User Guide

MD29

Microprocessor card and software for
Mentor II, Vector, CDE
and HPCDE Drives

Part Number: 0400-0030
Issue Number: 4

Safety Information

Persons supervising and performing the electrical installation or maintenance of a Drive and/or an external Option Unit must be suitably qualified and competent in these duties. They should be given the opportunity to study and if necessary to discuss this User Guide before work is started.

The voltages present in the Drive and external Option Units are capable of inflicting a severe electric shock and may be lethal. The Stop function of the Drive does not remove dangerous voltages from the terminals of the Drive and external Option Unit. Mains supplies should be removed before any servicing work is performed.

The installation instructions should be adhered to. Any questions or doubt should be referred to the supplier of the equipment. It is the responsibility of the owner or user to ensure that the installation of the Drive and external Option Unit, and the way in which they are operated and maintained complies with the requirements of the Health and Safety at Work Act in the United Kingdom and applicable legislation and regulations and codes of practice in the UK or elsewhere.

The Drive software may incorporate an optional Auto-start facility. In order to prevent the risk of injury to personnel working on or near the motor or its driven equipment and to prevent potential damage to equipment, users and operators, all necessary precautions must be taken if operating the Drive in this mode.

The Stop and Start inputs of the Drive should not be relied upon to ensure safety of personnel. If a safety hazard could exist from unexpected starting of the Drive, an interlock should be installed to prevent the motor being inadvertently started.

General information

The manufacturer accepts no liability for any consequences resulting from inappropriate, negligent or incorrect installation or adjustment of the optional operating parameters of the equipment or from mismatching the variable speed drive (Drive) with the motor.

The contents of this User Guide are believed to be correct at the time of printing. In the interests of a commitment to a policy of continuous development and improvement, the manufacturer reserves the right to change the specification of the product or its performance, or the contents of the User Guide, without notice.

Copyright © November 1997 Control Techniques Drives Ltd
Author: CT SSPD
Originators AH, PB
Issue Code: 29nu4
Issue Date: November 1997
S/W Version: V2.6.0 system files and later.

Contents

1 Introduction

On a Variable Speed Drive (such as Mentor II, Vector or CDE), timing functions necessary for the correct operation of power devices are performed by its own microprocessor operating in real-time. This imposes limitations on the microprocessor when carrying out other duties, resulting in a reduction of flexibility of the Drive.

To maximize this flexibility, a second processor can be used for running application-specific software. This second microprocessor is the MD29 which allows the Drive to be easily adapted to applications by programming software in the MD29.

1.1 Overview

The MD29 is a compact microcomputer using surface-mount components on a single printed circuit board. The board is designed for easy installation.

Together with the DPL Toolkit, the MD29 allows the programmer to write software or use pre-written software in order to enhance the flexibility of a Variable Speed Drive.

The MD29 is compatible with the following Drives:

- Mentor II, Vector, CDE, HPCDE

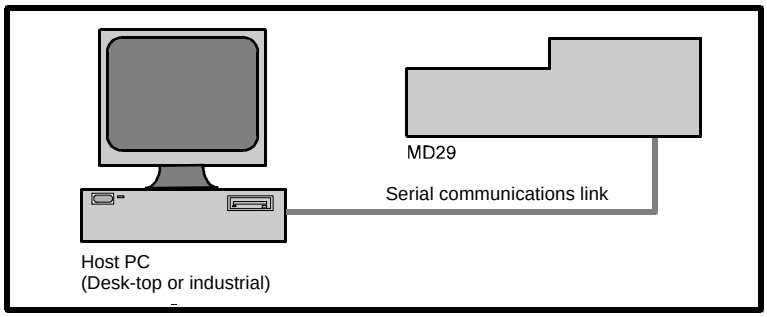
The MD29AN is a special version of the MD29. It has the full functionality of the MD29 plus a CTNet interface for Mentor II Drives only. This interface replaces the dedicated RS485 port for the Control Techniques I/O Box.

Note

The term MD29 in this manual also refers to the MD29AN, unless specified otherwise.

The terms CDE750 and CDE7500 are used in this manual. CDE750 refers to the bookcase CDE 0.75kW to 11kW range (1HP to 15HP); CDE7500 refers to the CDE 11kW to 90kW range (7.5HP to 150HP), including the HPCDE.

The DPL Toolkit is contained on two diskettes. It is a program which runs in Microsoft® Windows™ Version 3.1x and Windows™ 95. Programs for the MD29 are written on a host PC using the DPL Toolkit.



The MD29 uses a high-level programming language called **DPL** (Drive Programming Language) which is in many respects similar to the BASIC language. DPL is a compiled program which gives it the ability to run at high speed.

The DPL Toolkit is used to write, compile and download a DPL program to an MD29. The Toolkit also has a comprehensive set of de-bugging facilities to aid the development and testing of the DPL program.

Connection between the MD29 and host PC is via an RS232 serial communications link. This link need only be used during program development, testing and commissioning. It can be disconnected after the software has been successfully loaded.

1.2 Memory

The compiled MD29 program and the user-created source program are stored in non-volatile EEPROM memory on the MD29 card. This type of memory allows the programs to be loaded using the serial port.

Latest versions of programs can be easily updated without removing any integrated circuits or without using any specialized programming equipment.

The filing system of the MD29 allows only one program to be stored in the MD29 at any one time.

The compiled program can be stored along with the DPL source code. This allows the site engineer to read the program stored in the MD29, even if the program is not on the host PC. (This option can be disabled if it is not required.)

1.3 PC requirements

The **minimum** requirement for the DPL Toolkit is as follows:

IBM AT compatible 386SX PC, Windows™ 3.1, 4Mb RAM, DOS5

A 486 PC with 8MB RAM is recommended

1.4 Technical data for the MD29

Intel 960 32-bit RISC processor

96kb of user program storage

8kb user RAM

16MHz clock

RS232 port for programming (IBM AT compatible)

RS485 optically isolated port for permanent serial communications

Dedicated, optically-isolated RS485 port for a Control Techniques I/O Box (not available on MD29AN)

1.5 User knowledge

This User Guide assumes that the user has at least superficial knowledge of Microsoft® Windows™. Refer to the Windows User's Guide for specific information on performing operations in Windows™.

2 Installation

Warning

The voltages present in the Drive are capable of inflicting a severe electric shock and may be lethal. The Stop function of the Drive does not remove dangerous voltages from the Drive or the driven machine.

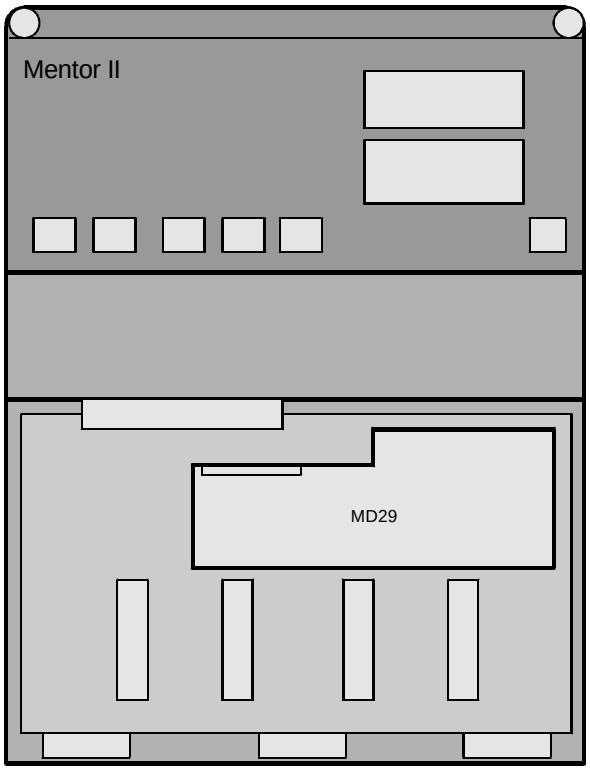
AC supplies to the Drive must be disconnected at least 15 minutes before any cover is removed or servicing work is performed.

2.1 Installation procedure

Installation procedures are given for the following Drives:

Mentor II, CDE7500/ HPCDE, Vector

Mentor II Drive



Location of the MD29 in the Mentor II Drive

Refer to the Mentor II User Guide for the mechanical details.

Use the following procedure to fit the MD29 to the MDA2B circuit board of the Drive:

1. Isolate the AC supply from the Drive.
2. Remove the front cover from the Drive.
3. Fit the four small securing pillars to the corners of the MD29.
4. Find the 40-pin header connector on the MDA2B circuit board of the Drive.

Note

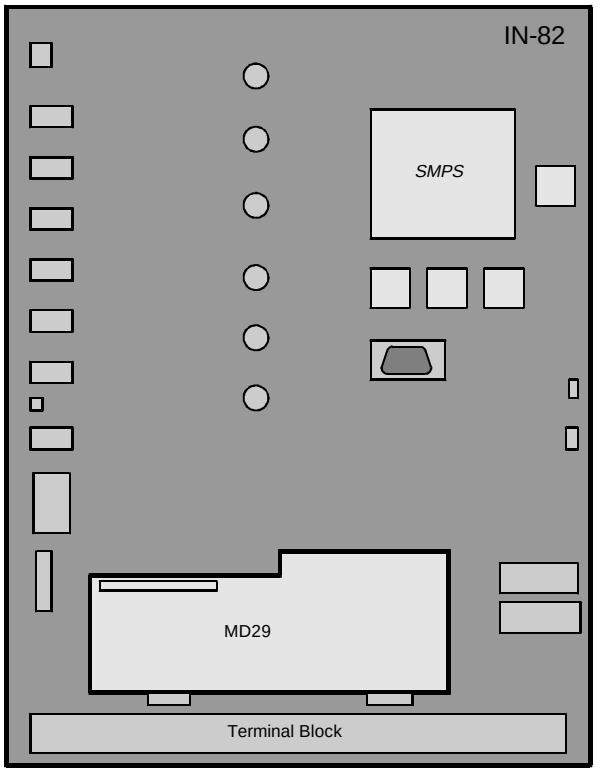
The following instruction requires you to fit the MD29 to the Drive. Correct location of the header connector is essential.

5. Fit the 40-pin connector of the MD29 on to the connector on the MDA2B circuit board, ensuring the pins are aligned, and that the pillars on the MDA2B circuit board are correctly aligned with the locating holes in the MD29.
6. Push the MD29 carefully into position.
7. Check again that the 40-pin connector is correctly plugged in.
8. Fit the four securing pillars on each corner of the MD29 to the Drive circuit board.
9. Make any necessary serial cable connections to the MD29.
10. Replace the Drive cover.

Refer to the *CDE* User Guide for the mechanical details.

Use the following procedure to fit the MD29 to the IN–82 control board in the Drive, or to the IN-90 board in the HPCDE Drive:

1. Isolate the AC supply from the Drive.
2. Remove the front cover from the Drive.
3. Fit the four long pillars to the corner holes of the MD29.
4. Find the 40-pin header connector on the IN–82 control board in the Drive.



Location of the MD29 on the IN-82 card in the CDE7500 Drive

Note

The following instruction requires you to fit the MD29 to the Drive. Correct location of the header connector is essential.

5 Fit the 40-pin connector of the MD29 on to the connector on the IN–82 control board, ensuring the pins are aligned, and that the pillars on the IN–82 control board are correctly aligned with the locating holes in the MD29.

6 Push the MD29 carefully into position.

7 Check again that the 40-pin connector is correctly plugged in.

8 Locate the four securing pillars on the corners of the MD29 to the IN–82 control board.

Note

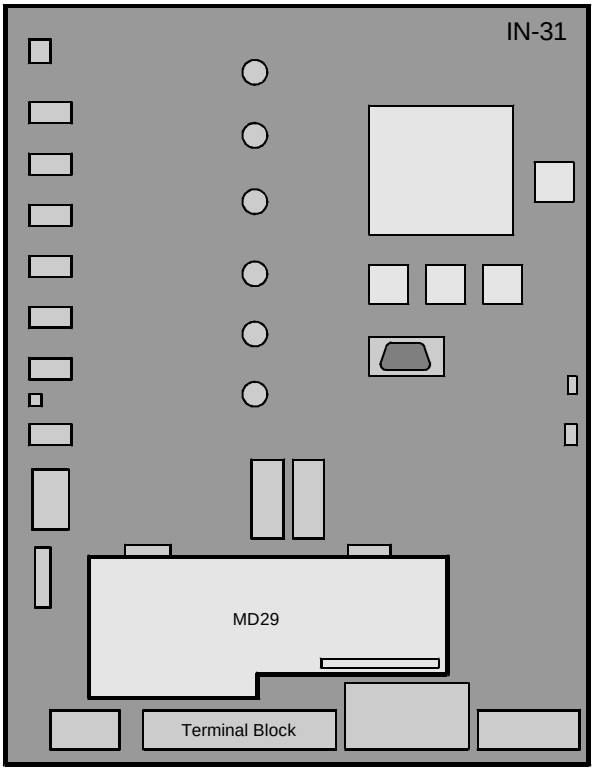
The Drive cover cannot be re-fitted directly to the case. The cover must be raised to allow room for the MD29 card. Use the following procedure to fit the cover.

9 Fit the four hole-stud pillars (supplied with the MD29) to the corners of the Drive.

10 Fit the pod extension connector (supplied with the MD29) to the D-type connector of the control pod.

11 Make any necessary serial cable connections to the MD29.

12 Fit the cover on to the pillars.



Location of the MD29 on the IN-31 control board in the Vector Drive

Refer to the Vector User Guide for mechanical details.

Use the following procedure to fit the MD29 to the IN-31 control board in the Drive:

1. Isolate the AC supply from the Drive.
2. Remove the front cover from the Drive.
3. Fit the four long pillars to the corner holes of the MD29.
4. Find the 40-pin header connector on the IN–31 control board in the Drive.

Note

The following instruction requires you to fit the MD29 to the Drive. Correct location of the header connector is essential.

5 Locate the 40-pin connector of the MD29 on to the connector on the IN–31 control board, ensuring the pins are aligned, and that the pillars on the IN–31 control board are correctly aligned with the locating holes in the MD29.

6 Push the MD29 carefully into position.

7 Check again that the 40-pin connector is correctly plugged in.

8 Fit the four securing pillars on the corners of the MD29 to the IN–31 control board.

Note

The Drive cover cannot be re-fitted directly to the case. The cover must be raised to allow room for the MD29 card. Use the following procedure to fit the cover.

9 Fit the four hole-stud pillars (supplied with the MD29) to the corners of the Drive.

10 Fit the pod extension connector (supplied with the MD29) to the D-type connector of the control pod.

11 Make any necessary serial cable connections to the MD29.

12 Fit the cover on to the pillars.

Bookcase CDE

It is recommended that fitting of the MD29 to bookcase Drives is carried out by an authorized distributor since internal access to the Drive is required.

2.2 Configuring the system

Host PC connections

RS232 Port The RS232 serial port is a dedicated link to the host PC. The port is a 9-way female D-type connector. Ready-made cables for RS232 serial communications are generally available.

The table below gives the minimum required connections between the MD29 and a 9-way and 25-way pin COM port connector.

MD29 pin no.	9-pin connector pin no.	25-pin connector pin no.
2	2	3
3	3	2
5	5	7

The RS232 port should be used only for commissioning because isolation or protection of the port is not included.

Use the following instructions for connecting a host PC:

1. Ensure no static charge has built up when the plug is inserted.
2. Using a maximum cable length of not more than 3 metres (10 feet), connect an RS232 cable to the RS232 serial port on the MD29 and to the communications serial port of the host PC.

Installing the DPL Toolkit in the host PC

Use either of the following procedures:

Windows 3.1

The DPL Toolkit requires a minimum of 4Mb of computer memory. This may be RAM or virtual memory. Virtual memory may be set in the **386 Enhanced** section of Windows Control Panel.

1. Start Microsoft Windows
2. Insert disk 1 of the DPL Toolkit into the **A:** drive of the host PC.
3. In Windows Program Manager, select **File** on the menu bar. Select **Run**.
4. Type **A:\SETUP.**
5. Click on 

Windows 95

1 Insert disk 1 of the DPL Toolkit into the **A:** drive of the host PC.

2 In the Start menu, select **Run...**.

3 Type **A:\SETUP**.

4 Click on OK

Downloading the system file

The MD29 has no pre-loaded system software. The first task is to program the system software using the DPL Toolkit. Use the following procedure:

1 Connect the serial communications cable to the MD29.

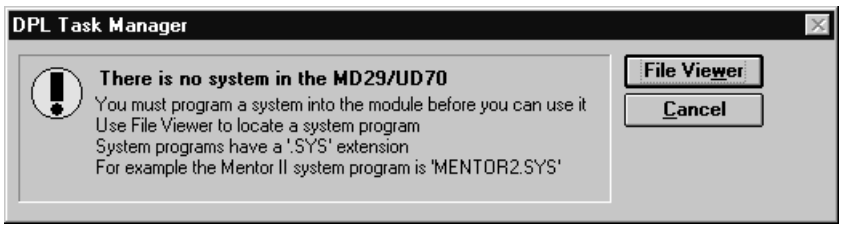
2 Apply AC power to the Drive.

3 In Windows 3.xx Program Manager, or in the Windows 95 Start menu, click on:

The **DPL Toolkit** window appears. At the top of the window are a menu bar and toolbar.

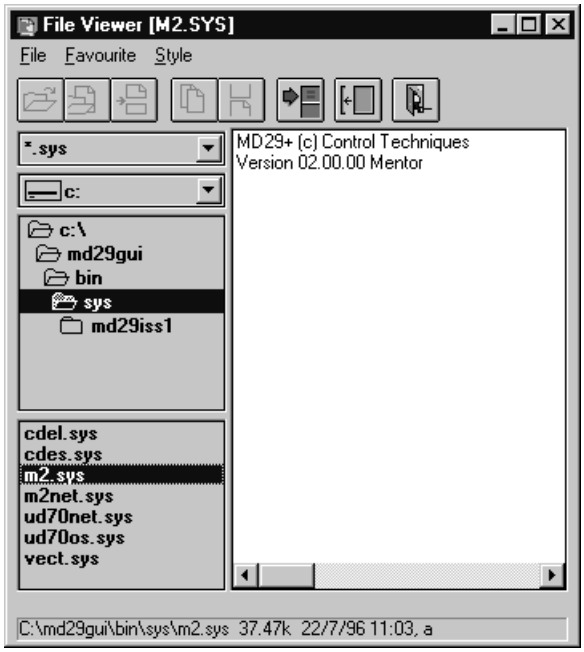
4 If the serial port of the host PC is not COM1, open the **Projects** menu and select **Configure**. In the drop-down menu that appears, select **Comport**. This opens a further drop-down menu which allows selection of the required communications port.

5 Click on (**Open Task Manager**). After a few moments, the **DPL Task Manager** dialog box appears.



If the dialog box does not appear, and all the buttons in the lower toolbar of the window appear shaded (inactive), communications could not be established with the MD29. Check the connecting cable is correct, and the correct COM port is used.

6 Click on File Viewer. The **File Viewer** dialog box appears.



In the panels on the left side of the dialog box the path and names of the .SYS files can be selected.

Note

The system files are located in directory MD29GUI\BIN\SYS.

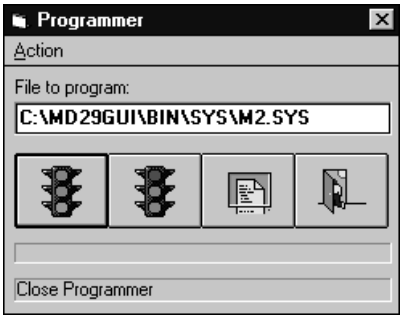
7 Select the correct path for the required system files. Double-click on the .SYS file specified in the following table:

Drive	File
Mentor II	M2.SYS
Mentor II (MD29AN)	M2NET.SYS
Vector	VECT.SYS
CDE (bookcase)	CDES.SYS
CDE and HPCDE	CDEL.SYS

Note

The system files for MD29 issue 1 are located in directory MD29GUI\BIN\SYS\MD29ISS1.

The **Programmer** dialog box appears.



8 Click on  (green light showing). The system file is now loaded into the MD29.

3 Getting Started

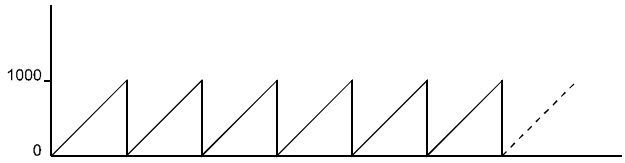
3.1 Introduction

This chapter explains the key elements of DPL programming, and the methods used to create, compile and run an example program using the DPL Toolkit.

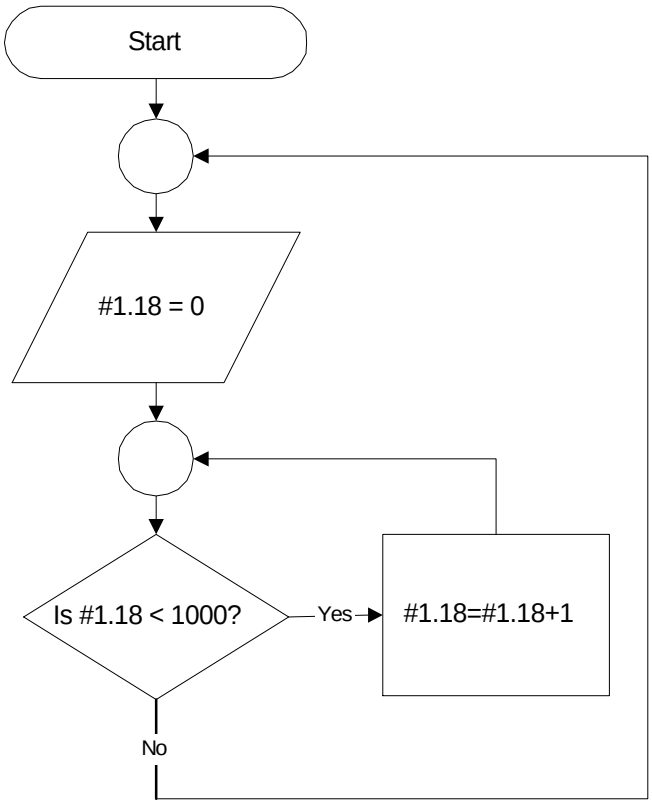
An example of a short DPL program is given below, followed by explanations of the program instructions.

3.2 Example DPL program

The DPL program described here is called **Sawtooth**, because it applies a repetitive cycle consisting of a linear increase in speed demand followed by an instantaneous reduction to zero, as shown in the following diagram.



*Repetitve cycle produced by the Sawtooth **program***



Flow diagram of program Sawtooth

Program instructions

```
$TITLE Sawtooth
$VERSION 1.1.1
$DRIVE Mentor
$AUTHOR MyName
$COMPANY MyCo
//Note: This is a comment.
BACKGROUND{
  Top:
    #1.18=0
    DO WHILE #1.18<1000
      #1.18=#1.18+1
    LOOP
  GOTO Top:
}
```

Explanation of the example program

$TITLE Sawtooth

The first line of a program must be **$TITLE** program name. The name can have a maximum of 64 characters.

$VERSION 1.1.1

The second line is **$VERSION** number. The number can have a maximum of eight characters. The recommended format is **$VERSION 1.0.0.**

Updates are easily shown by increasing the last number, eg. **1.0.1**. Major modifications are shown by **2.0.0**, **3.00**, etc.

$DRIVE Mentor

The third line is **$DRIVE** drive name. This tells the compiler which Drive it is installed in. (Since the DPL Toolkit can be used with different types of Drive, the name of the Drive must be stated.) The list of suffixes are as follows:

- MENTOR
- CDE750 (bookcase CDE)
- CDE7500 (large CDE and HPCDE)
- VECTOR

$AUTHOR MyName
$COMPANY MyCo

The fourth and fifth lines are used to define the author of the program and the company name.

Note

Unless these lines are included, the program will not be compiled.

//Note: This is a comment.

The program ignores comment lines which can be placed anywhere in a program. Comments are always preceded by either a double forward slash [//] or a semi colon[;].

Comments are useful for inserting descriptions, or for giving explanations for the benefit of the user or programmer.

BACKGROUND{

BACKGROUND is a type of Task. (All executable code must be contained within a Task.) There are many different types of Task, which, in effect, define the priority of the code and allow blocks of code to be run on different time-bases.

The BACKGROUND task is a free-running task which can be compared to the way a PLC runs a program, for example. Full details of the Tasks are given in Real-time programming in Chapter 4 DPL Programming.

Top:

Top: is a label which marks an absolute position in the program. A label must always be followed by a colon [**:**].

A label defines the destination of a **GOTO** statement. It can be given any name (eg. **mylabel**).

#1.18=0

A hash (**#**) expression accesses Drive parameters. In this case the parameter is 1.18 (menu 1, parameter 18). This is a preset speed reference parameter in the Mentor Drive, and it is set at zero.

DO WHILE #1.18<1000

DO WHILE is a loop statement. In this example, it gives the program an instruction to repeat the following block of code while the value of parameter 1.18 is less than 1000.

#1.18=#1.18+1

This line adds the value **1** to parameter 1.18. Every time this command is executed, **1** is added to the parameter value.

LOOP

LOOP is the end expression for the Instruction **DO WHILE**. **LOOP** tells the program to go back to the line **DO WHILE** and check that the **DO WHILE** instruction remains true. When the value of #**1.18 = 1000**, **DO WHILE #1.18 < 1000** becomes false. The instructions between **DO WHILE** and **LOOP** stop being repeated and the program goes to the next line after the **LOOP** command.

GOTO Top:

GOTO is a flow-control instruction. In this case, it tells the program to go to the label **Top:**. This causes the program to run continuously.

Note

The label name must be specified using a colon[:].

} Closing brace

Instructions within braces belong to the defined Task. Closing-braces work in conjunction with opening-braces. In this example, the opening- and closing-braces work in conjunction with the Task **BACKGROUND.**

3.3 Creating a DPL file using the DPL Toolkit

This section shows how to write, compile and download the example DPL program for the Mentor II Drive.

Opening the DPL Toolkit

In Windows 3.xx Program Manager, or Windows 95 Start menu, click on:

The **DPL Toolkit** window appears. At the top of the window are a menu bar and toolbar.

