Interfacing a DC Motor

This chapter will describe how you can interface a DC motor to your Boondog 8255 PC Interface Card. The QBASIC and Turbo C code will demonstrate how you can implement pulse-width-modulation (PWM) in software and control the motor's speed.

Introduction and Motivation:

Many PC hobbyists are interested in interfacing a DC motor to their PCs to build a robot or do mechanical work by computer control. Such motor control typically means having the PC turn the motor on, off, adjusting speed and reversing direction. There are many dedicated chips designed for this purpose. Allegro makes one called the UDN2993B which has a dual H-bridge driver that can handle continuous loads up to 30 V and 0.5 A. This chip requires no additional hardware and can be quickly interfaced to your 8255 PC Interface Card.

The UDN2993B is interfaced to the motor (see Figure 16). It only requires an ENABLE and PHASE signals. ENABLE must be a pulse-width-modulated (PWM) signal. PWM is a very common, highly efficient method for controlling motor speed. To understand PWM think how you ride your bicycle. There are times when you pedal (on-time), and times when you coast (off-time). When coasting you are using your momentum and are not physically exerting energy. When you feel (modulation) you are going too slow you pedal (pulse) again for some time (width) and then coast again. This pedal-coast-pedal cycle is called PWM. The amount of time you pedal is called the on-time. The amount of time you coast is called the off-time. The total time is the sum of on-time and off-time.

PWM signals are described by a duty cycle (D.C.) percentage. The duty cycle is the ratio of the on-time and total time. For example, a 100% D.C. means that the motor is receiving energy (you are pedaling) all the time. A 50% D.C. means that the motor receives energy (pedaling) only half the time (coasting half the time).
Interfacing an DC Motor

Typically motor speed controllers are designed in hardware with timer circuitry\(^\text{12}\). However, you can program a PWM signal in software. This is much more economical and efficient. You do not need any external circuitry, and can quickly reprogram a new duty cycle. The Program Listings will show how to do this later.

**Parts List, Numbers and Sources:**

<table>
<thead>
<tr>
<th>Part Description</th>
<th>Supplier and Number</th>
<th>QTY</th>
<th>Cost($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDN2993B (16-pin DIP)</td>
<td>Allegro (see Appendix)</td>
<td>1</td>
<td>3.00</td>
</tr>
<tr>
<td>DC Motor</td>
<td>e.g. Radio Shack 273-223</td>
<td>1</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 7: Parts List for DC Motor Interface

**Schematic Wiring Diagram:**

![Schematic Wiring Diagram](image-url)

Figure 17: Schematic wiring diagram for interfacing a DC motor to the 8255 PC Interface Card. The numbers 0-7 in the box labeled Port A refer to lines A0-A7 on the Terminal Expansion Board

\(^{12}\) For example, the 555 timer chip.
Construction

The schematic shows that the UDN2993B requires only two digital lines to control a single motor. You can add another motor using pins 9-15 and another two digital lines. The toy motor in Figure 16 requires a voltage supply of +5 V. This supply voltage is connected to pin 1. No external heat sinks are required. The UDN2993B can handle much stronger motors with supply voltages up to 30 V and 0.5 A. If you require even higher voltages and currents, Allegro makes the UDN2998 which can handle 50 V and 3 A.

