BALDOR®
MOTORS AND DRIVES

D-BSC
SERIES 1000 to 3000

SERVO CONTROL
FOR BRUSHLESS AC MOTOR
WITH
DIGITAL CONTROL PLATFORM
(DCP)

OPERATING MANUAL (How to use)

(Supports D3S Software)

IMN 1217

08/95
PN 22912

MANUAL MN 12..WG

BALDOR ASR GmbH has tried to ensure that the information given in this manual is correct at time of going to print.
However BALDOR ASR GmbH may not be held responsible for any typographical mistakes or errors in contents of this manual. The information is subject to change without prior notice.
BALDOR ASR GmbH would appreciate notice of any mistakes in the text.

Preliminary

Information contained in this manual is necessary for software setup of the DBSC 1000/1100 and 2000/3000 drives. Those drives are based on **DCP** (Digital Control Platform). For those who are using the D-BTS drive, please refer to the "**DSD** (Digital Servo Drive) How to Use Manual".

For the installation, the mechanical description, the wiring and hardware setup, refer to the "Installation and Instruction Manual" of the appropriate drive.

Also note that for DBSC 2000/3000 series, there is a third manual referring to the power supply: BPS 2000/3000.

Another document called the "**Host Protocol Communication**" is attached to this manual. This addendum explains how to program DBSC's commands from the host. This permits the user to create its own "Man-Machine" interface using programming languages such as "Basic", "C", "Pascal" etc. This is only necessary for the few people who need to send commands to the DBSC from their application program (without using the D3S software) and for the people who are using the IMAS option (see chapter 6).

BALDOR declines all responsibility for any damage caused by a none qualified personnel during the setup procedure or by any body who does not have full knowledge of the content of this manual as well as the DBSC and BPS hardware manuals. The directions of the setup procedure must be followed as described in the above mentioned manuals.

Table of contents

Table of contents (continued)

1. Introduction

This document is to be used for the DBSC with DCP (Digital Control Platform) drives equipped with the firmware version 8004.x to 8008x. Note that the "Inductive Absolute Modular System" (IMAS) feature which is added in Eprom version 8008.x is not yet fully described in this manual.

The DBSC (Digital Brushless Servo Control) will control an AC brushless motor with resolver feedback. It is a micro controller based on servo control platform and all servo tasks (current loop, velocity loop and position loop when hand wheel or pulse follower mode is active) are performed digitally. Other tasks (protection, communication, etc.) are also done digitally.

The DBSC initialization, setup and start-up procedure, is supported by a PC software called D3S (Digital Servo Setup Software) which is delivered with the amplifier. It performs various tasks, such as PLC programming, system evaluation, viewing parameters, displaying graphs (digital scope), operating mode definition, jogging, etc. The D3S software includes an on-line help that describes each function upon user request.

This document is a user guide for the operation of the D3S, it explains how to setup the system, configure the DBSC and tune the servo loops for any motor connected to the DBSC.

The working procedures are described step by step, and are illustrated by flow charts. Following these procedures will assist in rapidly setting up the system start-up.

1.1 Digital Servo Software Update

This manual is written for the D3S software version 1.0x. Note the modifications introduced in D3S version 1.05:

D3S for DCP version 1.05 dated 21.05.95 upgrade description

- New "Help" screens for "PLC programming", "Second Analog Command" and "Drive Errors".
- "DSD" name change to "DCP" (Type DCP to start D3S instead of DSD previously)
- "Gap" field in velocity controller screen is removed.
- Functionality in "System Definition": **<ESC>** cancels new values, **<Enter>** confirms new values
- Units changed in screen.
  Floating Point handler.
- PLC Jog parameters download: **<ESC>** cancels new values, **<Enter>** confirms new values.
- PLC programming screen redesigned.
- Downloading of Velocity and Position controllers does not affect control loop gains.
- In "Second Analog Command" setting, "Torque limit" is preceded by the following message: "Torque limit will disable power in velocity controller". (Should be "Enabling torque limit may result in no torque on the motor shaft if no voltage is seen on the **"CMD2+"** and **"CMD2-"** terminals)
- Bug with double press of **<F6>** and **<F5>** in various combinations fixed.
- Sequence **<F6>** - **<F5>** - **<F6>** (Hold Position - disable - Hold Position) is fixed: the system returns to "Hold Position".
- DSD_SW.EXE renamed DCP_SW.EXE.
- For DBSC 2000, 3000 drive over voltage increased:
  DBSC 2000: 420 V
  DBSC 3000: 840 V

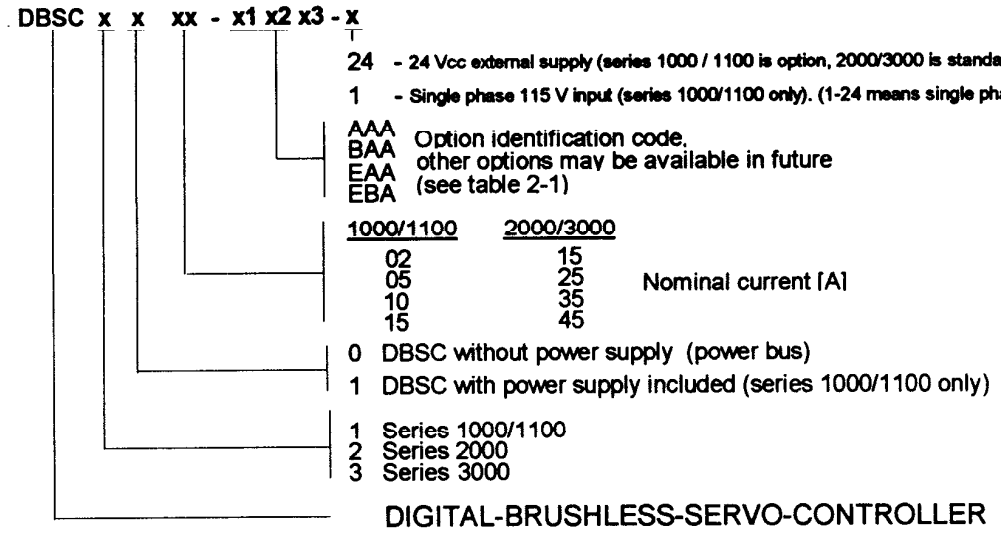
1.2 Operating Manual Update (Version 08/95)

This reprint includes the following changes:

- Page 1: Minor change
- Page 4: Added sections 1.1 and 1.2
- Page 5 and 6: Minor changes
- Page 9: Added note concerning maximum number of DBSC in daisy chain
- Page 10: Corrected and completed figure 3.2.
- Page 16, 44, 71: Minor change
- Page 72: Corrected figure C1 and table C1
- Page 76: Added faults correspondence with "7 segments" display.

2. Specifications

2.1 DBSC Identification Code



2.2 Option identification code

The DBSC option identification code is made of three characters (X1, X2, X3). The table below shows the available combinations:

option:	Option description	A	B	C	D	E
X1	RS 232	x	x			x
	Status display	x	x			x
	Binary encoder simulation	x	x			x
	Communication protocol	x	x			x
	12 bits ADC	x	x			x
	Power monitor (CPU reset)	x	x			x
	Connector X2 partially mounted (reduced to 5 pins)	x	x			
	X4, X5, and DIP switches installed (Daisy chaining possible)					x
	Connector X8 mounted		x			x
	Pulse and direction inputs (pulse follower)		x			
	Decimal encoder simulation		x			x
	Connection to optional extension board (internal)					x
	Second analog input command (torque limitation)					x
	Motor temperature sensor input (opto isolated)					x
	RS 485 (compatible RS 422, see section 3.6)					x
	Multi-axis synchronization					x
	4 none dedicated plus Fault reset inputs (opto-isolated)					x
	4 none dedicated outputs (opto-isolated)					x
	PLC available					x
	Encoder input for Hand wheel following					x
X2	IMAS option		x			
X3	CAN Bus		x		x	
	Positioning			x	x	

Table 2-1: option definition

Note:

On each DCP board, an identity number followed by an option number is printed on a label. For example: PRD - 00008000-20. The two last digits correspond to the option installed on the board. So far, the defined and produced options combinations correspondence are as follows:

05 = Same as EAA without decimal encoder, can be mounted only on DBSC 1000/1100
11 = Same as EAA without decimal encoder, can be mounted only on DBSC 2000/3000
20 = AAA
21 = BAA
22 = EAA
23 = Not applicable
24 = EBA

Those numbers are also shown on the display at power up and in the D3S software using the "Installed option" sub-menu. Note that the options 05 and 11 are superseded by the 22 = EAA option.

2.3 Specifications

2.3.1 Digital Control Platform Specifications (DCP card)

General	Units	
Microprocessor	--	NSC HCP 46100 (16 bits with DSP kernel)
RAM	K. Byte	32 (12 are used for data gathering)
EEPROM	Byte	512 (identification module)
Velocity / Current command input	V_{dc}	0...±10, 12 bits (16 bits optional)
Offset	--	Adjustable to zero (automatically or manually)
Offset range	Vdc	0...±10
Offset resolution	mV	1

Current loop	Units	
Servo cycle time	µs	117.6
Band width	Hz	1000

Velocity loop	Units	
Servo cycle time	µs	470.4
Band width	Hz	10..200 (adjustable)
Velocity feedback resolution	bits	16 (Max. Velocity 1500 RPM) 14 (Max. velocity 6000 RPM) 12 (Above 6000 RPM)

positioning loop (Hand wheel or Pulse follower)	Units	
Servo cycle time	µs	470.4
Operating mode 1) Encoder input signals 2) Pulse and direction signals	--	**Options 1 or 2 has to be ordered** (only one possible) Differential RS422; A, A/, B, B/, C, C/ Diff. RS422; A is pulse, B is direction, C is not used
Input max. frequency	KHz	500 (before quadrature)
Electronic gear ratio	--	+/- 0.01..100 (adjustable, floating point)

Resolver - Encoder simulation	Units	
Resolver to Digital Converter (RDC) resolution	bits	16 (Max. velocity 1500 RPM.) 14 (Max. velocity 6000 RPM.) 12 (above 6000 RPM)
Resolver Winding ratio	--	0.5
Resolver pole pairs	--	1
Encoder simulation	ppr	512 / 1024 / 2048 / 4096 (standard) 500 / 1000 / 1250 / 2500 (optional)* (Angular resolutions optional)*
Signals	--	5V TTL; A, A/, B, B/, C, C/
Reference pulse	--	Available, non-adjustable

Interface	Units	
Digital I/O		20..29 Vdc (opto-isolated)
Communication		RS 232 or RS422 or RS 485 (option dependent)
Bit rate	Baud	9600 (non-adjustable)

2.3.2 Environmental

	Units	
Operating temperature range	°C	0 to +40
Storage temperature range	°C	-25 to +70
Humidity	%	10..90; not-condensing; according to DIN 40 040, class F
Shock		10 G; according to DIN IEC 68-2-6/29
Vibration		1G; 10..150 Hz; according to DIN IEC 68-2-6/29

3. Installation and wiring

3.1 Mechanical installation

Refer to the "Installation and Instruction Manual".

3.2 Electrical installation

Refer to the "Installation and Instruction Manual".

3.3 Wiring

Refer to the "Installation and Instruction Manual".

Remark: The "At home" input (X2-6) and "Capture" input (X2-7) are not available for the moment. They are planned for future options.

3.4 Preset of the DIP switches AS1-8 *(Mounted only if this option was ordered)*

When the DBSC is ordered with the option Exx, the DIP switch is installed. The description of each switch is given in the appendix B of this manual.
Also refer to the "Installation and Instruction Manual".

3.5 Communication and D3S software installation

The serial communication port (RS232, RS422 or RS485) is used for initialization, setup via the PC software D3S (Digital Servo Setup Software), and continuous data transfer between the user host and the amplifier.

The connectors are DB9 female type on the PC side and male on the DBSC side. Refer to appendix C for the description of the communication cable and of the minimum PC requirements for the use of the D3S software.

The communication Baud rate is set to 9600 and is not adjustable.

- Connect the communication cable RS232, RS422 or RS485 (depending of the installed option) to the host computer and to the DBSC.
- Install the D3S software:
  1. Insert the D3S diskette into the PC floppy disk drive A (or B).
  2. Move to the root of your main drive: (Type: **cd** and press **<Enter>**)
  3. Make a directory for the D3S: (For example, for the D3S version 1.03 Type: **md DCP103** and press **<Enter>**)
  4. Move to the DCP103 directory: (Type: cd DCP103 and press **<Enter>**)
  5. Copy the D3S software from the diskette: (Type: **copy A:*.*** and press **<Enter>**)
  6. Open the file called "READ.ME" and read the information about the actual D3S software version and DCP firmware versions.

At this point, the installation procedure has been completed, you are now ready for the start-up procedure.

3.6 Multi-axis applications (Daisy chaining or Multi-drop)

Depending of the installed option, the DBSC is equipped with a RS232 (without accessibility to daisy chain), or with RS232 and RS485 (with accessibility to daisy chain and multi-drop). For example:

- Option AAA and BAA: RS232 only, **daisy chain is not possible.**
- Option EAA and EBA: RS232 and RS485. daisy chain and multi-drop are both possible

Remarks:

- If the DBSC is configured for RS232 via its internal jumpers, then only the daisy chain is available for multi-axis applications (no multi-drop is possible).
- The internal hardware design of the DCP permits the RS485 to be compatible with RS422. Therefore, daisy chain operation is possible. It is done over the RS485 line driver but data transmission is operated in the RS422 mode.
- RS485, RS422 and RS232 cannot be operated simultaneously. It is necessary to configure the hardware according to one of he choice in table 3.1.

Definitions:

- ***Daisy chain:*** This configuration is obtained by connecting the PC to connector X6 of only one DBSC. Then the remainer of DBSC's in the daisy chain must be linked via connector X5. Also jumpers SB501 to 510, solder bridge LB304 as well as DIP switches AS1 to AS4 must be set in accordance with the desired configuration. Note that daisy chain is possible for RS232 and RS422 only. Refer to figure 3.1, and table 3.1 for wiring principle, and also to paragraphs 6.1.2 and 4.3 in the "Installation and Instruction Manual".
- ***Multi-drop:*** This configuration is obtained by making a parallel connection of the PC to connector X6 of each DBSC. Connector X5 is **not** used. Also jumpers SB501 to 510, solder bridge LB304 as well as DIP switches AS1 to AS4 must be set in accordance with the desired configuration. Note that multi-drop is possible for RS485 only. Refer to figure 3.2, and table 3.1 for wiring priciple, and also to paragraphs 6.1.2 and 4.3 in the "Installation and Instruction Manual".

Mode of Data Transmission	DIP Switches AS1 to AS4	Jumpers SB501 to SB510	Solder bridge LB304	Remarks
RS232 single	All OFF	Jumper pins 1-2	Do not care	Factory default (LB304 is open)
RS232 Daisy chain	Set for desired address	Jumper pins 1-2	Do not care	
RS422 Single	All OFF	Jumper pins 2-3	Do not care	
RS422 Daisy chain	Set for desired address	Jumpers pin 2-3	**Open**	
RS485 single	All OFF	Jumpers pin 2-3	Do not care	
RS485 Multi-drop	Set for desired address	Jumpers pin 2-3	**Closed**	

Table 3.1: Serial data transmission possible configurations

Note: Although the DBSC and the D3S have an addressing capability of 16 cards, it is not possible to daisy chain more than 8 cards. This is due to a current consumption limitation of the daisy chain. If more than 8 DBSC's must be daisy chained, contact the factory for advice, or use the multi-drop configuration.

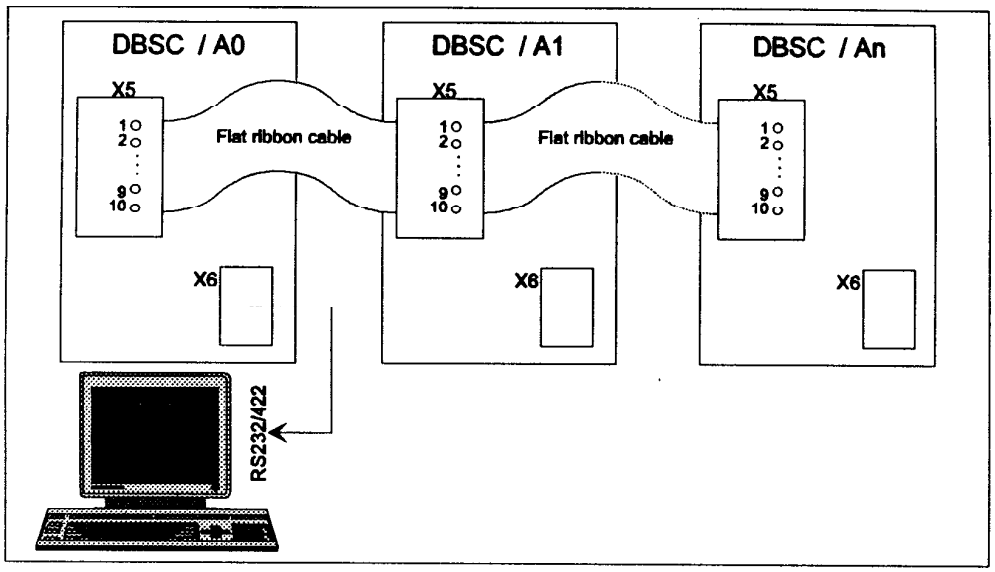


Figure 3.1: Wiring principle for RS232 and RS422 daisy chain

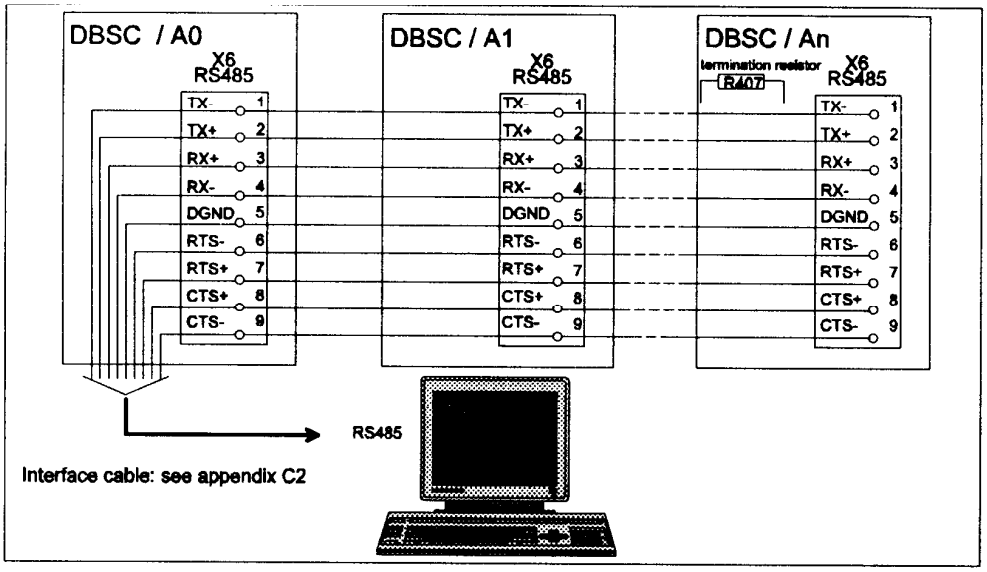


Figure 3.2: Wiring principle for RS485 multi-drop

4. Start-up procedure

4.1 Introduction

All of the steps of the start-up procedure must be followed to avoid all domage.

It is strongly recommended that the motor is disconnected from the load for the preliminary start-up procedures. The load will be reconnected for the final stage of the procedure.

Note that dangerous voltage can remain on the power connector for about 5 minutes after disconnecting and turning off power.

The first and most important step of the start-up procedure is system definition. Parameters of major importance such as those for the velocity and current loops, pole pair number etc. are based on this definition. Therefor, it is essential to define the system configuration very accurately. This chapter describes how to configure a DBSC for first use with the default values (fast configuration procedure). Before configuring the DBSC, the following must first be accomplished:

❶ Select the amplifier address (see section 4.4)
❷ Check which options are installed. (see sections 4.2 and 4.5).

There are three different ways to configure a DBSC:

- Using the D3S software, duplicating an existing configuration (see section 4.6).
- Using the D3S software, downloading the default parameters (see section 4.7).
- Duplicating the Eeprom device (see section 4.8).

Note: New versions and updates

Each D3S software version, has to correlate with the firmware (contained in the Eprom, IC 203) installed. The D3S version is only compatible with older firmware versions. The firmware version is indicated by a number and the update by the letter (example: 8004A, B, ...). New firmware updates (letter) indicates minor improvements, the compatibility is guarantied. New versions (number) have functional additions for new options. Changing to a firmware version higher number will not upgrade the amplifier functionality as other internal hardware options have to be installed ex-factory. Information about the D3S software version can be read from the **"About"** menu. Note that the firmware version number given in that menu is the highest version that is fully compatible with the D3S in use. The firmware version which is installed on the board is written on the Eprom (IC 203).

After a firmware update (Eprom version changed), the DBSC configuration held in the Eeprom may not be valid any longer. Therefore, a **"U"** indicating the version mismatch will appear on the DBSC display after power-up, the Eeprom content is discarded and the drive disabled. It is then necessary to reconfigure the drive via the D3S software. The amplifier can only be enabled after the downloading of the new configuration.

4.2 Power-up procedure and installed options checking

4.2.1 Power voltage check

It is highly recommend that the motor cable be disconnected for the following:

- Disable the DBSC: open the contact wired to X2 pin 2.
- Depending on the input power connections (with or without transformer), disconnect either the transformer's secondary or disconnect directly the main power.
- Power-up, measure and verify voltage on the transformer's secondary, or the main power if there isno transformer. It must not exceed:
  250 VAC for DBSC 1000/1100
  264 VAC for BPS 2000
  528 VAC for BPS 3000.
- Turn power off and rewire the power cable.
- If DBSC has the 24V option, disconnect the 24V input of the drive and power-up the auxiliary power supply.
  - Check that the auxiliary power supply output voltage is between:
    20 and 60 VDC for DBSC 1000/1100.
    20 and 29 VDC for DBSC 2000/3000.
  - Turn off the auxiliary power supply and reconnect it to the drive.

4.2.2 Verification of installed options and power-up

IMPORTANT: The amplifier must be disabled for its first power-up in all of the following cases:

- **When the DBSC has not yet been configured or installed.**
- **When modifications have been done on the system such as: installing a new motor type or the load has been altered.**
- **When the analog input voltage is not null.**
- **When there is the slightest doubt about the initialization or the system state.**

Verify that the installed options are in accordance with what was ordered. The code is in the 3 last letters of the catalog number (CAT.) which is printed on the identification label (Example: DBSC 1005-**AAA**). A two digits number associated to these letters is also printed at the end of the DCP identification word inside the drive (Example: PRD-00008000-**20**). It is also hardware burned into IC213. This number is shown on the 7 segment display upon power up (see below), and shown on the computer's screen (see chapter 4.5).

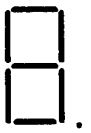
For example:

Last 3 characters of the drive type			2 digits number (Displayed at power-up)
A	A	A	**20**
B	A	A	**21**
E	A	A	**22**
E	B	A	**24**

The options definition and description is given in table 2.1, section 2.2.

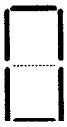
Power up the auxiliary 24 V supply (if auxiliary power supply is used). Then apply main power and Watch the 7 segment display during the DBSC initialization. The following sequence will occur during normal operation:

❶ All 7 segments and the decimal point are shown for a short time:



❷ The option number is displayed (the two consecutive characters, as explained above). The left digit (most significant) is displayed first and then the right digit (less significant). For example, for option number **20**, the left digit is:

and the right digit is:



❸ After the initialization procedure is completed, the character displayed shows the actual status (and error message) of the DBSC. It is constantly updated.

Refer to tables D1 to D3 in appendix D for status and errors description on the 7 segment display.

❹ If the DBSC is not yet configured for the motor and load, connect a PC (RS232, RS422 or RS485), activate the D3S software and perform the instructions following in this chapter (also see chapter 5).

Notes:

1. At first power-up, the DBSC should display a **"d"** indicating it is disabled; should a decimal point be shown, the unit is enabled.
2. If a number (1 to 7) is displayed, turn power off and check the type of error (tables D1 to D3 in appendix D). Also look at the troubleshooting section in appendix E and correct as needed.
3. If the green LED is not "on", check the presence of the main power supply (or the auxiliary power supply if the DBSC has this option).

Check that the DC bus voltage is between 50 and 350 VDC for DBSC 1000/1100, between 0 and 350 VDC for DBSC 2000 and between 0 and 740 VDC for DBSC 3000. The DC bus voltage depends on the AC input power voltage and is calculated as follows:

$$Vdc = Vac \cdot \sqrt{2}$$

Note: VAC = Voltage between two phases of the power input terminals on the DBSC 1100 or BPS 2000/3000.

Turn power off.

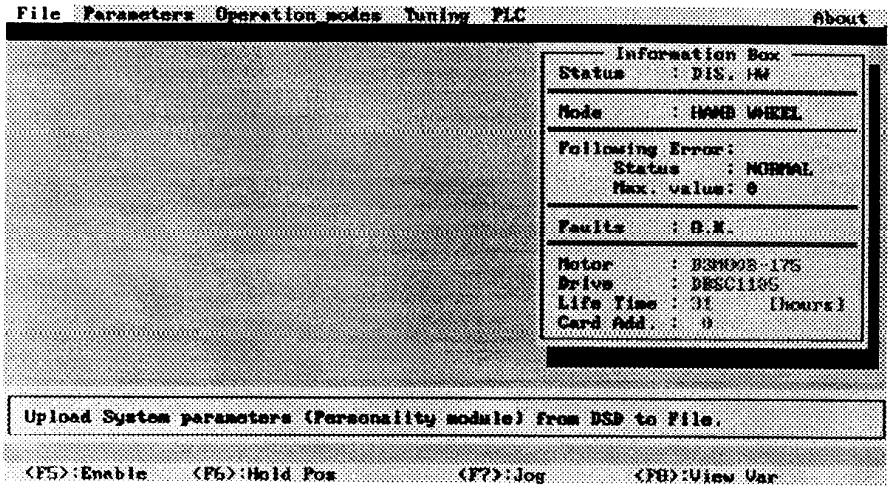
Warning ! Wait about 5 minutes after turning off the power supply. This to allow the discharge of the capacitors inside the main power supply. Do not touch the power bus line and terminals during this period of time.

4.3 D3S software description

Once all the basic checks are completed, link the DBSC to the computer via the RS232, RS422 or RS485. Verify that the DBSC is disabled (Connector X2, pin 2 opened, or DIP switch AS8 in the "OFF" position) Turn the power on and activate the D3S program:

❶ Move to the D3S directory. (Type: cd\DCP103 and press **<Enter>**)
❷ Activate the program. (Type **DCP** and press **<Enter>**)

The screen as shown below should be displayed:



At the top of the screen, is the **main menu**. It includes the following:

- **Files**
- **Parameters**
- **Operation_modes**
- **Tuning**
- **PLC**
- **About**

At the bottom of the screen, the toggle **hot keys** are listed. These offer quick execution of the most common used commands:

- **<F5>** Enable ⇔ Disable
- **<F6>** Hold Pos. ⇔ Release Hold Pos.
- **<F7>** Jog ⇔ Stop Jog
- **<F8>** View Var. ⇔ Release View Var.

Other hot keys which are often used are:

- **<F10>** Command execution
- **<F1>** Help

The function, on which the cursor is placed, is described in the line just above the hot keys. This is the **"Function description line"**.

The **Information box** is shown at the right side of the screen. It displays, in real time, the most important DBSC configuration and status information such as:

- **Status** ENA, DIS HW, DIS SW, etc. (See table D-3 in the appendix D2)
  The Status line displays the drive state at all time.
- **Mode** CURRENT, VELOCITY, HAND WHEEL
- **Faults** O.K., WARNING, ERROR.(see faults detailed description in the PLC menu. See also table D-3 in appendix D2)
- **Motor** Motor Model Number (ex: BSM80B-175)
- **Drive** Amplifier Model Number (ex: DBSC1105)
- **Life Time** Clock counting the operation time, in hours, from the first time the drive is powered-up. It stops counting while the power is down, but continues from the last value when the power comes up again.
- **Card Address** Daisy chain or multi-drop card currently addressed (0 to 15)

When the Hand-Wheel mode is enabled, two additional data are displayed:

- **Following Error Status** IN POS. / NORMAL / WARNING / FATAL
- **Max following error** **In units 1/4096 bits per revolution**

The indications shown in the **Information Box** are constantly updated (in real time).

To select an option from the menu and sub-menus, follow these steps:

❶ Use the arrow keys to locate the cursor on the desired option and press **<Enter>**:
→ (right)
← (left)
↑ (up)
↓ (down)
❷ The window which corresponds to the selected option is displayed.
❸ Specify the parameters or change them as required.
❹ Press **<F10>** to download the specified parameters to the system.

Remarks:

- Press **<ESC>** to quit from the current window without updating.
- Press **<Enter>** or ↓ to open the window which corresponds to a specific option in the main menu, when the cursor is located on the desired option.

In the following sections the D3S software is described through the operating procedures. The operating procedures and explanations are often illustrated by screens.

4.4 Amplifier address selection

The D3S software uses the address 0000 as default. Therefore, if only one DBSC is connected, it is recommended to select the DIP switches position AS1 to AS4 for the address 0000. If more than one DBSC is connected, select the desired address (see table B-1 in appendix B).

When several DBSC's are daisy chained or in multi-drop configuration, it is necessary to address the specific amplifier to which the communication is to be established. It is **imperative that the DIP switches AS1 to AS4 on each amplifier are configured at a different address.** (Refer to table B-1 in appendix B.)

The DBSC address software selection is done as follows:

❶ Select the **"File"** sub-menu and press **<Enter>**.
❷ Choose the **"Select Card"** option and press **<Enter>**.
All the DBSC's addresses which are connected in daisy chain or in the multi-drop configuration, are displayed.
❸ Move to the desired address.
❹ Press **<F10>** to select the specified address.

When a new address is selected, the parameters in the Information box are updated, in accordance to the selected drive.

Important reminder!

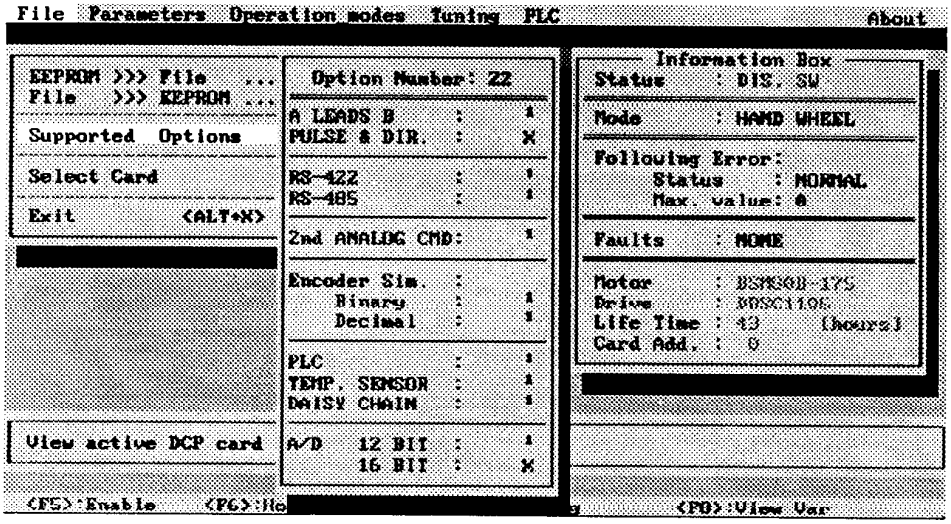
It is imperative that the DIP switches AS1 to AS4 on each daisy chained amplifier, or in multi-drop mode, are configured at a different address (Refer to table B-1 in appendix B). If this condition is not met, the communication data becomes unpredictable, this can be dangerous .

4.5 Supported options check

The operation which is described in this section is recommended for the first configuration procedure in order to verify if the installed options are the same as those ordered. This is an easy way to check the options which are present in the drive.

❶ Select the **"File"** sub-menu and press **<Enter>**.
❷ Choose the **"Supported Options"** option and press **<Enter>**.

The window with the supported options is displayed:



The symbol "1" indicates that the specific option is supported. The symbol **"X"** indicates that the specific option is not supported. The actual options, which are read directly from the DBSC, are displayed. In this example, the option number is **22**, which corresponds with the EAA word printed on the drive label after the product type. Make sure that your supported options and the **Option Number** correspond with what was ordered.

4.6 Duplication of an existing configuration (memorized on the PC hard disk)

When there are several axis composed of the same amplifier type and motor type, as well as the same load, it is possible to download to the DBSC Eeprom a known configuration which is saved on the hard disk of the PC. Once installed, the offset tuning procedure also needs to be executed (see section 5.2).
Note that, if option Exx is present, the PLC program is included in this configuration file.
The configuration duplication procedure is described bellow:

❶ Disable the drive - Press the **<F5>** key, or open the "Enable" input switch, wired in to connector X2 pin 2, or move DIP switch AS8 in the "OFF" position.
❷ Select the **"File"** sub-menu and press **<Enter>**.
❸ Choose the **"File**))) **EEPROM"** option and press **<Enter>**.
❹ Press the **"?"** key, all the files available are displayed (*.bin).
❺ Specify, or select, the file name **(*.bin)** from which the DBSC configuration should be copied.
❻ Press **<F10>** to download the personality module from the disk to the DBSC Eeprom.
❼ Enable the amplifier - Press the **<F5>** key (or use the "Enable" input, or the DIP switch AS8).

Remarks:

- **The selected file and the D3S program have to be located in the same directory.**
- The data in the file must have been created by the same version of D3S program that is presently used. **The data must also be cpoied from a DBSC that is equipped with the same Eprom version and with the same options as the one being in use.**
- After downloading, the information box always displays a "software disabled" status. Press **<F5>** to enable the DBSC. If it stays disabled, activate the hardware input X2-2 or the switch AS8 (Hardware enable). (The "Information box" indicates in priority the software disable.)

4.7 System configuration with default values

The **"Parameters"** sub-menu in the D3S program is used for this operation. Four different settings are described in this section:

1. General configuration (always needed).
2. Hand-Wheel configuration (only if the hand wheel or pulse follower option is installed and used).
3. Multi-resolver configuration (only if the multi-resolver option is installed and used).
4. User defined drive type.

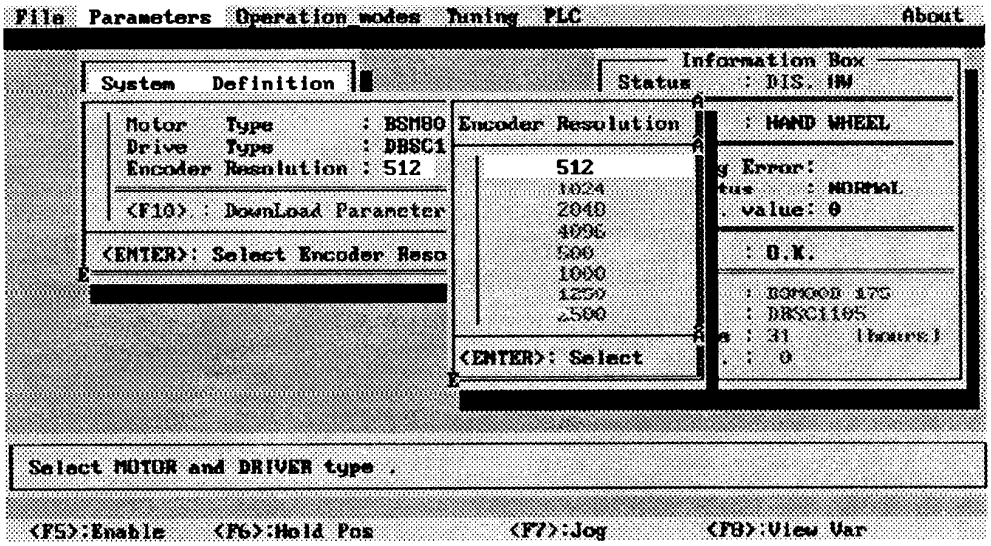
The procedures are explained in the paragraphs 4.7.1 to 4.7.4 bellow.

4.7.1 General system configuration

❶ Select the **"Parameters"** sub-menu and press **<Enter>**.
❷ Select the **"System Definition"** option and press **<Enter>**.
❸ Locate the cursor on the **Motor Type** and press **<Enter>**.
The list of the catalog numbers of BALDOR brushless motors families is displayed.
❹ Find your motor family in the list, locate the cursor on it, and press **<Enter>**.
A sub-list of motors is displayed.
❺ Locate the cursor on the desired motor and press **<Enter>**.
The motor parameters are displayed.
Press **<Enter>** once again.
❻ Locate the cursor on the **"Drive Type"** and press **<Enter>**.
The BALDOR drives families list is displayed.
❼ Find your drive family (DBSC) in the list, locate the cursor on it, and press **<Enter>**.
DBSC drive list is displayed.
❽ Locate the cursor on the desired drive (For the DBSC 1000/1100, the nominal current selection DIP switch setting located at the back of the drive has to be taken into account) and press **<Enter>**.
The amplifier parameters are displayed.
Press **<Enter>** once again.
❾ Locate the cursor on the "Encoder Resolution" and press **<Enter>**.
The encoder resolutions list is displayed.
Select the desired resolution and press **<Enter>**.
❿ Press **<F10>** to download the parameters to the DBSC (personality module).

Upon motor and drive selection, the motor and drive parameters are displayed on the screen.

