The Hyperdrive10 User Reference Manual

This manual describes the capability, functionality, operation and programming of the versatile high-powered
Hyperdrive10 Stepper Motor Controller

Kremford Pty Ltd
Melbourne, Australia

Revision 1.0

Revisions

1.0 1-5-2021

Note: There have been several software revisions since this manual was published. Please contact Kremford if you have questions based on details in this manual.

Table of Contents

Table of Tables

Table of Figures

1 Introduction

1.1 About This Manual

Thank you for purchasing your Hyperdrive10 Stepper Motor Controller.

This manual describes the Hyperdrive10 controller in considerable detail. It includes chapters on its maximum and working ratings and how it can be connected to the machine world. Its functionality is described in detail along with where and how those functions would be used. The Hyperdrive10 is configured and monitored by standard Modbus commands. The command set is described along with how they can be used, including how the Hyperdrive10 interprets the various Modbus packet sections. Several HMI programming examples using the freely available Maple Systems EZwarePlus Windows software demonstrate the message types to use. Similar software is available from all standard manufacturers of HMIs and the same protocols will apply for any Modbus control hardware.

Kremford Pty Ltd would appreciate your comments and suggestions about this manual. As a user of our documentation, you are in a unique position to provide ideas that can have a direct impact on future releases of this manual. If you have comments or suggestions, please post your request on the Kremford web site Kremford.com.au.

2 Kremford (Vic) Pty Ltd Conditions of Sale

2.1 Definitions

Goods means products supplied by Kremford.

GST refers to Australian Goods and Services Tax and is applicable to goods sold within Australia only.

Terms means these Standard Conditions of Sale.

Kremford means Kremford (Vic) Pty Ltd as a trading entity, 1st Floor 175 Collins St. Melbourne, Victoria, 3000 Australia.

2.2 Terms Overview

1. These terms and Conditions constitute the entire agreement between the parties for the goods. Any prior arrangement, representations or undertakings are hereby suspended.
2. Payment terms as defined below.
3. If the CUSTOMER breaches agreed Payment terms, the CUSTOMER must pay any collection, commission and/or legal fees charged by any third party to recover money due.
4. If the CUSTOMER makes any default in payment or commits any act of bankruptcy or, being an incorporated company, passes a resolution for winding up (except for the purposes of reconstruction) or an application is presented for its winding up, Kremford may, without prejudice to its other right, either suspend further deliveries, require payment in advance for all such deliveries or terminate any contract forthwith by written notice.

2.3 Ownership of Delivered Goods

1. Title to all goods and services sold shall only pass to the CUSTOMER free of encumbrances upon payment in full by the CUSTOMER of all monies due by the CUSTOMER on any account.

2.4 Liability

1. Any liability of Kremford to the Customer including, but not limited to, the liability for special, consequential or incidental damages or for breach of any term, condition, warranty, undertaking, inducement or representation whether express, implied, statutory or otherwise relating to these terms or to the goods shall be limited at the portion of Kremford, to any one or more of the following:
    a. if the breach related to the goods:-
        i. the repair of the goods
        ii. the payment of the cost of having the goods repaired or
    b. if the breach relates to services:-
        i. the supplying of the services again; or

ii. the payment of the cost of having the services supplied again

2. Whilst every care is taken in preparing all written communications, Kremford shall not be liable to any Customer relying on any fact, matter or representation. Customer should satisfy itself as to the suitability and fitness of any product before order.
3. Whilst every care is taken in compiling all information and technical data Kremford will not accept responsibility for subsequent loss or damage arising from its use. Nothing in these terms shall exclude or modify any conditional warranty implied by law where to do so would render such terms void.

2.5 Training and Service

1. It is the customer's responsibility to train operating staff.
2. It is the customer's responsibility to maintain the goods in accordance with the specifications.
3. Training and commissioning is not inclusive in the quoted price unless stated.

2.6 Insurance

1. Kremford follows standard accepted business practice for insurance of product in transit, and is responsible for insurance of product whilst in transit. The Customer accepts all responsibility for goods once they have been delivered.

2.7 Warranty

1. All product sold by Kremford is covered by the manufacturer's warranty for a period of twelve (12) months against defects in workmanship on a return to factory of manufacture basis. The warranty period begins when the goods arrive at the customer's facility.
2. The warranty does not cover goods damaged as a result of improper handling, misuse or neglect.
3. The warranty is conditional on the maintenance, service procedures and operating procedures having been correctly observed.
4. Kremford does not accept any responsibility for any loss of production or any costs associated with loss of production, due to goods being out of operation for any reason.
5. Due to the nature of precision engineering parts, Kremford do not warrant product for any damage caused to or by said product once the product has been installed and operated.
6. No warranty will be extended to machines where the goods are installed.

2.8 Miscellaneous

1. Any contract for the sale shall be deemed to be made in the state of Victoria and the Law, courts or arbitration facilities of the State shall be used to construe or apply the contract or to determine or resolve any dispute or action arising under it.
2. Kremford reserves the right to vary these terms immediately upon giving notice to the CUSTOMER of such variation.

3. The invalidity or unenforceability of any of these terms shall not affect the validity and enforceability of the remainder of these terms.
4. Unless otherwise agreed to in writing by Kremford, the terms appearing herein shall be incorporated into all agreements entered into with Kremford to supply the Customer with goods and shall be incorporated into the terms appearing in any document provided by or on behalf of the Customer. In any situation where the terms of the Customer conflict with the terms of Kremford, the terms of Kremford shall take precedence.
5. Except as stated herein, Kremford makes no other warranty, expressed or implied, and in no event shall be liable for incidental or consequential damages. Kremford makes no warranty as to fitness of goods for particular purpose. Kremford neither assumes nor authorizes anyone else to assume for it, any other obligation or liability relating to its goods. This warranty does not apply to damage to goods which, in the judgment of Kremford, has been subject to incorrect voltage supply, normal wear and tear, to misuse, neglect, or has been repaired or altered by unauthorized personnel. Defective parts must be returned to Kremford, freight prepaid, within 14 days of shipment of the replacement part, except for components valued at under $300 list price under the conditions stated above. Defective parts must be returned in their original state along with a fully completed Kremford warranty parts return form. Defective parts that have been disassembled, damaged during removal, or otherwise tampered with, will not be covered under warranty, unless otherwise stated in writing. Kremford's sole obligation under this warranty will be to provide repairs to components or replacement parts, f.o.b. Kremford's point of shipment except as stated above. All aspects of the above stated warranty and procedures related to ordering parts under warranty will be upheld with no exceptions.

3 Safety

3.1 Qualification of personnel

Only technicians who are familiar with and understand the contents of this manual and the other relevant documentation are authorized to work on and with this controller system. The technicians must be able to detect potential dangers that may be caused by setting parameters, changing parameter values and generally by the operation of mechanical, electrical, and electronic equipment. The technicians must have sufficient technical training, knowledge, and experience to recognise and avoid dangers. The technicians must be familiar with the relevant standards, regulations and safety regulations that must be observed when working with the controller and its drive system.

Only a qualified electrician should connect mains power to any power supplies used by the Hyperdrive10.

3.2 Intended Use

The Hyperdrive10 control system described here is a product for general use that conforms to the state of the art in technology and is designed to prevent any dangers. However, controllers that are not specifically designed for safety functions are not approved for applications where the functioning of the drive could endanger persons. The possibility of unexpected or unbraked movements can never be totally excluded without additional safety equipment. For this reason, personnel must never be in the danger zone of the controller's drive unless additional suitable safety equipment prevents any personal danger. This applies to operation of the machine during production to all service and maintenance work on the controller's drives and the machine. The machine design must ensure personal safety. Suitable measures for prevention of property damage are also required. In all cases the applicable safety regulations and the specified operating conditions, such as environmental conditions and specified technical data, must be observed.

3.3 Hazard Categories

The controller and drive system must not be commissioned and operated until completion of installation in accordance with the relevant government regulations and the specifications in this manual. To prevent personal injury and damage to property, damaged drive systems must not be installed or operated.

Changes and modifications to the Hyperdrive10 system are not permitted and if made no warranty or liability will be accepted. The controller system must be operated only with the specified wiring and approved accessories. The drive systems must not be operated in an environment subject to explosion hazard.

3.4 General Safety Instructions

UNINTENDED CONSEQUENCES OF EQUIPMENT OPERATION

When the system is started, the drive mechanism the Hyperdrive10 is controlling is usually out of the operator's view and cannot be visually monitored.

• Only start the system if there are no persons in the hazardous area.

Failure to follow these instructions can result in death or serious injury.

EXPOSED SIGNALS

Hazardous voltage levels may be present if using an open frame power supply to power the product.

Failure to follow these instructions can result in death or serious injury.

LOSS OF CONTROL

• The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop, overtravel stop, power outage and restart.

• Separate or redundant control paths must be provided for critical functions.

• System control paths may include communication links. Consideration must be given to the implication of unanticipated transmission delays or failures of the link.

• Observe all accident prevention regulations and local safety guidelines.

• Each implementation of the Hyperdrive10 must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death or serious injury.

4 Hyperdrive10 Functional Summary

The Hyperdrive10 provides a large number of operating modes and connect capabilities that should satisfy most machine control requirements. Motor and operational parameters can be set and/or changed via Modbus commands. Most setup type parameters are saved to memory and thus will survive power cycling. The controller responds directly to Modbus packets over duplex RS485 and to simple encapsulated Modbus packets over USB. The controllers are switch addressable from 0 to 7.

The Hyperdrive10,

- Is designed to drive bipolar stepping motors. It can be easily connected for unipolar and 8-wire motors.
- Can use supply voltages of up to 64 volts and drive up to 10 amperes peak.
- Can operate by itself as a stand-alone externally stepped controller.
- Can operate by itself as a single motor drive on a Modbus buss under HMI control.
- Can operate in a master-slave set of Hyperdrive10 controllers with many machine-wide parameters capable of being shared between controllers. The set can consist of up to seven controllers on the same RS485 buss.
- RS485 buss speeds can be up to 115,200 baud.
- A Hyperdrive10 can be controlled over USB using an encapsulated Modbus packet. USB and RS485 packets can be interleaved.
- Motor speed is specified in steps/second. It can also be specified as any linear derived equivalent such as RPM or Metres/minute.
- A gearing can be specified that describes the complete drive train from the motor to some form of output, either rotary or linear.
- Accelerations are specified in steps/second^2.
- The motor profile can separately specify both acceleration and deceleration rates. Rates can by synchronized between controllers.
- Power can be individually set for stopped, positive acceleration, running, and negative acceleration.
- Includes a temperature sensor in degrees C and visible to Modbus.
- The controllers are switch addressable with the so-called **master** always at address 1. In a machine with multiple Hyperdrive10/motor combinations, the Hyperdrive10 at address 1 typically drives the highest inertia load which usually decides the maximum machine acceleration rates.
- In master-slave mode:
    - The master can provide a master speed. Slaves can choose to use this speed, an arithmetic derivative of it (gearing), or a speed of their own.

- The master keeps the up and down accel values which slaves can optionally use to ensure all motors take the same accel or decel time, regardless of their individual speeds.
- A master override stop signal can be selected to stop all slaves.
- A master run signal can be selected to start the motor or else arm the motor to start based on some other input signal.
- A global maximum output or machine speed can be set. Gearing can mean the speed of individual motors can exceed this.

- Has four isolated inputs and two isolated switching outputs.
  - Input 1 can be designated as the signal to start the motor.
  - Input 2 can be designated as the signal to stop the motor.
  - Input 3 can be configured to control output 1 or to set the motor direction in external step mode.
  - Input 4 can be configured to control output 2 or as the step pulse input in external step mode.
  - Inputs 1 and 2 can specify action on a high-going or a low-going input signal.
  - Inputs 1 and 2 can have a debounce time-delay attached.
- Independent of Modbus commands, the controller can be configured so:
  - Input 1 will directly start the motor or start a timer that will subsequently start the motor after the time-out.
  - The motor can be set to run for a given number of milliseconds, or for a given number of steps.
  - Input 2 can be configured to stop the motor or start a delay that will stop the motor on timeout.
  - An additional timer can be used to block spurious signals to Input 2.
- The motor's inherent steps-per-revolution can be specified.
- The microsteps available are 1, 2, 4, 8 and 16.

5 Hyperdrive10 Ratings

5.1 Absolute Maximum Ratings

These absolute maximum ratings define conditions that should never be exceeded, even momentarily. Stresses above those listed here may cause permanent damage to the device. Functional operation of the device above those indicated is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

PARAMETER	SYMBOL	Min	Typ	Max	UNIT
Motor supply voltage	V_{MOTOR}	18	48	72	VDC
Motor current	I_{PEAK}			10[1]	A
Maximum speed	S/s			59590	Steps/sec
Maximum acceleration	S/s^2			59590	$Steps/sec^2$
Switch supply voltage	VO_{MAX}			60	VDC
Switch current	IO_{MAX}			500	mA
Operating temperature	T_{OP}	-10		50[2]	°C
RS485 speed	B	9600	19200	115200[3]	Baud
Label position jitter	PJ		0.4	0.8	mm

Notes:

1. The Hyperdrive10 can require high surge currents from this supply during motor accelerations and high-powered motors can produce high surge currents during decelerations that the power supply must absorb.
2. This ambient temperature limit can be exceeded if the internal Hyperdrive10 temperature is monitored to be less than 70°C.
3. Depends on the RS485 connecting cable length and the physical last Hyperdrive10 on the buss having its LNK2 link in place.

5.2 Input Specifications

INPUTS	SYMBOL	MINUMUM	MAXIMUM	UNIT
Input voltage range	V_{IN}	5	24	V_{DC}
Switching current @ 24V	I_{IN}^{MAX}		6	mA
Switching current @ 5V	I_{IN}^{MIN}		4	mA

These input specifications apply to all four inputs IN-1, IN-2, IN-3 and IN-4.

5.3 Replacement Parts

1. Fuse(M) – Littlefuse 5x20mm 6A – 0234006 (consider 10A for high-current motors)
2. Shorting link – RS Pro – 2518682

6 The Hyperdrive10 Hardware Configuration

Always be careful of static electricity when working with electronic components like the Hyperdrive10. Ensure all connections and your body are initially at the same potential as the negative power pin 2 of connector P5, the power/motor connector.