QBasic Program Listing for DC Motor Interface:

```qbasic
' FILE: PWM.C
' DESC: Control speed and direction of a DC motor using PWM

CLS:LOCATE 2, 1: PRINT "ENTER BASE ADDRESS IN DECIMAL (E.G. 608) =>"
LOCATE 2, 45: INPUT BASEADDR
PORTA = BASEADDR
PORTB = BASEADDR + 1
PORTC = BASEADDR + 2
CNTRL = BASEADDR + 3
OUT CNTRL, 128: ' Configure all ports for output
OUT PORTA, 0: ' motor should be off

LOCATE 3, 1: PRINT "Speed Options"
LOCATE 4, 1: PRINT "(f)aster"
LOCATE 5, 1: PRINT "(s)lower"
LOCATE 6, 1: PRINT "(e)verse direction"
LOCATE 7, 1: PRINT "(q)uit"
LOCATE 9, 1: PRINT "Selection => "
LOCATE 9, 14
DutyCycle = 0: ' start off at zero velocity
Enable = 1
Phase = 0

DO
  DO
    GOSUB pwm
    SpeedOption$ = INKEY$
    LOOP UNTIL SpeedOption$ <> ""
    SELECT CASE SpeedOption$
      CASE "f"
        ' increase speed by 5% DC
        DutyCycle = DutyCycle + 5
        IF DutyCycle > 100 THEN
          DutyCycle = 100
        LOCATE 20, 1: PRINT "Can't go faster"
        FOR d = 1 TO 1000: NEXT d
        LOCATE 20, 1: PRINT ("
        END IF
        LOCATE 13, 1: PRINT "Going faster..."
        Enable = 1
```

8255 PC Interface Card Applications Manual 41
GOSUB pwm

CASE "s"
' decrease speed by 5% DC
DutyCycle = DutyCycle - 5
IF DutyCycle < 0 THEN
DutyCycle = 0
LOCATE 20, 1: PRINT "Can't go slower"
FOR d = 1 TO 1000: NEXT d
LOCATE 20, 1: PRINT "               "
END IF
LOCATE 13, 1: PRINT "Going slower..."
Enable = 1
GOSUB pwm

CASE "r"
' reverse direction at present DC
Enable = 1
IF Phase = 0 THEN
Phase = 2
ELSEIF Phase = 2 THEN
Phase = 0
END IF
GOSUB pwm
LOCATE 13, 1: PRINT "Reversing....."

CASE "q"
LOCATE 22, 1: PRINT "Quitting"
OUT PORTA, 0
OUT PORTB, 0
OUT PORTC, 0
END

END SELECT
LOCATE 9, 14

LOOP WHILE (1): ' end of do

pwm:
TotalTime = .1: '100 msec = 0.1 sec
OnTime = TotalTime * DutyCycle / 100
OffTime = TotalTime - OnTime
Enable = 1
OUT PORTA, Enable + Phase
GOSUB DELAYOnTime
Enable = 0
OUT PORTA, Enable + Phase
GOSUB DELAYOffTime
LOCATE 10, 1: PRINT "Duty Cycle is "; : PRINT DutyCycle
RETURN

DELAYOnTime:
TIMERA = TIMER
DO
LOOP WHILE (TIMER - TIMERA) < OnTime
RETURN

DELAYOffTime:
TIMERA = TIMER
DO
LOOP WHILE (TIMER - TIMERA) < OffTime
RETURN
**Program Description:**

The program begins by asking the user to input the base address of the card. Then Ports A, B, C and the control word addresses are assigned. The three ports are configured with outputs roles. DutyCycle sets the initial duty cycle to be 0, so that the motor is initially at rest. Phase is set to 0 and Enable is set to 1, so that the motor is ready to run.

A nested do...loop awaits for the user to hit a key on the PC keyboard and loops through the pwm subroutine. The key menu describes how to control the speed of the motor. Once a key press has been detected (using the INKEY$) the program enters the select...case code. If an f key (faster) is hit the duty cycle is increased by 5%. If an s key (slower) is hit the duty cycle is decreased by 5%. Since the maximum duty cycle is 100%, an if...then is used as a threshold. A similar threshold is used for the minimum duty cycle of 0%. If an r key (reverse) is hit the phase bit is complimented from its present state. The motor will reverse direction at the present duty cycle. The major subroutine is pwm and calculates the OnTime and OffTime values. A divide by 100 is used because duty cycle is in percentages. The TotalTime value of the pulse is set for 0.1 seconds. Port A's A0 (ENABLE) line uses these values to enable the motor for OnTime seconds using the DelayOnTime function. This function uses the QBasic timer statement which accesses the PC's real-time internal clock. Pwm then disables the motor for OffTime seconds using the DelayOffTime function. If the q key (quit) is hit the motor stops and the program exits.

