intel®

AP-425

APPLICATION NOTE

Small DC Motor Control

JAFAR MODARES
ECO APPLICATIONS

September 1988

Order Number: 270622-001

SMALL DC MOTOR CONTROL

CONTENTS

PAGE

INTRODUCTION

This application note shows how an 83C51FA can be used to efficiently control DC motors with minimum hardware requirements. It also discusses software implementation and presents helpful techniques as well as sample code needed to realize precision control of a motor.

There is also a brief overview of the new features of the 83C51FA. This new feature is called the Programmable Counter Array (PCA) and is capable of delivering Pulse Width Modulated signals (PWM) through designated I/O pins.

It is assumed that the reader is familiar with the MCS-51 architecture and its assembly language. For more information about the 8051 architecture and the PCA refer to the Embedded Controller Handbook Volume 1 (order no. 210918-006).

This document will not discuss stepper motors or motor control algorithms.

DC MOTORS

DC motors are widely used in industrial and consumer applications. In many cases, absolute precision in movement is not an issue, but precise speed control is. For example, a DC motor in a cassette player is expected to run at a constant speed. It does not have to run for precise increments which are fractions of a turn and stop exactly at a certain point.

However, some motor applications do require precise positioning. Examples are high resolution plotters, printers, disk drives, robotics, etc. Stepper motors are frequently used in those applications. There are also applications which require precise speed control along with some position accuracy. Video recorders, compact disk drives, high quality cassette recorders are examples of this category.

By controlling DC motors accurately, they can overlap many applications of stepper motors. The cost of the control system depends on the accuracy of the encoder and the speed of the processor.

The 83C51FA can control a DC motor accurately with minimum hardware at a very low cost. The microcontroller, as the brain of a system, can digitally control the angular velocity of the motor, by monitoring the feedback lines and driving the output lines. In addition it can perform other tasks which may be needed in the application.

Almost every application that uses a DC motor requires it to reverse its direction of rotation or vary its speed. Reversing the direction is simply done by changing the polarity of the voltage applied to the motor. Figure 1 shows a simplified symbolic representation of a driver circuit which is capable of reversing the polarity of the input to the motor.

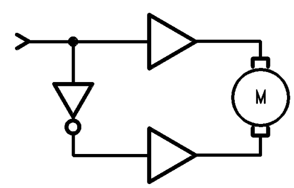


Figure 1. Reversible Motor Driver Circuit

Varying the speed requires changing the voltage level of the input to the motor, and that means changing the input level to the motor driver. In a digitally-controlled system, the analog signal to the driver must come from some form of D/A converter. But adding a D/A converter to the circuit adds to the chip count, which means more cost, higher power consumption, and reduced reliability of the system.

The other alternative is to vary the pulse width of a digital signal input to the motor. By varying the pulse width the average voltage delivered to the motor changes and so does the speed of the motor. A digital circuit that does this is called a Pulse Width Modulator (PWM). The 83C51FA can be configured to have up to 5 on-board pulse width modulators.

THE 83C51FA

The 83C51FA is an 8-bit microcontroller based on the 8051 architecture. It is an enhanced version of the 87C51 and incorporates many new features including the Programmable Counter Array (PCA).

Included in the Programmable Counter Array is a 16-bit free running timer and 5 separate modules.

The PCA timer has two 8-bit registers called CL (low byte) and CH (high byte), and is shared by all modules. It can be programmed to take input from four different sources. The inputs provide flexibility in choosing the count rate of the timer. The maximum count rate is 4 MHz ($^1/_4$ of the oscillator frequency).

Some of the port 1 pins are used to interface each module and the timer to the outside world. When the port pins are not used by the PCA modules, they may be used as regular I/O pins.

The modules of the PCA can be programmed to perform in one of the following modes: capture mode, compare mode, high speed output mode, pulse width modulator (PWM) mode, or watchdog timer mode (only module 4).

Every module has an 8-bit mode register called CCAPMn (Figure 2), and a 16-bit compare/capture register called CCAPnL & CCAPnH, where n can be any value from 0 to 4 inclusive. By setting the appropriate bits in the mode register you can program each module to operate in one of the aforementioned modes.

7	6	5	4	3	2	1	0
—	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCCFn

CCAPMn

ECOMn — Enables the comparator function. Must be set for functions which require comparing of the compare/capture registers with the 16-bit timer, i.e., software timer, high-speed output, watchdog timer, and PWM.

CAPPn — Capture on positive edge of signal.

CAPNn — Capture on negative edge of signal.

MATn — Find a match between the capture/compare and 16-bit timer.

TOGn — Toggle I/O pin upon a match between capture/compare registers and 16-bit timer.

PWMn — Generate PWM on I/O pin upon a match between the low byte of capture/compare and the low byte of PCA timer.