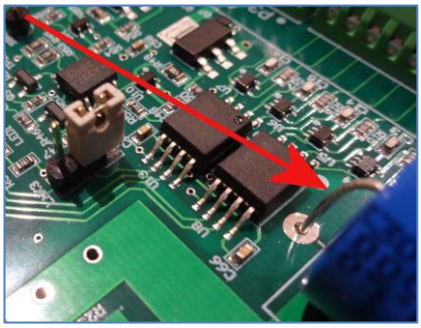

Figure 1. Touching this wire is a safe way to ensure your body and the Hyperdrive10 electronics are at the same potential.

The negative connections of capacitors C21 or C22 (see Figure 1) are directly connected to pin 2 of P5. When working with the cover off and before touching any other component, you should always momentarily touch these wires to equalise your body potential.

6.1 PCB Links and Switches

The Hyperdrive10 includes several links and switches mounted on the PCB and used for a number of configuration settings. These are typically set once on installation and are rarely changed in normal use.

SWITCH/LINK NAME	FUNCTION
LNK1	Short this link to have the Hyperdrive10 power up configured as a USB disk for downloading software updates.
LNK2	Terminates the RS485 RX line with a 120 ohm resistor. Short this link in the physically last Hyperdrive10 on the RS485 buss.
LNK3	This is a two-position link. Normal position connects Input-4 to the processor. Use the alternate position when the Hyperdrive10 is configured for external clocking.
SW1	This 8-position switch provides for several Hyperdrive10 configuration settings.

Table 1. The PCB Links and Switches.

The LNK3 is used when the Hyperdrive is configured to use an external step frequency. Figure 2 shows the two settings for reference. On the left the link is in its normal position. On the right it is configured to direct the external step clock to the controller.

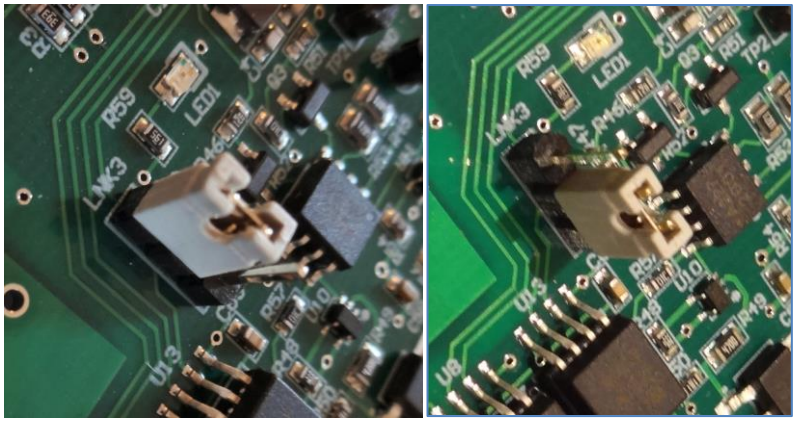

Figure 2. The two LNK3 settings.

The 8-position switch SW1 is used to set the Hyperdrive10 Modbus address as well as some other configuration settings.

SWITCH NUMBER	FUNCTION
1	Setting this switch ON will make the Hyperdrive10 start up in External Clocking mode.
2	RS485 Baud rate (MSB)
3	RS485 Baud rate (LSB)
4	Will load factory settings if on during power-up. Do not leave on.
5	NC
6	Modbus address (MSB)
7	Modbus address
8	Modbus address (LSB)

Table 2. Settings for the 8-position Switch SW1.

The address of the Hyperdrive10 on a Modbus serial line is set by switches 6, 7 and 8 in SW1.

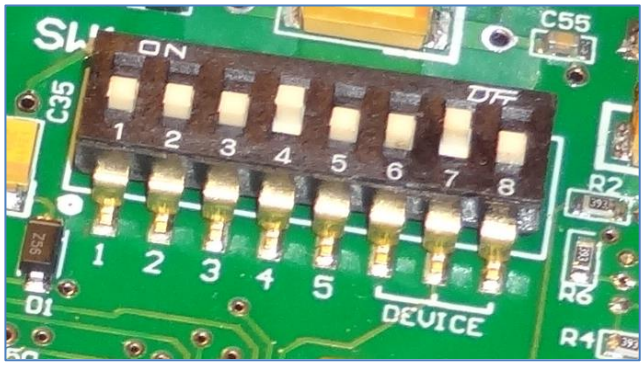

Figure 3. The Hyperdrive10 configuration switch. It is set here for a Baud rate of 9600, a device address of 2, and with switch 4 on to automatically load the factory defaults on switch on.

6.1.1 Modbus Address

A photo of the switch is shown in Figure 3 (not necessarily the actual switch on your Hyperdrive10), and the switch Modbus address settings are shown in Table 3.

SWITCH 6	SWITCH 7	SWITCH 8	ADDRESS
OFF	OFF	ON	1
OFF	ON	OFF	2
OFF	ON	ON	3
ON	OFF	OFF	4
ON	OFF	ON	5
ON	ON	OFF	6
ON	ON	ON	7

Table 3. Setting the Hyperdrive10 Modbus Address.

6.1.2 RS485 Baud Rate

Switches 2 and 3 set the RS485 Baud rate. The switch settings and corresponding rates are shown in Table 4.

SWITCH 2	SWITCH 3	BAUD RATE
OFF	OFF	9600
OFF	ON	19200
ON	OFF	57600
ON	ON	115200

Table 4. Setting the Hyperdrive10 Baud rate.

6.2 Input Connections

The Hyperdrive10 provides access to four inputs and two outputs via two connection blocks mounted along the PCB long edge. The inputs are on terminal block P4 and the outputs on block P3. These are shown in the middle of Figure 4. All connection blocks have a white dot marking terminal one. If that is difficult to see, they are the left-hand terminal of each block in Figure 4.

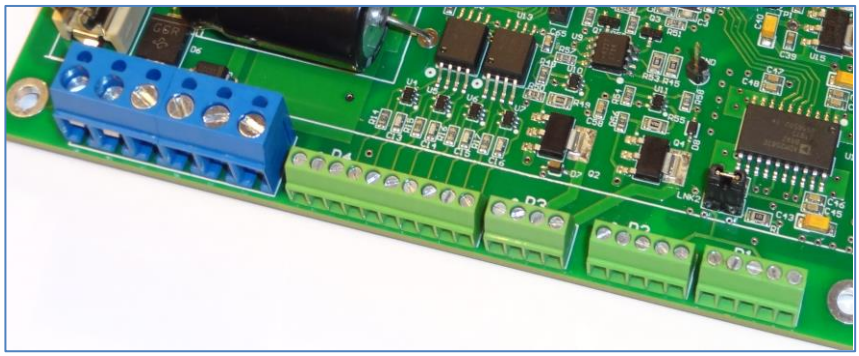

Figure 4. The Hyperdrive10 Power, Input, Output, and dual RS485 Connectors.

All inputs are electrically isolated from each other and the internal electronics. Figure 5 shows the general input circuitry. The internal switches are fed from constant current sources which limit the input current to under 10 mA for up to the rated input voltage maximum of 24 volts.

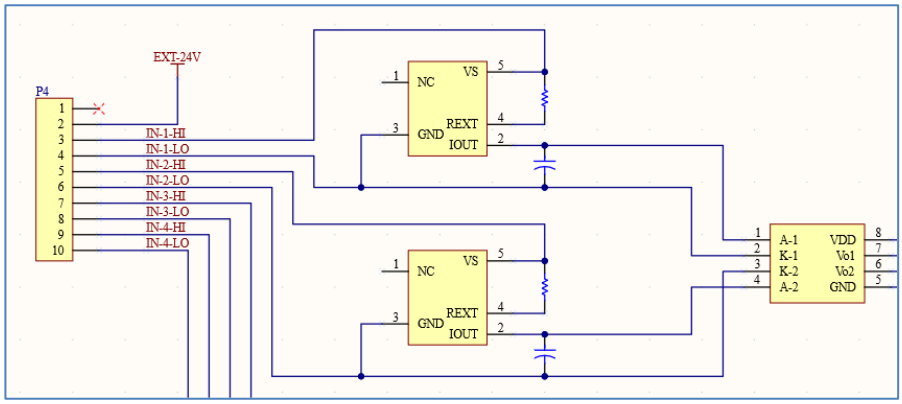

Figure 5. The input circuitry showing the constant current sources and how electrical isolation is implemented.

Table 5 lists the terminal numbers and functions for input connection block P4.

TERMINAL NUMBER	FUNCTION
1	NC
2	+24 volt reference
3	Input 1 hi side
4	Input 1 lo side
5	Input 2 hi side
6	Input 2 lo side
7	Input 3 hi side
8	Input 3 lo side
9	Input 4 hi side
10	Input 4 lo side

Table 5. The P4 terminal numbers and functions.

Note that the +24V nomenclature on terminal 2 is an input, not an output. If the external 24V power is referenced somewhere to the power ground, the 24V can be monitored from the **DF_24_VOLTS** Modbus register.

6.3 Output Connections

The Hyperdrive10 provides two switchable outputs that are isolated from the controller electronics. Figure 6 shows the typical circuit.

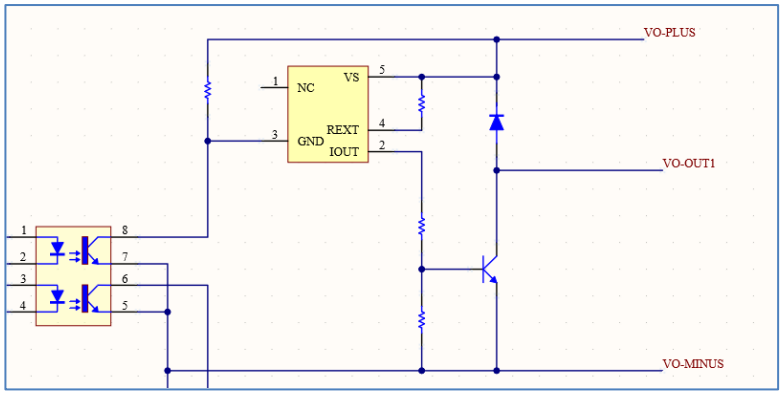

Figure 6. An example showing one of the two Hyperdrive10 outputs.

The internal processor provides separate drive for both outputs. OUT-1 is controlled by IN-3 and OUT-2 by IN-4 except when configured otherwise.

TERMINAL NUMBER	FUNCTION
1	Vo-MINUS (-)
2	Vo-PLUS (+)
3	Vo OUT-1
4	Vo OUT-2

Table 6. The P3 terminal numbers and functions.

6.4 RS485 Connections

The internal 4-wire plus ground RS485 connections are shown in Figure 7 and detailed in Table 7. The TX and RX connections are completely isolated from the Hyperdrive10 internal electronics. While the RS485-GND connection to terminal 5 on both P1 and P2 is used as a local RS485 shield on the PCB, it is also isolated from any internal connection.

The two 5-pin terminals P1 and P2 allow easy loop-in/loop-out of the RS485 wiring.

It is recommended that shielded, two twisted pair cable be used for the RS485 buss cabling. The shield should by grounded at the HMI or PLC end.

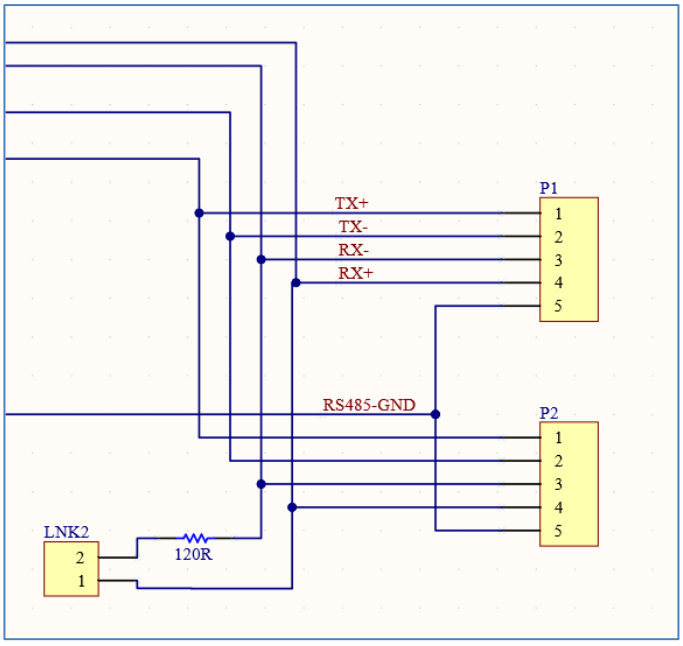

Figure 7. The internal wiring showing how the two RS485 connectors are wired and how the terminating resistor is switched in via LNK2.

TERMINAL NUMBER	FUNCTION
1	TX+
2	TX-
3	RX-
4	RX+
5	RS485-GND

Table 7. RS485 Connections.

Note: The wire designations are referenced to the Hyperdrive10. This means the RX lines will receive the messages from the HMI, and the TX lines will transmit the Hyperdrive10 responses.

6.5 Activity Light-Emitting-Diode (LED)

Without a connection to an HMI, the LED at position LED1 on the PCB will flash at a regular 5 Hz rate, indicating the software is running correctly. If it is not flashing, then something is wrong (see *Trouble Shooting* on page 11-58).

The LED will flash in response to each Modbus packet received when connected to a working HMI over the serial line. It will therefore be flickering constantly as packets are processed.

6.6 Motor and Power Connections

The motor DC power supply connections to connector P5 are shown in Table 8 and the typical input circuit protection is shown in Figure 8. Use wiring rated for the expected motor current.

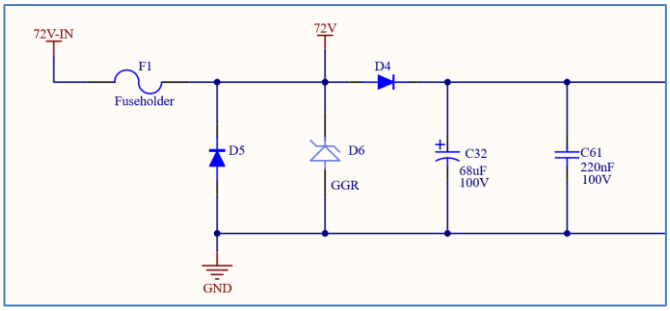

Figure 8. The main power input circuit protection.