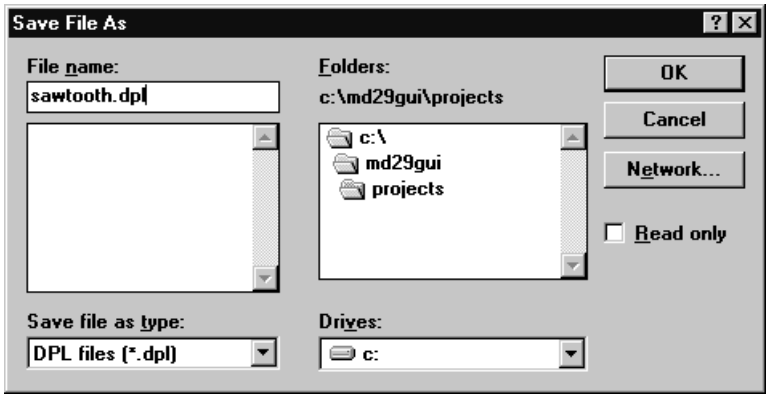
Creating a file

1 Click on  or open the **File** menu and select **New**.

2 Enter the following program exactly as it appears, using the tab key to indent lines.

```
$TITLE Sawtooth
$VERSION 1.1.1
$DRIVE mentor
$AUTHOR MyName
$COMPANY MyCo
//Note: This is a comment.
BACKGROUND{
   Top:
      #1.18=0
      DO WHILE #1.18<1000
         #1.18=#1.18+1
      LOOP
   GOTO Top:
}
```

3 Open the **File** menu and select **Save As...**. The **Save File As** dialog box appears.



4 In the **Folders:** list, select the **Projects** directory. In the **File name:** text box, type **SAWTOOTH.DPL**.

5 Click on OK. The file is now saved.

The program is ready for compiling into machine code.

Important Note

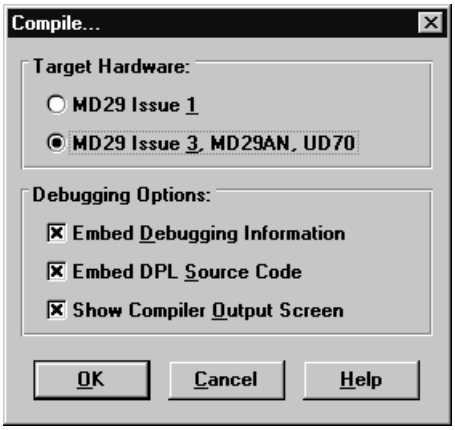
DPL programs must be saved as .DPL files. If this is not done, the program cannot be compiled into machine code. Only the saved version of the program is compiled.

Compiling the program

The DPL Toolkit contains a compiler which converts DPL programs from text format to binary machine code which the MD29 can understand. The compiler converts the .DPL file into a binary file with a .BIN extension.

Use the following procedure.

1 Click on  at the right of the Toolbar. The **Compile...** dialog box appears.



2 If the DPL source file is required to be downloaded to the MD29, ensure the **Embed DPL Source Code** check box is checked. This facility allows the DPL program to be read back to the PC at a later date (if the computer copy becomes lost, for example).

If the DPL source file is not to be downloaded, ensure the check box is unchecked. When the **Compile...** dialog box next appears, the check box retains the last setting.

(The other options in this dialog box are described in Compiling and running programs in Chapter 5 DPL Toolkit.)

3 Click on OK

4 The **Compilation** box appears for a few seconds. It is not necessary to observe the contents of the **Compilation** box.

5 The program is now compiled, ready for downloading to the MD29.

If instead a **Build errors** window appears with errors displayed, correct the program for typing mistakes and repeat the compilation. (Error messages are described in Chapter 9, Diagnostics.)

Connecting to the MD29

It is now necessary to establish communications from the host PC to the MD29 in order to download the compiled file.

Use the following procedure:

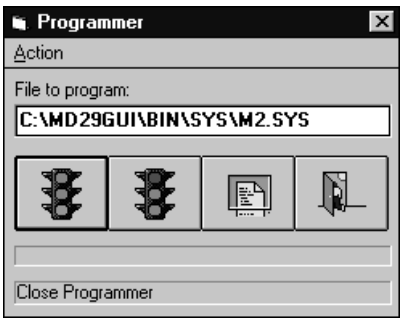
Click on  . The Task Manager opens with the Task Manager toolbar appearing below the standard toolbar.

Note

If all the buttons on the lower toolbar appear shaded, it is an indication that communications could not be established with the MD29. Check that AC power is applied to the Drive, and that the serial communications cable is correctly inserted.

Downloading the program

1 In the Task Manager toolbar, click on  . The **Programmer** dialog box appears.



2 In the **Programmer** dialog box, click on  (green light showing). The files **SAWTOOTH.BIN** and **SAWTOOTH.DPL** are now downloaded to the MD29. Down-loading takes a few seconds to complete.

Note

The MD29 can hold only one compiled program (ie. .BIN file) in memory at one time. A program that is downloaded to the MD29 will over-write an existing program.

Running the program

Warning

The Sawtooth program rapidly alters the speed reference parameter of the Drive. For safety, ensure the Drive is disabled before running the program.

In the Task Manager toolbar, click on 

The **Speed reference** parameter **#1.18** in the Drive will change value. Note that the ramping-up behavior cannot be observed since the program alters the parameter value at a faster rate than the display is updated.

4 DPL Programming

This chapter explains the following parts of a DPL program:

- Program headers
- Comments
- Variables
- Parameters
- Tasks
- User-defined sub-routines
- Instructions

The explanation is followed by a section on optimizing DPL programs.

4.1 Program headers

A DPL program must begin with five program headers in the correct order, as follows:

- Program title
- Program version
- Drive name
- Author name
- Company name

Each program header must be contained on a single instruction line in the program.

Program title

Syntax **$TITLE Program title**

The **$TITLE Program title** is for use by the programmer.

eg. **$TITLE Sawtooth generator**

Maximum length: 64 characters

Program version

Syntax **$VERSION *Version Number***

The **$VERSION Version Number** is for use by the programmer. It is recommended that the format of the version number should be as follows:

$VERSION 1.0.1

Minor updates can be shown by increasing the last digit, eg. **1.0.2**. Major modifications can be shown by increasing the first digit, eg. **2.0.0**.

Maximum length: 8 characters

Drive name

Syntax **$DRIVE *Drive name***

The type of Drive and its size must be specified in **$DRIVE Drive name** since the DPL Toolkit can be used with different types of Drive.

This program header ensures that the program is correctly compiled for the option module and Drive. The following is a complete list of **Drive names.**

$DRIVE MENTOR
$DRIVE CDE750
$DRIVE CDE7500
$DRIVE VECTOR

Note

If a different Drive is specified, the program may not be compiled, or run-time error 53 will occur when the program is downloaded to the MD29.

Author name

Syntax **$AUTHOR *Author name***

The **$AUTHOR *Author name*** is for use by the programmer.

Maximum length: 64 characters

Company name

Syntax **$COMPANY *Company name***

The **$COMPANY *Company name*** is for use by the programmer.

Maximum length: 64 characters

Example program headers

$TITLE Sawtooth
$VERSION 1.0.1
$DRIVE MENTOR
$AUTHOR A.H.
$COMPANY Control Techniques

4.2 Comments

Comments are purely for information and explanation purposes. They act in the same way as **REM** commands by not acting on the program.

Comments begin with a double forward slash [//] or a semi-colon [;]. They can be placed on their own line, or at the end of instruction lines. A Comment ends at the end of the line.

Example
```
//This line contains a comment, which ends with the line.
//If the comment flows onto the next line, double forward
//slashes must be used to start the next line.
```

4.3 Variables

Basic variables

There are two basic types of variable, as follows:

- Integer variable (INT)
- Floating-point variable (FLOAT)

Integer variables

Integer variables are denoted by placing a **%** symbol after the name of the variable, and are internally represented by a two's complement 32-bit number. This gives a decimal range of ±2147483647.

Floating-point variables

Floating-point variables have no symbol. These variables are IEEE double-precision (64-bit) numbers which give a range of approximately $\pm 1.7976 \times 10^{\pm 308}$.

Accessing the variables

All variables are global within a program (ie. they can be accessed and altered by any task). (There are no local variables.)

Bit-addressing of variables

All integer variables and arrays (see below) may be bit-addressed. This means that each individual binary bit in the variable may be separately read or written to. Bit-addressing is achieved by appending **.n** to the end of the variable name, where **n** is the bit number to be accessed.

Example
```
flags%.3 = 1              ;set bit 3 to 1
IF flags%.5 = 1 THEN ...  ;check bit 5
```

Naming conventions

The first character of a variable must be a letter. Subsequent characters may include letters, numbers and the underscore (_) character. These may be in any order.

Variable names are case sensitive (eg. the variable name **speed%** is not the same as **SPEED%**).

Preferred use of variables

It is recommended that integer variables are used where possible. Operations on integer variables perform much faster than for floating-point variables.

Arrays

Arrays are collections of variables of the same type (integer or floating point) under the same name. Note that only single-dimension arrays are allowed.

Each element (individual component) of an array is, in effect, a separate variable. An element is accessed by a program by specifying the array name, then placing the element number in square brackets **[ ]** after the array name

The two basic forms of arrays are as follows:

Dynamic arrays

Dynamic arrays can be set up and changed by DPL programs. A dynamic array must contain, integer variables or floating-point variables, but not both types of variable.

A dynamic array must first be specified using the DIM instruction (usually in the INITIAL task), and the number of elements specified in square brackets after the variable name. Dynamic arrays are placed in the 8kB of volatile memory in the MD29 which limits the maximum size of the array.

Example

```
DIM myarray%[20]      ;Integer array having 20 elements
DIM array2[30]        ;Floating-point array having 30 elements
```

The elements in an array are numbered as follows:

0 to [Number of elements] – 1

From the example of an integer array given above, the first element of **myarray%[]** is as follows:

myarray%[0]

The last element is as follows:

myarray%[19]

Constant arrays

Constant arrays contain fixed pre-defined values that cannot be changed by the DPL program when the program is being run. The values of the constant array are defined in the DPL program by using a special section called **CONST**. (This section is typed in exactly the same way as a task.) Only integer values can be defined in a constant array.

The advantage of using a constant array is that the array is placed in the 96kB of memory space in the MD29 which allows the size of the array to be limited only by the amount of available program space in the MD29, and not by the size of the 8kB RAM. The program space is used to store the compiled DPL program, constant array data, and (optionally) the DPL file itself.

Example

```
CONST c_array% {
100, 1500, 500, 0, −400, −1000
−400, −100, 0
}
```

This defines an array called **c_array%[]**, which containes nine elements. Note that the value of each element can be separated by a comma or a new line.

4.4 Parameters

There are two types of parameter, as follows:

- Drive parameters
- Virtual parameters

(See Chapter 10 Parameters.)

Parameters are denoted by a **#** (hash) symbol and are accessed using an **x,y** format, where **x** represents the menu and **y** represents the parameter in the menu.

For example, parameter **p7.05** is accessed by entering **#07.05**, and **p18.01** is accessed by entering **#18.01**. Leading zeroes in the parameter can be omitted, eg. **#7.5** is the same as **#07.05**.

Parameters can also be accessed indirectly using an integer variable to denote the parameter number. See Parameter pointers later in this chapter for details.

4.5 Operators

Operators perform mathematical or logical operations on values. The following operators are supported in DPL programming.

Note

Certain operators work only with integer values or variables.

Operators for floating-point and integer variables

+	Plus
–	Minus
/	Divide
*	Multiply

Operators for integer variables only

& Logic AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Example 5 & 14 = 4

| Logic OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Example 5 | 14 = 15

Logic XOR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Example 5 & 14 = 11

!Value *Bit invert*

This Operator inverts the least-significant bit, and converts all other bits to zero.

Example 100100 (binary) is converted to 000001 (binary)

!(value, bit-field-size)
Bit-field invert

This Operator inverts the specified number of least significant bits, and converts all other bits to zero. The **bit-field-size** specifies the number of least-significant bits that are to be converted.

Example Result% = !(value%, 3)

100**100** (binary) is converted to 000**011** (binary).

% *Remainder*

This Operator gives the remainder when an integer is divided by another integer.

Example 5 % 2 = 1

8 % 3 = 2

4.6 Tasks and real-time programming

Real-time programming runs with reference to a clock to enable the user to specify the actual times instructions are executed, not just the order in which they are executed. When real-time programming, a task Structure (or philosophy) has to be maintained.

MD29 programs contain sections called tasks, where a task enables a priority to be given to a sub-routine. Six levels of priority are defined by these tasks in the following order:

- INITIAL task
- BACKGROUND task
- CLOCK task
- ENCODER task
- EVENT task
- ERROR task

Each task is specified by its name in the program. The contents of each task must be placed in braces **{ }**.

Example

```
CLOCK{
instructions
}
```

INITIAL task

The INITIAL task is used typically to initialize program variables and Drive parameters in the DPL program. The task runs only when the MD29 is reset or at the moment AC power is applied.

The INITIAL task has total priority over all other tasks when running; the other tasks are prevented from running. This is significant when the CLOCK, EVENT or ENCODER tasks are to manipulate data which have initial values.

Example

```
INITIAL{
// This is the only place to reliably initialize 'timer'
timer% = 0
}

CLOCK{
//This task is set at 5ms
//The value of timer must be initialized before CLOCK is run
timer% = timer% + 1
IF timer% > 200 THEN
  //200, 5ms intervals = 1 second
  PRINT "1 Second expired"
  timer% = 0
ENDIF
}
```

BACKGROUND task

The BACKGROUND task is used for functions and commands that do not require time-related or encoder-related monitoring. This task would be used for the following:

- Data logging
- Checking digital inputs
- Setting output status

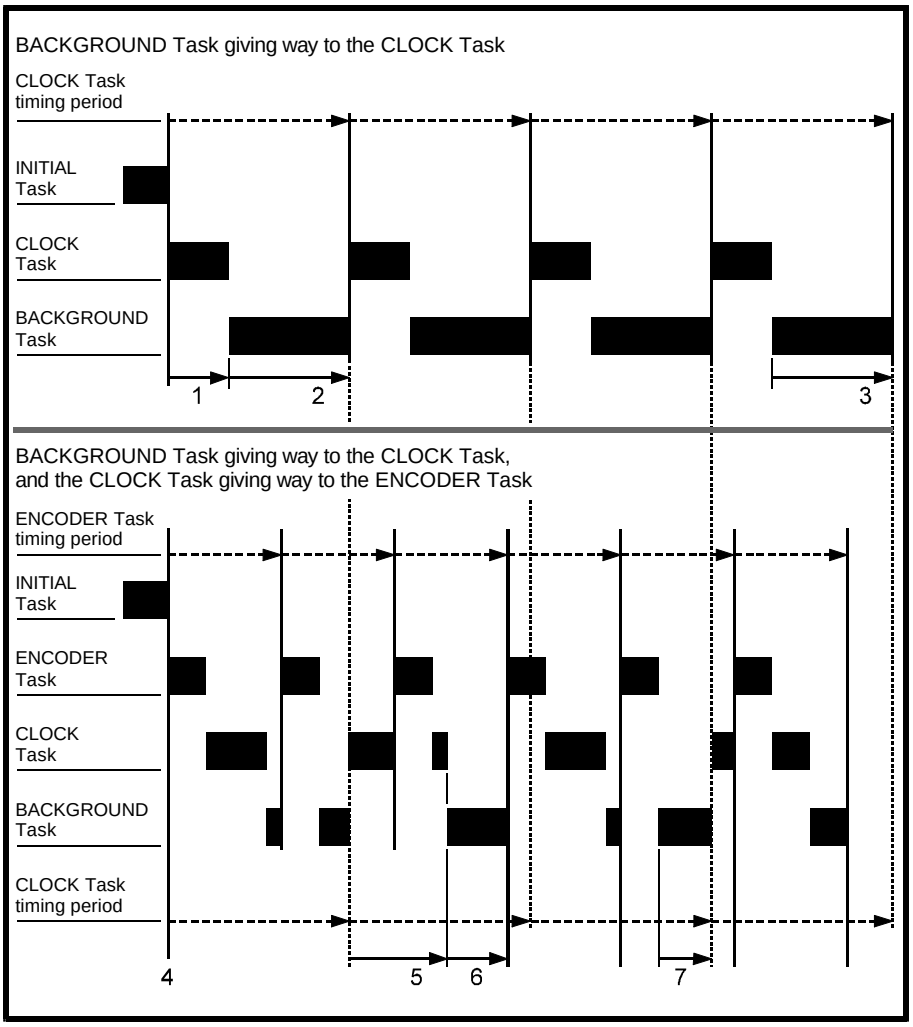
The BACKGROUND task runs after the INITIAL task is completed. It is recommended that the majority of the program is run in the BACKGROUND Task.

Note

The BACKGROUND task does not automatically loop.

Example

```
BACKGROUND{
  RAMP:
    #1.18 = 0
    DO WHILE #1.18<1000
      #1.18 = #1.18+1
    LOOP
  GOTO RAMP:
}
```



Examples of the BACKGROUND task giving way to the CLOCK and ENCODER tasks

Key to the diagram

BACKGROUND task giving way to the CLOCK task

1 The BACKGROUND task waits while the CLOCK task runs, and is then interrupted at the next CLOCK task.

2 The BACKGROUND task continues running until next interrupted by the CLOCK task.

3 The BACKGROUND task ends.

BACKGROUND task giving way to the ENCODER and CLOCK tasks

4 ENCODER and CLOCK timing periods begin.

5 The CLOCK task runs until it is interrupted by the next ENCODER task. The CLOCK task is completed when the ENCODER task has finished.

6 The CLOCK task ends, leaving time for the BACKGROUND task to run until interrupted by the next ENCODER task.

7 When the ENCODER task has finished the next CLOCK period has not arrived. The BACKGROUND task runs until interrupted by the next CLOCK task.

User-defined sub-routines

User-defined sub-routines are written by the user and are used in conjunction with the CALL instruction (see CALL in Chapter 7 Reference).

User-defined sub-routines can be given any name and can be inserted anywhere in a program. (Note that the task name is **case-sensitive**.)

The following sub-routine has the same function as the **Sawtooth** program given in Chapter 3 Getting Started. The name given to the sub-routine is **RAMP:**.

```
BACKGROUND{
  Loop:
    CALL RAMP:
  GOTO Loop:
}
RAMP: {
  #1.18=0
  DO WHILE #1.18<1000
    #1.18=#1.18+1
  LOOP
}
```

Important Note

Be careful not to allow a user sub-routine to be started by two different real-time tasks (a situation termed re-entry).

For example, a sub-routine is able to be started by a BACKGROUND task as well as a CLOCK task. If the BACKGROUND task starts the sub-routine, and the CLOCK task interrupts the BACKGROUND task while the sub-routine is being executed, the values of the variable being processed could be altered. This can occur because the CLOCK task will also run the sub-routine, but will apply its own values.

CLOCK task

The CLOCK task is used for time-related monitoring of the Drive, and commands to the Drive (eg. controlled acceleration or deceleration ramp).

The task has the second lowest priority. Only the BACKGROUND task gives way to the CLOCK task.

The task is executed on a constant timebase; the actual timebase used depends on the value of the set-up parameter on the Drive (see also MD29 set-up parameters in Chapter 10 Parameters), which can range from 1ms to 200ms.

Example This example produces a sine-wave.

```
CLOCK{
  #1.18 = SIN (rad)*1000
  rad =rad+0.01
  IF rad>6.283185  THEN   ; 6.283185 = 2 * pi
    rad = 0
  ENDIF
}
```

ENCODER task

The ENCODER task is primarily used to monitor the activity of an encoder.

The task is synchronized to a control loop in the Drive, so the execution frequency of the task is determined by the Drive. A set-up parameter can be used to multiply the time by two.

Drive	Switching frequency	Timebase parameter set at 0	Timebase parameter set at 1
	kHz	ms	ms
Mentor II	Not applicable	5.12	2.56
CDE	3, 6 or 12	5.52	11.04
CDE	4.5 or 9	7.36	14.72
Vector	Any	2.008	4.016

Example

```
ENCODER{
  master_inc% = #90.2
  slave_inc% = #90.4
  EPOS = EPOS + master_inc% – slave_inc%
}
```

EVENT task

The EVENT task runs when a specific event occurs. The source of the event is determined by the Timer/Counter Unit.

The EVENT task has the highest priority when the program is running. All other tasks give way to the EVENT task.

Refer to Timer/Counter Unit in Chapter 8 Features for further information.

ERROR task

The ERROR task is executed only when a run-time error has occurred in the DPL program. If the DPL Toolkit is connected to the MD29 at the time of the error, the error number will be displayed on the screen.

Run-time errors can be caused by a variety of occurrences. For example:

Attempting to write to a read-only parameter

A real-time task over-running

Errors are usually due to programming errors, but can sometimes occur due to external influences. For example, an error signifying a serial communications loss could occur if incoming data from an I/O Box is lost due to the cable being broken. Normally, the MD29 halts all tasks, and optionally trips the Drive.

If this is undesirable, the ERROR task can be used. The sequence when a runtime error occurs is then:

1. All tasks are stopped.
2. The Drive is tripped (if the trip is enabled). See the Trip enable parameters in MD29 setup parameters in Chapter 10 Parameters.
3. The number of the error is placed in parameter #88.01 of the MD29
4. The ERROR task is executed. The instructions in the ERROR task can determine the cause of the run-time error and take necessary action, such as stopping the drive system in a controlled manner.

For further information, see Advanced error-handling in Chapter 9 Diagnostics.

NOTES task

This is a pseudo task that is ignored by the compiler. The writer of the program uses the NOTES task to help the user of the Drive understand the program.

Example

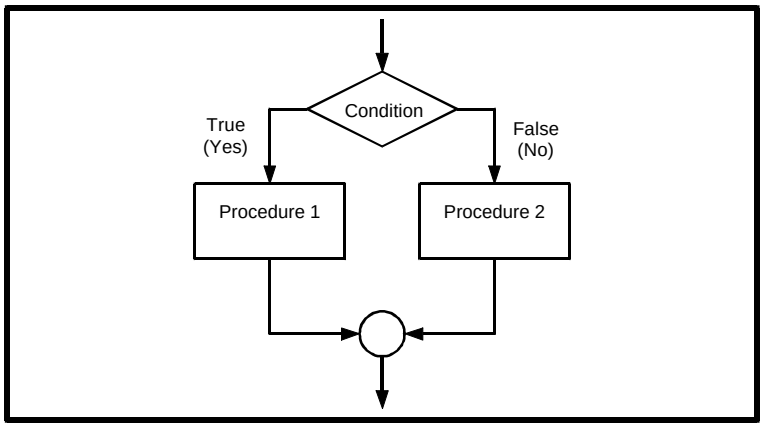
```
NOTES{
You can put your documentation here.
}
```

4.7 Instructions and functions

This section describes the different types of instructions which are used in DPL programming.

Conditional instructions

A conditional instruction performs an operation according to a set condition (eg. **IF**).

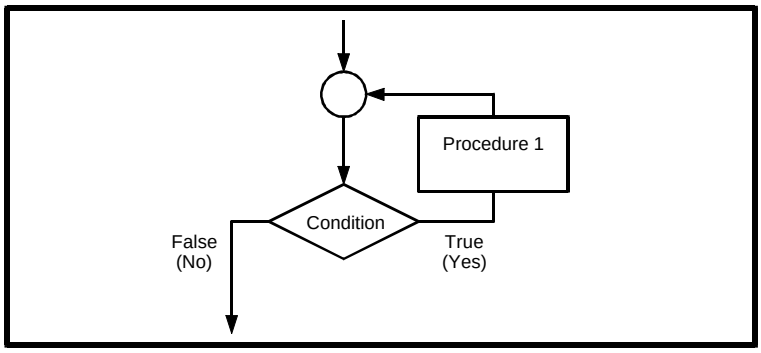


Example of an IF, THEN flow diagram

Loop instructions

A loop instruction repeats a block of instructions until a specified condition occurs.

Example DO WHILE
LOOP



Example of a DO WHILE, LOOP flow diagram

Flow-control instructions

A Flow-control instruction causes the program to jump to a specified instruction or to be terminated (eg. **GOTO**).

Maths functions

A Maths function applies a mathematical operative to an expression to return a value (eg. **SIN**).

Signal-processing functions

A Signal-processing function returns a value from a number of samples over a fixed time-period. Signal-processing functions can be used only in the CLOCK or ENCODER tasks (eg. **FILTER**).

Base-conversion functions

A Base-conversion function acts upon a value to convert Binary Coded Decimal to Binary and vice versa. Base-conversion functions are useful for data received from an IO Box. Refer to Chapter 6 Serial Communications.

Data-conversion functions

A Data-conversion function converts a floating-point variable to an integer variable and vice versa.

ANSI instructions

An ANSI instructions allows a DPL program to communicate via the RS485 port with other Drives and MD29 cards using the ANSI protocol. Refer to Chapter 6 Serial Communications.

4.8 Optimizing programs

In order for programs to run effectively, the following are recommended.

Integer variables

Use integer variables where possible, rather than floating-point variables. The processing of a floating-point variable is 20 times slower than for an integer variable. (See *INT* instruction in Chapter 7 Reference.)

Fixed-point arithmetic

To represent decimal places, use fixed-point arithmetic. For example, if a resolution of **.001** is required, let **1** be represented by **1000**. This allows accuracy to be maintained throughout mathematical operations.

The output from an expression must then be corrected by a relevant dividing factor.

Example

```
a% = 1500          //      "a% = 1.5"
b% = 2500          //      "b% = 2.5"
c% = a% * b%       //      c% = 3750000
// Divide by 1000 to adjust c%
c% = c%/1000       //      "c% = 3.750"
// To convert to the real value, we must divide by 1000 again
#1.21 = c%/1000    //      "c% = 3.75"
```

Temporary integer variables

Minimize the number of times parameters are accessed. Instead of accessing a parameter repeatedly, use temporary integer variables if a parameter value is needed more than once. The access time for a parameter is 50 times greater than that for a variable.

Example

```
IF #1.18 > 100 THEN
  range% = 1
ELSEIF #1.18 > 200 THEN
  range% = 2
ENDIF
This becomes:
temp% = #1.18
IF temp% > 100 THEN
  range% = 1
ELSEIF temp% > 200 THEN
  range% = 2
ENDIF
```

Integer division

When using integer division, accuracy may be lost in the result, as shown in the following expression:

If #1.18 is equal to 5

Then we have the following:

a = 4.5 * (#1.18 /4)

= 4.5 * (5 /4)

= 4.5 * 1

= 4.5

The DPL compiler uses an integer divide, converts the result to a floating-point value and uses a floating-point multiply.

To preserve accuracy, one of the arguments can be converted to a floating-point variable, as follows:

a = 4.5 * (#1.18 / FLOAT(4))

= 4.5 * (5 / FLOAT(4))

= 4.5 * 1.25

= 5.625

See *FLOAT* instruction in Chapter 7 Reference.

PRINT instruction

Do not over-use the PRINT instruction. (See PRINT instruction in Chapter 7 Reference). It is preferable to use the **Watch** window in the DPL Toolkit to monitor variables (see Chapter 5 DPL Toolkit).