ECCFn — Enables compare/capture flag CCFn in the CCON register to generate an interrupt.

Figure 2. CCAPMn Register

When a module is programmed in capture mode, an external signal on the corresponding port pin will cause a capture of the current value of the 16-bit timer. By setting bits CAPPn or CAPNn or both, the module can be programmed to capture on the rising edge, falling, edge, or either edge of the signal. If enabled, an interrupt is generated at the time of capture.

When module is to perform in one of the compare modes (software timer, high speed output, watch dog timer, PWM), the user loads the capture/compare registers with a calculated value, which is compared to the contents of the 16-bit timer, and causes an event as soon as the values match. It can also generate an interrupt.

PWM is one of the compare modes and is the only one which uses only 8 bits of the capture/compare register. The user writes a value (0 to FFH) into the high byte (CCAPnH) of the selected module. This value is transferred into the lower byte of the same module and is compared to the low byte of the PCA timer. While CL < CCAPnL the output on the corresponding pin is a logic 0. When CL > CCAPnL, the output is a logic 1.

In this application note we will see how a module can be programmed to perform as a PWM to control the speed and direction of a DC motor.

SETTING UP THE PCA

The 83C51FA has several Special Function Registers (SFRs) that are unknown to ASM51 versions before 2.4. The names of these SFRs must be defined by DATA directive or be defined in a separate file and be included at the time of compilation. Such a file has already been created and is included in the ASM51 package version 2.4.

Two special function registers are dedicated to the PCA timer to allow mode selection and control of the timer. These registers are CCON and CMOD and are shown in figure 3. CCON contains the PCA timer ON/OFF bit (CR), timer rollover flag (CF) and module flags (CCFn). Module flags are used to determine which module causes the PCA interrupt.

7	6	5	4	3	2	1	0
CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0

Address 0D8H Reset Value = 00X0 0000 B
Bit Addressable

CCON

7	6	5	4	3	2	1	0
CIDL	WDTE	—	—	—	CPS1	CPS0	ECF

Address 0D9H Reset Value = 00XX X000 B
Not Bit Addressable

CMOD

Figure 3. CCON and CMOD Registers

First the clock source for the PCA timer must be defined. The 16 bit timer may have one of four sources for its input. These sources are: osc freq/4, osc freq/12, timer 0 overflow, and external clock.

Two bits in the CMOD register are dedicated to selecting one of the sources for the PCA timer input. They are bits 1 and 2 of CMOD which are called CPS0 and CPS1. CMOD is not bit addressable, thus the value must be loaded as a byte. Figure 4 shows all the sources and the corresponding values of CPS0 and CPS1.

CPS1	CPS0	TIMER INPUT SOURCE
0	0	Internal clock, Fosc/12
0	1	Internal clock, Fosc/4
1	0	Timer 0 overflow
1	1	External clock (input on P1.2)

Figure 4. Timer Input Source

Next the appropriate module must be programmed as a PWM. As it was noted earlier, the 8-bit mode register for each module is called CCAPMn (see figure 2). Bit 1 of each register is called PWMn. This bit along with ECOMn (bit 6 of the same register) must be set to program the module in the PWM mode. PWM is one of the compare functions of the PCA, and ECOMn enables the compare function. Thus, the hex value that must be loaded into the appropriate CCAPMn register is 42H.

Now that the module is programmed as a PWM, a value must be loaded in the high byte of the compare register to select the duty cycle. The value can be any number from 0 to 255. In the 83C51FA loading 0 in the CCAPnH will yield 100% duty cycle, and 255 (0FFh) will generate a 0.4% duty cycle. See figure 5.

The next step is to start the PCA timer. The bit that turns the timer on and off is called CR and is bit 6 of CCON register (Figure 3). Since this register is bit addressable, you can use bit instructions to turn the timer on and off.

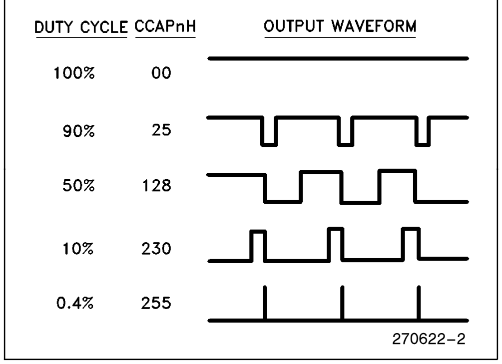


Figure 5. Selected Duty Cycles and Waveforms

In the following example module 2 has been selected to provide a PWM signal to a motor driver. An external clock will be provided for the timer input, so the value that needs to be loaded into CMOD is 06H.

