;
    ADSTPTL = 0x00;                     // Setpoint set to 0
    ADSTPTH = 0x00;
    ADACCU = 0x00;
    ADRPT = 0x20;                      // Set for 32 samples
    ADPCH = 0x00;
    ADCON1 = 0x00;                      // DSEN disabled;
    ADCON2 = 0x53;                      // CRS 5; Burst_average_mode
    ADCON3 = 0x57;                      // CALC Filtered value vs setpoint
    ADSTAT = 0x00;
    ADREF = 0x00;                      // NREF VSS; PREF VDD
    ADACT = 0x05;
    ADCLK = 0x3F;                      // CS FOSC/128;
    ADCONO = 0x84;                      // FM right; ON enabled; CS FOSC/ADCLK

    PIR2bits.ADTIF = 0;                // Clear the ADC Threshold interrupt flag
    PIE2bits.ADTIE = 1;                // Enabling ADCC threshold interrupt

    ADCC_SetADTIInterruptHandler(ADCC_DefaultInterruptHandler);
}
```

1.1.2.2 16-Bit Pulse-Width Modulator (PWM) Overview

The PWM module offers multiple outputs and a compare feature. The PIC18F57Q43 microcontroller contains three PWM modules. Each module offers a single output slice, which is composed of two PWM output channels. Both output channels have independent duty cycles which are set with their respective parameter registers, but share the same operating mode. The five main operating modes include:

- Left-Aligned mode
- Right-Aligned mode
- Center-Aligned mode
- Variable-Aligned mode
- Compare modes

The PWM period for Left-Aligned, Right-Aligned, Variable-Aligned and Compare modes can be calculated using [Equation 1-1](#).

Equation 1-1. PWM Period

$$\text{PWM period} = \frac{(P\text{WM}x\text{PR} + 1)}{f_{P\text{WM}x\text{CLK}}}$$

The PWM period for Center-Aligned mode can be calculated using [Equation 1-2](#).

Equation 1-2. PWM Period (Center-Aligned Mode)

$$\text{PWM period} = \frac{(P\text{WM}x\text{PR} + 1) \times 2}{f_{P\text{WM}x\text{CLK}}}$$

The PWM frequency for all modes can be calculated using [Equation 1-3](#).

Equation 1-3. PWM Frequency

$$f_{PWM} = \frac{1}{PWM \text{ period}}$$

The PWM duty cycle for Left-Aligned, Right-Aligned and Center-Aligned modes can be calculated using [Equation 1-4](#).

Equation 1-4. PWM Duty Cycle

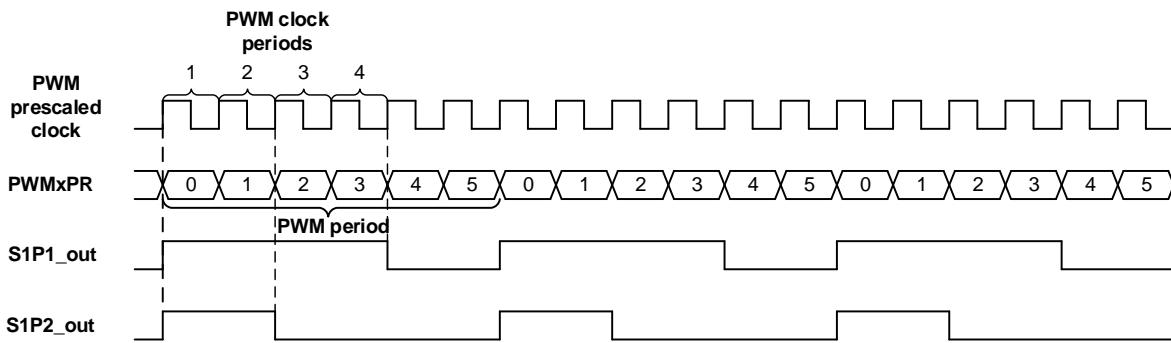
$$\text{PWM Duty Cycle} = \frac{PWMxS1Px}{PWMxPR + 1} \times 100$$

Note: The equation result represents the duty cycle as a percentage of the active output state relative to the total PWM period.

1.1.2.2.1 Left-Aligned Mode

In Left-Aligned mode, the active part of the duty cycle is at the beginning of the PWM period. The outputs start each period in the Active state and remain active for the number of prescaled PWM clock periods specified by each output's respective parameter register (PWMxS1P1, PWMxS1P2), and then transition to the Inactive state for the remainder of the PWM period. [Figure 1-1](#) shows the PWM outputs operating in Left-Aligned mode.

Figure 1-1. PWM Left-Aligned Mode



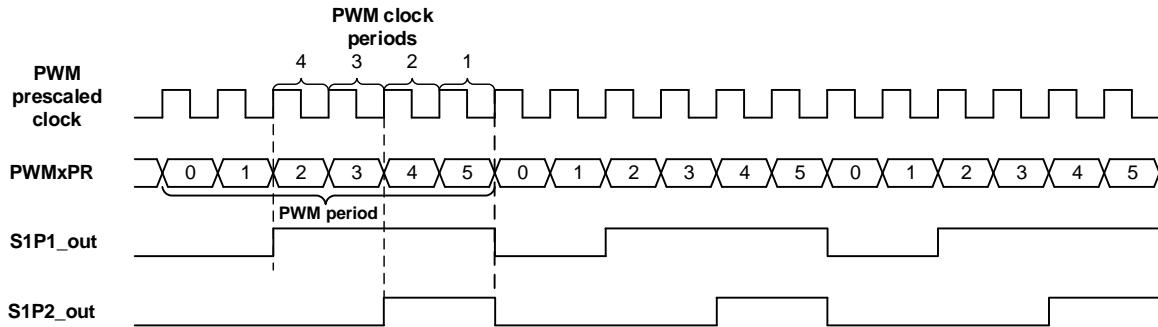
Note:

- PWMxPR = 5
- PWMxS1P1 = 4
- PWMxS1P2 = 2

1.1.2.2.2 Right-Aligned Mode

In Right-Aligned mode, the active part of the duty cycle is at the end on the PWM period. The outputs start each period in the Inactive state. It then goes into the Active state for the number of prescaled PWM clock periods before the end of the PWM period, as specified by each output's parameter register. [Figure 1-2](#) shows the PWM outputs operating in Right-Aligned mode.

