Software Considerations for Advanced Motor Control

Introduction

Small motors operating at less than 300 W are used in a wide variety of applications including automotive systems, printers, copiers, paper handlers, toys, factory automation, test equipment, robotics, aerospace and military, and many others. The most popular small motors types are DC, brushless DC and stepper motors. The quantity of motors produced is roughly inversely proportional to the power level. Small motors are produced in much higher quantities than larger motors.

Motor control-specific DSPs are designed primarily to address the requirements of large off-line motors. Off-line motors are typically AC induction or brushless DC motors operating from 110 to 480 VAC and ranging from 1/4 to 100 HP. Motor control-specific DSPs are often too costly for small motors control systems.

The C8051F3xx series of small form-factor microcontrollers (MCUs) from Silicon Labs is well suited for the control of small motors. These MCUs have several features that are very useful in motor control systems:

- In addition to the standard 8051 timers, the C8051F3xx MCUs also feature a programmable counter array (PCA) with several operating modes.
- The 8-bit pulse-width modulation (PWM) mode is ideally suited for most small motor control applications. The high-speed output mode can be used to generate multiple center-aligned PWM signals with dead-time. The MCUs' digital crossbar can be used to select which port pin receives the PWM signal. The crossbar also eliminates the need for an external multiplexer.
- The C8051F3xx MCUs' analog-to-digital controller (ADC) can be used to measure the motor current, supply voltage, back-emf and temperature of the motor. The analog multiplexer and differential measurement capabilities are very useful in measuring bidirectional motor currents and differential phase voltages.
- The standard 8051 timers T0 and T1 provide a useful second time-base in addition to the PCA.
- The analog comparators can be used to measure zero crossing, over-current, over-voltage, or over-temperature.

The purpose of this white paper is to provide software examples using the C8051F3xx MCUs to control various types of motors. While the examples are relatively simple, they demonstrate effective solutions for the various motor types. A typical motor control system requires additional features and higher complexity. These software examples may be used as a starting point for the development of more complex motor drive systems.

DC Motor Control

DC motors are the most common and least expensive of all small motors. In this white paper the term “DC motor” refers specifically to a brush-commutated permanent-magnet DC motor.

The characteristics of a DC motor make it the easiest motor to use in a variable-speed system. The torque speed characteristics of a DC motor are shown in **Error! Reference source not found.**. The no-load speed of a DC motor is proportional to the voltage applied across the motor. The voltage-speed characteristics of a DC motor driving a constant-torque load, linear-load or exponential-load are also continuous, positive-slope, and predictable. Thus, in most cases it is feasible to use open-loop control. By simply varying the voltage across the motor, one can control the speed of the motor. PWM can be used to vary the voltage applied to the motor. The average voltage applied to the motor is proportional to the PWM duty cycle (ignoring the second order effects of the motor inductance and discontinuous operation).

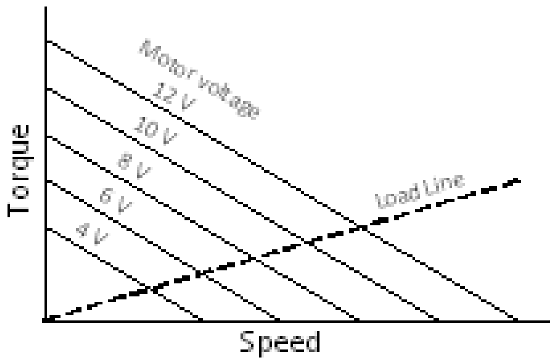

Figure 1. System-Level USB Host to MCU Host Connectivity

A basic example provides simple speed control of a DC motor using F3xx MCUs. This example reads the position of a potentiometer using the ADC and outputs a corresponding PWM signal using the PCA 8-bit PWM mode. The hardware configuration is illustrated in DC Motor Drive Circuit.

A single N-channel Power MOSFET Q1 is used to drive the DC motor. The Power MOSFET should be chosen for the particular motor voltage and current requirements. A free-wheeling diode D1 is connected across the DC motor. When the MOSFET is turned off, the current through the motor inductance will continue to flow. The MOSFET drain voltage will rise to one diode-drop above the motor supply voltage. The current will then flow through the free-wheeling diode.