Use the PRINT instruction only in the BACKGROUND task. If the PRINT instruction is included in the CLOCK or ENCODER tasks, the PRINT instruction may have insufficient time to be executed. Text waiting for printing may not then be printed.

BACKGROUND task

Place as much of the program as possible in the BACKGROUND task rather than in the CLOCK, ENCODER or other real-time tasks. Since the real-time tasks are on a fixed timebase, the processing must be completed in this time. The BACKGROUND task does not have this restriction.

The #INT instruction converts a parameter that requires floating-point variables to accept integer variables. This greatly increases processing speed.

Example

```
#2.00 = 2.1
// set #2.00 at 2.1 on CDE750 Drive
//is the same as
#INT2.00 = 21
// Reading is also possible:
value% = #INT2.00
```

4.9 Parameter pointers

A parameter pointer is an integer variable that represents a Drive parameter.

Example

```
A% = 118 // set A% to point to #1.18
#A% = 10//write 10 to the pr A% points to (#1.18)
```

Note

If the parameter contains a decimal-point, the decimal point is ignored. (For example, parameter 2.00 in the CDE Drive is in units of 0.1. A value of 2.3 must be written as 23.)

4.10 Defining aliases (constants)

Sometimes it is useful to assign a meaningful name to a parameter or a value. For example:

Parameter #1.18 could be referred to as SPEED_REFERENCE

Instructions can be written in the form:
SPEED_REFERENCE = MAX_SPEED

Aliases are created using the $DEFINE directive. The syntax is:
$DEFINE name value.

The $DEFINE directive can be used to assign the required value to a name that is used subsequently in the program; the name becomes an alias for the value. All occurrences of the name are replaced by the value when the program is compiled.

Note

Comments are not allowed at the end of a $DEFINE line.

There are two parts to an alias, as follows:

Name parameter	The name parameter specifies the name to be defined. This can be any combination of letters, digits and underscore characters. Spaces are not permissible.
Value parameter	The value parameter can be used to specify any constant value or parameter number.

Example This example (for the Mentor II Drive) demonstrates use of the $DEFINE directive to assign names to parameter numbers (#3.02 and #1.18) and to a value (500).

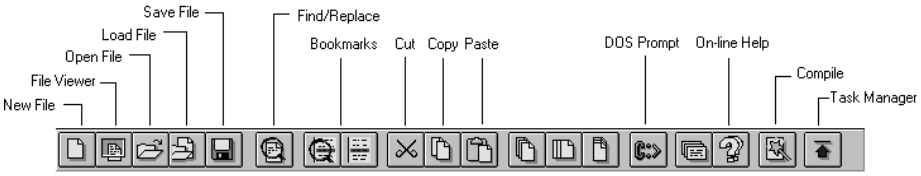
```
$define MAX_SPEED 500
$define SPEED #3.02
$define SPEED_DEMAND #1.18

BACKGROUND{
top:
IF SPEED < MAX_SPEED THEN
   SPEED_DEMAND = SPEED_DEMAND + 1
ENDIF
GOTO top:
}
```

5 DPL Toolkit

This chapter describes operation of the DPL Toolkit, compiling of programs, and the debugging facilities.

5.1 Overview of the DPL Toolkit

The DPL Toolkit enables the user of the MD29 to amend, write and download programs to the MD29. The Toolkit consists of a set of compilation tools and a comprehensive editor and debugger.



Main toolbar of the DPL Toolkit

The compilation tools enable the user to perform the following:

- Develop and edit real-time programs for the MD29.
- Cut and copy program text to the Windows clipboard.
- Paste program text from the Windows clipboard.
- Load an existing program from the MD29.
- Compile the program into machine code.

The debug facility has the following tools:

- Read the values of the Drive parameters on the screen and edit the values while the DPL program is running.
- Read the values of the Variable parameters on the screen and edit the values while the DPL program is running.
- Single-step mode for program checking.
- Breakpoints.

Note

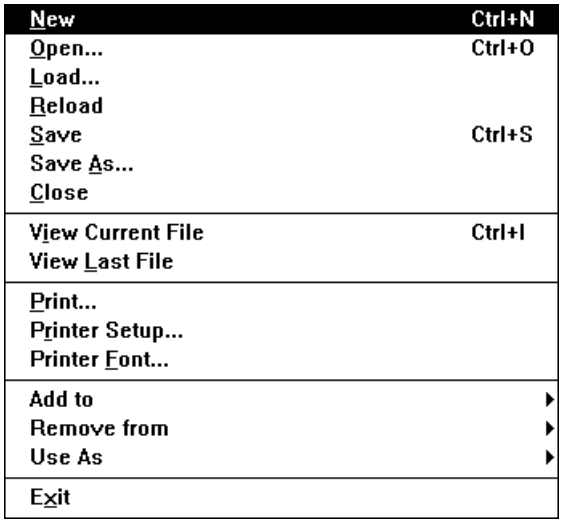
Only one program can be stored in the MD29 at any one time.

5.2 File management

File management in the DPL Toolkit follows similar principles to that in other Windows applications. In addition to the standard procedures, there are procedures specific to the DPL Toolkit. These are given below.

File menu

The **File** menu is as follows:



Creating a new file

In the **File** menu, select **New**, or click on  . A blank page is created for you to start work on.

Opening an existing file

There are two methods of opening a file,as follows:

Load into a new Window

In the **File** menu, select **Open...**, or click on 

Load into the existing window replacing the current contents

In the **File** menu, select **Load...**, or click on 

Re-loading the last-saved file

In the **File** menu, select **Reload**.

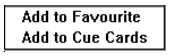
Saving a file

In the **File** menu, select **Save As...**, or click on 

DPL files must be saved with a .DPL filename extension before they can be compiled.

Add a filename to a menu

The **Add to** option allows files to be added to a menu for easy access. When the **Add to** option is selected, the following list appears.



This list refers to two of the main menu items **Favourite** and **Cue Cards** in the Toolkit. When one of these menus is selected, files which are added to the menu are listed in a drop-down menu. The file can then be immediately selected.

The options are as follows:

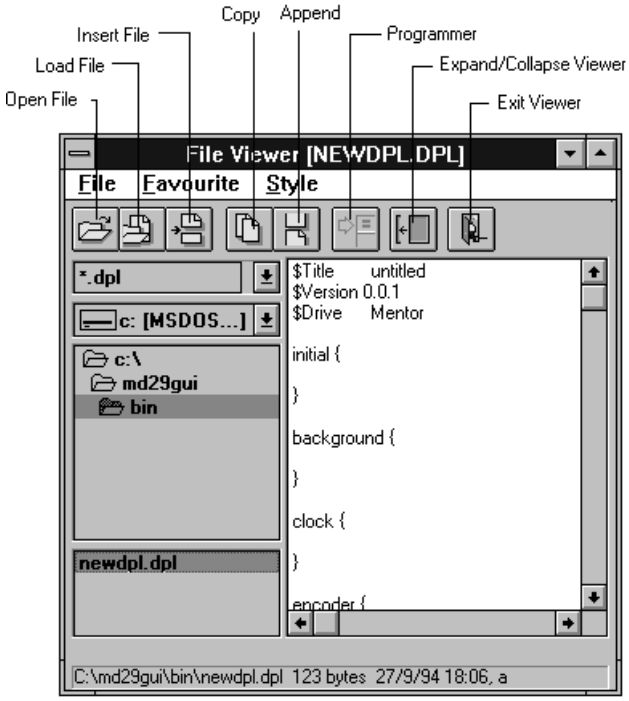
Add to favourite

This adds the open file to the **Favourite** menu.

Add to Cue Cards

This adds the open file to the **Cue cards** under the **Help** menu.

File Viewer



File Viewer allows the user to perform the following:

- View a file without opening it
- Copy text from an unopened file and paste it in the open file
- Pre-select individual lines for copying in one operation

Opening File Viewer

Do either of the following:

Click on  (**File Viewer**).

In the **File** menu, select **View Current File** or **View Last File**.

When **View Current File** is selected, File Viewer appears with the currently open file loaded.

When **View last file** is selected, File Viewer appears with the last-saved file loaded. This file is not necessarily the file that is displayed on the screen.

Copying and pasting text

1. Select the file and highlight the text that you want to copy.
2. Click on  (**Copy**).
3. Place the cursor in the required position for the selected text.
4. Open the **Edit** menu and select **Paste** in the drop-down menu.

Copying and pasting sub-routines

Use either of the following procedures to select a sub-routine and paste it into different programs.

Using File Viewer

1. Save the sub-routine as a file. See Chapter 4 DPL Programming.
2. Place the cursor in the open program where the text is to be inserted.
3. Open File Viewer. In the box at the bottom left corner of File Viewer is a list of saved files.
4. Select the name of the file that contains the required sub-routine.
5. Click on  (**Viewer Insert**).

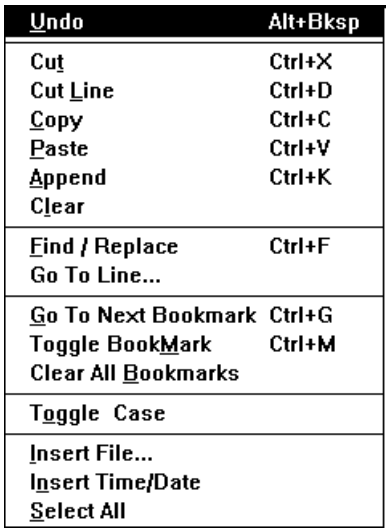
Using the main toolbar

1. Save the sub-routine as a file. See Chapter 4 DPL Programming.
2. Place the cursor in the open program where the text is to be inserted.
3. Open the **Edit menu** and select **Insert File** in the drop-down menu.

5.3 Editing a program

Edit menu

The **Edit** menu is as follows:



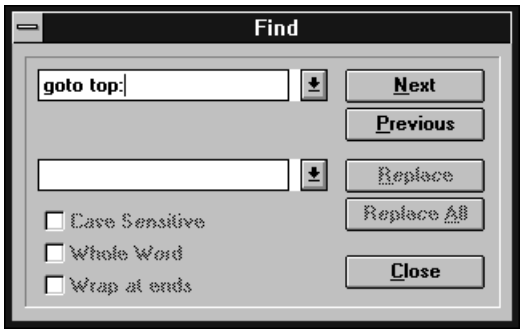
The basic editing tools are similar to other Windows applications. The tools allow you to cut, copy, paste, clear and undo.

Cutting a line

Select **Cut Line** to delete highlighted instruction lines.

Finding and replacing text

Select **Find/Replace** to find and replace characters and words. The **Find** dialog box appears.



Use this option in the same way as the Find/Replace option in Windows word processors.

Appending instruction lines

Use the following procedure to copy lines in a specific order from File Viewer to the program being written:

1. Select in turn the lines shown in File Viewer that are to be copied to the new program.
2. After each selection, click on  (**Append**).
3. Place the cursor in the required position for the lines to appear in the new program.
4. Open the **Edit** menu and select **Paste** in the drop-down menu.

BookMarks

BookMarks are useful for negotiating long programs. A BookMark is inserted into a program where the writer needs to refer to a location on a regular basis. A number of BookMarks may be used in a single program.

Setting a BookMark

1. Position the cursor in the program where the BookMark is to be placed.
2. Click on  (**BookMark**).

Returning to a BookMark

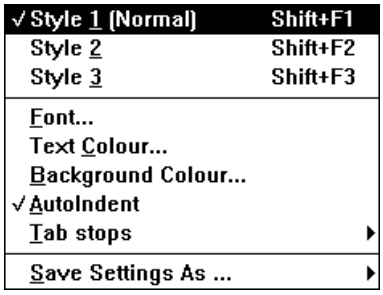
1. Click on  (**Next BookMark**).
2. The cursor goes to the **BookMark** that has been placed. If the **Next BookMark** button is clicked on again, the cursor highlights the next BookMark. BookMarks are highlighted in the order they were placed.

Clearing BookMarks

Open the **Edit** menu, and select **Clear All BookMarks**.

5.4 Applying styles

The **Style** menu is as follows:



Styles

Styles let you alter the way the DPL Toolkit screen appears. There are 48 background and text colours giving over 2000 combinations of colours that can be used.

Under **Font**, there is an extensive list of text fonts including TrueType fonts.

Auto-indent

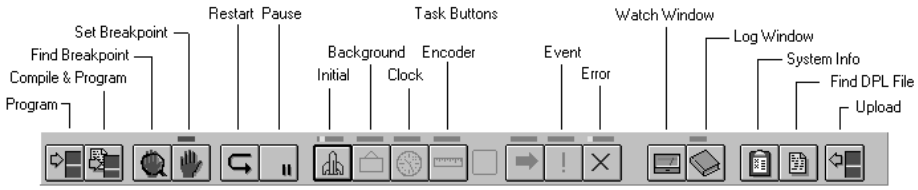
AutoIndent allows you to set a tab for DO WHILE...LOOP and IF...ENDIF commands. When the **Enter** key is pressed at the end of the line the indent is automatically retained for the next line. To delete the indent, press the **Backspace** key.

Using this method of indents, you can easily pick out discrepancies in the programming by ensuring that an IF statement ends with an ENDIF statement and that a DO statement ends with a LOOP statement (see Chapter 7 Reference).

5.5 Compiling and running a program

Task Manager toolbar

The DPL Task Manager contains powerful compilation and debugging tools. These tools enable the programmer to check the program in great detail. Some of the debugging tools are automatic and check for programming errors. Others allow the programmer to check the program line by line to verify the logic of the program.

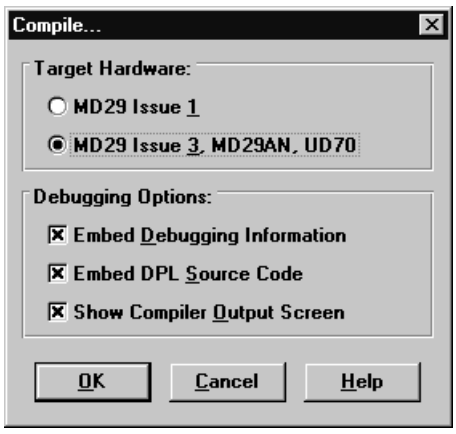


Compiling a program

1 Save the written program as a .DPL file.

2 Click on  (**Compile**), or to compile and download the program automatically, click on:

The **Compile...** dialog box appears.



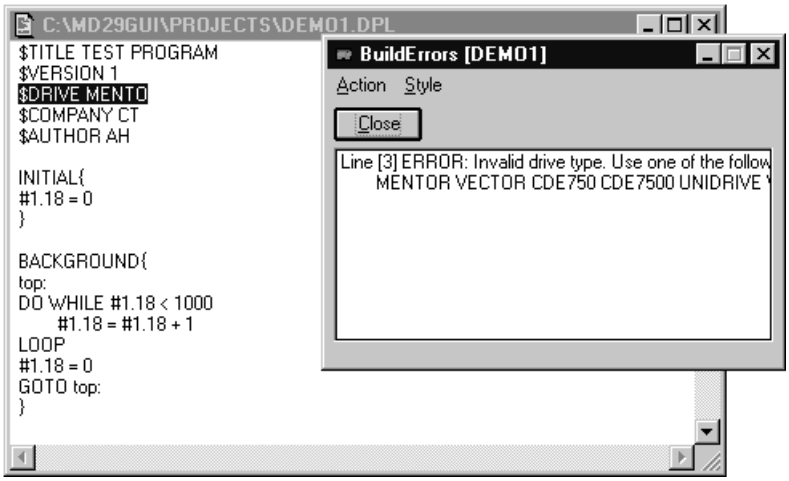
If the MD29 in use is hardware issue 1, check the **MD29 Issue 1** radio button.

Normally the options can be left as they are shown. In this case, to continue the compilation process click on:

Debugging options

The debugging options enable various debugging aids for the DPL program to be de-selected. See Program monitoring and debugging below for a description of these options.

If errors are encountered in compiling the file, the **Build Errors** window appears showing the program errors. As each error is highlighted in **Build Errors**, the corresponding line in the program is also highlighted.



Using Build Errors

1. Click on the first line that is shown as an error.
2. Correct the error (the type of error is indicated in the **Build Errors** window).
3. Click on  .
4. Click on 

in the **Save Changes** dialog box that appears.
5. Repeat the compile process to update the compiled program.

Note

The debugging tools in the compiler highlights problems in the program that are attributed to programming errors. The debugging tools will not highlight problems due to logic. If a program has been compiled and downloaded to the MD29 but it appears not to be running, it is likely to be a logic problem.

Errors and warnings

An error indicates that the compiler could not interpret a line or command in the DPL program. This could occur if a command is mis-spelt, incorrectly used, etc.

A warning indicates that the compiler understood the commands but the code may not function in the way you expected. The most common warning is Possible loss of accuracy in assignment, and can occur when integer variables or parameters are assigned a floating-point value.

A full list of errors and warnings can be found in Chapter 9 Diagnostics.

5.6 Downloading a program

A program can be downloaded only when it is free from **errors.** Once the program file is downloaded to the MD29, the DPL program is ready to be run.

To download the file, click on  (**Down-load**) in the Task Manager toolbar.

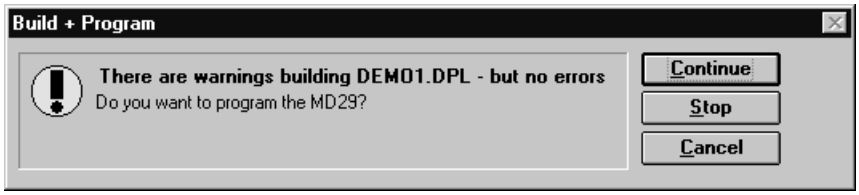
If Auto-Run is selected, the program will automatically run after downloading and the MD29 is initialized (reading the set-up parameters).

Compiling and downloading a program

To compile and download a program in a single step, click on 

Alternatively, select **Quick Start** from the **Project** menu.

If any errors are encountered, the download process will not occur. If the program contains warnings only, the **Build + Program** dialog box appears.



To ignore the warnings and continue to download, click on the **Continue** button, otherwise click on the **Stop** button.

5.7 Running a program

Click on  (**Restart**). This runs the DPL program from the beginning (ie. at the INITIAL task). The Task Manager buttons for the tasks present in the program are made active.

Stopping the program

Click on  (**Pause**).

Resuming the program

Click again on  (**Pause**).

5.8 Program monitoring and debugging facilities

Single-stepping

Single-stepping executes only one line of a DPL program at a time. By using single-stepping, the operation of a program can be monitored instruction-by-instruction. During single-stepping, all other tasks may run at full speed.

Single-stepping through a task

1 Click on the appropriate Task Manager button, as follows:

 INITIAL task

 BACKGROUND task

 CLOCK task

 ENCODER task

This halts the selected task. (The EVENT task cannot be single-stepped.)

2 Repeatedly clicking on the appropriate Task Manager button will advance the execution point to the next line.

Clear single-stepping

Use either of the following methods to clear the single-stepping function:

- Hold down the **Ctrl** key and click on the appropriate Task Manager button.
- Open the **Run** menu and select the appropriate **Run Task** option.

Breakpoints

A breakpoint is a line in a task at which point the task will stop the program running and enter into single-stepping mode.

Breakpoints are useful for checking when a program reaches a particular piece of code, or for checking the state of DPL variables at a particular point.

Setting a breakpoint

1. Place the cursor on the line where the breakpoint is to be set.
2. Click on  (**Set breakpoint**). The breakpoint is now active. When the program execution reaches the set line, the task is halted and single-stepping mode is started.

Note

Only one breakpoint may be set in a task at any one time. Breakpoints and single-stepping are not possible in user sub-routines.

Finding breakpoints in separate tasks

1. Click on  (**Next breakpoint**). The cursor goes to the next breakpoint in the DPL program.

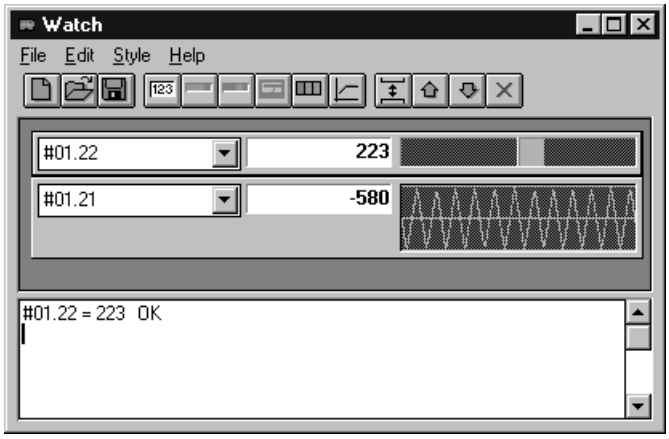
Removing breakpoints

1. Place the cursor on the line where the Breakpoint is to be removed, or use the **Next breakpoint** button to find the line that has the breakpoint.
2. Click on the **Set breakpoint** button.

Using the Watch window

The **Watch** window enables the programmer to check the logic of the program while it is running in the MD29 by reading and writing parameters and variables.

To display the **Watch** window, click on  in the Task Manager toolbar.



Example display in the Watch window

There are two section to the **Watch** window. The top section shows the values of the parameters and variables being **continuously** updated while the program is running. Values can be shown numerically and graphically. The bottom section allows snap-shot reading and writing of parameters and program variables.

Continuously monitoring a variable or a parameter

There are five ways of viewing an item in the top section of the **Watch** window, as follows:

Display method	Button
Value only	
Value with uni-polar bar graph	
Value with bi-polar bar graph	
Value with bi-polar line graph	
Value with individual bits displayed	

Use the following procedure for monitoring a parameter or variable:

1 Click on the toolbar button for the required display method.

2 Select the required parameter or variable to be watched, using one of the following methods:

- In the parameter text box on the left, type in the name of the parameter or variable to be watched.
- Click on  and select the required parameter or variable from the list that appears.
- Double-click on the parameter or variable name in the main DPL Toolkit editor window.

Changing the full-scale value

The full-scale value for the graphical display defaults to 1000. To alter this, double click in the value display box. In the **Max Value** dialog box that appears, type the required value for full-scale. Click on the **OK** button.

Changing to and from bipolar values

To change the graphical display for a watched item from one type to another (eg. uni-polar bar to bi-polar), move the mouse cursor over the graphical display region and press the right mouse-button. In the pop-up menu appears, select the required option.

Using the lower section of the Watch window

The lower section of the **Watch** window allows the user to take an instantaneous reading of a variable or parameter, and also write to any variable or parameter.

Reading a parameter or variable

1 Type the parameter or variable name (eg. **#1.21**).

2 Press ENTER. The value is shown at the right of the parameter/variable name.

Setting a value for a parameter or variable

1 Type the parameter or variable name, followed by an equals sign and the value to be written (eg. **#1.21 = 1000**)

2 Press ENTER. If the value was written successfully, **OK** is displayed at the right.

A parameter or variable entered in the lower section, can be automatically added to the top section by pressing the SHIFT and ENTER keys after typing the name.

Note

If any changes are made to the program, it has to be re-saved, compiled and downloaded to the MD29.

Saving the Watch window settings

The settings of the **Watch** window can be saved onto disk for later use. To achieve this, select **Save** in the **File** menu of the **Watch** window.

Further details of the **Watch** window are covered in the on-line help facility. Press F1 at any time to display the on-line help.

Uploading the DPL source file from the MD29 to the host PC

The .DPL source file can be uploaded to the host PC when the original program file is not available or if the program that is running is not exactly known.

Use the following procedure to retrieve the .DPL file:

1. Click on  (**Upload**) in the DPL Task Manager toolbar. If a version of the DPL program already exists in the host PC, a dialog box appears asking if the file is to overwritten.
2. To find out which program is resident in the MD29, and to attempt to load it, click on:

    (Source code)

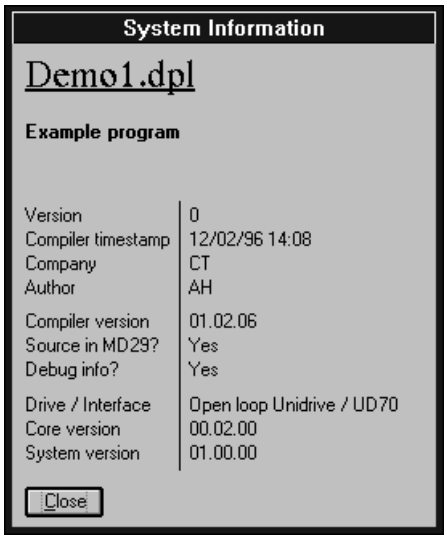
Important Note

It is safer to upload the *.DPL program from the MD29 than to retrieve the source code from the host PC, unless you are sure that the program that is in the host is the required program.

System information

System information gives the user information about the program in the MD29, such as the program name, date of compiling, version number of the MD29 operating system, etc.

To view this information, click on  (**System information**). The **System Information** message box appears, eg:



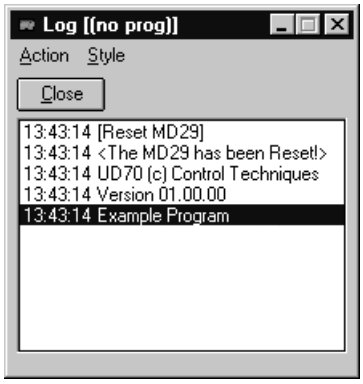
The Log window

The Log window can be used to show the following:

- System messages (eg. the starting and stopping of a program)
- Watch window values (useful for data logging)
- The output of the DPL PRINT instruction (see PRINT instruction in Chapter 7, Reference).

These functions are enabled and disabled using the **Action** menu.

To open the Log window, click on  (**Log window**). The Log window appears:



6 Serial Communications

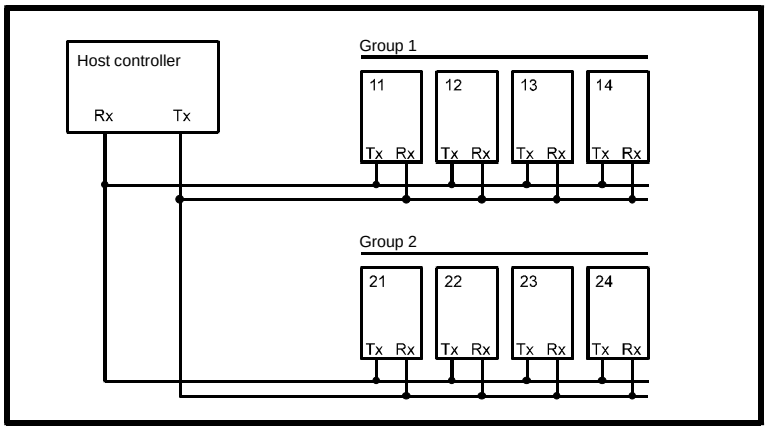
6.1 Introduction

A serial communications link enables one or more MD29 cards to be used in systems controlled by a host unit such as a PLC or computer. The communications link uses the RS485 standard.