This circuit and program is an example of open-loop control, and does not use any feedback. In our bicycle analogy, if the bicycle has no speedometer, there is no way of knowing your exact speed. Furthermore, the speed depends on a person's pedal rate (on time and off time) and torque. The rate and torque depend on which gear you are in (i.e. the motor's load voltage supply and hence supplied current), the road's slope (load on the motor) and how properly the tires are inflated and how lubricated the gears are (e.g. friction of the motor bearings). It is only by trial and error that you can match your pedal speed and torque with bicycle speed. This trial and error process is called tuning. Similarly, every DC motor needs to be tuned. In the above program listing, a TotalTime of 0.1 seconds was used. For your motor, you may have to tune your motor by changing the TotalTime to reflect your motor's properties. The Turbo C listing follows, and its operation mimics that of the QBasic listing.

**Turbo C Program Listing for DC Motor Interfacing:**

```c
/*
```

FILE: PWM.C
DESC: Control speed and direction of a DC motor using PWM
*/

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>          /* outportb, inportb defined here */
#include<conio.h>        /* formatted text functions defined here */

void pwm(int, int, int, int);

void main(void) {

    //Declare Variables
    int BASEADDR;
    int PORTA, PORTB, PORTC;
    int CNTRL;
    int DutyCycle;
    int SpeedOption;
    int Enable, Phase;  /* Enable on A.0, Phase on A.1 */

    clrscr();            /* clear screen */
    window(5,5,75,30);   /* set up text window */

    gotoxy(1,1); cprintf("Enter Base Address (decimal) e.g. 608 =>
    gotoxy(42,1); scanf("%d", &BASEADDR);
    PORTA = BASEADDR;
    PORTB = BASEADDR + 1;
    PORTC = BASEADDR + 2;
    CNTRL = BASEADDR+3;

    outportb(CNTRL, 128);   /* configure all ports for output */
    outportb(PORTA, 0);     /* motor should be off */

    gotoxy(1,3); cprintf("Speed Options");
    gotoxy(1,4); cprintf("(f)aster");
    gotoxy(1,5); cprintf("(s)lower");
    gotoxy(1,6); cprintf("(r)everse direction");
    gotoxy(1,7); cprintf("(g)uit");
    gotoxy(1,9); cprintf("Selection => ");
    gotoxy(14,9);

    DutyCycle = 0; /* start off at zero velocity */
    Enable = 1;
    Phase = 0;

    do {
        while (!kbhit()) {
            pwm(PORTA, DutyCycle, Enable, Phase);
        } /* end of while */

        SpeedOption = getch();
        switch(SpeedOption) {
        case 102 : /* 102 ASCII is f */ /* increase speed by 5% DC */
            DutyCycle = DutyCycle + 5;
            if(DutyCycle > 100) {
                DutyCycle = 100;
                gotoxy(1,20); cprintf("Can't go faster");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 108 : /* 108 ASCII is s */ /* reduce speed by 5% DC */
            DutyCycle = DutyCycle - 5;
            if(DutyCycle < 0) {
                DutyCycle = 0;
                gotoxy(1,20); cprintf("Can't go lower");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 111 : /* 111 ASCII is r */ /* switch direction */
            Enable = 1;
            Phase = 1;
            gotoxy(1,20); cprintf("Reversing");
            delay(200);
            gotoxy(1,20); cprintf(" ");

        case 104 : /* 104 ASCII is g */ /* exit program */
            gotoxy(1,20); cprintf("Exiting");
            delay(200);
            gotoxy(1,20); cprintf(" ");
        }
    } /* end of do */

    gotoxy(1,20); cprintf("Exiting");
    delay(200);
    gotoxy(1,20); cprintf(" ");

    return 0;
}

#include<conio.h>
#include<dos.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>          /* outportb, inportb defined here */
#include<conio.h>        /* formatted text functions defined here */

void pwm(int, int, int, int);