Figure 1-2. Right-Aligned Mode



Note:

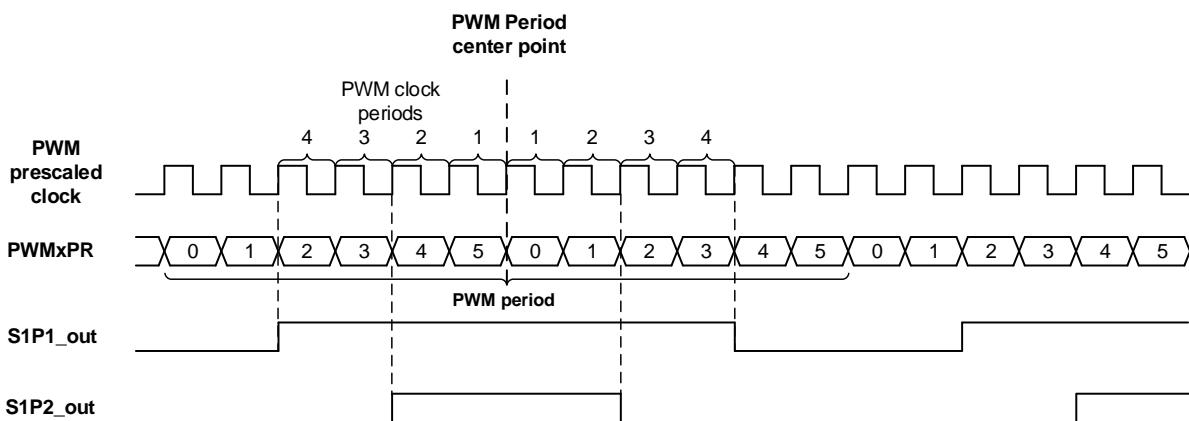
- $\text{PWMxPR} = 5$
- $\text{PWMxS1P1} = 4$
- $\text{PWMxS1P2} = 2$

1.1.2.2.3 Center-Aligned Mode

In Center-Aligned mode, the active part of the duty cycle is centered in the PWM period. The PWM period for Center-Aligned mode is twice that of the other modes as shown in [Equation 1-2](#).

Each respective output's parameter register specifies the number of prescaled PWM clock periods that the output goes to the Active state before the PWM period center. The output will then go to the Inactive state the same number of prescaled PWM clock cycles after the PWM period center. [Figure 1-3](#) shows the PWM outputs operating in Center-Aligned mode.

Figure 1-3. PWM Center-Aligned Mode



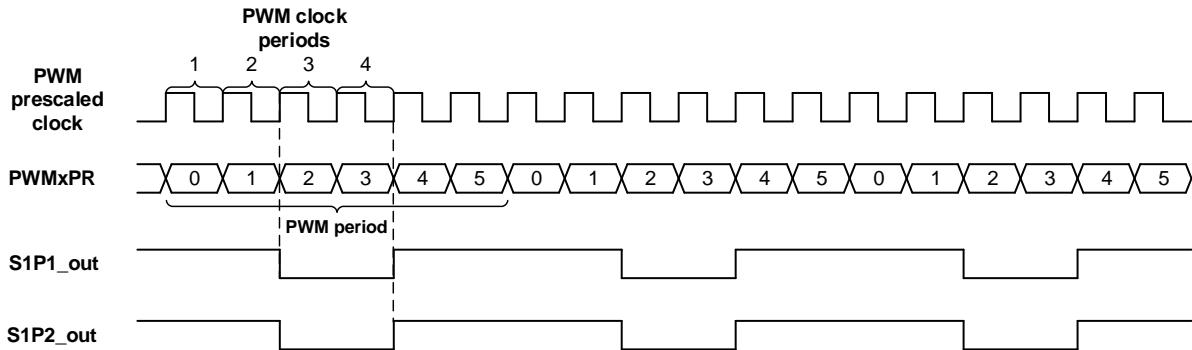
Note:

- $\text{PWMxPR} = 5$
- $\text{PWMxS1P1} = 4$
- $\text{PWMxS1P2} = 2$

1.1.2.2.4 Variable-Aligned Mode

In Variable-Aligned mode, the active part of the duty cycle begins when the parameter 1 value matches the number of prescaled PWM clock cycles and ends when the parameter 2 value matches the number of prescaled PWM clock cycles. In this mode, both outputs are identical because both parameter values are used for the same duty cycle. [Figure 1-4](#) shows the outputs operating in Variable-Aligned mode.

Figure 1-4. PWM Variable-Aligned Mode



Note:

- $\text{PWMxPR} = 5$
- $\text{PWMxS1P1} = 4$
- $\text{PWMxS1P2} = 2$

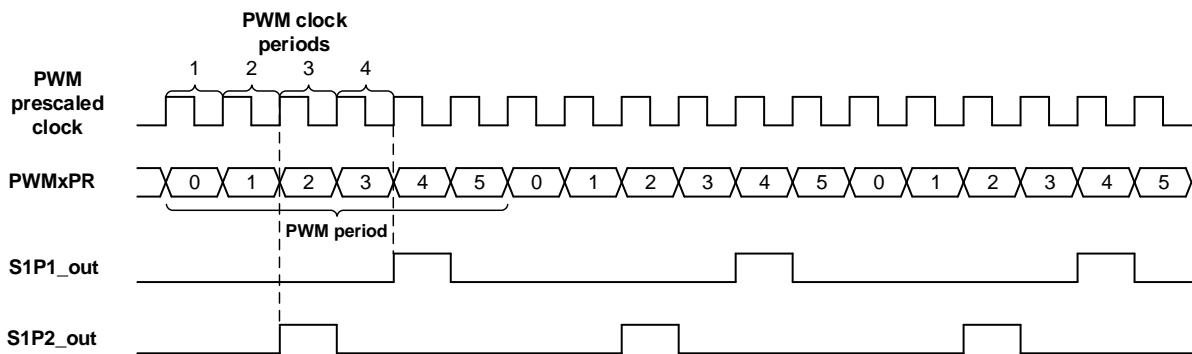
1.1.2.2.5 Compare Modes

The PWM module has built-in support for two Compare modes: Pulsed Compare and Toggled Compare. In both Compare modes, the number of prescaled PWM clock cycles is compared to the P1 and P2 parameter values. When a match occurs, the output is either pulsed or toggled.

Pulsed Compare Mode

In Pulsed Compare mode, the outputs begin in the Inactive state. When the number of prescaled PWM clock cycles matches each output's parameter value, the outputs are pulsed to the Active state for one prescaled PWM clock cycle and then returns to the Inactive state for the remainder of the PWM period. [Figure 1-5](#) shows the outputs operating in Pulsed Compare mode.

Figure 1-5. PWM Pulsed Compare Mode



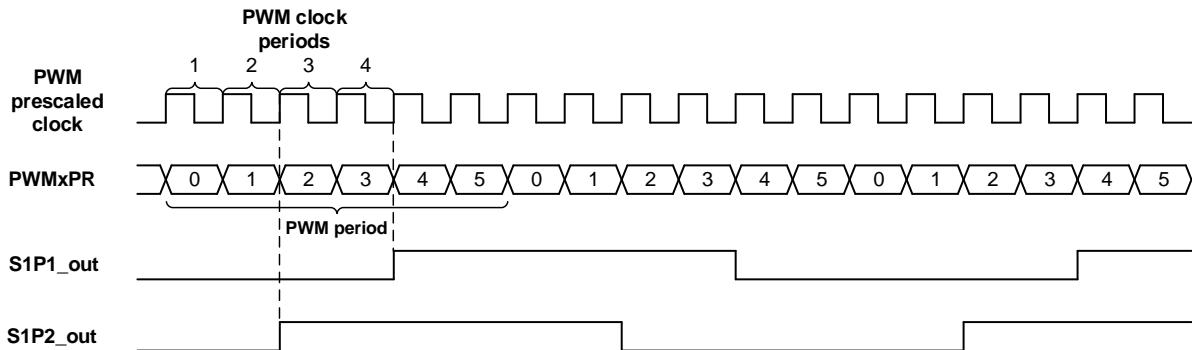
Note:

- $\text{PWMxPR} = 5$
- $\text{PWMxS1P1} = 4$
- $\text{PWMxS1P2} = 2$

Toggled Compare Mode

In Toggled Compare mode, the duty cycle is alternating full PWM periods. Each output goes active when the number of prescaled PWM clock cycles matches the respective parameter value, and goes inactive at the same match point in the next PWM period. [Figure 1-6](#) shows the outputs operating in Toggled Compare mode.

Figure 1-6. PWM Toggled Compare Mode



Note:

- PWMxPR = 5
- PWMxS1P1 = 4
- PWMxS1P2 = 2

For this project, the Left-Aligned mode was used to control the speed of each individual fan. As the thermistor temperature changes, each output's duty cycle is adjusted as required to match the cooling needs. [Example 1-2](#) shows the PWM initialization sequence.



Attention: The PWM duty cycle adjustments are based on both temperature and tachometer inputs. The actual duty cycle adjustments are preformed in the CCP interrupt service routine, which is covered in the Compare/Contrast/PWM (CCP) section.

Example 1-2. PWM Initialization

