PCL-832

3-axis servo motor control card

Acknowledgments
PC-LabCard is a trademark of Advantech Co., Ltd. IBM and PC are trademarks of International Business Machines Corporation. MS-DOS, Microsoft C and Quick Basic are trademarks of Microsoft Corporation. BASIC is a trademark of Dartmouth College. Intel is a trademark of Intel Corporation. Turbo C is a trademark of Borland International.

Part No. 2005832000 1st Edition
Printed in Taiwan December 1994

Contents

Chapter 1 General information 1

Introduction 2
Features 3
Applications 3
Specifications 4
General 4
Board layout 5

Chapter 2 Installation 7

Initial inspection 8
Switch and jumper settings 8
Base I/O address (SW1) 8
Quadrature input multiplier (JP1, JP2, JP3) 10
DDA time and Error counter overflow interrupt levels (JP4) 10
I/O Wait state level (JP5) 10
Connector pin assignments 11
Hardware installation 13

Chapter 3 Operation 15

PCL-832 Operating Principles 16
Digital Differential Analysis 16
Closed-loop position control 17

Chapter 3 Operation (ctd.)

Operation 18
The DDA cycle time 19
The DDA pulse buffer 20
The scaling gain 21
The Error counter 21
The Status register 23
The frequency-to-voltage (F/V) converter 23
Using multiple PCL-832s in one system 24

Chapter 4 Programming 25

Software installation 26
Microsoft C drivers and Turbo C drivers 27
Special Procedures 27
The Microsoft C Driver Parameter Files 28
PCLB.PAR 28
HOME.PAR 29
MACHINE.PAR 30
Microsoft C driver functions 32
System functions 32
Preparatory functions 32
Motion functions 35
Motion function return codes 42
Utility functions 42
Turbo C low-level driver functions 45

Chapter 5 Register structure and format 49

I/O Address space 50
I/O address Map 50
Register format 51

Appendix A Calibration 57

VR Assignments 58
Test-Points 59

CHAPTER 1

General information

Introduction

3-axis control

The PCL-832 3-axis Servo-motor Control Card turns your IBM PC or compatible computer into a sophisticated position controller. The card's custom ASIC implementation provides high performance at an affordable price.

The PCL-832 uses digital differential analysis techniques to implement position control. Each axis has its own position control chip, allowing completely independent control of up to three servo motors.

A special synchronization circuit synchronizes all three axes. The card can supply a simulated tachometer output to the servo motor driver. This signal makes a tachometer unnecessary in some applications, reducing overall system costs.

Software Drivers

The PCL-832's programming library (accessible from Microsoft C) supports high-level commands and functions, making control easy. The library includes commands to set the DDA cycle time and acceleration/deceleration curve as well as functions for linear interpolation, circular interpolation, return home and jog. Numerous examples of driving programs have also been included on the utility/ software disk.

Features

- Independent 3-axis servo control
- Fully continuous closed-loop P+offset controller
- Industry-standard two-phase index position encoder interface
- Single-ended or differential encoder interface inputs
- x1, x2, x4 quadrature feedback input
- 12-bit analog output with ±10 V range
- Built-in F/V converter
- Easy programming from C and other high-level languages
- 3-axis linear interpolation
- 2-axis circular interpolation
- Half-size AT (ISA bus) add-on card

Applications

- Precise position control
- Robotics control
- Machine control with up to three axes
- PC-based NC controller

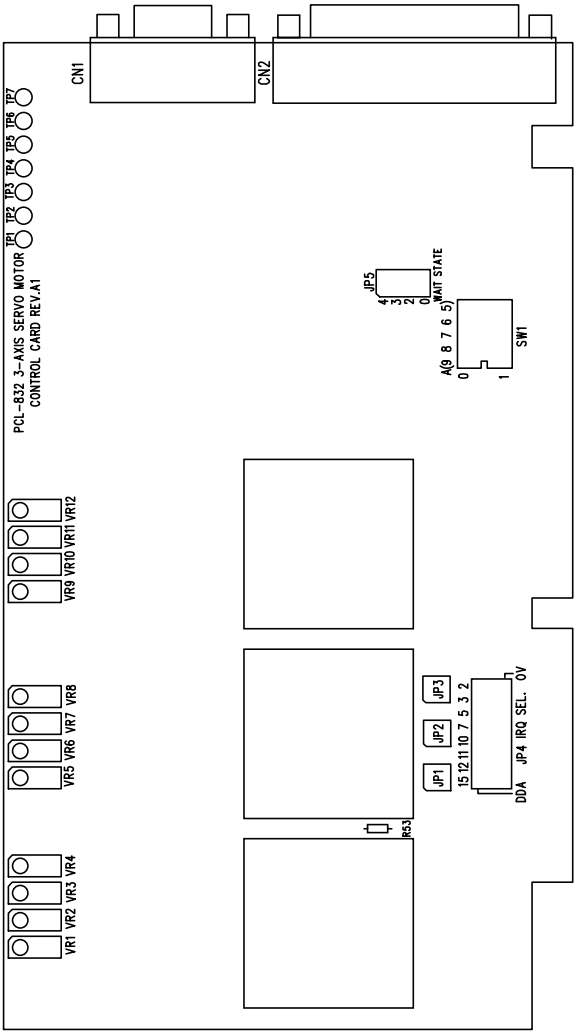
Specifications

- **No. axes**: 3 independent
- **Control algorithm**: Proportional control
- **Positional accuracy**: ±1 quadrature count
- **Effective travel length**: No limit
- **Output type**: 12-bit D/A, ±10 V full scale
- **DDA cycle time**: 1 msec. to 2 sec. (programmable)
- **Error counter**: ±12 bit
- **Tachometer simulation output (F/V converter)**: ±10 V at 250 KHz (default), VR adjustable
- **Home sensor input**: 1 channel per axis
- **Encoder input**: Single-ended or differential
- **Counts per encoder cycle**: x1, x2, x4 (jumper selectable)
- **Max. quadrature input freq.**: 250 KHz

General

- **Bus**: 16-bit AT (ISA bus)
- **IRQ**: 2, 3, 5, 7, 10, 11, 12 or 15
- **I/O addresses**: 32 I/O ports
- **Connector**: DB-9 for servo control
  DB-25 for encoder and home signals
- **Power consumption**: 5 V @ 500 mA max.
  12 V @ 200 mA max.
- **Dimensions**: 7.3" x 4" (185 mm x 100 mm)

Board layout



CHAPTER 2

Installation

Initial inspection

In addition to this manual the shipping container should contain the PCL-832 card and a utility diskette. We carefully inspected the PCL-832 mechanically and electrically before we shipped it. It should be free of marks and scratches and in perfect electrical order on receipt.

As you unpack the card, check it for signs of shipping damage (damaged box, scratches, dents, etc.). If it is damaged or fails to meet its specifications, notify our service department or your local sales representative immediately. You will need to contact the carrier so that it can inspect the shipping carton and packing material. We will then arrange to repair or replace the unit.

Remove the PCL-832 interface card from its protective packaging carefully. Keep the antistatic package. Whenever you are not using the board, please store it in the packaging for protection.

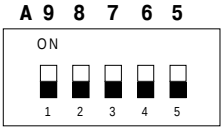
Warning! *Discharge any static electric charge on your body by touching grounded metal before you handle the board. You should avoid contact with materials that create static electricity such as plastic, vinyl, and styrofoam. Handle the board by its edges to avoid contacting the board's integrated circuits.*

Switch and jumper settings

Before you start using your PCL-832, you have to set the card's base I/O address, quadrature multiplier (for each channel), I/O wait state level and the interrupt levels for DDA time and the error counter.