Most low-voltage motor drive circuits employ Schottky power rectifiers for the free-wheel diode. Schottky rectifiers have a low forward voltage and a very fast reverse recovery time. Both are important factors in a motor drive application.

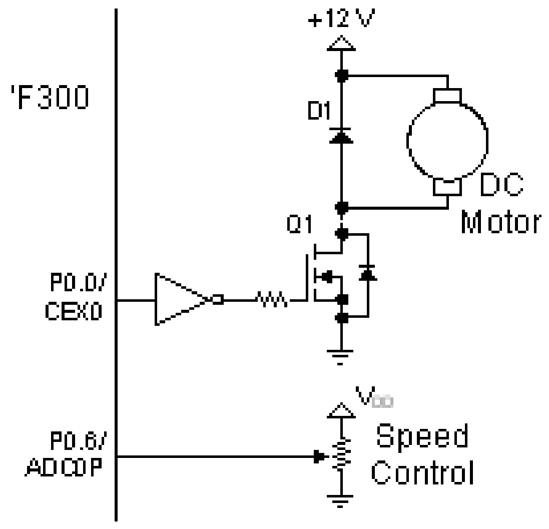

Figure 2. DC Motor Drive Circuit

The power MOSFET is driven by an inverting gate driver. The port pins of ‘F300 are configured by default as inputs with a weak 100 kΩ pullup enabled. The port pins will remain high until the port is configured and the crossbar and peripherals are enabled. The port pins will also be configured as inputs with the weak pullup enabled while the reset pin is held low. By using an inverting driver, the power transistor will be off in the default state. If a non-inverting driver is used, a 10 kΩ pulldown resistor should be connected between the port pin and ground.

The gate driver should have a 3 V compatible input level threshold for use with a 3 V microcontroller. If the motor voltage is between 5 V and 15 V, the gate drive can be powered directly off the motor supply voltage. If the motor voltage is higher than 15 V, a separate gate drive supply voltage is needed, typically 5 V or 12 V. A logic-level power MOSFET should be used when working with a gate drive supply voltage below 10 V.

The software is very simple. The `main()` function initializes the clock, ports, and peripherals and enters the `while(1)` loop. The `while(1)` loop reads the value of the potentiometer voltage using the `avgADC()` function and outputs the value to the 8-bit PWM.

The `PORT_Init()` function configures the port I/O, peripherals, and enables the digital crossbar. Here, the output is enabled for the 8-bit PWM and a push-pull output for the gate drive.

The system clock, SYSCLK, is configured to operate at the maximum speed of 24.5 MHz which allows the 8-bit PWM a clock period of 160 ns and a frequency of 24 kHz.

The `ADC0_Init()` function configures the ADC for polled mode. The ADC gain is set to 1 and a conservative frequency of 1 MHz is chosen for the ADC clock. It is important to remember to also initialize the voltage reference and configure the ADC to use V_{DD} for full-scale.

The function **`readADC()`** reads the voltage one time using polled mode and returns the ADC value. The function **`avgADC()`** calls the **`readADC()`** function and will return the average value of 64 samples. Averaging the ADC reading minimizes the effects of noise and reduces jitter in the PWM output.

When using the PCA 8-bit PWM mode, a value of 0x00 corresponds to a duty cycle of 100% and a value of 0xFF corresponds to a duty cycle of 0.39% at the CEX0 output. A duty cycle of 0% may be achieved by clearing the ECOM0 bit in PCA0CPM0 SFR.

When using an inverting driver, the relationship is reversed. A value of 0x00 corresponds to a 0% duty cycle and a value of 0xFF corresponds to a duty cycle of 99.6% on the MOSFET gate. All software examples in this white paper using 8-bit PWM are limited to 99.6% PWM for simplicity.