```
void PWM1_Initialize(void)
{
    PWM1ERS = 0x00;                      // PWMERS External Reset Disabled
    PWM1CLK = 0x02;                      // PWMCLK FOSC
    PWM1LDS = 0x00;                      // PWM LDS Autoload disabled
    PWM1PRL = 0x00;                      // 0x0A00=25 kHz PWM freq @ 64 MHz FOSC
    PWM1PRH = 0x0A;                      //
    PWM1CPRE = 0x00;                     // PWMCPRE No prescale
    PWM1PIPOS = 0x00;                     // PWMPIPOS No postscale
    PWM1GIR = 0x00;                      // Clear PWM general interrupt flags
    PWM1GIE = 0x00;                      // PWMS1P2IE disabled; PWMS1P1IE disabled
    PWM1S1CFG = 0x00;                     // PWM left aligned mode
    PWM1S1P1L = 0x00;
    PWM1S1P1H = 0x05;
    PWM1S1P2L = 0x00;
    PWM1S1P2H = 0x05;
    PWM1CON = 0x80;                      // PWMEN enabled
}
```

1.1.2.3 Capture/Compare/PWM (CCP) Module Overview

The Capture/Compare/PWM (CCP) module is a peripheral that can be used to measure time, control certain events and generate PWM signals. In Capture mode, the peripheral allows time measurements of events. Compare mode allows the user to trigger an external event when a predetermined amount of time has expired. PWM mode is used to generate PWM signals of varying frequency and duty cycle.

1.1.2.3.1 Capture Mode

Capture mode makes use of the 16-bit odd number timer resources, such as Timer1. When an edge event occurs on the capture source, the 16-bit CCPRx register captures and stores the 16-bit timer value. The trigger event may come from an internal source, such as a CLC output, or from an external source selected by the CCPxPPS input pin.

A capture event can be triggered:

- Every falling edge of the CCP input
- Every rising edge of the CCP input
- Every 4th rising edge of the CCP input
- Every 16th rising edge of the CCP input
- Every edge of the CCP input (both rising and falling)

When a capture is made, the CCP Interrupt Flag (CCPxIF) is set and must be cleared in software.



Important: If another capture event occurs before the value in the CCPRx register is read, the old value is overwritten by the new value. If the event occurs during a 2-byte read, the CCPRxH byte and CCPRxL byte data will be from different events. It is recommended to either disable the module while reading the CCPRx register pair, or to read the register pair twice to ensure data validity.

1.1.2.3.2 Compare Mode

Similar to Capture mode, Compare mode makes use of the 16-bit odd numbered Timer resources, such as Timer1. The 16-bit value loaded into the CCPRx register is constantly compared against the 16-bit value of the timer register. When a match occurs, one of the following events can occur based on the configuration of the CCP Mode Select (MODE) bits:

- Toggle the CCP output and clear the associated Timer register
- Toggle the CCP output without clearing the associated Timer register
- Set the CCP output
- Clear the CCP output
- Generate a Pulse output
- Generate a Pulse output and clear the associated Timer register

Additionally, when a match occurs, the CCP Interrupt Flag (CCPxIF) is set, which can also be used to trigger an ADC conversion. Software must clear the CCPxIF bit.

1.1.2.3.3 PWM Mode

The PWM contained in the CCP module operates as a 10-bit Pulse-Width Modulator and makes use of the 8-bit even numbered Timer resources, such as Timer2. The PWM is typically used to control power to a load by quickly switching between fully ON and fully OFF states. The high portion (pulse-width) of the PWM period is considered the ON or Active state, while the low portion of the PWM period is considered the OFF or Inactive state. The pulse-width can vary in time and is defined in terms of steps. As the number of steps increases, the width of the pulse increases and a higher amount of power is supplied to the load. As the number of steps decrease, the width of the pulse decreases and less power is applied to the load. The PWM period is defined as the duration of one complete cycle.

The PWM resolution defines the maximum number of steps that can be present in a single PWM period. A higher resolution allows for more precise control of the pulse-width time. In a 10-bit PWM, there are up to 1024 steps possible in a single PWM period, depending on the configuration of the duty cycle.

The PWM duty cycle describes the ratio of the pulse-width to the total PWM period and is expressed in percentages, where 0% is considered fully Off and 100% is considered fully On. A lower duty cycle corresponds to less power applied to the load, while a higher duty cycle corresponds to more power applied to the load.

For this project, the CCP is used in Capture mode to measure the fan tachometer output using Timer1 as the capture time base. The CCP is also configured to capture every rising edge of the tachometer output. [Example 1-3](#) shows the CCP initialization code.

Example 1-3. CCP Capture Mode Initialization