The following screen is an example of default DBSC configuration:



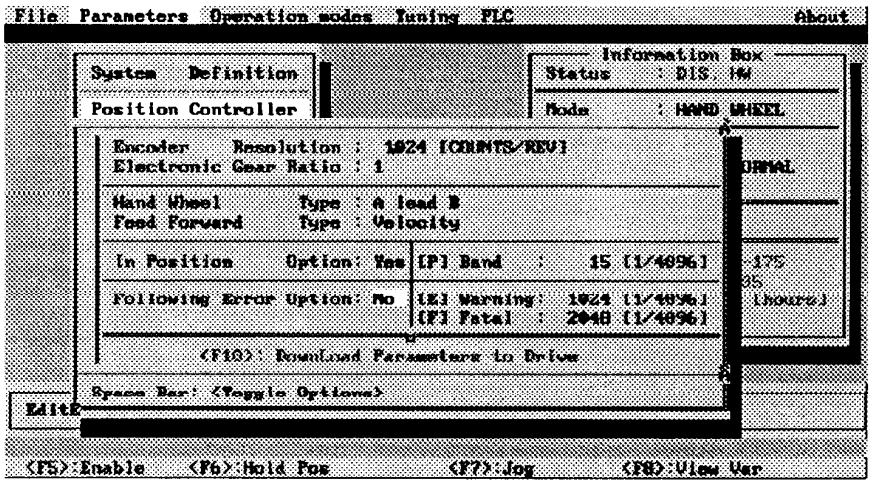
Remarks:

- In the **"Drive Type"**, you may not be able to select the **DBSC1000** and **1100** or the **DBSC2000** and **3000**. This is because the Digital Control Platform (DCP) is able to read which type of power stage is connected to it. It is therefor impossible to select, by mistake, a DBSC1000/1100 series instead of a DBSC2000/3000 series and vice versa.
- In the **"Motor Type"** and **"Drive Type"** options, the motors and the amplifiers are shown according to their catalog number. This library will be updated for each new motor and amplifier type that BALDOR will produce. The user can read and compare the parameters values with those which are printed in the catalog. They should be identical.
- If the motor and/or drive specifications are not found in the library, their parameters should be defined, based on their data sheets, by using the **"User Defined"** option. The activation of this option displays a list of motor (or amplifier) models which are all called "USER DEFINED". Those are only default names wich can be changed to any motor (or amplifier) model that does not exist in the BALDOR list. For example if the motor type BSM100-150-AA can not be found in the list, you can update it by using the user defined option.
- Motor Resistance and Inductance must be given for L-L (Line to Line or value between phases).
- The given Motor Torque Constant, Kt, is defined as the equivalent global motor torque constant (not per phase).
- The **"Encoder resolution"** option provides the choice between 8 different resolutions (4 decimal and 4 binary resolutions, or only 4 binary resolutions depending on the installed options) The selected resolution will provide the encoder simulation signal output for a CNC or any external position control card. The selected resolution can be multiplied by 1, 2 or 4 by the axis card. A, A/, B, B/, C and C/ channels are provided.

4.7.2 Hand Wheel configuration

When the Hand Wheel option (Exx), or pulse follower option (Bxx), is installed and used, it is necessary to configure few parameters for the position loop control. In this case follow the instructions below:

❶ Select the **"Parameters"** sub-menu and press **<Enter>**.
❷ Select the **"Position Controller"** option and press **<Enter>**.
❸ Choose the required position parameters as in the following example:



Press the space bar to select the desired **"Hand Wheel Type"** option:

- **"Pulse and direction"** (channel A is pulse, B is direction, C is not used). The pulse follower option has to be installed.
- **"A leads B"** (The Hand Wheel input uses the A, A/, B, B/ channels of encoder. C, C/ signals are not used).

Remark: The **Electronic Gear Ratio** may be a negative number, which is equivalent to **Hand-Wheel Type** of **B leads A**.

❹ Press **<F10>** to download the configuration to the Eeprom device.

Remark:
When the "Hand Wheel" mode is desired, it is recommended to set to zero (or a very low value) the **"Time to Max. Velocity"** parameter which is accessible in the **"parameters" / "Velocity Controller"** sub-menus. This precaution will allow to avoid motor overshoots on strong move variations of the master encoder. (Refer-to the section 5.3.2 for more information.)

4.7.3 Multi-resolver configuration (IMAS option)

IMAS is an absolute position sensor system made of several resolvers which are all linked between them by a geared transmission with a gear ratio of 1/16 between each. Depending on the number of resolvers, it is possible to get the absolute position over a range that can go up to 65536 revolutions. When the **IMAS** option is installed (option xBx for the drive and AI, or AJ, or AK for the motor), it is necessary to set-up the system for the absolute position reading. Three major points must be considered:

1. Update the DBSC with the option (AI, AJ, AK) that is fitted to the motor. This is the number of resolvers mounted in the motor. This should be done only once at configuration stage.)
2. An absolute position "*calibration*" set-up procedure must be done after the motor coupling has been tightened to the machine. This is to create a correspondence between the "*IMAS Absolute Value*" and the "*Absolute Actual Machine position*". This should be done only for the first machine set-up.
3. The "*Absolute Actual Machine Position*" reading procedure. This will be done at every power-up, then the host has to read this value, scale it, and download it to the position register of the CNC or axis controller board.

A detailed explanation of IMAS system, initialization, absolute position reading and correlation will be given in chapter 6. Before considering the above mentioned procedures, it is recommended to tune the system as described in chapter 5.

4.7.4 User defined drive Type:

This is mainly used when the current switch selectors at the back of a DBSC 1000/1100 are set for a current that does not appear in the "Drive type" list. Selecting the "USER DEFINED" option will open a window where the following parameters have to be entered:

- DRIVE TYPE: Enter a suitable name (For example: **BSC 1103.75** for 3.75 Amperes)
- Bus Voltage (DC): Enter 300 Volts for the DBSC 1000/1100 series or other if special case.
- Peak Current: Normally twice the nominal current. If you have a DBSC 5 Amperes and the switches at the back are set for 75% (or 3.75 Amperes), enter 7.5 Amperes or less if wished.
- I2t Limit: The range is 1.5 to 3.0 seconds. This is the time during which the peak current can be produced before the temperature warning appears (Error 7 on the display). This time will vary according to the current level above the nominal value.
- Bus Over-Voltage: This value must be set for the DBSC 2000/3000 series. It is meaningless for the DBSC 1000/1100.

Once all values are entered, press the **<Esc>** key and then the **<F10>** key to download the parameters to the drive.

4.8 Configuration duplication by Eeprom copy

Most of the time, the drive configuration is defined and downloaded to the DBSC via the RS 232, RS 422 or RS 485 communication link by using the D3S software. This configuration is saved in the DBSC Eeprom device.
When several axis have the same characteristics (same amplifier type, same motor type and same load), the configuration may only need to be done once for all axis as it is possible to copy an Eeprom content to other Eeproms which will be mounted afterwards on the DCP board. The procedure bellow describes this operation:

1. Turn the power off, disconnect a drive that has been already configured, remove the amplifier front plate and unplug the DCP board (control card) of the DBSC.
2. Unplug the Eeprom device (IC301 - This 8 pins IC is socket mounted, its type is: 93C46. Refer to DCP layout drawing in appendix F.)
3. Use a Prom programmer to read the data of the EEPROM device. Refer to the PROM programmer manual.
4. Unplug any desired number of Eeprom devices from drives that should be configured.
5. Use the Prom programmer to program the Eeprom devices with the loaded data.
6. Plug the already programmed devices into the drives and install the amplifiers in their cabinet.

4.9 Motor activation and save configuration to file

At this point, the system setup is completed, and the DBSC is ready to work. After the execution of the points A to C described bellow, we will be able to connect the load to the motor in order to start the system tuning (chapter 5). For the moment, select the operation mode (refer to section 8.5), then activate the motor and, if desired, save the configuration on the hard disk according to the procedure bellow:

A. Operation mode selection

1. Select the **Operation_modes** sub-menu and press **<Enter>**.
2. Select the **Control mode** option and press **<Enter>**.
3. Choose the mode of operation: **"Current"**, **"Velocity"** or **"Hand Wheel"**.
4. Press **<F10>** to download your selection.

B. Enable the system (one or several of the points described bellow are necessary)

1. Close the enable switch wired on the connector X2, pin 2.
2. Set the DIP switch AS8 in the "ON" Position (DIP switches AS1 to AS8 may not be installed depending on which option was ordered)
3. Press the **<F5>** key to enable the drive.

C. Save the configuration to disk file (optional)

1. Select the **"File"** sub-menu and press **<Enter>**.
2. Disable the drive - Press the **<F5>** key, or open the "Enable" input switch (wired in to X2-2), or move DIP switch AS8 in the "OFF" position.
3. Choose the **"EEPROM ⟩⟩⟩ File"** option and press **<Enter>**.
4. Specify the file name (***.bin**) to which the configuration should be copied, or press "?" for a list of files.
5. Press **<F10>** to save the personality module data in the disk file.
6. Enable the amplifier - Press the **<F5>** key, or use the "Enable" input, or the DIP switch AS8.

D. Exit the "D3S" software

1. Press **<Alt+X>** while in the main menu, or select the **"File"** sub-menu and press **<Enter>**.
2. Choose the **"Exit"** option and press **<Enter>**.

Remarks:

- It is necessary to disable the DBSC in order to proceed with the Eeprom reading for the save file to disk operation. This has been implemented to avoid problems in PLC operations.
- Make sure to Enable the system by software (**<F5>** key) prior to exit the D3S. Even if all the hardware inputs are in the "enable" position, the system is kept disable if the software enable is not activated (watch the **Status** in the **Information Box**).
- If the current mode is selected, the velocity and position loop control must be included in the CNC, or the external motion control card or the host.
  If the velocity mode is selected, only the position loop control will be included in the CNC, or the motion control card or the host (the "derivative" gain of the axis card must be set to zero).

5. System tuning

5.1 Introduction

There are two main purposes for the system tuning:

- Protection of the electromechanical system (by defining system limitations).
- Good performance (static and dynamic behavior) of the system (by defining the **controllers parameters**).

The following points are discussed in this chapter:

- Offset tuning.
- Limitations within the servo loops (current, velocity, position).
- Servo loops tuning (velocity, position).
- Advanced tuning.

5.2 Offset tuning

The first step, after system setup, is the offset tuning of the analog reference. Several factors such as CNC system, motion control card, input filters, A/D converters, can be the source of an offset of the analog input. This may cause a rotation of the motor, although the CNC (motion control card) command output is null. This situation is not desirable especially when the amplifier is used in velocity mode.

The offset tuning may be performed in three ways:
A) automatically, via hardware (using the DIP switch AS7),
B) automatically, using the D3S software,
C) manually, using the D3S software.

When the personality module has been pre-programmed for your application, the only needed operation for the system setup is the offset tuning. This is because the offset varies from system to system.

A. Hardware automatic tuning (DIP switch AS7)

1. Disable the drive - Open the "enable" input switch (wired in the connector X2, pin 2), or move DIP switch AS8 to OFF position.
2. Make sure that the analog command (the DAC output from the host, CNC or position control) is precisely zero and that the CNC or positioning card is in open loop.
3. Activate the automatic offset tuning procedure - Move switch AS7 of the DIP switch to "**ON**". At this point, the tuning is executed, the analog command input must stay at zero and the amplifier disabled for several seconds. This permits the microprocessor to read the analog input and to calculate the average offset value for compensation.
4. After several seconds, move the switch back to "**OFF**" position.

> IMPORTANT:
>
> It is recommended to execute this procedure only once, at the beginning of the system setup.
> When DIP switch AS7 is permanently "ON", the offset tuning is executed at each power-up and every time that the drive is disabled. Therefore, if you choose to leave switch AS7 to "ON", you must make sure that the analog command is precisely zero prior to disable the drive or to turn the power up

B. Software automatic tuning (using the D3S software)

1. Disable the drive - Press **<F5>** key, or open the "enable" input switch wired in the connector X2, pin 2, or move DIP switch AS8 to OFF position.
2. Select the "**Tuning**" sub-menu and press **<Enter>**.
3. Select the "**offset tuning**" option and press **<Enter>**.
4. Select the "**Auto**" option and press **<Enter>**.
5. Make sure that the analog command is zero (if the DBSC is controlled by a CNC or a motion control card, open the position loop) and press **<Enter>**.

At the end of the automatic procedure, the value of the offset is displayed in millivolts.

C. Software manual offset tuning (using the D3S software)

The manual offset compensation is done as described bellow. In that procedure, the user can modify the offset value printed on the screen. The resolution is of the millivolt.

1. Make sure that the analog command (the DAC output from the host, CNC or position control) is precisely zero and that the CNC or positioning card is in open loop.
2. Select the **"Tuning"** sub-menu and press **<Enter>**.
3. Select the **"offset tuning"** option and press **<Enter>**.
4. Select the **"Manual"** option and press **<Enter>**.
5. Enable the amplifier - Press the **<F5>** key, or use the "Enable" input, or the DIP switch AS8. The motor may turn due to the offset, make sure that the machine position limits are never reached.
6. The actual offset value is displayed. If the motor moves CW (CCW), write a lower (higher) offset value.
7. Press **<Enter>** to download the new offset.
8. Repeat steps 6-7 until the motor remains in position. You may watch the position display by the **"View Variables"**, to fine tune the offset.

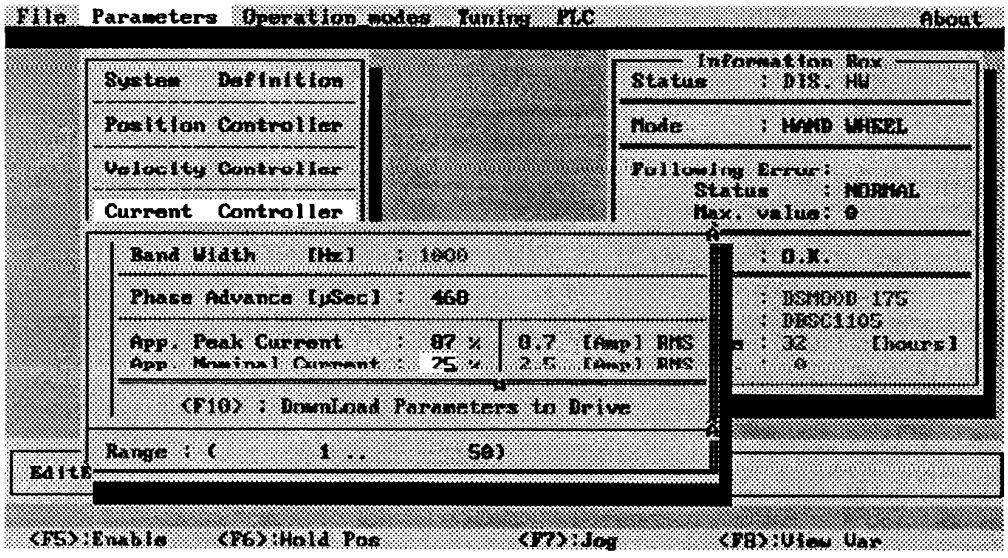
5.3 System limitations (current, velocity, acceleration, following errors)

This section explains the different limitations which are accessible by the user. They permit to protect the electromechanical system against overload situations.

5.3.1 Current Limitations

The limited peak and nominal current values can be changed as follows:

1. Select the **"Parameters"** sub-menu and press **<Enter>**.
2. Choose the **"Current Controller"** option and press **<Enter>**, the current loop parameters are displayed.
3. Change the default **Current Levels** (**Peak Current** and **Nominal Current**) as required:



4. Press the **<F10>** key to download the parameters to the drive.

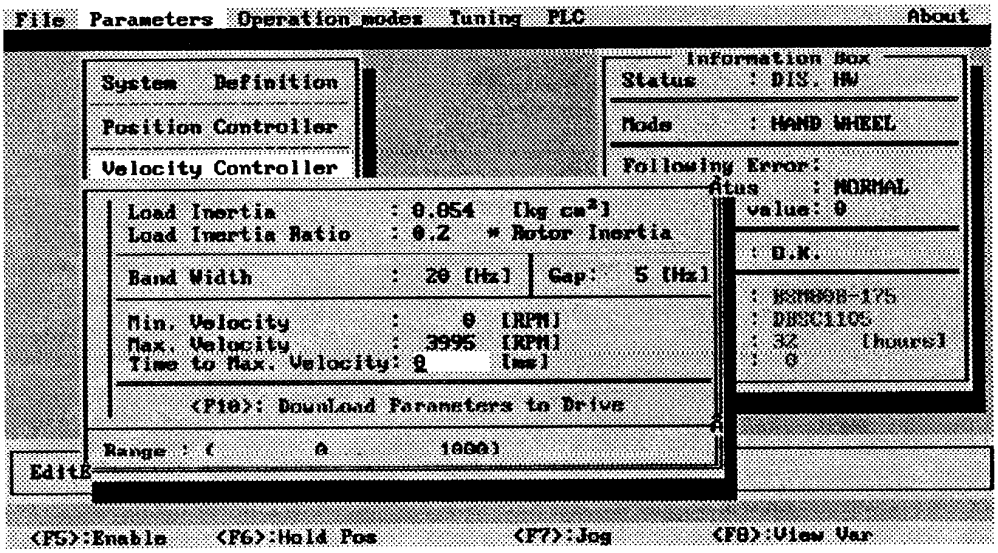
Remarks:

- The peak and nominal current limits are set automatically by the D3S to default values in accordance to the system basic configuration done in section 4.7.
- The peak current is the maximum instantaneous current that the amplifier can provide. The default value is the lowest amplifier and motor admissible peak current (the D3S selects it automatically). The modification of this value is accepted only if a lower limit is defined. In this case, the current analog input scaling is automatically redefined. 10 V input corresponds with the new the peak current.
- The nominal current is the maximum continuous current that the amplifier can provide. The default value is the amplifier nominal current (the D3S selects it automatically). The modification of this value is accepted only if a lower limit is given. this value is used by the DBSC for the "I^2t" limit calculation for the motor and amplifier protection. **It is highly recommended to modify and match the nominal current with the motor continuous current. This will prevent motor overheating. The motor continuous current value can be read on the motor label.**
- A torque limitation is also available (option Exx needed) via the second analog input (X1-3: CMD2+ and X1-4: CMD2-) and the PLC. This torque limit is proportional to the second analog input if the PLC "torque limit" function is enabled. In fact, the torque limit function is nothing more than a current limit which is permanently updated by the PLC. With 10 V input on the CMD2+ and CMD2- lines, the DBSC will allow 100% of the peak current selected in the above screen. (Refer to section 7.8 concerning the PLC).

5.3.2 Velocity and acceleration limits

The user can limit the maximal velocity and acceleration. Those limits are active only if the velocity mode or Hand Wheel mode is selected.

❶ Select the **"Parameters"** sub-menu and press **<Enter>**.
❷ Choose the **"Velocity Controller"** option and press **<Enter>**, the velocity loop parameters are displayed.
❸ Change the maximal and minimal velocity limits as well as the acceleration limit **(Time to Max. Velocity)** as required for the application:



❹ Press the **<F10>** key to download the parameters to the drive.

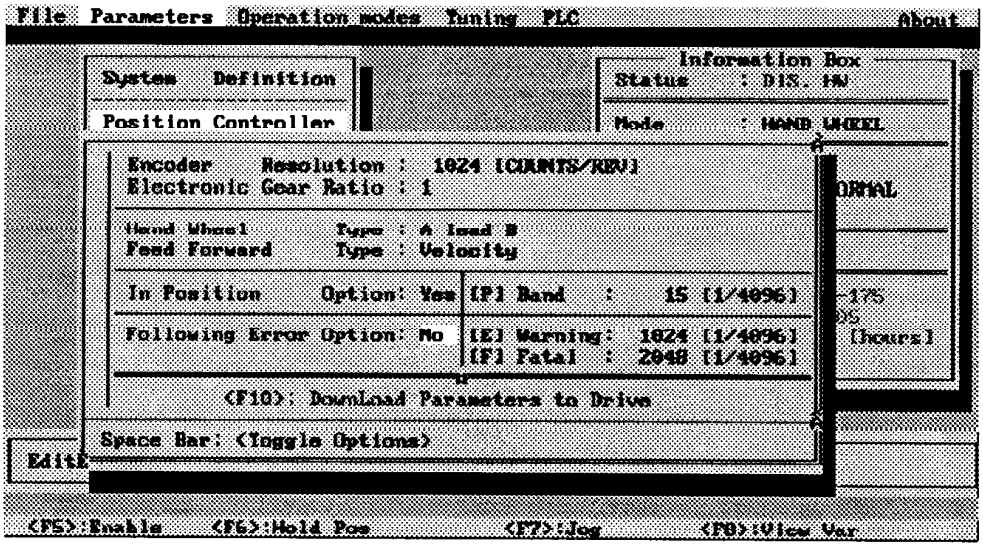
Remarks:

- In **Hand-Wheel** mode, it is important to specify a zero value (or very low value) for the acceleration time (**Time to Max Velocity**). This is to avoid motor overshoots when the master encoder move variations are abrupt.
- When a CNC or any motion controller is used, an external position loop is closed. In this case it is also important to specify a zero value (or very low value) for the acceleration time (**Time to Max Velocity**). **If a CNC move is programmed with an acceleration higher to the limit set in the DBSC, the motor can start strong oscillations, dangerous!**
- The "**Time to Max. Velocity**" feature permits to reach a smoother velocity response especially on abrupt variation of the analog velocity command input. Note that there is no link between this value and the one which is used in an external positioning card for the trajectory planning.
- The maximum velocity value can be changed. This permits to modify the default value set by the D3S in accordance with the selected motor and amplifier in the system basic configuration. **The analog input scaling is automatically redefined. 10 V input corresponds with the new maximum velocity value.** This feature has the great advantage to provide the highest possible resolution of the velocity command input.
- The maximum velocity value that can be entered is twice the default value. This permits to reach the maximum motor velocity when the analog command input can not exceed 5 Volts.
- The lower velocity limit avoids slow motions to occur if an offset appears on the command input. It may also avoid vibrations when the motor is at rest. This is specially useful when the velocity command input is noisy or when DAC output of a positioning system has a low resolution.

5.3.3 **Position following error** limits (Only available if Hand Wheel or Pulse Follower option is installed.)

The DBSC has 3 Hand Wheel following error alarm levels. The user can change any of those 3 values according to his needs. Those limits are active only when the "Hand Wheel" mode is selected.

❶ Select the "**Parameters**" sub-menu and press **<Enter>**.
❷ Choose the "**Position Controller**" option and press **<Enter>**. The position loop parameters are displayed.
❸ Change the default levels as required: "**[P] Band**" (In Position band), or/and "**[E] Warning**" (Following Error warning), or/and "**[F] Fatal**" (Fatal Following Error). Activate or deactivate the "**In Position**" Option and "**Following Error**" Option according to the needs:



❹ Press the **<F10>** key to download the parameters to the drive.

Remarks:

- The 3 following error levels are displayed in real time in the information box as well as on the "7 segments" display of the DBSC:
  P. - Following error is smaller than [P], (inside the position band).
  E. - Following error is somewhere between [E] value and [F] value. (warning)
  F. - Following error is greater than [F]. (Fatal. The amplifier stays enabled. If disable is needed, it can be done via the PLC.)
- The F error is a latched status. To cancel the status, you have to reset the **Following Error.** Refer to section 8.7.5: **"Reset Parameters"**. Reset is also programmable by the PLC (refer to chapter 7) and via hardware input **X2-9** (the option Exx must be installed).
- The **"Feed Forward Type"** is a toggle option choice. Use the space bar to cycle through the possibilities and select the desired one. (see paragraph 5.4.2)

5.4 Controller Parameters

The DBSC can operate either in current mode, velocity mode, or position mode (Hand wheel). The required mode has to be selected by the user.

This section explains in details how to tune the velocity and position loops. The special and advance tunings are explained in section 5.5. All the tuning procedures described bellow are important in order to obtain the required static and dynamic behaviors for the application.

It is necessary to tune the velocity controller even if the amplifier is used in current mode, since the velocity controller is automatically activated in special conditions, such as CW, CCW, and Hold-Position. In the above cases, an internal velocity command will over-write the analog input command.

Remarks:

- It is always recommended to disable the drive before updating the controller parameters.
- The current loop coefficients are automatically defined by the D3S program according to the amplifier and motor specifications which were selected in the "parameters" menu (see section 4.7). The only parameter which may be modified in the current loop, is the "phase advance" (see section 5.5). Therefor, the current loop is specifically defined for each combination of motor and amplifier. It assures an optimal tuning and high performance, but the system definition seen in section 4.7 needs to be accurate.

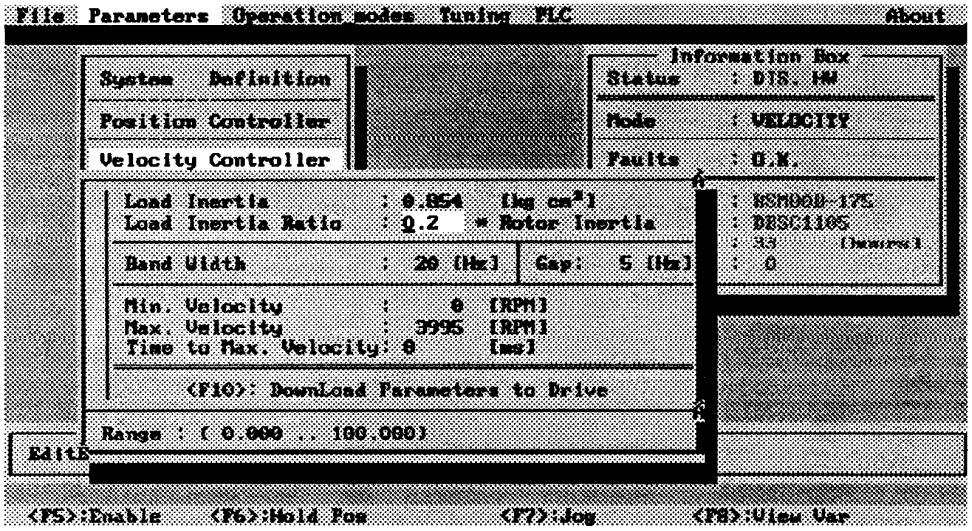
5.4.1 Velocity loop tuning

The velocity loop coefficients are automatically defined by the D3S program according to the specifications of the amplifier and motor as selected in the "parameters" menu (see section 4.7). To complete this tuning, it is necessary to define the load inertia value at the first place, and the required band width in the second place.

Note that when a new velocity controller is downloaded (velocity loop redesign), all the position and velocity control loop gains will be reset to 100%. Refer to paragraph 5.5.3.

5.4.1.1 Load Inertia estimation

❶ Disable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8.
❷ Select the sub-menu **"Parameters"** and press **<Enter>**.
❸ Choose the **"Velocity Controller"** option and press **<Enter>**, the velocity loop parameters are displayed.
❹ Select **"Load Inertia Ratio"**. This is the ratio between the load inertia reflected to the motor shaft and the motor inertia. To start, if there is no preliminary data, define an initial value of 0,2.

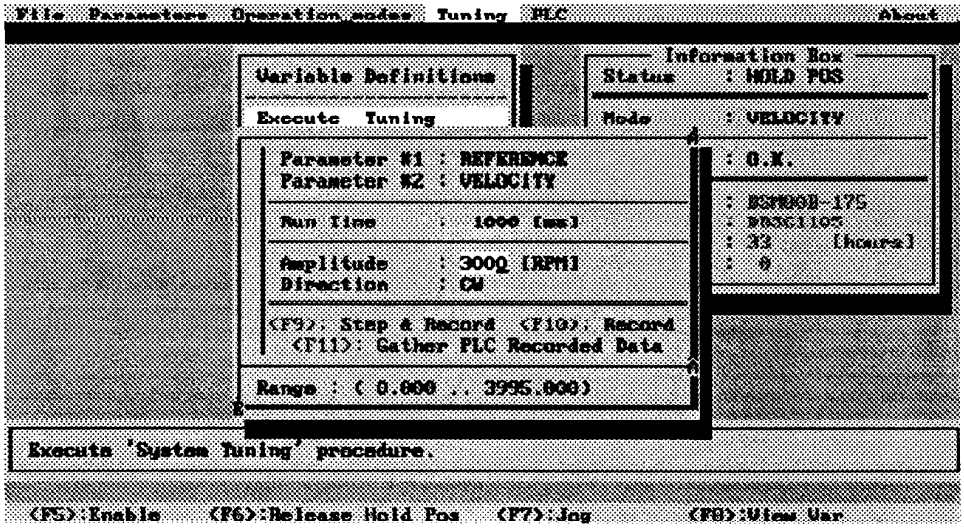


Note that when the **"Load Inertia Ratio"** is modified, the **"Load Inertia"** value is automatically changed; those two values are dependent of each other in relation with the motor inertia.

❺ Press the **<F10>** key to download the new parameters to the DBSC's personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen. Procede as follows:

① Enable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8.
② Activate the "Hold Position" by pressing the **<F6>** key.
③ Select the **"Tuning"** sub-menu and press **<Enter>**.
④ Choose the **"Execute Tuning"** option and press **<Enter>**.
⑤ Select the parameters as shown in the following screen:



The parameters #1 and #2 of which the data will be memorized and reported graphically to the screen are selected as follows:

- Locate the cursor on **"Parameter #1"** et press **<Enter>**. The parameters list is displayed.
- Locate the cursor on the desired parameter and press **<Enter>**.
- Repeat this procedure for the parameter **#2**.

⑥ Press the **<F9>** key to execute the move and the data gathering.

⑦ Look at the received response:

- Select the **"Tuning"** sub-menu and press **<Enter>**.
- Choose the **"View Graph"** option and press **<Enter>**.
- Observe the system response:
  - ♦ If the response shows an overshoot, the load inertia estimated value should be increased:
    Repeat steps ❶ to ❻ and then steps ① to ⑦.
    Increase the **"Load inertia ratio"** value by increments of 0.2 until the overshoot disappears.
  - ♦ If vibrations appear, decrease the estimated **"Load inertia ratio"** value.

Figure 5.1(a) shows the vibrations case when the **"Load Inertia"** value becomes too high.
Figure 5.1(b) shows the overshoot case when the **"Load Inertia"** value is too low.

The flow chart shown in figure 5.2 describes, in a simple and concise manner, the load inertia research method.

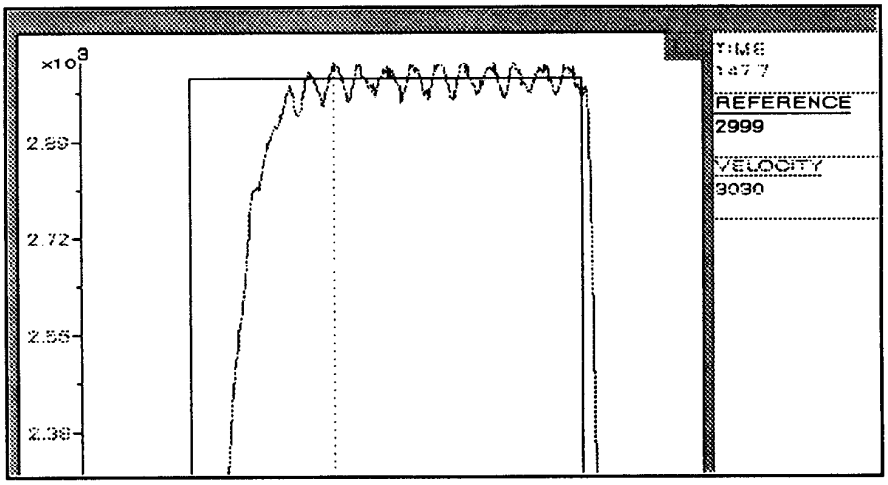


Figure 5.1 (a): The **"Load inertia"** value is too high

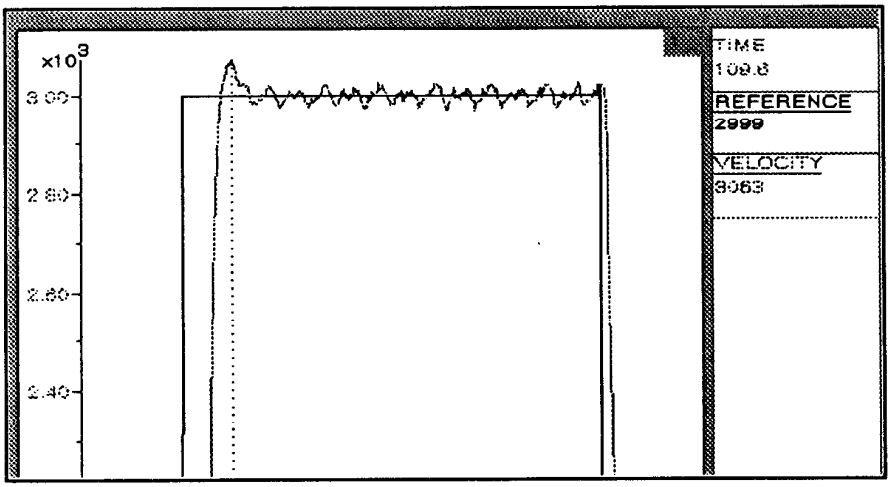


Figure 5.1 (b): The **"Load inertia"** value is too low

Remarks:

- If the **"Load inertia"** is not known, always start with a lower **"Load inertia ratio"** value than the actual ratio. In most cases, 0.2 is a good starting value.
- The **"Hold Position"** activation via the **<F6> key, before sending the "Step & record"** command via the **<F9> key, is necessary in order to prevent an undesirable interference of the system during** this operation.

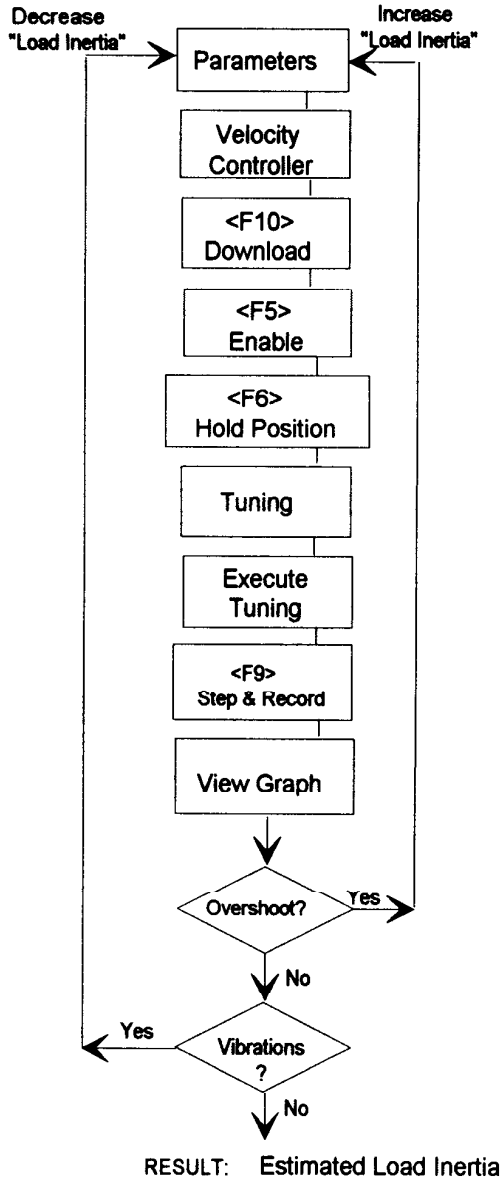


Figure 5.2: "Load Inertia" research method

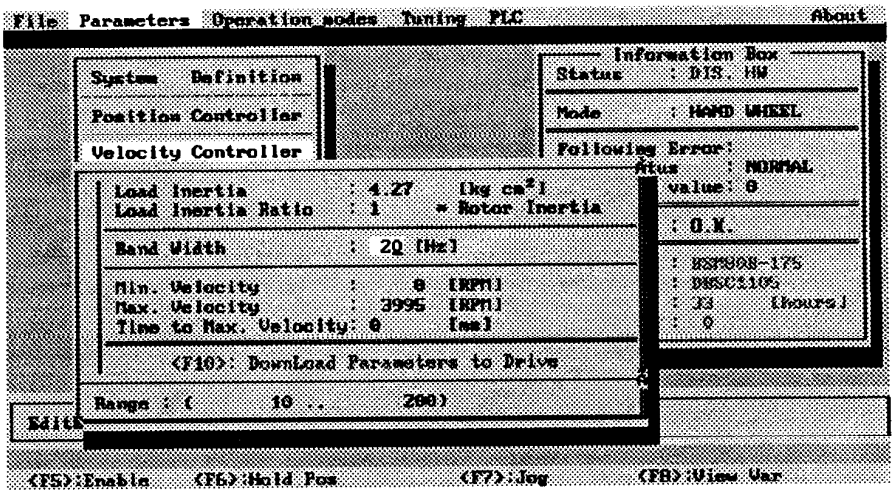
5.4.1.2 Band Width Tuning

When the estimated **Load Inertia** is well defined, the Band Width may be modified. It is recommended to increase the Band Width only if higher dynamic response is required. Increasing this value decreases the following error when the **"Hand-wheel mode"** is selected, or decreases the following error **variation** when the **"velocity mode"** is selected and the position loop is performed by a CNC or any axis controller board. Two methods are possible to optimize the Band Width:

- In **"Hand-Wheel mode"**, where the following error can be reported to the screen.
- In **"Velocity mode"** where the **actual velocity** and the **velocity reference** are visualized on the screen. The target is to minimize the system response time; subsequently this will also minimize the following variation in the CNC or in the axis controller board, but will increase the frequency of the oscillations. (A compromise has to be found).

Band Width Tuning in "Hand-Wheel mode"

❶ Disable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8
❷ Select the **"Operation_modes"** sub-menu and press **<Enter>**.
❸ Select the **"Control Mode"** option, press **<Enter>**, then choose the **"Hand-Wheel"** mode.
❹ Select the **"Parameters"** sub-menu and press **<Enter>**.
❺ Choose the **"Velocity Controller"** option and press **<Enter>**, the velocity loop parameters are displayed.
❻ Select **"Band With"** and define a Band Width of 20 Hz to start, as shown in the following screen:



❼ Press the **<F10>** key to download the parameters to the personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen according to the following procedure:

① Enable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8.
② Select the **"Tuning"** sub-menu and press **<Enter>**.
③ Choose the **"Execute Tuning"** option and press **<Enter>**.
④ Select the parameters **"Velocity"** and **"Following Error"** which will be gathered.
⑤ Generate a move via the Hand-wheel (preferably a constant velocity move and near to the maximal velocity required by the application) and press simultaneously the **<F10>** key to start the data gathering.
⑥ Look at the received system response:
- Select the **"Tuning"** sub-menu and press **<Enter>**.
- Choose the **"View Graph"** option and press **<Enter>**.
- The velocity and the following error are shown on the screen; observe this response, the target is to decrease the following error.
  If the response shows an undesirable following error, repeat steps ❹ to ❽ and ① to ⑥ with a new Band Width:
  - ♦ If the following error is too big, increase the Band Width.
  - ♦ If vibrations appears, decrease the Band Width.

Figure 5.3 shows a velocity and following error data gathering example. In this case, only the **Velocity Feed Forward** was activated (see paragraph 5.4.2 in order to understand the effect and the importance of the Velocity and Acceleration Feed Forwards).

The flow chart shown in figure 5.5 (b) describes, in a simple and concise manner, the Band Width Tuning while in **"Hand-Wheel mode"**.