The MD29 may also act as the host in a system, controlling Drives, UD70 modules, MD29 cards or other devices fitted with a suitable interface.

The host controller can operate up to thirty-two EIA RS485 devices with the use of line repeaters. Each transmitter and receiver of Control Techniques devices loads the line by two unit-loads. Therefore in two-wire mode, each Control Techniques device loads the line by four unit-loads. This means that no more than a total of seven such devices can be connected in a single group, allowing up to four unit-loads for the line repeater. Up to 15 devices can be connected if four-wire mode is used.

When line repeaters are used, up to 81 Control Techniques devices can be operated. In this case the devices are organized in up to nine groups of nine. A particular group or groups can be given commands without affecting other devices or groups of devices.



RS485 mulitdrop link having two groups of four units

The communications port of the MD29 is the male D-type connector on the right side of the board. The MD29 may be used in either 4-wire or 2-wire mode. The RS485 port is fully opto-isolated. RS422 is also supported.

Caution

An RS232 connection may be made to the RS485 port, but is not recommended due to its inferior specification (noise rejection, limited maximum cable length, etc). RS232 is not the same as two-wire RS485.

6.2 Hardware connections

The following table details the hardware connections for the RS485 communications port.

Pin	RS485 4-wire	RS485 2-wire
1	0V	0V
2	$\overline{Tx}$	$\overline{Tx/Rx}$
3	$\overline{Rx}$	$\overline{Tx/Rx}$
4	DI0 *	DI0 *
5	DI1 *	DI1 *
6	Tx	Tx/Rx
7	Rx	Tx/Rx
8	DO *	DO *
9	0VD *	0VD *

* Terminals 4, 5, 8 and 9 form the digital I/O connections of the MD29. Since they form no part of the serial communications connections they must not be connected to any serial communications lines or the serial communications 0V (pin 1).

Ground connection

It is recommended that the shield of the data communications cable should be connected by a low-inductance path to a 'clean' ground.

Routing the serial communications cable

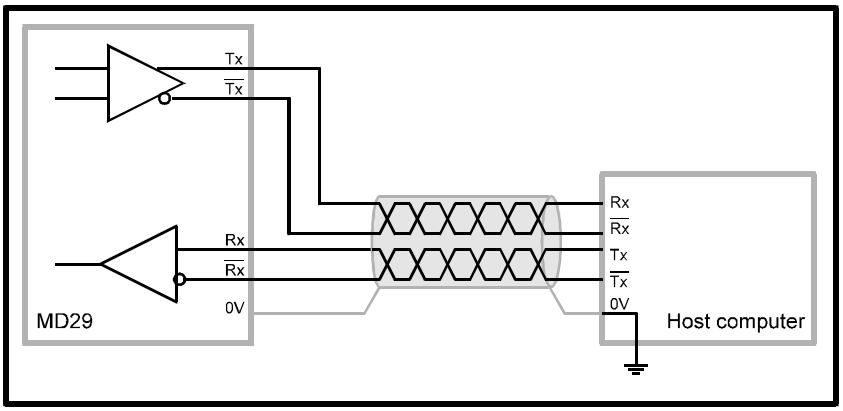
A data communications cable should not run parallel to any power cables, especially ones that connect Drives to motors. If parallel runs are unavoidable, ensure a minimum spacing of 300mm (1 foot) between the communications cable and the power cable.

Cables crossing one another at right-angles are unlikely to give trouble.

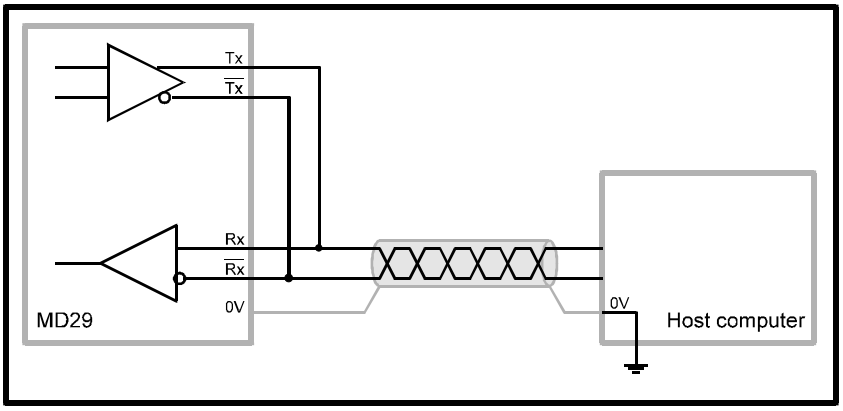
The maximum cable length for a RS485 link is 1200 metres (4,000 feet).

Terminating the cable

When a multi-drop RS485 system is used, connect a 120Ω resistor between the two receive lines of the last unit in the chain (ie. the unit farthest away from the host). Care must be taken to ensure that other units in the system do not have the resistor already fitted. Excessive signal loss will occur if termination resistors are connected to units other than the last one.



Connections for 4-wire mode



Connections for 2-wire mode

6.3 ANSI communications

Using the standard ANSI slave protocol

The standard built-in protocol which defines the message structure used to read and write parameters on the MD29 is ANSI x3.28-2.5-A4. This section explains this protocol.

The user may also create his own protocol by writing it in a DPL program, using low-level port commands such as GETCHAR and PUTCHAR (refer to Chapter 7 Reference).

ANSI slave protocol is enabled when the RS485-mode set-up parameter is set at 1 (4-wire) (which is the default setting), or 5 (2-wire). See Serial communications modes later in this chapter for details of other communication modes.

Fundamentals of data transmission

Data is transmitted at a fixed speed or baud rate in the form of a character. A character may typically comprise seven or eight bits.

In order for a receiver to recognize valid data, a frame is placed around each character. This frame contains a start bit, a stop bit, and an optional parity bit. Without this frame, the receiver will be unable to synchronize itself with the transmitted data.

A frame is shown below:

	Low ASCII character byte								
	1st hex character				**2nd hex character**				
Start bit	**Seven data bits**							**Parity bit**	**Stop bit**
0	LSB						MSB		1

This is known as a 10-bit frame, since there are 10 bits transmitted in total. The format is often described as follows:

1 start bit, 7 data bits, even/odd/no parity, 1 stop bit.

lsb refers to the least significant bit (ie. bit 0)

msb refers to the most significant bit (bit 6)

The Parity bit is used by the receiver to check the integrity of the data it has received

The character set used is called the low ASCII set. The set comprises 128 characters decimally numbered from 0 to 127. The first 32 characters in the ASCII set (hex. 00 to 1F) are used to represent special codes. These are the control codes, each of which has a particular meaning (eg. start of text is called STX and is ASCII code 02.)

On a computer or terminal, the **STX** character may be transmitted by pressing **Ctrl+B**. When the MD29 is in standard ANSI mode, it recognizes that a command follows the STX character.

The control code **EOT** (end of transmission) instructs all MD29 cards and Drives on the RS485 bus to be ready to receive a new message — it is often sent at the start of a message so that all the devices are set at **Ready to receive message**.

Control characters

Commands and requests are sent in message packets. Each message is started with a special control character, and may contain control characters. A list of all the control characters that can be used when sending a message, and receiving is as follows:

Character	Meaning	ASCII code (decimal)	Keyed as...
EOT	Reset Instructs the MD29 to prepare for a new message. Also indicates parameter does not exist.	04	Ctrl D
ENQ	Enquiry Used when interrogating.	05	Ctrl E
STX	Start of text Used to start a command.	02	Ctrl B
ETX	End of text Used at the end of a command.	03	Ctrl C
ACK	Acknowledge (message accepted)	04	Ctrl F
NAK	Negative acknowledge (message not understood)	21	Ctrl U
BS	Backspace (go to previous parameter)	08	Ctrl H

Baud rate

The user can select a baud rate between 2400 and 38400. The default is 4800. See setup parameters.

Addressing

Each unit on a ANSI communications bus must be given a unique identity or address so that only the target MD29 unit will respond. The address comprises two parts:

- The Group Address which is the first digit.
- The Unit Address which is the second digit.

Both the group address and unit address have a range of 1 to 9. A group or unit address of 0 is not allowed (addresses 01, 10, 20, etc. are invalid). The reason for this is that MD29 cards and some Drives can be grouped together (up to 9 units per group), and a message can be sent over the ANSI communications bus to all units of the group. To address a particular group, the unit address of zero (0) is used. For example, to address all units of group 6 the full address will be 60.

An additional feature is that a message can be sent to all units of all groups simultaneously using the address 00. This address can be used to send a Start command to a group of Drives which are mechanically coupled together to drive a conveyer line. All the Drives will then start simultaneously.

Note

It is important to realize that when using group addressing, the MD29 cards will not acknowledge the command. (If several cards try to reply at the same time, they would cause meaningless data to appear on the serial communications bus.)

For data integrity, the format of the transmitted address requires that each digit of the two-digit address is repeated: the address of MD29 number 23 is sent as four characters, eg:

2 2 3 3

The serial address follows immediately after the first control character of the message (EOT).

Parameter identification

All parameters are identified by four digits representing the menu and the parameter number, but without the decimal point.

Example To send a message to menu 4, parameter 8, write 0408 (the leading zero must be included)

To send to menu 16, parameter 10, write **1610**.

Vector Drive For the Vector Drive, the parameter set is accessed in a similar x,y manner, as shown in the following table.

	Parameter
Pr0 – 99	00.xx
b0 – 99	01.xx
F0 – 49	02.xx

Data field

Data to be sent or requested occupies the characters immediately after the parameter number. The minimum length of the data field within a message structure is two characters.

The data is normally expressed as a decimal numeric value. Hexadecimal format may also be used by specifying the first character of the data field as a **X**.

The first character of the data field (D1) can be only one of the following:

Space (32 dec.)

+

–

X (for hex.) — hex. is typically used to access I/O Box data.

Block checksum (BCC)

In order to ensure that the messages from or to the MD29 do not become corrupted during transmission, all write messages and data responses are terminated by the block checksum character (BCC). See Calculating the block checksum (BCC) later in this section.

Reading parameters

To read a parameter, the following message is sent:

Control	Address				Parameter				Control
EOT	**GA**	**GA**	**UA**	**UA**	**M1**	**M2**	**P1**	**P2**	**ENQ**

Where:

GA = Group Address
UA = Unit Address
M1 M2 = Menu number
P1 P2 = Parameter number

Note

No BCC character is sent in this message.

The MD29 will reply with the following structure if the message is understood:

Control	Parameter				Data			Control	BCC
STX	**M1**	**M2**	**P1**	**P2**	**D1**	**...**	**Dn**	**ETX**	**BCC**

Where:

M1 M2 = Menu number
P1 P2 = Parameter number
D1...Dn = Data
First character:
+ or Space for positive values
– for negative values
X for hex. values
BCC = Block checksum

If a requested parameter does not exist, the MD29 will reply with an **EOT** character (ASCII 04).

Example To read the speed set-point of a Mentor II Drive that is unit 2 of group 1, send:

Control	Address				Parameter				Control
STX	**1**	**1**	**2**	**2**	**0**	**1**	**2**	**1**	**ENQ**

The unit replies as follows:

Control	Parameter				Data					Control	BCC
STX	**0**	**1**	**2**	**1**	**-**	**4**	**7**	**.**	**6**	**ETX**	**7**

Note

When the MD29 replies to a command, the length of the data field returned is variable, depending upon the type of Drive, and the parameter being read.

Re-reading data

Once a read message has been received and understood (ie. valid data was returned), to request the parameter again, request the next parameter, or the previous parameter, a single control code character may be sent. These control codes are:

Control Code	Function	Keyed as...
NAK	Return the value of the same parameter	Ctrl U
ACK	Read the next parameter	Ctrl F
BS	Read the previous parameter	Ctrl H

This facility can be used to save time when monitoring a parameter over a period of time.

Writing to parameters

To write data to a parameter (Drive or virtual), the message structure is comprised as follows:

Control	Address				Control	Parameter				Data			Control	BCC
EOT	**GA**	**GA**	**UA**	**UA**	**STX**	**M1**	**M2**	**P1**	**P2**	**D1**	**...**	**Dn**	**ETX**	

Where:

GA = Group address
GU = Unit address
M1 M2 = Menu number
P1 P2 = Parameter number
D1...DN = Data
First character:
+ or Space for positive values
– for negative values
X for hex. values
BCC = Block checksum

The data field can be of a variable length with the maximum length being dependent on the parameter being edited.

The MD29 will respond with a single control character, as follows:

Control Code	Meaning
ACK	Acknowledge — Message has been understood and implemented.
NAK	Message invalid, data is too long or out of range, parameter is invalid, parameter is read-only, or the BCC is incorrect.

Example Set parameter **Pre-set frequency 1** at +76.4 for a CDE Drive (unit 6, group 2) send:

Control	Address				Control	Parameter				Data						Control	BCC
EOT	**2**	**2**	**6**	**6**	**STX**	**0**	**1**	**2**	**5**	**+**	**0**	**7**	**6**	**.**	**4**	**ETX**	**%**

Re-writing data

Once a write message which includes the address field has been sent and accepted with either a <ACK> or <NAK> response, subsequent write messages to that particular MD29 can use a re-write message structure in which the address does not need to be re-transmitted. The re-write structure is as follows:

STX	M1	M2	P1	P2	D1	...	Dn	ETX	BCC

When a different MD29 is addressed, or an invalid character is received, the re-write facility no longer functions. The first MD29 can be addressed again only by using the full write message with the address.

Calculating the block checksum (BCC)

The block checksum is calculated by applying an exclusive OR function to all of the characters of a message after the STX control character.

XOR truth table

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

For example, the serial command to set **Pre-set speed 1** at −34.5Hz on a CDE Drive:

The message will be:

Control	Address				Control	Parameter				Data					Control	BCC
EOT	**1**	**1**	**2**	**2**	**STX**	**0**	**1**	**2**	**1**	**–**	**3**	**4**	**.**	***5***	**ETX**	**0**
Not included in the calculation						Included in the calculation										Result

The first character of the BCC calculation is **0** (00110000 in binary), the value of which is taken as a starting or result value. The next character is **1** (00110001 in binary), which now has the exclusive OR (XOR) operator act upon it. With the previous result value, a new result occurs of 00000001 in binary.

The complete calculation is show in the table below:

Character	Binary Value	XOR result
0	0011 0000	–
1	0011 0001	0000 0001
2	0011 0010	0011 0011
5	0011 0101	0000 0110
–	0010 1101	0010 1011
3	0011 0011	0001 1000
4	0011 0100	0010 1100
.	0010 1110	0000 0010
5	0011 0101	0011 0111
ETX	0000 0011	0011 0100

The final value is the BCC, provided that its equivalent decimal value exceeds 31 (ASCII characters from 00 to 31 are used as control codes).

When the final XOR result produces a decimal value less than 32, 32 is added. In this example, 0011 0100 is 52 decimal which is above 31, so this is the final BCC value. 52 decimal is the character 4. The complete message will be:

EOT	1	1	2	2	STX	0	1	2	5	-	3	4	.	5	ETX	4

Example QuickBasic program to calculate BCC

```
mess$ = CHR$(4)+"1122"+CHR$(2)+"0125"+"-34.5"+CHR$(3)
bcc% = 0
FOR n% = 7 to LEN(mess$)'start at the character after 'chr$(2).
bcc% = bcc% XOR ASC(MID$(mess$, n%, 1))
NEXT
IF bcc% < 32 THEN bcc% = bcc% + 32
mess$ = mess$ + CHR$(bcc%)
PRINT mess$
```

In DPL, the ANSIREAD and ANSIWRITE functions automatically calculate the BCC.

6.4 Serial communications modes

The MD29 has ten communication modes as follows:

- 4-wire and 2-wire standard ANSI slave
- High-speed parameter transfer
- Programmable master/slave modes
- User modes
- The MD29AN has an additional mode for use with the I/O Box.

The available options are as follows (see Set-up parameters in Chapter 10 Parameters for details of the actual parameters):

Mode 1 **Standard 4-wire**

RS485 using ANSI slave protocol (default).

Mode 2 **Master mode**

In this mode, data is taken from the defined parameter, scaled to +/-16000 then continuously transmitted out to a slave drive. The slave unit can be a UD70 applications module, a Mentor II Drive, a CDE Drive or another MD29..

Mode 3 **Slave mode**

In this mode, data is received from the serial port and is scaled by the value of the scaling set-up parameter before being placed in the destination parameter.

Mode 4 **Cascade mode**

With this mode, a defined parameter is transmitted to a remote drive unit (no scaling) and received data is placed into the destination parameter. The data is not scaled.

The transmitting unit and/or receiving unit can be either another MD29, UD70, or a Mentor II Drive.

This mode is typically used in wire-drawing type applications – the first Drive is given a speed reference from an external source (eg. a potentiometer). Mode 4 communications are used to pass that reference down to the next Drive on the line which uses an Applications Option to receive the data and apply draw. That Drive then passes the new reference down to the next Drive, etc.

Mode 5 **2-wire RS485**

Using standard ANSI slave protocol. Note that RS232 is not 2-wire RS485.

Mode 6 **User mode**

This mode turns off all internal protocols and allows the user to use the RS485 port directly from a DPL program. Typically, this mode will be used in conjunction with the DPL ANSI master commands (ANSIREAD, ANSIWRITE, etc.). User-defined protocols can also be implemented in DPL with the low-level PUTCHAR and GETCHAR commands. The communications data-frame is organized as follows:

1 start bit, 7 data bits, EVEN parity, 1 stop bit, 10 bits total

Mode 7 **User mode**

This mode turns off all internal protocols and allows the user to write and read directly from the ANSI port using the PUTCHAR and GETCHAR functions. The communications data-frame is organized as follows:

1 start bit, 8 data bits, EVEN parity, 1 stop bit, 11 bits total

Mode 8 **User mode**

This mode turns off all internal protocols and allows the user to write and read directly from the ANSI RS485 port using the PUTCHAR and GETCHAR functions. The communications data-frame is organized as:

1 start bit, 8 data bits, NO parity, 1 stop bit, 10 bits total

Mode 9 **User mode**

The communications data-frame is organized as:

1 start bit, 9 data bits, NO parity, 1 stop bit, 11 bits total

Mode 10 **I/O Box mode (MD29AN only)**

This mode allows a single I/O Box to be connected directly to the MD29AN using the EIA RS485 port. Only one I/O Box may be connected to the MD29AN when mode 10 is used. If multiple I/O Boxes are required, they must be set up in standard ANSI mode, mode 6 must be selected, and the DPL RS485 commands used to transfer data.

Mode 11 **User mode**

This mode bypasses the internal software buffers and interfaces directly to the hardware. This reduces the delay in passing data through the UD70 RS485 port. The baud rate is programmable, in the same way as the other modes. The communications data-frame is organized as follows:

1 start bit, 9 data bits, NO parity, 1 stop bit, 11 bits total

Mode 12 Reserved

Mode 13 Modbus – RTU (slave mode only)

Mode 14 Modbus – ASCII (slave mode only)

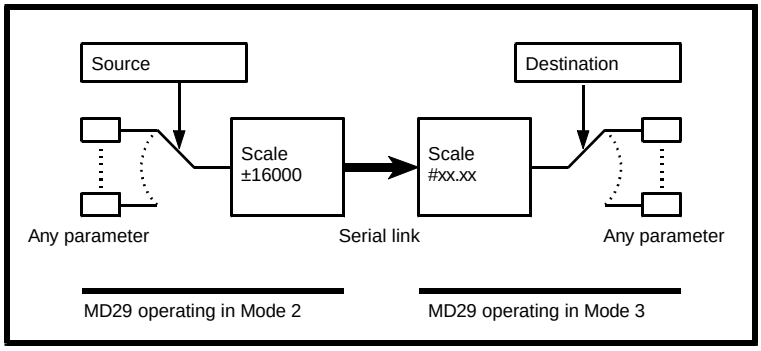
The Modbus RTU and ASCII slave modes support the functions Read Multiple Registers, Preset Single Registers and Preset Multiple Registers. This mode limits the number of consecutive registers to 20, and the node address range is limited from 11 to 99.

Contact your local Drive Centre for information about the Modbus Protocol.

Note

There is minimal data integrity checking with modes 2, 3, and 4.

Using Modes 2 and 3



Functions of Modes 2 and 3

Using Modes 6, 7, 8 and 9

In Modes 6, 7 or 8, it is possible to use the MD29 as an ANSI master device controlling other MD29 cards, UD70 modules, Drives or other ANSI compliant devices. This facility may alleviate the need for a custom computer or PLC to control a system or process. When an I/O Box is used, there may be no need for a PLC in a system.

Modes 6 to 9 also allow you to create custom serial protocols which you may use for communicating with non-ANSI compliant devices such as intelligent display modules, etc.

The ANSI RS485 port is buffered as follows:

Receive: 60 bytes

Transmit: 25 bytes

Using Mode 11

Using other modes, it can take up to 8ms for data to start being transmitted or received. In mode 1, the data bypasses the internal software FIFO buffer, which reduces the delays in the system.

Using Modes 13 and 14

The Modbus protocol is only supported as a slave. Modes 13 and 14 do not allow the MD29 to act as a Modbus Master. The baud rate is selected using parameter #17.07.

6.5 ANSI instructions

The following RS485 ANSI instructions are available:

ANSIREAD	Issue a Read command to a remote unit
ANSIREPLY	Read a reply from a remote unit
ANSIWRITE	Issue a Write command to a remote unit
GETCHAR	Receive a single character directly
PUTCHAR	Transmit a single character directly

Refer to Chapter 7 Reference for information about these commands.

6.6 Example ANSI instructions

The following program fragment gives an example of reading a speed reference from a Mentor II Drive, and writing it to a CD Drive.

Both Drives are connected to the RS485 bus. The Mentor Drive is at address **11**; the CD Drive is at address **12**.

For this program to operate, the RS485 mode should be set at *6*.

```
BACKGROUND{
top:
// send read command to read parameter #1.17
result% = ANSIREAD(11, "0117")
IF result% = 0 THEN
        PRINT "Message could not be sent"
        GOTO top:
ENDIF
CALL get_reply:   // get reply from read command
IF reply% <= –65536 THEN   // valid reply?
        PRINT "Error in reading data from Mentor"
        GOTO top:
ENDIF
// scale speed value so that 1000 on the Mentor will equate
// to 50.0Hz on the CD drive
reply% = reply% / 2
// send write command to CD, using 1 decimal place
result% = ANSIWRITE(12,"SP", reply%, 1)
IF result% = 0 THEN
        PRINT "Message could not be sent"
        GOTO top:
```

```
ENDIF
CALL get_reply:   // write accepted?
IF reply% <> -65540 THEN   // –65540 is ACK
        PRINT "Error in sending message to CD"
ENDIF
GOTO top:
}
get_reply: {   // sub-routine to wait for a reply
        timeout% = 0   // reset timer
        DO
        // returns –65536 if the reply has not been received yet
        reply% = ANSIREPLY
        LOOP WHILE reply% = –65536 AND timeout% < 50
        // loop exits only when a reply has been received, or
        // a time-out has occurred.
        // if the clock task is set at 5ms, the timeout will be 250ms
}
CLOCK {
timeout% = timeout% + 1   // increment timeout variable
}
```

Note

Refer to the help file for further examples of the ANSI commands.

7 Reference

7.1 Tasks

Refer also to Tasks and real-time programming in Chapter 4 DPL Programming.

INITIAL task

The INITIAL task is used typically to initialize program variables and Drive parameters.

The INITIAL task boots-up the DPL program and runs only when the MD29 is reset or at the moment AC power is applied to the Drive.

The INITIAL task is special in that all other tasks are guaranteed not to be running when it is being run. This is significant when other real-time Tasks are to manipulate data which have initial values.

BACKGROUND task

The BACKGROUND task is used for functions and commands that do not require time-related or encoder-related monitoring. It would be used for the following:

- Data logging
- Checking digital inputs
- Setting output status

The BACKGROUND task runs after the INITIAL task is completed. It is recommended that the majority of the program is run in the BACKGROUND Task.

Note

The BACKGROUND task does not automatically loop.

Example

```
BACKGROUND{
  RAMP:
  #1.18 = 0
  DO WHILE #1.18 < 1000
    #1.18 = #1.18+1
  LOOP
  GOTO RAMP:
}
```

CLOCK task

The CLOCK task is used for time-critical monitoring of the Drive and commands to the Drive (eg. controlled acceleration or deceleration ramp).

The CLOCK task is executed on a fixed timebase, asynchronously to the Drive. The actual timebase used depends on the set-up parameter (see Set-up Parameters in Chapter 10 Parameters), which can range from 1ms to 200ms).

Example This example produces a sine wave.

```
CLOCK{
  #1.18 = SIN (rad)*1000
  rad = rad+0.01
  IF rad>6.283185 THEN
    rad = 0
  ENDIF
}
```

ENCODER task

A typical use for the ENCODER task is to monitor the activity of an encoder.

The task is synchronized to a control loop in the Drive, so the execution frequency of the task is determined by the Drive. A set-up parameter can be used to multiply the time by two.

Drive	Switching Frequency	Timebase = 0	Timebase = 1
	kHz	ms	ms
Mentor II	Not applicable	5.12	2.56
CDE	3, 6 or 12	5.52	11.04
CDE	4.5 or 9	7.36	14.72
Vector	Any	2.008	4.016

Example

```
ENCODER{
  master% = #90.2
  slave% = #90.4
  EPOS = EPOS + master% – slave%
}
```

EVENT task

The EVENT task is a special task which runs when a specific event occurs. The source of the event is determined by the Timer/Counter Unit.

Refer to Timer/Counter Unit in Chapter 8 Features for further information.

ERROR task

The ERROR task is executed when a run-time error occurs in a DPL program.

Refer to section Advanced error-handling in Chapter 9 Diagnostics for further information.

User-defined task

User-defined tasks are sub-routines written by the user and are used in conjunction with the CALL instruction (see *CALL* instruction later in this section).

User-defined tasks can be given any name and can be defined anywhere in the program.

Example Note that this task is named **RAMP:**.

```
BACKGROUND{
  Loop:
  CALL RAMP:
  GOTO Loop:
}
RAMP: {
  #1.18=0
  DO WHILE #1.18<1500
    #1.18=#1.18+1
  LOOP
}
```

Case sensitivity The name for the sub-routine is case sensitive. If the CALL instruction in the preceding example is written as **CALL ramp**, the program will not be compiled since the word **ramp** should be in upper-case letters. Note that the compiler error that is displayed will be as follows:

Undefined reference to _0103ramp.

NOTES task

This is a pseudo task that is ignored by the compiler. The writer of the program can use the NOTES task for information for the user.