The Hyperdrive10 can control bipolar two-phase motors (see however *Alternate Motor Wiring* on page 6-17). The motor will have four leads. Connect the motor using the following table and as shown in Figure 10.

NUMBER	NAME	MOTOR
1	POWER +	
2	POWER -	
3	A1	Field 1 wire 1
4	A2	Field 1 wire 2
5	B1	Field 2 wire 1
6	B2	Field 2 wire 2

Table 8. Hyperdrive10 power and bipolar motor connections P5.

Use a multi-meter or the manufacturer's documentation to select the two lead pairs associated with each field.

Keep the motor leads as short as possible. Ensure the cable used is rated for at least 20 Amps to minimise voltage drop. If possible, use 4-wire shielded cable with the shield connected to a frame ground only at the Hyperdrive end.

Once connected and operating, should the direction the motor turns be opposite to your requirement, simply swap either the two B wires or the two A wires.

6.6.1 Motor Status

The motor status is updated several times a second into the **D16_CONTROLLER_STATUS** and **DB_CONTROLLER_STATUS** registers. These read-only registers are detailed in Table 9 and contain various bit patterns that indicate the controller state and status. Several patterns are not applicable to the Hyperdrive10 and are marked as NA.

BIT-15	BIT-14	BIT-13	BIT-12	BIT-11	BIT-10	BIT-9	BIT-8
NA	NA	OCD	TH_STATUS		NA	UVLO	STCK_MOD
BIT-7	**BIT-6**	**BIT-5**	**BIT-4**	**BIT-3**	**BIT-2**	**BIT-1**	**BIT-0**
CMD_ERR	MOT_STATUS		DIR	NA	NA	BUSY	HiZ

Table 9. Controller status register descriptions.

The **HiZ** bit high (1) indicates the power bridges are off. **BUSY** is low (0) when a constant speed command is under execution. **DIR** indicates CW or CCW. **MOT_STATUS** is 00 for stopped, 01 for accelerating, 10 for decelerating, and 11 for constant speed. **CMD_ERR** means an illegal command was executed (the Hyperdrive10 software takes great care this does not occur, so please report any incidences to Kremford). **STCK_MOD** will be high if the **BIT_EXT_CLOCK** bit is set. **UVLO** is low for an undervoltage. **TH_STATUS** is internal temperature warning – 00 is normal, 01 is warning, 10 is bridge shutdown, 11 is device shutdown. **OCD** is an overcurrent state – this bit goes low and the controller will shut down.

6.6.2 Alternate Motor Wiring

The Hyperdrive is a bipolar motor driver, but other types of stepper motors can be configured to run as bipolar. A unipolar stepper with six leads can be simply connected as bipolar by leaving the field centre-tap wires disconnected. However, this will give a field coil inductance of four times that of a single coil, a definite problem if the motor is required to run at any speed and develop its rated torque. The correct connection for a six-lead unipolar motor is to leave the two opposite field coils unconnected as shown in Figure 9.

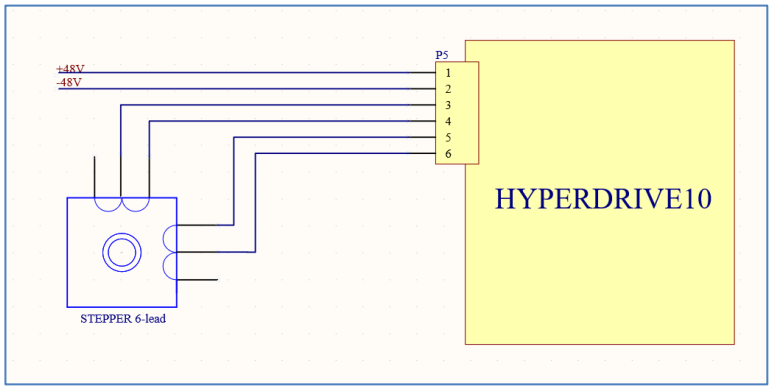

Figure 9. The connections for a 6-lead stepper motor.

6.6.3 Input Wiring for an External Control Configured Hyperdrive10

Figure 10 shows the P5 connections for a standard bipolar stepper motor.

It also shows the start stop connections on P4 typical of a Hyperdrive10 configured with the **BIT_START_BY_IN1**, **BIT_STOP_BY_IN2** and **BIT_DIRECTION_BY_IN3** bits set in the **BIT_CONTROLLER_MODE** register (See ***Error! Reference source not found.*** on page **Error! Bookmark not defined.**). In this configuration the motor is initially configured as to its operating conditions by Modbus commands such as power, accelerations, etc, but is started and stopped by external signals on inputs 1 and 2.

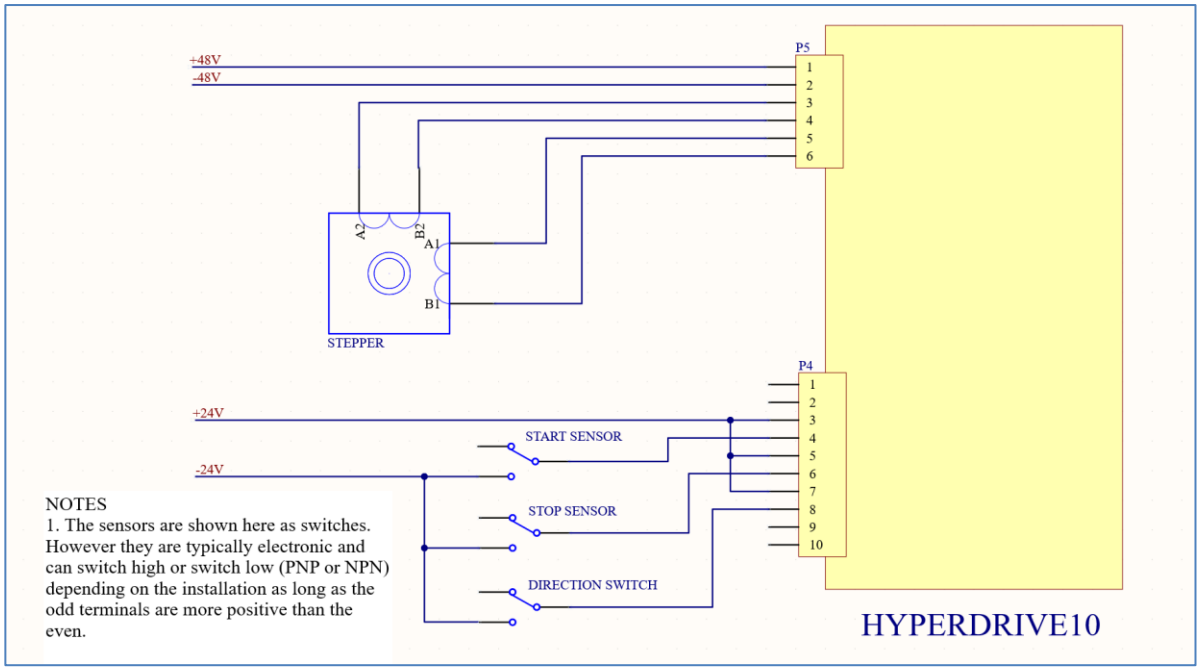

Figure 10. A typical bipolar motor and input wiring setup for a Hyperdrive10 in a start/stop external control system.

6.7 Starting for the First Time

The Hyperdrive10 is shipped configured as a stand-alone, 4-wire RS485 19,200 baud rate Modbus controlled stepper motor controller with a Modbus address of 2. This means the following hardware and Modbus settings have been pre-configured.

The Hardware settings are:

SETTING FOR	AS SHIPPED SETTING	NORMAL SETTING	COMMENTS
SW1	3 and 7 ON, all others off	1 OFF, baud rate in 2 and 3, address in 6, 7 and 8	See Figure 3 on page 6-11

The Modbus settings are:

MODBUS REGISTER	SETTING
DB_CONTROLLER_MODE	0
D16_STEP_MODE_INDEX	4

With no Modbus connection the activity LED on the PCB will regularly flash several times a second. However, once a Modbus connection that addresses this device (default 1) has been made the LED will flicker continuously indicating it is processing Modbus messages. See *A Stand-Alone Modbus Controlled Machine* on page 10-37 for examples of operating the Hyperdrive10 in this mode.

7 Introduction to the Hyperdrive10

The Hyperdrive10 controller controls one bipolar stepper motor. The motor size can range from medium to the very largest with a supply of up to 64 volts and up to 10 amps peak. The four electrically isolated inputs can respond in various ways to DC input signals of up to 24 volts. Inputs 1 and 2 can be configured to control the motor start and stop respectively. The two electrically isolated outputs can switch up to 64 volts at up to 500 mA.

The controller is configured by and responds to Modbus commands. These can be supplied by command packets sent over standard duplex (4-wire) serial RS485 or via a USB HID interface. The RS485 interface is electrically isolated. The legal Modbus packet types are 3, 4 and 16, providing standard read and write access to the internal registers. Legal number formats are 16-bit patterns, 16-bit integers, 32-bit integers, and 32-bit floating point numbers.

The following describes how Modbus interfaces with the Hyperdrive10 firmware.

7.1 Device Addresses

Each controller includes an eight-position switch (SW1) of which switches 6, 7 and 8 are used to select the controller's hardware Modbus ***device*** address. Up to seven controllers can be addressed using the usual binary switch pattern (see Table 3 on page 6-12). Address zero is special, so up to seven controllers can co-exist on a shared RS485 serial buss controlled by a supervising HMI, PLC, or other such Modbus controller. Switches 2 and 3 can select one of four standard Baud rates (see *Setting the Hyperdrive10 Baud rate.* on page 6-12.

There are many manuals and tutorials on the Modbus protocol and how it is used available in books and on the internet. The Modbus Organisation (modbus.org) has available a great number of documents.

Modbus messages include a destination device address. All Hyperdrive10 controllers on the buss receive and examine every Modbus message, but only the one with its switches set to this address will act and respond to that message. When a controller discovers it does not match the message address the message is harmlessly discarded.

> **Note**: In general, the Modbus device addresses are mutually exclusive.
> Ensure each Hyperdrive10 has a different address in Switches 6, 7 and 8.

Part of a Modbus message is the address of one or more ***registers.*** These are locations within the Hyperdrive10 memory that define various parameters associated with the controller functions. The message ***type*** determines whether the HMI is just querying for the value of that register, or the message contains a new value to update the register.

Each Hyperdrive10 contains two sets of registers called the ***device*** registers and the ***global*** registers (see ***Error! Reference source not found.*** on page **Error! Bookmark not defined.** and ***Error! Reference source not found.*** on page **Error! Bookmark not defined.**). Modbus messages addressed to a specific Hyperdrive10 will be addressing that controller's ***device*** registers. Modbus messages using a global address will address the ***global*** registers. The difference is that *device* reads and writes go to one specific Hyperdrive10, whereas *global* writes go to all Hyperdrive10 controllers on the RS485 buss.

For Hyperdrive10 controllers the *global* address is 99.

Both the device and global registers are divided into four groups that contain respectively, bit patterns, 16-bit integers, 32-bit integers, and floating-point numbers. The bit-patterns and numbers define the Hyperdrive10's internal functions, such as the motor speed, accelerations, motor powers, whether it is running or not, its step mode, temperature, etc.

All controllers, regardless of their switch address, will act upon a global message, updating their set of global registers as mentioned earlier.

The point of a global register is to store a bit or a number that is simultaneously available to all controllers. A bit pattern in one of the *device* registers can be used to designate that a Modbus command should use the equivalent *global* value rather than the *device* value. This means that after transmitting a global message the supervising HMI can execute a command confident that all controllers so configured will use that same global parameter. Common examples would be the overall machine speed and the machine start command – all motors must start at the same instant. Another would be a machine acceleration value, where all motors must start accelerating at the same time and remain in synch during the acceleration event.

The Modbus protocol requires a reply to every message but cannot handle multiple replies, so for Hyperdrive10 controllers a specific controller is configured to reply to *global* messages that go to all controllers. This is the controller at address 1 – the master and ***global responder***. While this protocol means there is no indication that each slave Hyperdrive10 controller really did successfully receive and store that message, that is reliably what happens. However, while slave controllers cannot write to any of their global addresses, those addresses are available for reading. Therefore, in a situation where the global write *must* be verified, after successfully writing to a global register the controlling PLC or HMI need just examine that global address in each slave to ensure all slaves were correctly set.

This mode is called the master/slave mode, where some commands rely on global or master register settings, but most use the device register settings.

7.2 Naming Modbus Registers

To an HMI the Modbus registers are referred to as being at a specific numerical address. For ordinary humans however the addresses are referred to by a specific name. In the following discussions dealing with the two sets of registers, the *device* register names begin with a **D**, and the *global* registers with a **G**. The four register types are 16-bit digital, 16-bit unsigned, 32-bit unsigned, and 32-bit floating-point. The associated *device* register names are preceded by **DB, D16**, **D32** and **DF**, respectively denoting the bit registers, the 16-bit integer registers, the 32-bit integer registers, and the floating-point registers. The *global* registers are similarly denoted by **GB**, **G16**, **G32** and **GF**. As an example, the 16-bit device register that defines the motor basic steps per revolution has the name **D16_STEPS_PER_REV**.

Here are some additional examples. Because the speed of the motor can be set separately or slaved to the global speed, there are two speed registers, both defining speed in steps per second: **D16_SPEED_SPS** and **G16_SPEED_SPS**. The speed can also be defined by the user to be some arbitrary but linear relationship such as RPM or Metres/Minute (see *Some Examples of Setting the Hyperdrive10 Gearing* on page 12-59). These map the arbitrary speed to steps per second. The

associated registers are naturally floating-point and their names are **DF_ARBITRARY_SPEED** and **GF_ARBITRARY_SPEED**.

The *device* registers are grouped by numeric type and are in the Modbus address range 0 to 499. The *global* registers are likewise grouped numerically and range from 600 to 1000. The **D16_SPEED_SPS** register for example is at address 215 (see *Hyperdrive10 Modbus Address Table*. on page 8-30).