HARDWARE REQUIREMENT

When using an 83C51FA, very little hardware is required to control a motor. The controller can interface to the motor through a driver as shown in figure 6.

```
  .
  .
  .
MOV   CMOD,#06      ; timer input external
MOV   CCAPM2,#42H   ; put the module in PWM mode.
MOV   CCAP2H,#0     ; 0 provides 100% duty cycle (5V)
SETB  CR            ; turn timer on
  .
  .
  .
END
```

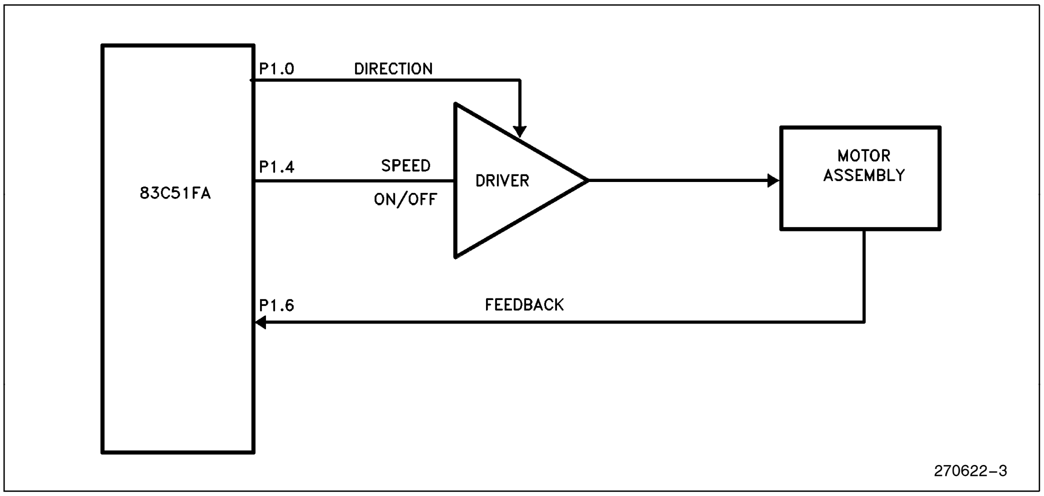


Figure 6. Simplified Circuit Diagram of a Closed Loop System

This configuration, a closed loop circuit, takes up only three I/O pins. The line controlling direction can be a regular port pin but the speed control line must be one of the port 1 pins which corresponds to a PCA module selected for PWM. Depending on how the feedback is generated and processed, it could be connected to a regular I/O, an external interrupt, or a PCA module. Feedback is discussed in more detail in the feedback section of this application note.

The diagram in Appendix A is an example of a DC motor circuit which has been built and bench-tested.

DRIVER CIRCUIT

Although some DC motors operate at 5 volts or less, the 83C51FA can not supply the necessary current to drive a motor directly. The minimum current requirements of any practical motor is higher than any microcontroller can supply. Depending on the size and ratings of the motor, a suitable driver must be selected to take the control signal from the 83C51FA and deliver the necessary voltage and current to the motor.

A motor draws its maximum current when it is fully loaded and starts from a stand still condition. This factor must be taken into account when choosing a driver. However, if the application requires reversing the motor, the current demand will even be higher. As the motor's speed increases, it's power consumption decreases. Once the speed of a motor reaches a steady state, the current depends on the load and the voltage across the motor.

Standard motor drivers are available in many current and voltage ratings. One example is the L293 series which can output up to 1 ampere per channel with a supply voltage of 36 V. It has separate logic supply and takes logical input (0 or 1) to enable or disable each channel. There are four channels per device. The L293D also includes clamping diodes needed for protecting the driver against the back EMF generated during the reversing of motor.

NOISE CONSIDERATIONS

Motors generate enough electrical noise to upset the performance of the controller. The source of the noise could be from the switching of the driver circuits or the motor itself. Whatever the cause of the noise may be, it must be isolated or bypassed.

Isolating the microcontroller from the driver circuit is helpful in keeping the noise limited.

Bypass capacitors help a great deal in suppressing the noise. They must be added to the power and ground (Figure 7 diagram a), on the driver circuit (diagram b), on the motor terminals (diagram c), and on the 83C51FA (diagram d). The capacitors must be as close to the component as possible. In fact the best location is under the chip or on top of it if packaging allows. The diagrams in figure 7 show the location and some typical values for the bypass capacitors.