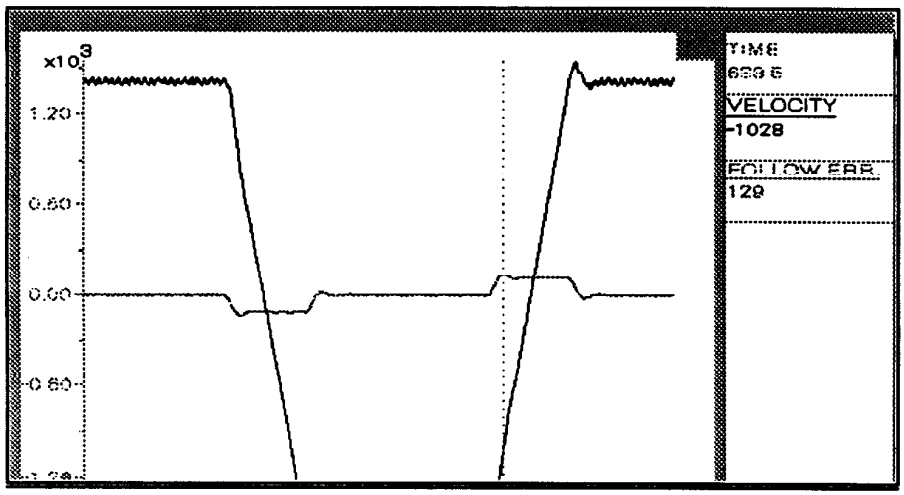


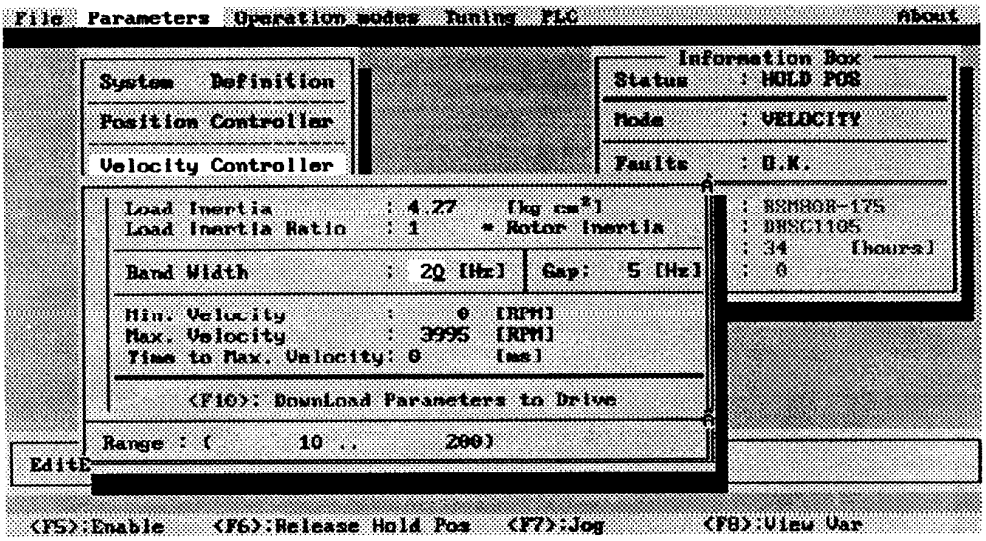
Figure 5.3: Velocity and Following Error versus time graph
(with Velocity Feed Forward activated)

Remarks:

- The "Hand-Wheel" move generation (point ⑤), requires manual synchronization between the system activation and the data recording (**<F10>** key). It is therefor desirable to choose a long enough data recording time (about 5 seconds).
- The following error is strongly reduced when both Feed Forwards (Velocity and Acceleration) are activated. Refer to the section 5.4.2 for more information.

Band Width Tuning in "Velocity mode"

❶ Disable the drive via the **<F5>** key, or the X2-2 input, or the DIP switch AS8.
❷ Select the **"Operation_modes"** sub-menu and press **<Enter>**.
❸ Select the **"Control Mode"** option, press **<Enter>**, then choose the **"Velocity"** mode.
❹ Select the **"Parameters"** sub-menu and press **<Enter>**.
❺ Choose the **"Velocity Controller"** option and press **<Enter>**, the velocity loop parameters are displayed.
❻ Select **"Band With"** and define a Band Width of 20 Hz to start, as shown in the following screen:



❼ Press the **<F10>** key to download the parameters to the personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen according to the following procedure:

① Enable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8.
② Activate the **"Hold Position"** by pressing the **<F6>** key.
③ Select the **"Tuning"** sub-menu and press **<Enter>**.
④ Choose the **"Execute Tuning"** option and press **<Enter>**.
⑤ Select the parameters **"Velocity"** and **"Velocity Reference"** which will be gathered.
⑥ Press the **<F9>** key to execute a velocity step response and the data gathering.
⑦ Look at the received system response:
- Select the **"Tuning"** sub-menu and press **<Enter>**.
- Choose the **"View Graph"** option and press **<Enter>**.
- The actual velocity and the velocity reference are shown on the screen; observe the response, the target is to minimize the actual velocity rising time.
  If the result is not satisfactory, repeat the steps ❶ to ❿ and ① to ⑦ with a **new Band Width:**
  - ♦ If the rising time is too high, increase the Band Width.
  - ♦ If vibrations appears, decrease the Band Width.

Figure 5.4 (a) shows the response of a system tuned with a low Band Width.
Figure 5.4 (b) shows the response of a system tuned with a high Band Width.

The flow chart shown in figure 5.5 (a) describes, in a simple and concise manner, the Band Width Tuning while in **"Velocity mode"**.

Remark:

- The Band Width tuning must be performed after the Load Inertia Tuning. **Don't ever change the Load Inertia value while tuning the Band Width.** In this way, the response will always look similar: clean, without oscillations and without overshoot; this also indicates that the system definition saved in the DBSC is correct.

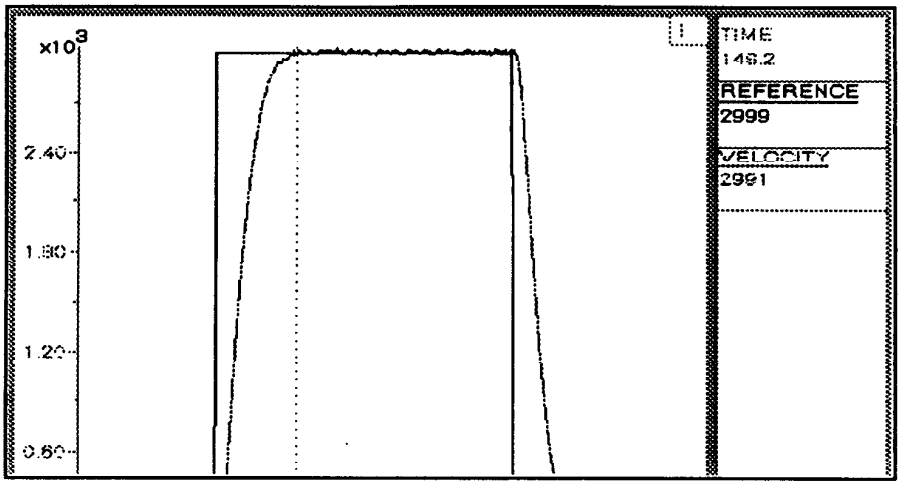


Figure 5.4 (a): response of a system tuned with a low Band Width

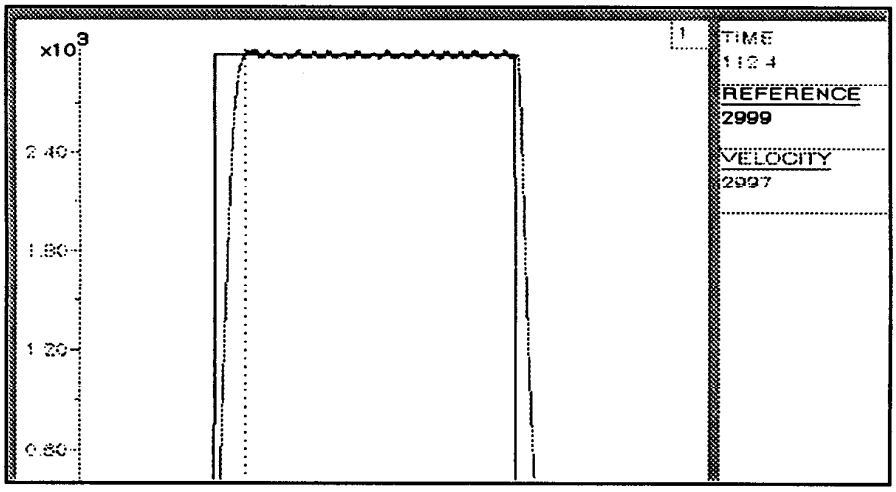


Figure 5.4 (b): response of a system tuned with a high Band Width.

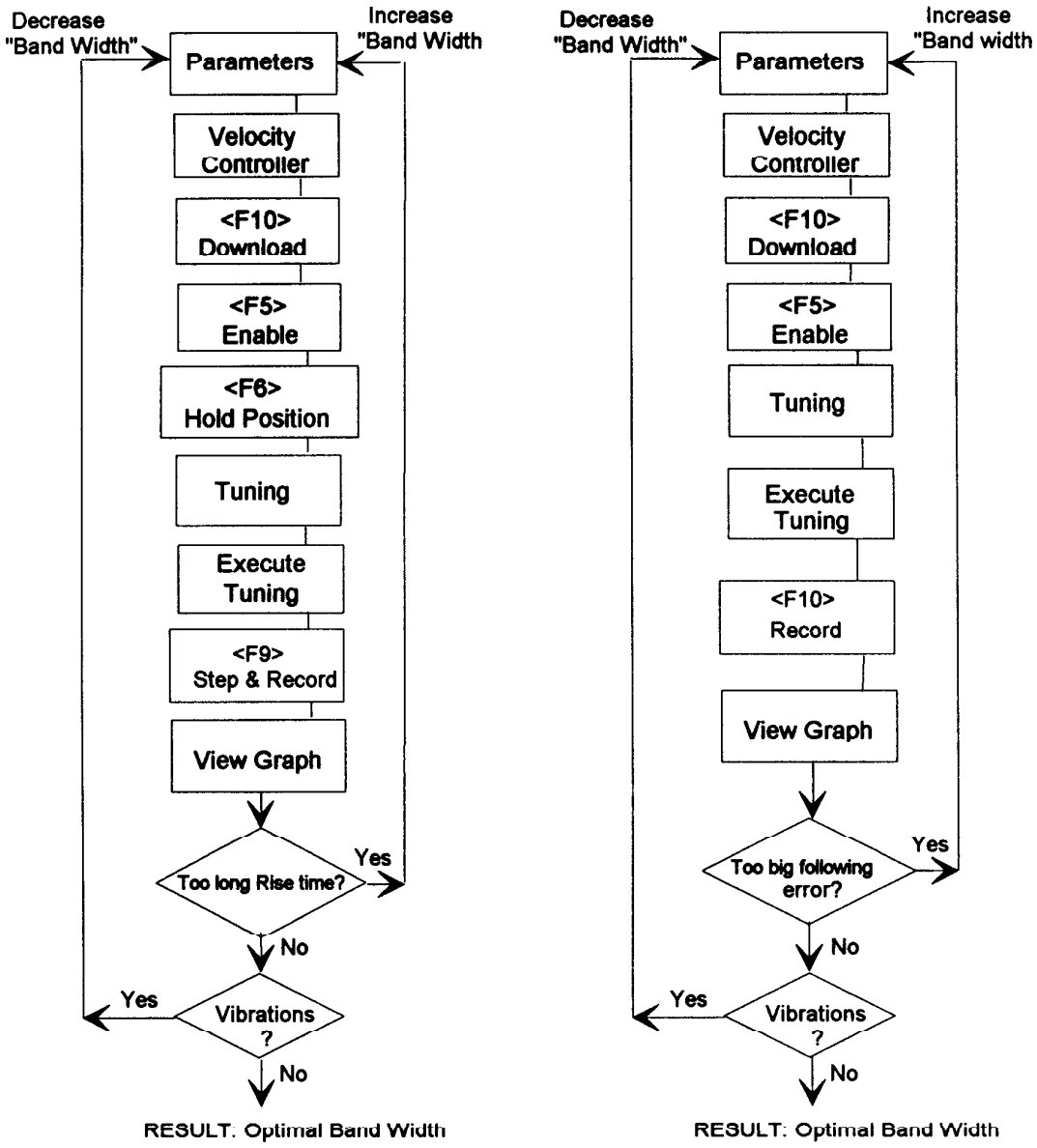


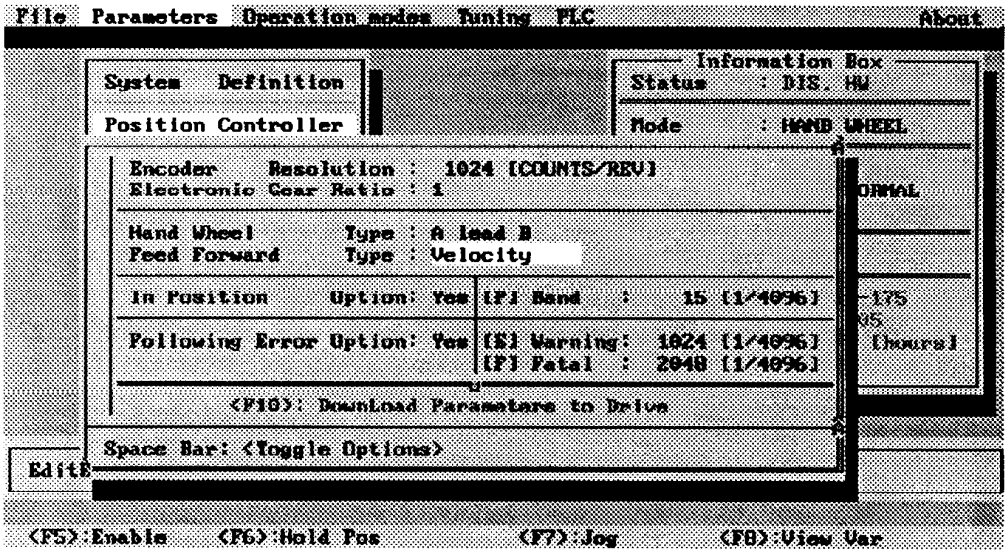
Figure 5.5: Band Width Tuning procedure:
(a) in "Velocity" mode, **(b)** in "Hand-Wheel" mode.

5.4.2 Position controller

In the DBSC, the position loop is executed when the **"Hand-Wheel"** mode is selected. It is also activated under particular situations such as **"CW"** or **"CCW"** limits, or **"Hold Position"** (**<F6>** key or X2-3 input). The position controller structure is simple, it's tuning is based on the velocity controller and is automatically done. The only possible user's action on this controller is the activation or deactivation of the Velocity and Acceleration Feed Forwards. The activation of the Feed Forwards decreases significantly (up to 10 times) the following error while tracking the Hand-Wheel. on the other hand, it turns the system to be more nervous (the move smoothness may be affected as the system becomes more sensitive to the hand-wheel command input).

By default, only the Velocity Feed Forward is activated. In order to make the best choice for the application, the procedure bellow should be followed:

❶ Disable the drive via the **<F5>** key, or X2-2 input, or the DIP switch AS8.
❷ Select the **"Operation_modes"** sub-menu and press **<Enter>**.
❸ Select the **"Control Mode"** option, press **<Enter>**, then choose the **"Hand-Wheel"** mode.
❹ Select the **"Parameters"** sub-menu and press **<Enter>**.
❺ Choose the **"Position Controller"** option and press **<Enter>**, the position loop parameters are displayed.
❻ Select **"Feed Forward Type"** and choose **"Velocity"** to start, as shown in the following screen



❼ Press the **<F10>** key to download the parameters to the personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen according to the following procedure:

① Enable the drive via the **<F5>** key, or the X2-2 input, or the DIP switch AS8
② Select the **"Tuning"** sub-menu and press **<Enter>**.
③ Choose the **"Execute Tuning"** option and press **<Enter>**.
④ Select the parameters **"Velocity"** and **"Following Error"** which will be gathered.
⑤ Generate a move via the Hand-wheel (preferably a trapeze velocity profile move) and press simultaneously the **<F10>** key to start the data gathering.
⑥ Look at the received system response:
- Select the **"Tuning"** sub-menu and press **<Enter>**.
- Choose the **"View Graph"** option and press **<Enter>**.
- The actual velocity and the following error are shown on the screen: observe the response. the target is to minimize the following error.
  If the response shows an undesirable following error, repeat steps ❶ to ❶ and ① to ⑥ with a new Feed Forward selection:

Figure 5.6 (a) shows a system response with a position loop which uses both Feed Forwards (Velocity and Acceleration).
Figure 5.6 (b) shows a system response with a position loop which does not use any Feed Forward.
Figure 5.3 shows a system response with a position loop which uses only the Velocity Feed Forward.

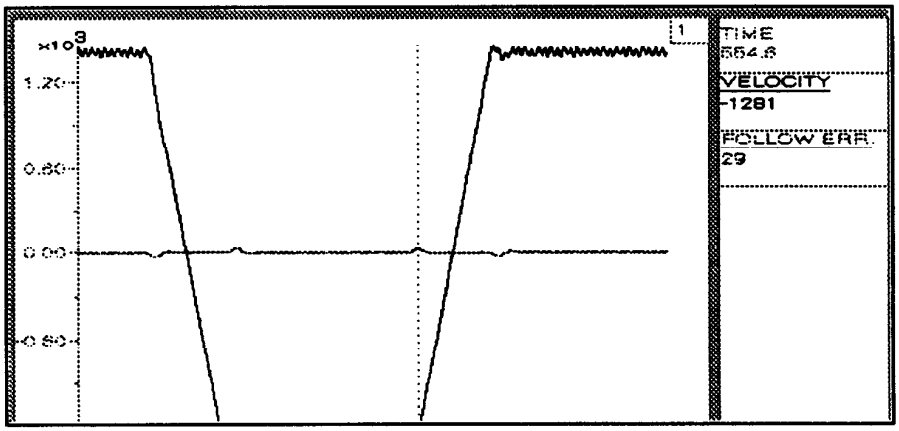


Figure 5.6 (a): system response with both Feed Forwards (Velocity and Acceleration) activated

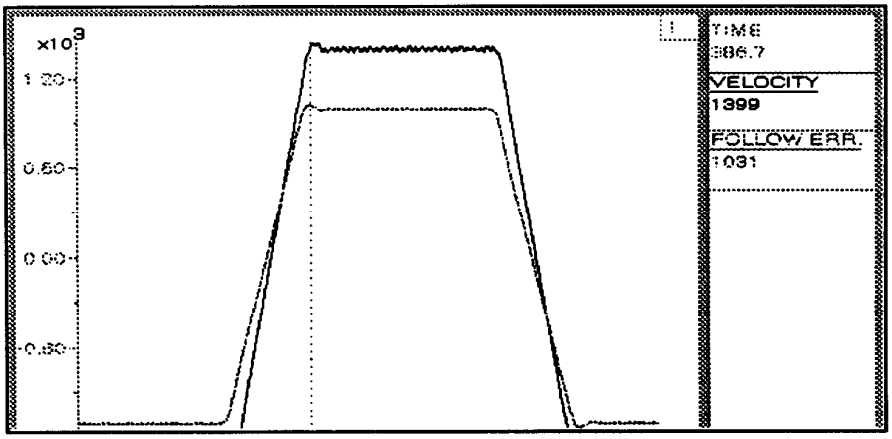


Figure 5.6 (b): system response without any Feed Forward

Remarks:

- The "Hand-Wheel" move generation (step ⑤), requires manual synchronization between the system activation and the data recording (**<F10>** key). It is therefor desirable to choose a long enough data recording time (about 5 seconds).
- The recommendations for the best use of the Velocity and Acceleration Feed Forwards are listed below:
  - If the Hand-Wheel move variations are smooth and if the Following Error must be small (or approximately zero during constant velocity and Acceleration), activate both Feed Forwards (Velocity and Acceleration).
  - If the Hand-Wheel move variations are not so smooth and if the following error must be small, (or approximately zero when velocity is constant), activate only the Velocity Feed Forward.
  - If the Hand-Wheel move variations are strong, and if the motor moves must be smooth, and that following error is not too important, deactivate both Feed Forwards.

5.5 Advanced tuning

This section is about special tunings that may help to optimize the system in some particular cases. Most of the time, these tunings are not required.

5.5.1 Phase advance tuning (current loop)

The **Phase advance** tuning allows to decrease the current required when the motor is loaded and moving. The effect of an optimal phase advance tuning is specially noticeable at high velocity and acceleration: lower heat of the motor and, in some cases, avoids reaching the "I^2t" limit (Error 7. on the "7 segments display"). In most cases the default value (468 µSec) is sufficient. If there is a doubt, phase advance should be tuned.

To optimize the phase advance, observe the current command while the motor is moving at constant velocity (500 RPM or more) while it is loaded. It is recommended to execute a move with the maximal acceleration required by the application and look at the peek current. The phase advance will be optimal when the current will reach a minimum value under a given load and motion.

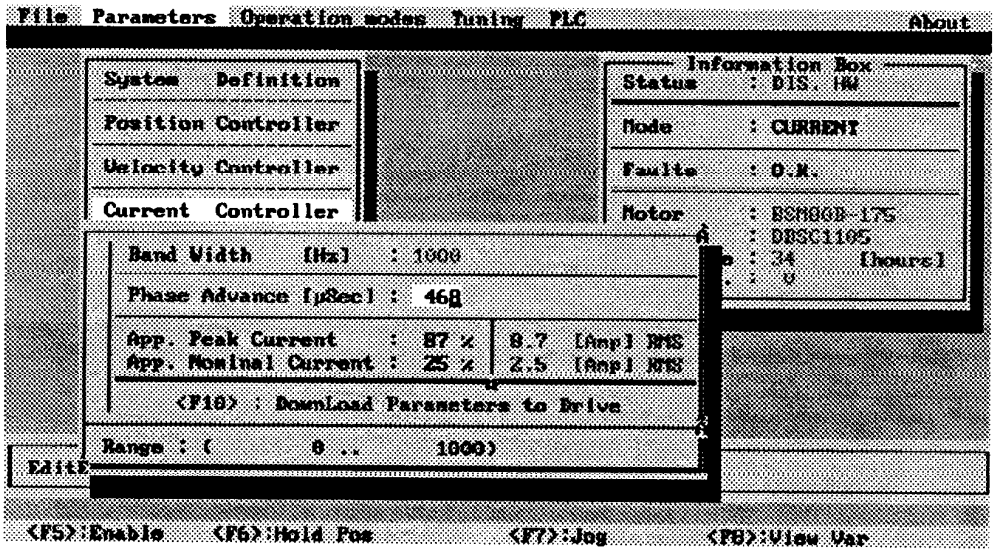
Two cases are considered for this tuning:

1. Phase advance optimization while in "**Current** mode"
2. Phase advance optimization while in "**Velocity** mode"

Phase advance optimization while in "Current mode"

Warning!
Do not execute the phase optimization while in "Current mode" unless the CNC or the motion control card is set for controlling a current mode amplifier. The CNC must have it's own velocity loop active and tuned (Derivative gain tuned). Some CNC and motion control boards are not equipped with velocity feedback capabilities. Instability and hazardous moves will occur if the position loop is closed with such systems while the DBSC "Current" mode" is selected.

❶ Disable the drive via the **<F5>** key, or X2-2 input, or the DIP switch AS8.
❷ Select the **"Operation_modes"** sub-menu and press **<Enter>**.
❸ Select the **"Control Mode"** option, press **<Enter>**, then choose the **"Current"** mode.
❹ Select the **"Parameters"** sub-menu and press **<Enter>**.
❺ Choose the **"Current Controller"** option and press **<Enter>**, the current loop parameters are displayed.
❻ Select **"Phase Advance"**, start with the default value (468 µSec.) as shown in the following screen:



❼ Press the **<F10>** key to download the parameters to the personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen according to the following procedure:

① Enable the drive via the **<F5> key**, or the X2-2 input, or the DIP switch AS8
② Select the **"Tuning"** sub-menu and press **<Enter>**.
③ Choose the **"Execute Tuning"** option and press **<Enter>**.
④ Select the parameters **"Velocity"** and **"Current"** which will be gathered.
⑤ Generate a move via the CNC or axis controller board, and press simultaneously the **<F10>** key to start the data gathering.
⑥ Look at the received system response:
- Select the **"Tuning"** sub-menu and press **<Enter>**.
- Choose the **"View Graph"** option and press **<Enter>**.
- The actual velocity and current are shown on the screen. If necessary, the velocity graph can be hidden for a better view of the current (refer to paragraph 8.8.5, under **"soL"**).
  The target is to minimize the current.

⑦ Change the phase advance value (by 100 μSec. increments) and repeat steps ❶ to ❺ and ① to ⑥. (we will start by decreasing the value.)
- If the current is lower than before, the phase advance value should be further reduced. Repeat steps ❶ to ❺ and ① to ⑥.
- If the current is higher than before, the phase advance value will be increased some more (by 100μSec. increments). Repeat steps ❶ to ❺ and ① to ⑥.
- The phase advance will be optimal when the maximum current reaches a minimum value.

Figure 5.7 (a) shows the current profile in one phase of the motor. The current peeks that we observe are due to the acceleration and the deceleration of the load.
Figure 5.7 (b) shows the current profile after the phase advance optimization. The acceleration peek current has been reduced from 12.65 A to 9.625 A.

The flow chart shown in figure 5.8 describes, in a simple and concise manner, the phase advance optimization method while in **"Current mode"**.

Note:
The equivalent current command "CURRENT", and the phase current feedback "CURRENT U" and "CURRENT V", can be shown on the screen. **The current values reported to the screen are true values (not RMS).** For example, a 5 Amps DBSC is able to deliver 10 A peak RMS that are 14.1 A peak true value. This is also true when the current is viewed by using the **<F8>** key.

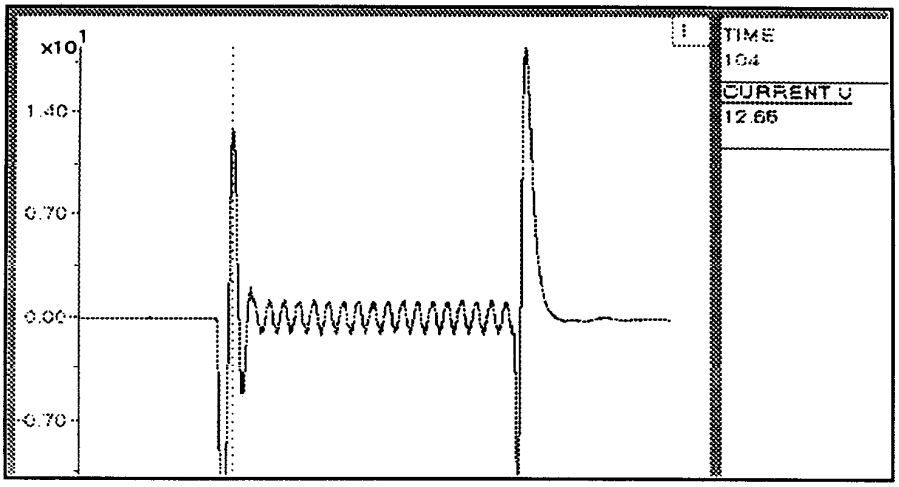


Figure 5.7 (a): current profile in one motor phase, phase advance none optimized

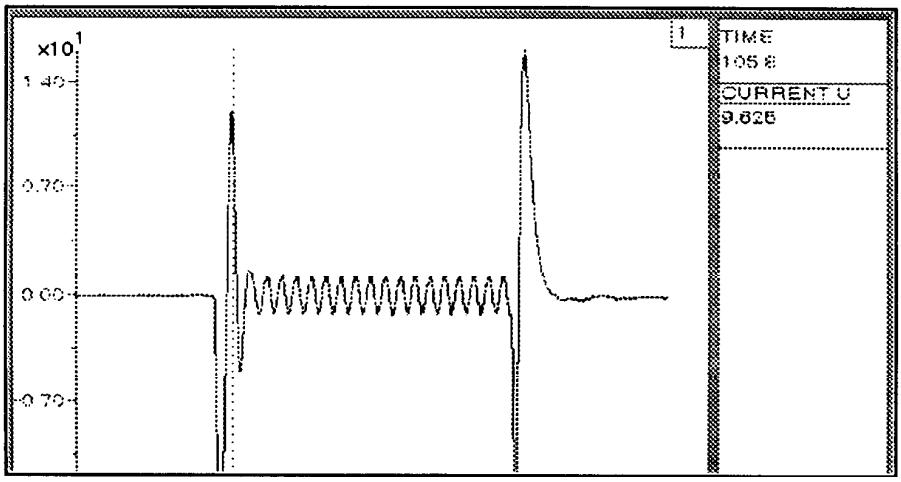


Figure 5.7 (b): current profile in one motor phase, phase advance is optimized

Warning!
Do not execute the flow chart below unless the CNC or the motion control card is set for controlling a current mode amplifier. The CNC must have it's own velocity loop active and tuned (Derivative gain tuned). Some CNC and motion control boards are not equipped with velocity feedback capabilities. Instability and hazardous moves will occur if the position loop is closed with such systems while the DBSC "Current" mode" is selected.

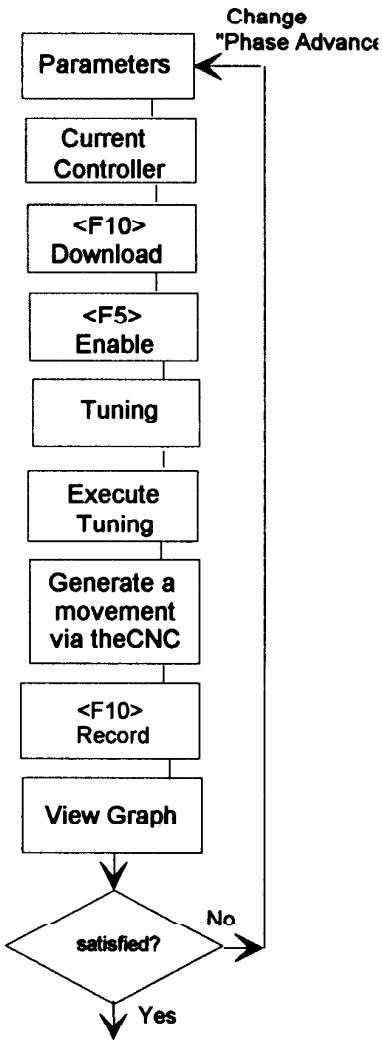
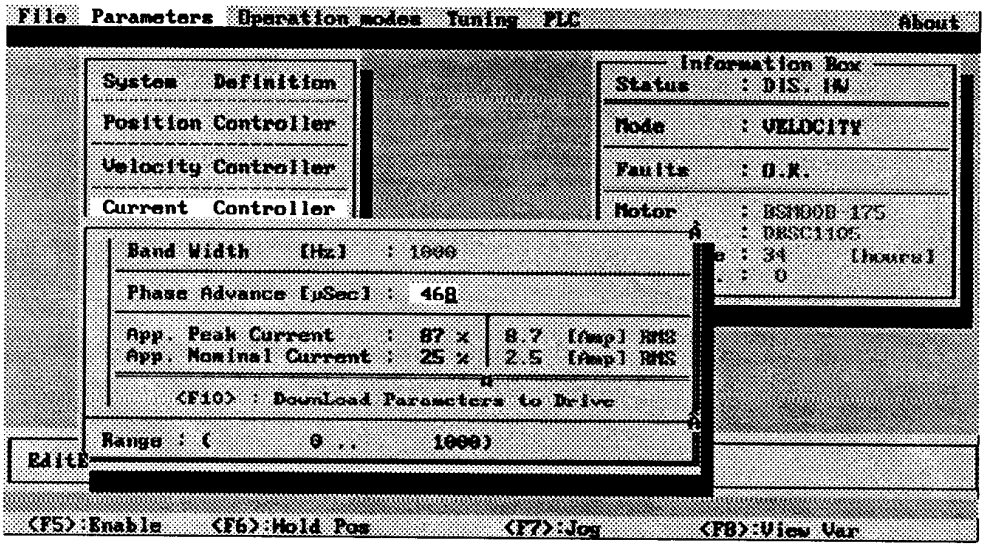


Figure 5.8: Phase Advance optimization procedure in **"Current mode"**

Phase advance optimization while in "velocity mode

❶ Disable the drive via the **<F5>** key, or X2-2 input, or the DIP switch AS8.
❷ Select the **"Operation_modes"** sub-menu and press **<Enter>**.
❸ Select the **"Control Mode"** option, press **<Enter>**, then choose the **" Velocity "** mode.
❹ Select the **"Parameters"** sub-menu and press **<Enter>**.
❺ Choose the **" Velocity Controller"** option and press **<Enter>**, the velocity loop parameters are displayed.
❻ Define the velocity and the "Time to Max. Velocity" (acceleration rate) equivalent to the one which will be used in the application.
❼ Press the **<F10>** key to download the parameters to the personality module.
❽ Choose the **" Current Controller"** option and press **<Enter>**, the current loop parameters are displayed.
❾ Select **"Phase Advance"**, start with the default value (468 µSec.) as shown in the following screen:



❿ Press the **<F10>** key to download the parameters to the personality module.

To test the system performances, it is necessary to execute a move with the motor while performing a data gathering, and then to watch the system response on the screen according to the following procedure:

- Enable the drive via the **<F5>** key, or the X2-2 input, or the DIP switch AS8.
- Activate the **"Hold Position"** by pressing the **<F6>** key.
- Repeat the steps ② to ⑦ above (in section 5.5.1), but press this time the **<F9>** key for the "Step and Record option" instead of the **<F10>** key.

5.5.2 Feed Forward Gains

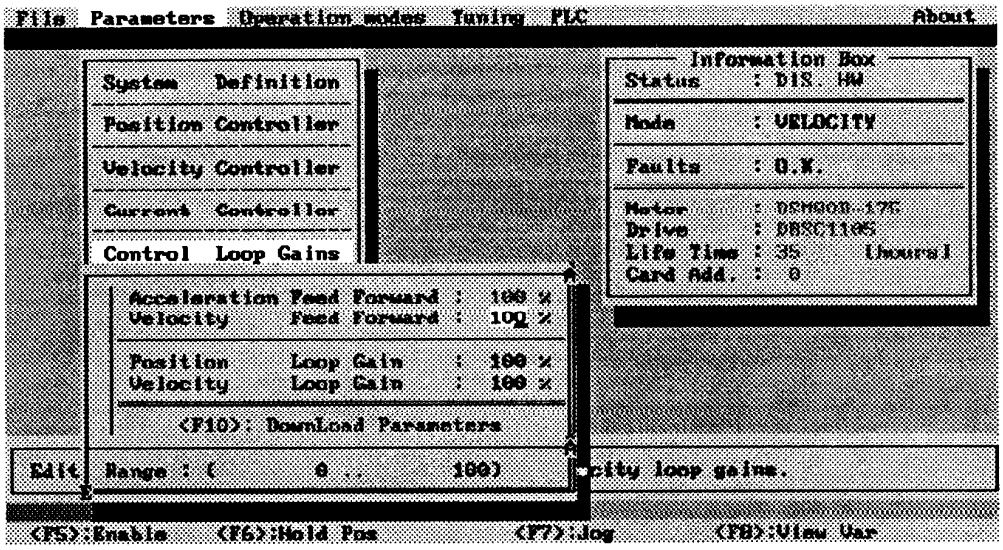
Generally, when the "Hand-Wheel" is used, the Velocity and Acceleration Feed Forward selection procedure seen in section 5.4.2 gives the optimal result. It is not recommended to change those gains. Nevertheless, there are situations where we desire to obtain a compromise between a small following error and the movement smoothness, or other situations where the "position overshoots" (caused by the Feed Forwards) have to be reduced. This is why the possibility of changing the Feed Forward gains have been implemented.

Before changing the Feed Forwards gains, display the Following Error on the screen according to the instructions given in the section 5.4.2.

- If only the Velocity Feed Forward is enabled, and the position shows an overshoot at the end of the move (position overshoot or following error sign change), decrease the Velocity Feed Forward gain. Repeat the procedure until satisfactory result is achieved.
- If the Velocity and Acceleration Feed Forwards are activated and the position shows an overshoot at the end of the move (position overshoot or following error sign change), decrease the Velocity and the acceleration Feed Forward gains. Repeat the procedure until satisfactory result is achieved.

To change the Feed Forwards gains, follow the points below:

❶ Select the **"Parameters"** sub-menu and press **<Enter>**.
❷ Select the **"Control Loop Gains"** option, press **<Enter>**
❸ Select the desired gains as shown in the following screen:



❹ Press the **<F10>** key to download the parameters to the personality module.

5.5.3 Position loop and Velocity loop Controller gains

Although the D3S program allows the user to change those gains, it is **not recommended** to modify any of those gains. In fact, the experience has shown that any modification of those gains has always lead to system performance degradation. If you still want to change those gains, please take good note of the points bellow:

- **The servo loop gains are analytically defined by the filter calculation during the operations executed in sections 4.7 and 5.4. Any change of those gains modifies the system transfer function. Subsequently, in some cases, it may result in a loss of the system stability which causes dangerous and hazardous situations.**
- **Increasing the velocity loop gain may lead to vibrations and even to a loss of the system stability. Decreasing the gain is possible, but it will lower the actual band width.**
- **Increasing the position loop gain may lead to low frequency vibrations appearance (limit cycle). Decreasing the gain is possible, but it will increase the following error.**
- **When a new velocity controller is downloaded (as explained in section 5.4.1), the position and velocity loop gain are reset to 100%.**

5.5.4 Band Width tuning and inertia moment setting knowing the Bode plot of the system.

This paragraph can be very important in some particular cases. It is recommended to read it even if sound knowledge of system theory is needed for a full understanding.

In most cases, the tunings done in paragraphs 5.4.1.1 and 5.4.1.2 provide full satisfaction. If some instability or oscillation problems appear when we try to increase the **"Load Inertia"** or the **"Band Width"** values in the **"velocity controller"** sub-menu, it is necessary to consider the following:

The transfer function of a motor is composed of a simple pole. When a load is connected to the motor, the pole natural pulsation (ω_0) is reduced. As long as transmission (coupling, gear, timing belt etc.) is very rigid or the load very small, the model change is negligible. But when the transmission is flexible, an anti-resonance and a resonance appear in the Bode plot.

Figure 5.9 illustrates the three cases described above.

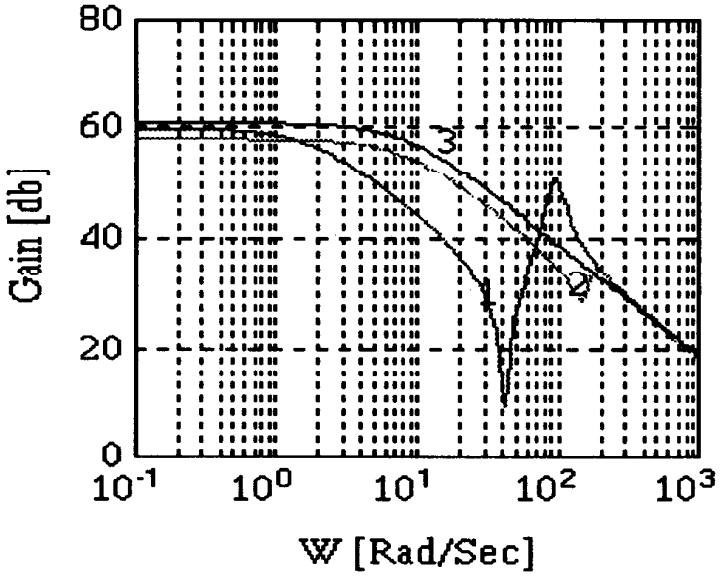


Figure 5.9: Bode plot of three electromechanical systems.
1. Flexible transmission or/and big load.
2. Rigid transmission or/and small load.
3. Motor and load with perfectly rigid coupling (simple pole model).