7.3 Operating Modes

All controllers will simultaneously action global write commands. This can be advantageous with parameters such as acceleration, where all motors will accelerate together regardless of their individual speeds. Similarly, slaved speed means a master speed change will produce a relational speed change in slaves so configured, where the relationship can be direct or an arithmetic ratio.

Error! Reference source not found. on page **Error! Bookmark not defined.** describes the bits in the **DB_CONTROLLER_MODE** register. This is a bit pattern register and resides at address 7.00. Bits can be set individually or in groups using MODBUS RTU commands such as `4x_Bit` that include a device address and a bit pattern. The pattern is embedded in the address. For example, writing to the address 701 will set bit 1 of address 7. Some example configurations are described in the section *Hyperdrive10 Example Configuration Screens* starting on page 10-34.

7.4 External Clocking

The Hyperdrive10 can be operated as a simple, high powered stand-alone stepper controller using an external step pulse train to spin the motor. This mode is entered by setting the **BIT_EXT_CLOCK** bit in the **DB_CONTROLLER_MODE** register or having Switch 1 on at power up. In this mode the Hyperdrive10 needs no connection to a Modbus control system. However, it would typically have an initial setup phase where the parameters that apply in this mode would be set. They are retained in memory and so would survive power cycling.

The LNK3 link must be repositioned to enable external step pulses. See Figure 2 on page 6-11.

> **Note**. Do not attempt to return to normal operation by just clearing the **BIT_EXT_CLOCK** bit. Always perform a power cycle afterwards and before continuing.

There are only a few parameters that apply in this mode as the direction and step pulses (and consequently starting, stopping, accelerations and speed) are all provided externally.

When first selected (or at power up when Switch 1 on), Input 4 is isolated from the software (it must also be disabled in hardware by LNK3), and Input 3 is configured so the software can monitor its current state for the purpose of changing the motor direction.

The configuration for inputs 1 and 2 will be as initially set by previous Modbus commands. The running power and stopped power can be set using the two possible states of Input 1, and in this case the **BIT_EXT_CLOCK_IN1_EN** bit in the **DB_CONTROLLER_MODE** register must be set. The input *state* of Input 1 is sensed every 25 mSecs. Do not enable Input 1 interrupts for this case (ensure the **BIT_INx_ARMED** bit is clear in the **DB_INPUT_1_STATE** register). Note this means there is no debounce sensing, so ensure the input is electronic so debounce will not be a problem.

Use a pulse width ratio near 50% for the step pulses. The maximum step pulse frequency depends on the setting of the **D16_STEP_MODE_INDEX** register and the motor and its load. The Hyperdrive10 switching limit is way above any possible motor drive frequency. Medium sized high-quality motors should be capable of stepping at up to about 10,000 Hz or 3,000 RPM for a 200 steps/rev motor.

The external step pulses step the motor at whatever was the saved microstep setting in the **D16_STEP_MODE_INDEX** when the Hyperdrive10 was powered down. It is read when the controller is powered up.

For a 200 steps/rev motor, if the **D16_STEP_MODE_INDEX** was 0 then a 200 Hz pulse train will run the motor at 1 revolution/second or 60 RPM. Table 10 shows the relationship between the step mode index and how that will divide the input frequency within the motor's basic step angle.

For example, with the same motor, a step mode index of 3 will divide each step by 8, so now the steps/rev become 1600, and to get the same 60 RPM as before the step frequency will need to be 1600 Hz.

STEP MODE INDEX	MICRO-STEPS
0	1
1	2
2	4
3	8
4	16

Table 10. Relationship Between Step-mode-index and Microstepping

Direction control is provided by Input 3. Its state is sampled every 5 mSecs. It will have an effect on the next falling edge of the external step pulse. Up to what rate this will have an effect will depend on the motor and its load.

7.5 Starting, Stopping and Direction Control Using External Events

This is an extremely versatile Hyperdrive10 configuration that allows very fine control of the motor events where three inputs control the motor starting, stopping and direction, and the Modbus configuration defines exactly how these events are actioned.

When the **BIT_START_BY_IN1** bit is set a transition on the IN-1 input will either start the motor directly or start the motor after a delay. The motor will either run for a given number of steps or for a given time.

When the **BIT_STOP_BY_IN2** bit is set a transition on the IN-2 input can either stop the motor or stop the motor after a delay. An option allows spurious IN-2 transitions to be ignored.

When the **BIT_DIRECTION_IS_IN3** bit is set the IN-3 input will be examined just before the motor is started and its high or low state will set the motor direction.

The transition on IN-1 starts a sequence of events that depend on the content of several 32-bit registers and the state of several other bits. The sequence below demonstrates the possible sequences, starting immediately after the IN-1 state transition causes an IN-1 interrupt.

1. If the **BIT_DIRECTION_IS_IN3** bit is set, read the state of IN-3 and set whether the rotation direction will be CW or CCW.

2. If the **D32_PRERUN_TIME** register is non-zero, load and start a timer with its contents. The delay is in microseconds.
3. If the **D32_PRERUN_TIME** register is zero, or when the **D32_PRERUN_TIME** timer concludes, start the motor.
4. Examine the **BIT_STOP_BY_IN2** bit. This bit switches between two modes of subsequent operation:
5. If the **BIT_STOP_BY_IN2** bit is set:
   5.1. Examine the **D32_LOCKOUT_TIME** register. If it is non-zero, start a timer with its contents and set the lockout state (the delay is in microseconds). While the lockout timer is running transitions on IN-2 will be ignored.
   5.2. If the **BIT_RUN_STEPS** bit is set the motor will run in step mode for **D32_RUNTIME_STEPS** steps, otherwise it will run for **D32_RUNTIME_MSECS** milliseconds.
   5.3. When an IN-2 transition occurs, or when the first IN-2 transition occurs after the lockout timer concludes, the **D32_POSTRUN_TIME** register is examined. If the **D32_POSTRUN_TIME** register is zero the motor is stopped, otherwise its contents are loaded into a timer and the motor will continue running for that many microseconds before it stops.
6. If the **BIT_STOP_BY_IN2** is clear:
   6.1. If the **BIT_RUN_STEPS** bit is set the motor will run in step mode for **D32_RUNTIME_STEPS** steps, otherwise it will run for **D32_RUNTIME_MSECS** milliseconds.

The Hyperdrive10 keeps a count of the IN-1 transitions in this mode and it is available in the **D32_EVENT_COUNT** register.

Some examples of how this configuration can be applied are given in *Hyperdrive10 Example Configuration Screens* on page 10-34. The input wiring is shown in Figure 10 on page 6-18.

7.6 Speed and Gearing Control

Speed internally is implemented as a 16-bit integer and is always specified in steps/second. The *device* speed address is **D16_SPEED_SPS** and the *global* address is **G16_SPEED_SPS**.

> **Note**: Both the speed and accelerations are independent of the step mode index – they are always specified in terms of the motor's basic steps/rev.

While the speed can be set via these two sps registers, it is most often set in the **DF_ARBITRARY_SPEED** or **GF_ARBITRARY_SPEED** registers using a machine-based speed such as RPM or Metres/Minute. The Hyperdrive10 uses this value and the **DF_DRIVE_GEARING** register to produce the actual motor speed in steps/second to load into the **D16_SPEED_SPS** register.

Unless the **BIT_GLOBAL_SPEED** bit is set in the **DB_CONTROLLER_MODE** register, speed commands addressed to a particular device will use the *device* **D16_SPEED_SPS** register. If the **BIT_GLOBAL_SPEED** bit is set the Hyperdrive10 will use the *global* **G16_SPEED_SPS** register. In global speed control these registers are usually set indirectly by the **DF_ARBITRARY_SPEED** and **GF_MASTER_GEARING** registers.

The relationship between the motor speed and the output speed is provided by the multiplier **DF_DRIVE_GEARING** parameter. This number describes the entire gear train from the motor including

any translation to linear motion. The relationship must be defined by the engineer with knowledge of the drive gear train and the resulting conversion number entered into the **DF_DRIVE_GEARING** register. For systems where multiple motors must be synchronised, the gearing value for the master motor must be entered into the global **GF_MASTER_GEARING** register. The chapter *Some Examples of Setting the Hyperdrive10 Gearing* on page 12-59 describes several examples of how to develop the Gearing term for various machines.

7.7 Motor Acceleration Matching

All controllers will use the global accel/decel rates when a global speed command arrives in multi-motor installations if the **BIT_GLOBAL_ACCEL** and **BIT_GLOBAL_SPEED** bits are set. By convention, in a multi-motor machine the controller at address 1 is considered the mechanical master (the drive system with the highest inertia) which all other controllers must match during accel/decel events. However, different gearing requirements can mean that although their gearbox or drive outputs are at the same speed, the motors themselves may run at different speeds and so a common accel/decel rate cannot be used.

By setting the **BIT_MATCH_ACCEL_TIME** bit, each Hyperdrive10 will use the ratio between its own **DF_DRIVE_GEARING** and the **GF_MASTER_GEARING** to modify the device accelerations so it will take the same time to reach a new speed as the master system.

This means that connected systems with multiple drives can remain exactly in synchronism during speed changes. A machine with multiple connected conveyors is an example where this capability would be a requirement.

7.8 Constant Deceleration

When motors are using high deceleration rates greater than about 8000 sps^2, there is often a requirement for the deceleration time to be constant no matter what the motor speed. An example would be a motor presenting something like a label for pickup. The label length presented must be the same no matter what the running speed of the motor or machine, meaning the time the motor takes to stop must be the same for any speed.

The Hyperdrive10 provides this capability if the **BIT_CONST_DECEL** bit is set. It works by always matching a pre-computed deceleration time based on the programmed deceleration rate of **D16_DECEL_SPS2** (or **G16_DECEL_SPS2**) and the device max speed of **D16_MAX_DEVICE_SPS**.

This means the deceleration time will always be the same as if the motor is decelerating from the maximum speed for that device.

> **Note**. This option is only applicable if the **BIT_START_BY_IN1** bit is set.

The default setting for **D16_MAX_DEVICE_SPS** (and also the global machine maximum speed **G16_MAX_MACHINE_SPS**) is 4000 steps/second. This is quite high, and for a standard 200 steps/rev motor is 1200 RPM.

> **Note**: When configuring a new machine, you should always confirm these HMI maximum speeds match the actual machine maximum speeds.

As an example, a 200 steps/rev motor driving a 60 mm diameter squeeze roller in a machine with a maximum speed of 30 Metres/min, that maximum motor speed is just 530 sps (see *Some Examples of Setting the Hyperdrive10 Gearing* on page 12-59).

A constant deceleration is produced by inserting a delay between the external stop signal and the actual stop signal. The delay is computed from:

$$InsertDelay = \frac{MaxSpeed_{SPS} - CurrentSpeed_{SPS}}{DecelRate_{SPS^2}}$$

The resolution of the internal timer limits the maximum allowable deceleration time to 0.65536 seconds. When using this constant decel option, ensure all possible cases of the above satisfy this relationship:

$$InsertDelay < 0.65536$$

If the constant is exceeded the constant deceleration rate option will just stop normally. The HMI can use the **BIT_TOO_SLOW_DECEL_RATE** bit in the **DB_MACHINE_STATE** register to monitor this test after changes to any of these parameters. It is set if the constant is exceeded.

7.9 Incremental Acceleration

In complex machines it is often a requirement that the motors must remain synchronised during machine acceleration events. The Hyperdrive10 solves this problem for motors with similar acceleration times by providing the acceleration matching described in *Motor Acceleration Matching* on page 7-24.

The difficulty arises where one or more motors are performing start/stop functions where their run speed must match the machine, but their individual acceleration rates are much higher. An example would be the label feed motor in a standard labeller machine – the label must be quickly accelerated up to the conveyor speed to match the passing product and just as suddenly stopped ready for the next feed. While the basic machine acceleration rate might be around 1,000 steps/second^2, the stop/start motors may be as high as 25,000 sps^2.

Unless the high rate motors change their speed in synchronism with the changing conveyor speed throughout the acceleration event the synchronization between the label and the product will be lost during that acceleration event and only regained when all motors are up to the new speed.

If the start-to-stop time of the high rate motor is significantly less than the machine acceleration time, then the speed of that motor must be progressively changed when it runs to maintain synchronisation. In this case the high rate motor controller will need to be re-calculating the required instantaneous speed throughout the machine acceleration event. Figure 11 shows an example situation of a conveyor accelerating from 10 to 20 M/m at 2,000 sps^2 (red) and the label controller (at 20,000 sps^2) matching the acceleration rate every 20 mSecs throughout the conveyor event. It is obvious that without this mechanism, the label controller would have matched the destination conveyor speed in a little over 10 mSecs, giving an initial speed difference of over 8 M/m that decreases over the rest of the acceleration event. A label applied during that time would not be correctly aligned.

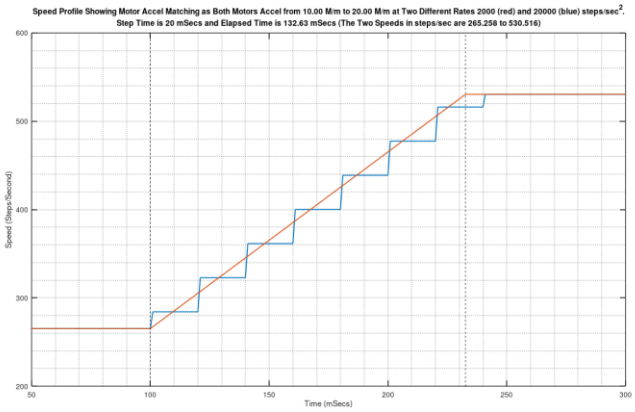

Figure 11. The slow rate motor accelerating (red) and the high rate motor (blue) matching its speed incrementally throughout an acceleration event.

This speed matching is provided automatically by the Hyperdrive10 when operating in slave mode and the **BIT_INCREMENTAL_ACCEL** bit is set. Speed updates will occur throughout the acceleration event and update the **G16_SPEED_SPS** register whether the motor is running or not. Whenever the high-rate motor starts it will use the latest speed setting.