Base I/O address (SW1)

The PCL-832 requires 32 consecutive I/O addresses. DIP switch SW1 (shown below) sets the base I/O address.



Choose a base address that is not in use by any other I/O device. A conflict with another device may cause one or both devices to fail. The factory address setting (hex 240) is usually free as it is reserved for PC prototype boards.

Switch settings for various base addresses appear below:

Card I/O addresses (SW1)

Range (hex)	Switch position				
	1	2	3	4	5
200 - 21F	●	○	○	○	○
220 - 23F	●	○	○	○	●
*240 - 25F	●	○	○	●	○
×					
300 - 31F	●	●	○	○	○
×					
3E0 - 3FF	●	●	●	●	●

○ = On ● = Off * = default

Note: Switches 1-5 control the PC bus address lines as follows:

Switch	1	2	3	4	5
Address Line	A9	A8	A7	A6	A5

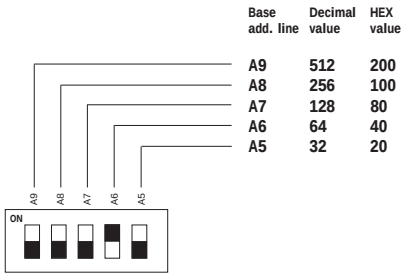
Calculating the base address

The base address is calculated using simple mathematics. The following example illustrates this.

Example

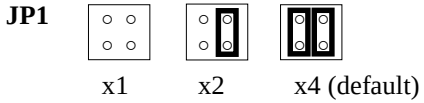
Calculate the base address of switch SW1.

Base address = 512+256+128+32 = 928 = 3A0 (Hex).



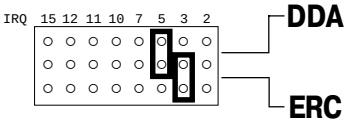
Quadrature input multiplier (JP1, JP2, JP3)

Jumper switches JP1, JP2 and JP3 select the quadrature input multiplier for channels 1, 2 and 3 respectively. Set these switches as below : (only channel 1, JP1, is shown)



DDA time and Error counter overflow interrupt levels (JP4)

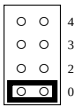
The jumper JP4 selects both the DDA time's and the Error counter's interrupt levels (2, 3, 5, 7, 10, 11, 12, 15), as shown below:



Do not select a level that is being used by another device unless you have performed special programming to share several devices on one interrupt.

I/O Wait state level (JP5)

Jumper JP5 (shown below) sets the wait state level of the PCL-832 to 0, 2, 3 or 4.

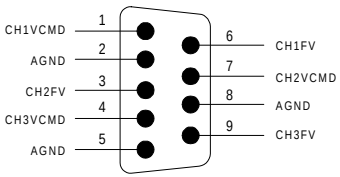


Connector pin assignments

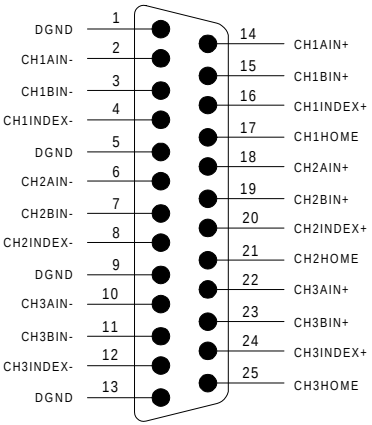
You make all connections to the PCL-832 through one DB-25 and one DB-9 connector, as shown below:

Connector pin assignments appear below: (only channel 1 shown - channels 2 and 3 are similar)

CN1



CN2



Pin	Function
AGND	Analog signal ground
CH1VCMD	Channel 1 voltage command output
CH1FV	Channel1 F/V output

DGND	Digital signal ground
CH1AIN+	Channel 1 A differential positive input
CH1AIN-	Channel 1 A differential negative input
CH1BIN+	Channel 1 B differential positive input
CH1BIN-	Channel 1 B differential negative input
CH1INDEX+	Channel 1 index differential positive input
CH1INDEX-	Channel 1 index differential negative input
CH1HOME	Channel 1 home sense input

Connector wiring

Differential/single-ended input

With differential inputs connect the negative wire to the negative pin and the positive wire to the positive pin. For example, with channel 3 A connect the negative input wire to CH3AIN- and the positive wire to CH3AIN+.

With single-ended inputs connect the input to the positive pin and leave the negative pin open.

Test-Points and VRs

Test-Points

TP1	:	CH1 F/V output
TP2	:	CH2 F/V output
TP3	:	CH3 F/V output
TP4	:	CH1 DA output
TP5	:	CH2 DA output
TP6	:	CH3 DA output
TP7	:	Analog ground

VR

VR1	:	CH1 DA output full scale adjustment
VR2	:	CH1 DA output offset adjustment
VR3	:	CH1 F/V output full scale adjustment
VR4	:	CH1 F/V output offset adjustment
VR5	:	CH2 DA output full scale adjustment
VR6	:	CH2 DA output offset adjustment
VR7	:	CH2 F/V output full scale adjustment
VR8	:	CH2 F/V output offset adjustment
VR9	:	CH3 DA output full scale adjustment
VR10	:	CH3 DA output offset adjustment
VR11	:	CH3 F/V output full scale adjustment
VR12	:	CH3 F/V output offset adjustment

Hardware installation

Warning! *Disconnect power from your PC whenever you install or remove the PCL-832 or its cables*

Installing the card in your computer:

1. Turn off the computer and all peripheral devices (such as printers and monitors).
2. Disconnect the power cord and any other cables from the back of the computer. Turn the chassis so the back of the unit faces you.
3. Remove the chassis cover (see your computer users guide if necessary).
4. Locate the expansion slots at the rear of the unit and choose an unused slot.
5. Remove the screw that secures the expansion slot cover to the chassis. Save the screw to secure the PCL-832.
6. Carefully grasp the upper edge of the PCL-833 card. Align the hole in the retaining bracket with the hole on top of the expansion slot, and align the gold striped edge connector with the expansion slot socket. Press the board firmly into the socket.
7. Replace the screw in the expansion slot retaining bracket.
8. Attach necessary accessories to the card.
9. Replace the chassis cover. Connect the cables you removed in step 2. Turn on the computer.

Hardware installation is now complete. The installation of the software is described in Chapter 4.

CHAPTER 3

Operation

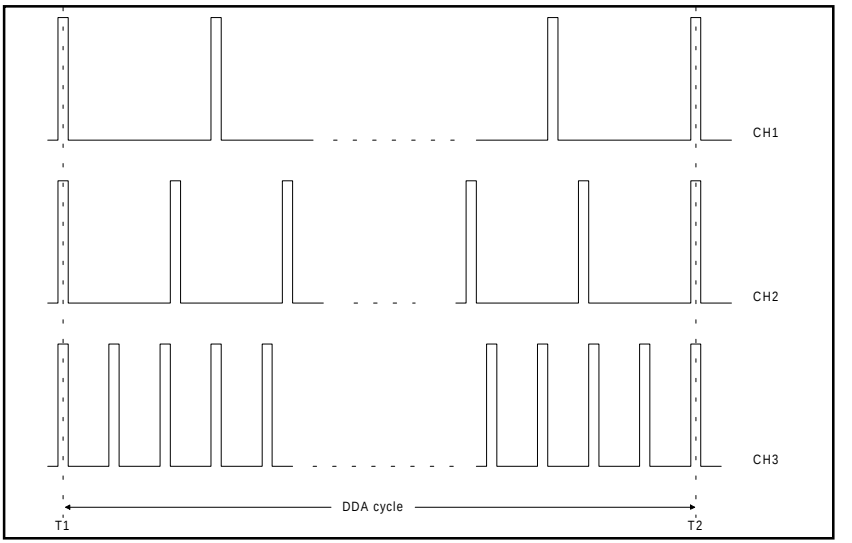
PCL-832 Operating Principles

Digital Differential Analysis

To understand how the PCL-832 works, it is necessary to understand the principles of DDA (Digital Differential Analysis).