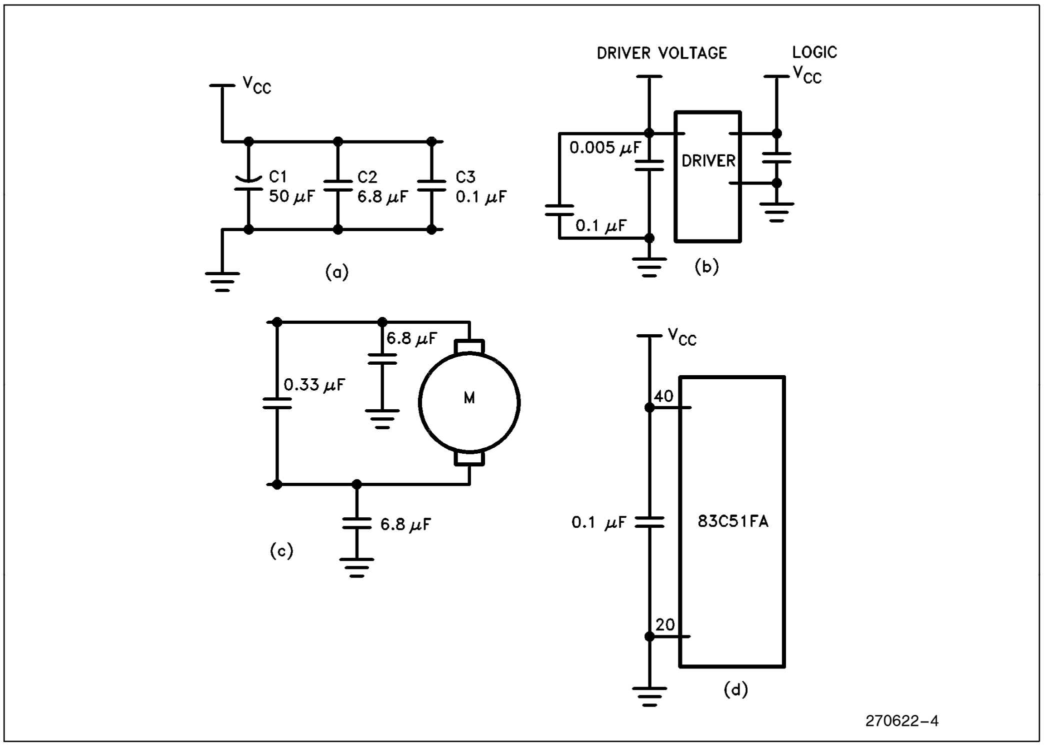


Figure 7. Typical Locations and Values for Bypass Capacitors

OPEN LOOP & CLOSED LOOP SYSTEMS

There are two types of motor control systems: open loop and closed loop.

In the open loop system the controller outputs a signal to turn the motor on/off or to change the direction of the rotation based on an input that does not come from the motor. For example, the position of a manual or timer switch becomes the input to the controller, which varies the input to the motor. In another case, the controller may take input from data tables in the program to run, vary the speed, reverse direction, or stop the motor.

Closed loop systems can use one or more of the above mentioned examples for the open loop system, plus at least one feedback signal from the motor. The feedback signal provides such information as speed, position, and/or direction of motion.

Many applications require that a motor run at a constant speed. The controller has to continuously make adjustments to keep the speed within the limits. In some cases the speed of the motor is synchronized to another motor or moving part of the system.

Depending on the type of feedback signal, the 83C51FA may have to use other modules of the PCA along with other on-chip peripherals such as Timer/Counters, Serial Port, and the interrupt system to precisely control a DC motor.

The example in the following section uses one PCA module to generate PWM, and another module (in capture mode) to receive feedback from a DC motor.

FEEDBACK

The feedback comes from a sensing device which can detect motion. The sensing device may be an optical encoder, infrared detector, Hall effect sensor, etc. Depending on the application, one or more of the above mentioned sensing devices may be suitable.

The optical sensors should be encapsulated for better reliability. If they are not enclosed, factors such as ambient light, dust, and dirt can lessen their sensitivity.

Hall effect sensors are insensitive to any type of light. They change logic levels going into and coming out of a magnetic field. The sensing device is normally mounted on some stationary part of the system and the magnet is installed on the rotating part. The potential problem with the Hall effect sensors are that if the gap between the magnet and the sensing device is too big, the sensing device may not be affected by the magnetic field. Also the number of magnets is limited which means fewer feedback pulses will be provided.

Whatever the means of sensing, the result is a signal which is fed to the controller. The 83C51FA can use the feedback signal to determine the speed and position of the motor. Then it can make adjustments to increase or decrease the speed, reverse the direction, or stop the motor.

In the following example module 3 of PCA is set up to perform in the capture mode. In this mode module 3 will receive feedback signals from a Hall effect transistor fixed behind a wheel which is mounted on the shaft of a DC motor. Two magnets are embedded on this wheel in equal distances from each other (180 degrees apart). Every time that the Hall effect transistor passes through the magnetic field, it generates a pulse.