void main(void) {

    //Declare Variables
    int BASEADDR;
    int PORTA, PORTB, PORTC;
    int CNTRL;
    int DutyCycle;
    int SpeedOption;
    int Enable, Phase;  /* Enable on A.0, Phase on A.1 */

    clrscr();            /* clear screen */
    window(5,5,75,30);   /* set up text window */

    gotoxy(1,1); cprintf("Enter Base Address (decimal) e.g. 608 =>
    gotoxy(42,1); scanf("%d", &BASEADDR);
    PORTA = BASEADDR;
    PORTB = BASEADDR + 1;
    PORTC = BASEADDR + 2;
    CNTRL = BASEADDR+3;

    outportb(CNTRL, 128);   /* configure all ports for output */
    outportb(PORTA, 0);     /* motor should be off */

    gotoxy(1,3); cprintf("Speed Options");
    gotoxy(1,4); cprintf("(f)aster");
    gotoxy(1,5); cprintf("(s)lower");
    gotoxy(1,6); cprintf("(r)everse direction");
    gotoxy(1,7); cprintf("(g)uit");
    gotoxy(1,9); cprintf("Selection => ");
    gotoxy(14,9);

    DutyCycle = 0; /* start off at zero velocity */
    Enable = 1;
    Phase = 0;

    do {
        while (!kbhit()) {
            pwm(PORTA, DutyCycle, Enable, Phase);
        } /* end of while */

        SpeedOption = getch();
        switch(SpeedOption) {
        case 102 : /* 102 ASCII is f */ /* increase speed by 5% DC */
            DutyCycle = DutyCycle + 5;
            if(DutyCycle > 100) {
                DutyCycle = 100;
                gotoxy(1,20); cprintf("Can't go faster");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 108 : /* 108 ASCII is s */ /* reduce speed by 5% DC */
            DutyCycle = DutyCycle - 5;
            if(DutyCycle < 0) {
                DutyCycle = 0;
                gotoxy(1,20); cprintf("Can't go lower");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 111 : /* 111 ASCII is r */ /* switch direction */
            Enable = 1;
            Phase = 1;
            gotoxy(1,20); cprintf("Reversing");
            delay(200);
            gotoxy(1,20); cprintf(" ");

        case 104 : /* 104 ASCII is g */ /* exit program */
            gotoxy(1,20); cprintf("Exiting");
            delay(200);
            gotoxy(1,20); cprintf(" ");
        }
    } /* end of do */

    gotoxy(1,20); cprintf("Exiting");
    delay(200);
    gotoxy(1,20); cprintf(" ");

    return 0;
}

#include<conio.h>
#include<dos.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>          /* outportb, inportb defined here */
#include<conio.h>        /* formatted text functions defined here */

void pwm(int, int, int, int);

void main(void) {

    //Declare Variables
    int BASEADDR;
    int PORTA, PORTB, PORTC;
    int CNTRL;
    int DutyCycle;
    int SpeedOption;
    int Enable, Phase;  /* Enable on A.0, Phase on A.1 */

    clrscr();            /* clear screen */
    window(5,5,75,30);   /* set up text window */

    gotoxy(1,1); cprintf("Enter Base Address (decimal) e.g. 608 =>
    gotoxy(42,1); scanf("%d", &BASEADDR);
    PORTA = BASEADDR;
    PORTB = BASEADDR + 1;
    PORTC = BASEADDR + 2;
    CNTRL = BASEADDR+3;

    outportb(CNTRL, 128);   /* configure all ports for output */
    outportb(PORTA, 0);     /* motor should be off */

    gotoxy(1,3); cprintf("Speed Options");
    gotoxy(1,4); cprintf("(f)aster");
    gotoxy(1,5); cprintf("(s)lower");
    gotoxy(1,6); cprintf("(r)everse direction");
    gotoxy(1,7); cprintf("(g)uit");
    gotoxy(1,9); cprintf("Selection => ");
    gotoxy(14,9);

    DutyCycle = 0; /* start off at zero velocity */
    Enable = 1;
    Phase = 0;

    do {
        while (!kbhit()) {
            pwm(PORTA, DutyCycle, Enable, Phase);
        } /* end of while */