In order to obtain synchronization in a multi-joint servo motor system, all axis start sending position commands simultaneously (T1), and stop sending these commands simultaneously (T2) as well. The duration of this period (T2 - T1) is defined as one DDA cycle. The duration of the DDA cycle can be set (by software) from 1 msec. to 2 seconds.

Consider the figure below. For every pulse output, the servo motor driver will advance the servo motor "one step" (*n* degrees). One pulse therefore represents one position-command. You can set the number of pulses in one DDA cycle, from 0 to 4095. The number of pulses in one DDA cycle represents the total position change possible by that DDA cycle.

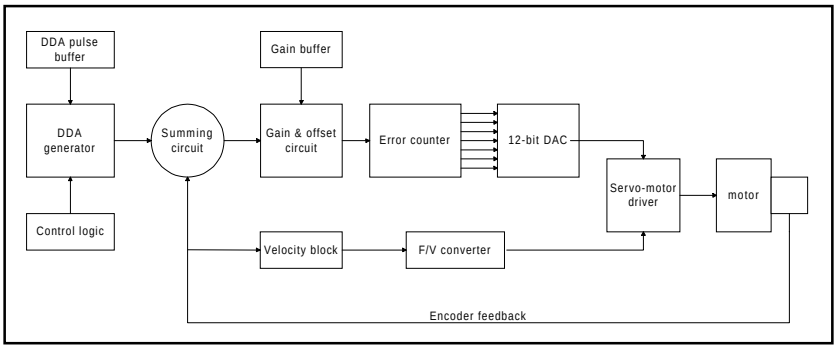


The continuous pulse sequence output to the servo motor driver ensures that a smooth position response is obtained. You determine the direction of the motion by pulsing the CMD+ or CMD- channel. The above work will be done by the motion control chip itself, you only need to write the position information to the motion control chip buffers before the DDA cycle starts.

Closed-loop position control

The PCL-832 uses proportional closed-loop position control to obtain reliable and accurate results. It features an internal velocity feedback loop and offset techniques to compensate for the steady-state error that is caused by using small values on the P controller.

The functional block diagram of the PCL-832 motion control card is shown below:



The DDA generates continuous command pulses through CMD+ and CMD- channels (not shown on the figure). These command pulses are fed into a summing circuit, together with the pulses generated by the servo motor encoder device. The summing circuit determines the difference between the two signals. The computed result is then fed into the P pulse-offset controller. The P pulse-offset controller, which has programmable gain (Kp), outputs a pulse. This pulse is fed into the error counter, which drives the DAC chip in real-time.

This is a complete closed-loop position control system.

A velocity block is provided in the motion control chip. Its purpose is to add a velocity feedback loop in the whole system through a frequency-to-voltage (F/V) converter. This internal loop impoves the motion dynamics of the servo motor system.

These facilities make the PCL-832 very powerful and an easy-to-use solution for your servo application needs.

Operation

In this section we take a look at how the PCL-832 accomplishes servo motor control.

When the DDA cycle generator is enabled, the PCL-832 will generate an interrupt at the start of the next (completion of the current) DDA cycle (rising edge triggered). When this happens the figure set in the DDA pulse buffer, *n*, is transferred to the DDA generator. This is the number of pulses that will be output during the next DDA cycle.

You have to set a number in the DDA pulse buffer **before** the current DDA cycle finishes, otherwise the next DDA cycle will output no pulses.

The DDA generator generates the first of the *n* pulses. The pulse is transferred to the Gain & offset circuit by the summing circuit. The gain circuit can be seen as a programmable divider, which you can control by software.

The gain circuit outputs a pulse to the Error counter. The error counter increases/descreases (depending on the direction +/- of the pulse) its value. The DAC is directly driven by the error counter, therefore when the error counter's value is increased, the output voltage of the DAC also increases. The servo-motor driver should respond by issuing a step command to the motor.

If the motor responds and performs the required movement, a pulse is generated by the encoder on the motor. This signal is fed back to the summing circuit *as a signal of the opposite direction.* In other words - if the original pulse was clockwise (+), then the feedback to the

summing circuit will be anti-clockwise (-).

This signal is again fed to the error-counter, through the gain circuit, which causes the error-counter to be decremented/incremented (depending on the direction). If the whole operation was successful, the error counter will therefore have a zero value at completion of the pulse/motion.

It the motor does NOT respond, no signal is fed back to the summing circuit, the error counter does not decrement, and a voltage will still be output on the DAC. When the next pulse reaches the error counter (through all the steps discussed), it will increment the value once again. The output voltage on the DAC will therefore be higher, prompting the servo motor (or perhaps the servo driver) to respond.

When the error counter reaches its maximum limit, 4095, it means that there is a problem with the servo motor, the servo motor driver or the encoder on the servo motor. Operation will be halted at this stage, and the PCL-832 will have to be reset before normal operation can continue.

The DDA cycle time

As previously discussed, the DDA cycle time refers to the time that elapses between the starting of the pulse sequence, and the completion of pulse output (for that cycle). To ensure synchronization between all three channels, one DDA cycle time is used for all three channels.

To calculate the DDA cycle time you use the following formula :

DDA cycle time = 0.512ms x Register value (BASE + 04)

The register value referenced above can be set from 2 to 4094 (in steps of 2) by writing the value into the register's address (or using one of the library functions to do that). This means that the DDA cycle time can be set from 1ms to 2 seconds. Remember that you have to set up the DDA pulse buffer (with the next DDA cycle's no. of pulses) before the completion of the current DDA cycle.

If the DDA cycle time is not set prior to operation, the default value of 1.024 ms (0.512 ms x 2) will be used. As your program probably has to do some other work as well (like displaying the coordinates of the motor etc.), it is best not to use the default value. This time can also be too small for the type of processor you use. The overhead that the processor has to cope with varies from application to application, with many variables. The programming language you use, other tasks that are running, your application program itself etc. are all factors that influence the minimum DDA cycle time. A DDA cycle time of 24ms is a value often used in PCL-832 applications.

The DDA pulse buffer

Even though the DDA cycle time is constant for all three channels, the number of pulses output in the DDA cycle, is set for every channel. Each channel therefore has its own DDA pulse buffer that contains the desired number of pulses in the next DDA cycle, for that channel.

When the DDA cycle starts, the number in the DDA pulse buffer is transferred from the buffer to the DDA pulse generator. The DDA pulse buffer is cleared as soon as the value has been transferred to the pulse generator. If you do not set a value in the DDA pulse buffer before completion of the current DDA cycle, no pulses will be output during the next DDA cycle.

The DDA pulse generator outputs the required number of pulses evenly over the DDA cycle. This means that the time between pulses will be (DDA cycle time / number of pulses) seconds.