```
void CCP1_Initialize(void)
{
    CCP1CON = 0x85;                                // MODE every rising edge
    CCP1CAP = 0x00;                                // CCP1CTS CCP1 pin
    CCPR1H = 0x00;
    CCPR1L = 0x00;
    CCPTMRS0bits.C1TSEL = 0x1;                     // Select Timer 1
    PIR3bits.CCP1IF = 0;                            // Clear CCP1IF
    PIE3bits.CCP1IE = 1;                            // Enable CCP1 interrupt
}
```

1.1.2.4 Universal Asynchronous Receiver Transmitter (UART) Module Overview

The UART module is a serial communications peripheral. The UART contains all the clock generators, shift registers and data buffers required to perform an input or output serial data transfer, independent of device program execution. The UART can be configured as a full-duplex asynchronous system, or one of several automated protocols, such as LIN. The UART can also be configured in Half-Duplex mode, which allows for either Transmission Only or Receive Only modes.

The UART has five asynchronous modes:

- 7-bit
- 8-bit
- 8-bit with even parity in the 9th bit
- 8-bit with odd parity in the 9th bit
- 8-bit with an address indicator in the 9th bit

The UART transmits and receives data using the standard Non-Return-to-Zero (NRZ) format. NRZ implies that consecutively transmitted data bits of the same value stay at the output level of that bit without returning to a Neutral state between each bit transmission. NRZ is implemented with two logic levels:

- Voltage output high (V_{OH}) Mark state, which represents a logic '1' data bit
- Voltage output low (V_{OL}) Space state, which represents a logic '0' data bit

An NRZ transmission port idles in the Mark state. Each character transmission consists of one Start bit, followed by seven or eight data bits, one optional parity or address bit and is always terminated with one or more Stop bits. The Start bit is always a space and Stop bits are always marks. The most commonly used format is 8-bits with no parity. Each transmitted bit persists for a period of 1/Baud Rate.

In all asynchronous modes, the UART transmits and receives the Least Significant bit (LSb) first. The UART's transmitter and receiver are fully independent, but share the same data format and baud rate.

For this project, the UART is used as a transmitter only. Temperature and fan speeds are transmitted to a PC terminal so that the user can monitor fan operation. [Example 1-4](#) shows the UART initialization and transmit routines. Additionally, the 'printf' library functions are used as a fast way to transmit characters in a string format.

Example 1-4. UART Initialization and Transmit Routines

```
void UART1_Initialize(void)
{
    U1CON0 = 0xA0;                                // BRGS HS; TXEN enabled
    U1CON1 = 0x80;                                // ON enabled;
    U1CON2 = 0x00;                                // TXPOL not inverted; 1 Stop bit
    U1BRGL = 0xC7;                                // 9600 BR @ 64 MHz FOSC
    U1BRGH = 0x06;
    U1FIFO = 0x00;
    U1UIR = 0x00;                                // Auto-baud not enabled
    U1ERRIR = 0x00;
    U1ERRIE = 0x00;
    UART1_SetFramingErrorHandler(UART1_DefaultFramingErrorHandler);
    UART1_SetOverrunErrorHandler(UART1_DefaultOverrunErrorHandler);
    UART1_SetErrorHandler(UART1_DefaultErrorHandler);
    uart1RxLastError.status = 0;
}

void putch(char txData)                         // Used by 'printf' library
{
    UART1_Write(txData);
}

void UART1_Write(uint8_t txData)
{
    while(0 == PIR4bits.U1TXIF)                  // Wait until TX buffer is clear
    {
    }
    U1TXB = txData;                            // Write data byte to UART
}
```

1.2 Theory of Operation

In this application, three main tasks are performed:

- Acquire a temperature value
- Capture the actual fan speed and adjust PWM output
- Transmit the temperature value and actual fan speed

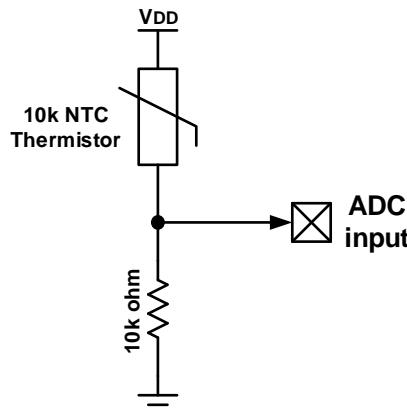
1.2.1 Acquire a Temperature Value

Temperature acquisition is accomplished using a 10k ohm Negative Temperature Coefficient (NTC) thermistor. An NTC thermistor is a type of non-linear resistor whose resistance decreases as the temperature increases. In terms of ADC voltage, as temperature increases, less thermistor resistance allows a higher voltage to reach the ADC input. **Table 1-1** shows the relation between temperature and the ADC readings. These values were taken by placing the thermistor in a temperature-controlled environment. The thermistor is configured as part of a voltage divider circuit (see [Figure 1-7](#)) and the output of the divider circuit is fed into an ADC input.

Table 1-1. Temperature vs ADFLTR Values

Temperature (°C)	ADFLTR Value
0	918
20	1800
25	2028
30	2258
35	2468
40	2668
45	2850
50	3012
55	3156
60	3280
65	3391
70	3488

Figure 1-7. Thermistor Voltage Divider Circuit



The project software is configured to read the thermistor input once every second. This one second delay between measurements was chosen for demonstration purposes only. Most applications will not see large variations in temperature so quickly, so temperature measurements may be taken at longer intervals. Measuring at longer intervals may also help prevent the fans from continuously changing speeds, which in turn may create unwanted audible noise.

For this application, the ADC is configured in Burst-Average mode, taking a quick burst of 32 samples of the ADC input and then averaging those samples. Burst-Average mode was chosen because of its simplicity; software only has to set the GO bit once to begin sampling and when all 32 samples have been acquired, hardware automatically clears GO. This saves instruction time, making the application more efficient.

Once all 32 samples have been taken and averaged, the average value is loaded into the ADFLTR register and a threshold test is performed. In this case, the threshold test calculates the error or difference between the ADFLTR value and a user-defined set point. Hardware then compares the error value to two user-defined thresholds: an upper threshold and a lower threshold. For this application, the threshold values were chosen based on two possible thermistor failure conditions. The lower threshold value was set at an ADC value of '50', while the upper threshold

value was set at an ADC value of '4000'. The lower threshold represents a thermistor open failure condition, in which the thermistor's resistance would theoretically be infinite, preventing any voltage to pass through the thermistor. Instead, the ADC input would be pulled to ground via the pull-down resistor of the voltage divider circuit. The upper threshold represents a thermistor short failure condition, in which the thermistor's resistance would be zero, allowing all of the voltage to reach the ADC input.

After the threshold test has completed, the ADC Threshold Interrupt Flag (ADTIF) bit is set. The ADC Threshold Interrupt can be configured to be triggered under a variety of conditions, such as a lower or upper threshold violation. For this project, it is set to always interrupt, regardless of the results of the threshold test. This method allows software to:

- Manually check the ADC Module Greater-than Upper Threshold (UTHR) and Lower-than Lower Threshold (LTHR) flags.
- Calculate the desired fan speed based on the ADC input.
- Calculate the thermistor temperature.

The UTHR and LTHR status flags will be set if the ADC error calculation value violates the upper or lower threshold values found in the respective ADC Upper Threshold (ADUTH) and ADC Lower Threshold (ADLTH) registers. In the event that either of these status flags are set, the fans are instructed to go to full speed and a message will be transmitted to the PC terminal program stating that a thermistor failure occurred.

If neither of the status flags are set, software calculates the desired fan speed using the line equation, which allows for linearization of the fan speed. [Equation 1-5](#) shows both the standard line equation and the equivalent line equation in terms of fan speeds and ADC values.

Equation 1-5. Line Equation

$$y = mx + b$$

where:

y = fan speed

$$m = \frac{(\text{Fan maximum speed} - \text{Fan minimum speed})}{(\text{ADFLTR} @ 70C - \text{ADFLTR} @ 20C)}$$

x = current ADFLTR value

$$b = \text{Fan maximum speed} - (m \times \text{ADFLTR} @ 70C)$$

After the fans' desired speeds are calculated, the thermistor temperature value is calculated. This calculation is accomplished using the Simplified β (beta) Parameter equation ([Equation 1-6](#)), which is a derivative of the Steinhart-Hart equation. The Simplified β Parameter equation uses terms that are easily attainable, such as the thermistor resistance value at 25°C and reduces the complexity of the calculations that software must perform. To further simplify the software calculations, several of the more math-intensive variables have been pre-calculated by hand and listed as software constants.

Equation 1-6. Simplified β Parameter Equation

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln\left(\frac{R}{R_0}\right)$$

where:

T = Temperature in Kelvin (K)

$T_0 = 298.15$ K (room temperature (25°C))

B = Thermistor coefficient (in this case 3950)

R = measured thermistor resistance (ADC value (in voltage) must be converted to resistance)

R_0 = Thermistor's rated resistance at 25°C (in this case 10k ohms)

[Example 1-5](#) shows the Interrupt Service Routine in which the threshold status flags are monitored, desired fan speed is calculated and thermistor temperature calculations are performed, as well as listing the pre-calculated constants used in the calculations. For this project, the fans are set to rotate at a fixed minimum speed of 1000 RPM

for temperatures 20°C and below, a fixed maximum speed of 8000 RPM at temperatures at or above 70°C (or in the event of a thermistor failure condition) and have a linear speed profile between 20°C and 70°C.

Example 1-5. ADC Threshold ISR