7.10 Using Distance to Define Time

Often in machines that translate the rotary motion of the motor into a linear motion, it is necessary for some linear measurement to be expressed as a time delay. A typical example is where a product on a conveyor is detected at some fixed point along the conveyor and at some later fixed point some action is taken with or to that product. A simple delay may be adequate for a fixed speed conveyor, but where the conveyor speed can be varied at will, it is much easier for the user to specify the fixed measurement between the two stations and have the machine automatically provide the correct speed-based delay.

The three IN-1 timers (**D32_PRERUN_TIME**, **D32_POSTRUN_TIME** and **D32_LOCKOUT_TIME**) can be loaded with delays that are in-use transferred to internal Hyperdrive10 timers. The **PRERUN** delay is used to specify the delay between the IN-1 signal and the motor start. The **D32_POSTRUN_TIME** and **D32_LOCKOUT_TIME** delays are started by the IN-2 event with the end of the **D32_POSTRUN_TIME** delay stopping the motor (see *Starting, Stopping and Direction Control Using External Events* on page 7-22).

When the **BIT_USE_DISTANCE** bit is set these three registers are loaded with the result of converting the user entered distances from the three distance registers **D16_PRERUN_DISTANCE**, **D16_POSTRUN_DISTANCE** and **D16_LOCKOUT_DISTANCE** into delays that correspond to the measurement at the current speed. The user enters values to these registers that are machine-based measurements typically in millimetres. The measurements in these registers are defined as:

D16_PRERUN_DISTANCE	The distance to travel after an Input 1 event until the motor must start.
D16_POSTRUN_DISTANCE	The length delivered after an Input 2 event before the motor is stopped.
D16_LOCKOUT_DISTANCE	The distance to travel in lockout mode after the motor is started. The intent is to block any spurious Input 2 events that may possibly occur before the genuine Input 2 event.

When the **BIT_USE_DISTANCE** bit is clear the distance registers play no part and the user enters delays in microseconds into the **D32_PRERUN_TIME**, **D32_POSTRUN_TIME** and **D32_LOCKOUT_TIME** registers as required.

> **Note**: It is important to decide which mode to use for a particular machine. While both the distance and time registers *can* both be displayed simultaneously on an HMI, you should consider the possible confusion this may cause for the user.

When the delay represents distance, some simple internal math is required using a constant to convert the distance to time. The constant is stored in the **DF_DISTANCE_NUMERATOR** register. The simple derivation of the constant is given in *How to Derive the Distance Measuring Multiplier* on page 13-61, but it is just the inverse of the arbitrary speed measurement converted to microseconds. The metric example given derives the conversion constant as 60,000 for an arbitrary speed expressed in Metres/minute.

If the **BIT_USE_DISTANCE** bit is set, this computation is performed on all three registers every time a new arbitrary speed is set and individually for every change to their respective measurements.

7.11 The Four Movement Profile Power Settings

The Hyperdrive10 can set individual power settings for the four states of a movement profile. These are acceleration, constant speed, deceleration, and stopped. Each has its power controlled by the corresponding Modbus register. An additional switch option turns the output bridges off altogether to allow manual motor movement without powering down the system.

The four registers are **D16_RUN_PWR**, **D16_STOPPED_PWR**, **D16_ACCEL_PWR** and **D16_DECEL_PWR**. The output bridge can be turned off at any time the motor is not running by setting the **BIT_SOFT_HIZ** bit in the **DB_MOTOR_STATE** register.

The four power settings have the range 0-100 where 100 sets the peak current delivered to the motor at 10 amperes. Note that the average current as read on an ammeter will be considerably less than this. A setting of 100 is very unusual and you should always try and use the minimum power setting the load requires. Sometimes a load with high inertia can use a higher acceleration power setting that the run setting. The stopped power setting should be just enough to hold the load in place when stopped. Anything higher will just unnecessarily raise the stationary motor temperature.

The fan in the Hyperdrive10 will start at a sensed internal temperature of 40°C. This temperature can be monitored from the **D16_CHIP_TEMPERATURE** register. Apart from this overall temperature monitor, the Hyperdrive10 repeats the two TH_STATUS controller internal temperature bits in either the **DB_CONTROLLER_STATUS** or **D16_CONTROLLER_STATUS** registers (see Table 9 on page 6-16). These two bits describe the internal thermal state of the controller. Both bits should normally be clear, but if your application consistently drives the Hyperdrive10 at high power, it would be wise to monitor their state. Table 11 lists the temperature thresholds at which the given event occurs.

TH_STATUS		EVENT	EVENT TEMPERATURE THRESHOLD (°C)
0	0	NORMAL	
0	1	WARNING	135
1	0	BRIDGE SHUTDOWN	155
1	1	DEVICE SHUTDOWN	170

Table 11. The controller status register parameters.

The controller also includes protection against high instantaneous peak currents that consistently exceed the 10 amps peak rating. Should this situation occur, it will raise the OCD bit in the status register (see Table 9) and shut down the bridge. This will be obvious if it occurs but by monitoring and perhaps latching this bit, the shutdown reason will be obvious.

8 The Modbus Messages

8.1 The Hyperdrive10 Modbus Message Structure

The Hyperdrive10 uses only a subset of the Modbus command set, types 3, 4 and 16. In these commands it has a restricted format for multiple reads and writes. The Modbus message structures used in the Hyperdrive10 are defined below.

8.1.1 Message Type 3

Read bit register value.

INDEX	SIZE	DESCRIPTION	COMMENTS
0	BYTE	Device address	Range is 1 to 7 and 99
1	BYTE	Function	3
2	16-BITS	Starting register address	See below
4	16-BITS	Number of registers	Usually 1
6	16-BITS	CRC checksum	

A message type 3 can contain several register requests, but these situations require the register updates to have consecutive addresses in the Hyperdrive10 memory. In most cases this will not apply and there will be only one register update per message.

8.1.2 Message Type 4

Read register value.

INDEX	SIZE	DESCRIPTION	COMMENTS
0	BYTE	Device address	Range is 0 to 7 and 99
1	BYTE	Function	4
2	16-BITS	Starting register address	See below
4	16-BITS	Number of registers	Usually 1
6	16-BITS	CRC checksum	

A message type 4 can contain several register requests, but these situations require the register updates to have consecutive addresses in the Hyperdrive10 memory. In most cases this will not apply and there will be only one register update per message.

The request can be for both 16-bit and 32-bit bit numbers, with the latter including floating point numbers.

8.1.3 Message Type 16

Write multiple registers.

INDEX	SIZE	DESCRIPTION	COMMENTS
0	BYTE	Device address	Range is 0 to 7 and 99
1	BYTE	Function	16
2	16-BITS	Starting register address	See below
4	16-BITS	Number of registers	
6	BYTE	Count of data bytes	
7	16-BITS	data	
9	16-BITS	data	
11	16-BITS	CRC checksum	

A message type 16 can contain several register requests, but these situations require the register updates to have consecutive addresses in the Hyperdrive10 memory. In most cases this will not apply and there will be only one register update per message.

Because both 16- and 32-bit numbers are transferred by message 16, the data starting at index 7 can sometimes be 16-bits and at other times 32-bits (including FP numbers). The Hyperdrive manages this by subdividing the device and global register bank addresses into four subsections and specifying what data type resides at those addresses.

The address map is shown in Table 12 with each entry designating the start address of each sub-group.

DEVICE		GLOBAL	
TYPE	ADDRESS	TYPE	ADDRESS
DB_	100.00	GB_	600.00
D16_	201	G16_	701
D32_	301	G32_	801
FP_	401	GF_	901

Table 12. Hyperdrive10 Modbus Address Table.

Thus, to address the **D32_PRERUN_TIME** register within the HMI code, the PLC is the MODBUS RTU and the address **type** is 4x_MAX2W (two consecutive Modbus words). The **format** is **32-bit Unsigned** specified as 8 digits to the left of the decimal point, and 0 digits to the right (8.0). In most HMI software the device and address are specified with the device number first, a hash (#) separator, and then the register address. The **D32_PRERUN_TIME** register is at address 305 in the Hyperdrive10, and so for device 2 would be written as 2#305.

Figure 12 shows the dialog from the EZwarePlus HMI software where the settings above are being entered for the **D32_PRERUN_TIME** register of device 2.

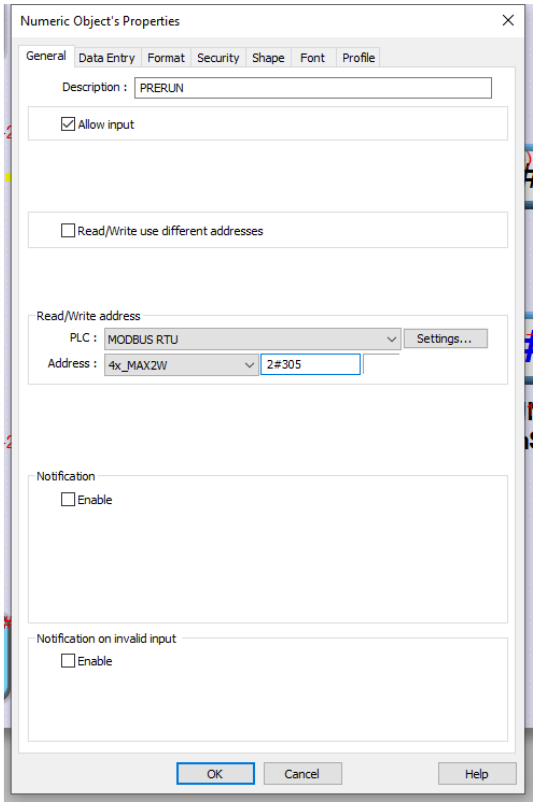

Figure 12. The EZwarePlus HMI software showing the property page for the D32_PRERUN_TIME register for device 2 at address 305.

Another example would be the Hyperdrive10 device **DF_DISTANCE_NUMERATOR**. This is at address 413 and is therefore a floating-point number (see Table 12). The address **type** in this case is still 4x_MAX2W but the **format** is now a **32-bit Float** with a 5.1 display format and an address for the same device two of 2#413.

A 16-bit example would be **D16_ACCEL_SPS2**. This is at address 213 and so is a 16-bit number. The address type is 4x_MAX1W for just one Modbus word, the **format** is **16-bit Unsigned** with a 5.0 display format.

Bit patterns are a little trickier, especially the global ones. The **DB_CONTROLLER_MODE** register detailed in ***Error! Reference source not found.*** on page **Error! Bookmark not defined.** has the address 700 which at first would seem to be buried in the global G16 address range. The difference here is that this is a bit pattern, and in fact the address is just 7. The 16 mode bits in this register are addressed individually in the two left-most digits. The range is 0 to 15 and so accessing the **BIT_GLOBAL_ACCEL** bit at BIT_3 is addressed in the register as 703. The bit at 700 is **BIT_EXT_CLOCK**. The bit at 713 is **BIT_USE_DISTANCE**.

The Global bit register called **GB_MASTER_STATE** is at address 60100 which is really address 601 with the trailing two digits designating the bit pattern as above. The **BIT_MASTER_ON** bit is at address BIT_0 and so is addressed as being at address 60100. The type is 4x_bit and the address is written as 99#60100.

8.1.4 The USB Packet Description

The USB implementation in the Hyperdrive10 is a standard binary message HID format. Any HID driver should present no trouble in being configured as a driver. The HID packet structure is shown in Table 13.

Table 13. USB Packaged MODBUS Message.

SOP	REV	LENGTH	MODBUS	EOP
5E	01	<n>	packet	E5

The **LENGTH** is the length of the Modbus message in bytes. The embedded message is standard Modbus without the Modbus CRC. The packet is preceded by an 0x5E SOP byte and a Revision byte, currently 0x01. The trailing 0xE5 EOP byte completes the packet. The maximum allowable packet size is 20 bytes. The Hyperdrive10 software unpacks this packet from the standard USB data block, adds a standard Modbus CRC, and inserts it into the internal message queue for processing. This is the same queue also filled from the RS485 USART interface, so a USB message seamlessly fits into the messages to be processed. Both message types include a message source byte that remains visible during processing, so the Modbus reply is always returned to the correct source.

The Modbus reply to a USB message begins with the standard USB packet-sent byte **USB_EVT_IN** (3) followed by a length byte followed by the standard Modbus reply message.

9 Configuring the Inputs to Outputs

There are several different configurations for the inputs that are associated with controlling the motor in some way. Of the four inputs, inputs 1 and 2 use interrupts to ensure high speed responses and can be configured to trigger on high or low going input transitions. Inputs 3 and 4 are sampled every 5 mSecs and are level sensitive.

When the Hyperdrive10 is not configured in external clocking mode, Inputs 3 and 4 are configured to control Outputs 1 and 2 respectively. However, both these inputs are involved in controlling the external clocking mode, so if the SW1 switch 1 is on both *outputs* will be forced off.

Also, in the **BIT_START_BY_IN1** mode, if the **BIT_DIRECTION_IS_IN3** bit is on in the **DB_CONTROLLER_MODE** register, then Input 3 is involved with setting the motor direction. In this case Output 3 is disconnected and turned off, leaving only Input 4 to control Output 2.

MODE BIT SETTINGS	INPUT 3	OUTPUT 1	INPUT 4	OUTPUT 2
	Controls Out-1	Controlled by In-3	Controls Out-2	Controlled by In-4
BIT_EXT_CLOCK	Motor direction	Off	Step pulse source	Off
BIT_START_BY_IN1	Controls Out-1	Controlled by In-3	Controls Out-2	Controlled by In-4
BIT_START_BY_IN1 and BIT_DIRECTION_IN_IN3	Motor direction	Off	Controls Out-2	Controlled by In-4

10 Hyperdrive10 Example Configuration Screens

In this section we show some example HMI screens using standard HMI screen *graphic objects* that provide control for a Hyperdrive10. It is of course, the choice of the machine manufacturer which HMI and/or PLC to use, but these descriptions use Maple Systems HMIs and their free HMI programming software called EBPro. Most HMI manufacturers use the same standard Modbus terminology and provide similar PC based software for managing their HMI displays.