The signal is input to P1.6 which is the external interface for module 3 of the PCA. In this example, module 3 is programmed to capture on the rising edge of the input signal. The time between the two captures corresponds to $\frac{1}{2}$ of a revolution. Thus, two consecutive captures can provide enough information to calculate the speed of the motor as explained in the next paragraph. By storing the value of the capture registers each time, and comparing it to its previous value, the controller can constantly measure and adjust the speed of the motor. Using this method one can run a motor at a precise speed, or synchronize it to another event.

In the PCA interrupt service routine, each capture value is stored in temporary locations to be used in a subtract operation. Subtracting the first capture from the second one will yield a 16-bit result. The resultant value, which will be referred to as "Result" in the rest of this document, is in PCA timer counts. An actual RPM can be calculated from Result. Although the 83C51FA can do the calculation, it would be much faster to provide a lookup table within the code. The table will contain values which have been calculated for a possible range of Results.

The following code is an example of how to measure the period of a signal input to module 3 of the 83C51FA. The diagram in figure 8 shows how the period corresponds to the rotation of the wheel. In the diagram "T" is the period and "t" is the time that the magnet is passing in front of the Hall effect transistor.

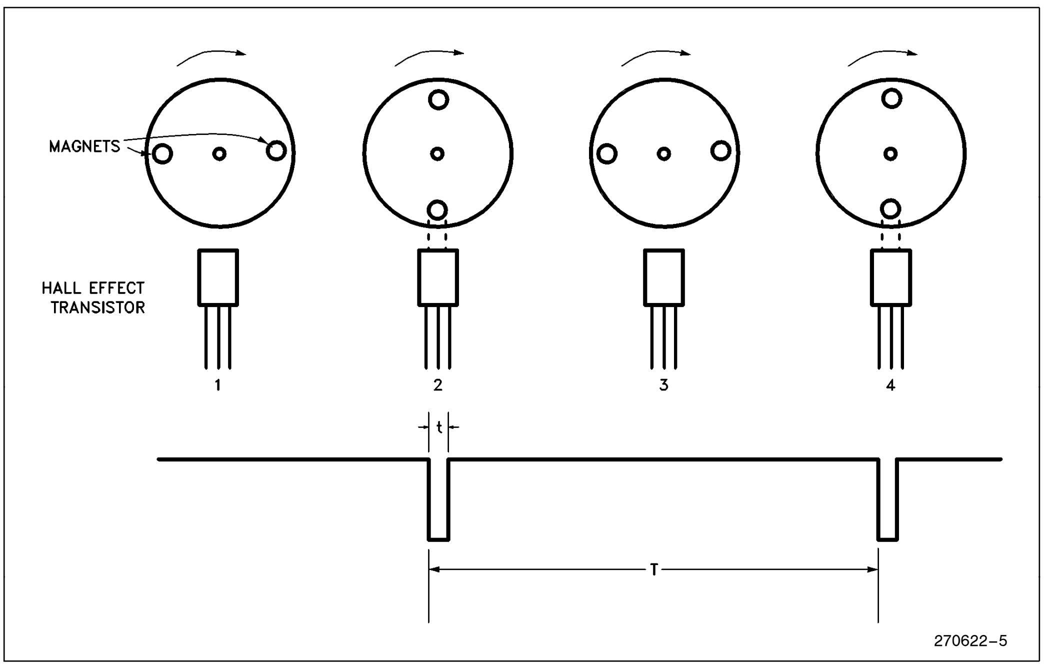


Figure 8. The Output Waveform of the Hall Effect Transistor as it goes Through the Magnetic Field

```
    FLAG    BIT      0          ; test flag
    HI_BYTE_TMP    DATA    45H
    LO_BYTE_TMP    DATA    46H
    HI_BYTE_RESULT DATA    47H
    LO_BYTE_RESULT DATA    48H

        ORG     00H
        JMP     BEGIN

        ORG     33H
        JMP     PCA_ISR

BEGIN:
        MOV     CMOD,#0            ; SET PCA TIMER InPUT fOSC/12.
        MOV     CCAPM3,#21H        ; MODULE 3 IN POSITIVE CAPTURE MODE.
        MOV     CCAP3H,#9AH        ; PWM AT 60 PERCENT DUTY CYCLE.
        SETB    IP.6               ; SET PCA INT. AT HIGH PRIORITY.
        MOV     IE,#0C0H           ; ENABLE PCA INTERRUPT.
        CLR     FLAG
        SETB    CR                 ; TURN PCA TIMER ON.
          .
          .
          .
PCA_ISR:
        JB      FLAG,CAP_2         ; FLAG BIT IS SET TO SIGNIFY 1st
        SETB    FLAG               ; CAPTURE COMPLETE.
        MOV     HI_BYTE_TMP,CCAP3H; SAVE FOR NEXT CALCULATION.
        MOV     LO_BYTE_TMP,CCAP3L
        CLR     CCF3               ; RESET PCA INT. FLAG MODULE 3
        RETI
CAP_2:
        CLR     C                  ; FOR SUBTRACT OPERATION.
        MOV     A,CCAP3L           ; SUBTRACT OLD CAPTURE FROM NEW CAPTURE.
        SUBB    A,LO_BYTE_TMP
        MOV     LO_BYTE_RESULT,A   ; SUBTRACTION RESULT OF LOW BYTE.
        MOV     A,CCAP3H
        SUBB    A,HI_BYTE_TMP      ; HIGH BYTE SUBTRACTION.
        MOV     HI_BYTE_RESULT,A   ; SUBTRACTION RESULT OF HIGH BYTE.
        CLR     IE.6               ; DISABLE PCA INTERRUPT.

In this example only one measurement is taken. That is why
the PCA interrupt is disabled in the above line of instruction.

RET_PCA:
        CLR     CCF3               ; RESET PCA INT. FLAG MODULE 3
        RETI
        END
```