```

#define MAXADC      4095.000      // 12-Bit ADCC
#define K          273.1500      // Kelvin constant
#define LN          0.4343      // Log(e) equivalent

#define MIN_THERM_ADC 1800      // ADFLTR value at 20C
#define MAX_THERM_ADC 3488      // ADFLTR value at 70C
#define MIN_SPD      1000      // Fan min speed
#define MAX_SPD      8000      // Fan max speed
#define SLOPE_M      4.15      // = ((MAX_SPD - MIN_SPD) / (MAX_THERM_ADC - MIN_THERM_ADC))
#define B_OFFSET     -6475      // (MAX_SPD - (SLOPE_M * MAX_THERM_ADC))

void ADCC_DefaultInterruptHandler(void)
{
    float T0 = 0.00335;          // 1 / 298.15
    float B = 0.00025;          // 1 / 3950 (beta value)

    adc_val = ADCC_GetFilterValue(); // Get thermistor value
    if(adc_val <= MIN_THERM_ADC) // Is the temp at or below 20C?
    {
        if(ADSTATbits.LTHR == 0) // Lower threshold not violated
        {
            y_therm = MIN_SPD; // Set fan to min speed
            therm_short = 0;
            therm_open = 0;
        }
        else // Lower thresh violation
        {
            y_therm = MAX_SPD; // Set fan to max speed
            therm_open = 1; // Therm failed open
        }
    }
    else if(adc_val >= MAX_THERM_ADC) // Is the temp at or above 70C?
    {
        y_therm = MAX_SPD; // Set fan to max speed
        if(ADSTATbits.UTHR == 1) // Check for therm short
        {
            therm_short = 1; // Therm failed short
            therm_open = 0;
        }
    }
    else // 20C < Temp < 70C
    {
        // Speed based on y = mx + b line equation
        y_therm = SLOPE_M * adc_val + B_OFFSET;
        therm_short = 0;
        therm_open = 0;
    }

    temperature = (MAXADC/(float)adc_val)-1.00000; // R equiv of input voltage
    temperature = log10f(temperature); // Log of the temperature
    temperature = temperature / LN; // Converts log(x) to ln(x)
    temperature = B * temperature; // Multiply by B constant
    temperature += T0; // Add T0 constant
    temperature = 1 / temperature; // Invert to get Kelvin (K)
    temperature -= K; // Subtract K = temp in C

    PIR2bits.ADTIF = 0;
}

```

1.2.2 Capture Fan Speed and Adjust PWM

During PWM initialization, the PWM period is set at a 25 kHz frequency and a 50% duty cycle. The PWM frequency was selected to match the fans' expected input frequency, while the duty cycle was selected simply to allow the fans to begin rotating and provide a tachometer output signal while the first temperature calculations are being performed.

Once a temperature value has been acquired and the expected speed has been calculated, the CCP's Capture module begins to measure the fan tachometer output signal. The San Ace 40 Brushless DC (BLDC) model fans

output a two pulse per mechanical revolution tachometer signal (see [BLDC Basics](#) for a simplified explanation of BLDC fan operation). The CCP is configured to capture each rising edge of the tachometer signal, effectively measuring each tachometer period using Timer1 as a time base. Once module hardware detects the rising edge, CCP hardware records the Timer value and generates an interrupt event, in which the actual fan speed is calculated and fan speed adjustment takes place.

The CCP's capture Interrupt Service Routine begins by finding the difference between the last stored CCPR register value and the new CCPR register value. Since the Timer1 resource is shared between the three CCP modules used in this project, Timer1 is operating in Free Running mode and is not cleared or Reset in between capture events. This means that each CCP module must save the previous captured value and compare it to the new captured value. The difference between the two values is the tachometer period in terms of Timer counts, which must then be converted into speed in terms of RPM. [Equation 1-7](#) shows the calculations necessary to convert the Timer count value into RPM and uses the calculations for Fan 1.

Equation 1-7. Converting the Capture Value into RPM

$$fan1_spd = \frac{TIMER_CONST}{fan1_tmr_val}$$

where:

fan1_spd = fan 1's measured speed in RPM

$$TIMER_CONST = \frac{\text{Timer1 Clock}}{\text{Timer1 Prescaler}} \times \frac{60}{\text{Magnetic Pole Pairs}}$$

fan1_tmr_val = current Capture value – previous Capture Value

The converted tachometer signal represents the actual fan speed, which may or may not match the desired fan speed. After the actual fan speed has been calculated, it is compared to the desired fan speed. If the actual fan speed is less than the desired speed, the current PWM duty cycle value stored in the PWMxS1Py parameter register is incremented by one; if the actual speed is greater than the desired speed, the value stored in the parameter register is decreased by one. This process continues with each capture interrupt; the actual fan speed is calculated and compared to the desired speed and the duty cycle is either increased or decreased until the two match.

[Example 1-6](#) shows the capture ISR for CCP1, which monitors and corrects the speed of Fan 1. CCP2 and CCP3 use the same routines for their respective fans.

Example 1-6. Capture ISR for CCP1