        SpeedOption = getch();
        switch(SpeedOption) {
        case 102 : /* 102 ASCII is f */ /* increase speed by 5% DC */
            DutyCycle = DutyCycle + 5;
            if(DutyCycle > 100) {
                DutyCycle = 100;
                gotoxy(1,20); cprintf("Can't go faster");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 108 : /* 108 ASCII is s */ /* reduce speed by 5% DC */
            DutyCycle = DutyCycle - 5;
            if(DutyCycle < 0) {
                DutyCycle = 0;
                gotoxy(1,20); cprintf("Can't go lower");
                delay(200);
                gotoxy(1,20); cprintf(" ");
            }

        case 111 : /* 111 ASCII is r */ /* switch direction */
            Enable = 1;
            Phase = 1;
            gotoxy(1,20); cprintf("Reversing");
            delay(200);
            gotoxy(1,20); cprintf(" ");

        case 104 : /* 104 ASCII is g */ /* exit program */
            gotoxy(1,20); cprintf("Exiting");
            delay(200);
            gotoxy(1,20); cprintf(" ");
        }
    } /* end of do */

    gotoxy(1,20); cprintf("Exiting");
    delay(200);
    gotoxy(1,20); cprintf(" ");

    return 0;
}
Interfacing an DC Motor

8255 PC Interface Card Applications Manual

};
gotoxy(1,13); cprintf("Going faster...\n");
Enable = 1;
pwm(PORTA, DutyCycle, Enable, Phase);
break;

case 115  : /* 115 ASCII is s */
        /* decrease speed by 5% DC */
    DutyCycle = DutyCycle - 5;
    if(DutyCycle < 0) {
        DutyCycle = 0;
        gotoxy(1,20); cprintf("Can't go slower");
        delay(200);
        gotoxy(1,20); cprintf(
    }
    gotoxy(1,13); cprintf("Going slower...\n");
    Enable = 1;
pwm(PORTA, DutyCycle, Enable, Phase);
break;

case 114 : /* 114 ASCII is r */
        /* reverse direction at present DC */
    Enable = 1;
    if(Phase == 0) {
        Phase = 2;
    } else if(Phase == 2) {
        Phase = 0;
    }
    pwm(PORTA, DutyCycle, Enable, Phase);
gotoxy(1,13); cprintf("Reversing.....\n");
break;

case 113 : /* 113 ASCII is q */
    gotoxy(1,22); cprintf("Quitting");
    outportb(PORTA, 0);
    outportb(PORTB, 0);
    outportb(PORTC, 0);
    exit(0);
    break;

}; /* end of switch */
gotoxy(14,9);
}
/* end of do */

); /* end of main */

void pwm(int PORTA, int DutyCycle, int Enable, int Phase) {
    int OnTime, OffTime, TotalTime;
    TotalTime = 100; /* 100 msec */
    OnTime = (int)(TotalTime * DutyCycle / 100);
    OffTime = (int)(TotalTime - OnTime);
    outportb(PORTA, Enable+Phase);
    delay(OnTime);
    Enable = 0;
    outportb(PORTA, Enable+Phase);
    delay(OffTime);
    gotoxy(1,10); cprintf("Duty Cycle is %3d", DutyCycle);
    return;
This chapter showed how to interface a DC motor to your Boondog 8255 PC Interface Card. The hardware circuitry requires only one chip. The QBasic and Turbo C code implemented a pulse-width-modulated signal. This demonstrated how software can replace many hardware components.

Many companies sell dedicated DC motor controller cards that plug into your PC. These cards often cost over $100. This chapter shows how you can build one for less than $5. These expensive boards give you control routines and additional hardware\(^\text{13}\) to have very precise motor control. This may be overkill for your application and the circuit in this chapter may be all you need. The source code provides a starting point for you to implement your own control algorithms.

\(^\text{13}\) Typical control routines are trapezoidal motion profiles. Typical circuitry consist of chopper drivers and filters.