SOFTWARE/CPU OVERHEAD

It takes the 83C51FA no more than 250 bytes of code to control a DC motor. That is to run the motor at various speeds, monitor the feedback, use electrical braking, and even run it in steps. However, the CPU time spent on the above tasks can add up to 70 to 75% of the total time available (clock frequency 12 MHz).

The section of software which turns the motor on and off, or sets the speed is very short. In fact, all of that can be done in less than 30 instructions. Thus, in an open loop system, the controller spends an insignificant amount of time on controlling the motor. However, in a closed loop system the controller has to continuously monitor the speed and adjust it according to the program and the feedback.

The rest of this section talks about electrical braking, stepping a DC motor, and offers examples of code to implement these techniques.

ELECTRICAL BRAKING

Once a DC motor is running, it picks up momentum. Turning off the voltage to the motor does not make it stop immediately because the momentum will keep it turning. After the voltage is shut off, the momentum will gradually wear off due to friction. If the application does not require an abrupt stop, then by removing the driving voltage, the motor can be brought to a gradual stop.

An abrupt stop may be essential to an application where the motor must run a few turns and stop very quickly at a predetermined point. This could be achieved by electrical braking.

Electrical braking is done by reversing the direction of the motor. In order to run in reverse direction, the motor has to stop first, at which time the driving voltage is eliminated so that the motor does not start in the new direction. Therefore the length of time that the reversing voltage is applied must be precisely calculated to ensure a quick stop while not starting it in the reverse direction.

There is no simple formula to calculate when to start, and how long to maintain braking. It varies from motor to motor and application to application. But it can be perfected through trial and error.

In a closed loop system, the feedback can be used to determine where or when to start braking and when to discontinue.

During the electrical braking, or any time that the motor is being reversed, it draws its maximum current. To a motor which is turning at any speed, reversing is a heavy load. The current demand of a motor, when it has been reversed,is much higher than when it has just been powered on.

The following shows a code sample for electrical braking on a DC motor. The code is designed for the hardware shown in Appendix A. The subroutine DELAY provides the period that the reverse voltage is applied to the motor. The code for this subroutine is available in the TIME DELAYS section of this document.

```
BEGIN:
        MOV     CMOD,#0         ; SET PCA TIMER INPUT fOSC/12.
        MOV     CCAPM1,#42H     ; SETTING THE MODULE TO PWM MODE.
        SETB    CR              ; PCA TIMER RUN.

; DRIVE MOTOR CLOCKWISE
        CLR     P1.0            ; P1.0 AND THE PWM OF MODULE 1-
        MOV     CCAP1H,#00      ; CONTROL THE SPEED AND DIRECTION.

; 00 IN THIS REGISTER PUTS OUT MAX PWM (LOGICAL 1)
        CALL    STOP_MOTOR
          .
          .
          .
STOP_MOTOR:
        SETB    P1.0            ; REVERSING THE MOTOR.
        MOV     CCAP1H,#0FFH    ;
        CALL    DELAY           ; WAITING FOR 0.5 SECOND.
        CLR     P1.0            ; REDUCING VOLTAGE TO 0.
        RET                     ; RETURN FROM SUBROUTINE.
          .
          .
          .
```

STEPPING A DC MOTOR

Using the 83C51FA, it is possible to run a simple DC motor in small steps. The resolution of the steps will be as high as the resolution of the encoder. If this resolution is sufficient, here is a technique to run a DC motor in steps.

Using a gear box to gear down the motor will increase the resolution of steps. However, putting too much load through the gears will cause sluggish starts and stops.

Electrical braking is used in order to stop the motor at each step. Therefore, the routine that runs the motor in steps will consist of turning it on with full force, waiting for certain period, and stopping it as fast as possible. The wait period depends on the number of steps per revolution.

As the steps and the intervals between them become smaller, the average current demand of the motor increases. This is because the motor is operated at its maximum torque condition every time it starts to rotate and every time it is reversed for electrical braking.