```

#define TIMER_CONST 3750000
// = (TMR1 Clock / TMR1 Prescaler) * (60 /pole pairs)
// = (1 MHz / 8) * (60 / 2)

void CCP1_CaptureISR(void)
{
    CCP1_PERIOD_REG_T module;
    module.ccpr1l = CCPR1L;
    module.ccpr1h = CCPR1H;
    prevCapt1 = currentCapt1;
    currentCapt1 = module.ccpr1_16Bit; // Get new capture value

    if(currentCapt1 > prevCapt1)
    {
        fan1_tmr_val = currentCapt1 - prevCapt1; // Find tach period
    }
    else // Timer1 rolled over
    {
        fan1_tmr_val=(0xFFFF-prevCapt1)+currentCapt1;
    }

    fan1_spd = TIMER_CONST / fan1_tmr_val; // Calculate actual speed

    if(fan1_spd < y_therm) // Fan 1 too slow?
    {
        PWM1S1P1 += 1;
        PWM1CONbits.LD = 1; // Increase duty cycle
        // Latch new PWM value
    }
    if(fan1_spd > y_therm) // Fan 1 too fast?
    {
        PWM1S1P1 -= 1;
        PWM1CONbits.LD = 1; // Decrease duty cycle
        // Latch new PWM value
    }
    PIR3bits.CCP1IF = 0;
}

```

1.2.3 Transmit the Temperature Value and Actual Fan Speed

The final task performed by software is to transmit the thermistor temperature value and each fan's actual speed in RPM. Data transmission is handled by the UART, which is connected to a PC terminal program. This allows the user to monitor temperature and individual fan speed.

For this project, the UART is configured with a baud rate of 9600 bits per second (bps) and is operating in asynchronous 8-bit transmission mode with no Parity bit and one Stop bit. The PC terminal program must be configured the same way. [Figure 1-8](#) shows the PC terminal window during project operation. [Example 1-7](#) shows the main while loop, in which the UART's 'printf' statements are used to transmit temperature and fan speeds.

Figure 1-8. PC Terminal Output Window

	RPM1	RPM2	RPM3
1535	1493	1506	1506
22.73C			
1490	1535	1518	1518
22.76C			
1489	1487	1523	1523
22.73C			
1531	1487	1521	1521
22.71C			
1489	1529	1509	1509
22.78C			
1534	1485	1526	1526
22.71C			
1533	1533	1510	1510
22.71C			
1492	1488	1514	1514
22.73C			
1536	1488	1513	1513
22.73C			
1536	1531	1512	1512
22.73C			
1491	1488	1518	1518
22.76C			
1533	1535	1522	1522
RPM3			

Example 1-7. Project Main `while()` Loop

```
while (1)
{
    ADCC_StartConversion(Thermistor);                                // Read thermistor

    if(therm_short == 1)
    {
        printf("Thermistor shorted failure!\r\n");
    }
    if(therm_open == 1)
    {
        printf("Thermistor open failure!\r\n");
    }
    if((therm_open == 0) && (therm_short == 0))
    {
        printf("%1.2fC \r\n", temperature);                            // Display temp
    }
    // Display fan speeds
    printf("%d RPM1 %d RPM2 %d RPM3\r\n", fan1_spd, fan2_spd, fan3_spd);