Example

```
NOTES{
You can put your documentation here.
}
```

CONST section

The CONST section is used to define constant arrays data and is not a Task. See Variables in Chapter 4 DPL Programming.

7.2 Instructions and functions

The following instructions and functions may contain more than one form of syntax. Where there is more than one form, the first is for use with integer variables; the second is for use with floating-point variables. (See Chapter 4 DPL Programming).

ABS

Syntax result% = ABS (expression%)
result = ABS (expression)

This mathematical function returns the absolute value of an expression without taking into account either the negative or positive sign of the expression (negative numbers are made positive).

Example ABS(45.5 – 100) ;the output is 54.5

ANSIREAD

Syntax result% = ANSIREAD (drive address%, "mnemonic")

This is a 4-wire RS485 port function which always returns an integer.

This function transmits a parameter read-request via the ANSI RS485 port to a remote Drive or unit.

The Drive address is an integer expression, usually between 01 and 99.

The mnemonic is a string which contains the parameter number. The format and length of this string depends on the remote Drive.

The function returns **1** if the read request was sent successfully, or **0** if the message could not be sent (eg. transmission already in progress).

Note

The ANSIREAD instruction does not wait for a response from the remote Drive.

Example result% = ANSIREAD(12,"0122") \\ read #1.22 from drive 12

See also Example ANSI commands in Chapter 6 Serial Communications.

ANSIREPLY

Syntax result% = ANSIREPLY

This is a 4-wire RS485 port function which is used in conjunction with the ANSIREAD and ANSIWRITE functions. This function is typically used to obtain the returned data from a Drive immediately after the issuing of ANSIREAD or ANSIWRITE.

The following information is returned:

−65536 No reply received yet
−65537 Reply received, but with bad checksum
−65538 EOT received (i.e. parameter does not exist)
−65539 NAK received
−65540 ACK received

Any other value is the value of the mnemonic written in ANSIREAD.

See also Example ANSI commands in Chapter 6 Serial Communications.

ANSIWRITE

Syntax result% = ANSIWRITE(drive address%, "mnemonic",value%, attribute%)

This is a 4-wire RS485 port function which is used to transmit a parameter write request to a remote Drive via the ANSI RS485 port.

The Drive address is an integer expression, usually between 01 and 99. The mnemonic is a string which contains the parameter number. The format and length of this string depends on the Drive. The value must be an integer expression.

The following are the attribute arguments that are specified:

0 No decimal place
1 One decimal place
2 Two decimal places
3 Three decimal places
128 Hex write to the CT I/O box (six characters prefixed by **x**)
129 Hex write to CD Drive (four characters prefixed by >)
130 Hex write to CD Drive (two characters prefixed by >)

This function returns **1** if the write request was sent successfully, or **0** if the message could not be sent (eg. transmission already in progress).

Note

This instruction does not wait for a response from the remote Drive.

Example

```
result% = ANSIWRITE(13,"0211", 150, 1)   \\ set #2.11 to 15.0
on remote drive 13
```

See also Example ANSI commands in Chapter 6 Serial Communications.

ARCTAN

Syntax result = ARCTAN (expression)

This mathematical function returns the arctangent of the expression in radians.

Example x = ARCTAN(0.8)

This returns the value 0.674740942 radians.

See also *TAN* instruction.

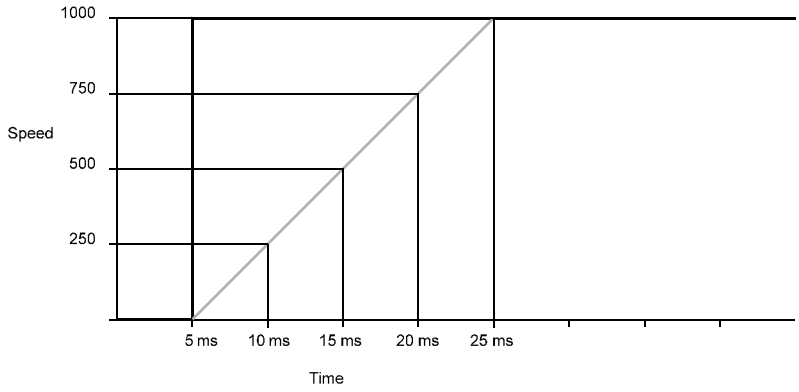
AVERAGE

Syntax result% = AVERAGE (input expression%, number of samples%)

This signal-processing function returns the running average of the input for the desired number of samples. The number of samples must be a constant integer value. The input expression is also an integer.

This function can be used only in the CLOCK or ENCODER tasks. AVERAGE requires execution on a regular timebase.

If a step change of 0 to 1000 is applied to parameter #1.18, the output of the averager will be as follows (assuming a 5ms clock timebase).



Example

```
CLOCK{
input% = #1.17
#1.18 = AVERAGE (input%, 4)
}
```

See also *FILTER* instruction.

BCD2BIN

Syntax int% = BCD2BIN (integer expression%)

This function converts a binary coded decimal number to a normal binary integer. It is used to operate on digital **input** data **from** the I/O Box (eg. thumbwheel switches). (Binary coded decimal is a method of writing a decimal number in a binary format.)

954 in decimal format is represented as 1001 0101 0100 in BCD.

9				***5***				***4***			
1	0	0	1	0	1	0	1	0	1	0	0

Example r% = BCD2BIN(#82.46)

BIN2BCD

Syntax int% = BIN2BCD (integer expression%)

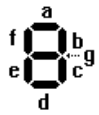
This function converts a normal binary integer to a binary coded decimal number. It is used to operate on digital output data to the I/O Box.

See *BCD2BIN* instruction for an explanation for converting decimal to binary coded decimal.

BCD2SEG

Syntax int% = BCD2SEG (integer expression%)

This function converts binary coded decimal to 7-segment LED codes and is used to operate on digital output data to the I/O Box. The output consists of drive signals for up to three 7-segment display digits, as follows:



Bit number	7	6	5	4	3	2	1	0
Segment	–	g	f	e	d	c	b	a

The least significant byte represents the first of the three digits. The maximum value which can be converted is 999. Leading zeroes are sent.

Example

```
a% = BIN2BCD(123)
seg_data% = BCD2SEG(a%)
#83.46 = seg_data%   ;output 5200646 (0x4F5B06 = 321)
```

CALL

Syntax CALL program-label:

This is a flow control instruction which is used to execute a Standard Application Program, or a user-defined Task. After the sub-routine has been completed, the program returns to the line following the instruction [CALL].

Example 1

```
ENCODER{
CALL diglk_encoder: // Standard application Task
//Digital Lock
...Rest of ENCODER Task
}
```

Example 2

```
INITIAL{
CALL SETUP:
}
SETUP:{ //This is the name of the sub-routine
#1.21 = 0
}
```

Note

It is not possible to single-step through a user-defined task.

The name of the sub-routine is case-sensitive.

COS

Syntax result = COS(expression)

This mathematical function returns the cosine of an angle expressed in radians. This function always returns a floating-point variable.

See *SIN* instruction and *TAN* instruction.

Example value = COS (3.1416)

CRC16

Syntax CRC% = CRC16 (character%, CRC%)

This function is used to calculate a 16-bit CRC (cycle redundancy check, commonly used in serial communications protocols).

DELAY

Syntax DELAY (Integer expression%)

The DELAY instruction causes the program to pause for a time specified by the integer expression in increments of 0.1 second. This instruction cannot be relied on to produce an accurate time delay. In the worst case, the time delay could be 100ms shorter than that specified (it will never be longer).

DELAY can be used only in INITIAL or BACKGROUND Tasks.

Example

```
#15.12 = 10   //set parameter 15.12 to 10
DELAY(10)    //delay program for 1 second.
#15.12 = 0
```

DIM

Syntax

```
DIM variable% [number_of_elements%]
DIM variable [number_of_elements%]
```

The DIM instruction is used to specify an array of a set of variables of the same type (integers or floating-point). The instruction does not produce any code, but tells the compiler to reserve space for a dynamic array.

An array must be specified before the array is used. It is recommended that an array is specified in the INITIAL Task, but it can be specified in the task the array is to be used in.

Example

```
INITIAL{
  DIM myarray%[100] //declares an integer called
  //[myarray] which has 100 elements.
  myarray%[0] = 10 //initialise the first element to   //10
  myarray%[1] = 20 //initialise the second element    //to 20
  myarray%[99] = 50 //initialise the last element
  index% = 1
}
CLOCK{
  myarray%[index%] = #3.02 //get speed feedback
  //on Mentor II
  index% = index + 1 //increment index counter
  IF index% = 100 THEN
    index% = 0
  ENDIF
}
```

See also Arrays under Variables in Chapter 4 DPL Programming.

DO WHILE

Syntax 1 DO WHILE Conditional expression
Instruction
LOOP

Syntax 2 DO
Instruction
LOOP WHILE Conditional expression

Syntax 3 DO WHILE Conditional expression LOOP

This is a loop or iterative instruction which causes a block of instructions to be repeated until a specific expression becomes false.

Syntax 1 allows the conditional expression be evaluated first. If the outcome is true, the instructions are executed. The program will continue in the loop until the conditional expression becomes false.

Syntax 2 allows the instructions be executed first, the conditional expression is evaluated. This ensures that the instructions in the loop are executed at least once.

Syntax 3 is identical to Syntax 1, except there are no executing instructions in the loop.

Example 1
```
DO WHILE #1.18 < 1000
    #1.18 = #1.18 + 1
LOOP
```

Example 2
```
DO
    a = a + 0.001
LOOP WHILE a < 6
```

Example 3
```
DO WHILE #3.02 > 10 LOOP
```

(If the value of #3.02 is greater than 10, the program will continue to loop.)

EXIT

Syntax EXIT

This is a flow-control instruction which provides a quick method of terminating the current task.

Example
```
CLOCK{
IF #18.22 = 1 THEN EXIT
...
}
```

(If the value of #18.22 is 1, exit from the rest of the CLOCK task.)

EXP

Syntax result = EXP (expression)

This mathematical function returns the exponential function ($e^{expression}$).

Example X = EXP(4.5)

This returns the value 90.0171313.

See also *LN* instruction.

FILTER

Syntax result% = FILTER (input_expression%, time_constant %)
result = FILTER (input expression, time_constant%)

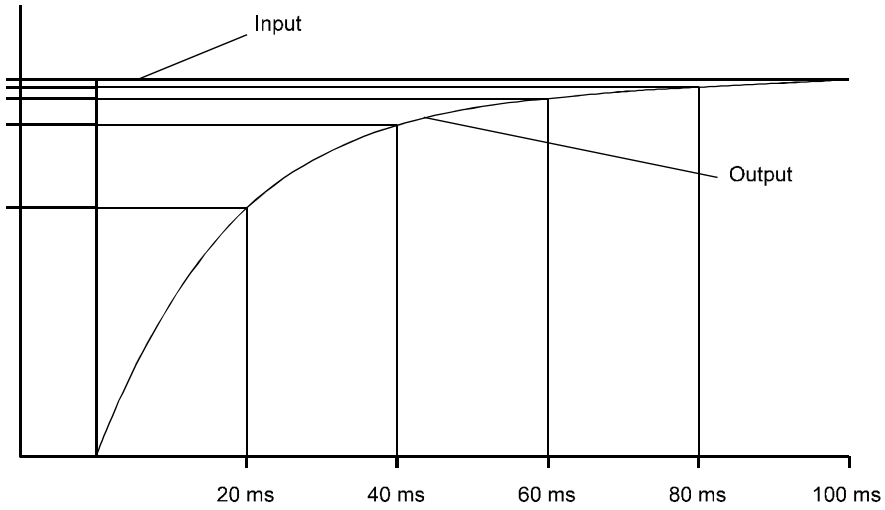
This is a signal processing function which returns the output of a first-order filter function with input expression as the input.

The time-constant of the filter depends on the value specified, and how often the FILTER instruction is executed.

Example
```
CLOCK{
  #1.21 = FILTER (#7.01, 4)
}
```

For a clock timebase of 5ms, the effective filter time-constant would be 4 x 5ms = 20ms.

If a step change of 0 to 1000 is applied to parameter #1.21, the output of the filter will be as shown in the following diagram.



Typically, the output reaches the final value after five time-constants.

This function must always be used in the CLOCK or ENCODER tasks.

FLOAT

Syntax result = FLOAT (integer expression%)

This function converts an integer variable into a floating-point variable.

Example

```
error% = 123
result = error% / 1000
```

This produces an output of [result = 0] since 123 / 1000 = 0 in integer arithmetic. To produce the correct answer, one of the operands need to be converted to floating-point.

```
error% = 123
result = FLOAT (error%) / 1000
```

This produces the correct answer: **result = 0.123**

See also *INT* instruction.

GETCHAR

Syntax result% = GETCHAR

This is a RS485 port function which reads in a character from the RS485 communications port.

The returned value is **–1** if no characters have been received in the buffer, otherwise the value is the ASCII code for the character read in.

See also *PUTCHAR* instruction.

GETKEY

Syntax result% = GETKEY

This is a RS232 port function which reads in a character from the RS232 PC communications port. This function can be used only when Toolkit communications are disabled (see Disable Toolkit communications in Chapter 10, Parameters).

This function can only be used if Dumb-terminal Mode is enabled (see Enable dumb-terminal mode in Chapter 10 Parameters).

The returned value is the ASCII code of the character read in. If there is no character, the value is **–1**.

The data format of the RS232 port is fixed as follows:

19200 baud, 8 data bits, No parity, 1 start bit, 1 stop bit.

See also *PUTKEY* instruction.

GOTO

Syntax GOTO label

This is an unconditional branch instruction which causes program execution to jump to, and continue from, the line specified by [label]. When declaring a label and when indicating which label to go to, a trailing [:] must be written with the label. The trailing [:] identifies the name as a label. This is different to other programming languages.

If the BACKGROUND task is to be continuously executed, a GOTO instruction must be included.

Example

```
BACKGROUND{
top:   //this is the name of the label
IF #15.22 = 1 THEN
  #1.04 = 100
ELSE
  #1.04 = 0
ENDIF
GOTO top:   //goto the line with the label [top:]
}
```

IF

Syntax 1

```
IF conditional expression  THEN
  Instruction
ENDIF
```

Syntax 2

```
IF  conditional expression  THEN
  Instruction if condition is true
ELSE
  Instruction if condition is false
ENDIF
```

Syntax 3

```
IF  conditional expression 1  THEN
  Instructions if true, goto ENDIF
ELSEIF  conditional expression 2  THEN
  Instructions if true, goto ENDIF
ELSEIF  conditional expression 3  THEN...
  Instructions if true, goto ENDIF
ELSE
   Instruction
ENDIF
```

Syntax 4

```
IF conditional expression THEN Instruction
```

The conditional instructions IF, THEN, ELSE, ELSEIF and ENDIF perform an operation until the specified condition is met.

Example 1

```
IF #1.21 = 25  THEN
  PRINT "25Hz"
ENDIF
```

Example 2

```
IF  a >= b AND (c%>– 1 OR z%>1)  THEN
  a = b
ELSE
  z = z+3
ENDIF
```

Example 3

```
IF A% = 1  THEN
  #1.11 = 0
ELSEIF a% = 2 OR a% = 3  THEN
  #1.11 = 1
ELSEIF a% > 4  THEN
  #18.21 = 1
ELSE
  #18.21 = 0
ENDIF
```

Example 3 uses the operator **OR** in the conditional expression. The following operators can be used to combine conditional expressions:

AND	Logical AND
OR	Logical OR
NOT	Logical NOT

The following conditional operators may be used in the conditional expression:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Note

The IF instruction for Syntax 1 to Syntax 3 must always end with ENDIF at the end of the set of conditional instructions.

INT

Syntax 1 result % = INT (float expression)

This function converts a floating-point variable to an integer variable (see *FLOAT* instruction) and rounds-up the result to the next whole number.

Example

```
        a = 14.234
res% = INT(a) / 2 //converts 'a' into an integer and divides
        //by 2
Result = 7
```

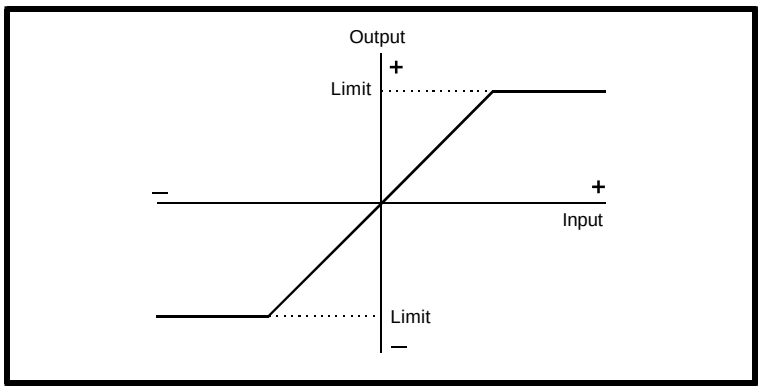
NOTE: if the integer value of the float expression exceeds 31 bits (214783647), the UD70 will trip on "Error 50 - Maths Error".

LIMIT

Syntax 1

```
result% = LIMIT (expression%, limit_value_expression%)
result = LIMIT (expression, limit_value_expression)
```

This mathematical function limits the value returned to the Drive parameters. This prevents out-of-range values being written to the parameters. This function applies to both the negative and positive values of the expression.



Example #1.18 = LIMIT (inc%, 1000)
;if inc% > 1000 then limit to 1000
;if inc% < −1000 then limit to −1000

This limits the output to ±1000.

See also *MIN* instruction and *MAX* instruction.

LN

Syntax result = LN (expression)

This mathematical function returns the natural logarithm of an expression.

Example y = LN(1.5)

This returns the value 0.405465108

See also *EXP* instruction.

MAX

Syntax result% = MAX (expression_A%, expression_B%)
result = MAX (expression_A, expression_B)

This mathematical function returns the greater of two expressions.

Example a% = 12
b% = MAX (a%, 100)

The value returned in **b%** is 100 because **a%** is less than 100. If **a%** is 105, the value returned in **b%** would be the value of **a%** (105).

MIN

Syntax result% = MIN (expression_A%, expression_B%)
result = MIN (expression_A, expression_B)

This mathematical function returns the smaller of two functions

Example a% = 12
b% = MIN(a%, 100)

The value returned in **b%** is 12 because **a%** is less than 100. If **a%** is 105, the value returned in **b%** would be 100.

PRINT

Syntax PRINT Expression list separated with a ",".

The PRINT instruction outputs strings or values to the PC RS232 serial port of the MD29. All items in the expression listing must be separated by commas. Strings are enclosed in double quotes (" ").

A tab character (ASCII 09) is automatically printed between each character separated by a comma. Negative values are prefixed by a – sign; positive values have no prefix.

Operation in various tasks

The way this instruction operates depends on the task, as follows:

In BACKGROUND Task, the PRINT instruction halts program execution until all characters are transmitted. It is recommended to use the PRINT instruction in the BACKGROUND task only.

In the CLOCK and ENCODER Tasks, program execution continues without waiting. At 19200 baud rate, it takes 0.5ms to transmit one character. The PRINT instruction would then quickly over-run the CLOCK or ENCODER Task. If another PRINT instruction is executed before the previous PRINT instruction is finished the remaining characters are not printed.

The string can contain non–printable characters by using an escape sequence. This begins with a back-slash [\] character followed by the non–desired character.

The available characters are as follows:

Character	Function	ASCII Character
\b	Backspace	8
\p	Line feed	9
\t	Tab	10

Table continued...

Character	Function	ASCII Character
\f	Form feed	12
\r	Carriage return	13
\v	Vertical tab.	11
\\	Single back-slash	92
\NNN	(N is an octal number)	

Example PRINT "Hello\r\n Goodbye\r\n"

This prints the following words:

Hello
Goodbye

Note

Do not over-use the PRINT instruction. It is better to use the Watch window in the Toolkit in order to monitor variables.

The PRINT instruction works in both normal-terminal and dumb-terminal modes. In normal-terminal mode, the DPL Toolkit must be used to monitor the print output.

The settings of the RS485 port are fixed at 19200 baud, 8 data bits, 1 stop bit, no parity.

PUTCHAR

Syntax result% = PUTCHAR (character%)

This is a RS485 port function that writes a character to the RS485 communications port.

If the character could not be written (eg. if the RS485 port buffer is full), the function returns **0**, otherwise it returns **1**.

PUTKEY

Syntax result% = PUTKEY (character%)

This is a RS232 port function that writes a character to the RS232 communications port when operating in dumb-terminal mode only. If the function fails, it returns **0**, otherwise it returns **1**.

Example r% = PUTKEY(65) // output 'A' to RS232 port

See also *PRINT* instruction.

REINIT

Syntax REINIT

Set up parameters are read by the MD29 only at the moment AC power is applied to the Drive or the MD29 is reset. If changes are made to the set-up parameters by a DPL program, the REINIT instruction can be used to force the MD29 to re-read them so that changes can take effect.

This instruction will not cause a reset. The program execution continues from the next instruction as normal.

Example

```
#14.04 = 25   //Changes the CLOCK task timebase to 25ms
reinit   //reinitializes the MD29 set-up to make change take effect
```

NOTE: REINIT does not read the POSITION LOOP ENABLE parameter. To enable or disable the position loop "on the fly", use parameter _Q20%.31. (See Chapter 8) To re-read the POSITION LOOP ENABLE parameter, the UD70 must be reset by writing 1070 to #88.01. This will cause the program to re-start, and pick up the change.

SIN

Syntax result = SIN (expression)

This mathematical function returns the sine of an angle expressed in radians.

See *COS* instruction and *TAN* instruction.

Example

```
value = SIN(3.1416)
```

SGN

Syntax

```
result% = SGN (expression%)
result = SGN (expression)
```

This mathematical function returns a value indicating the negative or positive sign of the input expression. When the input value is positive or zero, the function returns the value of **1**. When the input value is negative, it returns the value of **–1**.

Example

```
PRINT SGN (45), SGN (-16), SGN(0)
```

This prints values: 1, –1, 1

SQR

Syntax result = SQR (expression)

This mathematical function returns the square-root of an expression.

Example

```
PRINT SQR (25), SQR (16)
```

This prints values: 5 and 4

TAN

Syntax result = TAN (expression)

This mathematical function returns the tangent of an angle expressed in radians.

See *COS* instruction and *SIN* instruction.

Example result = TAN (3.1416)

TIME

Syntax result% = TIME

The TIME instruction returns the number of elapsed milliseconds since AC power was applied or the MD29 was last reset.

Example t% = TIME

WDOG

Syntax WDOG

The WDOG instruction is used to update the DPL program watchdog.

The watchdog facility is enabled by setting the appropriate set-up parameter at 1, and issuing a WDOG instruction. When enabled, the WDOG instruction must be executed within every 200ms. If a WDOG instruction is not executed within 200ms, the Drive trips on **Prc2 trip**.

The WDOG instruction can be used only in the INITIAL and BACKGROUND Tasks. The MD29 operating system automatically updates the individual watchdogs of the real-time tasks.

See the on-line Help for an example.

Notes

If any of these tasks are single-stepped a watchdog trip will occur.

When a watchdog trip occurs, the Drive trips on Prc2. Because the watchdog trip is a function of the Drive, the DPL program continues to run.

This function is not available on the Vector Drive.

8 Features

For Mentor only

This chapter covers the following features of the MD29:

- Single-axis position controller
- Timer/counter unit
- Digital I/O ports
- Non-volatile memory storage

8.1 Overview

Built into the operating system is a basic single-axis position controller. This can be used with a DPL program to control a Drive in closed-loop modes for applications such as cranes, indexing, etc.

8.2 PLC parameters

The position controller uses a special range of variables known as PLC parameters. These parameters are identified by a leading underscore (_) and the letter P, Q, R and S followed by a number, as follows:

Register	Range and type	Virtual menu used for access
_Px%	x = 0 to 99 (32-bit integer with polarity sign)	#70.xx
_Qx%	x = 0 to 99 (32-bit integer with polarity sign)	#71.xx
_Rx%	x = 0 to 99 (32-bit integer with polarity sign)	#72.xx
_Sx%	x = 0 to 99 (32-bit integer with polarity sign)	#73.xx

Note that the registers can also be accessed using the virtual menus 70 to 73. This gives the ability to alter the P, Q, R and S registers using RS485 ANSI serial communications.

The P and Q registers can be saved in the non-volatile memory of the MD29 by setting the appropriate set-up parameter at 1. See Non-volatile memory storage later in this chapter. The R and S registers cannot be saved.

The Q registers are reserved for use with the internal position loop if enabled. The P, R and S registers have no reservations on use, and can be used as general registers. If the CT Net option is fitted, the R and S registers may be used as transmit and receive registers for cyclic data transfer. If the position loop is disabled (#17.12 = 0), the Q registers may also be utilised as general registers.

Examples

```
_P1% = 15   // set _P1% to 15
_Q20%.5 = 1   // set bit 5 of _Q20% to 1
```

8.3 Introduction

An advanced position controller and profile generator is built into the operating system of the MD29. The function blocks are designed to provide position control and profile generation from one of the following:

- position reference.
- speed reference.
- incremental cam table.
- digital lock (provided an auxiliary encoder is used).

The main features of the position control software are:

- slave position control using linear or S-ramps for the velocity profile.
- slave speed control using linear or S-ramps for the velocity profile.
- smooth switching between position and speed control.
- rigid or non-rigid digital lock with a slave ratio range of 8, accurate to 8 decimal places.
- incremental cam table providing automatic control of the slave position, relative to the master position.
- smooth switching from cam or digital lock control to position or speed control
- position loop feedback source selectable between feedback (main) and auxiliary (reference) encoder inputs.
- three term PID control loop (D term configurable as feed-forward or derivative term) with the output written automatically to the fast access speed reference (#91.03).

When using the position control loop, it is important that the drive is also configured correctly. The drive should use #1.18 as the source of the speed reference (#1.14 = 1, #1.15 = 0) and bipolar reference should be enabled (#1.10 = 1). The speed ramps should be disabled (#2.02 = 0) as the profile generation blocks will provide ramp generation.

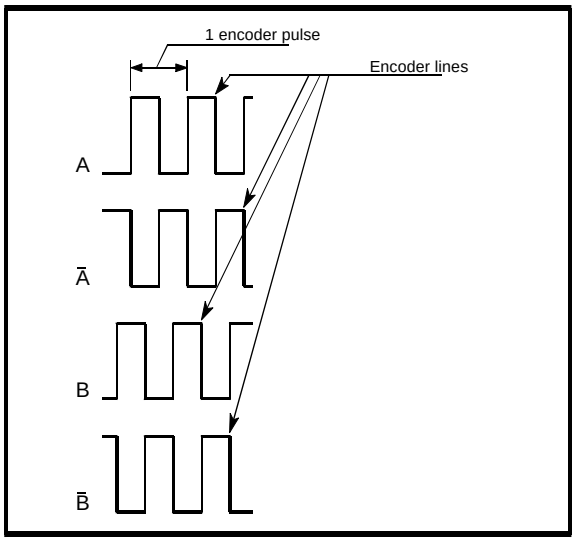
Selection between the different blocks is controlled using individual bits within registers _Q20% and _Q32%. These bits can be individually addressed or accessed as a whole word.

e.g.

```
_Q20% = 2       ; set bit 1 of _Q20% to 1, all other bits to 0
_Q20%.1 = 1     ; set bit 1 of _Q20% to 1, all other bits are unaffected
```

8.4 Encoder lines

All position, speed and acceleration parameters within the position controller use the units of Encoder lines. The diagram below shows how to determine the number of encoder lines for a particular encoder.