The graphs 1 and 2 converge and meet with the graph 3 at high frequency. This last graph represents the motor inertia and the part of the load inertia which is rigidly coupled to the motor. **The equivalent model at high frequency is therefor based on the motor inertia and a part of the load inertia.**

Two cases are then possible:

1. **The transmission and the coupling are very rigid or the load is very small:**

   The model calculated by the D3S software is accurate, the load inertia reported to the motor shaft is to be used as done in paragraph 5.4.1.1.

2. **The transmission or/and the coupling are flexible or the load is big:**

   Here two possibilities can be considered:

   A. We use the load inertia reflected to the motor shaft as above. In this case, the model will only be accurate for the low frequencies but will be inaccurate for the high frequencies. It will then be necessary to limit the band width to the anti-resonance frequency.
   B. We define a **small** inertia equal to the one which is rigidly coupled to the motor shaft. In this case the model is only accurate for the high frequencies. The controller band width can be increased and the total system band width is limited by the mechanics.

Remarks:

In the case B above, the band width in closed loop can be higher than the resonance frequency. It is therefor necessary to exercise caution as it may be possible that the system reaches the resonance!! It may vibrates, or even becomes unstable in certain cases.
You may notice the anti-resonance effect in the system step response.

6. Multi-resolver system (IMAS option) *preliminary information!*

6.1 Introduction

The "Inductive Modular Absolute System" (**IMAS**) is an option that has to be installed on the motor and on the DBSC. The advantage of the multi-resolver system is that the absolute position can be read after any power-up and, therefor, there is no need for the CNC or the positioning system to move the axis to a reference point.

6.1.1 IMAS option on motor

It provides absolute positioning up to 65535 revolutions. One fine resolver is directly coupled to the motor shaft; it provides the absolute position within one revolution. The fine resolver also transmits the commutation signal to the drive.

The absolute position range that can be up to 65535 revolutions, is given by the coarse resolvers. They can be read only when motor is at standstill and the amplifier disabled. A maximum of five resolvers in total can equip a motor (one fine resolver and 4 coarse). There is a gear ratio of 1/16 between each resolver of the IMAS option.

The **IMAS** option has to be ordered with the motor, it can not be installed on the field. The available absolute position ranges are 256, 4096 and 65535 revolutions; they are function of the number of resolvers installed. The feedback option suffix, written on the motor label, tells the absolute position range as shown in the table 6.1 bellow.

Feedback option suffix	Position range (rev.)	Total Number of resolvers	Number of coarse resolvers
I	256	3	2
J	4096	4	3
K	65535	5	4

Table 6.1: IMAS options with absolute position range

6.1.2 IMAS option on the drive

The DBSC must be ordered with the IMAS reading capability option. Refer to table 2.1 for the drive option definitions.

With the IMAS reading implementation, only the "*IMAS absolute value*" is available for the user. The "*Absolute actual machine position*" feature may be integrated in the firmware in future. At present time, it has to be executed by the host. The IMAS setup and reading operations can be executed only via the "Host Protocol Communication". The "D3S" software can not be used for these purposes. After the IMAS setup, the DBSC can calculate the "*IMAS absolute value*". the host can read the IMAS status and the result. The Host has to perform all other operations such as "*Offset*" calculation, "*mechanical correlation*" and "*absolute actual machine position*" calculation. Section 6.3 explains the user setup procedure and position reading operations.

The following commands have been added to the "*Host Protocol Communication*" (HCP):

- **Set number of course resolvers:** Sets-up and Stores the number of coarse resolvers present in the motor. The value can be between 0 and 4.
- **Read number of coarse resolvers:** Returns the number of coarse resolvers defined and saved in the Eeprom while executing the above command. The value returned should be between 0 and 4.
- **Calibrate IMAS:** Addresses and reads each resolver absolute value, then correlates and calculates the IMAS absolute value. This command is used mainly for IMAS first set-up (It may also be called "Electronic adjustment").
- **Calculate IMAS absolute value:** Addresses and reads each resolver absolute value, then and calculates the IMAS absolute value. This command is used at every power-up.
- **Read IMAS absolute value:** Reports the result found after the "Calculate IMAS absolute value", or "Resolver calibration" command.
- **IMAS status:** See next page.

- **IMAS status:** Will return one of the following hexadecimal numbers:
  - **00** No measurement was performed yet.
  - **01** Drive is not disable.
  - **02** Movement while measuring.
  - **03** Measurement still in process.
  - **04** IMAS busy.
  - **05** Disturbances while reading resolvers.
  - **06** Movement after measurement.
  - **07** IMAS option is invalid.
  - **08** Calculation error.
  - **09** Illegal coarse resolver number: more than 4 or less than 0.
  - **0A** Resolver error while measurement.
  - **10** Result exists, good correlation.
  - **11** Result exists, bad correlation (Worst than 90%).

6.2 Hardware and Wiring

A DBSC that is equipped of the IMAS option has the Analog output circuitry installed. The resolver selection is achieved by affecting the resolver reference signal of a DC voltage offset. The offset value will be generated by the analog output #2. The output X1-7 must **not be wired** for any other purpose as the solder bridge SB513 is closed for the IMAS operations. The DC voltage offset is internally generated, the user does not have any access to it. The analog output #1 (pin X1-5) can be used for any other purpose.

For the IMAS operations, the solder bridge **SB513** must be closed. It shall be closed ex-factory if the DBSC was ordered with the IMAS option. For the **SB513** location, refer to the component layout of the DBSC control card in the installation manual (paragraph 6.1.1).

There is otherwise no difference in wiring. The IMAS system resolver wiring is identical to any other DBSC that does not have the multi-resolver option installed.

6.3 IMAS setting and reading

6.3.1 Definitions

- **IMAS Absolute value:**
  This is the result of the multi-resolver reading. The command **"Calculate IMAS absolute value"** has to be sent via the host communication protocol (HCP). After this operation, the command **"Read IMAS absolute value"** is also needed in order to download this value from the DBSC to the host.
- **Offset:** (Offset = IMAS - Machine absolute mechanical reference)
  This value is the one that the host will subtract to the *"IMAS absolute value"* at each power-up executed after the first setting. If the *"Machine Absolute Mechanical Reference"* is zero, then the *"Offset"* will be equal to the *"IMAS Absolute value"* when the axis is placed on it's Absolute reference point (home position for example)
- **Machine Absolute Mechanical Reference:**
  This is the mechanical reference position on which the axis must be placed for the very first setting. It is needed for the offset calculation.
- **Absolute Actual Machine Position:**
  This has to be calculated in the host. There is a correlation that is made during the calculation where the *"IMAS absolute value"* and the *"Offset"* are taken in to account. The *"Absolute actual machine position"* is valid only after a resolver reading (under disable mode, no move is then allowed). As soon as the motor has moved, this value is not valid anymore. **This is not updated with the motor move.** It is only use at any power-up or after a power cut for example. In other words, this is the result of the position calculation at any normal power-up (power-up where calibration procedure is not needed as the offset value is already known). This is also the value that the CNC will read and place in it's position register. The unit will be in bits (16 bits per motor revolution) do not forget to scale!

6.3.2 IMAS Setting

First verify the number of coarse resolvers installed in the motor (Read motor label and compare with table 6.1). Send this number to the DBSC via the "HCP".

1. Send via the HCP the **"Set number of coarse** resolvers" command.
2. Send via the HCP the **"Get number of coarse** resolvers" command and verify the answer. The value sent back should be equal to the number of resolvers set above.

6.3.3 IMAS operations

Two main operations are needed for the use of IMAS:

A. The Calibration (first set-up after tightening the motor coupling to the machine)
B. "*Absolute Actual Machine Position*" reading at any further power-up

A. The Calibration procedure:

❶ Place the axis on its zero reference position ("*Machine Absolute Mechanical Reference*"). The axis may be moved by hand (without power or in disable mode), or with a jog command, or with a home command.
❷ Activate the brake (if any) and disable the drive, and Send, via HCP, the "**Resolver calibration**" command. (Correlates the resolvers and calculate the "*IMAS Absolute Value*".).
❸ Wait 5 seconds for calculation to be completed.
❹ Send via the HCP the "**IMAS Status**" command and verify the answer. The value sent back shall be 10. If any other value is received, check what it means in paragraph 6.1.2 and correct the problem.
❺ Send via the HCP the "**Read IMAS Absolute Value**" command in order to download the value from the DBSC to the host.
❻ Host has to save this value, it will be the "*Offset*" value for further power-up.
❼ The "*Absolute Actual Machine Position*" is therefor zero at this point (as we did place the axis on zero reference position in step ❶ above).
❽ Load the value zero in to the CNC position register.
❾ Enable the drive and release the brake.

B. "Absolute Actual Machine Position" reading at any further power-up:

Then, at any further power up, the "*Absolute actual machine position*" has to be calculated and the new result is placed in the CNC position register. The Host procedure is as follows:

❶ Brake (if any) is activated and the drive disabled.
❷ Power-up the system.
❸ Send, via HCP, the "**Calculate IMAS Absolute Value**" command.
❹ Send via the HCP the "**IMAS Status**" command and verify the answer. The value sent back shall be 10. If any other value is received, check what it means in paragraph 6.1.2 and correct the problem.
❺ Wait 2.5 seconds for calculation to be completed.
❻ Send via the HCP the "**Read IMAS Absolute Value**" command in order to download the value from the DBSC to the host.
❼ Host calculates, correlates and scales the "*Absolute Actual Machine Position*":
  ◊ X = "*IMAS Absolute Value*" - "*Offset*"
  ◊ Test 1: Is 0 <= X < "*Modulo*" ?
    If yes "*Absolute Actual Machine Position*" = X,
    if no, execute test 2.
  ◊ Test 2: Is X < 0 ?
    If yes, "*Absolute Actual Machine Position*" = X + "*Modulo*"
    if no, "*Absolute Actual Machine Position*" = X - "*Modulo*"

  Where: "*Modulo*" is the maximum IMAS possible value + 1. For example, if your motor is quipped with 3 resolvers (one fine and two coarse resolvers), "*Modulo*" value is:
  16 * 16 * 65536 = 2 EXP. 24 = 16777216 (24 bits max.).

  ◊ Scale the value that will be written in the CNC position register.
  The basic idea to calculate and scale the value for the CNC position register is explained in the following example:

  - the internal DBSC scale is 16 bits per revolution (65536 counts per rev.),
  - you have 10 millimeters per motor revolutions,
  - the "*Absolute Actual Machine Position*" is 196608 counts
  - the "*Absolute Actual Machine Position*" to be written in the CNC position register is:
    (196608 / 65536) * 10 = 30 millimeters

❽ Load the "*Absolute Actual Machine Position*" in to the CNC position register.
❾ Enable the drive and release the brake.

7. PLC functions

7.1 Introduction

Note: The PLC functions are only available on the DBSC equipped with option Exx.

One of the particularity of the DBSC amplifier is it's ability to monitor inputs, outputs and other internal functions through its own PLC (Programmable Logic Controller). This permits the drive to:

- Send information via outputs to the host, the CNC, the axis motion controller board or any other I/O board.
- Receive information through inputs from the host, the CNC, the axis motion controller board or any other I/O board.
- Monitor its own internal functions.

All the functions are programmable by the user to best suite the application. For example, the safety functions of a machine or the way to stop the machine under some conditions can be achieved with the help of this PLC. The PLC features are easy to program, user friendly and intentionally limited to a certain number of choices. It is not intended to create complex PLC features as the DBSC is not to be used as an I/O board but as an amplifier. Nevertheless, it gives a lot of power to the integrator to create a very safe application and a very easy way to implement the functions.

The basic rules are very simple:

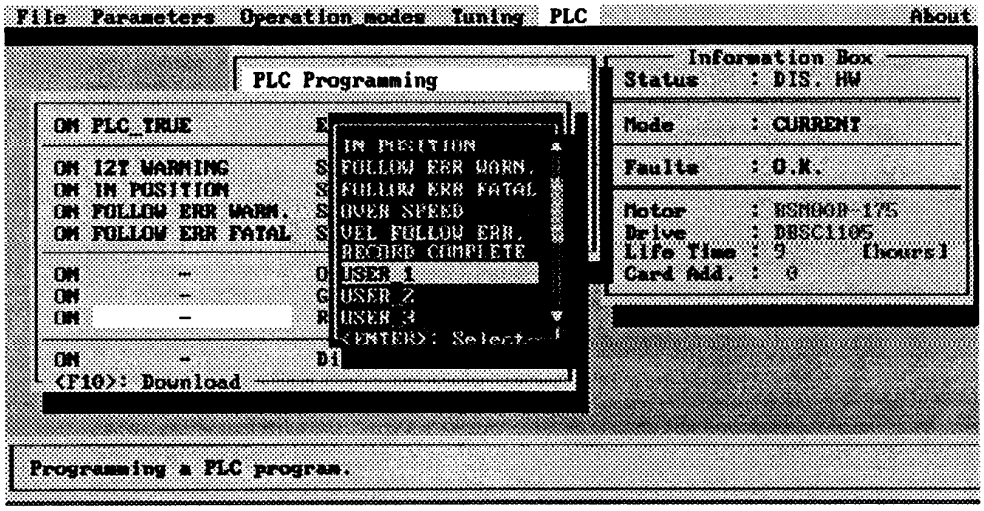
- The functions type are: **"IF Condition THEN Action"**
- Only one simple condition can start an action. (No compound conditions are possible, there is no "AND", "OR" operators.)
- One condition can initiate several actions.

7.2 PLC Program sub-menu description.

The PLC program is resident in the DBSC (Eeprom). When opening the **"PLC programming"** window, the "D3S" software reads the information directly from the drive. The PLC is divided in two parts: the actions and the conditions. When the PLC window is activated, the actions are listed on the right hand-side of the screen and the conditions on the left hand-side.

To access the PLC program, follow the procedure bellow:

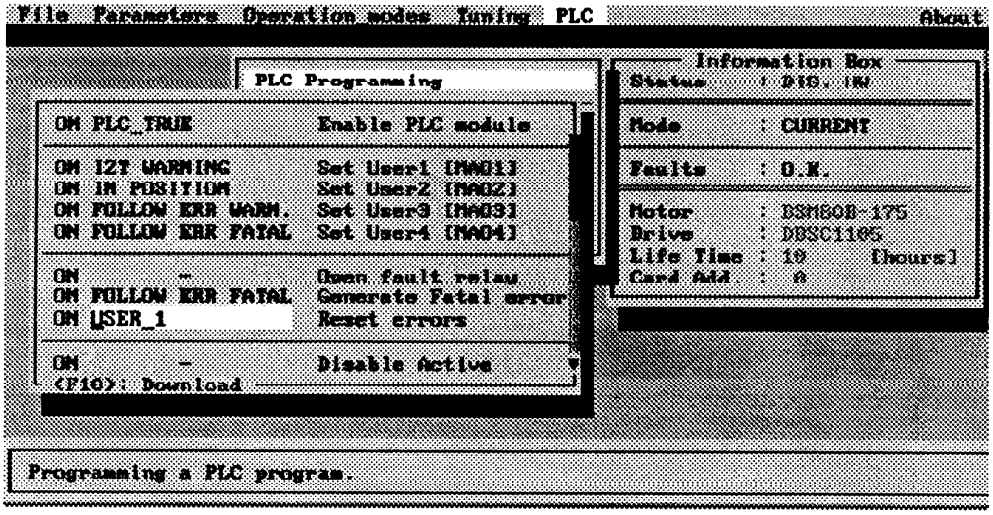
❶ Select the **"PLC"** sub-menu and press **<Enter>**.
❷ Choose the **"PLC Programming"** option and press **<Enter>**.
❸ Locate the cursor on the line of the desired action and press **<Enter>**.
❹ Select the desired condition as shown in the following window, and Press **<Enter>**.



❺ Repeat steps ❷ and ❹ until the PLC is fully programmed as needed.
❻ Press **<F10>** to download the PLC to the DBSC (If we don't want to change the PLC, press **<Esc>**).
❼ The PLC is enabled automatically or under conditional functions as set by the user. See PLC actions and Conditions in sections 7.3 and 7.4.

7.3 PLC Actions

The Actions are listed on the right hand-side of the screen as shown bellow:



All the available actions are explained bellow:

Enable PLC module: The PLC can be enabled when any of the condition listed in section 7.4 is true. If no condition is chosen ("-"), then the PLC will stay disabled. The "PLC_TRUE" condition means that PLC will always work as soon as it is downloaded (after pressing the **<F10>** key).

Set MAO1: Set output 1 (X2-15)
Set MAO2: Set output 2 (X2-16)
Set MAO3: Set output 3 (X2-17)
Set MAO4: Set output 4 (X2-18)

Open Fault Relay: The fault relay can be opened by the PLC. In any case, an internal error that requires to open this relay will take priority over the PLC. However, the user has the freedom to activate this flag under any fault condition such as a fatal following error which has occurred in the CNC for example.

Generate Fatal Error: The fatal error can be generated by the PLC. In any case, an internal error that requires to set the Fatal Error will take priority over the PLC. However, the user has the freedom to activate this flag under any fault condition such as a fatal following error which has occurred in the CNC for example.
The drive is then disabled, the red LED is lighted, the display shows error "**9**". This error is latched, and a "**Reset errors**" needs to be generated by the PLC or by the "D3S" software or by recycling power.

Reset Errors: This allows to reset of the errors shown on the display that are listed bellow:

Error 0:	Watch dog
Error 1:	Bus over voltage, BPS not ready
Error 4:	+/- 15 VDC Over and under voltage
Error 5:	Resolver fault
Error 6:	Electronic fuse.
Error 9:	General Error

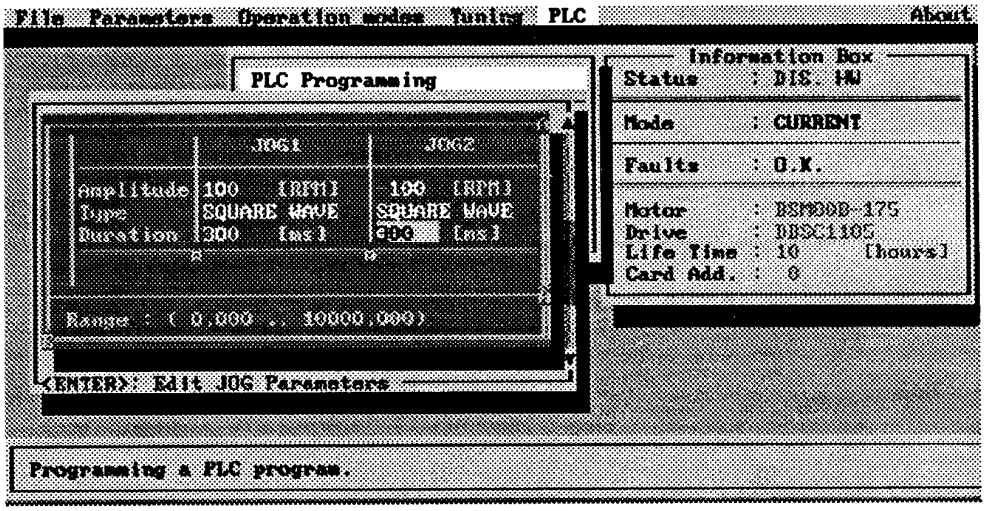
(See also appendices D3 and E)

Disable Active: Select the "**Active disable**" mode (see paragraph 8.4.2)
Disable Passive: Select the "**Passive disable**" mode (see paragraph 8.4.1)
Hold Position: Will cause the motor to hold its position, the analog input is ignored. (see the section 8.2)

Select Jog2:

Two different jog settings can be programmed as explained bellow and shown in the screen following screen:

- Select the **"PLC"** sub-menu and press **<Enter>**.
- Choose the **"PLC Programming"** option and press **<Enter>**.
- Locate the cursor on the "Jog parameters" option and press **<Enter>**.
- With the arrow keys, select the desired parameter, then modify the amplitude and the duration values as required.
- With the arrow keys and space bar, select the desired jog type (CONTINUOUS, or INCREMENTAL, or SQUARE WAVE) as required.
- Press **<Esc>** to move out of the Jog setting window.
- Press **<F10>** to download the PLC to the amplifier.



When "Select Jog2" is active, the Jog function is executed with the parameters listed in Jog2. In the other case, the Jog function is executed with the parameters listed in Jog1

Remarks:

When selecting the **"SQUARE WAVE"** JOG1 and JOG2 are used in the same function. Selecting the Jog 2 mode does not make sense any more.

The **"SQUARE WAVE"**, as shown in the above example, will generate a move of 100 RPM (clock wise direction) for 300 ms and a second move of -100 RPM (counter clock wise direction) for another 300 ms. This function can be very useful for testing the step response of the system as the PLC can also gather the data. Those data can later be shown on the screen by selecting the **"Execute Tuning"** sub-menu and pressing the **<F11>** key.

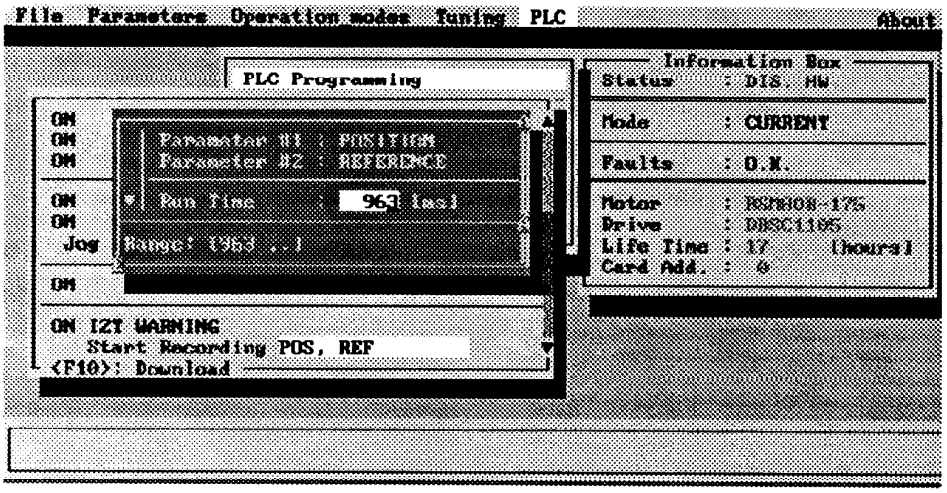
After changing the Jog1 and Jog2 parameters, press the **<Esc>** key to move out of the "Jog parameters" window. Re-activating this window before downloading the PLC to the amplifier with the **<F10>** key, will cause the values which were entered just before to be lost. This is because the "jog parameters" values are read from the DBSC at each activation of the Jog window.

Start Jog: PLC Jog command. If true, jog is executed; if false, jog is stopped.

Increase Gear by n: (The factor "n" can be 2, 4 or 8 and is selectable)
If true, the hand-wheel gear ratio is multiplied by a factor of "n".
If false, the hand-wheel gear ratio is reset to the original value.
Note that there is no ramp, therefor the change acts like a step. Exercise caution!

Start Recording: Will start gathering the selected parameters (position and the reference for example). To select what to record and the recording time, proceed as follows:

1. Select the "PLC" sub-menu and press <Enter>.
2. Choose the "PLC Programming" option and press <Enter>.
3. Locate the cursor next to "Start Recording" and press <Enter>.
4. Locate the cursor on the "Parameter #1", press <Enter> select the desired variable and press <Enter> once more.
5. Repeat point 4 for the "Parameter #2".
6. Locate the cursor on the "Run Time" and type in the desired recording time as shown bellow. (Range is 963 ms to 99228 ms)
7. Press <Esc> to move out of the recording setting window.
8. Press <F10> to download the PLC to the amplifier.



To view the data, select the "Execute Tuning" sub-menu and press the <F11> key.

Note:
The number of samples to be gathered is always 2048. Therefore, the time between two samples is dependent of the chosen "Run time" value. The recording time will always be at least the "Run Time" specified. For example, if the selected "Run Time" value is 1.2 second, the gathering time will last 1.927 second. This is calculated as follow:

$$RT < GT = K * SC$$

Where:
- RT The Run time
- GT The Gathering time
- K The number of samples (constant = 2048)
- SC The servo-sample gathering period (always a multiple of the servo cycle time: 470.4, 940.8 etc. microseconds)

In our example:

1.2 / 0.0004704 = 2551 samples; this number is too large, therefore SC is chosen larger (470.4 * 2 = 940.8 microseconds).

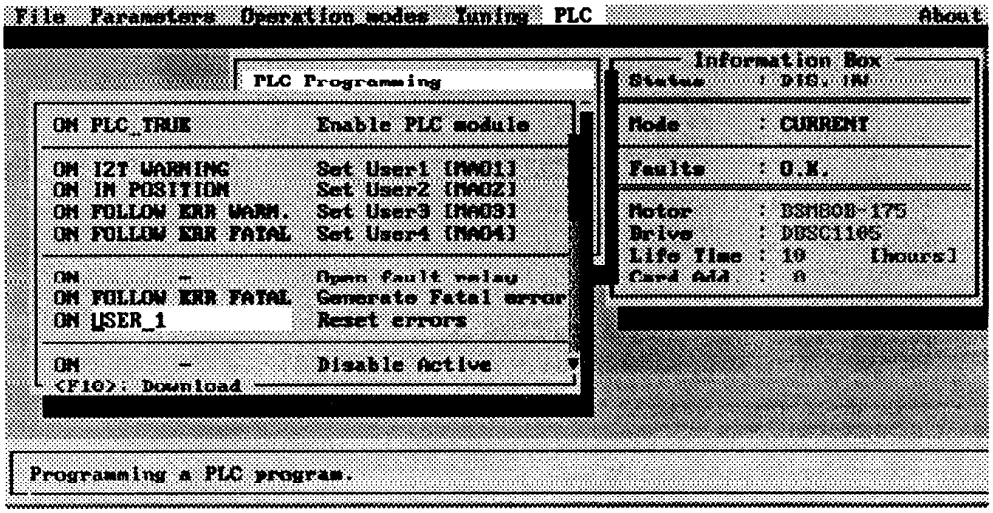
Afterward: GT = 2048 * 0.0009408 = 1.927 seconds.

Note that if the PC and the PLC decide to record at the same time, the PC has the priority.

After a gathering operation commanded from the PLC, the data can be visualized on the screen. Select the "Tuning" menu, followed by the "Execute tuning" option and then press the <F11> key.

7.4 PLC conditions

The conditions are listed on the left hand-side of the screen as shown bellow:



If a condition is true, the action is set; in the other case, the action is reset. As shown in the section 7.2, all the available conditions are listed on the screen when the **<Enter>** key is pressed. They are all explained bellow:

-: This is the "always false" state. When so defined, the selected action is disabled.

CW WARNING: If the Clock Wise limit switch is activated, the selected action is executed.

CCW WARNING: If the Counter Clock Wise limit switch is activated, the selected action is executed.

FOLLOW ERR WARN: In Hand-Wheel mode, if the Warning Position Following Error is reached, the selected action is executed.

FOLLOW ERR FATAL: In Hand-Wheel mode, if the Fatal Position Following Error is reached, the selected action is executed.

OVER SPEED: Not applicable

VEL FOLLOW ERR.: Not applicable

RECORD COMPLETE: Not applicable

USER_1 (MAI1): If Input 1 is active, the selected action is executed.

USER_2 (MAI2): If Input 2 is active, the selected action is executed.

USER_3 (MAI3): If Input 3 is active, the selected action is executed.

USER_4 (MAI4): If Input 4 is active, the selected action is executed.

USER_1 NOT (MAI1/): If Input 1 is not active, the selected action is executed.

USER_1 NOT (MAI2/): If Input 2 is not active, the selected action is executed.

USER_1 NOT (MAI3/): If Input 3 is not active, the selected action is executed.

USER_1 NOT (MAI4/): If Input 4 is not active, the selected action is executed.

DRIVE OVER TEMP: If the Drive Over-temperature signal appears (82 °C), the selected action is executed (Also see errors 6 and 7 in appendix E).

MOTOR OVER TEMP: If the Motor Over-temperature switch opens, the selected action is executed (Also see errors 6 and 7 in appendix E).

I2T WARNING: If the "I^2t" limit is reached, the selected action is executed (Also see errors 6 and 7 in appendix E).

PLC_TRUE: As an example, this condition can be used on the line pointing to the "Enable PLC Module" action (first line of the PLC actions list). If selected, the PLC is always enabled.

7.5 PLC Factory default

When a DBSC is delivered, there is a PLC program which is downloaded from factory and always active. It only sets the output 1 when the "I2t" limit is reached and the output 2 when "In position".

If no PLC program is needed, deactivate it by selecting the **"always false"** condition on the line of "Enable PLC Module" action. In any other case, program the PLC as required by the application.

7.6 Saving a PLC program in to a file

Not applicable. The option **"EEPROM ⟩⟩⟩ PLC PROGRAM"** in the **"PLC"** sub menu is not functional; use instead the "File" menu. Refer to section 8.3.

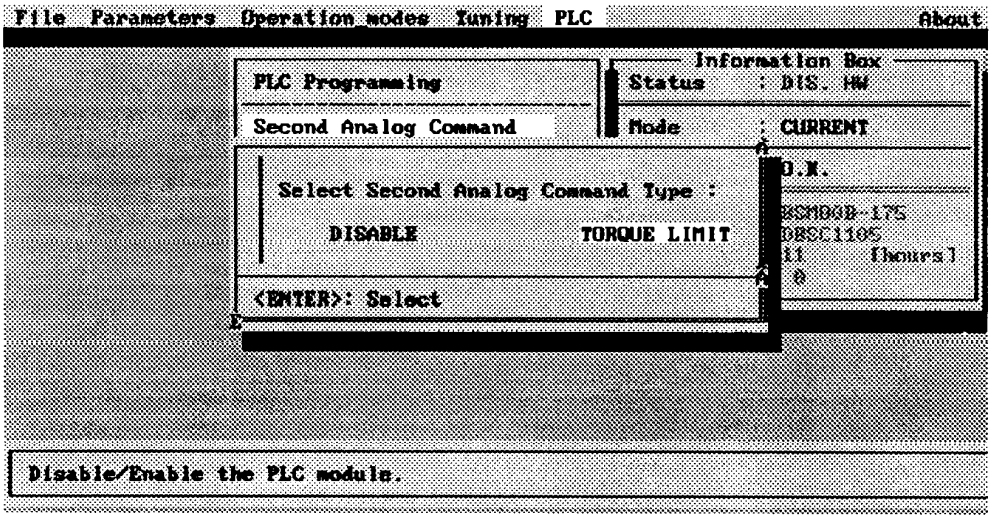
7.7 Duplicating an existing PLC program (memorized on the PC hard disk)

Not applicable. The option **"PLC PROGRAM ⟩⟩⟩ EEPROM"** in the **"PLC"** sub menu is not functional; use instead the "File" menu. Refer to section 4.6.

7.8 Second analog input command

The second analog input is used as a torque limitation. When enabled, the current will be limited proportionally to the voltage seen between the CMD+ (X1-3) and CMD- (X1-4) terminals. A 10 VDC value allows the drive to develop it's full peak current. For example, an 8 VDC value limits the current to 80 % of the amplifier peak current. The polarity is not significant (+8V or -8V will have the same effect on both directions). The second analog input acts on the "maximum current block" as shown in figure 9.1. To enable or disable the torque limitation, proceed as follows:

❶ Select the **"PLC"** sub-menu and press **<Enter>**.
❷ Choose the **"Second Analog Command"** option and press **<Enter>**.
❸ Locate the cursor on the line of the desired action (TORQUE LIMIT or DISABLE) and press **<Enter>**.



8. Additional functions

8.1 Introduction

This chapter explains the DBSC functions which can be activated and deactivated from the D3S software.

8.2 Hot keys

Several Hot keys are available in the D3S software. The Hot Keys enable quick operation of important options and ensure maximum system safety, as the higher priority is kept for the disable command. The available Hot Keys are listed below :

<F1>: Help

A Help window appears with the relevant information regarding the option that is pointed by the cursor at the moment (Context Sensitive Help).

<F5>: Disable / Enable

When **<F5>** is pressed, the amplifier passes to software disable mode. The disable type may be passive or active (controlled braking or immediate power stage disabling). It depends on the user selection which is explained in the section 8.4. A second stroke on the **<F5>** key will re-enable the drive.

<F6>: Hold / Release Position

When **<F6>** is pressed, the motor decelerate, stops, and holds position. When the amplifier is used in current mode or velocity mode, this operation turns on the position mode and reject any command comming from the analog input. When the "Hand Wheel" mode is selected, The Hand Wheel input or the Pulse Follower signal is rejected. A second stroke on **<F6>** will Release the system to its original state.

<F7>: Jog

This hot key opens the jog window which give access to the options "continuous Jog", "Incremental Jog" and "Jog direction". Once selected, the jog move can be started by pressing **<Enter>**. This function is permitted only if the drive is operated in velocity mode or position mode (See also the section 8.6).

<F8>: View Variables

This option displays, in real time, 2 variables that have been previously selected in the "Variable Definitions" menu, for example the velocity and the current (see paragraph 8.7.1). A second stroke of the key causes the variables to disappear from the screen.

8.3 Save configuration in a file (content of the personality module)

The Eeprom of the DBSC is used as a personality module. The system definition and parameters, as well as the PLC program if the option Exx is present, can be read from the Eeprom and saved on the hard disk. This file can be downloaded to another DBSC.

In order to save the DBSC configuration, perform the following steps:

1. Disable the drive (use **<F5>**, or the disable switch AS8, or X2-2)
2. Select the sub-menu **"File"** and press **<Enter>**.
3. Choose the **"EEPROM ⟩⟩⟩ File"** option and press **<Enter>**.
4. Specify the file name **(*.bin)** which will contain the configuration, or press the "?" key to see a list of available files.
5. Press **<F10>** to save the personality module data.
6. Re-enable the drive.

8.4 Disable modes

The disable command can be generated via the keyboard (**<F5>** key), the hardware input (X2-2), the dip switch (AS8) or the PLC.

Two disable modes are programmable: the **active** disable and the **passive** disable. The two modes are explained bellow:

8.4.1 Passive disable:

This is the default mode. On disable (software or hardware), the power stage is deactivated. In this case, the motor does not produce any torque, and the load may keep moving due to its inertia.

The passive disable mode is selected as follows:

❶ Select the sub-menu **"Operation_modes"** and press **<Enter>**.
❷ Choose the **"Disable Type"** option and press **<Enter>**.
❸ Locate the cursor on **"Passive"** and press **<Enter>**.

8.4.2 Active disable:

On disable (software or hardware), the power stage remains activated, the motor decelerate in a controlled manner until its velocity reaches zero. When the amplifier is used in current mode, this disable type activates automatically the velocity mode in order to be able to control the deceleration. Once the motor is stopped, then the power stage is disabled.

This mode is in fact equivalent to a "Hold" command with, in addition, the deactivation of the power stage when the motor velocity reaches zero.

The active disable mode is selected as follows:

❶ Select the sub-menu **"Operation_modes"** and press **<Enter>**.
❷ Choose the **"Disable Type"** option and press **<Enter>**.
❸ Locate the cursor on **"Active"** and press **<Enter>**.

Remark:

In vertical axis applications, in order to avoid the fall of the axis, it is strongly recommended to activate the "Hold" input instead of using the active or passive disable. After the complete stop of the motor the brake may be activated and only then the disable command can be sent. Note that such a function can be programmed directly to the PLC of the DBSC, if the option Exx is present.

8.5 Control mode selection (Current, Velocity, Position / Hand wheel)

The DBSC can be used either in current mode in velocity mode or in position mode (hand wheel). the user has to select the mode required for the application.

Remarks:

- **It is necessary to tune the velocity loop even if the DBSC is used in current mode. This is because the velocity loop is automatically closed when "Hold position" is commanded or when "active disable" or "CW" and "CCW" limits conditions occur.**
- **When the amplifier operates in current mode, a velocity loop must be closed externally to the DBSC. It will be part of the position control system (CNC, position controller card etc.) The position controller must be able to calculate the actual velocity by derivation of the position feedback.**
  When the DBSC operates in velocity mode, the position controller treats only the position loop and sends a velocity command to the amplifier. (The derivative gain of the position loop in the CNC or motion card must be set to zero).
- It is recommended to use the DBSC in velocity or position mode (Hand wheel). This will guaranty the best performance as well as a perfect synchronization between the velocity loop and the current loop. The velocity loop tuning procedures and the optimization is described in chapter 5.

8.5.1 Current Mode:

The basic mode of the DBSC is the current mode in which the current injected to the motor winding (or the torque applied to the load) is proportional to the analog input command received by the DBSC.It follows a current command given by the user via the analog command input. A 10V analog command input, corresponds with the amplifier specified application peak current (see paragraph 5.3.1).

To select the **current mode**, follow the steps below:

1. Select the sub-menu **"Operation_modes"** and press **<Enter>**.
2. Choose the **"Control"** option and press **<Enter>**.
3. Locate the cursor on **"Current"** and press **<Enter>**.

8.5.2 Velocity mode:

In the velocity mode, the voltage applied to the motor winding (or the velocity of the motor) is proportional to the analog input command received by the DBSC. Also, the velocity controller output signal is then the current input command to the current controller. The two modes are therefor activated. A 10V analog command input, corresponds with the maximum velocity specified by the user (see paragraph 5.3.5).

To select the **velocity mode**, follow the steps below:

1. Select the sub-menu **"Operation_modes"** and press **<Enter>**.
2. Choose the **"Control"** option and press **<Enter>**.
3. Locate the cursor on **"Velocity"** and press **<Enter>**.

8.5.3 Position mode (Hand Wheel):

This mode is used to operate the hand wheel. The DBSC performs all three regulation functions (position, velocity, current) and becomes a single axis system capable to follow an external reference signal received by an incremental encoder.

To select the **position mode**, follow the steps below:

1. Select the sub-menu **"Operation_modes"** and press **<Enter>**.
2. Choose the **"Control"** option and press **<Enter>**.
3. Locate the cursor on **"Hand-Wheel"** and press **<Enter>**.