    __delay_ms(1000);                                                 // 1 second delay
}
```

2. BLDC Basics

Brushless DC (BLDC) fans use electronic switches to control the amount of current passing through its motor in order to turn its rotor. BLDC fans rely on rotor position feedback so that the switches turn on or off in a manner that allows the motor to generate the proper amount of torque to make the rotor rotate. Rotor position feedback may come from a position sensor or by detecting the back electromotive force (back EMF). There are three BLDC motor classifications: single-phase, two-phase and three-phase.

BLDC fans consist of the following main components:

- Rotor
- Stator (motor)
- Frame (housing)
- Motor driver circuit

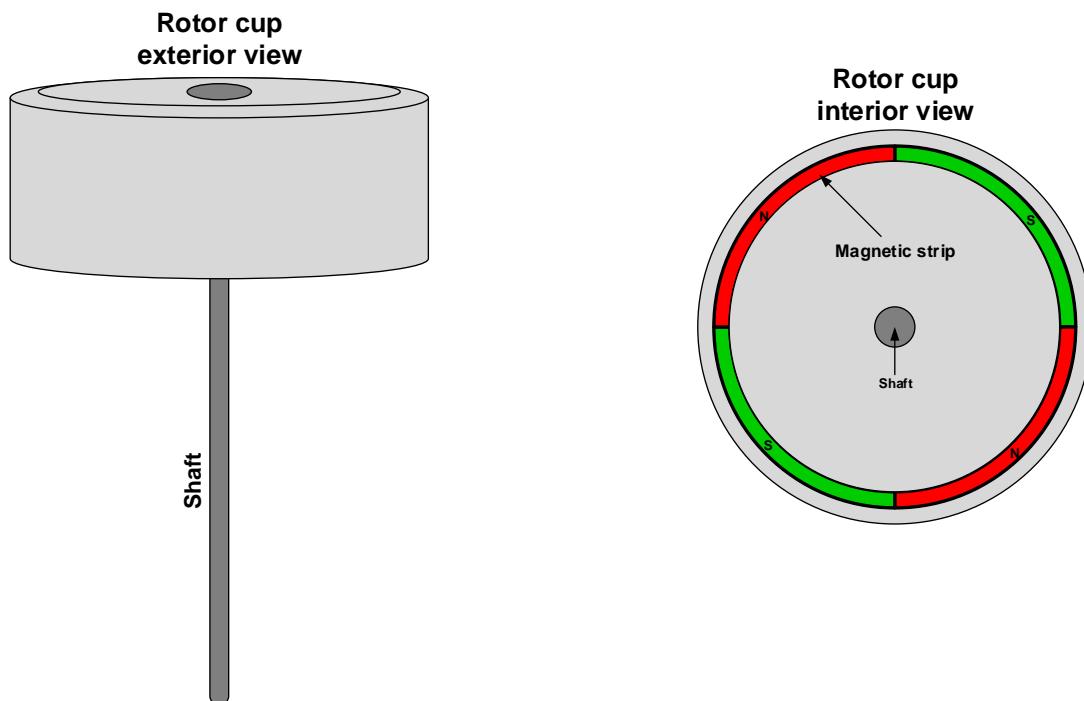
For this project, the fans use the single-phase motor type, and rely on Hall-effect sensors to monitor rotor positions. The fans are axial, meaning that the airflow is in parallel to the rotor shaft.

2.1 Rotor Assembly

The rotor of an axial fan consists of two main parts: A steel rotor cup and a plastic impeller.

The rotor cup is a circular-shaped steel assembly that has a long shaft centered in the inside of the cup area, which connects the rotor to the fan frame. A magnetic strip is fitted into the interior of the rotor cup, creating a magnetic field that interacts with the magnetic field produced by the motor. [Figure 2-1](#) shows a drawing of a rotor cup.

Figure 2-1. Rotor Cup



The impeller is a plastic piece that has flanges extending from the center and is typically glued or press-fitted onto the exterior surface of the rotor cup, creating the rotor assembly (see [Figure 2-2](#)). The flanges are designed to meet specific airflow characteristics and as the rotor assembly turns, the angle of each flange allows it to move air.

After the impeller is fitted onto the rotor cup, the whole assembly must be balanced. This may be performed using either single-plane or dual-plane balancing, which is based on two factors:

- The length-to-diameter ratio of the rotor cup (not including the shaft)
- The rotational speed of the rotor

When the rotor is improperly balanced, unwanted vibration may occur, which in turn causes additional acoustic noise and adds stress to other components, such as fan bearings. Over time, this can cause premature failure of the fan assembly.

Figure 2-2. Rotor Assembly



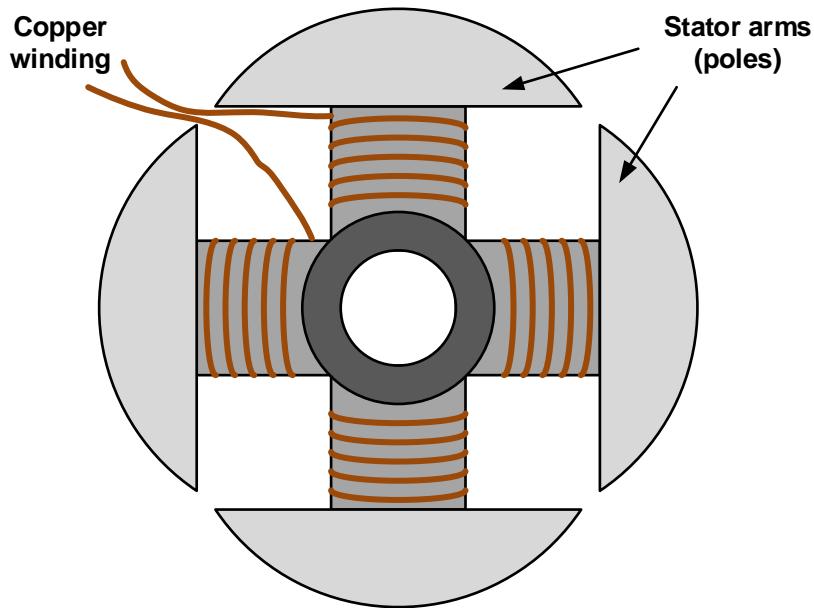
2.2

Stator

The stator, or motor, is used to push the rotor assembly in a certain direction through the use of magnetic force. The core of a stator is composed of multiple layers of steel laminates stacked vertically. Plastic insulators with contact pins surround the steel laminates, which allow the stator to be mounted onto a printed circuit board (PCB), while also preventing the stator windings from making contact with the steel layers. The stator windings are enamel-coated copper wires that are wound onto each of the stator arms (poles). In a single-phase motor, a single copper wire is wound around each stator pole, alternating winding directions for each pole (pole 1 wound clockwise, pole 2 counterclockwise, etc.). The wire is wound such that each end of the wire is connected to the motor driver circuit. [Figure 2-3](#) shows a four-pole stator, which is a common single-phase motor configuration.

As current flows through the stator winding, a magnetic field is created at each pole. These magnetic fields interact with the rotor assembly's permanent magnets, and the magnetic attraction between the stator pole and the permanent magnet pole causes the rotor to rotate. [Motor Drive Circuit](#) discusses the operation of the driver circuit and illustrates how the magnetic fields move the rotor.

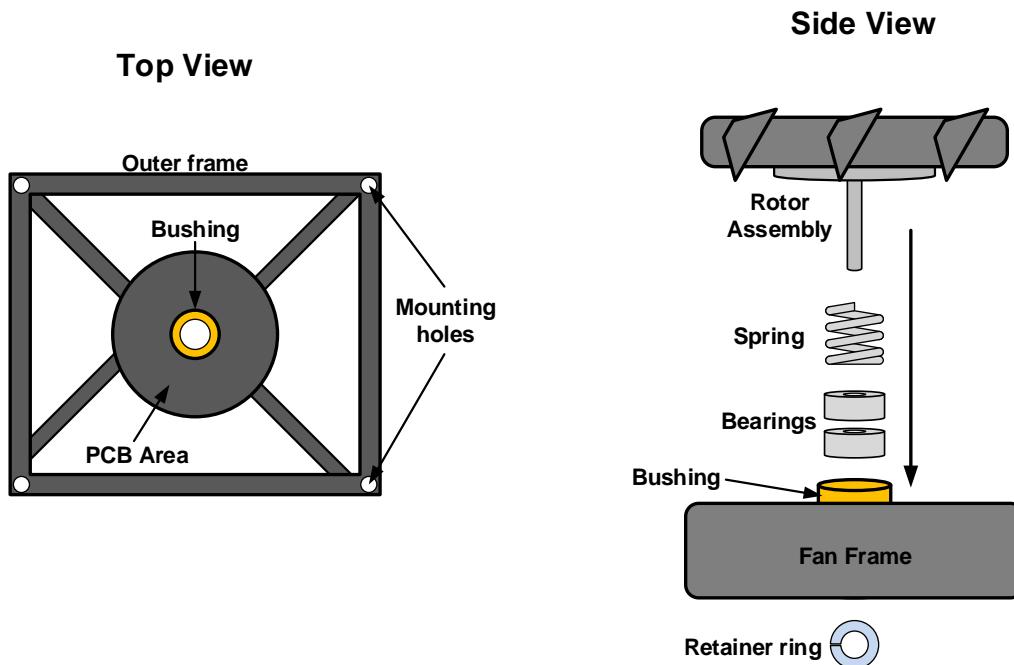
Figure 2-3. Four-Pole Stator Motor



2.3 Frame

A fan frame (housing) is typically made of plastic, but may also be made of aluminum. The outside of the frame has holes used to mount the fan to the object that requires cooling, such as a computer motherboard. Structural vanes connect the outer frame to an inner area, which has a hollow bushing in the center and is used to mount the PCB and stator. The center bushing is typically made of brass and serves as a heat sink for the stator. The inside of the bushing is hollow and holds a set of bearings and a spring through which the rotor shaft slides through, ensuring smooth rotation. [Figure 2-4](#) shows a typical axial fan frame.

Figure 2-4. Typical Axial Fan Frame