8.5 Position

Position values are absolute positions referenced from the power-up position. The position is measured in encoder lines where the number of lines per revolution = 4 * encoder pulses per revolution.

Velocity

The maximum velocity or speed will be specified for the overall system, and can be converted to the equivalent motor speed, N, in rpm.

To convert between the specified speed (N) and encoder lines per sec, use the following equations:

$$\text{lines per second} = \frac{\text{N * encoder ppr * 4}}{\text{60}}$$

$$\text{N} = \frac{\text{lines per second * 60}}{\text{encoder ppr * 4}}$$

Mentor II encoder speed feedback

If the Mentor II is using an encoder to provide the speed feedback from the motor, the automatic speed reference can be enabled.

Mentor II tacho speed feedback

If the Mentor II is using a tacho to provide the speed feedback, the speed reference conversion must be done in the DPL program. The Mentor II uses its own speed loop units, where 1000 units = maximum motor speed, while the position controller uses encoder lines per second to define the shaft speed of the feedback encoder.

To ensure that the demanded speed can be reached, the Mentor II must be scaled such that 1000 speed units produces at least the maximum required encoder shaft speed. The conversion between encoder lines per second (_Q0%) and Mentor II speed loop units must be executed in the ENCODER task. The equation that should be used is:

#91.3 = INT(LIMIT(_Q0% * 16000 / FLOAT(max_speed%)))

The variable max_speed% is the number of encoder lines per second being fed to the PID controller when the Mentor II is running at maximum speed or 1000 speed units. _Q24% or _Q25% will give a reading of the pulses per second when the encoder is turning, depending on which encoder inputs (feedback or reference) are being used.

The equation converts _Q0% to a fraction of the maximum speed, and multiplies by 16000 to convert to high resolution speed loop units. This is done as a floating point calculation to preserve accuracy. The result is limited to 16000, converted back to an integer value, and written to the high resolution speed reference.

Acceleration

For any axis on a particular machine, the maximum rate of acceleration or deceleration will be specified. The minimum time (t_{min}) to go from zero to N rpm can be calculated, and converted to equivalent changes in motor rpm.

To convert between the specified values (t_{min} and N) and encoder lines per sec^2, use the following equations:

$$\text{lines per second}^2 = \frac{N * \text{encoder ppr} * 4}{t_{min} * 60}$$

$$t_{min} = \frac{N * \text{encoder ppr} * 4}{\text{lines per second}^2 * 60}$$

When S-ramps are used, calculate the ramp rates as for a normal linear ramp. The S-ramp for that setting will take the same period of time, but will produce a higher peak acceleration. (See _Q13%).

NOTE: the units for Mentor II ramps have no effect on these calculations, as the internal ramps should be disabled. All profile ramps are generated and controlled by the position controller itself.

8.6 Enabling the position controller

The position controller must be enabled by setting #14.12 = 1. The position controller is synchronised to the ENCODER task timebase, which is set using #14.14.

The REINIT command does not read #14.12. To enable and disable the position controller while the drive is running, use _Q20%.31.

#14.14	ENCODER task timebase (ms)
0	5
1	2.5

Example calculations

The position loop is to be used to run the motor at a maximum of 2700rpm, with 300rpm headroom allowed for position recovery. The feedback device from the motor is a 1024 pulse per revolution encoder. The minimum linear acceleration time for the machine axis is 0.6 seconds go from zero to 2700 rpm.

Maximum speed _Q14%

$$\text{lines per second} = \frac{2700 * 1024 * 4}{60} = 184320$$

Digital lock speed _Q15%

$$\text{lines per second} = \frac{3000 * 1024 * 4}{60} = 204800$$

Acceleration rate _ Q12%

$$\text{lines per second}^2 = \frac{2700 * 1024 * 4}{0.6 * 60} = 307200$$

The settings for the position loop speed and acceleration parameters would be:

_Q14% = 184320	
_Q15% = 204800	maximum digital lock speed.
_Q12% = 256000	acceleration rate.
_Q13% = _Q12%	set acceleration and deceleration rates to the same value.

8.7 Default and Reset Values

The default values listed are values that will be assigned to certain parameters if the current value is invalid when the position controller is reset. "-----" indicates that there is no default value for that parameter.

The position controller can be reset in 3 ways:

MD29 HARD RESET,

MD29 SOFT RESET,

POSITION LOOP RESET.

A **HARD RESET** is a complete reset of the MD29 itself. With a Mentor II, this occurs when:

- #14.16 is set to 1.
- The Mentor is powered-up.
- The RESET button on the MD29 is pressed.
- The RESET button on the Mentor II is pressed, when the drive is disabled.

The system will restore the Q register values stored in the EEPROM memory, and all system file initial routines will run. (NOTE: #17.19 will store the current _Q values and immediately re-read them, resulting in no change.)

A **SOFT RESET** does not read the stored Q register values, but causes the DPL INITIAL task to run. This can only occur on a Mentor II when the MD29 is reset using the DPL Toolkit. This also causes the position controller to restart.

A **POSITION LOOP RESET** is caused by setting the position loop reset bit (_Q32%.17) to 1. This does not affect the operation of the MD29 DPL tasks, but does restart the position controller.

In the parameter descriptions, the RESET field indicates what value is assigned to a parameter by a hard reset (H), a soft reset (S) and a position loop reset (P).

S/V indicates that the saved value in _Qxx% is assigned. If #17.20 = 1, the last value before power-down is stored.

N/A indicates that the value is not affected.

CAL indicates that the parameter is re-calculated on every cycle of the position controller, and therefore does not have a default or reset value.

8.8 Parameter descriptions

_Q0%	**Final speed demand**
Units	Encoder lines per second
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = CAL S = CAL P = CAL

_Q0% gives the speed reference output from the PID controller in units of encoder lines per second.

_Q1%	**Auxiliary encoder position**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P =

_Q1% shows the position of the auxiliary (or reference) encoder, relative to the position at power up. The DPL program can change the value in this parameter at any time.

_Q2%	**Slave target position reference**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = 0

_Q2% is the target position for the slave, and is used as the input to the profile generation block. When selected, the control loop will attempt to make the feedback position (_Q8% or _Q1%) equal to _Q2%. The actual speed profile is modified by the ramp rates (_Q12% and _Q13%), ramp type (linear or S-ramp), and maximum speed (_

When running in speed mode (_Q32%.3 = 1), _Q2% is updated with the calculated stopping position.

_Q3%	**Internal speed reference**
Units	Encoder lines per second
Range	-2^{31} to 2^{31}
Default	0
Reset	H = 0 S = 0 P = 0

_Q3% is the target speed for the slave, and is used as the input to the profile generation block. The actual speed profile is modified by the ramp rates (_Q12% and _Q13%) and ramp type (linear or S-ramp). The speed profile is integrated to provide a position reference to the position control loop.

_Q4%	**Slave axis reference**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = 0

_Q4% is the final profiled position reference for the slave. The source of this value can be from the slave position profiler, the slave speed profiler, the digital lock controller or the cam table profiler, depending on the settings in _Q32%.2, _Q32%.3 and _

_Q5%	**Proportional gain**
Units	0.1%
Range	0 to 2^{31}
Default	1000
Reset	H = S/V S = N/A P = N/A

_Q5% defines the amount of proportional gain for the PID loop. For an error of 1, and a gain of 1000, the output of the P term will be 1.

_Q6%	**Integral gain**
Units	0.1% per second
Range	0 to 2^{31}
Default	0
Reset	H = S/V S = N/A P = N/A

_Q6% defines the amount of integral gain for the PID loop. For a constant error of 1, and a gain of 1000, the output of the I term will reach 1 after 1 second.

_Q7%	**Derivative/feedforward gain**
Units	0.1% per second
Range	0 to 2^{31}
Default	1000
Reset	H = S/V S = N/A P = N/A

_Q7% defines the amount of derivative or feed-forward gain (depending on _Q20%.0) for the PID loop. For a constant rate of change of error of 1 count per second and a gain of 1000, the output of the D term will be 1.

_Q8%	**Absolute slave axis position**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = 0

_Q8% shows the position of the slave (or feedback) encoder, relative to the position at power up. The DPL program can change the value in this parameter at any time.

_Q9%	**PID loop reference**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = 0

_Q9% is the position reference to the PID control loop. By setting _Q20%.1 to 1, the profiled slave axis reference (_Q4%) will be written to _Q9%. If _Q20%.1 is set to 0, the DPL program can write a position reference directly to _Q9%, but no ramps will be applied to the reference.

_Q10%	**PID following error**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	CAL
Reset	H = CAL S = CAL P = CAL

_Q10% shows the following error between the PID loop reference (_Q9%)and the feedback position (_Q8% or _Q1%). The value calculated is in encoder lines.

_Q11%	**Ratio for digital lock**
Units	Ratio * 10^8
Range	0 to 800000000
Default	100000000
Reset	H = S/V S = N/A P = N/A

_Q11% specifies the ratio applied to the digital lock function, multiplied by 10^8. This gives a maximum ratio of 8.00000000 (8 decimal places). Values in _Q11% are clamped to the maximum and minimum range.

_Q12%	**Maximum rate of acceleration**
Units	Encoder lines per second^2
Range	1 to 2^{31}
Default	20480
Reset	H = S/V S = N/A P = N/A

_Q12% defines the maximum rate of change of speed (acceleration) used within the speed and position profile generators. This applies to acceleration in both the forwards and reverse directions.

If S-ramps (_Q32%.8 = 1) are selected, _Q12% is not used.

_Q13%	**Maximum rate of deceleration/S-ramp**
Units	Encoder lines per second^2
Range	1 to 2^{31}
Default	_Q12%
Reset	H = S/V S = N/A P = N/A

_Q13% defines the maximum rate of change of speed (deceleration) used within the speed and position profile generators. This applies to deceleration in both the forwards and reverse directions.

If _Q13% is set to 0 or a negative value when the UD70 is reset, the value used for the acceleration ramp (_Q12%) will be written to _Q13%.

If S-ramps (_Q32%.8 = 1) are selected, _Q13% defines the ramp rate. This is set in the same way as for a linear ramp. The S-ramp will take the same period of time as the equivalent linear ramp, resulting in a higher (*$\sqrt{2}$ * linear acceleration) peak acceleration.

_Q14%	**Maximum speed**
Units	Encoder lines per second
Range	0 to 2^{31}
Default	20480
Reset	H = S/V S = N/A P = N/A

_Q14% defines the maximum rate of change of position (speed) used within the speed and position profile generators. The value set in _Q14% should not be higher than the maximum speed setting for the drive. (See _

_Q15%	**Maximum digital lock speed**
Units	Encoder lines per second
Range	0 to 2^{31}
Default	20480
Reset	H = S/V S = N/A P = N/A

_Q15% defines the maximum speed that the digital lock controller can use for the recovery of position lost during the acceleration of the slave axis. _Q15% must be set to a higher value than _Q14% to provide headroom for recovering lost position, and should generally be set to the same value as the maximum speed for the drive.

_Q16%	**Maximum PI output**
Units	Encoder lines per second
Range	0 to 2^{31}
Default	32000
Reset	H = S/V S = N/A P = N/A

_Q16% defines the maximum output of the PI loop. If the PI output reaches the defined limit, the integrator is held at the maximum value. This prevents the integrator from "winding up" during periods of large or constant error. _Q16% only limits the combined output of the Pand I terms; the D term is added to the output AFTER the limit has been applied.

_Q17%	**Reserved**

_Q18%	**Reserved**

_Q20%	**Bit mapped control word** All bits of _Q20% are read-write bits. They act as switches to select between the various functions available. (See logic diagrams.) The remaining bits without descriptions are reserved for future use.
_Q20%.0	Selects the mode of operation of the D term. 0 feed-forward term. 1 derivative term.
_Q20%.1	Set to 1 to enable automatic position reference writing to the PID loop (_Q9% = _Q4%).
_Q20%.2	set to 1 to apply a first order filter to the D term.
_Q20%.3	Reserved.
_Q20%.4	Reserved.
_Q20%.5	Reserved.
_Q20%.6	set to 1 to enable automatic conversion of _Q0% to rpm and write to the fast access speed reference #91.03. NOTE: the Mentor II must be running in encoder feedback for this function to work.
_Q20%.7	_Q20%.7 selects the source for the position loop feedback. 0 feedback (main) encoder 1 auxiliary (reference) encoder
_Q20%.31	Provides a means of enabling and disabling the position loop while the drive is running. #14.12 must be programmed and stored to configure the position loop, but the REINIT command does not re-read parameter #14.12. 0 enable position loop 1 disable position loop NOTE when _Q20%.31 is set to 1, all position parameters are reset to zero, except _Q8%, _Q0% and #91.03. This means that the last calculated speed reference value will remain in #91.03, and the drive will continue to run at that speed. The DPL program should ensure that _Q0% and #91.03 are to reset to zero when _Q20%.31 is set. 0 enable position loop 1 disable position loop

_Q21%	**Reserved**

_Q22%	**Reserved**

_Q24%	**Auxiliary encoder speed feedback**
Units	Encoder lines per second
Range	-2^{31} to 2^{31}
Default	CAL
Reset	H = CAL S = CAL P = CAL

_Q24% shows the speed of the auxiliary (or reference) encoder. This value is calculated directly from the encoder lines and is not filtered.

_Q25%	**Main encoder speed feedback**
Units	Encoder lines per second
Range	-2^{31} to 2^{31}
Default	CAL
Reset	H = CAL S = CAL P = CAL

_Q25% shows the speed of the feedback encoder. This value is calculated directly from the encoder lines and is not filtered.

_Q26%	**Slave position offset**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	0
Reset	H = 0 S = 0 P = 0

_Q26% defines the position offset to be added to _Q4%. The offset position profile can only be used when there are no other position profiles currently in operation. _Q26% can be applied when the slave is following a speed profile.

If position control is being used (_Q32%.3 = 0) and S-ramps are selected (_Q32%.8 = 1), _Q26% cannot be used.

_Q27%	**Slave speed offset**
Units	Encoder lines per second
Range	-2^{31} to 2^{31}
Default	0
Reset	H = 0 S = 0 P = 0

_Q27% defines the slave speed offset. The offset can be applied during other speed and position profiles, and is added to _Q4% to provide the final loop position reference, _Q9%.

_Q31%	**Bit mapped indication parameter** All bits of _Q31% are read-only bits controlled by the operating system. The remaining bits without descriptions are reserved for future use. A value of 0 in any bit shows that each statement is false. _Q31% is reset to 0 when_Q32%.17 is set to 1.
_Q31%.1	The position loop following error has exceeded the value in _Q37%.
_Q31%.9	The slave is recovering the position lost during acceleration.
_Q31%.10	Position profile in progress. The axis has not yet reached the target target set-point.
_Q31%.11	Position offset in progress. The axis offset position has not yet been reached.
_Q31%.12	The axis is fully locked to the master reference from the auxiliary encoder input.

_Q32%	**Bit mapped control parameter** All bits of _Q32% are read-write bits. They act as switches to select between the various functions available. (See logic diagrams.) The remaining bits without descriptions are reserved for future use.
_Q32%.1	Selects the direction of the digital lock. This allows the axis to be locked to the master in speed or position, but running in either direction. 0 same direction. 1 opposite direction.
_Q32%.2	Selects the digital lock output as the position reference. This parameter is used in conjunction with _Q32%.3 and _Q32%.4. 0 see _Q32%.3. 1 select digital lock position reference.
_Q32%.3	Selects between the target position reference _Q2%, and the position reference derived from the slave speed reference _Q3%. This parameter is used in conjunction with _Q32%.2 and _Q32%.4. 0 target position reference _Q2%. 1 slave speed reference, _Q3%.
_Q32%.4	Selects the cam table output as the position reference. The cam function can only be used if the cam table has been correctly initialised. This parameter used in conjunction with _Q32%.2 and _Q32%.3. 0 see _Q32%.2 and _Q32%.3. 1 select the cam table position reference.

_Q32%.8	Selects S-ramps for the position reference profile. The acceleration profile will be sinusoidal. When S-ramps are selected, the acceleration and deceleration rates are defined by _Q13%. If the digital lock or cam table references are selected, _Q32%.8 will be reset to disable the S-ramps. 0 linear ramps. 1 S-ramps.
_Q32%.12	Reserved
_Q32%.14	Reserved
_Q32%.17	Provides a static reset of all read-only position parameters. This does not affect configuration parameters or latched encoder position values. When all values have been reset, _Q32%.17 is reset. This parameter should not be set when the drive is running, as it will cause a hard stop without ramps, and all position information will be lost. 0 no action 1 reset all position parameters.
_Q32%.19	Selects S-ramps for the speed reference _Q3%. The S-ramp will take the same amount of time for as the linear ramp, but results in a higher peak acceleration rate. (See _Q13%). _Q12% is not used when S-ramps are selected. NOTE: S-ramps are not available when accelerating up to line speed under digital lock control. 0 linear ramps. 1 S-ramps.
_Q32%.20	Selects S-ramps for the speed reference offset, _Q27%. The S-ramp will take the same amount of time for as the linear ramp, but results in a higher peak acceleration rate. (See _Q13%). _Q12% is not used when S-ramps are selected. 0 linear ramps. 1 S-ramps.

_Q32%.27	Ramps can be applied to the slave reference to prevent excessive rates of acceleration and deceleration, particularly when digital lock is enabled. If ramps are not selected, the auxiliary encoder reference is simply multiplied by the ratio (_Q11%). 0 rigid digital lock. 1 non-rigid digital lock.
_Q32%.28	Ramps can be applied to the slave reference to prevent excessive rates of acceleration and deceleration, particularly when digital lock is enabled. If ramps are not selected, the slave reference is simply a ratio (_Q11%) of the auxiliary encoder reference. 0 enable ramps. 1 disable ramps.
_Q32%.29	Disables the automatic writing of the return position (_Q40%) to _Q2% when switching out of digital lock control to position control (_Q32%.2 from 1 to 0). When disabled, _Q2% is updated once when digital lock is deselected, allowing a smooth stop with no overshoot. 0 enable automatic home position. 1 disable automatic home position.
_Q32%.30	Provides a limit switch function to prevent the axis travelling beyond a certain point when using the position reference _Q2%. When _Q32%.30 is set, and the speed is positive (direction is forwards), _Q2%, _Q4% and _Q9% are forced to the current feedback position (_Q8%), causing an instantaneous stop with no ramps. NOTE This function will only work when running in position control, and the feedback source is _Q8%. 0 no action. 1 limit switch active.
_Q32%.31	Provides a limit switch function to prevent the axis travelling beyond a certain point when using the position reference _Q2%. When _Q32%.31 is set, and the speed is negative (Direction is backwards), _Q2%, _Q4% and _Q9% are forced to the current feedback position (_Q8%), causing an instantaneous stop with no ramps. NOTE This function will only work when running in position control, and the feedback source is _Q8%. 0 no action. 1 limit switch active.

_Q34%	Cam table index
Units	Integer
Range	0 to tablesize%
Default	-----
Reset	H = S/V S = N/A P = 0

_Q34% shows the current cam table index. This indicates which section of the cam profile is currently in progress.

_Q35%	Cam table master starting position
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = 0

_Q35% latches the master position _Q1% at the point when the cam table is enabled. The value is over-written each time the cam table is re-started. The value is used continuously when the cam table is enabled as the relative start position, and should not be over-written by the user.

_Q36%	Cam table slave starting position
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = N/A

_Q36% latches the slave position reference _Q4% at the point when the cam table is enabled. The value is over-written each time the cam table is re-started. The value is used continuously when the cam table is enabled, and should not be over-written by the user.

_Q37%	Maximum following error
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = N/A

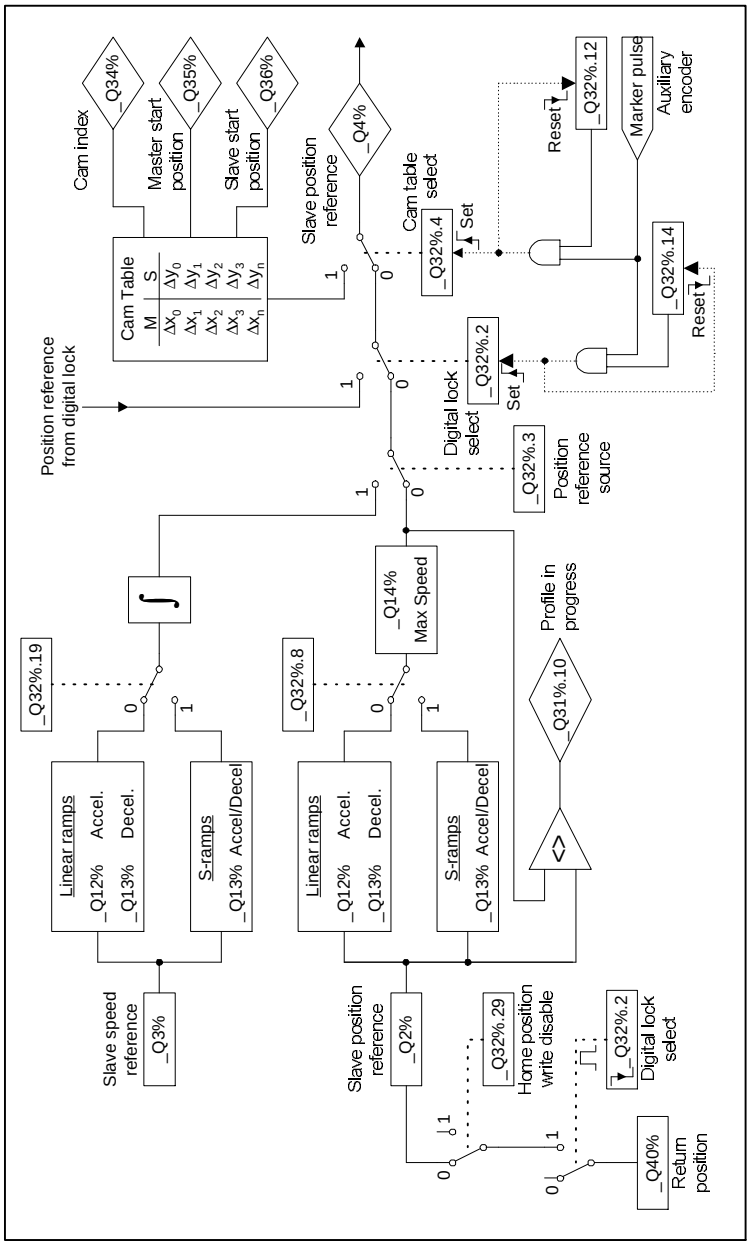
_Q37% defines the maximum value for the position or following error (_Q10%) error allowed for the PID loop. If the maximum following error is exceeded, _Q31%.1 is set.