All these sample projects include screens that allow setting the controller's modes and various motor power and acceleration requirements. The names of these registers and bits within the bit pattern registers that affect these are functions listed in . While access to the registers are nearly always required for interactive control, in production systems some can easily be hidden from unauthorised access using the various HMI manufacturer's access schemes. In some cases, the settings could be pre-programmed in the factory.

10.1 A Stand-Alone Externally Clocked Example

While something of an overkill, the Hyperdrive10 can be run without a Modbus interface by changing the SW1 and LNK3 settings on the Hyperdrive10 circuit board and providing an external clock source. This moves all responsibility for movement profiles to an external PLC or other controller, where the motor movement is controlled by an external step pulse stream.

In this mode Input 2 (P4-5,6) can control the motor power, Input 3 (P4-7,8) can control the motor direction, and Input 4 (P4-9,10) is connected to the external step pulse input. Figure 13 shows the complete connections for this mode.

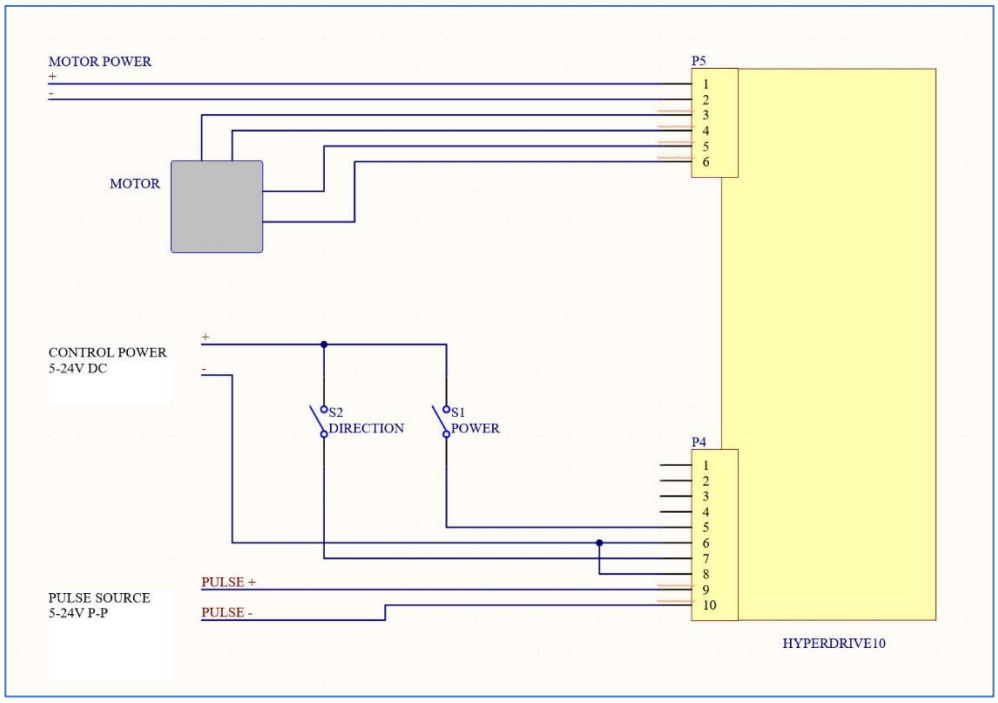

Figure 13. Connections for complete external stepping control.

If Input 2 is not connected or the input is low, the motor is energised at its **D16_RUN_PWR** setting. If this input is high the motor is energised at its **D16_STOPPED_PWR** setting. It can be changed at any

time. Note that this input always defines the motor power whether it is running or not, and in normal use would be used to switch the motor power between when it is running and when it is stopped, a function the Hyperdrive10 provides automatically in any other mode.

If Input 3 is not connected or the input is low, the motor will rotate in a direction based on its initial connections. If this input is high the motor will rotate in the opposite direction. This input can only be changed while the motor is stopped (no input pulse train).

Connect the pulse generator to Input 4. Pulse requirement is a pulse width ratio of nominally 50% and a voltage swing of 5V P-P to 24V P-P. The motor steps on the negative edge.

The following steps describe the necessary connections when using this mode.

- On the PCB set switch 1 in SW1 to ON, all others to OFF (see Figure 3 on page 6-11).
- Ensure LNK3 is linked as shown in the right-hand side of Figure 2 on page 6-11.
- Connect the power and motor cabling to connector P5 as per Figure 13.
- Connect a square-wave frequency source to connector P4 pins 9 and 10 with pin 9 the more positive. Set the frequency to zero, the pulse width nominally to 50%, and the voltage swing to 5V p-p or more but less than 25V p-p.
- Connect a switch to connector P4 pins 7 and 8 to control the motor direction as shown in Figure 13.
- Connect a switch to connector P4 pins 5 and 6 to control the motor power.
- Check again all the connections and apply the motor power. The LED should be flashing about five times a second.
- Gradually increase the frequency to a hundred or so hertz. You should see the motor slowly turning. If not go back and check the previous settings.
- The default step mode is 4, and that denotes a microstep of 16 (see Table 10 on page 7-22). For a 200 steps/rev motor this means a pulse frequency of 200 * 16 = 3200 hertz will produce one revolution per second or sixty revolutions per minute (RPM). Set the frequency source to 3200 Hz and check this.
- Try further changes to the frequency.

The motor settings used in this example can be modified by connecting a Modbus interface. This will allow modifying the default settings as well as setting both a run and a stopped power setting.

> **Note**: The default settings will automatically replace all current saved settings if the Hyperdrive10 is powered up with switch 4 in SW1 in the ON position. Do not leave this switch in the on position.

The screen shown in Figure 14 is a suggested HMI configuration for control in external stepping mode. The left-hand column contains three read-only numerics that show the controller temperature, the motor power voltage, and the sensor voltage (if connected). The right-hand column contains three numerics that show the normal power to the motor (actually the **D16_RUN_PWR**) and the switched or low power to the motor (actually the **D16_STOPPED_PWR**). You select the low power mode by applying a high voltage to Input 2 (S1 in Figure 13). In this mode the **LOW POWER** lamp will light (the

BIT_INPUT2_STATE bit in the **DB_MACHINE_STATE** register). The third numeric can select from the four microstep settings.

In external step mode the Hyperdrive10 by default is configured to energise the motor automatically on power up. This state is configured when the **BIT_EXT_ALWAYS_ON** bit in the **DB_MACHINE_STATE** register is set on power up. That bit being set means there is no need for any external communication connection and the motor will be energised about one second after power up, allowing the motor to run as soon as a step pulse arrives.

Clear the **BIT_EXT_ALWAYS_ON** bit should you need the motor to power up off.

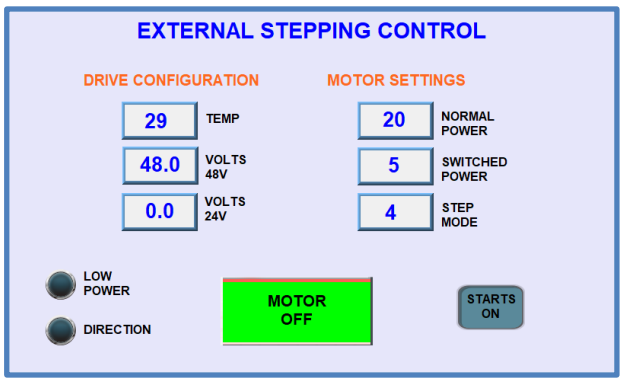

Figure 14. An example screen showing the configuration registers that are applicable when in external stepping mode

The large button marked MOTOR OFF is a Modbus toggle switch and controls the **BIT_DEVICE_ON** bit in the **DB_MOTOR_STATE** register. As a result of the **BIT_EXT_ALWAYS_ON** bit being set this button is showing that power is ON (green). The button marked STARTS ON controls the **BIT_EXT_ALWAYS_ON** bit and is showing the bit is set. Pressing this to clear the bit and then power cycling the Hyperdrive10 will change the screen to that shown in Figure 15.

Now the STARTS ON/STARTS OFF button is showing STARTS OFF and the motor power button is showing off (red). You now need to press this button to power the motor - you have power up control of the motor power state.

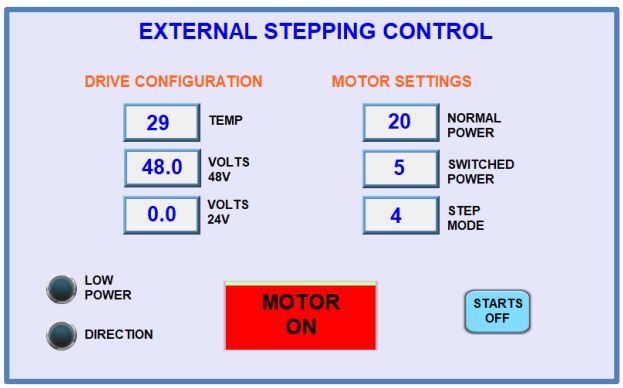

*Figure 15. The same example screen but showing how the motor is not energised after power up when the **BIT_EXT_ALWAYS_ON** bit is cleared.*

If required, an external PLC or similar can implement one or both the external switches called S1 and S2 in Figure 13 for direction and power control.

The function and register names of the HMI objects in Figure 14 are shown in the following table:

OBJECT FUNCTION	REGISTER NAME	FORMAT	TYPE	ADDRESS
TEMP	D16_CHIP_TEMPERATURE	16-bit unsigned	4x_MAX1W	210
VOLTS 48V	DF_48_VOLTS	32-bit floating point	4x_MAX2W	403
VOLTS 24V	DF_24_VOLTS	32-bit floating point	4x_MAX2W	405
NORMAL POWER	D16_RUN_PWR	16-bit unsigned	4x_MAX1W	206
SWITCHED POWER	D16_STOPPED_PWR	16-bit unsigned	4x_MAX1W	207
STEP MODE	D16_STEP_MODE_INDEX	16-bit unsigned	4x_MAX1W	205
LOW POWER	DB_MACHINE_STATE	BIT_INPUT2_STATE	4X_BIT	2.04
DIRECTION	DB_MACHINE_STATE	BIT_INPUT3_STATE	4X_BIT	2.05
MOTOR ON/OFF	DB_MOTOR_STATE	BIT_DEVICE_ON	4X_BIT	1.00
STARTS OFF/ON	DB_MACHINE_STATE	BIT_EXT_ALWAYS_ON	4X_BIT	2.01

10.2 A Stand-Alone Modbus Controlled Machine

The screen shown in Figure 16 is probably the minimum to get a Hyperdrive10 motor revolving on a Modbus connection. It includes a button to start and stop the motor and another button to change the direction. A small numeric in the top-right corner displays the **D16_MODBUS_ADDRESS** register which repeats the device address set in the Hyperdrive10 switches. Two large numerics provide for setting the steps/second **D16_SPEED_SPS** register, or more typically by indirectly setting the sps via the arbitrary speed register **DF_ARBITRARY_SPEED**.

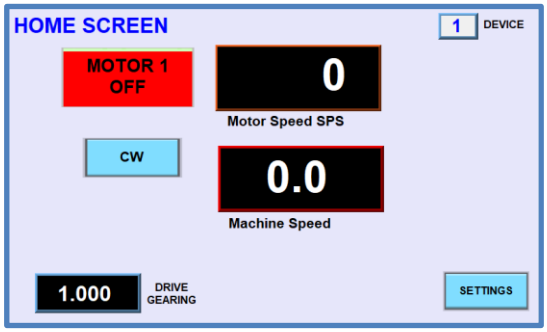

Figure 16. Basic Modbus motor control screen.

To allow testing this mode also included is a numeric providing access to the drive gearing register **DF_DRIVE_GEARING**. This defaults to 1.0. Doing some of the sums in the chapter *Some Examples of Setting the Hyperdrive10 Gearing* on page 12-59 and trying them in this numeric will show how they take the number entered into the Machine Speed numeric and are translated into a new Motor Speed SPS value.

For example, entering a gearing of 3.333 here for a 200 steps/rev motor will allow setting the motor shaft speed in RPM (60 * 3.333 = 200). The SETTINGS button just changes the display to the Motor Settings screen in Figure 17.

The HMI objects in this screen are listed here:

OBJECT FUNCTION	REGISTER NAME	FORMAT	TYPE	ADDRESS
Device Address	D16_MODBUS_ADDRESS	16-bit Unsigned	4x_MAX1W	211
Motor ON/OFF	BIT_DEVICE_ON	Toggle Switch	4x_Bit	100
Speed (SPS)	D16_SPEED_SPS	16-bit Unsigned	4x_MAX1W	215
Direction (CW/CCW)	BIT_MOTOR_DIRECTION	Toggle Switch	4x_Bit	101
Machine Speed (RPM)	DF_ARBITRARY_SPEED	32-bit Float	4x_MAX2W	409
Drive Gearing	DF_DRIVE_GEARING	32-bit Float	4x_MAX2W	407

10.2.1 The Minimum Motor Parameter Setup Screen

Figure 17 shows what would be a typical set of controls for configuring the motor in a Modbus environment. The column of numerics in the middle configure the motor itself with the four power settings and the two acceleration rates. The left-hand group includes the input power voltage monitor, the temperature, a button to manually control the fan, the motor's designated steps/revolution, the microstep setting (See Table 10), and the maximum allowed speed (in SPS). The right-hand button is a short-cut to setting the **DB_CONTROLLER_MODE** register for stand-alone Modbus control. The HOME button just returns to the Home screen.

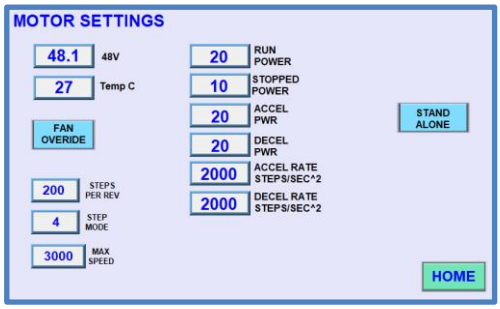

Figure 17. A minimum motor settings screen.