There are some cases where a 100% duty cycle is desirable. A 100% duty-cycle will effectively eliminate switching losses. Since the MOSFET never turns off, there are no switching losses in the MOSFET and no losses in the diode. The only power losses are conduction losses in the power MOSFET. If the motor is expected to run at full-speed most of the time, a maximum duty cycle of 100% is desirable. A duty cycle of 100% may be achieved by clearing the ECOM0 bit in PCA0CPM0 SFR.

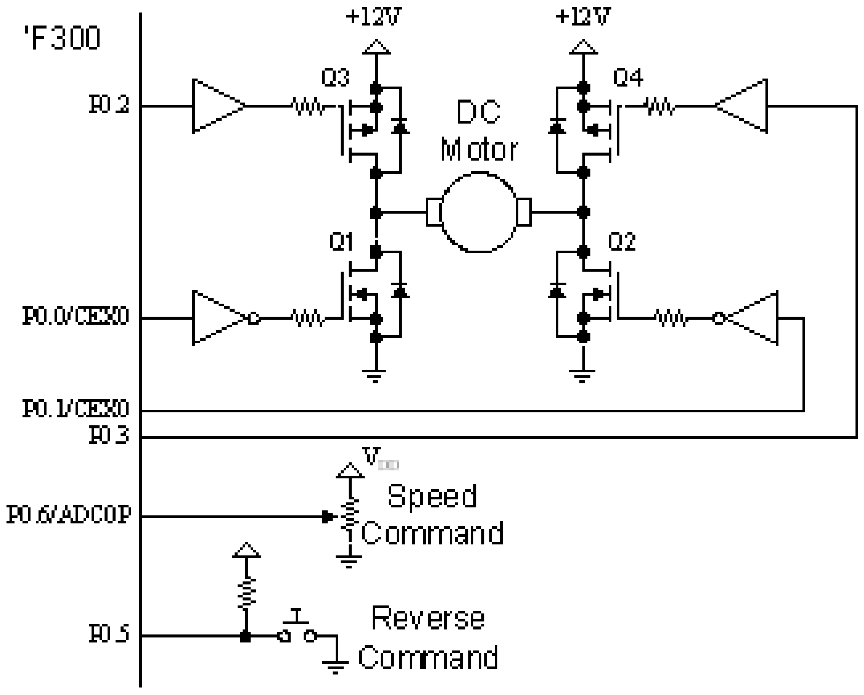

Figure 3. DC Motor Full-Bridge Circuit

DC Motor with Reversing

Permanent-magnet DC motors are often used in applications that require the ability to reverse the direction of the motor. To reverse the direction of rotation, it is necessary to reverse the polarity of the voltage on the motor. This requires the use of an H-Bridge. An H-Bridge has four transistors as shown in DC Motor Full-Bridge Circuit. When driving the motor in the forward direction, Q4 is turned on and a PWM signal is applied to transistor Q1. To drive the motor in the reverse direction, Q3 is turned on and a PWM signal is applied to Q2. In this example, the lower transistors are used for PWM speed control and the upper transistors are used for steering. Using this topology, it is possible to provide variable speed control in both directions.

In DC Motor Full-Bridge Circuit, N-channel power MOSFETs are used for the low-side transistors and P-channel power MOSFETs are used for the high-side transistors. Using complementary power MOSFETs is very cost effective solution for DC motor drives below 20 V. As shown in DC Motor Full-Bridge Circuit, the low-side gate drivers are inverting and the high-side gate drivers are non-inverting. The gate driver polarities are chosen to ensure that the power transistors are off while the port pins are in the reset configuration with the weak pull-ups enabled.

The software for this example builds on the code of the initial example. The main loop now includes an *if* statement that checks the state of the reverse switch SW1. When the reverse button is pressed, the PWM is disabled and all of the P0 outputs are disabled. When the button is released the motor will reverse directions.

The initialization functions are similar to Example 1, except that additional pins are configured as push-pull outputs.

The `reverse()` function is called to reverse the direction of the motor. A flag bit `Fwd` is used to save the state of the motor. The `Fwd` bit is toggled and then used to determine which outputs to activate.