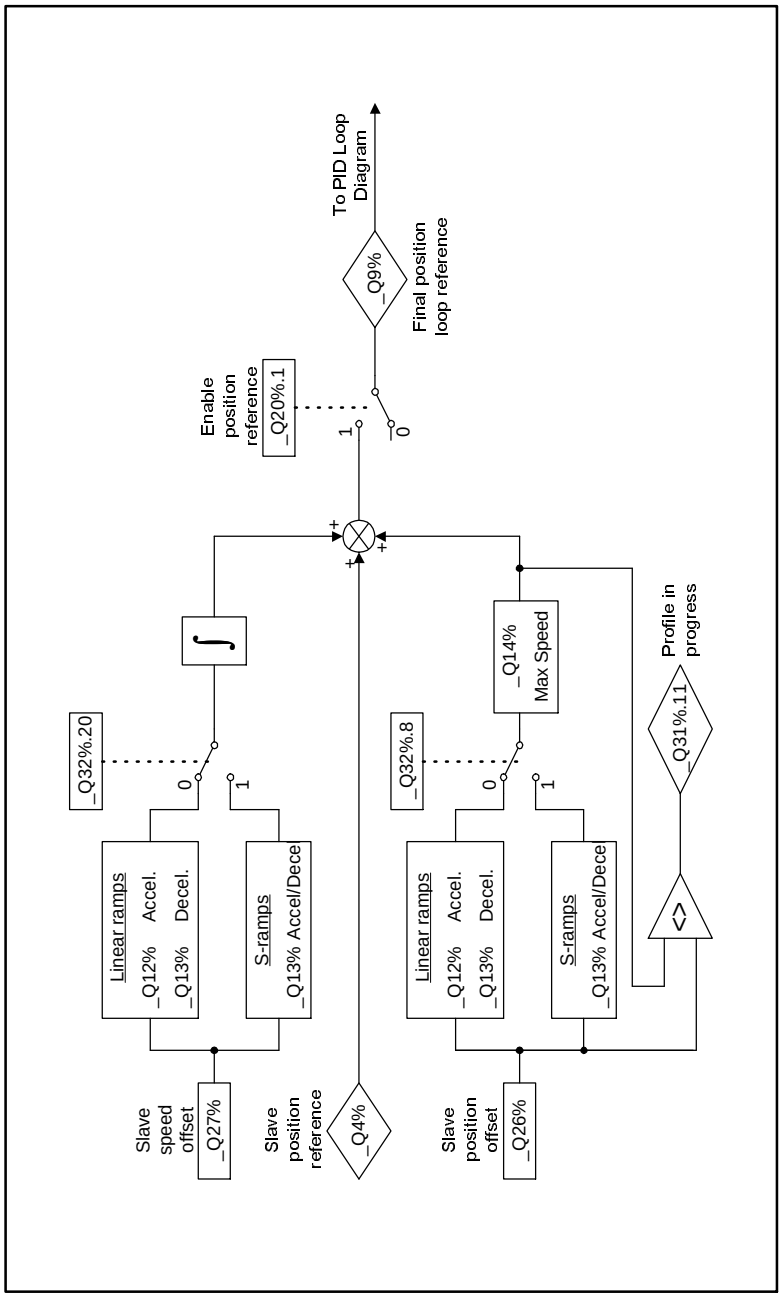
_Q40%	**Return position**
Units	Encoder lines
Range	-2^{31} to 2^{31}
Default	-----
Reset	H = S/V S = N/A P = N/A

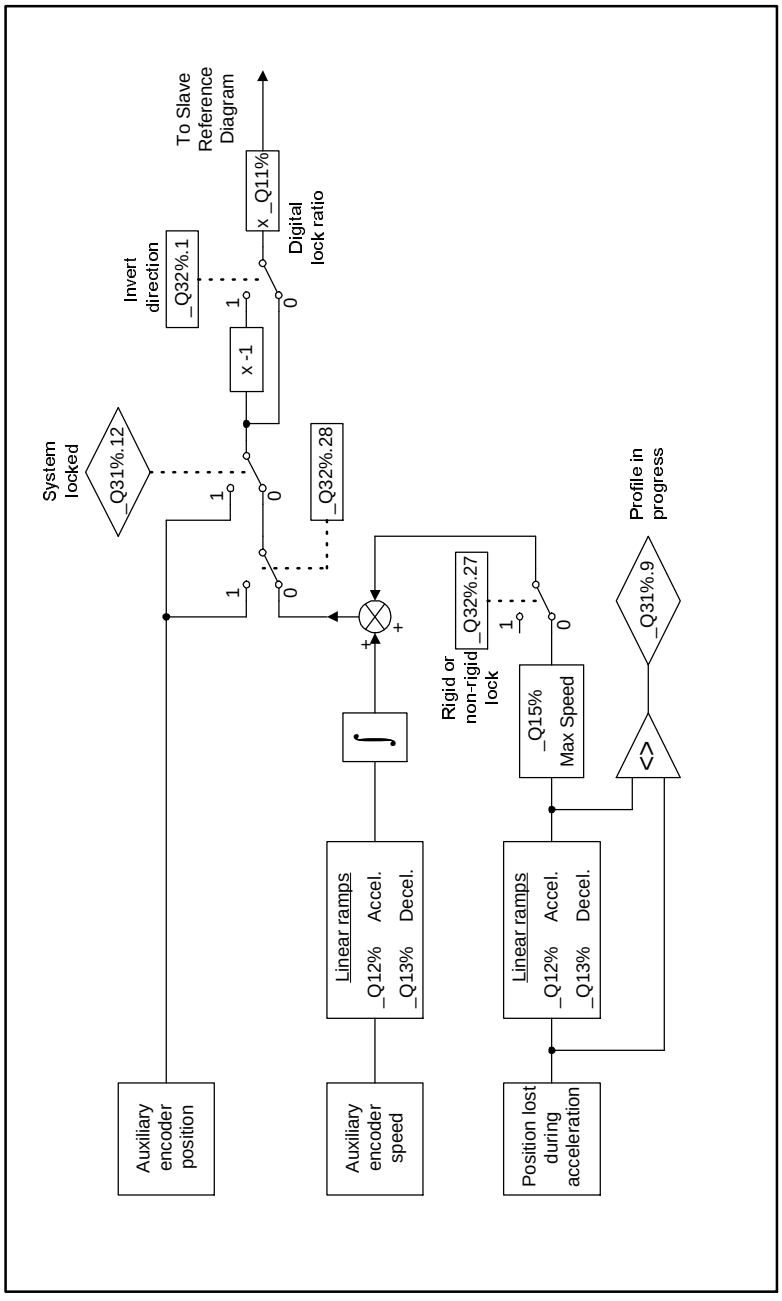
When switching out of digital lock and into position control mode (_Q32%.2 from 1 to 0), _Q40% is written to the main position reference _Q2%. The axis will return to the position given by _Q40%. The automatic write can be disabled by setting _Q32%.29 = 1.

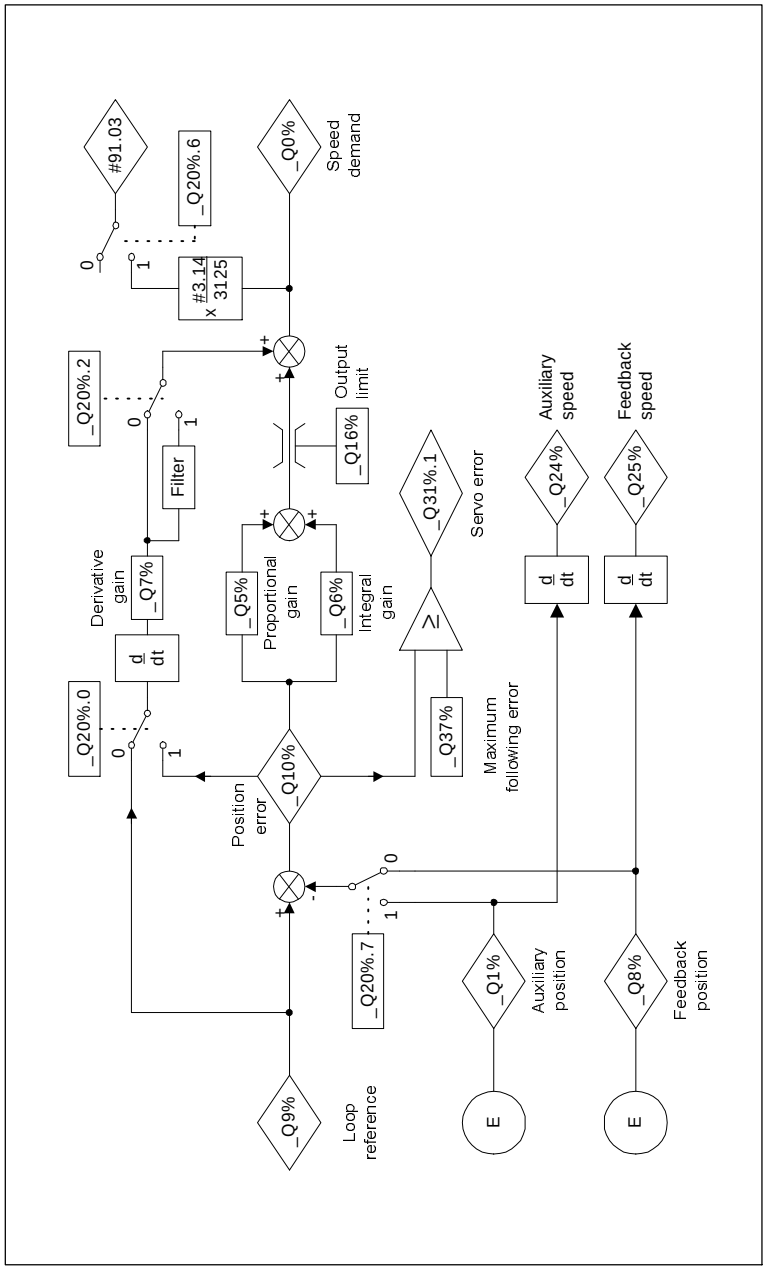
8.9 Logic diagrams

Slave main position reference




To PID Loop Diagram
_Q9%
Final position loop reference
Enable position reference
_Q20%.1
1
0
+
+
+
_Q32%.20
0
1
Linear ramps
_Q12% Accel.
_Q13% Decel.
S-ramps
_Q13% Accel/Dece
Slave speed offset
_Q27%
Slave position reference
_Q4%
_Q32%.8
0
1
Linear ramps
_Q12% Accel.
_Q13% Decel.
S-ramps
_Q13% Accel/Dece
Slave position offset
_Q26%
_Q14% Max Speed
Profile in progress
_Q31%.11
<>


To Slave Reference Diagram
x _Q11%
Digital lock ratio
Invert direction
_Q32%.1
1
0
x -1
System locked
_Q31%.12
1
0
_Q32%.28
1
0
+
+
Rigid or non-rigid lock
_Q32%.27
1
0
_Q15% Max Speed
Profile in progress
_Q31%.9
<>
Linear ramps
_Q12% Accel.
_Q13% Decel.
Linear ramps
_Q12% Accel.
_Q13% Decel.
Auxiliary encoder position
Auxiliary encoder speed
Position lost during acceleration



NOTE: Marker pulse functions are not supported by the MD29 on Mentor II.

Digital lock

The digital lock function allows the slave axis position and speed to be locked rigidly or non-rigidly to the master encoder. A digital lock ratio (_Q11%) can also be introduced between the reference and feedback, accurate to 8 decimal places.

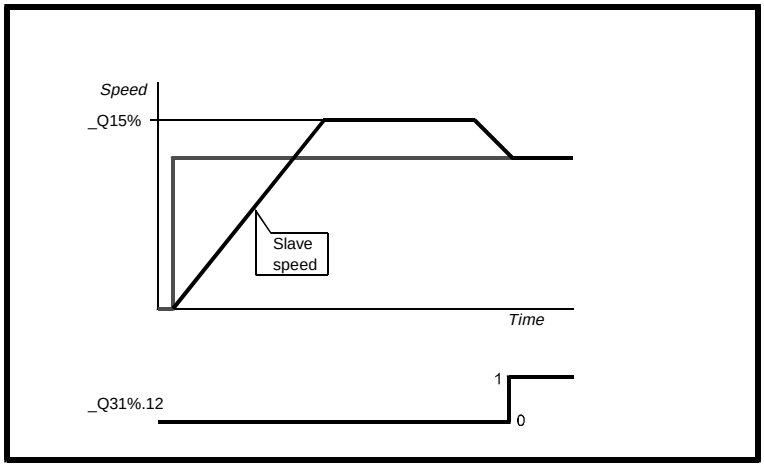
Rigid digital lock

Rigid digital lock is a position lock between the master reference and the slave axis. It is selected by setting _Q32%.27 = 0. When digital lock control is enabled (_Q32%.2 = 1), the slave axis will accelerate to the maximum digital lock speed (_Q15%) using linear ramps. (NOTE: if S-ramps are selected when digital lock is enabled, the slave will ramp up to the master speed using S-ramps, and recover position using linear ramps.) The axis will run at the maximum speed until any position error that built up during acceleration is recovered.

When position has been recovered, the slave speed will then lock to the master speed, multiplied by the ratio, _Q11%. _Q31%.12 is set once absolute position lock has been achieved.

NOTE:

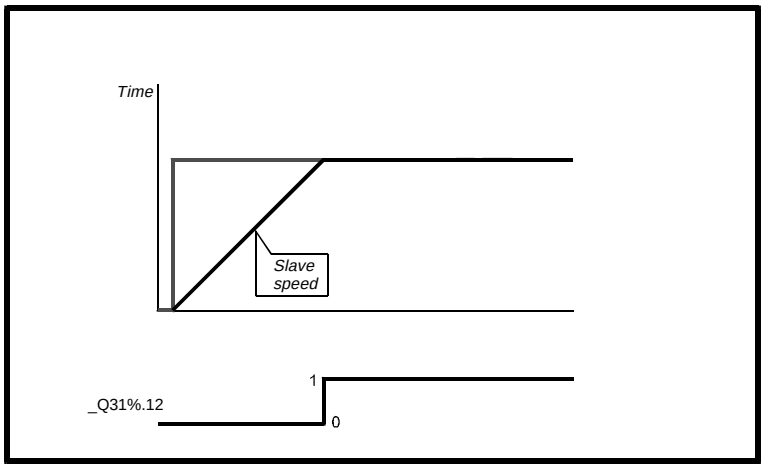
The maximum digital lock speed (_Q15%) should always be set to a value greater than the maximum line speed multiplied by the ratio (_Q11%). If this condition is not met, the axis may never achieve absolute position lock.



Non-rigid digital lock

Non-rigid digital lock is effectively a speed lock between the master reference and slave axis. It is selected by setting _Q32%.27 = 1. When digital lock is enabled (_Q32%.2 = 1), the slave axis will accelerate up to the master speed, multiplied by the ratio, _Q11%, using linear ramps. (NOTE: if S-ramps are selected when digital lock is enabled, the slave will ramp up to the master speed using S-ramps.) Any position error build up during acceleration is ignored.

When speed lock is achieved, _Q31%.12 is set, and position lock is used to keep the slave axis speed locked to the master reference.



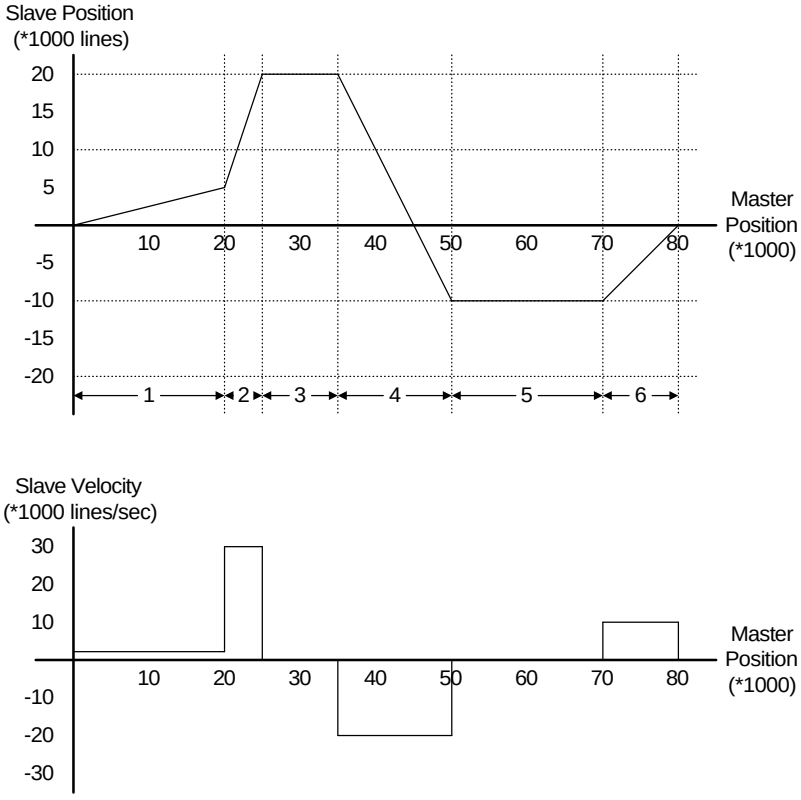
8.10 Using S-Ramps with Digital Lock

If S-ramps are selected, the master speed must be kept as constant as possible while the slave is accelerating to the master speed. Failure to ensure this may cause problems when completing the S-ramp profile, especially if the ramp rate is quite slow.

8.11 Cam function

The cam function provides a means of locking a slave axis to the continuous movement of a master encoder. The cam profile can be split into sections, for which the master and slave starting and finishing positions are known. The cam table is then built up by defining the change in position for each section, with respect to the section starting position, for both master and slave encoders. The reference point is the position of the slave at the instant the cam function is enabled. The software uses linear interpolation to move the cam through each section as the master position changes.

The following sections detail how to construct a cam table, using the example cam profile below. The velocity profile required to produce the position profile is also shown below. (The master speed is assumed to be constant at 10000 lines/second.)



Before the cam function can be used, an appropriate table must be initialised. This is best done at the start of the program in the INITIAL task.

setup% = CAMINIT(master%, slave%, tablesize%, reserved1%, reserved2%)

where:

setup%	=	0	initialisation was not successful.
	=	1	initialisation successful.
master%	=		name of the master array.
slave%	=		name of the slave array.
tablesize%	=		number of defined cam points. NOTE: each array can contain more defined positions, but both arrays must be at least this size to initialise correctly.
reserved1%	=	0	this argument is reserved for future use, and must be set to 0.
reserved2%	=	0	this argument is reserved for future use, and must be set to 0.

The master% and slave% arrays can be of different types, and both can be variable or constant arrays. The maximum number of elements in a single array is 500. Constant arrays are defined when the program is compiled, and cannot subsequently be changed unless the program is re-compiled. DPL programs are stored in the 96K of program memory, so the only limitation is that the compiled program size does not exceed 96K.

Dynamic arrays use the 8K of RAM, so the total number of array elements is far smaller. The advantage of a dynamic array is that values can be changed by the DPL program while the cam function is in use. If array elements are modified, changes should be complete before _Q34% gets to within 3 elements.

For the example cam profile below, the array names will be "master%" and slave%. The cam profile can be split into 6 separate sections, so 6 array elements will be used to re-create the profile.

The cam positions at the beginning and end of each section can be determined from the profile.

Section	Master		Slave		Position Change	
	Start	Finish	Start	Finish	master%	slave%
1	0	20000	0	5000	20000	5000
2	20000	25000	5000	20000	5000	15000
3	25000	35000	20000	20000	10000	0
4	35000	50000	20000	-10000	15000	-30000
5	50000	70000	-10000	-10000	20000	0
6	70000	80000	-10000	0	10000	10000

The last two columns produce the values required for each array. The arrays can be defined as follows:

```
CONST master%{
20000,5000,10000,15000,20000,10000
}
CONST slave%{
5000, 15000,0,-30000,0,10000
}
```

If the slave must always return to the same position at the end of each cam cycle, the sum of all the slave array elements must be 0. An easy way to do this is to construct the array using a spreadsheet, and copy and paste the series of numbers into the DPL editor.

When the cam is called in the DPL program, the following command line would be used.

```
setup% = CAMINIT(master%, slave%, 6, 0, 0)
```

8.12 Reference switching

The Position Controller provides the facility to switch between the various methods of control, without any discontinuities to the slave axis profile.

Switching to Position Control

When running in speed control, digital lock or cam table control, the axis stopping position is continuously re-calculated, and written to _Q2%. The current ramp settings in _Q12% and _Q13% are used to calculate the axis stopping position. S-ramps cannot be used when switching into position control.

NOTE: when a switch from digital lock control to position control occurs, a return position (_Q40%) will be written once into _Q2%, over-writing the calculated stopping position. This function can be disabled by setting _Q32%.29 = 1, so the axis will move to the calculated stopping position.

Switching to Speed Control

When the reference source is switched from position control, digital lock or cam table control to speed control, the speed reference will ramp up or down to the set speed reference, _Q3%. The ramps set in _Q12% and _Q13% are used to ensure a smooth switch. S-ramps can selected (_Q32%.19 = 1) when switching into speed reference control.

Switching to Digital Lock Control

When digital lock is enabled (_Q32%.2 = 1), the position controller locks the slave axis position to the master reference from that instant. The slave axis will accelerate to the master axis speed, multiplied by the ratio set in_Q11%.

If rigid lock is selected (_Q32%.27 = 0), the slave axis will accelerate to the maximum digital lock speed (_Q15%) until position is recovered. The speed will then reduce to the master line speed, multiplied by the ratio set in _Q11%.

Switching to Cam Table Control

When the cam table is enabled (_Q32%.4 = 1), the position reference follows a pre-defined profile, relative to the master axis. The starting positions of both master and slave are written to _Q35% and _Q36% respectively.

When switching from cam table control to position control, the axis stopping position is continuously re-calculated, and written to _Q2%. The current ramp settings in _Q12% and _Q13% are used to calculate the axis stopping position. S-ramps cannot be used when switching into position control.

For example applications and programs using digital lock, cam tables and reference switching, refer to the Help file.

Internal Control of Switches

Certain switch configurations of the profile generator are invalid, and will be automatically reset by the position controller.

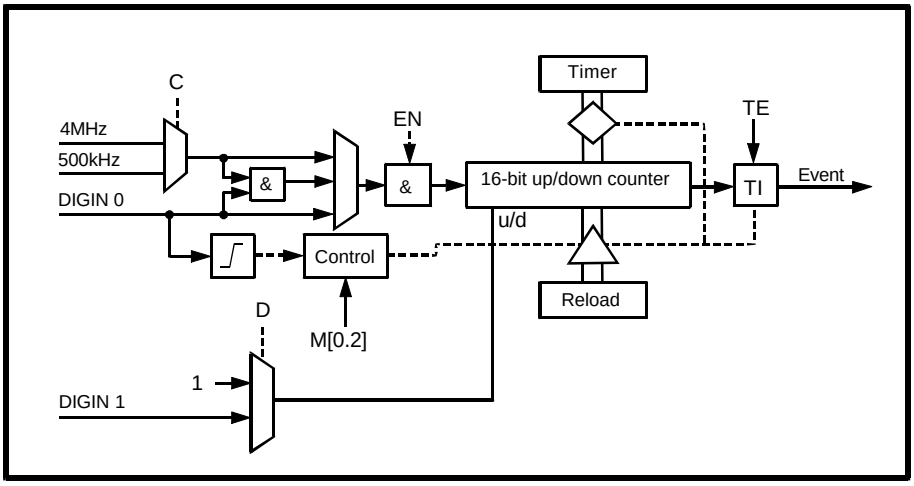
Digital Lock

When using digital lock, S-ramps cannot be used for position control, so _Q32%.8 is always reset to zero. When starting the digital lock using the auxiliary marker pulse, _Q32%.2 is set to 1 and _Q32%.14 is reset to 0 when the marker pulse input is detected.

Cam Table

When using the cam table, S-ramps cannot be used for position control, so _Q32%.8 is always reset to 0. When starting the cam table using the auxiliary marker pulse, _Q32%.4 is set to 1 and_Q32%.12 is reset to 0 when the marker pulse is detected. If the cam table has not been initialised correctly, _Q32%.4 cannot be set.

8.13 Timer/Counter unit



A hardware timer/counter is built into the MD29, and gives the following features:

- Count rate selectable between 500kHz and 4MHz
- The EVENT Task can be initiated when the timer/counter over/underflows, or by using digital input 0.
- The value of the timer/counter can be frozen by using digital input 0.
- The timer/counter unit can be clocked by activation of input 0 (digital TTL). The maximum input frequency is 1Mhz.
- Counter direction is selectable.

The timer/counter unit is controlled using virtual parameters in menu 85, as follows:

Parameter	Function
#85.01	Control Word
#85.02	Status Word
#85.03	Timer/counter value
#85.04	Timer/counter reload value
#85.05	Mode 2 Timer value

Control Word (#85.01)

Bit		Notes
0	TE	Timer EVENT enable 0 = Disable EVENT generation 1 = Enable EVENT generation When TE = 1 and TI = 1 (see below) the EVENT task will run.
1	EN	Enable timer 0 = Disable 1 = Enable timer
2	R	Reload mode on timer wrap-around 0 = No re-load. Timer simply wraps-around and continues to count 1 = Automatic re-load. The value of the reload reg is loaded into the counter Note 1: This flag has no effect in mode 2 (Capture mode) Note 2: Wrap-around is defined as over flow or under flow.
3	C	Internal clock pre-scale select 0 = 500KHz clock 1 = 4MHz clock
4	D	Direction control 0 = Counter counts up (u/d = 1) 1 = DIGIN1 controls the count direction. If DIGIN1 = 0 then count up If DIGIN1 = 1 then count down

Bit		Notes
5 ~ 7	M	Timer mode **Mode 0 (000): Normal mode** The internal clock source (4MHz or 500KHz) clocks the counter. The TI event flag is set when the counter wraps-around. **Mode 1 (001): External mode** The counter is clocked from an external source DIGIN0 (on the the −ve edge). The TI flag is set when the counter wraps-around. **Mode 2 (010): Capture mode** Capture mode. The counter is clocked by the internal clock source (4MHz or 500KHz). A 1 to 0 transition on DIGIN0 causes the current counter value to be latched into the TIMER register, the counter is then reloaded with the RELOAD register and the TI flag set. (Use #85.05 to read the timer value in mode 2, not #85.03.) **Mode 3 (011): Trigger mode** The internal clock source (4MHz or 500KHz) clocks the counter. However the TI event flag is set when a 1 to 0 transition is detected on DIGIN0, NOT when the counter wraps-around. The reload mode works in the same way as mode 0 or 1. **Mode 4 (100): Gated mode** The internal clock source (4MHz or 500KHz) clocks the counter; the count input is gated by the DIGIN0 input (active low). The TI event flag is set when the counter wraps-around.

Status Word (#85.02)

Bit		Notes
0	TI	Timer event flag 0 = No event 1 = Event has occurred. Two events are defined: Mode 0, 1 and 4: Counter wrap-around (overflow or underflow) Mode 2 and 3: 1 to 0 transition on DIGIN0 Note : The TI bit is automatically cleared if the TE flag is set in #85.01,.otherwise it is cleared when this status word is read.
1	OV	Wrap-around flag 0 = No wrap-around 1 = Counter/timer has wrapped around This flag is valid for ALL timer modes. Note: The OV bit is automatically cleared when the status register is read.

Timer #85.03 The current timer/counter value can be read and written at any time. When Mode 2 is selected, do not read this parameter (even in the **Watch** window), instead use #85.05 to read the timer/counter value.

The timer is a 16-bit timer with a range between 0 and 65535.

Re-load value #85.04 When the timer/counter overflows or underflows the OV flag is set. In Modes 0, 1 and 4, the TI event flag is also set. If the re-load mode is enabled (R set at 1), the timer/counter is initialized with the contents of the RELOAD register. (In Mode 2, re-load occurs when DIGIN0 transitions from 1 to 0).

When R (bit of #85.01) is set at 0, the timer/counter simply wraps-around and continues to count. The value in the re-load register can be read and written to at any time.

Mode 2 timer value (#85.03) Use this parameter to read the latched timer/counter value when Mode 2 is selected. Do not read #85.03.

For an example of using the timer/counter unit, refer to the **Timer Unit** topic in the DPL Toolkit on-line Help.

8.14 Digital I/O ports

The MD29 has two digital inputs and one digital output as standard. These inputs and output are TTL logic (ie. 5V logic), and are made available on the 9-way RS485 port connector, as follows:

Pin	Description
1	RS485 isolated 0V
2	RS485 /TX
3	RS485 /RX
4	DIGIN0 (TTL digital input 0)
5	DIGIN1 (TTL digital input 1)
6	RS485 TX
7	RS485 RX
8	TTL digital output
9	Drive 0V

The digital TTL inputs are used in conjunction with the timer/counter unit. In addition, they are also directly readable by a DPL program.

The digital TTL output can also be controlled directly from a DPL program. The following virtual parameters give access to the TTL I/O:

Parameter	Function
#86.01	Digital input 0 (DIGIN0)
#86.02	Digital input 1 (DIGIN1)
#86.03	Digital output

The inputs read **0** when at logic high (5V or unconnected), and **1** when at logic low (0V).

The output is at logic high (5V) when **0** is written.

Warning

The digital output is rated at a maximum of 15 milliamps (sink/source).

Connections to the digital inputs and outputs should be kept as short as possible (0.5 metre (20 in) maximum recommended). External buffering is required if longer cable lengths are used, or interfacing is needed to different logic levels.

8.15 Non-volatile memory storage

The MD29 has a feature which allows it to store the P and Q regions of the PLC register-set into the non-volatile memory of the MD29. The parameter used to initiate the save is as follows:

Mentor	Vector	CDE750	CDE7500
#14.16	F-14	#9.29	#15.20

This parameter immediately returns to zero once it has been set. Also, the MD29 is reset when the parameter is set. This re-starts the user program.

This feature can be used to store information such as the value of the diameter of a winder, or run-time counters, or any program variables. Also, when using the in-built position controller, all the setup parameters (prefix _Q) and position registers can be saved. The MD29 automatically restores all the registers when AC power is restored.