8.6 Jog

This function is available either via **<F7>** key, or the **"Operation_modes"** menu, or the PLC. This very useful command permits to rotate the motor shaft without the need of the external analog input. An internal digital velocity command is then issued by the DBSC itself. This function can be activated only in the **"velocity"** and **"position"** modes. Via the D3S, it is executed by selecting the sub-menu "Jog". The **"Jog"** mode selection initiates a command equivalent to **"Hold"**. The parameters definition for this mode is done as follows:

1. Select the sub-menu **"Operation_modes"** and press **<Enter>**.
2. Choose the **"Jog"** option, and press **<Enter>**. Alternatively, press **<F7>**.
3. Select the **"amplitude"** (jog velocity in RPM), **"Duration"** (in seconds), the **type**, and the **direction** of the move (CW or CCW).
4. Press **<Enter>** to activate the command.
5. To stop, locate the cursor on **"STOP"**, and press **<Enter>**
6. Re-enable the drive.

Leaving the "Jog" menu automatically deactivate this mode, and the DBSC responds then to the external analog input command.

8.7 The Tuning module (for the offset compensation and servo-loops tuning)

The "Tuning" module is a tool which allows the user to analyze the system performances. A move command or a step can be sent and the response (velocity, current, position etc.) can be visualized on the screen either graphically, or numerically. The choice of the variables to be displayed on the screen in real time and the hand wheel status reset are also done by this module. The **"Tuning"** sub-menu is composed of the following options:

- Variables Definitions (Choice of the variables to be visualized)
- Execute Tuning
- Offset Tuning
- Print Data
- Reset Parameters (Reset hand wheel and fault status)
- View graph

8.7.1 Choice of the variables to be visualized

The **"variable definitions"** option allows to choose and select two variables from a given list. Using the **<F8>** function key, the user will be able to display in real time, numerically, those variables on the screen (see section 8.2). The following variables can be selected:

- Position
- Following error
- Velocity
- Current U (current in phase U)
- Current V (current in phase V)
- Reference (Actual command, analog input or internal reference, depends on operation mode)
- Current (Current command)

The choice is done as follows:

❶ Select the sub-menu **"Tuning"** and press **<Enter>**.
❷ Choose the **"Variable Definitions"** option and press **<Enter>**.
❸ Locate the cursor on the **"parameter #1"** and press **<Enter>**, the parameters list is displayed.
❹ Select the variable to be visualized and press **<Enter>**.
❺ Repeat this procedure for **"parameter #2"**.

Note:
The equivalent current command "CURRENT", and the phase current feedback "CURRENT U" and "CURRENT V" **are true values (not RMS).** For example, a 5 Amps DBSC is able to deliver 10 A peak RMS that are 14.1 A peak true value.

8.7.2 "Execute Tuning"

This option allows to perform a data gathering in real time of the above selected variables. Up to 2000 data can be memorized to be later displayed on the screen. The servo loops tuning examples seen in sections 5.4 and 5.5 were issued with this option. Three possibilities are given for the data gathering via the **<F9> <F10> and <F11>**.

- The **<F9>** key issues an internal velocity step command. Its amplitude and duration are defined by the operator. This function is comparable to the "Jog", the velocity mode must therefor be selected. The data is automatically memorized in the RAM device while the step command is being executed, and is sent to the PC via the communication port. The velocity step response can be displayed on the screen. For example, it is useful for the velocity loop tuning as described in paragraph 5.4.1.1.
- The **<F10>** key automatically memorizes the data in the RAM device and sends it to the PC via the communication port. This make possible to visualize on the screen the response of the system to the external analog input or hand-wheel command. This function can be activated at all time to analyze the system behavior during a production phase for example. It is usable in position, velocity and current modes.
- The **<F11>** key is used to upload the data gathered upon a PLC command from the RAM to the PC. Refer to the PLC functions in section 7.3 for details.

8.7.3 "Offset Tuning"

Section 5.2 (points B and C) describes the use of this option.

8.7.4 "Print Data"

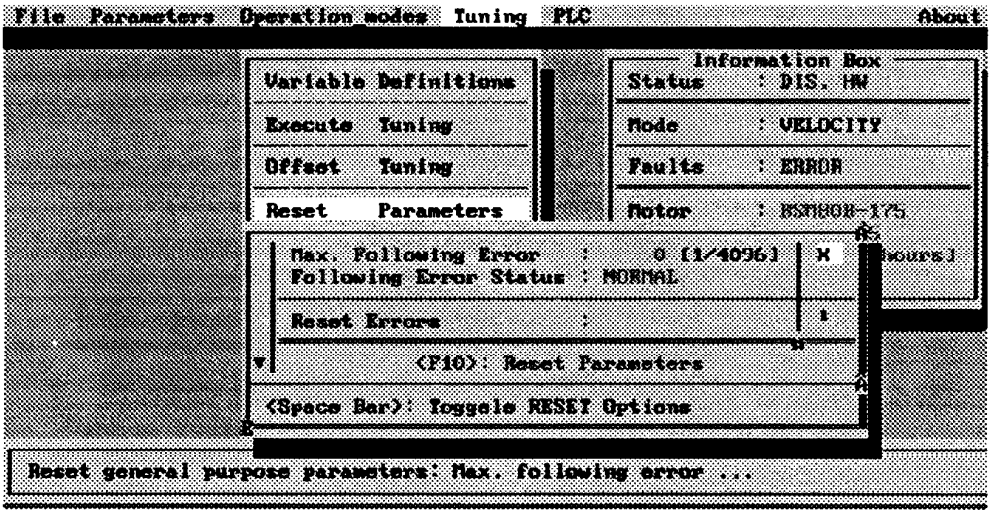
"Print Data" saves the recorded data in to a **"Text"** file (*.txt). The following steps are needed:

1. Select the **"Tuning"** sub-menu and press **<Enter>**.
2. Choose the **"Print Data"** option and press **<Enter>**.
3. specify the file name (*.txt) which will contain the data (in ASCII format).
4. Press **<F10>**.

8.7.5 "Reset Parameters" (Reset the hand-wheel and the error status)

Several errors that may appear on the display can be reset through this function. Also, when the **"Hand wheel"** operation mode is in use, the **"Following error"** status and **"max. value"** shown in the information box can be reset. Proceed as follows:

1. Select the **"Tuning"** sub-menu and press **<Enter>**.
2. Choose the **"Reset Parameters"** option and press **<Enter>**.
3. Locate the cursor on the desired option and select it by pressing the space bar. The options marked with the symbol "1" will be reset and those marked with an "x" will not be reset.
4. Press **<F10>** to execute the reset command.



The "Reset Errors" option can clear several latched errors that may appear on the display (Errors 1, 4, 5, 6 and 9).

8.7.6 "View Graph"

"View Graph" allows the user to view graphically the data that has been collected during the **"Execute Tuning"** operation seen in the paragraph 8.7.2. This option offers the great advantage to replace the need of an oscilloscope, The transient phenomena can be gathered and analyzed accurately by using the zooming. The "View Graph" option operates a graphic module which is external to D3S. The graphic module is described in section 8.8.

8.8 Graphic Module

This module is activated via the **"View Graph"** option. A mouse may be used, it facilitates the access to the menus and sub-menus.
The graphic screen is divided into three parts:

A. "Information box"
B. "Plots box"
C. "Help box"

A. "Information box"

It is located on the right side of the screen and contains the following information:

- Cursor position on horizontal (X) axis (or time axis).
- Title of each plot.
- Color and line color of each plot.
- Vertical (Y) cursor position on each plot.

↓ key (arrow down) closes the information box.
↑ key (arrow up) opens the information box.

B. "Plots box"

This is the main part of the graphic screen. It displays up to 2 selected plots on one X-Y axis coordinate system.

C. "Help box"

It is located on top of the screen. It displays a brief overlook of the hot keys that are accessible while using the graphic module. Refer to paragraphs 8.8.1 to 8.8.5 bellow, which describe the functions that are available via the keyboard (or mouse).

8.8.1 Cursor movement hot keys

"F" Fast:	increments the cursor velocity by a factor of 2 (max. value: 1024).
"S" Slow:	decrements the cursor velocity by a factor of 2 (min. value: 1).
→ (Right arrow):	moves the cursor to the right.
← (Left arrow):	moves the cursor to the left.
<shift> + →:	moves the cursor to the right with a velocity increased by a factor of 10.
<shift> + ←:	moves the cursor to the left with a velocity increased by a factor of 10.
<End> :	moves the cursor to the end of the data (right hand side of the screen).
<Home>:	moves the cursor to the beginning of the data (left hand side of the screen).

8.8.2 Information box hot keys

↓ (arrow down):	Closes the information box
↑ (arrow up):	Opens the information box

8.8.3. "Zoom" hot keys

The key "Z" (Zoom) opens a new window allowing the zooming of the data shown on the screen. The <Esc> key closes the "Zoom" window. The keys "W", "X", "Y", "M" explained bellow define the zooming options:

- W ("Window"): Zooming defined between two opposite corners of a rectangle:

  A star "*" appears in a the newly opened window. Move this star up, down left or right as wanted by using the arrow keys, place it on the first desired corner and press <Enter> to select. Repeat the operation once more in order to select the second corner. As soon as the <Enter> key is pressed, the zoom is executed. Press the "U" key (Unzoom) in order to resume the graph to it's original dimensions.

  Description of the keys which may be needed during zooming operation:

  - → / ← / ↑ / ↓ (Arrows left / right / up / down) for the cursor movement
  - <Home> / <END> to place the cursor to the left or right of the screen
  - <Enter> to select the corners
  - <Esc> to move out of the "Zoom" command

- X: Horizontal zooming

  The "X" key zooms the data on the abscissa axis only. Locate the cursor on the first desired limit by using the left and right arrow keys or / and the <Home> and <END> keys. Then press <Enter>. Repeat the operation once more in order to select the second limit. As soon as the <Enter> key is pressed, the zoom is executed. Press the "U" key (Unzoom) in order to resume the graph to it's original dimensions.

  Description of the keys which may be needed during this operation:

  - → / ← (Arrows left / right) for the cursor movement
  - <Home> / <END> to place the cursor to the left or the right of the screen
  - <Enter> to select the limits
  - <Esc> to cancel the "X" command

- Y: Vertical zooming

  The "Y" key zooms the data on the vertical axis only. Locate the cursor on the first desired limit by using the up and down arrow keys or / and the <Home> and <END> keys. Then press <Enter>. Repeat the operation once more in order to select the second limit. As soon as the <Enter> key is pressed, the zoom is executed. Press the "U" key (Unzoom) in order to resume the graph to it's original dimensions.

  Description of the keys which may be needed during this operation:

  - ↑ / ↓ (Arrows up / down) for the cursor movement
  - <Home> / <END> to place the cursor to the top or the bottom of the screen
  - <Enter> to select the limits
  - <Esc> to cancel the "Y" command

- M ("Manual"): Manual zooming

  The "M" key (Manual) zooms the data according to the limits that are defined by the user. Select as you please the "X", "H", "L", "Y" keys in order to edit the limits, enter their values via the numerical keys and press <Enter>. Once the limits are selected, press <ESC> for zooming. Press the "U" key (Unzoom) in order to resume the graph to it's original dimensions.

  Description of the keys which may be needed during this operation

  "X": edit lower limit in horizontal.
  "H": edit upper limit in horizontal.
  "L": edit lower limit in vertical.
  "Y": edit upper limit in vertical.
  <Enter>: save the defined limits.
  <Esc>: execute zooming.

- Miscellaneous keys

  The following keys can also be used within the **"Zoom"** window:

  - **"F"** (Fast)
  - **"S"** (Slow)
  - → (right arrow)
  - ← (left arrow)
  - ↑ (up arrow)
  - ↓ (down arrow)
  - **"G"** (Authorize the continuous updating of the information box according the mouse position.)

8.8.4. Text editing keys

The **"T"** (Text) key activates the text edition, the following options are available:

- **"Font":**

  The **"F"** (Font) key is used to activate the choice of the characters size. Four sizes can be selected with the numerical keys **"1"** to **"4"**.

- **"Color":**

  The **"C"** (Color) key is used to activate the choice of the text color. The color is selected via the keys **"1"** to **"9"** and **"A"** to **"F"** (Sixteen colors possible).

- **"cleaR":**

  The **"R"** (cleaR) key will clear all the text from the graph.

- **"Add":**

  The **"A"** (Add) key allows the user to add a new text on the graph. A cursor is shown on the screen; move it to the top left desired text location and press **<Enter>**. The text can then be written, it will have the size and the color selected above. The **<ESC>** key will quit the text writing.

  The following keys are available during this operation:

  - **<Home>**, **<CTRL>** + **<Home>**
  - **<End>**, **<CTRL>** + **<End>**
  - → (right arrow), ← (left arrow)
  - ↑ (arrow up), ↓ (arrow down)
  - **<Insert>**, **<Delete>**, **<SHIFT>** + **<Delete>**
  - **<Enter>**, **<Backspace>**

- **"Edit":**

  The **"E"** (Edit) key is used to re-edit an existing text. A cursor is shown on the screen; locate the cursor on the text which shall be modified and press **<Enter>**. The text can then be re-written, it will have the size and the color selected above. The **<ESC>** key will quit the text writing.

  The other keys which are available during the "Edit" operation are the same as in the "Add" operation (see above).

- **"Move":**

  The **"M"** (Move) key is used to re-locate an existing text. A cursor is shown on the screen; locate the cursor on the text which shall be moved and press **<Enter>**. Then, move the cursor to the desired new location and press **<Enter>** once more.

- **"Save":**

  The **"S"** (Save) key is used to save the written text in a file. Press the **"F"** (File) key, edit the file name and press **<Enter>** when finished. Then press **<Esc>** to execute the save operation.

- **"Load":**

  The **"L"** (Load) key is used to load a text from the disk to the screen. Press the **"F"** (File) key, write the file name and press **<Enter>** when finished. Then press **<Esc>** to execute the load operation.

- **"Print":**

  The **"P"** (Print) key will print the screen content. This function uses the PC's resident printing program.

The **<Esc>** key exits the **"Text"** editing command and returns to the main menu.

8.8.5 Miscellaneous keys:

"Reset": The **"R"** (Reset) key refreshes the screen to it's initial state.

"Gin": The **"G"** key updates continuously the values contained in the information box as a function of the mouse position on the screen. In other words, the values contained in the information box are always corresponding to the cursor position on the graph.

"seL": The **"L"** (seLect) key permits the visualization of any graph accessible within the open data file. Each graph can be placed in one of the three following states:

- **"Hide":** The graph is not displayed.
- **"Show":** The graph is displayed without the information box data.
- **"Active":** The graph is displayed with the information box data.

"Print": The **"P"** (Print) key sends the content of the screen to the printer. This function uses the PC's resident printing program.

"Quit": The **"Q"** (Quit) key is used to quit the graphic module and returns to the D3S main screen.

9. DCP (Digital Control Platform) theory of operation

9.1 Introduction

This chapter talks about the current, velocity and position loops. It also explains how the limit switches algorithm works. This last point is particularly important and it is recommended to read it carefully.

9.2 System structure

The system includes three control loops: **current, velocity and position.**
Three operational modes are possible: **Current (Torque), Voltage (Velocity)** and **Hand-wheel (Position)** modes.

In Current mode (Torque), only the current loop is enabled. The current reference is sent via the analog input. The offset compensation is automatically executed.

In Voltage mode (Velocity), the current and the velocity loops are enabled. The velocity reference can be received either via the analog input, or via the serial port in a digital manner. An internal "Jog" command is also possible.

In Hand-wheel mode (position), the three control loops are enabled. The DBSC tracks the pulses received from the hand-wheel with an electronic gear ratio which is programmable by the user. The gear ratio is represented in floating point and it can be either positive or negative.

Further more, the three operational modes are taking in consideration the saturation levels between servo-loops which are equipped with anti-windup algorithms. The system safety is assured by other algorithms that look after the limit switches, the over temperature situations, the over current etc.

Figure 9.1 shows the complete structure of the system and its three servo-loops. (All the selection switches represented in the figure are internal software switches).

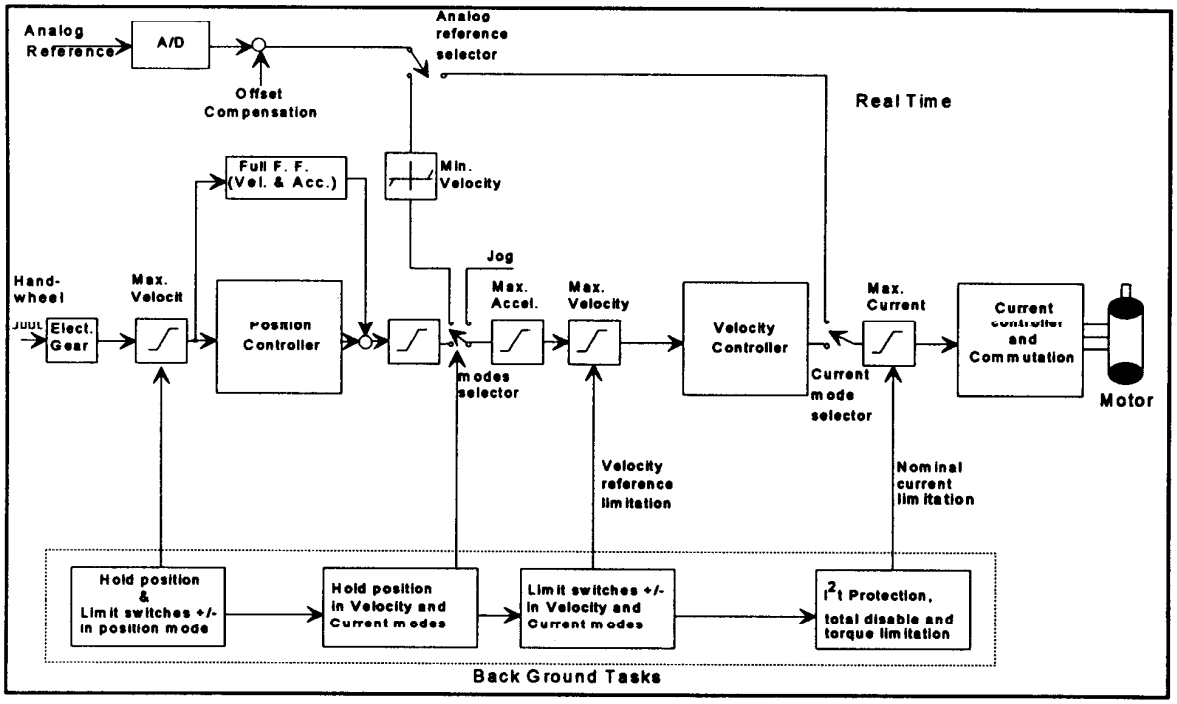


Figure 9.1: DBSC global structure

9.3 Current control (torque)

The fundamental particularity of the DBSC is its **digital current loop**. (Most digital amplifiers existing on the market are not fully digitized as they still use an analog current loop). The current command value which is sent to the winding phases is based on the angular position of the rotor related to the stator; the angle value is given by the resolver. Thus, the current command of each phase is a sinusoidal signal, with a frequency that depends on the number of pole pairs in the motor, and varies continuously with the motor velocity. The current control of each motor phase requires to be perfectly synchronized with the other phases; it must also have a high dynamic response in order to minimize the delay of the current loop. The DBSC performs the current control, digitally, with high performance.

The basic mode of the drive is the Current mode. The current injected in the motor winding (or torque produced at the motor shaft) is proportional to the analog reference command sent to the drive. A peak command of 10 volts is equivalent to the peak current specified for the drive (see paragraph 5.3.1). When the current mode is enabled, a velocity loop must be closed externally to the DBSC. This loop is then included in the positioning system (axis control card, CNC etc.) that has to be able to calculate the actual velocity by differentiating its position feedback.

The coefficients of the current controller are determined automatically by the "D3S" software as a function of the motor and drive specifications that were selected in the "parameters" sub-menu (see section 4.7). The current loop is thus specifically defined for each combination of motor and drive. It ensures a highly efficient response but needs an accurate system definition as seen in the section 4.7. Nevertheless, the tests have shown that the current control is very robust, stability and acceptable performance are ensured even with approximated parameters. The current controller, which has a polynomial structure, is equipped with anti-windup algorithm that protects the control loop from overshoots and instability situations in case of saturation of the PWM (Pulse Width Modulation) command. The current loop tuning, which is done by the "D3S" software, uses the pole placement method.

Even if the current controller is entirely defined by the "D3S" software, it can be useful to change some of its characteristics. The access to the current loop is possible through the parameters situated in the **"Current controller"** sub-menu. They are listed bellow:

- "Band Width"
- "Phase advance"
- "Application peak current"
- "Application nominal current"

9.3.1 Band Width

The Band Width is not alterable, it is always 1000 Hz.

9.3.2 Phase Advance

This parameter affects the current command. The current sent to each phase is affected by a lead which is proportional to the motor velocity times the phase advance parameter value. In other words, the current command of each phase , which is a function of the rotor position related to the stator, is sent in advance. The resolver gives the rotor position information. The amount of advance is proportional to the velocity times the phase advance value. This parameter is used to compensate the current loop equivalent delay and that should minimize the current in the winding for a produced torque at any speed.

The system performances degrade if the phase advance value is higher or lower than necessary. This is especially noticeable at high velocity where the current injected to the motor may become much higher than the one that produces the torque. Thus such situation leads to reduced efficiency as the electrical power becomes much greater than the mechanical power produced at the motor shaft output. The excess of electrical power produces heat in the motor winding. It is therefor important to find the optimal phase advance value. Refer to the paragraph 5.5.1 for the phase advance optimization method.

9.3.3 Application peak and nominal current

Refer to paragraph 5.3.1.

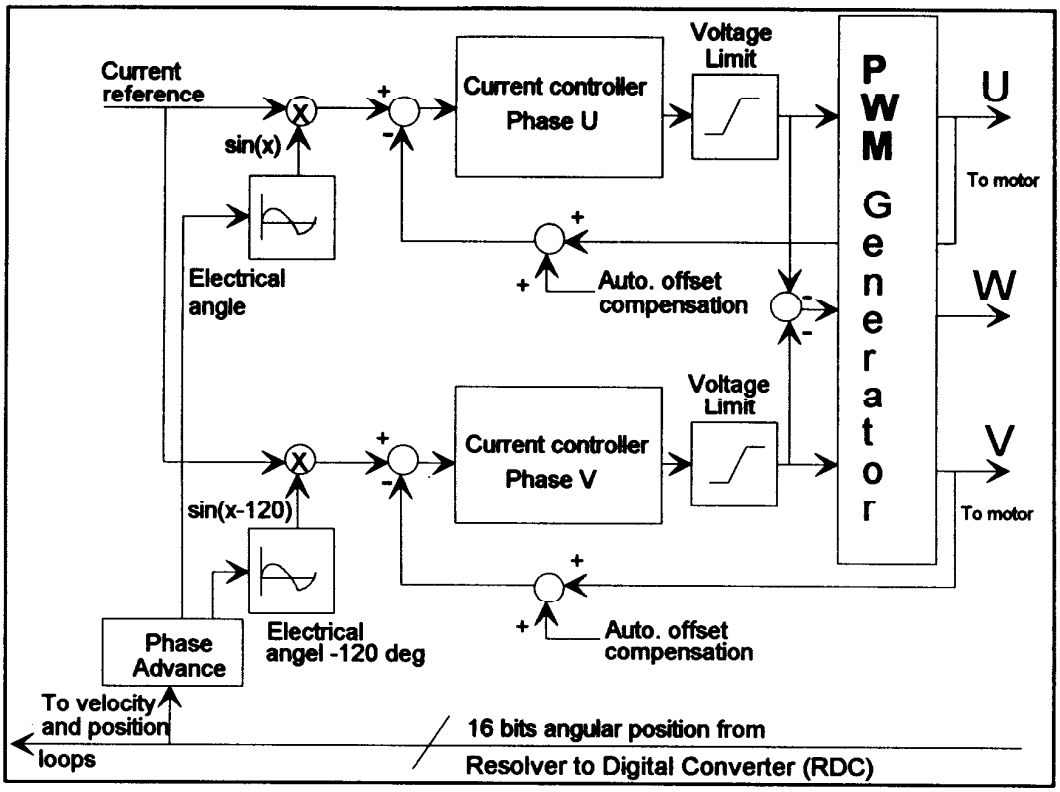


Figure 9.2: Structure of the current loop

9.4 Voltage control (velocity)

Unlike the current control, the velocity control is not a basic function, but an additional function. The DBSC current loop polynomial structure is much more adequate for the complex systems than the conventional "PID". The velocity controller is also equipped with a non-unity velocity feedback. Its global structure ensures a smooth response with no overshoot. The high integrator gain ensures good tracking performance with no steady state velocity error and high torque rejection. The anti-windup algorithm prevents overshoots and instability situations in case of current command saturation.

The velocity controller coefficients are determined automatically (pole placement method) by the "D3S" software as a function of the motor and drive selected, of the estimated load inertia, and of the Band Width value. Like the current loop, the velocity loop needs an accurate system definition as shown in section 4.7. In addition, the user is involved in the velocity controller configuration; he needs to define the Band Width and the Load Inertia.

The **"Velocity controller"** sub-menu allows to introduce the physical system parameters (see chapter 5). The following options are available:

- Load inertia [Kg * cm2]
- Load inertia ratio
- Band width [Hz]
- Minimum velocity [RPM]
- Maximum velocity [RPM
- Time to max. velocity [ms]

9.4.1 Load inertia

The load inertia (JI) needs to be evaluated by the user. This parameter has to include the load and the gear inertia transferred to the motor shaft (Load inertia divided by the gear ratio square). The default value is JI=Jm. This assumption is sufficient in many cases; it is still recommended to evaluate, as accurately as possible, the JI value.

An evaluation method of the load inertia (JI) is presented in paragraph 5.4.1.1.

9.4.2 Band Width

The velocity dynamic response is defined by the Band Width (B.W.) and depends on the application requirements. The user should have a good knowledge of the application needs before defining the Band Width. The default is 20 Hz, that should comply with most system requirements. Nevertheless, the Band Width definition is not a straight forward decision. It is strongly recommended to think well and make several tests before the optimal value final decision is taken. The Band Width tuning methods are presented in paragraph 5.4.1.2.

9.4.3 Velocity and acceleration limits

Refer to paragraph 5.3.2

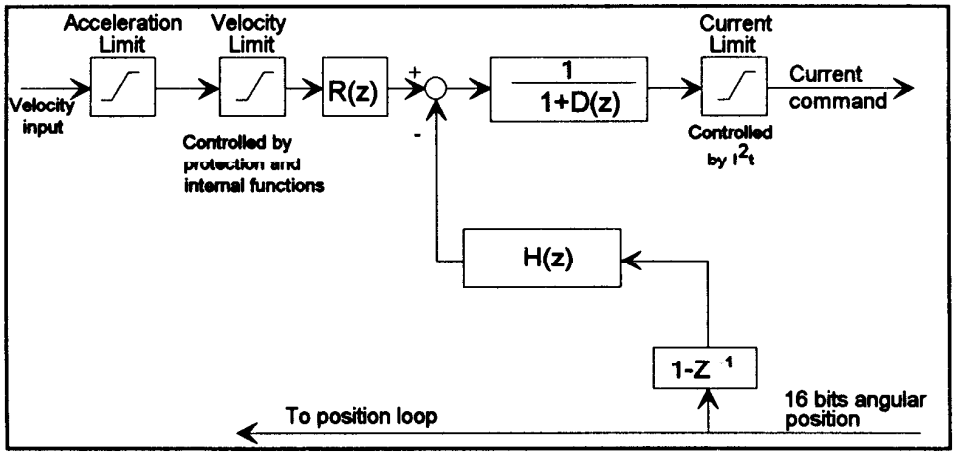


Figure 9.3: Structure of the velocity loop

9.5 Position control (Hand-wheel)

The DBSC position loop is enabled only when the "Hand Wheel" mode is selected or during "Hold position". Its structure is simple, the tuning is done automatically and is based on the velocity loop controller. The position controller so defined is very robust and does not need any action from the user except for the activation or deactivation of the Feed Forward terms (velocity and acceleration). Those two terms allow the user to get tighter and better Hand-wheel tracking performances. With the default setting, only the Velocity Feed Forward is enabled. Both Feed Forwards coefficients are automatically defined by the "D3S" software. Nevertheless, it is possible to change the Velocity and Acceleration Feed Forwards gains as explained in paragraph 5.5.2.

The Hand-wheel function is defined by the encoder resolution and by the electronic gear ratio. The DBSC can receive A, A/, B, and B/ encoder signals if the option Exx is installed or the pulse and direction signals if the option Bxx is installed. It is not possible to have both options installed on the same board. The encoder resolution is a positive integer and its value can not exceed 32767 encoder counts per revolution. The gear ratio is a "floating point" value which has an admissible range of: 0.001 to 100.0. The multiplication of the resolution by the gear ratio define the Hand-wheel gain.

The electronic gear ratio can be positive or negative. The sign determines the motor sense of rotation in relation with the one of the Hand-wheel reference.

The **"Position controller"** sub-menu allows to introduce the Hand-wheel parameters.
(see paragraphs 4.7, 5.4.2, 5.5.2 and 5.5.3.)

Remark:
When the Hand-wheel mode is selected, it is recommended to set the "Time to max. velocity" parameter to zero or to a very small value. This parameter is located in the "Velocity controller" sub-menu (see paragraph 5.3.2). This caution avoids the motor overshoots when the Hand-wheel is subject to strong move variations.

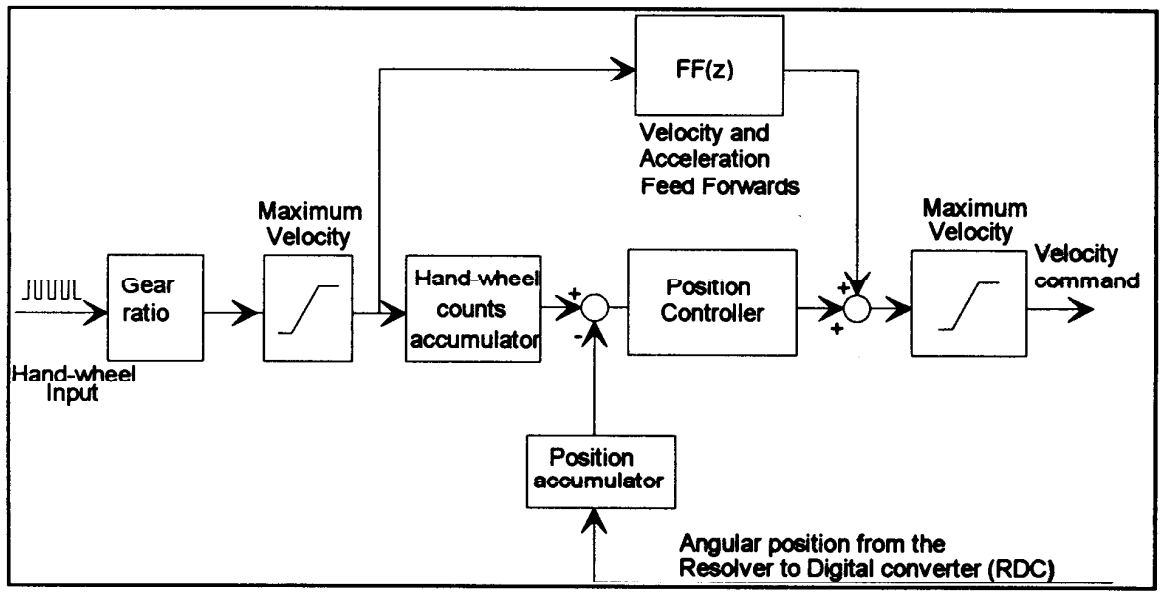


Figure 9.4: Structure of the position loop

9.6 Limit switches (CW and CCW)

To maintain a maximum protection of the electromechanical system, the DBSC reacts as soon as a limit switch is encountered. All possible situations are listed below:

9.6.1 Hand-wheel mode (position):

CW Limit: Only a positioning command which moves the motor in a negative direction (CCW) is allowed to get away from the limit. A positioning command that shall move the motor in a positive direction (CW) is discarded.

CCW Limit: Only a positioning command which moves the motor in a positive direction (CW) is allowed to get away from the limit. A positioning command that shall move the motor in a negative direction (CCW) is discarded.

9.6.2 Velocity mode:

CW Limit: Only a velocity command which moves the motor in a negative direction (CCW) is allowed to get away from the limit. A velocity command that shall move the motor in a positive direction (CW) is discarded.

CCW Limit: Only a velocity command which moves the motor in a positive direction (CW) is allowed to get away from the limit. A velocity command that shall move the motor in a negative direction (CCW) is discarded.

9.6.3 Current mode (Torque):

CW Limit: The control mode is automatically changed to **"Velocity mode"**. The motor brakes and holds position. The **"current mode"** is reactivated only when the current command is lower than the actual current needed to hold the position. This ensures that only a command which moves the motor away from the limit is allowed.

CCW Limit: The control mode is automatically changed to **"Velocity mode"**. The motor brakes and holds position. The **"current mode"** is reactivated only when the current command is higher than the actual current needed to hold the position. This ensures that only a command which moves the motor away from the limit is allowed.

9.6.4 CW and CCW Limits simultaneously active:

When both limits switches are operated on the same time, the DBSC treats this situation as if the disable was activated, but this is also an error situation therefor a **"L"** is shown on the DBSC display. This permits to see the difference between a drive disable situation and limit wiring errors. It is then necessary to check the limit switches and their wiring.

Remarks:

- **Even if the drive is in current mode, it is necessary to tune the velocity controller. This allows full protection if a limit switch is reached.**
- **The limit switches must be connected such that CW limit is encountered during CW rotation of the motor (positive resolver direction) and CCW is encountered during CCW rotation (negative resolver direction).**

9.7 "Drive OK" (X2-14) output and "Amplifier fault" Relay contact (X3-3, X3-4)

Those two outputs are similar, there is a slight difference in the way they are handled:

- The "Drive OK" is only controlled by the hardware.
- The Fault relay is controlled by the firmware and the hardware.

When the drive is OK, the firmware can control the fault relay. For example, a "Fatal Error" generated by the PLC program will open the relay contact. When the drive is not OK (fault conditions 1 to 6), the hardware takes the control over the firmware and opens the relay contact.

At power-up, the firmware will close the Fault relay contact only after the "Drive OK" has been set by the hardware.

There is not any direct link between the two signals.

9.8 The outputs state at power-up

The outputs MAO1 to MAO4 and the "Drive OK" are all opened at power-up during the internal initialization procedure.

APPENDIX A

The solder bridges and jumpers (SBxxx)

This appendix is here mainly for information. Most solder bridges and jumpers must not be touched, except for very few of them in particular cases. To access the jumpers, it is necessary to power-off the drive, to remove the front plate and to pull the DCP board out of its housing.

Solder Bridge	Description	Factory default
SB 101	AGND to DGND link, must always be closed.	CLOSED
SB 201	Not needed, must always be closed (Hardware dependent)	CLOSED
SB 202	Watchdog, must always be closed (factory test purpose only)	CLOSED
SB 203	CPU clock, must always be closed (factory test purpose only)	CLOSED
SB 301	Burn in mode, must always be opened (factory test purpose only)	OPENED
SB 302	Resolver ground (RGND) to AGND link, is closed internally on the printed circuit board (Etch). Must be left as it is.	CLOSED
SB 303	For resolver reference signal lead, must always be closed.	CLOSED
SB 304	Only usable with option Exx: Opened: RS 485 multi-drop mode Closed: RS 422 daisy chain mode Also see table 3.1 in section 3.6.	OPENED
SB 511	For DBSC 2000 / 3000 only, Opened: External 24 V (CIV) opto-isolation ICs supply Closed: Internal 24 V opto-isolation ICs supply	OPENED
SB 512	For DBSC 2000 / 3000 only, Opened: External customer ground (CGND) for opto-isolation ICs. Closed: Internal 24 V Ground for opto-isolation ICs	OPENED
SB 513	Only for IMAS option: Opened: Analog output 2 (DAC2) can be wired and used externally Closed: Analog output 2 (DAC2) is used for IMAS offset generation Also see section 6.2.	OPENED

Jumper	Description	Factory default
SB 501 to 510	1-2 installed: RS 232 serial communication 2-3 installed: RS 422 or RS 485 communication ports (option Exx) Also see section 3.6 and installation manual section 6.1.1, 6.1.2 and 10.5.	1-2 installed (RS 232 com.)

Table A-1: Solder bridges and jumpers description

APPENDIX B

The DIP switches AS1-8 (Functions)

The DIP switches AS1-8 are installed on the front of the DBSC if option Exx is present. Verify that the position of each switch corresponds with the desired configuration for the application. The functions of each switch are described bellow.

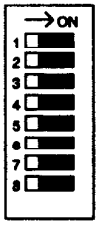


Figure B.1: The DIP switches (AS1-8)

TABLE B-1

switches	Description	Default:	Remark
AS1..4	addressing: ON = 1, Switch 1 = LSB, Switch 4 = MSB	Address = 0	①
AS5	Not used	OFF	leave "OFF"
AS6	Hold position	OFF	"ON" = hold position ②
AS7	Automatic offset calibration	OFF	"ON" = calibration activated (See section 5.2) ③
AS8	Drive disable	OFF	"ON" = drive enable ④

Remarks:

① Up to 16 DBSC can be addressed through the daisy chain with the RS232 or RS422 serial port or through the multi-drop configuration (RS485). If only one DBSC is connected to the host, the address is not important.

② The "Hold position" can be used for a definite motor stop while in current mode or in velocity mode. While in "Hold position", the amplifier is enabled and the motor generates torque. The Hold position can be activated through the switch AS6, or the digital input X2-3, or via the D3S software.

③ As long that the automatic offset calibration is active (AS7 = "ON"), and the amplifier is disabled, a measuring procedure of the mean analog input value is executed and offset compensated.
At power-up, if the automatic offset calibration is selected (AS7 = "ON"), the above procedure is executed before the power stage activation.
If the automatic offset calibration is selected (AS7 = "ON"), make sure that the analog command (DAC output of the CNC or motion control board) is precisely zero during power-up or while the amplifier is disabled.

④ At power-up, it is recommended to disable the amplifier. The drive disable can be commanded through switch AS8, or the digital input X2-2 or via the D3S software.

With the standard DBSC (low cost option Axx), the DIP switches AS1-8 are not installed. Therefor the drive can not be daisy chained to others. The "Hold position" and the offset calibration functions are only available through the D3S software. The amplifier enable function is possible through the digital input X2-2 and via the D3S software. The equivalent position of the switches 1 to 7 is "OFF", and of switch 8, is "ON".

APPENDIX C1

Communication via PC.