The DDA pulse buffer format is shown below:

DDA Pulse buffer register format

B15	B14-B12	B11 ~ B0
DIR	N/A	D11 ~ D0

B15 ~ B0 Bit numbers

DIR DDA pulse direction
0 - clockwise (CW)
1 - anti-clockwise (CCW)

D11 ~ D0 DDA pulse number

The scaling gain

Each of the three PCL chips has its own scaling gain. The scaling gain lets you set the gain for each channel independently, and enables the use of different gains accross different applications. The scaling gain works as follows: when the SG is 1, the Error Counter is stepped up/down with every pulse from the DDA pulse generator or the encoder feedback. When the scaling gain is *n*, where $1 < n < 64$, the Error Counter will only be updated with every n^{th} pulse. A larger scaling gain reduces the system response, and position accuracy is $\pm n$ counts.

The Error counter

Each channel also has an independant 13-bit error counter. This counter records the difference between the number of desired motor movements, and the actual movements as registered by the servo motor's encoder. The output of the error counter drives a DAC (digital-to-analog convertor) which in turn is output to the servo motor driver.

The layout of the error counter is as follows :

Error counter register format

B15	B14-B12	B11 ~ B0
DIR	N/A	E11 ~ E0

B15 ~ B0 Bit numbers

DIR DDA pulse direction
0 - clockwise (CW)
1 - anti-clockwise (CCW)

E11 ~ E0 Error counter value

As previously discussed, the error counter will accumulate with actual/desired pulse differences. When the error counter reaches its limit (4095 if direction is 0, <0 if direction is 1), an overflow error occurs and the counter is cleared. You will need to reset the PCL-832 when this happens.

DAC ranges

The table below shows the three ranges that output from the DAC can be classified in. They are the CW saturation area, active area and the CCW saturation area.

D15	D11~D0	DA OUT (Vcmd)		
0	FFF	+9.995 V	full scale	CW Saturation
0	FFE	+9.995 V	full scale	
0	801	+9.995 V	full scale	
0	800	+9.995 V	full scale	
0	7FF	+9.995 V	full scale	Active Area
0	7FE	+9.990 V	full scale - 1LSB	
0	'002	+9.765 mV	+2 LSB	
0	'001	+4.883 mV	+1 LSB	
0	'000	0V	0	
1	FFF	-4.883 mV	-1 LSB	
1	FFE	-9.765 mV	-2 LSB	
1	801	-9.995 V	-full scale - 1 LSB	
1	800	-10.0 V	-full scale	
1	7FF	-10.0 V	-full scale	CCW Saturation
1	7FE	-10.0 V	-full scale	
1	'002	-10.0 V	-full scale	
1	'001	-10.0 V	-full scale	

DAC ranges

The Status register

The PCL-832 provides a status register for each channel. This register contains information on the channel's sensor home, encoder index input and the error counter overflow flag. If an overflow has occured, you will have to reset this flag (which will be 0) to 1, by issuing the reset command (BASE+0x1A).

The frequency-to-voltage (F/V) converter

In many servo motor applications the tachometer (TG) feedback is used to improve system performance. The PCL-832 provides an alternative to this method, as it features a F/V converter. This converter uses feedback from the servo motor to produce an output that can be applied to the servo motor driver's velocity feedback port.

This is called closed-loop velocity feedback.

To use this function your servo driver has be equipped with a F/V input port, or the TG circuitry has to be modified to meet the F/V output specifications.

The relationship between F/V convertor output voltage and encoder output frequency is calculated as follows :

Motor maximum velocity :	3000 RPM
Encoder resolution:	1000 Pulses / revolution
PCL-832 AB phase multiplier:	x2

Maximum encoder pulse output:

3000 RPM / 60 sec	= 50 Revolution / sec.
Encoder output	= 50 x 1000 = 50 K pulses / sec.
F/V input pulse rate	= 50 K x 2 = 100 K pulses / sec.

You have to calibrate the F/V 's full scale voltage output to 10V or -10V with the F/V input at 100KHz. You also have to adjust the F/V offset to 0V when there is no encoder output. As system conditions may vary, it is best to calibrate your PCL-832 under various conditions.

Using multiple PCL-832s in one system

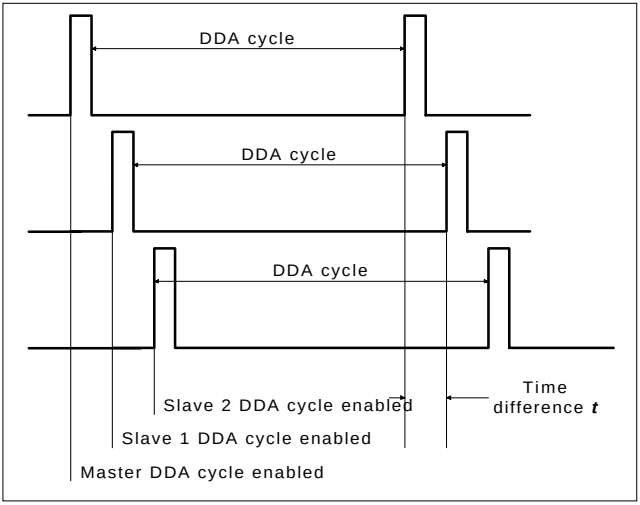
Installation

When using multiple PCL-832 cards in one system, the cards have to be setup as follows:

Select one card as the master card
Setup the interrupt levels for the DDA cycle time and the error counter overflow on the master card (using JP4)
Leave the slave cards' DDA cycle time interrupt **open**
Set the slave cards' error counter overflow interrupt the same as the master card's
Remove resistor R53 from all the slave cards

Programming

The instructions to enable the cards' DDA cycles must be called as close in succession as possible. The time difference between the DDA cycles of the cards should be as small as possible. The following diagram shows the time differences between the DDA cycles of the cards.



DDA cycle time differences

CHAPTER 4

Programming

Programming the PCL-832

There are two ways to program the PCL-832: using the driver programs included on the utility disk and accessing the card's registers directly. Using the drivers simplify programming to a large extent. The driver functions handle the necessary I/O port instructions and you only have to call them from your program.

Accessing the I/O ports directly requires a better understanding of the PCL-832's operation and hardware, register structure and data format. You will also need a basic understanding of PC interrupts and the DDA principles. Although this is more complicated than using the supplied drivers, it gives you all the flexibility you need to customize the control for your specific application.

Software installation

Insert the utility floppy disk in your PC's floppy drive. Let's assume it's drive B.

Create a directory on your hard disk where the files will be stored, eg. C:\PCL832, and change to that directory.

The DOS commands will be

```
cd\
md PCL832
cd PCL832
```

Now you have to copy all the files from the floppy drive to your hard drive. We use XCOPY because it creates all the subdirectories, and this simplifies the copy process.

The DOS command will be

```
XCOPY B:\. /s
```

The files should now be copied to the following directories:

`C:\PCL832\MSC60.C\DEMO`	MicroSoft C examples
`C:\PCL832\MSC60.C\LIB`	MicroSoft C Libraries
`C:\PCL832\TURBOC`	Turbo C include file and example program

Microsoft C drivers and Turbo C drivers

The driver routines supplied with your PCL-832 were written for use with Microsoft C and Turbo C. The functions covered by the Microsoft C library are divided into system functions, preparatory functions, motion functions and utility functions. The next two sections describe the use of these functions.

The third section describes the Turbo C low-level drivers. These drivers were written in Turbo C, and although only the basic functions have been supplied, they should provide an adequate backbone for your application. The source code is also supplied, which you can use and customize as needed.

However, if you do not have a basic understanding of the concepts of how the PCL-832 operates, the explanations/instructions given may not make a lot of sense. It is advisable to read chapters 3 and 5 before you go any further.