The HMI objects in this screen are:

OBJECT FUNCTION	REGISTER NAME	FORMAT	TYPE	ADDRESS	READ ONLY
Input Voltage	DF_48_VOLTS	32-bit Float	4x_MAX2W	403	YES
Temperature	D16_CHIP_TEMPERATURE	16-bit Unsigned	4X_MAX1W	210	YES
Fan Override	BIT_FAN_OVERIDE_ON	Toggle Switch	4x_Bit	200	
Run Power	D16_RUN_PWR	16-bit Unsigned	4x_MAX1W	206	
Stopped Power	D16_STOPPED_PWR	16-bit Unsigned	4x_MAX1W	207	
Accel Power	D16_ACCEL_PWR	16-bit Unsigned	4x_MAX1W	208	
Decel Power	D16_DECEL_PWR	16-bit Unsigned	4x_MAX1W	209	
Acceleration Rate	D16_ACCEL_SPS2	16-bit Unsigned	4x_MAX1W	213	
Deceleration Rate	D16_DECEL_SPS2	16-bit Unsigned	4x_MAX1W	214	
Steps/Rev	D16_STEPS_PER_REV	16-Bit Unsigned	4x_MAX1W	212	
Step Mode	D16_STEP_MODE_INDEX	16-Bit Unsigned	4x_MAX1W	205	
Stand Alone	BIT_STAND_ALONE	Toggle Switch	4x_Bit	202	

These are all fairly straightforward. The power settings have the range 0-100, but very few motors will need settings much over 50. They very much depend on the motor's internals and its load torque requirement. The STAND ALONE button sets the **BIT_STAND_ALONE** bit in the **DB_MACHINE_STATE** register which then clears all bits from the **DB_CONTROLLER_MODE** register, clearing any dependence on global registers.

10.2.2 Using this HMI Configuration

Power up the Hypedrive10 and your HMI. At this point the output bridges are still off and the Hyperdrive10 should be drawing around 40-50 mA. Later when the motor is stopped this could be 100-300 mA depending on the **D16_STOPPED_PWR** setting. If all is well tap the SETTINGS button and confirm the settings on that screen. Kremford suggest using all default values to start with. Make sure the STEPS PER REV match your motor. Tap the STAND ALONE button to ensure the **DB_CONTROLLER_MODE** register is cleared. Tap the HOME button to return to the home screen.

Assuming you have followed the layouts of Figure 16 and Figure 17 on your HMI, enter 200 into the SPS numeric, tap the ON/OFF button, and the motor should start. Because the current Gearing is 1.0 you should see the 200 repeated in the RPM numeric. Note that 200 steps/sec for a 200 steps/rev motor is 60 RPM.

Set accel and decel rates of 2000 sps^2. With the motor running, tap the CW/CCW button. Note how the motor does an ordered deceleration to stopped and then accelerates back up to speed in the opposite direction, all under control.

You should experiment with the acceleration settings and with different torque loadings for your motor as well as various gearing values.

10.3 An HMI Configuration for External Control

Many motor control situations require external control for starting and stopping the motor. The next set of HMI screens show the typical control requirements for this situation along with a number of debugging objects that show what else is happening during the motor cycles.

The basic requirement for this example was to implement the several control schemes described in *Starting, Stopping and Direction Control Using External Events* on page 7-22. This demonstration HMI consists of two screens with the home screen shown in Figure 18. While rather busy it collects all input, motor speed and pre- and post-delays in the one screen with a set of HMI objects that would allow for setting up a test configuration. The MOTOR ON button does not start the motor but arms it ready.

The INPUT-1 ARM button enables Input 1. The toggle switch below sets the IN-1 transition, either low or high going. The event can immediately start the motor or after a delay set in the DELAY numeric (0.000001 second increments). The time the motor runs for can be switch selected to be either milliseconds (0.001 second increments) or motor steps. The speed can be set in either steps/sec or nominal RPM after setting the Gearing.

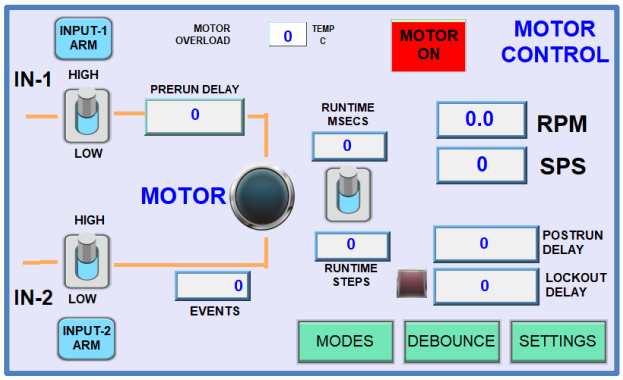

Figure 18. Test screen for examining the external control capabilities.

The HMI objects in this screen are:

OBJECT FUNCTION	REGISTER NAME	FORMAT	TYPE	ADDRESS	READ ONLY
INPUT-1 ARM	BIT_Inx_ARMED	TOGGLE SWITCH	4x_Bit	301	
1 Hi or Lo Selector	BIT_INx_FALLING	TOGGLE SWITCH	4x_Bit	300	
PRERUN DELAY	D32_PRERUN_TIME	32-Bit Unsigned	4x_MAX2W	305	
INPUT-2 ARM	BIT_Inx_ARMED	TOGGLE SWITCH	4x-Bit	401	
2 Hi or Lo Selector	BIT_Inx_FALLING	TOGGLE SWITCH	4x_Bit	400	
Motor Running Lamp	BIT_MOTOR_RUNNING	BIT LAMP	3x-Bit	102	YES
Motor Overload[1]	BIT_MOTOR_OVERLOAD	BIT LAMP	3x_Bit	813	YES
Temperature	D16_CHIP_TEMPERATURE	16-Bit Unsigned	4x_MAX1W	210	YES
RUNTIME MSECS	D32_RUNTIME_MSECS	32-Bit Unsigned	4x_MAX2W	301	
Steps/Time Selector	BIT_RUN_STEPS	TOGGLE SWITCH	4x_Bit	711	
RUNTIME STEPS	D32_RUNTIME_STEPS	32-Bit Unsigned	4x_MAX2W	303	
MOTOR ON/OFF	BIT_DEVICE_ON	TOGGLE SWITCH	4x_Bit	100	
RPM	DF_ARBITRARY_SPEED	32-Bit Float	4x_MAX2W	409	
Steps/Second	D16_SPEED_SPS	16-Bit Unsigned	4x_MAX1W	215	
POSTRUN DELAY	D32_POSTRUN_TIME	32-Bit Unsigned	4x_MAX2W	307	
LOCKOUT DELAY	D32_LOCKOUT_TIME	32-Bit Unsigned	4x_MAX2W	309	
Locked Out Lamp	BIT_LOCKED_OUT	BIT LAMP	3x-Bit	106	YES
EVENTS	D32_EVENT_COUNT	32-bit Unsigned	4x_MAX2W		

1. 2. Note the overload bit 13 is active low, so this lamp must be inverted.

Note: You cannot change input parameters while the motor is running.

This test configuration needs to set some bits in the **DB_CONTROLLER_MODE** register. While the next screen is the standard Hyperdrive10 screen we have seen earlier that has toggle buttons for all mode bits, only the required subset would be necessary in a screen built specifically for a similar real project.

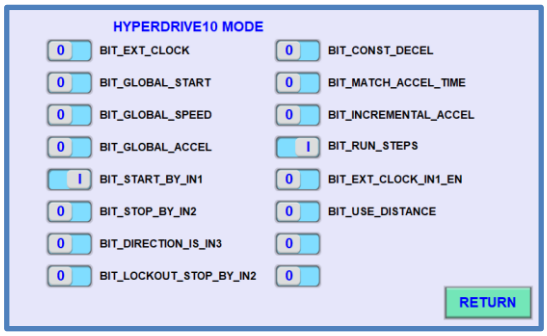

Figure 19. A screen providing for setting of any bit in the Hyperdrive10 mode control register DB_CONTROLLER_MODE.

Each one of the graphics in this screen is a toggle switch configured to set or clear the respective bit in the **DB_CONTROLLER_MODE** register. Table 14 shows the object descriptions for the first three objects – they are all toggle switches with addresses from 700 to 715.

OBJECT FUNCTION	FORMAT	TYPE	ADDRESS
BIT_EXT_CLOCK	TOGGLE SWITCH	4x_Bit	700
BIT_GLOBAL_START	TOGGLE SWITCH	4x_Bit	701
BIT_GLOBAL_SPEED	TOGGLE SWITCH	4x_bit	702
etc			

Table 14. Showing the object descriptions for the first three entries in the screen shown in Figure 19.

Like the previous example shown in Figure 17, this example requires a screen for setting the various motor parameters. How you lay out these screens is a personal preference, but Figure 20 shows the Settings screen for this example.

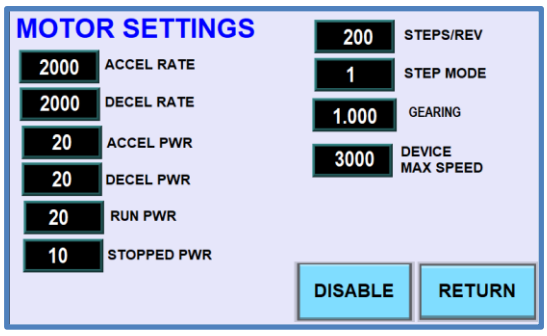

Figure 20. The Motor Settings screen.

10.3.1 Start and Run for Steps

Note that for the motor to run in this configuration, there must be an Input 1 transition, the INPUT-1 ARM button must be on, there must be a steps entry if in steps mode, there must be an SPS entry, and the **BIT_START_BY_IN1** bit must be set in the **DB_CONTROLLER_MODE** register.

If you are using some sort of mechanical switch on Input 1 (and later on Input 2), you will need to set a debounce delay for each switch. Almost all mechanical switches will exhibit contact bounce and while it can be over in just a few milliseconds, the electronics in the Hyperdrive10 will register each transition as separate signals, probably indicating a sequence you were not expecting.

The Hyperdrive10 provides for debouncing on the high-speed interrupt inputs 1 and 2, and the screen shown in Figure 21 is an example of how this is implemented.

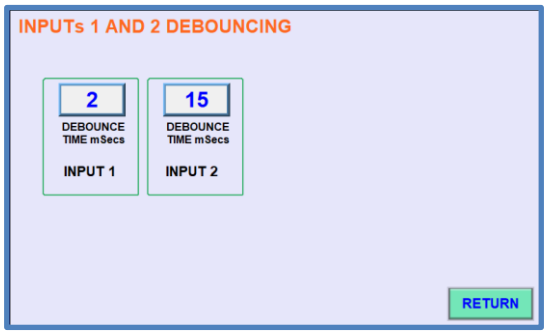

Figure 21. Setting contact debounce times for inputs 1 and 2.

The two standard 16-bit numeric objects in this screen are described in Table 15.

OBJECT FUNCTION	FORMAT	TYPE	ADDRESS
D16_INPUT_1_DEBOUNCE	16-BIT UNSIGNED	4x_MAX1W	201
D16_INPUT_2_DEBOUNCE	16-BIT UNSIGNED	4x_MAX1W	202

Table 15. The two numeric object definitions in Figure 21.

Here is a suggested complete setup and test procedure using these screens to test external motor control.

1. Connect an electronic on/off sensor to Input 1. It should switch a typical output of either PNP or NPN with a 0-24V range. Use the suggested NPN connections of Figure 10. An alternative would be a SPST switch with a suitable debounce setting.
2. Connect a SPST switch to Input 3, again using the switch wiring described in Figure 10.
3. Power up the test configuration.
4. Tap the MODES button and configure the **DB_CONTROLLER_MODE** register for this project with both the **BIT_START_BY_IN1** and **BIT_RUN_STEPS** bits set as shown in Figure 19.
5. Return to the home screen and enter 200 into the SPS numeric. The default gearing is 1.0, so the RPM numeric should also show 200.
6. Move the Step/Time switch to RUNTIME STEPS and enter 3200 into the numeric.
7. Tap the INPUT-1 ARM button to enable Input 1.
8. Tap the On/Off button.

The screen should now look something like Figure 22.

If the motor is clear, produce a momentary pulse on Input 1. The motor should run for a complete revolution and the MOTOR lamp should light while the motor runs. If all is well, increase the steps to 6,400 steps and again pulse Input 1. The motor should rotate 720°. Enter 32,000 steps and again pulse Input 1. The motor should rotate ten times and return to exactly the same position.

Increase the SPS setting to 1000. The acceleration and deceleration periods should now be obvious. Try increasing both accelerations to 10,000 and note the difference.

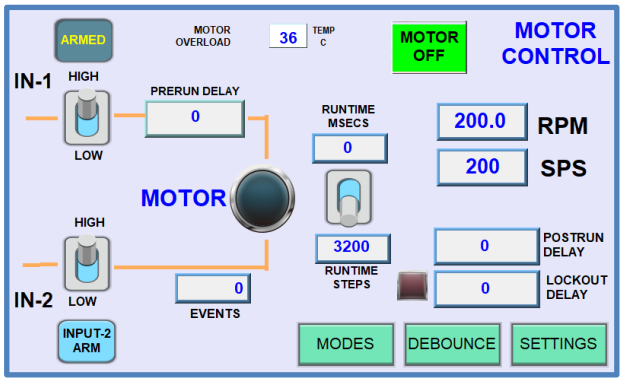

Figure 22. Test screen configured to run the motor by a transition on Input 1.

Go to the Mode screen of Figure 19 and tap the **BIT_DIRECTION_IS_IN3** button. Now you can set the direction the motor will run by changing the Input 3 level before setting the Input 1 state transition.

Try switching to RUNTIME MSECS mode. With a RUNTIME MSECS setting of zero the motor will run forever or until the MOTOR OFF button is pressed. Set 5000 in the numeric and switch Input 1 - the motor should run for 5 seconds.

Note that in this example the **D16_STEP_MODE_INDEX** was left at the default 4. This is the 1/16 microstep mode setting as shown in Table 16, and can be considered as a x16 multiplier of the basic motor's Steps/rev. The basic motor 200 steps in one revolution becomes 3200 when using this index. For a 200 Steps/rev motor, Column 4 shows the required steps for one revolution linked to the **D16_STEP_MODE_INDEX** of column 1.