The following code sample shows a continuous loop which runs the motor in steps. The number of steps per revolution depends on the duration of the delay generated by DELAY subroutine. Subroutine WAIT provides the time between the steps.

Subroutine DELAY is the period of time that the motor is kept in reverse. This period must be determined through trial and error for each type of motor and system.

TIME DELAYS

While the 83C51FA is controlling a motor it must frequently wait for the motor to move to certain position before it can proceed with the next task. For example, in the case of electrical braking when the controller reverses the polarity of voltage across the motor, depending on the type, size, and the speed of the motor, it may have up to a second of CPU time before it will turn the motor off.

The wait may be implemented in different ways. Any of the Timer/Counters or unused PCA modules could be utilized to provide accurate timing. The advantage in using the timers is that while the timer is counting, the processor can be taking care of some other tasks. When the timer times out and generates an interrupt the processor will go back and continue servicing the motor.

If there are no timers or PCA modules available for this purpose, a software timer maybe set up by decrementing some of the internal registers. In this method the processor will be tied up counting up or down and will not be able to do anything else. An example of such a timer is:

```
         .
         .
         .
LOOP:
        CLR     P1.0            ; SET DIRECTION CLOCKWISE
        MOV     CCAP1H,#0       ; MAX PWM

The above instruction sets the motor running clockwise. The controller can
be doing other tasks if need be, or just stay in a wait loop, then stop the
motor as shown below.

        SETB    P1.0            ; REVERSING THE MOTOR.
        MOV     CCAP1H,#0FFH    ;.
        CALL    DELAY           ; WAIT FOR IT TO STOP.
        CLR     P1.0            ; REDUCE VOLTAGE TO 0.
        CALL    WAIT            ; TIME BEFORE NEXT STEP.
        JMP     LOOP
          .
          .
          .
```

```
DELAY:
        MOV     R4,#25          ; (decimal)
        MOV     R5,#255         ; (decimal)
DELAY_LOOP:
        DJNZ    R5,DELAY_LOOP
        DJNZ    R4,DELAY_LOOP
        RET
```

Subroutine DELAY provides approximately 6.4 ms with a 12 MHz clock or 4.8 milliseconds with a 16 MHz clock. The length of this delay can be controlled by loading smaller or larger values to R4 to vary from 260 microseconds up to 65 milliseconds at 12 MHz or 48 milliseconds at 16 MHz oscillator frequency. Larger delays may be obtained by cascading another register and creating an outer loop to this one.

Let us assume that it will take a motor 500 milliseconds to stop from its CW rotation and we are going to use Timer/Counter 0 to provide the wait period. Subroutine DELAY1 will keep track of this timing. Module 1 of PCA is selected to provide the PWM.

```
        ORG     0BH
        JMP     TIMER_INTERRUPT_ROUTINE
          .
          .
          .
        CLR     P1.0            ; SET DIRECTION CW
        MOV     CCAP1H,#0       ; MAX PWM

Now the motor is running and the controller can do other tasks.
Some typical tasks are called in the following segment.

BUSY_LOOP:
        CALL    MONITOR_DISPLAY
        CALL    SCAN_KEY_BOARD
        CALL    SCAN_INPUT_LINES
        JNB     STOP_FLAG,BUSY_LOOP

STOP_FLAG gets set by a feedback signal and denotes that the motor must
stop.

        SETB    P1.0            ; REVERSING THE MOTOR
        MOV     CCAP1H,#0FFH    ;
        CALL    DELAY           ; WAIT TILL MOTOR STOPS
        CLR     P1.0            ; REDUCE VOLTAGE TO 0
          .
          .
          .
DELAY1:
        SETB    EA
        SETB    ET0             ; enable timer 0 interrupt
        MOV     TL0,#0D8H
        MOV     TH0,#5EH
```

```
            SETB      TR0      ; timer 0 on
            MOV       R7,#8    ; keep track of how many times
    ; timer 0 must roll over

 ;  continue performing other tasks

  MONITOR_LOOP:
            CALL      MONITOR_DISPLAY
            CALL      SCAN_KEY_BOARD
            CALL      SCAN_INPUT_LINES
            JB        TR0,MONITOR_LOOP
            RET

 TIMER_INTERRUPT_ROUTINE:
            DJNZ      R7,FULL_COUNT
            CLR       TR0
 FULL_COUNT
            RETI
```

To implement a 500 milliseconds delay, timer 0 is used here. In mode 1 timer 0 is a 16-bit timer which takes 65.535 milliseconds at 12 MHz to roll over. Dividing 500 milliseconds to 65.535 shows that timer has to overflow more than 7 times but less than 8 times. How much more than 7 times? The following calculation yields the initial load value of the timer.