8.16 Using the RS232 port for Drive-to-Drive communications

The MD29 card and UD70 module have a mode in which the RS232 port can be linked to another MD29 or UD70 and two 32-bit variables can be transmitted between them.

The RS232 port can be used for Drive-to-Drive communications when the RS485 port is being used by another device (a user-interface (MMI), for example). The protocol used employs CRC checking to ensure a high degree of data integrity.

Warning

Due to the inferior specification of RS232, the serial communications cable connecting the Drives must be as short as possible (less than 10 metres (30 feet)). RS232–RS485 converters can be used to extend the cable length if required.

Note that serial communications multi-drop is not possible. The RS232 Drive-to-Drive link is designed solely for connecting to one external MD29 card or UD70 module.

No handshaking is performed on this link (ie. no error will reported if the connection is broken, for example).

Connections

The connections required to be made to the RS232 connectors are as follows:

Source unit	Destination unit
2	3
3	2
5	5

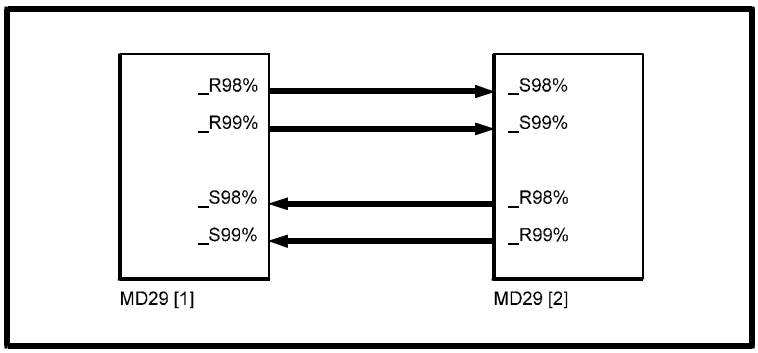
To enable the RS232 mode, set the following parameters at 1:

- **Disable Toolkit communications** set-up parameter
- **Drive-to-Drive communications** set-up parameter

See MD29 set-up parameters in Chapter 10 Parameters

Data-exchange parameters

The data to be transmitted must be placed in the special internal parameters _R98% and _R99%. Data received from the remote unit is placed in parameters _S98% and _S99%.



Source and destination parameters for transfer of data

9 Diagnostics

This chapter covers the following:

- Run-time errors and trip codes
- Compiler errors and warnings
- Advanced error-handling

9.1 Run-time errors

A run-time error is an error which occurs in a specific operation of the MD29. It could happen as a result of an error in the DPL program (such as trying to write to a parameter that doesn't exist, or trying to divide a value by zero), or in an automatic operation such as loss of communications with an I/O Box.

Certain run-time trips can be disabled (such as parameter write over-range). The parameters to enable the trips are as follows:

Function	Mentor	Vector	CDE750	CDE7500
Global run-time trip enable	#14.07	F-6	#9.21	#15.12
I/O Box trip enable	#14.08	F-7	#9.22	#15.13
Over-range trip enable	#14.10	F-9	#9.24	#15.15

The actions taken when a run-time error occurs are as follows:

- All DPL Tasks stop being executed.
- If the **Global run-time trip enable** parameter is set at 1, and the error does not involve serial communications, the Drive will be tripped. The trip causes the Drive to immediately disable its power outputs. If the error is related to the I/O Box, the Drive trips if the I/O Box Trip-enable parameter is also set at 1.
- If an ERROR Task is present in the DPL program, it starts being executed (see Advanced error-handling later in this chapter for details).

9.2 Run-time trip codes

When the Drive is tripped, the display shows the following:

Mentor II	**A29**
Vector	**St**
CDE	**trnn**

On the CDE Drive, the trip code is shown after **tr**.
The cause (and location) of the run-time error can be determined by reading the following parameter:

Type	Mentor	Vector	CDE750	CDE7500
Trip code	#10.35	F-3	#8.35	#10.30
Line number of trip	#16.63	Pr60	#9.07	#15.01

Alternatively, in a DPL program the trip-code number can be read in **#88.01**.
The possible trip codes are as follows:

Error number	Description	Action
41	Parameter does not exist.	Always trip
42	Parameter write failed: parameter is read only.	Trips if Global = 1
43	Parameter read failed: parameter is write only.	Trips if Global = 1
44	Parameter write failed: parameter value is over-range.	Trips if Global = 1 and Overrange = 1
45	Virtual parameter access failed: IOLINK is not running.	Trips if Global = 1 and IO Box = 1
46 ~ 48	Internal error.	Always trips
49	Wrong system loaded.	Always trips
50	Maths error in the program, eg. divide by zero, overflow, etc.	Trips if Global = 1
51	DPL array index is out of range.	Trips if Global = 1
52	Reserved.	Always trips
53	DPL program incompatible	Always trips
54	DPL overload – a Task has run out of time.	Trips if Global = 1
55	RS485 trip (mode 3, mode 4, etc).	Trips if Global = 1 and IO Box = 1*
56	Reserved	Always trips
57	Illegal Operating System Call	Always trips
58 - 59	Internal error	Trips if Global = 1
60 - 69	Reserved (Used on MD29 AN) **	
Prc2	See Watchdog trip (WDOG command)	

(* Trip 55 occurs only when an I/O Box is connected and operating, and Mode 3 or 4 serial communications fail.)
(** For further details refer to CTNet manual.)

9.3 Compiler error messages

This sections lists all the errors that can be generated when compiling a DPL program.

ERROR: **Argument x must be an integer**

The argument has been written as a floating-point number instead of as an integer.

Example ANSIWRITE (11, "SP", 50, 3.1)

Where:

11 = Argument 1
SP = Argument 2
50 = Argument 3
3.1 = Argument 4

Argument 4 must be an integer.

ERROR: **Array must be dimensioned**

An array must be defined before referencing it by a DIM statement. This gives the array its dimension before it is required (see DIM instruction in Chapter 7 Reference).

ERROR: **CALL can call only in-built functions or user tasks**

The CALL instruction can be used only to call a Standard Application or a user-defined Task (ie. the CALL instruction cannot call a label). (See the CALL instruction in Chapter 7 Reference).

ERROR: **DELAY can be used only in the INITIAL and BACKGROUND tasks**

By the nature of the ENCODER and CLOCK Tasks, these Tasks cannot support the DELAY instruction. A DPL program can be halted only when in the INITIAL or BACKGROUND Tasks (see DELAY instruction in Chapter 7 Reference).

ERROR: **DIM must have an integer number of elements**

The dimension of an array must be defined by an integer.

ERROR: **Empty Tasks are not permitted — remove the Task and recompile**

A Task has been defined without any instructions inside the braces. Remove the Task and the braces.

ERROR: **Expression is already a float — remove FLOAT instruction**

An expression which is already a floating-point variable, has been given a FLOAT instruction. Remove the FLOAT instruction.

ERROR: **Expression is already an Integer variable — remove INT instruction**

An expression which is already an integer variable has been given an INT instruction. Remove the INT instruction.

ERROR: **Expression too complex — break into parts**

ERROR: **Invalid Drive type. Use one of the following:**
MENTOR, VECTOR, CDE750, CDE7500

ERROR: **Label duplicated**

A label of the same name has been given more than once. Check other Tasks for duplication.

ERROR: **Label is in another task**

The GOTO instruction is in a different Task to where the label is defined. The label needs to be in the same Task as the GOTO instruction.

ERROR: **Label not found**

A label has not been defined. Define a label.

ERROR: **Operators only allowed on integer arguments**

ERROR: **Maximum bit-field size is 32**

The Bit-field invert Operator has been used and the specified bit-field is greater than 32. Re-specify the size of the bit-field.

ERROR: **Syntax error**

ERROR: **Variable has not been initialized**

Before a variable can be used as an argument, it must be given a starting value. This is typically performed in the INITIAL task. Remember that variable names are case sensitive.

ERROR: **Variable is not an array**

Warning messages

WARNING: **Title will be truncated to 64 characters**

WARNING: **Version will be truncated to 8 characters**

WARNING: **Possible loss of accuracy in assignment**

This warning indicates that a floating-point number has been assigned to an integer parameter. For example:

#1.18 = 3.142

As #1.18 can be set only with integer variables #1.18 now = 3.0

9.4 Advanced error-handling

Errors that occur when the program is running are usually due to programming errors, but can sometimes occur due to external influences. For example, an error signifying a serial communications loss could occur if incoming data from an I/O Box is lost due to the cable being broken. Normally, the MD29 halts all Tasks, and optionally trips the Drive.

If this is undesirable, the ERROR Task can be used. The sequence of events is as follows:

1. All Tasks are stopped.
2. The Drive is tripped (if the trip is enabled). See the Trip enable parameters in MD29 setup parameters in Chapter 10 Parameters.
3. The number of the error is placed in parameter #88.01 of the MD29
4. The ERROR task is executed. The instructions in the ERROR Task can determine the cause of the run-time error and take necessary action, such as stopping the drive system in a controlled manner.

All standard DPL instructions can be used in the ERROR Task.

The cause of the error can be determined by reading the virtual parameter #88.01. This gives the appropriate error code defined in Run-time Trip Codes earlier in this chapter.

When the **Global run-time trip enable** parameter is not set at 1, the Drive is not automatically tripped. If the Drive is required to be tripped, write the error code to the appropriate trip code parameter (shown in Trip codes earlier in this chapter).

To reset the MD29 and restart the DPL program, set parameter #88.01 at 1070.

Remember that no other DPL tasks will be running after a run-time error has occurred.

See the on-line help for an example of the ERROR task.

10 Parameters

10.1 MD29 set-up parameters

The set-up parameters take effect only after the MD29 has been reset. This occurs when any of the following actions is performed:

AC supply is applied to the Drive

A REINIT instruction in a DPL program is executed

When **Reset Target** is selected in the **Run** menu in the DPL Toolkit

The value 1070 is written to parameter #88.01

The following table lists all the MD29 set-up parameters.

Function	Mentor II	CDE750	CDE7500	Vector
ANSI Serial address	#14.01	#9.10	#15.05	F-0
This parameter defines the addresses for serial communications. Range: 11 to 99.				
RS485 Mode	#14.02	#9.11	#15.06	F-1
This parameter sets the Mode for serial communications. (See below).				
RS485 Baud rate (Mode 1, 5, 6 and 7)	#14.03	#9.12	#15.07	F-2
This parameter sets the Baud rate: 24 = 2400... 192 = 19200 etc. Maximum is 38400 which is set by a code of 38.				
Clock section timebase (ms)	#14.04	#9.13	#15.08	F-4
Defines the clock period time in milliseconds for executing the CLOCK task of a DPL program. Range: 1 to 200ms				
CTNet Node ID (MD29AN only)	#14.05	N/A	N/A	N/A
Specifies the node address for CTNet.				
Auto-run mode	#14.06	#9.20	#15.11	F-5
When set at 1, this parameter enables a DPL program to be automatically started when the MD29 is reset or AC power is applied. When set at zero, a command must be issued from the MD29 Toolkit software before the DPL program will start.				
Global run-time trip enable	#14.07	#9.21	#15.12	F-6
When set at 1, any run-time trips that occur will cause the Drive to trip. When set at 0, most run-time trips will not cause the Drive to trip.				

Function	Mentor II	CDE750	CDE7500	Vector
Trip if CT I/O box link fails	#14.08	#9.22	#15.13	F-7
When set at 1, and global trip is enabled, the Drive will be tripped if there is a communication link failure between the MD29 and the CT I/O box.				
Enable watchdog	#14.09	#9.23	#15.14	F-8
When this parameter is set at 1, the DPL program must execute a WDOG command at least every 200ms, otherwise the Drive will trip. See the WDOG command for more details.				
Trip if a parameter write over-ranges	#14.10	#9.24	#15.15	F-9
Each Drive parameter has a finite range of values which can be accepted. Any value which is outside the parameter limits could signify a program failure. When this parameter is set at 1, and global trips are enabled, the Drive will trip if a parameter is outside the limits. When it is set at 0, the MD29 places a limit on the value written.				
Disable Toolkit communications	#14.11	#9.25	#15.16	F-10
Setting this parameter at 1 puts the RS232 serial port into plain ASCII mode, or Drive-to-Drive RS232 comms mode. The DPL Toolkit will not function when this parameter is set at 1.				
Position controller enable	#14.12	#9.26	#15.17	F-11
When this parameter is set at 1, the internal position controller is enabled. See Chapter 8 Features. REINIT command does not read this parameter.				
I/O link synchronisation source	#14.13	#9.27	#15.17	F-12
When set at 0, the transfer of data to an I/O Box is synchronized to a CLOCK task. When set at 1, the transfer of data is synchronized to the ENCODER task. See below.				
Encoder timebase select	#14.14	#9.28	#15.18	F-13
This parameter sets the timebase for the ENCODER task. Mentor II: #14.14 (0 = 5.12ms, 1 = 2.5ms) CDE750, CDE7500: #9.28 (0 = 5.3ms at 3, 6 or 12kHz switching frequency) (0 = 7.36ms at 4.5 or 9kHz switching frequency) (1 = 11.04ms at 3, 6 or 12kHz switching frequency) (1 = 14.72ms at 4.5 or 9kHz switching frequency) Vector: F-13 (0 = 2.008ms, 1 = 4.016ms)				

Function	Mentor II	CDE750	CDE7500	Vector
Flash store request	#14.16	#9.29	#15.20	F-14
Set at 1 to save the PLC parameter registers (_Px% and _Qx%) into the non-volatile memory of the MD29. The parameter value immediately returns to zero. See Chapter 8 Features.				
Drive-to-Drive RS232 communications enable	#14.17	#9.30	#15.21	F-15
When set at 1, and DPL Toolkit communications are disabled (see Disable Toolkit communications above), Drive-to-Drive RS232 communications are enabled. See Chapter 6 Serial Communications.				
DPL line number where occurred	#16.63	#9.07	#15.01	Pr60
This parameter indicates the line on which an error has occurred, providing the DPL program has been compiled with the Debugging Information.				
RS485 parameter pointer #1	#11.09	#9.00	#15.03	Pr61
This parameter defines the destination and source for data to be transmitted to, or received from when in Serial Communications Mode 2 or 3.				
RS485 parameter pointer #2	#11.10	#9.02	#15.05	Pr63
This parameter defines the destination for data to be received from when in Serial Communications Mode 4. There will be no decimal point displayed (ie. to set parameter #1.21 as the destination, enter a value of 121).				
RS485 Mode 3 scaling	#11.10	#9.01	#15.04	Pr62
This parameter is used to scale data read into the MD29 while in Serial Mode 3. Refer to RS485 port modes later in this chapter. Parameter #14.05 in the Mentor II Drive is reserved.				

Operating in dumb-terminal mode

When using the MD29 with Toolkit communications disabled (also known as dumb-terminal mode), the DPL Toolkit is not able to communicate with the MD29.

Synchronization of the I/O link

When the CLOCK task is specified as the synchronization source for the I/O link, the link remains synchronized to a minimum CLOCK task timing period of 5ms. When the CLOCK task timing period is less than 5ms, the I/O link will run at a fixed 8ms, asynchronously to the CLOCK task.

When the ENCODER task is specified as the synchronization source for the I/O link, the I/O link will run as follows:

Mentor II	At every second ENCODER task	5.12ms
Vector	At every third ENCODER task	6ms
CDE	At every ENCODER task	5.5/7.8ms

10.2 Virtual parameters

Virtual parameters are special parameter which are not part of the standard parameter set of the Drive. Virtual parameters exist only in the MD29 and are used for the following:

- Accessing values not present as standard Drive parameters (eg. encoder counters)
- Accessing Drive parameters at a faster rate
- Accessing Drive parameters at an increased resolution
- Accessing MD29 parameters such as Timer/counter Unit control parameters, I/O Box data, etc.

Virtual parameters can be accessed by a DPL program or via the RS485 port. The virtual menus are explained in the following order:

90, 91, 70–73, 85, 86, 88, 80

Menu 90
General parameters

Parameter	Description
#90.01	Main encoder position (RO)
#90.02	Main encoder increments (RO)
#90.03	Auxiliary encoder position (RO)
#90.04	Auxiliary encoder increments (RO)
#90.10	Returns Drive type (RO) Codes are: 0 Mentor II 1 Vector 2 CDE750 3 CDE7500
#90.11	See Status word (RO)

Status word #90.11

Mentor Drive #90.11 contains a 16 bit number. The upper eight bits of #90.11 correspond with parameter #11.21 LED status in the Drive. The lower eight bits correspond with #10.25 Current trip number of the Drive. When the Drive is operating normally, the value of these bits is zero.

The upper eight bits of the status word are as follows:

MD29	b15	b14	b13	b12	b11	b10	b9	b8
Drive	#11.21							

The lower eight bits of the status word are as follows:

MD29	b7	b6	b5	b4	b3	b2	b1	b0
Drive	#10.25							

Refer to the Mentor II User Guide for further details.

Vector Drive The upper eight bits of #90.11 correspond with bit parameters 72 to 75 and 80 to 83 in the Drive. The lower 8 bits correspond with Pr57.

The upper eight bits of the status word are as follows:

b15	b14	b13	b12	b11	b10	b9	b8
b83	b82	b81	b80	b75	b74	b73	b72

The lower eight bits of the status word are as follows:

b7	b6	b5	b4	b3	b2	b1	b0
Pr57							

Refer to the Vector User Guide for further details.

Bookcase CDE #90.11 returns the inverted value of #8.18 in the Drive.

0 = Drive normal (not tripped)

1 = Drive tripped

CDE/HPCDE #90.11 returns the inverted value of #10.01.

0 = Drive normal (not tripped)

1 = Drive tripped

Menu 91
Drive-specific virtual parameters

Mentor II Drive parameters

Parameter	Description
#91.01	Mentor II digital feedback mode. 0 = disable MD29 feedback 1 = enable MD29 feedback
#91.02	Digital feedback (into #3.02) Range: ±16000 ±100%
#91.03	High resolution # 1.18 Range: ±16000 ±100%
#91.04	High resolution # 3.18 Range: ±16000 ±100%
#91.05	High resolution # 1.03 (RO) Range: ±16000 ±100%
#91.06	High resolution #3.02 (RO) Range: ±16000 ±100%
#91.07	High resolution #4.08 Range: ±16000 ±100%
#91.08	High resolution #4.09 Range: ±16000 ±100%
#91.09	High resolution #7.05 (RO) Range: ±16000 ±100%
#91.10	High resolution #3.01 (RO) Range: ±16000 ±100%

CDE Drive parameters

Parameter	Description
#91.01	Fast enable. Set at 1 to enable #91.2
#91.02	Fast frequency set-point.
#91.03	Analog Input 1
#91.04	Analog Input 2
#91.05	Analog Input 3
#91.06	Torque feedback

Vector Drive parameters

Parameter	Description	Related virtual parameter
#91.01	Set bit 0 at 1 to enable **Pr1**	#91.02
	Set bit 1 at 1 to enable **Pr8**	#91.03
	Set bit 2 at 1 to enable **Pr16** #91.02	#91.04
#91.02	Fast-access to **Pr1**	
#91.03	Fast-access to **Pr8** (integer)	
#91.04	Fast-access to **Pr16** (integer)	

When changing a parameter value in the Vector Drive, there may be a 64ms delay before the new value takes effect. Similarly, when reading a parameter, the value read back could be up to 64ms out-of-date.

The three commonly-used parameters **Pr1**, **Pr8**, and **Pr16** have a fast-access mode that gives an update rate of 2ms. To use the fast-access mode, virtual parameters must be assigned to the parameters. To do this, set at 1 the related bit in virtual parameter **#91.01**, as shown below:

Note

The virtual parameters for the Vector Drive are integer only. Parameters Pr8 and Pr16 are accurate to one decimal place. Setting parameter #91.03 at 15, sets Pr8 at 1.5.

Menus 70 to 73

These menus refer to the PLC register-set, as follows:

#70.xx	_Px%
#71.xx	_Qx%
#72.xx	_Rx%
#73.xx	_Sx%

Where x is a value from 0 to 99.

Menu 85
Timer/counter parameters

Refer to Timer/counter unit in Chapter 8 Features.

Parameter	Function
#85.01	Control Word
#85.02	Status Word
#85.03	Timer/counter value
#85.04	Timer/counter reload value
#85.05	Mode 2 Timer/counter value

Menu 86
Digital I/O parameters

See Digital I/O ports in Chapter 8 Features for connection information.

Parameter	Function
#86.01	Digital input 0 (DIGIN0)
#86.02	Digital input 1 (DIGIN1)
#86.03	Digital output

An input will read 1 when not connected, or at a logic high (+5V). Maximum input frequency is 1Mhz. Setting #86.03 at 0 causes the output to go logic low (0V).

Warning

The digital output is rated at a maximum of 15 milliamps (sink/source).

Connections to the digital inputs and outputs should be kept as short as possible (0.5 metre (20 in) maximum recommended). External buffering is required if longer cable lengths are used, or interfacing is needed to different logic levels.

Menu 88
Status parameters

Parameter	Function
#88.01	READ Run-time error code. Only valid in ERROR task. WRITE Writing a value of 1070 resets the MD29. See Advanced error-handling in Chapter 9 Diagnostics.

Menu 80
I/O Box parameters

Parameter number	Menu 80 analog IN	Menu 81 analog OUT	Menu 82 digital IN	Menu 83 digital OUT	Menu 84 control
0	Null	Null	Null	Null	Null
1	ADC1 (±4000)	DAC1 (±1000)	bit 1	bit 1	0 = ADC1 1 = 4–20mA
2	ADC2 (±1000)	DAC2 (±1000)	bit 2	bit 2	
3	ADC3 (±1000)	DAC3 (±1000)	bit 3	bit 3	
4	ADC4 (±1000)		bit 4	bit 4	
5	ADC5 (±1000)		bit 5	bit 5	
....					
32			bit 32	bit 32	
40			bits 1 – 8	bits 1 – 8	
41			bits 9 – 16	bits 9 – 16	
42			bits 17 – 24	bits 17 – 24	
43			bits 25 – 32	bits 25 – 32	
44			bits 1 – 16	bits 1 – 16	
46			bits 9 – 32	bits 9 – 32	

Note

Menus 81 and 83 parameters are write-only parameters. They cannot be read back via the Serial Communications link.

10.3 RS485 port modes

These modes are set using the RS485 Mode parameter. Refer to MD29 set-up parameters at the beginning of this chapter for details of the source, destination and scaling parameters.

Mode	Description
1	Standard 4-wire RS485, ANSI protocol.
2	Master mode. A binary protocol is used to continually transfer the value of a defined parameter to another MD29, UD70 or Drive. The data is scaled ±16000. The baud rate is fixed at 9600.
3	Slave mode. The MD29 receives data, applies a scaling and places the final value in a specified parameter.
4	Cascade mode. The value of a defined parameter is transmitted, and received data goes into another defined parameter. No scaling is applied.
5	2-wire RS485, ANSI protocol.
6	User Mode. This Mode turns off all internal protocols and allows the user to use the RS485 port directly from the DPL program. Typically, this Mode is used in conjunction with the DPL ANSI master commands (ANSIREAD, ANSIWRITE, etc.). User defined protocols can also be implemented in DPL with the low level PUTCHAR and GETCHAR commands. The communication data-frame is organised as: 1 start bit, 7 data bits, EVEN parity and 1 stop bit (10-bits total).
7	User Mode. Same as Mode 6, but the communications data-frame is organized as: 1 start bit, 8 data bits, EVEN parity and 1 stop bit (11-bits total).
8	User Mode. Same as Mode 6, but the communications data-frame is organized as: 1 start bit, 8 data bits NO parity and one stop bit (10-bits total)
9	User Mode. Same as Mode 6, but the communications data-frame is organised as: 1 start bit, 9 data bits NO parity and one stop bit (11-bits total)
10	I/O Box Mode (MD29AN only). This mode allows a single I/O Box to be connected directly to the MD29AN using the EIA RS485 port.
11	User Mode. Data bypasses the software FIFO buffer, reducing the delay in data being transmitted or received. The communications data-frame is organised as: 1 start bit, 9 data bits NO parity and one stop bit (11-bits total)
12	Reserved
13	Modbus RTU
14	Modbus ASCII

10.4 General-purpose parameters

In each Drive there are general-purpose parameters for use with MD29 programs.

Parameters marked as RO can be altered by the MD29, but they cannot be altered using the keypad of the Drive. R–W indicates that the parameter can be altered using the keypad.

The term EEPROM in the tables below refers to the parameter storage memory in the Drive, and is not related to memory in the MD29.

Mentor II Drive

Parameter numbers	Description	Type	Range
#15.01 to #15.05	Variable parameters	RO	±1999
#15.06 to #15.10	Variable parameters	R–W	±1999
#15.11 to #15.20	Integer parameters	R–W	0 to 255
#15.21 to #15.36	Bit parameters	R–W	0 or 1
#15.37 to #15 59	Integer parameters not stored in EEPROM	RO	0 to 255
#16.01 to #16.05	Variable parameters	RO	±1999
#16.06 to #16.10	Variable parameters	R–W	±1999
#16.11 to #16.20	Integer parameters	R–W	0 to 255
#16.21 to #16.36	Bit parameters	R–W	0 or 1
#16.37 to #16.63	Integer parameters not stored in EEPROM	RO	0 to 255

CDE Drive (0.75kW to 11kW)

Parameter numbers	Description	Type	Range
#9.03 to #9.06	Variable parameters	R–W	±999.9
#9.08 to #9.09	Variable parameters not stored in EEPROM	R–W	±999.9
#9.14 to #9.19	Integer parameters	R–W	0 to 255
#9.31 to #9.39	Bit parameters	R–W	0 or 1

CDE Drive (11kW to 75kW) and HPCDE

Parameter number	Description	Type	Range
16.01	R–W parameters (automatically saved when AC power is removed from the Drive)	R–W	±32767
16.02 to 16.05	Variable parameters not stored in EEPROM	R–W	±32767
16.06 to 16.20	Variable parameters	R–W	±32767
16.21 to 16.36	Bit parameters	R–W	0 or 1

Parameter number	Description	Type	Range
17.01	R–W parameters (automatically saved when AC power is removed from the Drive)	R–W	±32767
17.02 to 17.05	Variable parameters not stored in EEPROM	R–W	±32767
17.06 to 17.20	Variable parameters	R–W	±32767
17.21 to 17.36	Bit parameters	R–W	0 or 1

Vector Drive (11kW to 75kW)

Parameter number	Description	Type	Range
Pr63 to Pr69	Variable parameters	R–W	±6000
b48 to b55	Bit parameters	RO	0 or 1
b56 to b62	Bit parameters	R–W (RO using the MD29)	0 or 1