2.4 Motor Drive Circuit

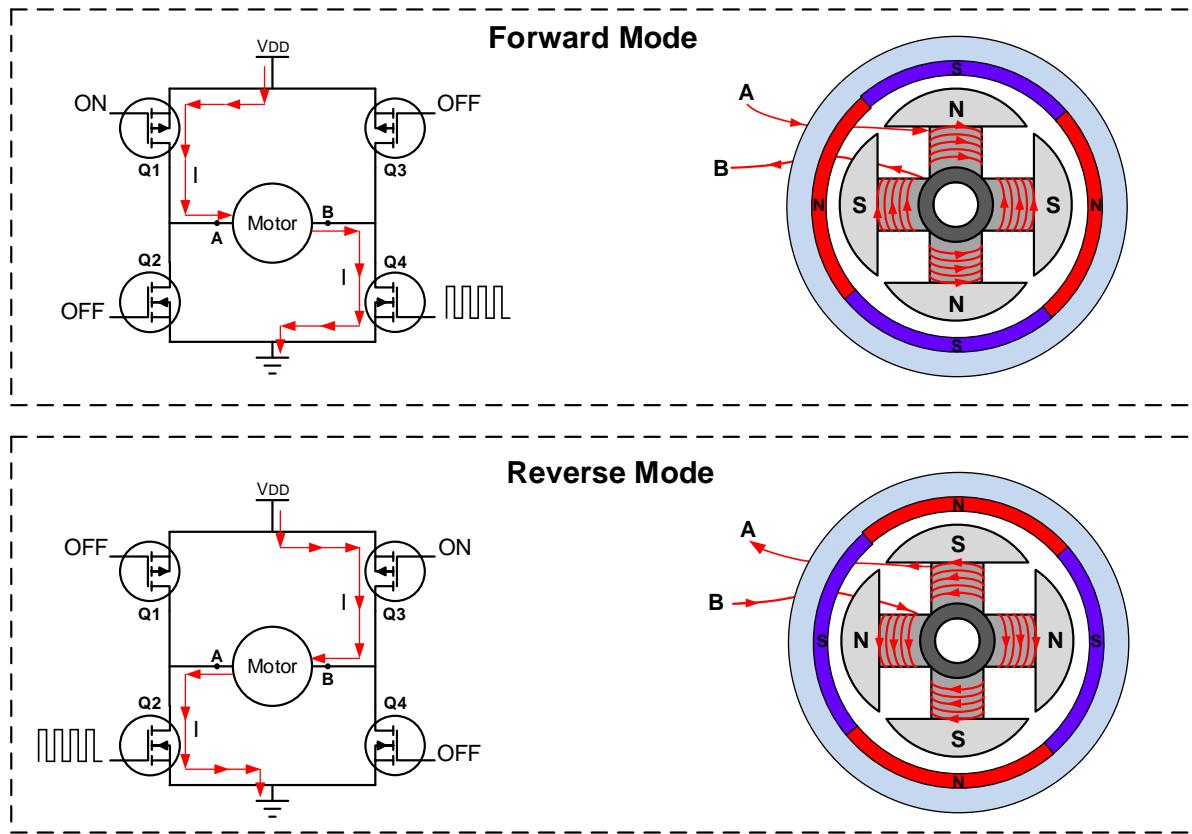
The BLDC's motor drive circuit contains the components necessary to control the rotation of the rotor assembly. The main part of the motor drive circuit is the H-bridge, consisting of switching devices, typically MOSFETs, and an embedded controller, such as a microcontroller. For this project, we will refer to a simple microcontroller-based motor drive circuit, with an H-bridge consisting of two P-channel FETs and two N-channel FETs and a Hall sensor.

The P-channel FETs are configured such that their sources are connected to V_{DD} , while their drains are connected to the motor winding. The N-channel FETs have their drains connected to the motor winding, while their sources are connected to ground.

The microcontroller determines when to turn on/off or apply a PWM signal to the H-bridge, depending on feedback from the Hall sensor. When the Hall sensor detects a change in the magnetic field of the rotor assembly's permanent magnet, it sends a binary '1' or '0' to the microcontroller, depending on the detected magnetic pole (north or south). The microcontroller then decides which FETs to enable or disable and which to apply the PWM signal to. Additionally, the Hall sensor output is used to drive the tachometer signal.

The H-bridge controls the rotation of the rotor assembly by allowing current to flow through the motor windings. [Figure 2-5](#) shows how the current flows through the H-bridge and the motor using two current flow modes: Forward mode and Reverse mode.

Figure 2-5. Motor Driver H-Bridge Circuit



In Forward mode, Q1 is turned fully on and Q4 is switched on and off via a PWM signal. In Forward mode, Q2 and Q3 are off. Current flows through Q1, in and out of the motor winding and through Q4 to ground. Since each stator arm's winding alternates its winding direction, the flow of current through each arm creates alternating north and south magnetic fields. These fields attract the opposite fields of the rotor assembly's permanent magnet such that the stator's north fields attract the permanent magnet's south fields and vice versa. This results in the rotation of the rotor assembly.

In Reverse mode, Q3 is turned fully on and Q2 is switch on and off via a PWM signal, while Q1 and Q4 are off. In Reverse mode, the current flows through the motor windings in the opposite direction as Forward mode. In Forward mode, current enters the motor through point A and leaves through point B, while in Reverse mode, current enters the motor through point B and exits the motor through point A. This means that the stator arms that had a polar north magnetic field during Forward mode will have a polar south magnetic field in Reverse mode. As the Hall sensor detects the changing magnetic fields, the microcontroller continues to switch current flow direction, causing the stator poles to switch magnetic fields, which cause the permanent magnet to follow the changing stator fields.

Rotational speed is controlled through the PWM duty cycle. When a FET receives a zero percent duty cycle signal, the FET is not turned on, so current cannot flow through the FET to ground, resulting in a loss of any stator magnetic field. As the duty cycle increases, the amount of current flowing through the motor winding increases. This results in a stronger magnetic field at each stator pole, which in turns pulls the opposite field in the permanent magnet into alignment faster. At 100% duty cycle, the maximum amount of current flows through the motor winding, resulting in the strongest possible field density and the rotor spins at full speed.

3. Conclusion

This application note discussed the use of the PIC18-Q43 microcontroller family to monitor temperature, control a set of fans' speed and provide visual feedback for viewing. The ADCC module was used to determine the temperature of a thermistor, the 16-bit PWM module was used to control each fan's rotational speed, the CCP module read the fans' tachometer output to determine the actual rotational speed and make speed adjustments, and the UART module was used to send temperature and fan speed to a PC terminal program for viewing. Additionally, BLDC fan operation was briefly discussed.

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., SQL, 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.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-5638-4

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	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