Special Procedures

When you use the Microsoft C libraries, the following is important:

1) The function `main()`is already used by the driver. Use `usermain()` instead

2) The 8253 counter 0 (the IRQ generator in the PC) is used by the driver, and the original value is modified for driver purposes. DO NOT modify this register's value.

3) On the software diskette, there are three parameter files that are used by the driver. Copy these files to the program directory otherwise the driver will not work. The next section gives a brief description of the parameter files.

4) When using the driver, reference is made to an incremental coordinate system and an absolute coordinate system. When you operate in the absolute coordinate system, references are made to three fixed axis. In the incremental system, any position change commands will occur relative to the current position of the motors.

5) The libraries require a numeric co-processor to be present.

The Microsoft C Driver Parameter Files

There are three parameter files, namely PCLB.PAR, HOME.PAR and MACHINE.PAR. Their contents is described below. To change the parameter files to suit your application, use any ASCII editor to make the modifications.

PCLB.PAR

This file contains information on the PCL-832 card.

File contents

```
Base address =     240h
DDA irq no.  =        5
Over irq no. =        3
GAIN1 =        1
GAIN2 =        1
GAIN3 =        1
```

Parameter descriptions

Parameter	Function
Base address	The base address of the PCL-832
DDA irq no.	The DDA cycle generator interrupt number
Over irq no.	The error counter overflow interrupt number
GAIN1 GAIN2 GAIN3	The gain for each channel

HOME.PAR

This file contains information on the home position of the servo system.

File contents

```
P dir - X  = 0 (0-7)
P dir - Y  = 0 (0-7)
P dir - Z  = 0 (0-7)
S on - X  = 1
S on - Y  = 1
S on - Z  = 1
I on - X   = 1
I on - Y   = 1
I on - Z   = 1
```

Parameter descriptions

Parameter	Function
P dir - X P dir - Y P dir - Z	This parameter consists of three bits. The purpose of the bits are as follows: **Bit1**: When the 'Go Home' command is received, the motor will move in this direction, until it finds home. 0 = CW, 1 = CCW
	Bit2: When the motor arrives at the home position, it stops, and then starts moving in this direction to leave home sense. 0 = CW, 1 = CCW
	Bit3: When the motor leaves home, it stops, and then starts moving in this direction to find the motor index. This is necessary because the mechanical home switch (maybe a broken light-beam etc) is very wide (eg.100 pulses). The first index position found is therefore a much more reliable reference position. 0 = CW, 1 = CCW
S on - X S on - Y S on - Z	Home signal - active high or active low 0 = active low 1 = active high
I on - X I on - Y I on - Z	Index signal - active high or active low 0 = active low 1 = active high

MACHINE.PAR

The parameters in this file describe the physical environment that the servo system is used in.

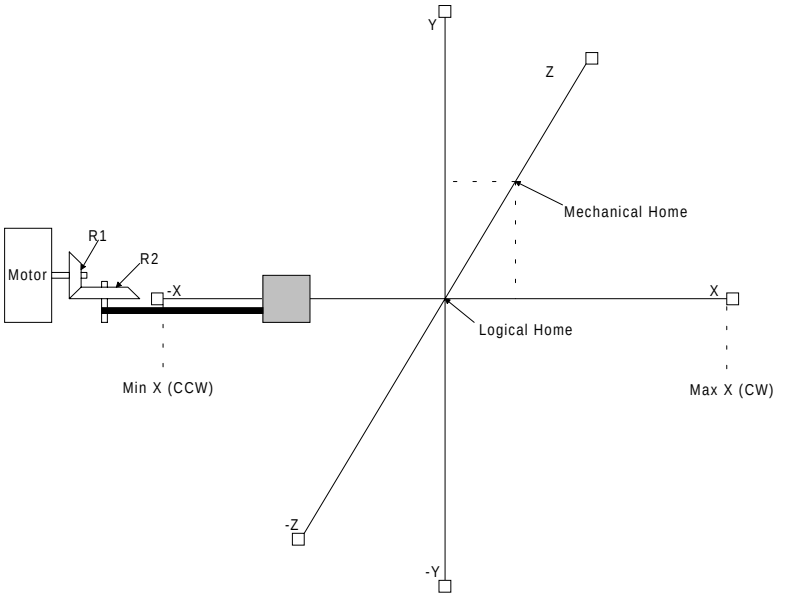
File contents

```
P001 = 4500.000
P003 =        0
P011 =   10.000
P012 =   10.000
P013 =   10.000
P021 =    1.000
P022 =    1.000
P023 =    1.000
P031 =     1000
P032 =     1000
P033 =     1000
P041 =     1000
P042 =     1000
P043 =     1000
P051 =     5000
P052 =     5000
P053 =     5000
P061 =    -5000
P062 =    -5000
P063 =    -5000
P071 =    0.000
P072 =    0.000
P073 =    0.000
```

Parameter descriptions

Parameter	Function
P001	maximum speed (mm/sec)
P003	logical and mechanical direction (1=reversed) 3 bits: bit0 = 1st, bit1 = 2nd, bit3 = 3rd axis
P011, P012, P013	distance moved per revolution (mm/rev)
*P021, P022, P023	gear ratio R2/R1 (motor/device)
P031, P032, P033	#pulses per revolution
P041, P042, P043	RPM
*P051, P052, P053	max. length in CW (+) direction
*P061, P062, P063	max. length in CCW (-) direction
*P071, P072, P073	offset between logical and physical origin (mm)

* - refer to the diagram on the next page



Servo motor system - physical layout

Microsoft C driver functions

This section describes the Microsoft C library functions that are supplied with the PCL-832. If you do not use Microsoft C, these functions will not be of much use to you, as the source code is not included.

Each function is also illustrated in various program examples. The relevant example names appear in square brackets **[ ]** after the function name. These example files are located in the \PCL832\DEMO directory.

System functions

MCC_init_motion [all]

Function: This function initializes the PCL-832, and must be called before any other functions

Syntax: `int MCC_init_motion()`

Return value: 0 if succesful, other value if not

MCC_close_motion [all]

Function: Closes motion control functions. This function should be called when operation is complete. After this function is called, other functions will not respond as intended.

Syntax: `int MCC_close_motion()`

Return value: 0 if successful, other value if not

Preparatory functions

MCC_set_dda [MEX2.C]

Function: Set DDA cycle time

Syntax: `unsigned int MCC_set_dda(unsigned int t)`

Parameter: t : value of DDA cycle time, unit:msec

Return value: *t* if successful, other value if not

MCC_get_dda [MEX2.C]

Function returns the DDA cycle time

Syntax
```
unsigned int MCC_get_dda()
```

Return value the current DDA cycle time

MCC_get_errcnt [MEX3.C]

Function gets the error counter value of each axis

Syntax
```
int MCC_get_errcnt(int *cnt1,
                   int *cnt2,int *cnt3)
```

Parameter
*cnt1 : a pointer to the 1st axis error counter value

*cnt2 : a pointer to the 2nd axis error counter value

*cnt3 : a pointer to the 3rd axis error counter value

Return value 0 if successful, other value if not

MCC_set_fspd [MEX4.C]

Function specifies the move speed

Syntax
```
float MCC_set_fspd(float fspd)
```

Parameter fspd : the move speed, unit mm/sec

Return value *fspd* if successful, other value if not

MCC_get_fspd [MEX4.C]

Function gets the move speed

Syntax
```
float MCC_get_fspd()
```

Return value current move speed if successful, <0 if not

MCC_set_acc_step [MEX4.C]