There is a potential problem reversing the motor. While the reverse switch SW1 is held down the motor may continue to spin for some time due to the inertia of the motor. While the motor is turning, it will generate a back-emf proportional to the speed of the motor. If the reverse button is released before the motor stops spinning, the motor back-emf will be shorted out by the upper transistors as described below.

Referring to DC Motor Reversing Hazard, suppose Q4 is initially on and the motor is turning in the forward direction. Assume the motor is turning and the back-EMF is about 6 V. Now the switch is pressed and all four transistors are turned off. The right side of the motor will be 6 V higher than the left side of the motor. Then the switch is released and Q3 is turned on. The left side of the motor is pulled up to the supply voltage and the back-emf of the motor is shorted by the internal diode of Q4.

The end result is that the motor stops and all energy stored in the mechanical inertia of the motor is dumped into Q4. This could easily damage the upper transistors during reversal. In some applications with a large frictional load, a fixed delay may be adequate to ensure the motor has time to stop. In other applications, the motor may take several seconds to come to a complete stop. A universal solution to this problem is illustrated by Figure 4.

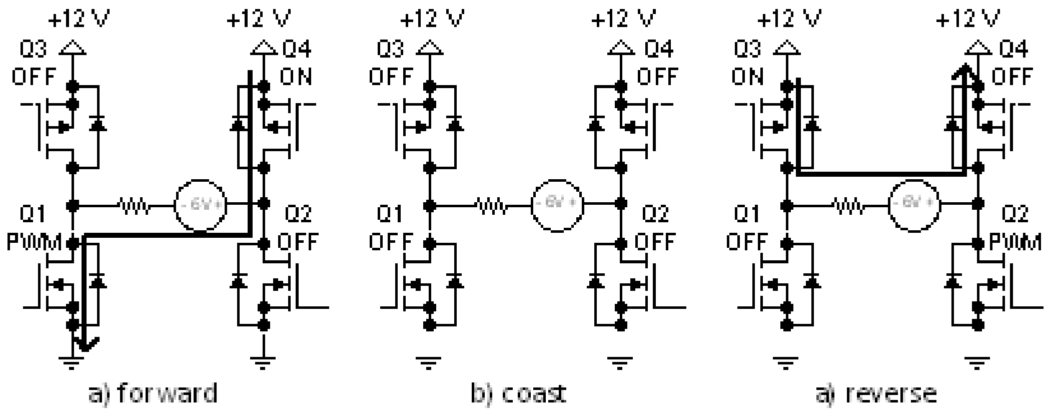

Figure 4. DC Motor Reversing Hazard

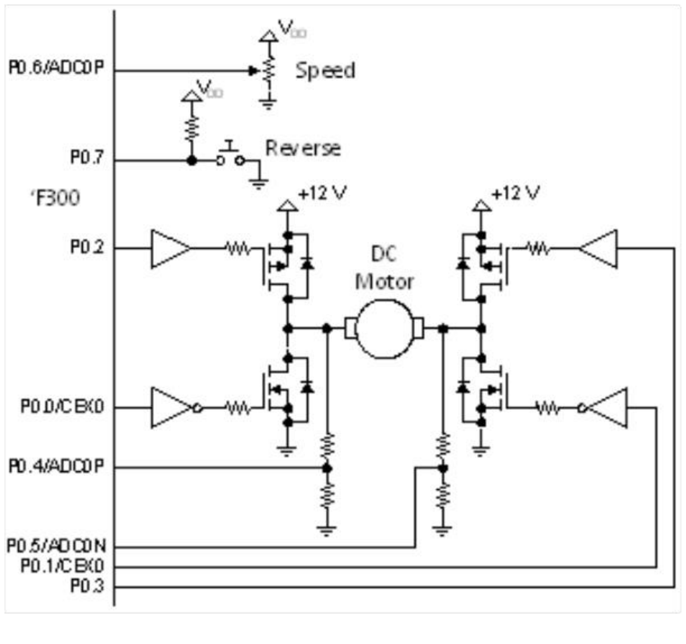

Figure 5. DC Motor Drive with Voltage Sensing

DC Motor with Soft Reversing

This software example for a DC motor builds on the second example and provides soft reversing. To safely reverse a DC motor, it is necessary to determine if the motor is still in motion.