RS 232

The RS232 is the standard interface. The communication Baud rate is fixed to 9600 and can not be modified. The serial communication cable is shown bellow:

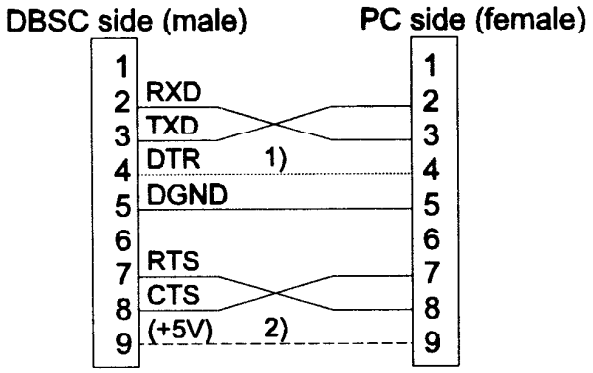


Figure C-1: RS 232 serial link definition

Notes:

1) The DTR line is optional:
   a) If the DBSC communicates with the D3S software, the DTR line is not used.
   b) If it communicate through the "HCP", a routine could be installed in the host in order to check if the DBSC is powered and is ready to communicate. The DBSC sets this signal to "level high" as soon as it is powered-up.
2) The "+5V" signal is optional. The jumper SB509 has to be installed in position 1-2 (factory default). This output can be used for powering a hand-held-terminal.

Recommendations:

- To prevent interference, the cable length should not exceed 1 meter.
- This cable should be shielded and should not be place along high power and AC signal devices such as motor cables, amplifiers etc.

For the communication protocol, refer to the **"communication protocol"** document.

APPENDIX C2

Communication via PC.

RS422 / RS485

The communication link RS422 / RS485 is mounted only if option Exx is installed, it must be defined when the order is placed (see table 2-1). It is not possible to modify the amplifier in a later stage in order to change from RS232 to RS422 / RS485 configuration or vise versa. In case of multi-axis application, use the RS422 setting for daisy chain or the RS485 for multi-drop configuration (Refer to section 3.6 for details). In case of single axis, the RS485 is compatible with RS422. The RS422 / RS485 cable is shown bellow:

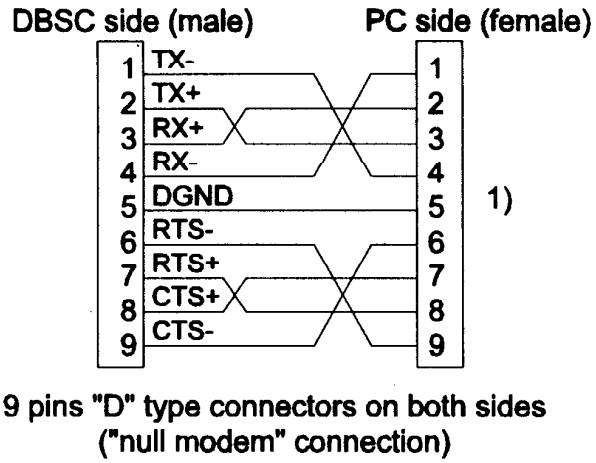


Figure C-2: RS422 / 485 communication link

Note:

1) The interface wiring on the PC side can be different than shown above. Refer to your PC hand book for details.

Recommendations:

- This should be a twisted pairs and shielded cable. It should not be place along high power and AC signal devices such as motor cables, amplifiers etc.
- It is possible to install termination resistors (R408 and R407) for the RS485 communication. Ex-factory, those resistors are not mounted, they are not needed if the cable is short enough. If needed, the value depends on the cable type, section and length.

For the communication protocol, Refer to the **"communication protocol"** document.

PC minimum requirements

The minimum PC requirements for the use of D3S software are as follows:

Device	Minimum requirement
CPU board	AT 386
Monitor	EGA or VGA
Operating system	DOS 5.0
RAM	640 Kbytes
Available space on hard disk	2 Mbytes
Communication port	RS232, RS422 or RS485

Table C-1: Minimum PC characteristics

APPENDIX D1

"7 segments" Display (error messages)

Tables D-1 to D-3 bellow show the different possible status that can appear on the display and in the information box of the D3S software. The errors and status shown on the display and in the information box on the screen provide limited information. The D3S software, through the **"Drive Errors"** sub-menu, provides the complete status information (Refer to appendix D3). The trouble shooting section (appendix E) also explains how the error can occur and how to cure it.

Display	Status	Remarks
	Power-up test	Normal for a short time during initialization. Error if continuously displayed.
	General error	This can be Eeprom, Ram, Eprom checksum error, also an error which is intentionally generated by the user via the PLC or a fault relay failure. (Refer to appendices E).
	In operation, drive is enabled	The point indicates that the drive is enabled whatever message is displayed.
	Disabled	The disable can have been commanded through the switch AS8, or the digital input X2-2, or by communication (software), or after downloading a new configuration file to the DBSC.
	"Jog" mode	A "Jog" move can only be commanded via communication (software).
	Hold position	The Hold position can be commanded through the switch AS6, or the digital input X2-3 ,or by communication (software).
	Version mismatch	The Eeprom content (identity module) mismatches the installed Eprom version. The amplifier is disabled, use the "D3S" to reconfigure the drive.
	CW limit	The Clock Wise limit switch is activated. The motor can turn only in the Counter Clock Wise direction.
	CCW limit	The Counter Clock Wise limit switch is activated. The motor can turn only in the Clock Wise direction.
	CW and CCW Limits	Both limit switches are activated. This is an error situation, the amplifier is disabled.
	In position	The following error is smaller than the "In position" band value which is a parameter in the DBSC, (see paragraph 4.7.2.). This is valid only when the "Hand-wheel" mode is selected.
	Following Error (warning level)	The following error is greater than the "In position" band value which is a parameter in the DBSC, (see paragraph 4.7.2.). This is valid only in "Hand-wheel" mode. The drive stays enabled.
	Fatal following error (Fatal level)	The following error has exceeded the "fatal following error" level which is a parameter in the DBSC, (see paragraph 4.7.2.). This is valid only in "Hand-wheel" mode. The drive **stays enabled** and the error is latched. We can reset the error by communication, or digital input (X2-9), or power down. If it is desired to disable the drive on the Fatal following error, the user can do so by programming this function in the PLC (see chapter 7)

Table D-1: Status detected by microprocessor

Remark: The decimal point always indicates that the amplifier is enabled.

APPENDIX D1 (continued)

"7 segments" Display (error messages)

Tables D-1 to D-3 show the different possible status that can appear on the display and in the information box of the D3S software. The errors and status shown on the display and in the information box on the screen provide limited information. The D3S software, through the **"Drive Errors"** sub-menu, provides the complete status information (Refer to appendix D3). The trouble shooting section (appendix E) also explains how the error can occur and how to cure it.

Display	Status	Remarks
	Over voltage	Fault condition. The drive is disabled and the error is latched. It is necessary to reset the DBSC through the digital input (X2-9), or the "D3S" reset command, or power down.
	BPS not ready	Fault condition (Only on DBSC 2000/3000). The drive is disabled the error is not latched. The drive will be re-enabled as soon as the fault disappears. (Refer to appendix E.)
	Over current	Fault condition. The drive is disabled and the error is latched. It is necessary to power down the drive. (See also additional explanations in Appendix E)
	Control voltage fault	Fault condition. The drive is disabled and the error is latched. It is necessary to reset the DBSC through the digital input (X2-9), or the "D3S" reset command, or power down.
	software fault or resolver fault.	Fault condition: "watchdog", resolver fault or EEPROM missing. To see the exact error type, Use the "D3S" software. Refer to table D3, and to appendices D3 and E. It is necessary to reset the DBSC through the digital input (X2-9), "D3S", or power down.
	Electronic fuse	Fault condition: if Error 7 (warning) last longer than the I2t time programmed in the "Drive Type" sub-menu (see section 4.7.4), the drive is disabled and the error is latched. It is necessary to reset the DBSC through the input (X2-9), "D3S", or power down.
	I^2t, motor or drive over-temperature warnings	Motor temperature (option Exx), or drive temperature or I^2t warning conditions. The amplifier stays enable. Error 6 will be generated if this state last too long. (also see appendix E)

Table D-2: Errors detected by the hardware and sent to the microprocessor.

APPENDIX D2

Correspondence between the display and the information box error messages

Tables D-1 to D-3 show the different possible status that can appear on the display and in the information box of the D3S software. The errors and status shown on the display and in the information box on the screen provide limited information. The D3S software, through the **"Drive Errors"** sub-menu, provides the complete status information (Refer to appendix D3). The trouble shooting section (appendix E) also explains how the error can occur and how to cure it.

Status	7 segments	Operation mode	Fault
Normal	decimal point	ENABLE	NO FAULT
Jog	J	JOG	NO FAULT
Clock wise Limit	⊣.	CW	NO FAULT
Counter clock wise Limit	⊢.	CCW	NO FAULT
Both Limits CW + CCW	L	DISABLE HW	ERROR
Hold position	H	HOLD POS	NO FAULT
Software disabled	d	DISABLE SW	NO FAULT
hardware disabled	d	DISABLE HW	NO FAULT
Amplifier type fault	d	DISABLE HW	ERROR
Version mismatch	U	DISABLE SW	ERROR
Over voltage	1	DISABLE HW	ERROR
BPS not ready (For DBSC 2000/3000 series only)	2	DISABLE HW	ERROR
Over-current	3	DISABLE HW	ERROR
Control voltage fault	4	DISABLE HW	ERROR
resolver fault	5	DISABLE HW	ERROR
EEPROM missing	5	DISABLE HW	ERROR
Software fault	5	DISABLE HW	ERROR
I2t Error, motor or drive over-temperature	6	DISABLE HW	ERROR
I2t warning	7	Current Limitation	WARNING
Motor or drive over-temperature warnings	7	ENABLE	WARNING
General error	9	DISABLE HW	ERROR

Table D-3: Status shown on the "7 segments" display and on the screen ("D3S" software)

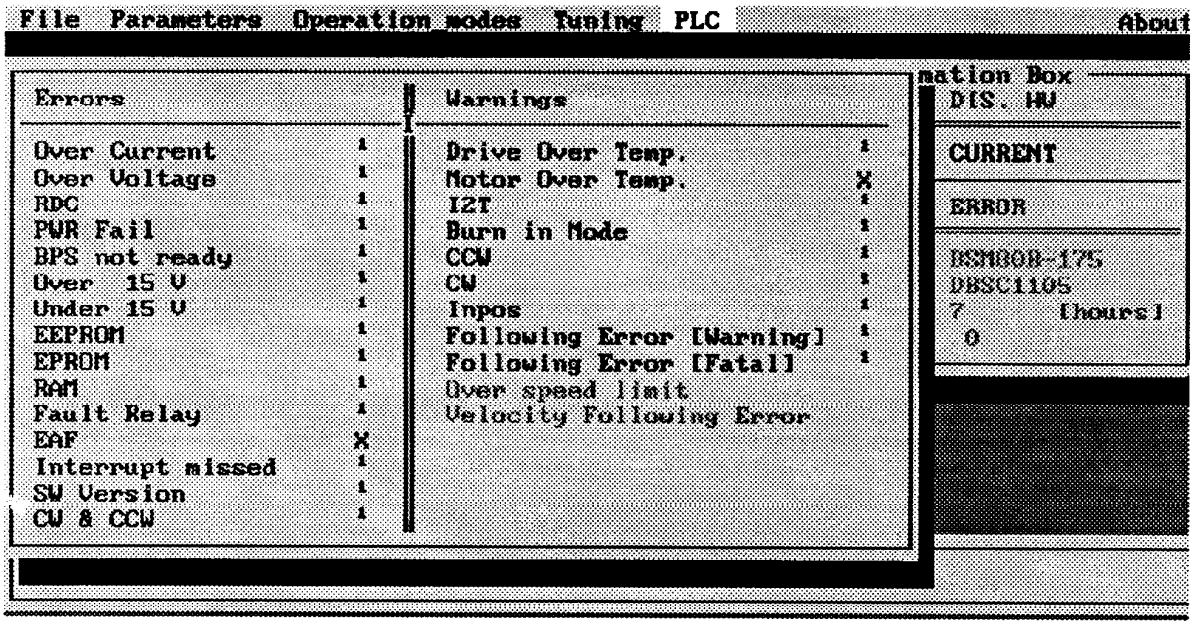
APPENDIX D3

The "DRIVE ERROR" sub-menu (debugging)

Located in the "PLC" menu on the top tool bar, the **"Drive Errors"** option allows the user to analyze more accurately the source of error. A complete list of status updated in real time is shown on the screen. In any application, if an error keeps occurring, connect the host and use the "D3S" software to try to understand what is causing a problem.

❶ Select the **"PLC"** sub-menu and press **<Enter>**.
❷ Choose the **"Drive Errors"** option and press **<Enter>**.

The window with the complete DBSC status is displayed:



The symbol **"1"** indicates that the specific condition O.K. The red **"X"** symbol indicates that the specific condition is an error. In the above example, we see that the drive "Electronic Armature Fuse" (EAF) was activated as a result of a "motor over temperature" condition.

Note that the "Over speed limit" and "Velocity Following Error" are not relevant.

Correspondence with the "7 segments" display:

Over current	3	Drive over temperature	7.
Over voltage	1	Motor over temperature	7.
RDC error (Resolver error)	5	I2t limit	7.
Power fault	9	Burn in mode	N/A
BPS fault	2	CCW limit	(see appendix D1)
Over voltage (15V)	4	CW limit	(see appendix D1)
Under voltage (15V)	4	In position	P
EEPROM error	9	Following error (warning)	E
EPROM error	9	Following error (fatal)	F
RAM error	9	Over speed limit	N/A
Fault relay contact opened	9	Velocity following error	N/A
EAF (Electronic Armature Fuse)	6		
Missing interrupt	9		
Software version mismatch	U		
CW & CCW limits both activated	L		

APPENDIX E

Trouble shooting

When one of the following faults is detected, the drive is disabled: errors 1 to 6, and errors 9, U, and L). The following paragraphs provide few idea of what could happen, how to check the error source and how to cure the problem.

Note that several of these errors (1, 4, 5, 6 and 9) can be reset through the "Reset" input, or the "Reset" command sent via the "D3S" or PLC program. See also the section 7.3 and 8.7.5.

What to do if I see one of the following faults?

ERROR 1: This appears when the voltage exceeds the 400 VDC for the DBSC 1100/1000 and 2000 series, 800 VDC for the DBSC 3000 series.

Note that for the DBSC 2000 and 3000 series, the over voltage value must be downloaded to the drive while executing the setup procedure. (see section 4.7) This is because it is impossible for the DCP board to automatically know which type of power stage is linked to it. This is not the case for the DBSC 1000/1100 series.

Error 1 can occur randomly at power-up; it may be due to transient voltage overshoot at the transformer secondary end. The higher the power of the transformer is, the bigger is the transient overshoot at the secondary.

Possible Solutions: Decrease the secondary voltage output by wiring the primary on the +10 % input, Load the transformer secondary for a short time (500 ms) at power-up by commuting a load resistor to the transformer output.

ERROR 2: (BPS not ready) This error can occur only on DBSC 2000 and 3000 series. It is generated and sent to the DBSC by the BPS itself. For this to work, the terminals X10-3 and X10-4 must be wired between the DBSC and the BPS. Refer to the paragraph 4.3.1 of the BPS 2000/3000 manual.
The "BPS not ready" signal appears 100 ms after the problem has occurred. As soon as the DBSC receives this signal, the amplifier is disabled and ERROR 2 is displayed. This error is not latched and the drive will re-enable as soon as it disappears.
The BPS output signal "alarm" (X4-2 terminal on the BPS) is active as soon as the error occurs. This could be wired in one of the DBSC input MaI1 to MaI4 or to the CNC control to stop the machine before ERROR 2 appears.
The following conditions on the BPS will generate ERROR 2:

- One phase on the main supply is missing.
- Bus under-voltage (Only for DBSC 1100 / 1000 series, without option 24V. the under-voltage value is about 160 VDC).
- Dynamic brake fault (Regeneration problem).
- Input main supply for the 24 Volts control supply missing.
- Under-voltage of the 24 Volts supply. (For the DBSC 1100 / 1000 series with option 24V and for the DBSC 2000 / 3000 series. The under-voltage value is 20.4 Volts).

(Also see sections 3 and 4 of the BPS 2000 / 3000 manual.)

ERROR 3: This fault indicates that a short circuit has occurred either in the motor cable, or in the motor itself, or in the power stage.
Switch off the power, make sure that the motor cable and the motor are in good order and correctly connected, disable the drive and power-up the drive again. If the fault comes as soon as the power stage is enabled, verify the cables or even the motor itself.

ERROR 4: This can be seen when one of the control voltages +15 VDC, -15 VDC, +5 VDC (and 24 VDC for all amplifiers equipped with the 24V option) is too low or too high. (under-voltage limits: +/-12 V, +4.75 and +20 VDC for the option 24 V). The under-voltage can also occur if the control voltages are affected of a too big delay (over 300 ms) after the 24V power-up (for the drives equipped with the 24V option) or after the rise of the bus voltage (for the drive without the 24V option). This fault is very rare, it could indicate a problem in the DC/DC converter or in an integrated circuit. Make sure that all control voltages are normal. If you do not detect any apparent reason for the fault, contact the factory.

APPENDIX E (continued)

Trouble shooting

ERROR 5: First check the exact type of fault using the **"Drive Error"** sub-menu in the D3S software (see appendix D3).

If the status window indicates **"Resolver fault"**, the resolver cable may be damaged, or may not be correctly connected, or the resolver may be damaged. Check, with an oscilloscope, the reference signal between points R1 and R2 at the motor side (signal: 21 Vpp, 7,2 KHz), between points S1 and S3 (cosine) and between points S2 et S4 (sine) at the motor side and at the amplifier side (signals values are between 0 and 21 Vpp depending on the motor shaft position, 7.2 KHz). If the reference signal is not present, verify lines R1 et R3. If the Sine and Cosine signals are present at the motor output, but not present at the amplifier side, verify lines S1, S2, S3 and S4. There is certainly a wire which is cut in the cable, or a bad contact in the resolver connector at the motor end or in the X9 connector at the amplifier side.

ERROR 6: Electronic fuse: When the current in the motor is above the amplifier nominal current, the error 7 is displayed ("I^2t" warning). Error 7 also happens when the thermo-switch of the drive opens. Further more, when option Exx is installed and the thermo-switch from motor is opened, or not wired, (connectors X4-1 and X4-2 not jumper wired), error 7 will also occur. If it last longer than programmed in the "drive type" sub-menu under "I2t Limit" (section 4.7.4), Error 6 is displayed and latched. The length of time can be set from 1.5 to 3.0 seconds. The default value is 2.88 seconds. Three cases have to be considered: the amplifier thermo-switch, the motor thermo-switch and the "I^2t" limit. First check the exact type of fault using the **"Drive Error"** sub-menu in the D3S software (see appendix D3).

The case of the amplifier thermo-switch:
When the motor load is big and the cabinet temperature is too high, the amplifier can not dissipate the heat fast enough through its heat sink. As soon as the temperature reaches 82 °C, a thermal contact opens and provokes the fault.
Solution: Ventilate the cabinet or, if the machine it is in a hot country, install the air conditioning inside the cabinet.

The case of the motor thermo-switch:
This case can only happen if the DBSC is equipped with option Exx. It is commonly seen at the first system start-up when wiring principle is not fully understood. Check the following points:

1. Are the opto-insulation IC's powered?
   If the input is wired for **source configuration**, a 24 volts external supply has to be wired as follows: **+ 24 V. to the CIV input (X2-19) and the 0 V. to the CGND (X2-20).**
   If the input is wired for **sink configuration**, a 24 volts external supply has to be wired as follows: **+ 24 V. to the CGND input (X2-20) and the 0 V. to the CIV (X2-19).**
2. Is the motor thermo-switch connected?
   If yes, using an ohmmeter, check that the motor thermo-switch is not opened.
   If it is opened, the motor is too hot, the current needed for the application may be too high, and the amplifier nominal current as set in the "drive type" sub-menu may be above the motor nominal current.
   possible solutions:
   a) Set the amplifier nominal current equals or lower than the motor nominal current.
   b) Reduce the load and acceleration in the system.
   c) Tune the phase advance (see paragraph 5.5.1)
   d) Try to use lower band width, and also make sure that the CNC or motion control board position loop is correctly tuned tune.
   If no, connect the motor thermo-switch or, if you do not want to connect it to the drive, wire a jumper between terminals X4-1 and X4-2.

APPENDIX E (continued)

Trouble shooting

The case of the "I^2t" limit:
This can be caused by a load which is too high on the motor (brake which is ON during the move, or gear box which is blocked for example), or by bad motor or resolver wiring (motor phases crossed, resolver sine and cosine crossed), or by a missing phase (motor cable cut, bad electrical contact, connector problem). If this fault is present with an unloaded motor, verify the wiring, the motor and resolver cables. In the other case, make sure that there is not a mechanical problem. You may also optimize the phase advance. Refer to paragraph 5.5.1. If no success is achieved, the motor and / or amplifier may be under-sized for the application; decrease the accelerations or use a bigger amplifier and / or motor.

ERROR 7: "I^2t", motor over-temperature, drive over-temperature Warning: (see ERROR 6)

ERROR 9: This can be an Eeprom, Ram, Eprom checksum error, or also an error which is intentionally generated by the user via the PLC or a fault relay failure. To see the exact type of fault, use the "D3S" software. (Refer to appendix D3). It is necessary to reset the drive via the software command, or the digital input X2-9, or to power down. (See also section 7.3.)

ERROR U: Error U indicates a mismatch between Eeprom content and the firmware. This happens when a configuration file which was done with another firmware version has been downloaded to the DBSC. In that case, redefine the configuration as shown in section 7 and reset the drive via the "D3S" ("Fault reset" sub-menu in the PLC menu), or through the digital input X2-9, or by power down.
It can also indicate that the content of the Eeprom has been lost. In that case, if you are sure that no wrong manipulation was done, contact the factory.

ERROR L: This means that both limits are activated. Make sure that both limit switches are correctly wired, that there is not a contact problem or cable break. If no limit switch is used, wire X2-4 and X2-5 to the +24 Volts if sink configuration, or to the 0 Volt if source configuration. Refer to the "installation and instruction" manual.

STATUS d: Several cases can cause the amplifier to be disabled and to display the "d". If this is a "Hardware disable", look at points "a" to "c" bellow. In the other case look at point "d".

a) Check the information box In the "D3S" main screen. It will tell you if the disable is hardware or software. In case both are activated, it will show a "Software disable".

b) Verify switch AS8 and set it to the "ON" position

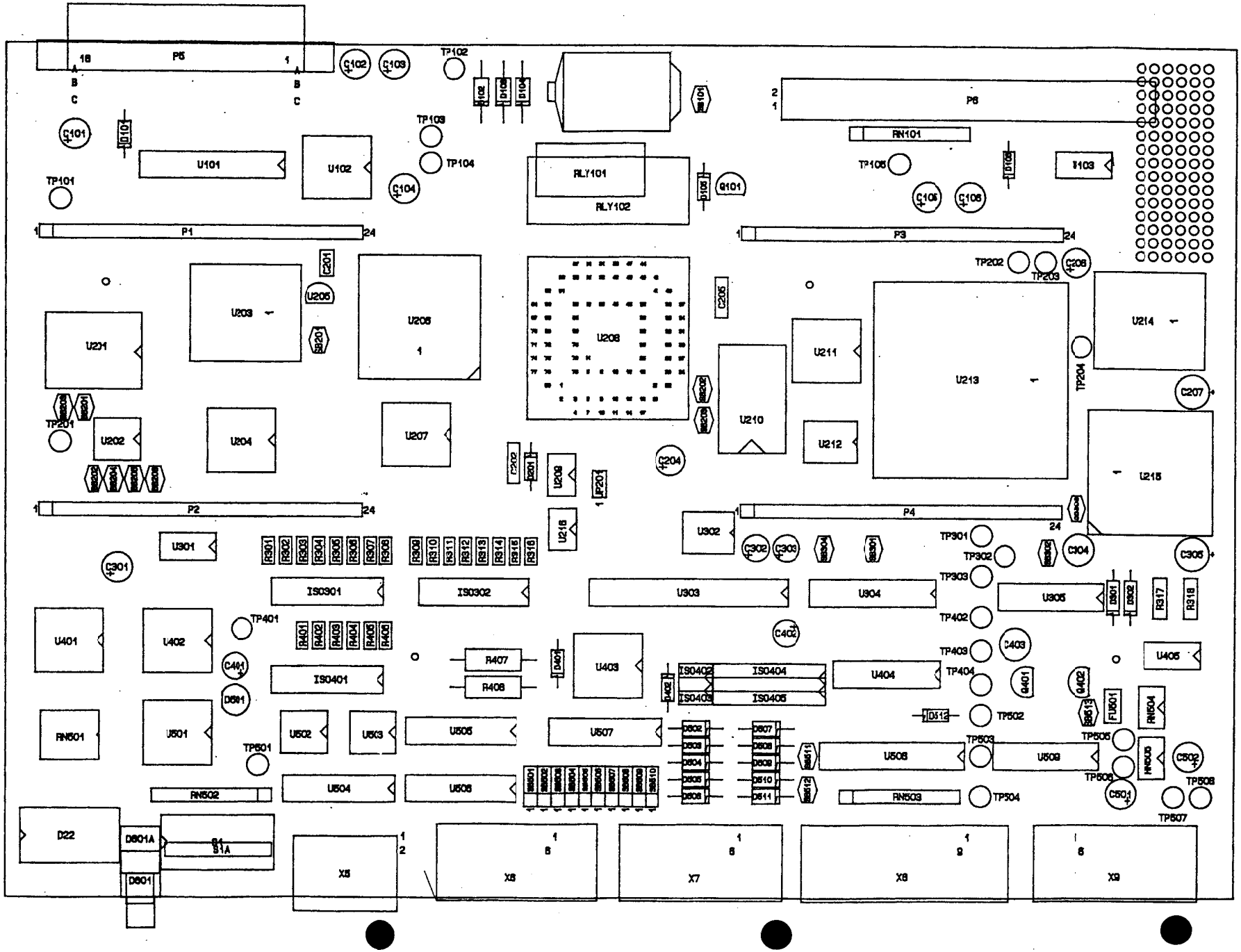
c) Check the disable switch (wired on X2-2) and close it. If no disable switch is used, wire X2-3 to the +24 Volts if sink configuration, or to the 0 Volt if source configuration. Refer to the "installation and instruction" manual.

d) Check the 24 Volts external supply for the disable, CW and CCW switches. This voltage must be over 24.4 Volts.

e) Start the "D3S" software and press the **<F5>** key.

Also note that after downloading a configuration file to the DBSC, the display will always show a "d" and the information box a "Software disable", this even if a hardware switch is activated. Press <F5> to enable the drive and close any switch that may still cause the disable status.

NAME : SER-DCP	CAT. NO. : 00008000-01	REV. : 01	
LAYER : ASSY C.S.	DATE : 10/01/95	FILM NO. :	



BALDOR®
MOTORS AND DRIVES

D-BSC
SERIES 1000 to 3000

SERVO CONTROL
FOR BRUSHLESS AC MOTOR
WITH
DIGITAL CONTROL PLATFORM
(DCP)

HOST COMMUNICATION PROTOCOL

(Addendum to operating manual)

Version 1.0 31.08.95
Addendum to ID-Nr. 22912

Table of contents

1. Introduction

This document describes the software protocol **for firmware version 4008** that is requires to communicate with the DBSC. This will allow the user to issue commands, set operation modes and gather internal data with any host computer. The communication routines need to be written by the user and included in to his application software.

Before describing the set of commands that the DBSC accepts, we will explain the internal communication "mechanism". This will help understanding the communication protocol and translating it in the language you require (Basic, Pascal, C, C++ etc.).

Be aware that for the DBSC first initialization and servo tuning you will still use the **"D3S"** (Digital Servo Set-up Software) delivered with the amplifier. All commands described in this document are built in the **"D3S"** software, so in normal cases it is not necessary to know about the host communication protocol. Nevertheless, in some applications the user wants his own software to operate directly the DBSC; for example, to set the hand wheel mode or scale factor, to read velocity etc. The Host protocol communication is absolutely needed when the IMAS option is installed and used (Option EBx). At power-up, the host will read the "Absolute IMAS value" using this protocol. It is necessary to fully understand the host communication protocol as well as to master the programming language used. Examples of routines written in BASIC and C are given in this document. Please note that they are only examples and that they do not necessarily meet your application. Also it is recommended that you learn which functions in DBSC are not accessible while a specific mode is activated (current or hand wheel modes for example). This will make the program safer. A message telling the operator that such function is not allowed in the present mode may be displayed to the screen, or the present mode may be change automatically to be able to operate the function.

BALDOR does not intend to support programming language and will not write communication routines for the user's application. It is the responsibility of the user to write the routines, to make them safe and usable, to handle correctly the hand shaking, timings etc.

BALDOR still intends to help the user by giving hints and ideas as well as answering questions. Please do not hesitate to call your nearest dealer to get our advice.

Note:

The host serial communication port should be configured to 1 start bit, 1 stop bit, 9600 baud, 8 data bits, no parity, and no hardware hand shake. Also, the port address has to be defined as 3F8 Hex for COM 1 or 2F8 Hex for COM 2.

The protocol is based on software handshake: the DBSC echoes every byte back to the host. It is the responsibility of the host to check the correctness of the data. Only the host can evoke a communication process.

The host uses "RTS" and signal manually to indicate that an addressing command is being sent.

2. Communication

2.1 Communication principle

Basically, all communication operations (write to DBSC or read from DBSC) are based on the following rules and sequence:

1. The host always starts a communication operation (DBSC is always waiting data from the host). The host sends a **"header"** to the DBSC. The **"header"** tells the DBSC to "listen" as a command is about to come.
2. The DBSC acknowledge the host by sending the **"header"** back. The application program should check if the character received back is the same that the one he has sent. If yes jump to point 3. If not, or if no signal is received back within 100 ms, the software should jump to an end of communication sub routine and indicate a communication error message.
3. The host send the **"command"** to the DBSC.
4. The DBSC acknowledge the host by sending the **"command"** back. The application program should check if the character received back is the same that the one he has sent. If yes jump to point 5. If not, or if no signal is received back within 100 ms, the software should jump to an end of communication sub routine and indicate a communication error message.
5. Depending of the operation that DBSC has to perform, it needs some time to get an **"argument"** from the host (for example scale factor of the hand wheel or jog velocity) which can be one byte long or more. Only one byte can be transmitted at a time, it will be described in detail in the communication protocol section. when the argument is passed to the DBSC and checked by application software, the host will send a **"Trigger"** signal. This will activate the function.
   Some time the DBSC does not need any **argument** and just waits for the **Trigger** to activate the function. For example, read the actual velocity does not need any **argument**.
6. The host read back the value, two cases are possible:
   a) If the function does not require to send a value back (for example jog command), the DBSC will only acknowledge the host by sending the Trigger character back to the host.
   b) If the function requires to send a value back (for example get velocity), only the value will be sent back (not the Trigger character). If this value is longer than one byte, the host has to store and acknowledge the DBSC by sending back the intermediate bytes.
7. If necessary, the host computes the values it reads from the DBSC and display them on the screen or use them for any other purpose.

Remarks:

- DBSC is only able to read and send **character** type values (8 bits values). No integer, or fixed point, or floating point values can be send by the DBSC, as they take more than one byte of length. Therefor the user needs to built a program which is able to convert the characters read from the DBSC in the correct format. In the other direction, it is also necessary to transform the numerical values in character format in order to transmit them to the DBSC. These transformations are not straight forward and require some practice on bits manipulation.
- A time out function (100 ms) should be built by the user in case there is a loss of communication, this is mainly to avoid the application software to hang up and to display the error information for the operator.
- A time out function is also built on the DBSC side. When it does not get the acknowledge from the host in less than 150 ms, it will return to it's original state, waiting for a new header.

2.2 Addressing a DBSC

At power-up all DBSC's in the daisy chain are in "Sleep mode". This is also true if only one DBSC is connected to the host. Therefore, the first communication starts always with an addressing procedure. This procedure has to be repeated each time that communication with an other DBSC in the daisy chain is desired. Multi axis communication is supported from firmware version 4007 and higher.

The addressing principle is as follows:

1) pull down RTS line (set RTS bit in modem register: bit 0 of port_base_address+4 for a PC.).
2) Send over the communication port the address (Hexadecimal byte value C0 to CF: C0 for DBSC0, C1 for DBSC1, ... , CF for DBSC15.)
3) The newly addressed card echoes is address. all other cards are in "Sleep mode".
4) Pull up the RTS line (reset RTS bit).

The setting of the RTS bit is the key to address a new DBSC. It indicates to the DBSC's that the byte sent is an address. In other word, when the RTS line is pulled down, all DBSC's are "listening" to the host while waiting the address.

In programming language, using macros, this procedure could be written as follows:

```
Set_RTS()          /* Set the RTS bit the modem control register:
                      bit 0 of port_base_address+4 for a PC. */
_Write(0xC?)       /* send the DBSC address over the communication port,
                      ? represents a value of 0 to F. */
ack = _Read()      /* The addressed DBSC returns its address for acknowledgment. If no
                      response is received, there is no DBSC responding at this address. */
Clr_RTS()          /* Reset the RTS bit and returns to default mode.
```

3. DBSC Command set

As explained above, the DBSC is only capable to receive character type values (8 bits words). Therefore, each command understood by the DBSC is a value situated between 0 and 255 which is called operational code. The list of operational codes with a short function description is given bellow (Tables 1 and 2).

A suggested macro mnemonic is given in the third column of the tables. Macros are widely used in programming languages such as Pascal and C. They make the program easier to write and to read. It is advisable to write such files and to include them in your communication program for compiling. An example of C language macro file for the operational codes is given after the tables. A few other suggestions of macro definitions are also included, they may be helpful for some typical DBSC constants.

The complete explanation of each command, including type of value, units and how to compute, will be given in the section 4. The Macro mnemonics as suggested in tables 1 and 2 will be used.

Operational code	Function description	Suggested Macro mnemonic	Section
88 (or "X" in ASCII)	Header: this command always starts a communication sequence	Header	
84 (or "T" in ASCII)	Trigger: this command start any internal DBSC routine pre-selected by one of the command described bellow.	Trig_read or Trig_Operation	
0	Get master absolute resolver position	SYS_MRPOS?	4.1
1	Get hand wheel actual position following error	HW_PFE?	4.5
2	Get actual velocity	VEL_VEL?	4.4
3	Get velocity ramp (acceleration limit)	VEL_RAMP?	4.4
4	Upload personality module (read set-up parameters from EEPROM)	SYS_UL	4.1
5	Get acceleration feed forward	HW_AFF?	4.5
6	Get velocity feed forward	HW_VFF?	4.5
7	Get hand-wheel (position loop) proportional gain factor	HW_KP?	4.5
8	Get velocity loop gain factor (KV)	VEL_KV?	4.4
10	Get following error warning margins	HW_FEWRN?	4.5
11	Get following error fatal margins	HW_FEFAT?	4.5
12	Get in position band margins	HW_IPOS?	4.5
13	Get maximum following error value	HW_MXFE?	4.5
14	Get DBSC card address	SYS_ADDR?	4.1
15	Read IMAS absolute value register	IMS_AVAL?	4.6
16	Read IMAS number of coarse resolvers register	IMS_NBRES?	4.6
22	Set Jog mode with velocity value and execute	JOG_SET&EXE	4.3
24	Download personality module to DBSC (write new parameters in EEPROM)	SYS_DL	4.1
27	Set hand wheel parameters (Gear ratio & resolution)	HW_PAR	4.5
28	Set acceleration feed forward	HW_AFF=	4.5
29	Set velocity feed forward	HW_VFF=	4.5
30	Set position (hand wheel) loop gain factor	HW_KP=	4.5
31	Set velocity loop gain factor KV	VEL_KV=	4.4
33	Set following error warning margins	HW_FEWRN=	4.5
34	Set following error fatal margins	HW_FEFAT=	4.5
35	Set in position band margins	HW_IPOS=	4.5
36	Set IMAS number of coarse resolvers	IMS_NBRES=	4.6

Table 1
DBSC Command set (operational codes 1 to 36)

Operational code	Function description	Suggested Macro mnemonic	Section
40	Get hardware fault status	SYS_HST?	4.1
42	Get operation mode (Current, Velocity or Hand-Wheel)	SYS_OPMODE?	4.1
43	Get DBSC normal status	SYS_ST?	4.1
44	Get firmware version	SYS_VER?	4.1
45	Get following error status	HW_FEST?	4.5
46	Get DBSC warning status	SYS_WRNST?	4.1
47	Get IMAS measurement status	IMS_ST?	4.6
60	Disable DBSC (software disable)	CMD_DIS	4.2
61	Hold position	CMD_HOLD	4.2
62	Enable DBSC	CMD_ENA	4.2
63	Set current mode	SYS_CURMODON	4.1
64	Set velocity mode	SYS_VELMODON	4.1
65	Set hand wheel mode (or position mode)	SYS_HWMODON	4.1
66	Enable following error / warning monitoring	HW_FEWRNON	4.5
67	Disable following error / warning monitoring	HW_FEWRNOF	4.5
68	Enable In position monitoring	HW_IPOSON	4.5
69	disable In position monitoring	HW_IPOSOF	4.5
70	Reset maximum following error value	HW_MXFERST	4.5
71	Reset following error status bit	HW_FERST	4.5
72	Fault Reset (clear hardware and software fault)	SYS_FRST	4.1
73	Calculate IMAS absolute value	IMS_CAV	4.6
74	Calibrate IMAS	IMS_CAL	4.6
80	Get PLC command	PLC_CMD?	4.7
81	Get PLC flag status	PLC_ST?	4.7
83	Get PLC recording time	PLC_RECTM?	4.7
84	Get PLC recording variables	PLC_RECVAR?	4.7
85	Get PLC Jog velocity 1	PLC_JOGVEL1?	4.7
86	Get PLC Jog velocity 2	PLC_JOGVEL2?	4.7
87	Get PLC Jog time 1	PLC_JOGTM1?	4.7
88	Get PLC Jog time 2	PLC_JOGTM2?	4.7
89	Get PLC Jog flags	PLC_JOGFLGS?	4.7
90	Set PLC command	PLC_CMD=	4.7
92	Clear PLC	PLC_CLR	4.7
93	Set PLC recording time	PLC_RECTM=	4.7
94	Set PLC recording variables	PLC_RECVAR=	4.7
95	Set PLC Jog velocity 1	PLC_JOGVEL1=	4.7
96	Set PLC Jog velocity 2	PLC_JOGVEL2=	4.7
97	Set PLC Jog time 1	PLC_JOGTM1=	4.7
98	Set PLC Jog time 2	PLC_JOGTM2=	4.7
99	Set PLC Jog flags	PLC_JOGFLGS=	4.7

Table 2
DBSC Command set (operational codes 40 to 99)

The application program will send the operational code directly to the DBSC. For example, in Basic the line is written as follows:

```
OUT &H3F8, 88        'This will send the "Header" over the COM 1 port (at address
                      3F8 Hexadecimal) to the DBSC in order to start a communication
                      operation.
```

In programming languages such as Pascal and C, macro files can be written. An example of C language macro file for the operational codes is given bellow; you also find a few other suggestions of macro definitions at the end of it, they may be helpful for some typical DBSC constants. Note that the commands are listed according to their group in alphabetic order. This makes it easier to spot a command by using the macro mnemonic. The same order is respected in section 4 where the communication protocol of each command is explained in detail.