Function	specifies the acceleration step
Syntax	`int MCC_set_acc_step(int step)`
Parameter	step : the step of DDA cycle
Return value	acceleration step if successful, other value if not

MCC_get_acc_step [MEX4.C]

Function	gets the acceleration step
Syntax	`int MCC_get_acc_step()`
Return value	acceleration step if successful, <0 if not

MCC_set_dec_step [MEX4.C]

Function	sets the deceleration step
Syntax	`int MCC_set_dec_step(int step)`
Parameter	step : the step of DDA cycle
Return value	deceleration step if successful, other value if not

MCC_get_dec_step [MEX4.C]

Function	gets the deceleration step
Syntax	`int MCC_get_dec_step()`
Return value	deceleration step if successful, <0 if not

MCC_set_ptp_spd [MEX5.C]

Function	specifies the percentage ratio of the point to point movement speed, relative to P001 in MACHINE.PAR
Syntax	`int MCC_set_ptp_spd(int ratio)`
Parameter	ratio : movement speed ratio
Return value	*ratio* if successful, other value if not

MCC_set_abs [MEX12.C]

Function specifies absolute coordinate system

Syntax `MCC_set_abs()`

MCC_set_inc [MEX12.C]

Function specifies incremental coordinate system

Syntax `MCC_set_inc()`

MCC_get_pos_type [MEX12.C]

Function gets the current coordinate system mode

Syntax `int MCC_get_pos_type()`

Return value
1 : absolute coordinate system mode
0 : incremental coordinate system mode

Motion functions

MCC_go_home [MEX12.C]

Function homes the machine

Syntax
```
int MCC_go_home(float spd1,
float spd2, float spd3,
int odr1, int odr2, int odr3)
```

Parameter
spd1 : 1st axis return home speed (mm/sec)
spd2 : 2nd axis return home speed (mm/sec)
spd3 : 3rd axis return home speed (mm/sec)
odr1 : the priority of 1st axis return home
odr2 : the priority of 2nd axis return home
odr3 : the priority of 3rd axis return home

priority level 1, 2 or 3
set the priority level to 255 if the axis is not used

Return value 0 if successful, other value if not

MCC_check_home [MEX12.C]

Function — checks if the MCC_go_home function has completed

Syntax — `int MCC_check_home()`

Return value — 0 : function still in progress
1 : function has completed
2 : MCC_go_home was not called

MCC_line [MEX4.C]

Function — moves the current point to a specified point by linear interpolation

Syntax — `int MCC_line(float dx1,float dx2, float dx3)`

Parameter —
dx1 : the end position of X axis in absolute coordinate mode
the movement length of X axis in incremental coordinate mode
dx2 : the end position of Y axis in absolute coordinate mode
the movement length of Y axis in incremental coordinate mode
dx3 : the end position of Z axis in absolute coordinate mode
the movement length of Z axis in incremental coordinate mode

Return value — 0 if successful, other value if not

MCC_line_x [MEX4.C]

Function moves the X axis from the current point to a specified point by linear interpolation

Syntax `int MCC_line_x(float dx)`

Parameter dx : the end position of X axis in absolute coordinate mode

the movement length of X axis in incremental coordinate mode

Return value 0 if successful, other value if not

MCC_line_y [MEX4.C]

Function moves the Y axis from the current point to a specified point by linear interpolation

Sytax `int MCC_line_y(float dx)`

Parameter dx : the end position of Y axis in absolute coordinate mode

the movement length of Y axis in incremental coordinate mode

Return value 0 if successful, other value if not

MCC_line_z [MEX4.C]

Function moves the Z axis from the current point to a specified point by linear interpolation

Sytax `int MCC_line_z(float dx)`

Parameter dx : the end position of Z axis in absolute coordinate mode

the movement length of Z axis in incremental coordinate mode

Return value 0 if successful, other value if not

MCC_circle_xy [MEX6.C]

Function plots a circle on the xy plane

Sytax
```
int MCC_circle_xy(float cx1,
                  float cx2)
```

Parameter cx1, cx2 : the center point of the circle in xy plane

Return value 0 if successful, other value if not

MCC_circle_zx [MEX6.C]

Funtcion plots a circle on the zx plane

Syntax
```
int MCC_circle_zx(float cx3,
                  float cx1)
```

Parameter cx3, cx1 : the center point of the circle in zx plane

Return value 0 if successful, other value if not

MCC_circle_yz [MEX6.C]

Function plots a circle on the yz plane

Syntax
```
int MCC_circle_yz(float cx2,
                  float cx3)
```

Parameter cx2, cx3 : the center point of the circle in yz plane

Return value 0 if successful, other value if not

MCC_arc_xy [MEX7.C]

Function plots an arc on the xy plane

Syntax
```
int MCC_arc_xy(float rx1,
 float rx2, float dx1, float dx2)
```

Parameter rx1, rx2 : any point of the arc between current point and end point

dx1, dx2 : the end point of arc in xy plane

Return value 0 if successful, other value if not

MCC_arc_zx [MEX7.C]

Function: plots an arc on the zx plane

Syntax:
```
int MCC_arc_zx(float rx3,
 float rx1, float dx3, float dx1)
```

Parameter:
- rx3, rx1 : any point of the arc between the current point and end point
- dx3, dx1 : the end point of arc in zx plane

Return value: 0 if successful, other value if not

MCC_arc_yz [MEX7.C]

Function: plots an arc on the yz plane

Syntax:
```
int MCC_arc_yz(float rx2,
 float rx3, float dx2, float dx3)
```

Parameter:
- rx2, rx3 : any point of the arc between current point and end point
- dx2, dx3 : the end point of arc in yz plane

Return value: 0 if successful, other value if not

MCC_ptp [MEX5.C]

Function: moves from current point to a specified point at maximum velocity and acceleration

Syntax:
```
int MCC_ptp(float dx1, float dx2,
                              float dx3)
```

Parameter:
- dx1, dx2, dx3 : the end position in absolute coordinate mode
- the movement length in incremental coordinate mode

Return value: 0 if successful, other value if not

MCC_ptp_x [MEX5.C]

Function	moves the x axis current point to a specified point at maximum velocity and acceleration		
Syntax	`int MCC_ptp_x(float dx)`		
Parameter	dx	:	the end position of X axis in absolute coordinate mode the movement length of X axis in incremental coordinate mode
Return value	0 if successful, other value if not		

MCC_ptp_y [MEX5.C]

Function	moves the y axis current point to a specified point at maximum velocity and acceleration		
Syntax	`int MCC_ptp_y(float dx)`		
Parameter	dx	:	the end position of Y axis in absolute coordinate mode the movement length of Y axis in incremental coordinate mode
Return value	0 if successful, other value if not		

MCC_ptp_z [MEX5.C]

Function	moves the z axis current point to a specified point at maximum velocity and acceleration		
Syntax	`int MCC_ptp_z(float dx)`		
Parameter	dx	:	the end position of Z axis in absolute coordinate mode the movement length of Z axis in incremental coordinate mode
Return value	0 if successful, other value if not		

MCC_motion_delay [MEX12.C]

Function: delays for a specified time

Syntax:
```
int MCC_motion_delay(unsigned int t)
```

Parameter: t : the delay time value, unit : 0.1sec

Return value: 0 if successful, other value if not

MCC_jog_i [MEX8.C]

Function: moves a specified # pulses in one DDA cycle

Syntax:
```
int MCC_jog_i(char x, char d,
                        float d_pulse)
```

Parameter: x(x = 1, 2, 3) : specifies the axis

d : the movement direction

d = +1 CW
d = -1 CCW

d_pulse : # pulses

Return value: 0 if successful, other value if not

MCC_jog_s [MEX9.C]

Function: moves a specified distance at a specified speed

Syntax:
```
int MCC_jog_s(char x, char d,
          int s_ratio, float d_mm)
```

Parameter: x(x = 1, 2, 3) : specifies the axis

d : the movement direction

d = +1 CW
d = -1 CCW

d_mm : movement distance (unit : mm)

s_ratio : movement speed ratio to P001 in MACHINE.PAR

Return value: 0 if successful, other value if not

MCC_jog_c [MEX10.C]

Function: moves at a specified speed

Syntax:
```
int MCC_jog_c(char x, char d,
                              int s_ratio)
```

Parameter:
- x(x = 1, 2, 3) : specifies the axis
- d : the movement direction
  - d = +1 CW
  - d = -1 CCW
- s_ratio : the movement speed ratio (%)

Return value: 0 if successful, other value if not

Note: The motor will keep on moving at the specified speed until MCC_motion_hold is executed. To abort the movement, call MCC_motion_abort.