$500 \div 65.535 = 7.2695$ taking it backward

$65.535 \times 7 = 458.745$ milliseconds

$500 - 458.745 = 41.255$ milliseconds or 41255 microseconds.

In hexadecimal it is A127H. The initial load value is the complement of this value which is 5ED8H.

CONCLUSION

The 83C51FA with all its on-chip peripherals is a system on one chip. It can simplify the design of a control board and reduce the chip count. Up to 5 DC motors can be controlled while doing other tasks such as monitoring feedback lines, human interfacing (scanning a keyboard, displaying information), and communicating with other processors. The MCS-51 powerful instruction set provides maximum flexibility with minimum hardware.

With its onboard program memory capability, the need for the external EPROM and address latch is eliminated. The 83C51FA can have up to 8K bytes of code and the 83C51FB can have up to 16K bytes of code onboard.

This microcontroller can be used in industrial, commercial, and automotive applications.

APPENDIX A

Figure A-1 shows a symbolic view of the L293B driver. This driver has 4 channels but only two are shown here. Note the inputs A and B and how they are related to each other. You can input the PWM to either one of the inputs and by toggling the other input start or stop the motor. While running, the PWM input controls the speed. Pin P1.4 corresponds to module 1 of the PCA, and pin P1.0 is used as a regular I/O pin.

Figure A-2 shows the schematic of the motor driver, motor, feedback path, and the supporting components.

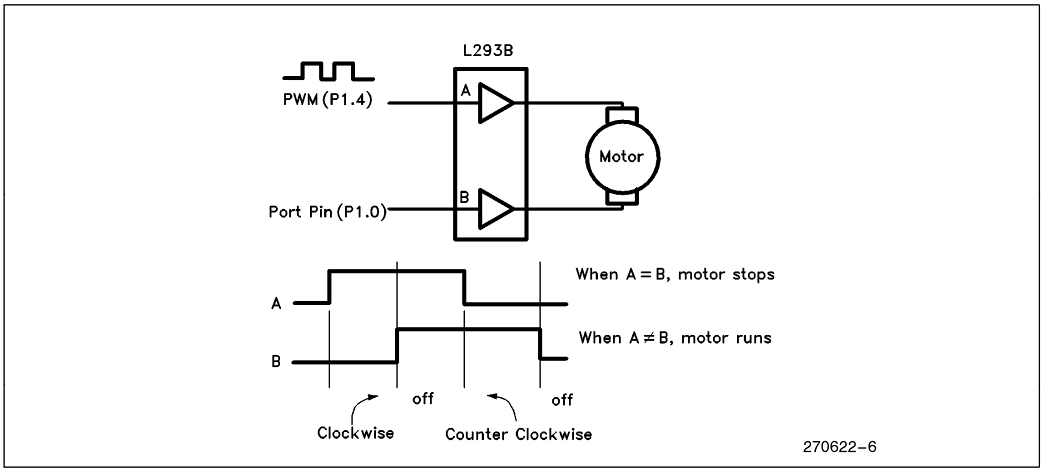
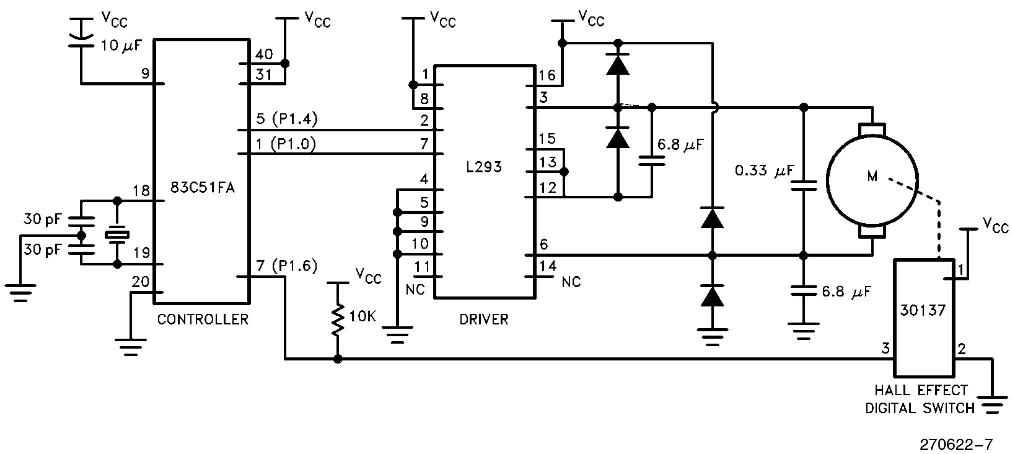


Figure A-1. The L293B Motor Driver



All diodes are the same and could be any of the 1N4000 series.

Figure A-2. Full Schematic of a Motor-Control System