A simple and effective method to determine if the motor is still spinning is to measure the differential voltage across the motor terminals. The ADC can be configured to measure the differential voltage between any two inputs of the analog multiplexer. The programmable window detector may also be used to determine if the differential voltage has fallen within preset limits. In this example, the motor will reverse after the differential motor voltage remains below 3% of full scale for 100 ms.

The hardware for a DC motor drive with voltage sensing is similar with the addition of two resistor dividers connected to the motor terminals, as shown in DC Motor Drive with Voltage Sensing.

The main loop has been modified to detect motor stop. The **detectStop()** function first configures the ADC to measure the differential voltage. The ADC and window detector are both used in polled mode. If the ADC value is within the preset window a counter is incremented. A 10 ms delay using timer T0 sets the sample time. Any sample outside the window will reset the counter. It will take 10 consecutive samples within the window before exiting the while loop. The **detectStop()** function will re-configure the ADC to measure the speed potentiometer before returning to the main loop.

Brushless DC Motor Control

Brushless DC (BLDC) motors offer some advantages over conventional brush-commutated DC motors. The electronics and sensors effectively replace the role of the brushes, offering long life, reduced maintenance, and no brush noise. The torque-speed characteristics of a properly commutated BLDC motor are identical to the DC motor as shown in **Error! Reference source not found.**. Thus, brushless DC motors exhibit the same desirable qualities that make DC motors so well suited for variable speed control. This example provides simple open-loop control of a BLDC motor using Hall-effect sensors to control the motor commutation. The speed of the BLDC motor is controlled using a simple potentiometer. The characteristics of the BLDC motor controlled in this manner are similar to the original DC motor control example.

The hardware required for this example is illustrated in

. Since additional outputs are required for a BLDC motor, the C8051F330 is a recommended MCU. If more memory resources are required for the application, the C8051F336 is also a good choice because it has a larger 16 kB code space and is code compatible with the C8051F330. The motor is driven by six power transistors in a three-phase bridge configuration. The lower transistors Q1–3 are N-channel power MOSFETs. The upper three transistors are P-channel power MOSFETs. This simplifies the gate drive arrangement. Again, complementary gate drivers are used so that the power transistors are off in the default state.