Motion function return codes

The return codes of the motion functions have the following meaning:

0: Function successful
-1: The system queue buffer is full. Call the function again
-2: An error has occurred. Call MCC_get_errcode to get the error code, and call MCC_clear_error to clear the error flag
-3: fspd = 0

Utility functions

MCC_motion_hold [MEX11.C]

Function: holds the current machine curve

Syntax:
```
int MCC_motion_hold()
```

Return value: 0 if successful, other value if not

MCC_motion_abort [MEX10.C]

Function: aborts the motion curve that was held

Syntax:
```
int MCC_motion_abort()
```

Return value: 0 if successful, other value if not

MCC_motion_conti [MEX11.C]

Function continues the motion that was held

Syntax
```
int MCC_motion_conti()
```

Return value 0 if successful, other value if not

MCC_get_cpos [MEX11.C]

Function gets the current motion position, unit : mm

Syntax
```
int MCC_get_cpos(float *x1,
                 float *x2, long *x3)
```

Parameter
- *x1 : the pointer to the X axis current position data
- *x2 : the pointer to the Y axis current position data
- *x3 : the pointer to the Z axis current position data

Return value 0 if successful, other value if not

MCC_get_ppos [MEX11.C]

Function gets the current motion position, unit : pulse

Syntax
```
int MCC_get_ppos(long *x1,
                 long *x2, long *x3)
```

Parameter
- *x1 : the pointer to the X axis current position data
- *x2 : the pointer to the Y axis current position data
- *x3 : the pointer to the Z axis current position data

Return value 0 if successful, other value if not

MCC_check_stop [MEX10.C]

Function	checks if the current motion curve has finished
Synatx	`int MCC_check_stop()`
Return value	0 : current motion still in progress 1 : no motion in progress

MCC_get_errcode

Function	Returns the last error code
Synatx	`int MCC_get_errcode(int *errcode)`
Parameters	*errcode : a pointer to an array of integers The array should have more than 20 elements
Error code list:	0xF101 : Error opening parameter file 0xF104 : *Arc* function parameters invalid 0xF201 : DDA interrupt error 0xF202 : Error counter overflow 0xF301 : X-axis out of range 0xF302 : Y-axis out of range 0xF303 : Z-axis out of range 0xF304 : *Circle or Arc* interpolation error 0xF601 : Contact technical support 0xF602 : Contact technical support
Return value	The error code

MCC_clear_error

Function	clear error buffer
Synatx	`int MCC_clear_error()`
Return value	0 if successful, other value if not

MCC_check_delay

Function	checks if the MCC_delay_motion has finished
Synatx	`int MCC_check_delay()`
Return value	0 : current motion_delay function in progress 1 : no motion_delay in progress

Turbo C low-level driver functions

This section describes the Turbo C low-level driver functions that were supplied on the utility disk. These functions are not in a library, they come in a source code file. You can directly include the source file into your program, modify it, compile it to a library etc.

The source code and an example file can be found in the \PCL832\DRIVER directory.

These are by no means a complete set of functions for the PCL-832. Their only purpose is to show you how easy it is to program the registers of the PCL-832 directly. The example program illustrates the use of these functions, and how to include them in a C program.

The following functions are available:

PCL832_reset

Function	Resets the PCL-832
Syntax	`PCL832_reset()`
Return value	none;

PCL832_enable_dda

Function	Enables the PCL-832 DDA pulse generator
Syntax	`PCL832_enable_dda();`
Return value	none;

PCL832_set_dda_cycle_time

Function	Sets the PCL-832 dda cycle time		
Syntax	`int PCL832_set_dda_cycle_time(int ddatime)`		
Parameter	ddatime	:	Specifies the DDA cycle time value(unit ms), 1 - 2000
Return value	0	:	success
	-1	:	dda time value error

PCL832_set_dda_pulse

Function: Sets the dda pulse number for next dda_cycle

Syntax:
```
int PCL832_set_dda_pulse(int
 axis, int dir, int dda_pulse)
```

Parameter:
- axis : Specifies the axis number 1,2 or 3
- dir : specified the move direction,
  0 = CW
  1 = CCW
- dda_pulse : no. of pulses, 0 - 4095

Return value:
- 0 : success
- -1 : axis number error
- -2 : direction error
- -3 : dda pulse number error

PCL832_set_gain

Function: Sets the gain for an axis

Syntax:
```
int PCL832_set_gain(int axis, int
                                gain)
```

Parameter:
- axis : axis number 1,2 or 3.
- gain : gain value, 0 - 63

Return value:
- 0 success
- -1 : axis number error
- -2 : gain value error

PCL832_get_errcnt

Function	Returns the error counter value		
Syntax	`int PCL832_get_errcnt(int axis, int *errcnt)`		
Parameter	axis	:	axis number 1,2 or 3.
	*errcnt	:	pointer to the error counter data
Return value	0	:	success
	-1	:	axis number error

PCL832_get_status

Function	Returns the home, index and error counter status		
Syntax	`int PCL832_get_status(int axis, int *status)`		
Parameter	axis	:	axis number 1,2 or 3.
	*status	:	pointer to the status data
			bit 0 : Home switch status bit 1 : Index status bit 2 : Error counter overflow status
Return value	0	:	success
	-1	:	axis number error

CHAPTER 5

Register structure and format

I/O Address space

The PCL-832 uses 32 consecutive addresses in the PC I/O address space. DIP switch SW1 sets the card's base, or beginning address. Specific I/O ports are referred to by their offset from the base address, BASE. For example, the address for the seventh register is BASE+6.

I/O address Map

The following table gives the assignment of each of the card's ports.