Macro file example:

```
/* Header and Triggers */

#define     Header              'X'     /* 'X' or 88, 'X' is also the value 88 in ASCII */
#define     Trig_read           'T'     /* 'T' or 84, 'T' is also the value 84 in ASCII */
#define     Trig_operation      'T'     /* 'T' or 84, basically the same as above. */

/* GROUP SYSTEM (SYS) */

#define     SYS_ADDR?           14      /* Get DBSC card Address */
#define     SYS_CURMODON        63      /* Set current mode */
#define     SYS_DL              24      /* Download personality module to DBSC */
#define     SYS_FRST            72      /* Fault reset (Clear hardware and software faults) */
#define     SYS_HST?            40      /* Get hardware fault status */
#define     SYS_HWMODON         65      /* Set hand-wheel follower mode (position) */
#define     SYS_MRPOS?          0       /* Get master resolver absolute position */
#define     SYS_OPMOD?          42      /* Get operation mode (Current, Velocity, Hand-Wheel) */
#define     SYS_ST?             43      /* Get DBSC normal status */
#define     SYS_UL              4       /* Upload personality module */
#define     SYS_VELMODON        64      /* Set velocity mode */
#define     SYS_VER?            44      /* Get firmware version */
#define     SYS_WRNST?          46      /* Get warning status */

/* GROUP COMMAND (CMD) */

#define     CMD_DIS             60      /* Disable DBSC */
#define     CMD_ENA             62      /* Enable DBSC */
#define     CMD_HOLD            61      /* Hold position */

/* GROUP JOG */

#define     JOG_SET&EXE         22      /* Set jog mode with velocity value and execute */

/* GROUP VELOCITY (VEL) */

#define     VEL_RAMP?           3       /* Get velocity ramp (acceleration limit */
#define     VEL_KV?             8       /* Get velocity loop gain factor (KV) */
#define     VEL_KV=             31      /* Set velocity loop gain factor (KV) */
#define     VEL_VEL?            2       /* Get actual Velocity */
```

```
/* GROUP Hand-Wheel follower (HW) or position tracking */

#define     HW_AFF?         5     /* Get Acceleration Feed Forward */
#define     HW_AFF=         28    /* Set Acceleration Feed Forward */
#define     HW_FEFAT?       11    /* Get Following Error Fatal limit */
#define     HW_FEFAT=       34    /* Set Following Error Fatal limit */
#define     HW_FERST        71    /* Following Error Reset */
#define     HW_FEST         45    /* Get Following Error Status */
#define     HW_FEWRNOF      67    /* Disable Following Error Warning monitoring */
#define     HW_FEWRNON      66    /* Enable Following Error Warning monitoring */
#define     HW_FEWRN?       10    /* Get Following Warning Error limit */
#define     HW_FEWRN=       33    /* Set Following Warning Error limit */
#define     HW_IPOSOF       69    /* Disable In position monitoring */
#define     HW_IPOSON       68    /* Enable In position monitoring */
#define     HW_IPOS?        12    /* Get In Position band limit */
#define     HW_IPOS=        35    /* Set In Position band limit */
#define     HW_KP?          7     /* Get hand-wheel (position loop) proportional gain */
#define     HW_KP=          30    /* Set hand-wheel (position loop) proportional gain */
#define     HW_MXFE?        13    /* Get Maximal Following Error value */
#define     HW_MXFERST      70    /* Maximal Following Error Reset  */
#define     HW_PAR          27    /* Set hand-wheel parameters (gear ratio & resolution) */
#define     HW_PFE?         1     /* Get actual Position Following Error */
#define     HW_VFF?         6     /* Get Velocity Feed Forward */
#define     HW_VFF=         29    /* Set Velocity Feed Forward */

/* GROUP IMAS (IMS) Inductive Modular Absolute System or multi-resolver system */

#define     IMS_AVAL?       15    /* Read IMAS absolute Value register */
#define     IMS_CAL         74    /* Calibrate IMAS */
#define     IMS_CAV         73    /* Calculate IMAS Absolute Value */
#define     IMS_NBRES?      16    /* Read IMAS number of coarse resolvers register */
#define     IMS_NBRES=      36    /* Set IMAS number of coarse resolvers */
#define     IMS_ST?         47    /* Get IMAS measurement Status */

/* GROUP PLC */

#define     PLC_CLR         92    /* Clear PLC */
#define     PLC_CMD?        80    /* Get PLC command */
#define     PLC_CMD=        90    /* Set PLC command */
#define     PLC_JOGFLGS?    89    /* Get PLC jog flags */
#define     PLC_JOGFLGS=    99    /* Set PLC jog flags */
#define     PLC_JOGTM1?     87    /* Get PLC jog time 1 */
#define     PLC_JOGTM1=     97    /* Set PLC jog time 1 */
#define     PLC_JOGTM2?     88    /* Get PLC jog time 2 */
#define     PLC_JOGTM2=     98    /* Set PLC jog time 2 */
#define     PLC_JOGVEL1?    85    /* Get PLC jog velocity 1 */
#define     PLC_JOGVEL1=    95    /* Set PLC jog velocity 1 */
#define     PLC_JOGVEL2?    86    /* Get PLC jog velocity 2 */
#define     PLC_JOGVEL2=    96    /* Set PLC jog velocity 2 */
#define     PLC_RECTM?      83    /* Get PLC recording time */
#define     PLC_RECTM=      93    /* Set PLC recording time */
#define     PLC_RECVAR?     84    /* Get PLC recording variables */
#define     PLC_RECVAR=     94    /* Set PLC recording variables */
#define     PLC_ST?         81    /* Get PLC flag status */
```

/* Typical DBSC constants */

```
#define     SERVO_CYCLE       0.0004704   /* constant, 470.4 micro seconds */
#define     RES_RESOLUTION    65536       /* constant, 65536 bits par revolution */
#define     PLTY_BYTE_NO      512         /* constant, personality module 512 bytes long */
```

Now, an instruction like **WRITE(JOG_SET&EXE)** will send automatically the operational code **"22"** over the communication port. (In that example, WRITE is also a macro command.) This way make much more simple reading and writing the communication source file. We will use these macro definitions in the communication protocol section.

4. Communication protocol

This section describes the sequence of operation to be executed by the user's application program in order to send the operational codes explained in section 3 and their arguments, and to read the information's being sent back from the DBSC. The operations are written here in "C", the sign "*/" indicates the start of a comment, and the sign "*/" indicates the end of a comment.

4.1 Group system (SYS)

SYS_ADDR? **Get DBSC card address (op. code 14):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received is equals to 88 */
           _Write(SYS_ADDR?)       /* send value 14 over the serial port */
           _Read()                 /* Read 14 back, DBSC acknowledgment */
                                   /* program should check if the value received is equals to 14 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read value returned by the DBSC */

           /* Format:    Unsigned character */
           /* Range:     0..15 (0..F Hexadecimal) */
           /* Units:     None */
```

SYS_CURMODON **Set Current Mode (op. code 63):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(SYS_CURMODON)    /* send value 63 over the serial port */
           _Read()                 /* Read 63 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 63 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

SYS_DL **Download personality module to DBSC (op. code 24):**

- **This operation is valid only while the DBSC is disabled.**
- Prior to start the protocol described bellow, your application program should prompt the operator to enter the different parameters. It is the user responsibility to translate these values in "characters" values (bytes) understandable for the DBSC, to built the safety if the value entered is out of range etc.
- After the "trigger operation" command, the user should wait for 30000 milliseconds before re-enabling the DBSC. This time is required to accomplish the operation (update memory location within the micro controller and write the new set of parameters to the EEPROM. If any HCP command is sent during this time, the data will be lost and the memory locations corrupted.

```
Protocol:  _Write(Header)              /* Send value 88 ('X') over the serial port */
           _Read()                     /* Read 88 back, DBSC acknowledgment */
                                       /*program should check if the value received equals 88 */
           _Write(SYS_DL)                    /* send value 24 over the serial port */
           _Read()                     /* Read 24 back, DBSC acknowledgment */
                                       /* program should check if the value received equals to 24 */

           for (i = 0, i<PLTY_BYTE_NO; i++)  /* initialize i = 0, and while "i" is smaller than 512 (see macro
                                                definitions in section 4), increment "i" value and loop through
                                                the following block section. */

           {                                 /* Block begin */
               _Write();                     /* Send i'th byte to the DBSC */
               _Read();                      /* Read back the i'th byte, (acknowledgment )*/
                                             /* program should check if the value received is equal to the
                                                value sent */
           }                                 /* End of block */

           _Write(Trig_operation)      /* Send value 84 ('T') over the serial port */
           _Read()                     /* Read 84 ('T') back, DBSC acknowledgment */
```

SYS_FRST **Fault reset (Clear hardware and software faults (op. code 72):**

```
Protocol:  _Write(Header)              /* Send value 88 ('X') over the serial port */
           _Read()                     /* Read 88 back, DBSC acknowledgment */
                                       /*program should check if the value received equals 88 */
           _Write(SYS_FRST)            /* send value 72 over the serial port */
           _Read()                     /* Read 72 back, DBSC acknowledgment */
                                       /* program should check if the value received equals to 24 */
           _Write(Trig_operation)      /* Send value 84 ('T') over the serial port */
           _Read()                     /* Read 84 ('T') back, DBSC acknowledgment */
```

SYS_HST? **Get hardware fault status (op. code 40):**

This is a two bytes word from which each bit is dedicated to a specific status (discrete flag). Refer to appendix D3 and D1 for explanation of each fault.

Meaning of each bit:				
	bit 0	Over current	bit 8	EPROM error
	bit 1	Over voltage	bit 9	RAM error
	bit 2	RDC error	bit 10	Fault relay error
	bit 3	Power fault	bit 11	EAF error
	bit 4	BPS fault	bit 12	N/A
	bit 5	Over voltage (15V)	bit 13	Missing interrupt
	bit 6	Under voltage (15V)	bit 14	Software version
	bit 7	EEPROM error	bit 15	CW / CCW error

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(SYS_HST?)        /* send value 40 over the serial port */
           _Read()                 /* Read 40 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 40 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read LSB from DBSC */
                                   /* program should store the received LSB */
           _Write()                /* Send LSB back, DBSC acknowledgment  */
           _Read()                 /* Read MSB from DBSC */
                                   /* program should store the received MSB*/

           /* Compute the error code /*

           /* Format:             Unsigned integer flag array */
           /* Dimension:          discrete flags (bit array containing bit error identifiers) */
           /* Units:              None */
```

SYS_HWMODON **Set hand-wheel follower mode (position) (op. code 65):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(SYS_HWMODON)     /* send value 65 over the serial port */
           _Read()                 /* Read 65 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 65 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

SYS_MRPOS? **Get Master Resolver Position (op. code 0):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received is equal to 88 */
           _Write(SYS_MRPOS?)      /* send value 0 over the serial port */
           _Read()                 /* Read 0 back, DBSC acknowledgment */
                                   /* program should check if the value received is equal to 0 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back MSB to acknowledge the DBSC */

           /* Compute the absolute position: MSB * 256 + LSB = POS /*

           /* Format: Unsigned integer */
           /* Range:  0..65535 (0..FFFF) */
           /* Units:  1/65536 of a revolution (resolver bits) */

           /* Conversion in degrees = POS * 360 / RES_RESOLUTION */
```

SYS_OPMOD? **Get operation mode (Current, Velocity, Hand-Wheel) (op. code 42):**

This is a one byte word with a range of 0..2. The returned values have the following meanings:

0	Current mode
1	Velocity mode
2	Hand wheel mode (position)

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(SYS_OPMOD?)      /* send value 42 over the serial port */
           _Read()                 /* Read 42 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 42 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read byte value from DBSC */
                                   /* program should store the received value */

           /* Format:  Unsigned character */
           /* Range:   0..2 */
           /* Units:   None */
```

SYS_ST? **Get DBSC normal status (op. code 43):**

This is a one byte word with a range of 0..0x40. The returned values have the following meanings:

DEC 0	HEX 0	Enable (normal mode)
DEC 4	HEX 4	CW limit
DEC 8	HEX 8	CCW limit
DEC 16	HEX 10	Disable
DEC 32	HEX 20	Hold position
DEC 64	HEX 40	Jog mode

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back. DBSC acknowledgment */
                                  /* program should check if the value received equals 88 */
          _Write(SYS_ST)          /* send value 43 over the serial port */
          _Read()                 /* Read 43 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 43 */
          _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
          _Read()                 /* Read byte value from DBSC */
                                  /* program should store the received value */

          /* Format:      Unsigned character */
          /* Range:       0..64 (0..40 Hexadecimal) */
          /* Units:       None */
```

SYS_UL **Upload personality module (op. code 4):**

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 4 */
          _Write(UL)              /* send value 4 over the serial port */
          _Read()                 /* Read 4 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 4 */
          _Write(Trig_read)   /* Send value 84 ('T') over the serial port */

          for (i = 0, i<PLTY_BYTE_NO; i++)  /* initialize i = 0, and while "i" is smaller than 512 increment "i"
                                               value and loop through the following block section. */
          {                               /* Block begin */
              User_Buffer[i] = Read();    /* Read the i'th byte (User_Buffer is a macro command which
                                             stores the read value into an appropriate memory location) */
              _Write()                    /* Send back i'th byte to acknowledge the DBSC */
          }                               /* End of block */

          /* 512 unsigned character values are sent back (units in bits)*/

          /* Format:      Unsigned characters */
          /* Units:       None */

          /* It is the user responsibility to allocate the "User_Buffer" memory upon each call */
```

SYS_VELMODON **Set velocity mode (op. code 64):**

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 88 */
          _Write(SYS_VELMODON)    /* Send value 64 over the serial port */
          _Read()                 /* Read 64 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 64 */
          _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
          _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

SYS_VER? **Get firmware version (op. code 44):**

This is a one byte word (unsigned character type).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(SYS_VER)         /* send value 44 over the serial port */
           _Read()                 /* Read 44 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 44 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read byte value from DBSC */
                                   /* program should store the received value */

           /* Format:      Unsigned character */
           /* Range:       0..255 (0..FF Hexadecimal) */
           /* Units:       None (software version number) */
```

SYS_WRNST? **Get warning status (op. code 46):**

This is a two bytes word from which each bit is dedicated to a specific status (discrete flag). Refer to appendix D3 and D1 for explanation of each warning.

Meaning of each bit:	bit 0	DBSC Over temperature	bit 8	FE fatal warning
	bit 1	Motor Over temperature	bit 9	Over speed warning
	bit 2	I2t warning	bit 10	Velocity following error
	bit 3	Burn in mode	bit 11	N/A
	bit 4	CCW warning	bit 12	N/A
	bit 5	CW warning	bit 13	N/A
	bit 6	In position	bit 14	N/A
	bit 7	FE warning	bit 15	N/A

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 88 */
           _Write(SYS_WRNST)       /* send value 46 over the serial port */
           _Read()                 /* Read 46 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 46 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the warning status code /*

           /* Format:      Unsigned integer flag array */
           /* Dimension:   discrete flags (bit array containing bit warning identifiers) */
           /* Units:       None */
```

4.2 Group COMMAND (CMD)

CMD_DIS **Disable DBSC (op. code 60):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(CMD_DIS)         /* send value 60 over the serial port */
           _Read()                 /* Read 60 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 60 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

CMD_ENA **Enable DBSC (op. code 62):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 88 */
           _Write(CMD_ENA)         /* send value 62 over the serial port */
           _Read()                 /* Read 62 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 62 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

CMD_HOLD **Hold position (op. code 61):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 88 */
           _Write(CMD_HOLD)        /* send value 61 over the serial port */
           _Read()                 /* Read 61 back, DBSC acknowledgment */
                                   /* program should check if the value received equals to 61 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

4.3 Group JOG

JOG_SET&EXE **Set jog mode with velocity value and execute (op. code 22):**

Prior to start the protocol described bellow, your application program should prompt the operator to enter the velocity value in percent time 100 (signed integer with range between -10000 to +10000). It is the user responsibility to translate this "signed integer" value (16 bits) in two "character" values (16 bits word) understandable for DBSC (bits manipulation), to built the safety if the value entered is out of range etc. (see examples in section 5)

```
Protocol:  _Write(Header)            /* Send value 88 ('X') over the serial port */
           _Read()                   /* Read 88 back, DBSC acknowledgment */
                                     /*program should check if the value received equals 88 */
           _Write(JOG_SET&EXE)       /* send value 22 over the serial port */
           _Read()                   /* Read 22 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 22 */
           _Write()                  /* Send MSB of velocity percentage value */
           _Read()                   /* Read back MSB (DBSC acknowledgment .)*/
                                     /*program should check if the value received is equals to the value
                                     sent */
           _Write()                  /* Send LSB of velocity percentage value */
           _Read()                   /* Read back LSB (DBSC acknowledgment .)*/
                                     /*program should check if the value received is equals to the value
                                     sent */
           _Write(Trig_operation)    /* Send value 84 ('T') over the serial port */
           _Read()                   /* Read 84 back ('T'), DBSC acknowledgment */
                                     /* program should check if the value received equals 84 */

           /* Format:     Signed integer */
           /* Range:      -10000..10000 (D8F0..2710 Hexadecimal) */
           /* Units:      Percentage of max. RPM / 100 */
           /*             Example: for 1.5%, write the value 150 to the DBSC. */
```

4.4 Group VELOCITY (VEL)

VEL_RAMP? **Get velocity ramp (acceleration limit) (op. code 3):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(VEL.RAMP?)       /* send value 3 over the serial port */
           _Read()                 /* Read 3 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 3 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the ramp value: MSB * 256 + LSB = RAMP /*

           /* Format:             Unsigned integer */
           /* Range:              0..10000 (0..2710 Hexadecimal) */
           /* Units:              in percentage time 100 */
```

VEL_KV? **Get velocity loop gain factor (KV) (op. code 8):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(VEL_KV?)         /* send value 8 over the serial port */
           _Read()                 /* Read 8 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 8 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read value returned by the DBSC */

           /* Format:             Unsigned characters */
           /* Range:              50..200 (32..64 Hexadecimal) */
           /* Units:              None (in percentage of full velocity KV gain) */
```

VEL_KV= **Set velocity loop gain factor (KV) (op. code 31):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(VEL_VEL=)        /* send value 31 over the serial port */
           _Read()                 /* Read 31 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 31 */
           _Write()                /* Send " value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 back ('T'), DBSC acknowledgment */
                                   /* program should check if the value received equals 84 */

           /* Format:             Unsigned characters */
           /* Range:              50..200 (32..C8 Hexadecimal) */
           /* Units:              None (in percentage of full velocity Kp gain) */
```

VEL_VEL? **Get actual Velocity (op. code 2):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(VEL_VEL?)        /* send value 2 over the serial port */
           _Read()                 /* Read 2 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 2 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the velocity value: MSB * 256 + LSB = VEL */

           /* Format:     Signed integer */
           /* Range:      -32768..32767 (8000..7FFF Hexadecimal) */
           /* Units:      1/16384 of a revolution / servo cycle (resolver bits * 4 / servo cycle) */

           /* Conversion in RPM = (VEL * 60 * 4) / (SERVO_CYCLE * RESOLVER_RESOLUTION) */
```

4.5 Group Hand-Wheel follower (HW) / (position loop)

HW_AFF? **Get Acceleration Feed Forward (op. code 5):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_AFF?)         /* send value 5 over the serial port */
           _Read()                 /* Read 5 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 5 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read value returned by the DBSC */

           /* Format:    Unsigned characters */
           /* Range:     0..100 (0..64 Hexadecimal) */
           /* Units:     None (in percentage of full acceleration feed forward gain) */
```

HW_AFF= **Set Acceleration Feed Forward (op. code 28):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_AFF=)         /* send value 28 over the serial port */
           _Read()                 /* Read 28 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 28 */
           _Write()                /* Send "HW_AFF" value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:    Unsigned character */
           /* Range:     0..100 (0..64 Hexadecimal) */
           /* Unit:      None (Percentage of full acceleration feed forward gain) */
```

HW_FEFAT? **Get Following Error Fatal limit (op. code 11):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_FEFAT?)       /* send value 11 over the serial port */
           _Read()                 /* Read 11 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 11 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the fatal following error margin value: (MSB * 256 + LSB = FFE /*

           /* Format:    Unsigned integer */
           /* Range:     0..32767 (0..0x7FFF Hexadecimal) */
           /* Units:     1/4096 of motor revolution */
```

HW_FEFAT= **Set Following Error Fatal limit (op. code 34):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_FEFAT=)       /* send value 34 over the serial port */
           _Read()                 /* Read 34 back, DBSC acknowledgment */
                                   /* program should check if the value received  equals 34 */
           _Write()                /* Send "FEFAT" MSB value to DBSC */
           _Read()                 /* Read back the MSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send "FEFAT" LSB value to DBSC */
           _Read()                 /* Read back the LSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:      Unsigned integer */
           /* Range:       0..32767 (0..7FFF Hexadecimal) */
           /* Units:       1/4096 of motor revolution */
```

HW_FERST **Following Error Reset (op. code 71):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_FERST)        /* send value 71 over the serial port */
           _Read()                 /* Read 71 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 71 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_FEST **Get Following Error Status (op. code 45):**

This is a one byte word (unsigned character type) with a range of 0..3. The returned values have the following meanings:

0	Normal state
1	In position
2	Following error warning
3	Fatal following error

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(FE_STATUS)       /* send value 45 over the serial port */
           _Read()                 /* Read 45 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 45 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read byte value from DBSC */
                                   /* program should store the received value */

           /* Format:      Unsigned character */
           /* Range:       0..3 */
           /* Units:       None */
```

HW_FEWRNOF **Disable Following Error Warning monitoring (op. code 67):**

```
Protocol:  _Write(Header)            /* Send value 88 ('X') over the serial port */
           _Read()                   /* Read 88 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 88 */
           _Write(HW_FEWRNOF)        /* send value 67 over the serial port */
           _Read()                   /* Read 67 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 67 */
           _Write(Trig_operation)    /* Send value 84 ('T') over the serial port */
           _Read()                   /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_FEWRNON **Enable Following Error Warning monitoring (op. code 66):**

```
Protocol:  _Write(Header)            /* Send value 88 ('X') over the serial port */
           _Read()                   /* Read 88 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 88 */
           _Write(HW_FEWRNON)        /* send value 66 over the serial port */
           _Read()                   /* Read 66 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 66 */
           _Write(Trig_operation)    /* Send value 84 ('T') over the serial port */
           _Read()                   /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_FEWRN? **Get Following Warning Error limit (op. code 10):**

```
Protocol:  _Write(Header)            /* Send value 88 ('X') over the serial port */
           _Read()                   /* Read 88 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 10 */
           _Write(HW_FEWRN?)         /* send value 10 over the serial port */
           _Read()                   /* Read 10 back, DBSC acknowledgment */
                                     /* program should check if the value received equals 10 */
           _Write(Trig_read)         /* Send value 84 ('T') over the serial port */
           _Read()                   /* Read first byte (LSB) from DBSC */
                                     /* program should store the received value */
           _Write()                  /* Send back LSB to acknowledge the DBSC */
           _Read()                   /* Read second byte (MSB) from DBSC */
                                     /* program should store the received value */

           /* Compute the warning followong error margin value: (MSB * 256 + LSB = FEWARN /*

           /* Format:    Unsigned Integer */
           /* Range:     0..32767 (0..7FFF Hexadecimal) */
           /* Units:     1/4096 of motor revolution */
```

HW_FEWRN= **Set Following Warning Error limit (op. code 33):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_FEWRN=)       /* send value 33 over the serial port */
           _Read()                 /* Read 33 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 33 */
           _Write()                /* Send "FEWRN" MSB value to DBSC */
           _Read()                 /* Read back the MSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send "FEWARN" LSB value to DBSC */
           _Read()                 /* Read back the LSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)          /* Send value 84 ('T') over the serial port */

           /* Format:    Unsigned integer */
           /* Range:     0..32767 (0..7FFF Hexadecimal) */
           /* Units:     1/4096 of motor revolution */
```

HW_IPOSOF **Disable In position monitoring (op. code 69):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_IPOSON)       /* send value 69 over the serial port */
           _Read()                 /* Read 69 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 69 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_IPOSON **Enable In position monitoring (op. code 68):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_IPOSON)       /* send value 68 over the serial port */
           _Read()                 /* Read 68 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 68 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_IPOS? **Get In Position band limit (op. code 12):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_IPOS?)        /* send value 12 over the serial port */
           _Read()                 /* Read 12 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 12 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the "in position band" value: (MSB * 256 + LSB = IPOS /*

           /* Format:             Unsigned integer */
           /* Range:              0..32767 (0..7FFF Hexadecimal) */
           /* Units:              1/4096 of motor revolution */
```

HW_IPOS= **Set In Position band limit (op. code 35):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_IPOS=)        /* send value 35 over the serial port */
           _Read()                 /* Read 35 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 35 */
           _Write()                /* Send "IPOS" MSB value to DBSC */
           _Read()                 /* Read back the MSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send "IPOS" LSB value to DBSC */
           _Read()                 /* Read back the LSB value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:             Unsigned integer */
           /* Range:              0..32767 (0..7FFF Hexadecimal) */
           /* Units:              1/4096 of motor revolution */
```

HW_KP? **Get hand-wheel (position loop) proportional gain (op. code 7):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_KP?)          /* send value 7 over the serial port */
           _Read()                 /* Read 7 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 7 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read value returned by the DBSC */

           /* Format:             Unsigned characters */
           /* Range:              50..200 (32..C8 Hexadecimal) */
           /* Units:              None (in percentage of full position Kp gain) */
```

HW_KP= **Set hand-wheel (position loop) proportional gain (op. code 30):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_KP=)          /* send value 30 over the serial port */
           _Read()                 /* Read 30 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 30 */
           _Write()                /* Send "KP" value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:    Unsigned characters */
           /* Range:     50..200 (32..C8 Hexadecimal) */
           /* Units:     None (in percentage of full position Kp gain) */
```

HW_MXFE? **Get Maximal Following Error value (op. code 13):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_MXFE?)        /* send value 13 over the serial port */
           _Read()                 /* Read 13 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 13 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read first byte (LSB) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read second byte (MSB) from DBSC */
                                   /* program should store the received value */

           /* Compute the "maximum following error " value: (MSB * 256 + LSB = MXFE /*

           /* Format:    Unsigned integer */
           /* Range:     0..32767 (0..7FFF Hexadecimal) */
           /* Units:     1/4096 of motor revolution */
```

HW_MXFERST **Maximal Following Error Reset (op. code 70):**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_MXFERST)      /* send value 70 over the serial port */
           _Read()                 /* Read 70 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 70 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

HW_PAR **Set hand-wheel parameters (gear ratio & resolution) (op. code 27):**

Prior to start the protocol described bellow, the application program should prompt the operator to enter the values for the "Hand wheel encoder resolution" and the "Electronic gear ratio". It is the user responsibility to translate these values in "characters" values (bytes) understandable for the DBSC. to built the safety if the value entered is out of range etc.

User inputs Definitions:

HW_RES	(unit: counts / revolution,	range: 1..7FFF hexa.)
HW_GEAR	(unit: none	range: -9999 99999 decimal)

preliminary computing:

To control the hand wheel parameters (Encoder input resolution and Electronic gear value), the user has to perform the following computations:

```
/* Declare "RES_RESOLUTION", "HW_GEAR" and "HW_RES" as double words (64 bits floating point) */
double RES_RESOLUTION = 65536:
double HW_GEAR:
double HW_RES:
```

```
/* Case of Hand wheel encoder */

/* calculates the ratio between the resolver (slave) and the encoder input (master) */
HW_TO_RES_RATIO = (RES_RESOLUTION * 4 * HW_GEAR / (HW_RES * 4))
HW_RES &= ~0x8000  /* Set MSB of "HW_RES" word to 0, this selects the "A leads B" mode. */

/* Example: gear ratio = 0.5, encoder resolution: 1024 (lines) * 4 (quadrature) = 4096 (counts). */
/* The ratio is:                65536  * 4 * 0.5 / (1024 *4) = 32 */
/* This tells the motor that, for each hand wheel encoder count, it must move 32 resolver bits. */
/* In other word, for each revolution of the hand wheel, it must execute 1/2 revolution */
```

```
/* Case of Pulse and direction signals */

/* calculates the ratio between the resolver (slave) and the pulse signal input (master) */
HW_TO_RES_RATIO = (RES_RESOLUTION * 4 * HW_GEAR / (HW_RES * 2))
HW_RES |= 0x8000     /* Set MSB of "HW_RES" word to 1, this selects the Pulse & direction mode. */

/* Example: gear ratio = 0.5, pulse signal resolution: 1024 (pulses) * 2 (rising and falling edge) = 2048 cts. */
/* The ratio is:                65536  * 4 * 0.5 / (1024 *2) = 64 */
/* This tells the motor, that for each rising and falling edge of the input signal, it must move 64 resolver bits*/
/* In other word, for each 1024 pulses, it must execute 1/2 revolution */
```

```
If HW_RES is power of 2 (512, 1024, 2048...)
        fixed_truncate = 1.5
else
        fixed_truncate = 1.0

if (fabs(HW_TO_RES_RATIO) > 1.0)
        HW_GEAR_VOLUME = (int) (log (fabs (HW_TO_RES_RATIO)) / log (2.0) + fixed_truncate)
else
        HW_GEAR_VOLUME = (int) (log (fabs (HW_TO_RES_RATIO)) / log (2.0))

HW_GRSH =   (int) (1.5 -HW_GEAR_VOLUME);                    /* GRSH means gear shift */
HW_GRFX =   (int) (HW_TO_RES_RATIO * pow(2.0, HW_GRSH));    /* GRFX means gear fix */
HW_GRRD =   -(int) pow(2.0, HW_GRSH);                       /* GRRD means gear rounding */
```

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(HW_PAR)          /* send value 27 over the serial port */
           _Read()                 /* Read 27 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 27 */

           _Write()                /* Send "HW_GRSH" MSB value to DBSC */
           _Read()                 /* Read back the MSB (acknowledgment .) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send " HW_GRSH" LSB value to DBSC */
           _Read()                 /* Read back the LSB (acknowledgment .) */
                                   /* program should check if the value received is equal to the value
                                   sent */

           _Write()                /* Send "HW_GRRD" MSB to DBSC */
           _Read()                 /* Read back the MSB (acknowledgment .) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send " HW_GRRD" LSB to DBSC */
           _Read()                 /* Read back the LSB (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */

           _Write()                /* Send "HW_GRFX" MSB value to DBSC */
           _Read()                 /* Read back the MSB (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send " HW_GRFX" LSB value to DBSC */
           _Read()                 /* Read back the LSB (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */

           _Write()                /* Send "HW_RES" MSB value to DBSC */
           _Read()                 /* Read back the MSB (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write()                /* Send "HW_RES" LSB value to DBSC */
           _Read()                 /* Read back the LSB (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */

           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
```

HW_PFE? **Get actual Position Following Error (op. code 1):**

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 88 */
          _Write(HW_PFE?)         /* send value 1 over the serial port */
          _Read()                 /* Read 1 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 1 */
          _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
          _Read()                 /* Read LSB from DBSC */
                                  /* program should store the received value */
          _Write()                /* Send back LSB to acknowledge the DBSC */
          _Read()                 /* Read MSB from DBSC */
                                  /* program should store the received value */

          /* Compute the position following error: MSB * 2 EXP8 + LSB = FE /*

          /* Format:   Signed integer */
          /* Range:    -32768..32767 (8000..7FFF hexadecimal) */
          /* Units:    1/4096 of a revolution */

          /* Conversion in degrees = FE * 360 / 4096 */
```

HW_VFF? **Get Velocity Feed Forward (op. code 6):**

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 6 */
          _Write(HW_VFF?)         /* send value 6 over the serial port */
          _Read()                 /* Read 6 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 6 */
          _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
          _Read()                 /* Read value returned by the DBSC */

          /* Format:   Unsigned characters */
          /* Range:    0..100 (0..64 Hexadecimal) */
          /* Units:    None (in percentage of full velocity feed forward gain) */
```

HW_VFF= **Set Velocity Feed Forward (op. code 29):**

```
Protocol: _Write(Header)          /* Send value 88 ('X') over the serial port */
          _Read()                 /* Read 88 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 88 */
          _Write(HW_VFF)          /* send value 29 over the serial port */
          _Read()                 /* Read 29 back, DBSC acknowledgment */
                                  /* program should check if the value received equals 29 */
          _Write()                /* Send " value to DBSC */
          _Read()                 /* Read back the value (acknowledgment .)*/
                                  /* program should check if the value received is equal to the value
                                     sent */
          _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

          /* Format:   Unsigned characters */
          /* Range:    0..100 (0..0x64 Hexadecimal) */
          /* Units:    None (in percentage of full velocity feed forward gain) */
```

4.6 Group IMAS (IMS) / (Inductive Modular Absolute System) / (or multi-resolvers)

The command described in this section are only needed if the IMAS option is installed (option Ebx) Refer to chapter 6 for the IMAS description and operating principle.

IMS_AVAL? **Read IMAS Absolute Value register (op. code 15):**

This is a 32 bits value, the 16 upper bits represent an integer number of motor revolutions (0 to 65535 revolution) , and the 16 lower bits a fractional part of a revolution (1/65536 of a revolution).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_AVAL?)       /* send value 15 over the serial port */
           _Read()                 /* Read 15 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 15 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first word LSB (LSB1) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the first word MSB (MSB1) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back MSB to acknowledge the DBSC */
           _Read()                 /* Read the second word LSB (LSB2) from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second word MSB (MSB2) from DBSC */
                                   /* program should store the received value */

           /* Compute the IMAS Absolute Value:
              MSB2 * 2 EXP24 + LSB2 * 2 EXP16 + MSB1 * 2 EXP8 + LSB1 = IMS_AVAL /*

           /* Format:    Unsigned integer */
           /* Range:     0..4294967295 (0..FFFFFFF hexadecimal) */
           /* Units:     1/65536 of a revolution */

           /* Conversion in degrees = IMS_AVAL * 360 / 65536 */
```

IMS_CAL **Calibrate IMAS (op. code 74):**

While calibration execution, the DBSC must be disabled and the motor shaft must be in stand still position (activate the brake if the motor is equipped with this option) . This set-up command must be performed once after the first power-up of the system and each time that a new motor is installed and linked to the DBSC. Once executed, verify the integrity of the calculated data by using the IMAS measurement status (see "IMS_ST" command).

The calibration operation measures the absolute position of each IMAS resolver, calculates the correlation and save the offset values found between each resolver in the EEPROM. Then a "calculate IMAS absolute value" command is issued. DBSC takes about 5 seconds to execute this instruction.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_CAL)         /* send value 74 over the serial port */
           _Read()                 /* Read 74 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 74 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

IMS_CAV **Calculate IMAS Absolute Value (op. code 73):**

While calculating the IMAS absolute value, the DBSC must be disabled and the motor shaft must be in stand still position (activate the brake if the motor is equipped with this option). This command must be performed after each power-up of the DBSC (excepted the very first one before set-up is done). Once executed, verify the integrity of the calculated data by using the IMAS measurement status (see "IMS_ST" command).

The IMAS absolute value calculation measures the absolute position of each IMAS resolver, Reads the offsets values saved in the EEPROM when the "Calibration" command was executed, and calculates the absolute value. DBSC takes about 2.5 seconds to executed this instruction.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_CAV)         /* send value 73 over the serial port */
           _Read()                 /* Read 73 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 73 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment */
```

IMS_NBRES? **Read IMAS Number of coarse Resolvers register (op. code 16):**

This command returns the number of coarse resolvers as defined by the last "set IMAS number of coarse resolvers" command. **The value returned by this command will be wrong if the "Set IMAS number of coarse resolvers" command was omitted during the initial set-up procedure.**

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_NBRES?)      /* send value 16 over the serial port */
           _Read()                 /* Read 16 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 16 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read "NBRES" value from DBSC */

           /* Format:    Unsigned integer */
           /* Range:     0..4 */
           /* Units:     None */
```

IMS_NBRES= **Set IMAS Number of coarse Resolvers (op. code 36):**

This is the very first command which must be performed for IMAS set-up. It tells the DBSC how many resolvers are installed in the motor. The DBSC saves this value in EEPROM, and uses it for the "Calibration" operation and for any further IMAS absolute value calculation. It only needs to be executed once during set-up procedure.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_NBRES=)      /* send value 36 over the serial port */
           _Read()                 /* Read 36 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 36 */
           _Write(IMS_NBRES)       /* Send " value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:    Unsigned characters */
           /* Range:     0..4 */
           /* Units:     None */
```

IMS_ST? **Get IMAS measurement Status (op. code 47):**

It is highly recommended to verify the measurement status after each "Calibration" command and each "IMAS Absolute Value" calculation. It reports to the Host any problem which may has occurred during a measurement operation.