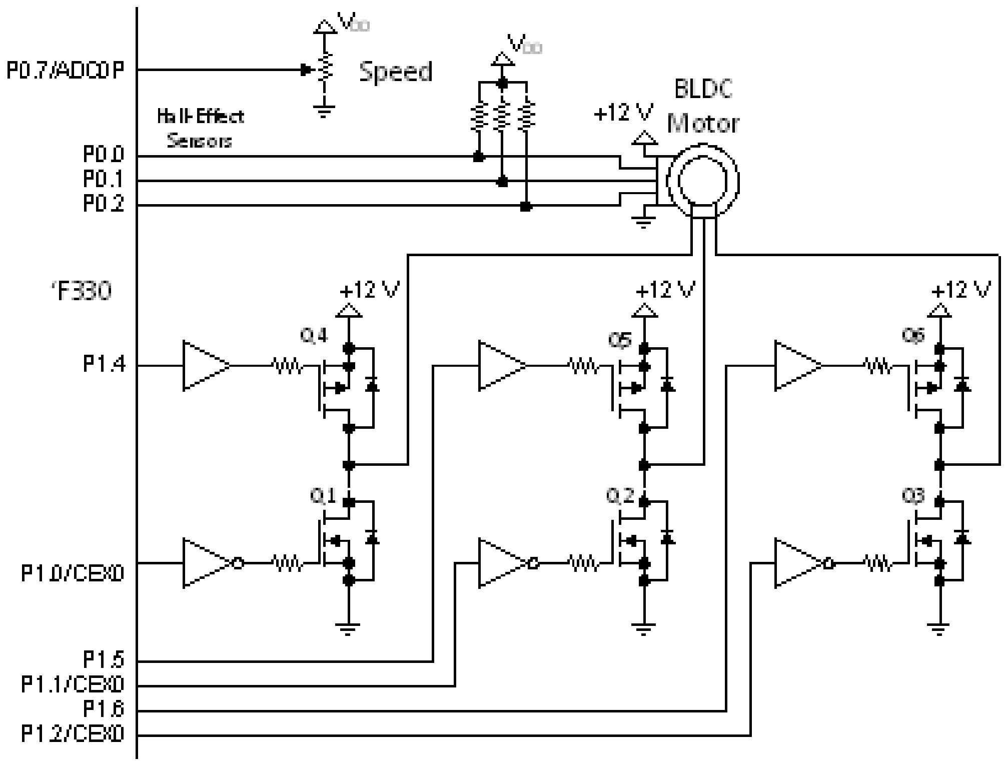

Figure 6. Brushless DC Motor Drive

Hall-Effect sensors have open-collector outputs and require pull-up resistors. Check the motor specifications to ensure the Hall-effect sensors are configured properly. The open-collector outputs are usually 3 V compatible. However, the Hall-effect sensors also require a bias supply that typically requires more than 3.0 V. In most systems, the Hall-effect sensors can be powered off the motor supply voltage or the gate drive supply voltage.

Debugging software with breakpoints can place the motor and MOSFET's in an undesirable state. When the MCU hits a breakpoint, the pins are effectively frozen in time and may leave the PWM outputs in the active state. The recommended procedure is to always disconnect the motor leads before single stepping code or using breakpoints. A BLDC motor will stall with full voltage across one winding. The BLDC motor stall current is only limited by the internal resistance of the winding. This will most likely damage the power MOSFETs.

The software for the BLDC motor example contains many new elements as discussed below.

The `PORT_Init()` function configures the crossbar and output pin assignments. The additional control pins are configured as push-pull outputs for the 3-phase control and inputs to read the hall sensors

The programmable counter array time-base is configured to use the 160 ns time base, and the counter is started. However, the Module 0 mode SFR is not initialized for 8-bit PWM. No motor drive will be enabled until the Hall-effect position is determined.

The **`main()`** function first initializes everything and sets the **`start`** flag bit. The main loop first checks the position of the Hall-effect sensors using the **`hallPosition()`** function. If the **`start`** flag bit is set or the Hall position has changed, the motor is commutated by calling the **`commutate()`** function. Next the speed input is read and the speed setting is written to the PWM output.

The **`hallPosition()`** function returns a zero on an error condition. This occurs if the Hall-effect inputs are all high or all low. If an error occurs, the main loop disables all outputs by calling the **`coast()`** function. The start bit is also set on an error condition to force a commutation on the next valid Hall position reading.

The **`readHalls()`** function reads and debounces the Hall-effect code on the Hall-effect input port pins. This function waits for three consecutive identical readings. This reduces the likelihood of an erroneous reading while the Hall-effect code is changing.

The **`hallPosition()`** first reads the Hall-effect code by calling the **`readHalls()`** function described above. The Hall code pattern is stored in the constant array **`hallPattern[]`**. A single line *for* loop with post decrement is used to find the corresponding index for the matching Hall-effect code. The **`hallPosition()`** function returns a value 1 through 6 if it finds a matching pattern. If no match is found the **`hallPosition()`** function returns a zero value.

The **`commutate()`** function is used to initialize the outputs on start-up, to change the state of the outputs when the Hall position changes, and to restart the motor after a Hall error has been corrected. The **`commutate()`** function first disables the PWM and the upper transistors. It then uses the index obtained from the **`hall Position()`** function.

Conclusion

There is no universal standard for the Hall-effect pattern or the commutation pattern. Consult the motor manufacturer's data sheet for the particular motor you are using. Carefully check both patterns against the manufacturer's data sheet. Also, check the correspondence between the Hall-effect pattern and the commutation pattern. It may be necessary to change the offset between the two patterns.

More Information

This white paper is a subset of an Application Note offered by Silicon Labs titled “AN191 Motor Control Software Examples.” To view the entire Application Note, including the software listing, please click here to read AN191.

# # #