I/O port assignments

Port	Read	Write
BASE+0	CH1 error counter value	CH1 DDA pulse buffer
BASE+2	CH1 status	CH1 scaling gain
BASE+4		DDA cycle time
BASE+8	CH2 error counter value	CH2 DDA pulse buffer
BASE+0A	CH2 status	CH2 scaling gain
BASE+10	CH3 error counter value	CH3 DDA pulse buffer
BASE+12	CH3 status	CH3 scaling gain
BASE+18		DDA cycle gen. enable
BASE+1A		PCL-832 RESET

Register format

BASE + 0

Read CH1 error counter value

Base + 0: Register Layout (Read)

D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0

DIR : The direction of the movement
0 : CW
1 : CCW

E11 ~ E0 : The error counter value

Write CH1 DDA pulse buffer

Base + 0: Register Layout (Write)

D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

DIR : The direction of the DDA pulse
0 : CW
1 : CCW

D11 ~ D0 : The DDA pulse number

BASE+02

Read CH1 Status

Base + 02: Register Layout (Read)

D15 ~ D4	D3	D2	D1	D0
N/A	E-OV	N/A	Index	Home

E-OV : The Error counter overflow status
0 : An overflow has occurred
1 : An overflow has not occurred

INDEX : The index input status
0 : The index input is LOW
1 : The index input is HIGH

HOME : The home input status
0 : The home input is HIGH
1 : The home input is LOW

Write CH1 Scaling Gain

Base + 02: Register Layout (Write)

D15 ~ D8	D7	D6	D5	D4	D3	D2	D1	D0
N/A	N/A	N/A	S5	S4	S3	S2	S1	S0

S5 ~ S0 : The Scaling Gain value

BASE+04

Write The DDA cycle time multiplier

Base + 04: Register Layout (Write)

D15~D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
N/A	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0

F11 ~ F0 : The DDA cycle time value

DDA cycle time = 0.512ms * Register value (F11~F0)

BASE+08

Read CH2 Error counter value

Base + 08: Register Layout (Read)

D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0

DIR : The direction of movement
0 : CW
1 : CCW

E11 ~ E0 : The Error counter value

Write CH2 DDA pulse buffer

Base + 08: Register Layout (Write)

D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

DIR : The direction of DDA pulse
0 : CW
1 : CCW

D11 ~ D0 : The DDA pulse number

BASE+0A

Write CH2 Scaling Gain

Base + 0A: Register Layout (Write)

D15 ~ D8	D7	D6	D5	D4	D3	D2	D1	D0
N/A	N/A	N/A	S5	S4	S3	S2	S1	S0

S5 ~ S0 : The Scaling Gain value

BASE+0A

Read The CH2 Status

Base + 0A: Register Layout (Read)				
D15 ~ D4	D3	D2	D1	D0
N/A	E-OV	N/A	INDEX	HOME

E-OV : The Error counter overflow status
- 0 : An overflow has occurred
- 1 : An overflow has not occurred

INDEX : The index input stauts
- 0 : The index input is LOW
- 1 : The index input is HIGH

HOME : The home input status
- 0 : The home input is HIGH
- 1 : The home input is LOW

BASE+10

Read CH3 Error counter value

Base + 10: Register Layout (Read)													
D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0

DIR : The direction of movement
- 0 : CW
- 1 : CCW

E11 ~ E0 : The Error counter value

Write CH3 DDA pulse buffer

Base + 02: Register Layout (Write)													
D15	D14 ~ D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	N/A	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

DIR : The direction of DDA pulse
0 : CW
1 : CCW

D11 ~ D0 : The DDA pulse number

BASE+12

Read CH3 Status

Base + 12: Register Layout (Read)													
D15 ~ D4	D3	D2	D1	D0									
N/A	E-OV	N/A	INDEX	HOME									

E-OV : The Error counter overflow status
0 : The Error counter is overflow
1 : The Error counter is not overflow

INDEX : The index input stauts
0 : The index input is LOW
1 : The index input is HIGH

HOME : The home input status
0 : The home input is HIGH
1 : The home input is LOW

Write CH3 Scaling Gain

Base + 02: Register Layout (Read)								
D15 ~ D8	D7	D6	D5	D4	D3	D2	D1	D0
N/A	N/A	N/A	S5	S4	S3	S2	S1	S0

S5 ~ S0 : The Scaling Gain value

BASE+18

Write DDA cycle generator enable

You can write any value to this register, as external logic on the address lines enables the DDA cycle generator.

BASE+1A

Write PCL-832 reset

You can write any value to this register, as external logic on the address lines resets the PCL-832.

APPENDIX A

Calibration

Calibration

It is important to ensure that all measurement devices are calibrated regularly in order to maintain accuracy. A calibration program, CALB832.EXE, is provided on the PCL-832 software disk to assist with the calibration procedure.

The minimum equipment required to perform a satisfactory calibration is a 4½-digit digital multimeter.

Calibration is easily performed using the CALB832.EXE program. This program is very user-friendly and will guide you through the calibration and set-up procedure. A variety of prompts and graphic displays will direct you to make the appropriate adjustments.

You should use this section in conjunction with the CALB832.EXE program, as it does not contain precise calibration instructions.

VR Assignments

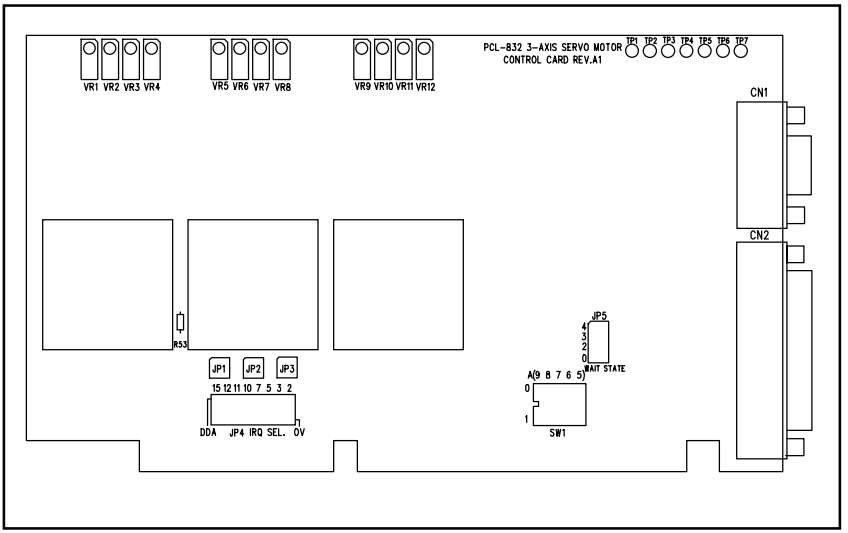
The PCL-832 has 12 on-board VRs (variable resistors). They enable you to make accurate calibration adjustments for each axis's DA offset, F/V offset and full-scale output. The location of each VR is indicated in the figure on the next page. The function of each VR is listed below:

VR1	:	CH1 DA output full scale adjustment
VR2	:	CH1 DA output offset adjustment
VR3	:	CH1 F/V output full scale adjustment
VR4	:	CH1 F/V output offset adjustment
VR5	:	CH2 DA output full scale adjustment
VR6	:	CH2 DA output offset adjustment
VR7	:	CH2 F/V output full scale adjustment
VR8	:	CH2 F/V output offset adjustment
VR9	:	CH3 DA output full scale adjustment
VR10	:	CH3 DA output offset adjustment
VR11	:	CH3 F/V output full scale adjustment
VR12	:	CH3 F/V output offset adjustment

Test-Points

The PCL-832 has seven test points. They enable you to connect the DAC or F/V output to a voltage meter when you are doing calibration. The location of each test-point is indicated in the figure below. The following can be measured at each test-point:

TP1	:	CH1 F/V output
TP2	:	CH2 F/V output
TP3	:	CH3 F/V output
TP4	:	CH1 DA output
TP5	:	CH2 DA output
TP6	:	CH3 DA output
TP7	:	Analog ground



VR assignments and test points

ProSoft

Москва: Телефон: (095) 234-0636 (4 линии)
Факс: (095) 234-0640
BBS: (095) 336-2500
Web: http://www.prosoft.ru
E-mail: root@prosoftmpc.msk.su
Для писем: 117313, Москва, а/я 81
С.-Петербург: (812) 325-3790
Екатеринбург: (3432) 49-3459