This is a one byte word with a range of 0..80 hexadecimal. The returned values have the following meanings:

DEC 0	HEX 00	No Measurement was performed yet.
DEC 1	HEX 01	DBSC is not disabled.
DEC 2	HEX 02	Movement while measuring.
DEC 3	HEX 03	Measurement still in process.
DEC 4	HEX 04	IMAS busy.
DEC 5	HEX 05	Disturbances while reading resolvers.
DEC 6	HEX 06	Movement after measurement.
DEC 7	HEX 07	IMAS option is invalid.
DEC 8	HEX 08	Calculation error.
DEC 9	HEX 09	Illegal coarse resolver number (more than 4 or less than 0).
DEC 10	HEX 0A	Resolver error while measurement.
DEC 16	HEX 10	Result exists, good correlation.
DEC 17	HEX 11	Result exists, bad correlation.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(IMS_ST?)         /* send value 47 over the serial port */
           _Read()                 /* Read 47 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 47 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read byte value from DBSC */
                                   /* program should store the received value */

           /* Format:    Unsigned character */
           /* Range:     0..17 (0..11 Hexadecimal) */
           /* Units:     None */
```

4.7 Group Programmable Logic Controller (PLC)

PLC_CLR **Clear PLC (op. code 92):**

This function clears the PLC. No argument is needed.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_OUTCLR)      /* send value 92 over the serial port */
           _Read()                 /* Read 92 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 92 */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read 84 ('T') back, DBSC acknowledgment  */
```

PLC_CMD? **Get PLC command (op. code 80):**

Reads which is the programmed condition that triggers the specified action. In other words, it reads the condition code linked to the specified action code in the array of the PLC commands. The returned value is the condition code that triggers the related action. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_CMD?)        /* send value 80 over the serial port */
           _Read()                 /* Read 80 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 80 */
           _Write(ACT_COD)         /* Send the action code to the DBSC */
           _Read()                 /* Read back the action code, DBSC acknowledgment */
                                   /* program should check if the value equals the one sent */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the condition code number from DBSC */
                                   /* program should store the received value */

           /* Format:    unsigned character */
           /* Range:     0..255 (0..FF hexadecimal) */
           /* Units:     None */
```

PLC_CMD= **Set PLC command (op. code 90):**

This function programs the PLC. It has two arguments:

1) "Action Code" - Which action to program.
2) "Condition Code" - Which event will trig the action.

It is highly recommended to program the action code 0, which enables the PLC, at last. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_CMD=)        /* send value 90 over the serial port */
           _Read()                 /* Read 90 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 90 */
           _Write(ACT_COD)         /* Send the Action Code to the DBSC */
           _Read()                 /* Read back the Action Code, DBSC acknowledgment */
                                   /* program should check if the value equals the one sent */
           _Write(COND_COD)        /* Send the Condition Code to the DBSC */
           _Read()                 /* Read back the Condition Code, DBSC acknowledgment */
                                   /* program should check if the value equals the one sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
```

PLC_JOGFLGS? **Get PLC jog flags (op. code 89):**

This is a one byte word. Each bit is dedicated to a specific status (discrete flag). The returned value indicates which "PLC jog" structure is selected: "CONTINUOUS", "INCREMENTAL", "SQUARE WAVE".

Meaning of each bit:		
	bit 0	Not used.
	bit 1 = 0	"SQUARE WAVE" setting is released, "CONTINUOUS" and "INCREMENTAL" settings are enabled.
	bit 1 = 1	Jog 1 and 2 are in "SQUARE WAVE mode", "CONTINUOUS" and "INCREMENTAL" settings are disabled.
	bit 2 = 0	Jog 1 is in "INCREMENTAL mode". Jog 2 is as last set.
	bit 2 = 1	Jog 1 is in "CONTINUOUS mode". Jog 2 is as last set.
	bit 3 = 0	Jog 2 type is in "INCREMENTAL mode". Jog 1 is as last set.
	bit 3 = 1	Jog 2 type is in "CONTINUOUS mode". Jog 1 is as last set.
	bits 4 to 7	Not used.

```
Protocol:  _Write(Header)              /* Send value 88 ('X') over the serial port */
           _Read()                     /* Read 88 back, DBSC acknowledgment */
                                       /* program should check if the value received equals 88 */
           _Write(PLC_JOGFLGS?)        /* send value 89 over the serial port */
           _Read()                     /* Read 89 back, DBSC acknowledgment */
                                       /* program should check if the value received equals 89 */
           _Write(Trig_read)           /* Send value 84 ('T') over the serial port */
           _Read()                     /* Read byte value from DBSC */
                                       /* program should store the received value */

           /* Format:      Unsigned character flag array */
           /* Range:       discrete flags (bit array containing bit jog flags identifiers) */
           /* Units:       None */
```

PLC_JOGFLGS= **Set PLC jog flags (op. code 99):**

This is a one byte word. Each bit is dedicated to a specific status (discrete flag).

Meaning of each bit:		
	bit 0	Not used.
	bit 1 = 0	Release "SQUARE WAVE" setting. Enable "CONTINUOUS" and "INCREMENTAL" settings.
	bit 1 = 1	Set jog 1 and 2 to "SQUARE WAVE mode". Disable "CONTINUOUS" and "INCREMENTAL" settings.
	bit 2 = 0	Set jog 1 to "INCREMENTAL mode". Jog 2 type stays as last set.
	bit 2 = 1	Set jog 1 to "CONTINUOUS mode". Jog 2 type stays as last set.
	bit 3 = 0	Set jog 2 to "INCREMENTAL mode". Jog 1 type stays as last set.
	bit 3 = 1	Set jog 2 to "CONTINUOUS mode". Jog 1 type stays as last set.
	bits 4 to 7	Not used.

```
Protocol:  _Write(Header)              /* Send value 88 ('X') over the serial port */
           _Read()                     /* Read 88 back, DBSC acknowledgment */
                                       /* program should check if the value received equals 88 */
           _Write(PLC_JOGFLGS=)        /* send value 99 over the serial port */
           _Read()                     /* Read 99 back, DBSC acknowledgment */
                                       /* program should check if the value received equals 99 */
           _Write(JOGFLGS)             /* Send "JOGFLG" value over the serial port */
           _Read()                     /* Read "JOGFLG" byte value from DBSC */
                                       /* program should check if the value received equals "JOGFLG" */
           _Write(Trig_operation)      /* Send value 84 ('T') over the serial port */

           /* Format:      Unsigned character flag array */
           /* Range:       discrete flags (bit array containing bit jog flags identifiers) */
           /* Units:       None */
```

PLC_JOGTM1? **Get PLC jog time 1(op. code 87):**

This variable is used only for the PLC incremental jog mode. It is the time in tens of milliseconds for the first jog step.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGTM1?)     /* send value 87 over the serial port */
           _Read()                 /* Read 87 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 87 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first byte LSB from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second byte MSB from DBSC */
                                   /* program should store the received value */

           /* Compute the time value:
                                   MSB * 2 EXP8 + LSB = IJOGTM1 /*

           /* Format:             Unsigned integer */
           /* Range:              0..65535 (0..FFFF hexadecimal) */
           /* Units:              1/10 of a millisecond */
```

PLC_JOGTM1= **Set PLC jog time 1(op. code 97):**

This variable is used only for the PLC incremental jog mode. It is the time in tens of milliseconds for the first jog step.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGTM1=)     /* send value 97 over the serial port */
           _Read()                 /* Read 97 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 97 */
           _Write(MSB_JOGTM1)      /* Send " the "JOGTM1" MSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(LSB_JOGTM1)      /* Send " the "JOGTM1" LSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:             Unsigned integer */
           /* Range:              0..65535 */
           /* Units:              1/10 milliseconds */
```

PLC_ JOGTM2? Get PLC jog time 2(op. code 88):

This variable is used only for the PLC incremental jog mode. It is the time in tens of milliseconds for the second jog step.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGTM2?)     /* send value 88 over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first byte LSB from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second byte MSB from DBSC */
                                   /* program should store the received value */

           /* Compute the time value:
                                   MSB * 2 EXP8 + LSB = IJOGTM2 /*

           /* Format:     Unsigned integer */
           /* Range:      0..65535 (0..FFFF hexadecimal) */
           /* Units:      1/10 of a millisecond */
```

PLC_ JOGTM2= Set PLC jog time 2(op. code 98):

This variable is used only for the PLC incremental jog mode. It is the time in tens of milliseconds for the second jog step.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGTM2=)     /* send value 98 over the serial port */
           _Read()                 /* Read 98 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 98 */
           _Write(MSB_JOGTM2)      /* Send " the "JOGTM2" MSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(LSB_JOGTM2)      /* Send " the "JOGTM2" LSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:     Unsigned integer */
           /* Range:      0..65535 */
           /* Units:      1/10 milliseconds */
```

PLC_JOGVEL1? Get PLC jog velocity 1(op. code 85):

Reports the value of jog 1 velocity. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGVEL1?)    /* send value 85 over the serial port */
           _Read()                 /* Read 85 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 85 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first byte LSB from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second byte MSB from DBSC */
                                   /* program should store the received value */

           /* Compute the time value:
                                   MSB * 2 EXP8 + LSB = IJOGVEL1 /*

           /* Format:             Signed integer */
           /* Range:              -10000..10000 (D8F0..2710 hexadecimal) */
           /* Units:              Percentage of max. RPM / 100 */
           /*                     Example: a value of 150 means 1.5 %.
```

PLC_JOGVEL1= Set PLC jog velocity 1 (op. code 95):

Write the value of jog 1 velocity. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGVEL1=)    /* send value 95 over the serial port */
           _Read()                 /* Read 95 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 95 */
           _Write(MSB_JOGVEL1)     /* Send " the "JOGVEL1" MSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(LSB_JOGVEL1)     /* Send " the "JOGVEL1" LSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:             Signed integer */
           /* Range:              -10000..10000 (D8F0..2710 hexadecimal) */
           /* Units:              Percentage of max. RPM / 100 */
           /*                     Example: for 1.5%, write a value of 150.
```

PLC_JOGVEL2? **Get PLC jog velocity 2(op. code 86):**

Reports the value of jog 2 velocity. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGVEL2?)    /* send value 86 over the serial port */
           _Read()                 /* Read 86 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 86 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first byte LSB from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second byte MSB from DBSC */
                                   /* program should store the received value */

           /* Compute the time value:
                                   MSB * 2 EXP8 + LSB = IJOGVEL2 /*

           /* Format:             Signed integer */
           /* Range:              -10000..10000 (D8F0..2710 hexadecimal) */
           /* Units:              Percentage of max. RPM / 100 */
           /*                     Example: a value of 150 means 1.5 %.
```

PLC_JOGVEL2= **Set PLC jog velocity 2(op. code 96):**

Write the value of jog 2 velocity. Refer to PLC programming specifications (section 4.8).

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_JOGVEL2=)    /* send value 96 over the serial port */
           _Read()                 /* Read 96 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 96 */
           _Write(MSB_JOGVEL2)     /* Send " the "JOGVEL2" MSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(LSB_JOGVEL2)     /* Send " the "JOGVEL2" LSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:             Signed integer */
           /* Range:              -10000..10000 (D8F0..2710 hexadecimal) */
           /* Units:              Percentage of max. RPM / 100 */
           /*                     Example: for 1.5%, write a value of 150.
```

PLC_RECTM? **Set PLC recording time (op. code 83):**

Returns the PLC variable recording time. This value is based on internal gathering sampling time variable.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_RECTM?)      /* send value 83 over the serial port */
           _Read()                 /* Read 83 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 83 */
           _Write(Trig_read)       /* Send value 84 ('T') over the serial port */
           _Read()                 /* Read the first byte LSB from DBSC */
                                   /* program should store the received value */
           _Write()                /* Send back LSB to acknowledge the DBSC */
           _Read()                 /* Read the second byte MSB from DBSC */
                                   /* program should store the received value */

           /* Compute the time value:
                                   MSB * 2 EXP8 + LSB = RECTM /*

           /* Format:             Unsigned integer */
           /* Range:              0..9999 (0..270F hexadecimal) */
           /* Units:              1/100 of milliseconds */
```

PLC_RECTM= **Set PLC recording time(op. code 93):**

Programs the internal gathering sampling time according to the Recording time specified value. DBSC will computes the gathering sampling time that will give the best possible recording resolution for the defined time. Use the "D3S" software to read and display the graphs on the screen.

```
Protocol:  _Write(Header)          /* Send value 88 ('X') over the serial port */
           _Read()                 /* Read 88 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 88 */
           _Write(PLC_RECTM=)      /* send value 96 over the serial port */
           _Read()                 /* Read 96 back, DBSC acknowledgment */
                                   /* program should check if the value received equals 96 */
           _Write(MSB_RECTM)       /* Send " the "RECTM" MSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment .)*/
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(LSB_RECTM)       /* Send " the "RECTM" LSB value to DBSC */
           _Read()                 /* Read back the value (acknowledgment) */
                                   /* program should check if the value received is equal to the value
                                   sent */
           _Write(Trig_operation)  /* Send value 84 ('T') over the serial port */

           /* Format:             Signed integer */
           /* Range:              0..9999 (0..270F hexadecimal) */
           /* Units:              1/100 of milliseconds */
```

PLC_RECVAR? **Get PLC recording variables(op. code 84):**

Returns the present setting of the variables that will be gathered upon the PLC "Start recording" action. Two characters (two byres) are returned. They both hold an index which is decoded as follows:

Master resolver position:	1
Velocity:	2
Current:	3
Actual Current Phase U:	4
Actual Current Phase V:	5
Position following error:	6
Current loop reference:	7
Velocity loop reference:	8
Hand wheel input (position reference):	9
Second analog command (N/A):	10
I2t value (N/A):	11

Protocol:

```
_Write(Header)          /* Send value 88 ('X') over the serial port */
_Read()                 /* Read 88 back, DBSC acknowledgment */
                        /* program should check if the value received equals 88 */
_Write(PLC_RECVAR?)     /* send value 84 over the serial port */
_Read()                 /* Read 84 back, DBSC acknowledgment */
                        /* program should check if the value received equals 84 */
_Write(Trig_read)       /* Send value 84 ('T') over the serial port */
_Read()                 /* Read the first variable number code from DBSC */
                        /* program should store the received value */
_Write()                /* Send back the value to acknowledge the DBSC */
_Read()                 /* Read the second variable number code from DBSC */
                        /* program should store the received value */

/* Format:     Two unsigned characters */
/* Range:      0..11 (0..B hexadecimal) */
/* Units:      None */
```

PLC_RECVAR= **Set PLC recording variables(op. code 94):**

This function sets the variables that will be gathered upon the PLC "Start recording" action. Two characters (two byres) are sent. They both hold an index which is decoded as follows:

Master resolver position:	1
Velocity:	2
Current:	3
Actual Current Phase U:	4
Actual Current Phase V:	5
Position following error:	6
Current loop reference:	7
Velocity loop reference:	8
Hand wheel input (position reference):	9
Second analog command (N/A):	10
I2t value (N/A):	11

```
Protocol: _Write(Header)            /* Send value 88 ('X') over the serial port */
          _Read()                   /* Read 88 back, DBSC acknowledgment */
                                    /* program should check if the value received equals 88 */
          _Write(PLC_RECVAR=)       /* send value 94 over the serial port */
          _Read()                   /* Read 94 back, DBSC acknowledgment */
                                    /* program should check if the value received equals 94 */
          _Write(VAR1)              /* Send the first variable number to the DBSC */
          _Read()                   /* Read the first variable number back from DBSC */
                                    /* program should check if the value is equal to the one sent */
          _Write(VAR2)              /* Send the second variable number to the DBSC */
          _Read()                   /* Read the second variable number back from DBSC */
                                    /* program should check if the value is equal to the one sent */
          _Write(Trig_operation)          /* Send value 84 ('T') over the serial port */

          /* Format:      Two unsigned characters */
          /* Range:       0..11 (0..B hexadecimal) */
          /* Units:       None */
```

PLC_ST? **Get PLC flag status (op. code 81):**

Reads the content of eight consecutive internal system flags starting from a specified flag location. Refer to PLC programming specifications (section 4.8).

```
Protocol: _Write(Header)            /* Send value 88 ('X') over the serial port */
          _Read()                   /* Read 88 back, DBSC acknowledgment */
                                    /* program should check if the value received equals 88 */
          _Write(PLC_ST?)           /* send value 81 over the serial port */
          _Read()                   /* Read 81 back, DBSC acknowledgment */
                                    /* program should check if the value received equals 81 */
          _Write(FLAG_BASE_ADD)     /* Send the flag base address to the DBSC */
          _Read()                   /* Read back the flag base address, DBSC acknowledgment */
                                    /* program should check if the value equals the one sent */
          _Write(Trig_read)         /* Send value 84 ('T') over the serial port */
          _Read()                   /* Read the first vector value from DBSC */
                                    /* program should store the received value */

          /* Format:      unsigned character flag array */
          /* Dimension    discrete flags (bit array containing bit flag status */
          /* Range:       0..255 (0..FF hexadecimal) */
          /* Units:       None */
```

4.8 PLC programming specifications

A PLC is represented by an "*array of PLC commands*". Each command in the array contains a couple of arguments: one "*defined action*" linked with one "*programmable condition*". Several actions can be linked to the same condition. But one action is available only once in the complete array. This permits to trigger several actions with the same condition. Programming a PLC command consist in sending two bytes after the "PLC_CMD=" statement (op. code 90). The first byte contains the "*Action code*", and the second byte the "*Condition code*". Conditions and actions codes are listed below. When programming several PLC commands, each of them requires a sequence with "Header" and "Trigger" logic.

Example: We desire to program the following conditional action:

IF Input 1 = 1, THEN Output 1 = 1, ELSE Output 1 = 0

The command contains the two bytes "0x13 and 0x28". (Output 1: Action code HEX 13, Input 1: condition code HEX 28.). The PLC command will be programmed as follows:

```
_Write(Header)          /* Send value 88 ('X') over the serial port */
_Read()                 /* Read 88 back, DBSC acknowledgment */
                        /* program should check if the value received equals 88 */
_Write(PLC_CMD=)        /* send value 90 over the serial port */
_Read()                 /* Read 90 back, DBSC acknowledgment */
                        /* program should check if the value received equals 90 */
_Write(0x13)            /* Send the action code to the DBSC. */
                        /* 13 hexadecimal (first byte): action "set output 1"
_Read()                 /* Read back the action code, DBSC acknowledgment */
                        /* program should check if the value equals the one sent */
_Write(0x28)            /* Send the condition code to the DBSC */
                        /* 28 hexadecimal (second byte): condition "If input 1 = 1" */
_Read()                 /* Read back the condition code, DBSC acknowledgment */
                        /* program should check if the value equals the one sent */
_Write(Trig_operation)  /* Send value 84 ('T') over the serial port */
```

Condition codes:

Always true flag	DEC 0	HEX 00
Drive over temperature	DEC 20	HEX 14
Motor over temperature	DEC 21	HEX 15
I2t warning	DEC 22	HEX 16
CW warning limit	DEC 24	HEX 18
CCW warning limit	DEC 25	HEX 19
In position	DEC 26	HEX 1A
FE warning	DEC 27	HEX 1B
FE fatal	DEC 28	HEX 1C
Over speed	DEC 29	HEX 1D
Vel following error	DEC 30	HEX 1E
User's input 1 to 4	DEC 40, 41, 42, 43	HEX 28, 29, 2A, 2B
never true flag	DEC 255	HEX FF

Action codes:

PLC Enable	DEC 0	HEX 00
Open Fault relay	DEC 1	HEX 01
Generate Fatal Error	DEC 2	HEX 02
Reset Errors	DEC 3	HEX 03
Set Disable Active mode	DEC 4	HEX 04
Hold position	DEC 6	HEX 06
Set Disable Passive mode	DEC 7	HEX 07
Increase Gear Ratio factor	DEC 11	HEX 0B
Start Jog	DEC 15	HEX 0F
Select Jog 2	DEC 16	HEX 10
User's output 1 to 4	DEC 19, 20, 21, 22	HEX 13, 14, 15, 16

5. Examples

5.1 Example written in BASIC

The following program example demonstrates velocity reading from the DBSC and how to set jog velocity and mode.

```
REM    This Basic program demonstrates velocity reading from the DBSC and velocity writing in jog mode
REM    to the DBSC.

REM  COMMUNICATION INITIALIZATION
REM  ============================

REM  Open the RS-232 communication protocol without the hand shake lines:

OPEN "COM1:9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB2048,RB2048" FOR RANDOM AS #1

REM  INITIALIZATION
REM  ============

FLAG = 0        'ready for keyboard input
i = 0           'set communication counter to 0
a = 0           'temporary storage register
B = 0           'second temporary storage register
Byteok = 1      'set acknowledgment "Byteok" to 1 (byte received by DBSC is OK)

  WHILE 1                       'Loop for ever
    IF FLAG = 0 THEN            'if ready for keyboard input
      INPUT DD$                 'prompt for user input

'     USER REQUEST VELOCITY from DBSC
'     ===============================

        IF DD$ = "v" THEN       'if user input v (here case sensitive!) is received from key board
                                '(DD$ says that input is a string type).
         FLAG = 1               'then discard any keyboard command input.

              'Enable event trapping on port 1:

         ON COM(1) GOSUB ComHandler   'Interrupt for communication:
                                      'if anything received on com1 go to sub program labeled
                                      'ComHandler; this, from anywhere in the program.

         COM(1) ON              'Enable com1 event trapping

         PRINT #1, "X";         'send X to com1 (88 dec.) protocol header for DBSC in order to respond.
        END IF
```

```
'    SEND JOG COMMAND TO DBSC
'    ========================

        IF DD$ = "j" THEN    'If j (here case sensitive!) is received from key board (DD$ says that
                             'input is a string type).
         FLAG = 1            'then discard any keyboard command input but accept argument for the
                             'j command

         'display "text" on screen and input argument "Vel" for j command:

         INPUT " Enter required speed in % time 100> ", Vel

         'The following protection is also implemented in the DBSC. So it is not really necessary to
         'have it here.
         IF Vel > 10000 THEN Vel = 10000                    'saturate % value to max 10000
         IF Vel < -10000 THEN Vel = -10000                  'saturate % value to max 10000

         Absvel = ABS (Vel)                                 'Calculate the absolute value of the velocity
         Veldmsb = Absvel \ 256                             'Calculate the MSB (integer division)
         Veldlsb = Absvel - (Velmsb * 256)                  'Calculate the LSB

         IF Vel < 0 THEN Velmsb = 255 - Velmsb              'Calculate the MSB of the negative velocity
         IF Vel < 0 THEN Vellsb = 256 - Vellsb              'Calculate the MSB of the negative velocity

         ON COM(1) GOSUB JogComHandler    'if anything received on com1, go to sub program
                                          'labeled JogComHandler.

         COM(1) ON           'Enable com1 event trapping
         PRINT #1, "X";      'Send X to com1 (88 dec.) protocol header for DBSC in order to respond.
        END IF

'    SEND JOG STOP COMMAND TO DBSC
'    =============================

        IF DD$ = "js" THEN   'If js (here case sensitive!) is received from key board (DD$ says that
                             'input is a string type).

         FLAG = 1            'then discard any keyboard command input
         ON COM(1) GOSUB JsComHandler        'if anything received on com 1 go to sub program
                                             'labeled JsComHandler.

         COM(1) ON           'Enable com1 event trapping
         PRINT #1, "X";      'Send X to com1 (88 dec.) protocol header for DBSC in order to respond.
        END IF

        IF DD$ = "s" THEN STOP    'If s (here case sensitive!) is received from key board (DD$ says
                                  'that input is a string type), then stop running the program.
    END IF

    WEND              'End of while
    COM(1) OFF        'Disable com1 event trapping
    END
```

```
'    COMMUNICATION SUBROUTINES
'    =========================

ComHandler:
    a = INP(&H3F8)    'set "a" equal to the value read back on com 1 port address (ex: 88 dec. for X)
    PRINT a           'print "a" value to the screen
    i = i + 1         'increment the communication counter
    IF i = 1 THEN OUT &H3F8, 2      'If i = 1, send a value of 2 to DBSC which command it to send
                                    'back the velocity value.
    IF i = 2 THEN OUT &H3F8, 84     'if i = 2, send 84 value (character "T" for Trigger) to DBSC, activate
                                    'the sending of the information requested (here LSB).
    IF i = 3 THEN                   'if i = 3, send "a" value to DBSC to
        OUT &H3F8, a                'acknowledge the DBSC
        B = a                       'set "B" equal LSB
    END IF
    IF i = 4 THEN             'if i = 4, send "a" value to acknowledge DBSC
        OUT &H3F8, a
        COM(1) OFF            'Discard any character received on COM 1.
        x = a * 256 + B       'Calculate the velocity value from the two bytes received: shift 8 times
                              'the MSB to the left and add LSB => 16 bits integer (velocity in resolver
                              'bits per servo cycle).
        IF x > 32767 THEN x = -(65536 - x)  'If value is negative, calculate the two's complement,
                                            '(velocity in resolver bits per servo cycle)
        y = x / 65536 / .0004704# * 60      'calculate the velocity in RPM
        PRINT x; " bits per ts", y; " rpm"  'Display velocity values x and y
        i = 0                               'Reset communication counter.
        FLAG = 0                            'Ready for keyboard input
    END IF
    RETURN

JogComHandler:
    a = INP(&H3F8)    'set a equal to the value read back on  'com 1 port address (ex: 88 dec. for X)
    PRINT a           'print "a" value to the screen
    i = i + 1         'increment the communication counter
    IF i = 1 THEN OUT &H3F8, 22     'If i = 1, send a value of 22 to DBSC which command the
                                    'DBSC to set the jog velocity, mode and run it.
    IF i = 2 THEN OUT &H3F8, Velmsb             'If i = 2 send the MSB of velocity value to DBSC
    IF i = 3 THEN
      IF a = Velmsb THEN Byteok = 1     'acknowledgment byte received by DBSC is OK)
      END IF
    IF i = 4 THEN OUT &H3F8, Vellsb     'If i = 4 send the LSB of velocity value to DBSC
    IF i = 5 THEN
      IF a = Vellsb THEN Byteok = 1     'acknowledgment byte received by DBSC is OK)
      END IF
      IF Byteok = 1 THEN OUT &H3F8, 84  'send 84 value (character "T" for Trigger) to DBSC, activate
                                        'the jog function.
      ELSE
        PRINT " communication error, bad Ack"
        i = 0                 'Reset communication counter.
        COM(1) OFF            'Discard any character received on COM 1.
        FLAG = 0              'Ready for keyboard input
      END IF
    END IF
    IF i = 6 THEN
        i = 0                 'Reset communication counter.
        COM(1) OFF            'Discard any character received on COM 1.
        FLAG = 0              'Ready for keyboard input

    END IF
    RETURN
```

```
JsComHandler:
    a = INP(&H3F8)    'set a equal to the value read back on com 1 port address (ex: 88 dec. for X)
    PRINT a                          'print "a" value to the screen
    i = i + 1                        'increment the communication counter
    IF i = 1 THEN OUT &H3F8, 62      'If i = 1, send a value of 62 to DBSC which command the DBSC to
                                     'be enabled. The "enable" also deactivates the jog mode.
    IF i = 2 THEN OUT &H3F8, 84      'send 84 value (character "T" for Trigger) to DBSC, activate the
                                     'enable function, disable the jog function.
    IF i = 3 THEN
        i = 0                'Reset communication counter.
        COM(1) OFF           'Discard any character received on COM 1.
        FLAG = 0             'Ready for keyboard input
    END IF
    RETURN
```

5.2 Example written in "C"

The following program example demonstrates velocity reading from the DBSC and how to set jog velocity and mode.

```
#include <stdio.h>   /* File used for screen and keyboard interface */
#include <conio.h>   /* as above */

#include "com_def.h"  /* Macro-definitions file for PC communication routines */
#include "glb_cnst.h"  /* Macro-definitions file for constants definitions */

/* Host Communication Protocol to set the jog mode and velocity (Jog)*/
#define         JOG_SET&EXE                                     22

/* Host Communication Protocol to set HOLD POSITION */
#define         CMD_HOLD                                        61

int     Jog(int vel);       /* declare Jog function with "vel" argument as a integer type, the function will
                return an integer. */

int     Port_Base;      /* the address of the UART (RS232): 0x2f8, 0x3f8   */

/*
*               INITIALIZATION of RS232 registers:
*
*               Example: InitComX(Port_Base,9600,0,1,8)
*
*               port_base:      0x3f8 for COM1, 0x2f8 for COM2
*               baudrate :      9600
*               parity:         0 --> no parity
*               stop:           1 --> one stop bit
*               databits:       8 --> 8 data bits
*/

void InitComX(int port_base, int baudrate, int parity, int stop, int databits)
{
        unsigned        char    parmbyte;
        unsigned        int     divisor;
        unsigned                char    msb,lsb;
        int                                     sine_val ;
/*
*               Initialize the BAUD RATE to 9600
*/

        divisor = (int)(115200.f / (float)baudrate);
        msb =(unsigned char)(divisor >>8);
        lsb =(unsigned char)((divisor     << 8 ) >> 8);

        outp(port_base + 3,128);
        outp(port_base,lsb);
        outp(port_base + 1,msb);
        parmbyte = (unsigned char)(databits - 5);
        parmbyte |=(unsigned char)( (stop-1) <<2 ) ;
        if ( parity ) parmbyte |=8;
        if ( parity==2 ) parmbyte |=16;
        outp(port_base + 3, parmbyte);
/*
        inp(port_base+5);
*/
}
```

```
/*      ********** DBSC READ SUBROUTINE **********
*
*   READ a character from the DBSC.
*
*               Example: my_char = _Read();
*
*               void    - no function parameters are expected.
*               char    - this function returns a character.
*/
char _Read(void)                /* declare routine label as _Read */
{                               /* start of the routine */
        int             linestatus;
        unsigned int    i=0;

        do
        {
                linestatus = inp(Port_Base+5);
                if ( linestatus & _OVERRUN )    /* overrun error */
                        goto EXIT_F;
                if ( linestatus & _PARITY )     /* parity error  */
                        goto EXIT_F;
                if ( linestatus & _FRAME )      /* frame error   */
                        goto EXIT_F;
                if ( linestatus & _BREAK )      /* break */
                        goto EXIT_F;
                if (i++ > 0x7ff0)               /* Counter: if i > 7ff0 HEX, exit Time out if DBSC does
                                                   not send back value in time */
                        goto EXIT_F ;
        }
        while ( !(linestatus & _DATAREADY) );
/*
*               Return a character from RS232 port:
*/
        return((char)inp(Port_Base)) ;          /* take the received character from DBSC, returns the
                                                   value to the calling routine and end this sub-routine. */
EXIT_F:                                         /* communication error */
        return((char)COM_ERR);
}                                               /* end of routine _DBSC_Read */
```

```
/*      ********** DBSC WRITE SUBROUTINE **********
*
*       WRITE a character to the DBSC.
*
*       Example: _Write('A'); --> write the 'A' character to the RS232
*/

char _Write(char c)
{
        int                             linestatus;
        unsigned int    i=0;

        do
        {
                linestatus = inp(Port_Base+5);

                if ( linestatus & _OVERRUN )    /* overrun error */
                        goto EXIT_F;
                if ( linestatus & _PARITY )     /* parity error  */
                        goto EXIT_F;
                if ( linestatus & _FRAME )      /* frame error   */
                        goto EXIT_F;
                if ( linestatus & _BREAK )      /* break */
                        goto EXIT_F;
                if (i++ > 0x7ff0)
                        goto EXIT_F ;
        }
        while ( !(linestatus & _TRANSMITREADY) );
/*
*               send character to RS232 port:
*/

        return((char)outp(Port_Base,(int)c)) ;

EXIT_F:
        return((char)COM_ERR);
}
```

```
/*      ********** JOG FUNCTION SUBROUTINE **********
*/

int Jog(void)
{
        char            ack;
        int             vel = 0;        /* "vel" variable is integer type and is initialized to 0 */
        unsigned int    abs_vel
        char            vel_msb;
        char            vel_lsb;

        printf("\nJOG Function\n");      /* print a title on screen */

        while (1 == 1)                  /* loop forever */
        {
                printf("\nEnter velocity in percentage time 100 (type 12345 to stop JOG) [-10000,10000]:");

                scanf("%d",&vel);       /* read and wait for a character from the keyboard */

                if ( vel == 12345 )     /* exit if vel is equal to 12345 */
                        break;          /* go out of the while loop */

                /* protection to make sure that: -10000 <= vel <= 10000 */

                if (vel >  10000) vel =  10000;
                if (vel < -10000) vel = -10000;

                /* calculate the MSB and LSB of the velocity to be sent to the DBSC */

                abs_vel = ABS (vel);                    /* calculate the absolute value of velocity */
                vel_msb = INT (abs_vel / 256);          /* positive velocity MSB */
                vel_lsb = abs_vel - (vel_msb * 256);    /* positive velocity LSB */

                if (vel < 0) vel_msb = 255 - vel_msb    /* negative velocity MSB */
                if (vel < 0) vel_lsb = 256 - vel_lsb    /* negative velocity LSB */

                /* JOG protocol */

/*
*   1. Send 'X'
*   2. Read the ack.
*   3. compare the ack. to 'X'
*/

                _Write('X');
                ack = _Read();
                if (ack != 'X')
                        return(0);              /* Error termination */

/*
*   1. Send JOG_SET&EXE (equal to 22)
*   2. Read the ack.
*   3. compare the ack. to JOG_SET&EXE
*/

                _Write(JOG_SET&EXE);
                ack = _Read();
                if (ack != JOG_SET&EXE)
                        return(0);              /* Error termination */
```

```
/*
*   1. Send 'vel_msb'
*   2. Read the ack.
*   3. compare the ack. to 'vel_msb'
*/

                _Write(vel_msb);
                ack = _Read();
                if (ack != vel_msb)
                        return(0);              /* Error termination */
/*
*   1. Send 'vel_lsb'
*   2. Read the ack.
*   3. compare the ack. to 'vel_lsb'
*/

                _Write(vel_lsb);
                ack = _Read();
                if (ack != vel_lsb)
                        return(0);              /* Error termination */
/*
*   1. Send a trigger 'T'
*   2. Read the ack.
*   3. compare the ack. to 'T'
*/
                _Write('T');
                ack = _Read();
                if (ack != 'T')
                        return(0);              /* Error termination */
        }

/*              Send a ZERO (0) velocity command to the DBSC before exit the Jog procedure.
*/

/*
*   1. Send 'X'
*   2. Read the ack.
*   3. compare the ack. to 'X'
*/

        _Write('X');
        ack = _Read();
        if (ack != 'X')
                return(0); /*           Error termination */
/*
*   1. Send JOG_SET&EXE (equal to 22)
*   2. Read the ack.
*   3. compare the ack. to JOG_SET&EXE
*/

        _Write(JOG_SET&EXE);
        ack = _Read();
        if (ack != JOG_SET&EXE)
                return(0);              /* Error termination */
```

```
/*
*   1. Send ZERO (0) velocity (MSB)
*   2. Read the ack.
*   3. compare the ack. to ZERO (0)
*/

        _Write(0);
        ack = _Read();
        if (ack != 0)
                return(0);              /* Error termination */

/*
*   1. Send ZERO (0) velocity (LSB)
*   2. Read the ack.
*   3. compare the ack. to ZERO (0)
*/

        _Write(0);
        ack = _Read();
        if (ack != 0)
                return(0);              /* Error termination */

/*
*   1. Send a trigger 'T'
*   2. Read the ack.
*   3. compare the ack. to 'T'
*/

        _Write('T');
        ack = _Read();
        if (ack != 'T')
                return(0);              /* Error termination */

        return(1);                      /* normal termination */
}
/*      ********** HOLD SUBROUTINE **********/
int Cmd_Hold(void)
{
        char  ack;

/*
*   1. Send 'X'
*   2. Read the ack.
*   3. compare the ack. to 'X'
*/

        _Write('X');
        ack = _Read();
        if (ack != 'X')
                return(0);              /* Error termination */

/*
*   1. Send CMD_HOLD (equal to 61)
*   2. Read the ack.
*   3. compare the ack. to CMD_HOLD
*/

        _Write(CMD_HOLD);
        ack = _Read();
        if (ack != CMD_HOLD)
                return(0);              /* Error termination */
```

```
/*
*   1. Send a trigger 'T'
*   2. Read the ack.
*   3. compare the ack. to 'T'
*/

        _Write('T');
        ack = _Read();
        if (ack != 'T')
                return(0);              /* Error termination */

        return(1);                      /* normal termination */
}
/*      ********** MAIN PROGRAM **********
*/
void main(void)
{
/*
*   Set Port_Base value to 0x3f8 (COM 1):
*/

        Port_Base = 0x3f8;

/*
*   Initialize RS232 port registers:
*/

        InitComX(Port_Base,9600,0,1,8);

/*
*   Jog:
*/

        if (Jog() == 0)                 /* activate jog function and check if returned argument is 0 */
                printf("\n ***: COMMUNICATION ERROR");
        else
        {
                if (Cmd_Hold() == 0)
                        printf("\n ***: COMMUNICATION ERROR");
        }
}
```

Baldor Europe:

Germany:

Baldor ASR GmbH
Dieselstraße 22
D-85551 Kirchheim
Phone (089)905080
Fax (089)90508491

Switzerland:

BALDOR ASR AG
Schützenstraße 59
P.O. Box 73
CH - 8245 Feuerthalen
Phone (052)647 47 00
Fax (052)659 23 94

Great Britain

BALDOR ASR UK LIMITED
Unit 13
Charlwoods Road
GB - East Grinstead
Sussex RH19 2JB
Phone (0342)31 59 77
Fax (0342)32 89 30

France

BALDOR AUTOMATION
2, Rue du Vallon
F - 94440 Marolles-en-Brie
Phone (01)45 69 23 32
Fax (01)45 99 08 64

Italy:

BALDOR Italia S.R.L.
Via Giolitti 8
I - 10123 Torino
Phone (011) 56 24 440
Fax (011) 56 25 660

Distributors Europe:

Belgium:

ABB
Asea Brown Boveri
Hoge Wei 27
B - 1930 Zaventem
Phone (02) 7 18 63 11
Fax (02) 7 18 66 66

Netherlands:

VECTOR
Aandrijftechniek B.V.
P.O. Box 1 00 85
NL - 3004 AB Rotterdam
Phone (010) 4 46 37 00
Fax (010) 4 15 55 52

Sweden:

G & L Beijer Electronics AB
P.O. Box 325
S - 20123 Malmö
Phone (040) 35 86 00
Fax (040) 93 23 01

Finland:

G & L Beijers OY
PL 13
SF - 00621 Helsinki
Phone (080) 79 99 11
Fax (080) 79 61 26

Austria:

Lenze
Antriebstechnik GesmbH
Mühlenstraße 3
A - 4470 Enns
Phone (072 23) 34 21
Fax (072 23) 32 80

Switzerland:

Lenze Bachofen AG
Ackerstraße 42
CH - 8610 Uster
Phone (01) 9 44 12 12
Fax (01) 9 44 12 33

France:

SOCIETE COPRE
5, rue du Girou
Cidex 1042
F - 31240 L'Union
Phone (061) 74 68 04
Fax (061) 09 21 06

Norway:

G & L Beijer Electronics AS
P.O. Box 487
N - 3002 Drammen
Phone (03) 89 42 70
Fax (03) 89 51 01

Denmark:

Dyrbaek Teknologie A/S
Blaakrogvej 2
Varnaes
DK - 6200 Aabenraa
Phone (074) 68 08 10
Fax (074) 68 08 49

United States:

BALDOR ELECTRIC COMPANY
P.O. Box 2400
Fort Smith, AR 72902
Phone (501) 6 46 47 11
Fax (501) 6 48 57 92

Australia:

AUSTRALIAN
BALDOR PTY. LTD.
P.O. Box 33
Kenthurst NSW 2156
Sydney
Australia
Phone (02) 6 74 54 55
Fax (02) 6 74 24 95

Far East:

Republic of Singapore:

BALDOR ELECTRIC PTE. LTD.
51 Kaki Bukit Road 2 No.01-15
KB Warehouse Complex
Singapore 1441
Republic of Singapore
Phone (7) 44 25 72
Fax (7) 47 17 08