STEP MODE INDEX	STEP MODE	STEPS PER STEP	STEPS/REV
0	Full-step	1	200
1	Half-step	2	400
2	1/4 microstep	4	800
3	1/8 microstep	8	1600
4	1/16 microstep	16	3200

Table 16. The relationship between the motor steps/revolution and the microstep selection in D16_STEP_MODE_INDEX.

So for example, with a **D16_STEP_MODE_INDEX** setting of 1 (half-step), and a requirement to spin the motor for five full rotations, the value entered into the RUNTIME STEPS numeric would be 2,000 (5 * 400).

10.3.2 Start and Stop by Inputs

We can use the same set of screens to simulate both starting and stopping by two different external level changes.

> **Note**: You cannot change input parameters while the motor is running.

Here are the steps to try.

1. Connect another electronic on/off sensor to Input 2. It should switch a typical output of either PNP or NPN with a 0-24V range. Use the suggested NPN connections of Figure 10.
2. Tap the INPUT-2 ARM button to arm Input 2. Both inputs should now be armed.
3. Tap the **BIT_STOP_BY_IN2** button in the Mode screen. Now the **BIT_START_BY_IN1**, **BIT_STOP_BY_IN2, BIT_DIRECTION_BY_IN3** and the **BIT_RUN_STEPS** buttons should be on.
4. Move the Step/Time switch to RUNTIME MSECS and enter 0 into the numeric so without an IN2 event the motor will run forever.
5. Tap the Start/Stop button to arm the motor.
6. Now a transition on Input 1 will start the motor and a subsequent transition on Input 2 will stop the motor. Note that the stop signal on Input 2 just initiates the stop – the motor must still decelerate to physically stopped at whatever the current **D16_DECEL_SPS2** rate is.
7. Changing the switch state on Input 3 between runs will change the direction next time the motor starts.
8. Use the Start/Stop button to stop and start the motor, and while it is stopped, switch Input 2 between arm and disarmed. Watch what happens when the motor is started again. Watch what happens when the Input 2 trip state is changed from high to low. In each case you should see how the change to the state of Input 2 results in the motor being stopped or not by the Input 2 transition.
9. Explore what happens by changing the speed and acceleration rates.
10. If you can independently measure RPM, try computing different Gearing values for hypothetical gearboxes and linear drive systems.
11. Try entering a delay time in microseconds (note *micro*seconds) into the PRERUN delay numeric and noting how the motor starts that many microseconds after the Input 1 on transition. For example entering 1000000 will produce a 1 second delay.
12. Enter a delay in the POSTRUN numeric and see how that delays the motor turning off after the Input 2 off transition.

10.3.3 Start and Stop with Lockout

The Lockout function is optional and can be used to prevent spurious Input 2 transitions that may occur before the real transition arrives for stopping the motor. If the system is setup to start via Input 1 and stop via Input 2, the Lockout allows setting a blanking time during which Input 2 transitions are ignored.

If the **BIT_LOCKOUT_STOP_BY_IN2** bit is set and the **D32_LOCKOUT_TIME** counter is non-zero, the lockout starts either on the Input 2 transition, or when the **D32_PRERUN_TIME** counter concludes.

> **Note**: The POSTRUN counter extends the point in time when the motor stops until some exact time after the occurrence of the Input 2 event. The LOCKOUT counter however, starts when the motor starts and runs to conclusion regardless. If the POSTRUN count is increased to extend the motor runtime, the LOCKOUT count may need a matching increase to ensure the lockout effect also extends to match the POSTRUN extension.

10.3.4 Stop with Constant Deceleration

A typical motor drive situation is where the motor is driving some sort of web via a pressure roller. Some event is detected in the web and the motor must immediately stop and the amount the web moves after the stop signal must always be the same. An example is the label drive in a labelling machine – a sensor detects the gap between labels which immediately commands the motor to stop. The label web must come to rest so a specified length of label (the 'flag') protrudes past the peel plate.

However, this can cause a problem if the machine can run at different speeds. For a constant and linear deceleration rate, as the machine speed is increased the *time* it will take to stop will also increase. Consequently, the web being transported after the sop signal will lengthen as the speed increases, possibly causing a problem for the machine function.

This problem is usually contained by choosing high acceleration rates for the drive motor. However, even choosing very high rates like 20,000 steps/sec^2, in a typical machine the difference in stopping distance between rates of, say, 20 M/m and 5 M/m can be nearly 20 mm. Obviously some other method is required that can hold the difference to something around ±1 mm.

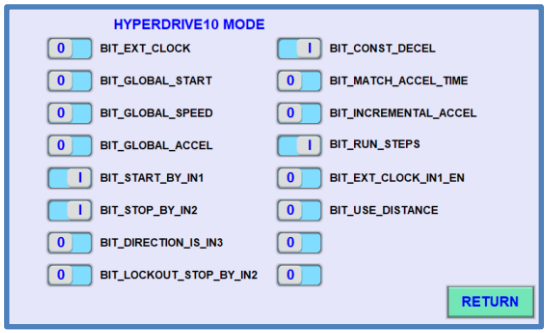

Figure 23. Settings for the DB_CONTROLLER_MODE register when using external start stop control and a constant deceleration rate.

The Hyperdrive10 uses the simple method of setting the motor deceleration time to be always as though it is decelerating from the rated device highest speed for the motor (see the **DEVICE MAX SPEED** numeric in Figure 20). Other machine settings can be used to set the 'flag' length now the operator is assured of a constant stopping distance. Figure 23 shows the basic mode settings for this configuration.

Note that this mode is only meant to apply to very high deceleration rates, typically greater than 15,000 sps^2. Always use the following test where speeds are in Steps/sec and rates in Steps/sec^2 to check your situation.

$$if\ \left[0.65536 - \frac{MaxSpeed - CurrentSpeed}{DecelRate} > 0\right]$$

If your variables give a negative result the motor will just stop as normal without the constant deceleration feature. The HMI can use the **BIT_TOO_SLOW_DECEL_RATE** bit in the **DB_MACHINE_STATE** register with an indicator such as a bit lamp to monitor this test after changes to any of these parameters. It is set for a negative result.

10.4 The Switched Output Functions in this Configuration

The switched Outputs 1 and 2 are controlled by Inputs 3 and 4 except when the Hyperdrive10 is in **BIT_EXT_CLOCK** mode.

However, Input 3 can be switched away from this task to controlling the motor direction via the **BIT_DIRECTION_IS_IN3** mode bit, so this needs to be kept in mind. Apart from in external clock mode, the Input 4 Output 2 combination is always active.

10.5 A Master/Slave HMI Configuration

Up to seven Hyperdrive10 controllers can be configured to operate together controlled by a single HMI over a common RS485 buss. The HMI would include control screens for each Hyperdrive10 along with a separate screen that sets the global parameters. The global parameters are by convention considered those of the Hyperdrive10 at address 1, and the motor it drives is assumed to have the highest inertia load of the machine. This means when required, other motors can have their accelerations slaved to that part of the machine, a significant advantage where the machine assembly must remain synchronised during accel and decel events.

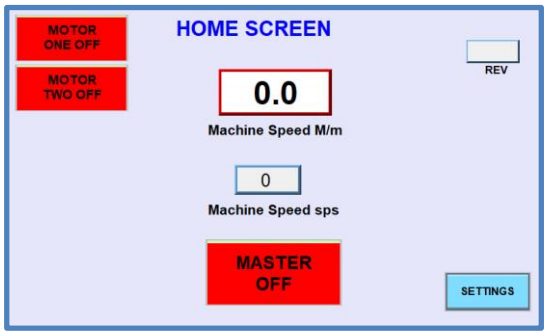

Figure 24. The home screen of the example master/slave Hyperdrive10 multi-motor configuration HMI.

Figure 24 shows the example home screen for a two-motor system. It simulates a machine where both motors drive their gearing systems that translate to linear outputs that at some point in their cycle must synchronise with each other. The motor one system does the heavy lifting while the motor two system is integrated in some way – perhaps it attaches something to a product travelling on the motor one system or drives a mechanism that must synchronously engage with the motor one system. It is assumed this two-motor system is part of some larger machine that requires operational speed changes from time to time, requiring that synchronisation within this two-motor system is speed independent, not only for speed but also during acceleration events.

The home screen (Figure 24) includes separate on/off buttons for each motor and a common on/off button to control them both (the **BIT_MASTER_ON** bit in the **GB_MASTER_STATE** register). The central numeric displays the global speed (here in Metres/min) and another numeric to display the resulting global sps for reference. The SETTINGS button gives access to the despatch sub-screen shown in Figure 25 which provides access to the settings and modes screens for the two motors along with access to the global setup screen.

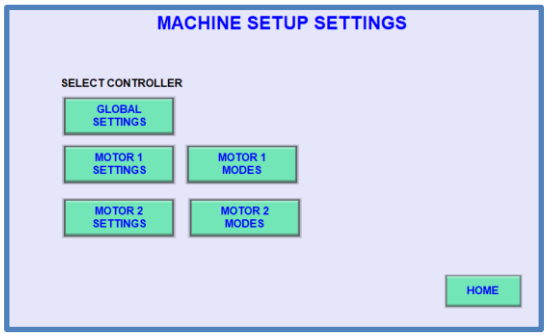

Figure 25. The despatch screen to the global and two motor setup screens.

All these six buttons in Figure 25 are window access buttons (called Function keys in EZWarePlus) and pressing them brings up their associated screen. On this screen, pressing the GLOBAL SETTINGS button will bring up Figure 26. This has numerics for the primary global settings of gearing and accelerations along with a setting for the overall machine maximum speed, typically specifying some physical limit. Note the latter is the motor speed limit in Steps/sec.

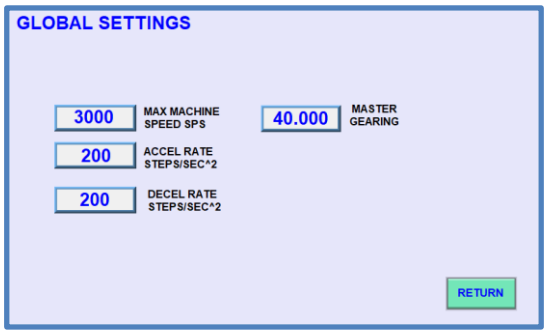

Figure 26. The global setup screen.

The objects listed in Table 17 from the global settings screen of Figure 26 all have the device address 99, the global address. The MASTER_GEARING object for example has the complete address 99#901. This means at any one time both Hyperdrive10s will have copies of those values in their matching global address registers.

OBJECT FUNCTION	REGISTER NAME	FORMAT	TYPE	REGISTER ADDRESS
MAX MACHINE SPEED	G16_MAX_MACHINE_SPS	16-bit Unsigned	4x_MAX1W	704
MASTER GEARING	GF_MASTER_GEARING	32-Bit Float	4x_MAX2W	901
ACCEL RATE	G16_ACCEL_SPS2	16-bit Unsigned	4x_MAX1W	701
DECEL RATE	G16_DECEL_SPS2	16-bit Unsigned	4x_MAX1W	702

Table 17. The objects in the global setup screen.

Note: For the Hyperdrive10 device at address 1, whenever the **GF_MASTER_GEARING** register at 99#901 is set, the software automatically copies the value to its **DF_DRIVE_GEARING** register at 1#407.

Figure 27 is the setup screen for motor 1, the master controller.

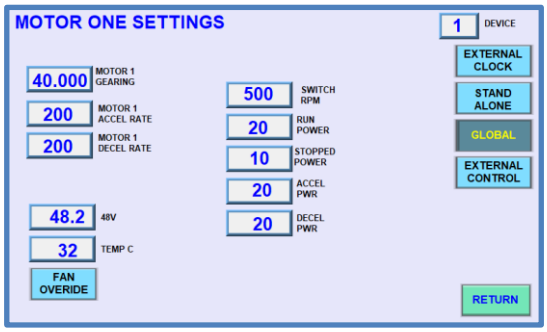

Figure 27. The device 1 (global) setup screen.

The four buttons down the right-hand side are elaborations of the STAND ALONE button on the right-hand side of Figure 17 and set bits in the **DB_MACHINE_STATE** register. These are radio buttons – turning one on will turn any other off. They set and clear bits in the **DB_CONTROLLER_MODE** register as shown in Table 18. Bits not listed as being set are cleared.

BUTTON	BIT	BITS SET IN DB_CONTROLLER_MODE REGISTER
EXTERNAL CLOCK	BIT_1	BIT_EXTERNAL_CLOCK
STAND_ALONE	BIT_2	BIT_STAND_ALONE
GLOBAL CONTROL	BIT_3	BIT_GLOBAL_START, BIT_GLOBAL_SPEED, BIT_GLOBAL_ACCEL
EXTERNAL CONTROL	BIT_4	BIT_START_BY_IN1, BIT_STOP_BY_IN2, BIT_DIRECTION_IS_IN3, BIT_RUN_STEPS

Table 18. Bits set in the DB_CONTROLLER_MODE register by single bits in the DB_MACHINE_STATE register.

These four bits in **DB_MACHINE_STATE** are just setup shortcuts. It is up to the programmer whether these are used rather than explicitly setting the necessary bits in the **DB_CONTROLLER_MODE** register. As we shall see, most master/slave setups will often require the setting of additional bits.

10.5.1 Setting Matching Acceleration Rates

We have seen the MOTOR MODES screen of Figure 28 before. In this case it is the screen displayed when the MOTOR 1 MODES button is pressed in Figure 25. While the bits set in the left-hand column are the result of pressing the GLOBAL button in Figure 27, the **BIT_MATCH_ACCEL_TIME** bit set in the right-hand column has been set in this screen as we wish to test this function.

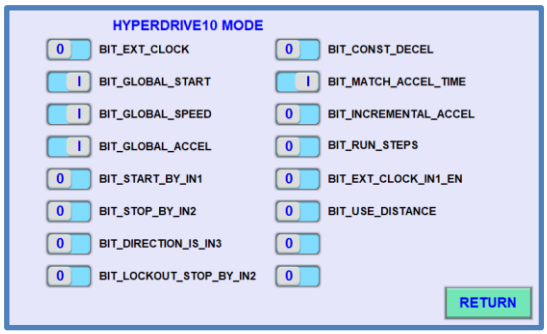

Figure 28. The motor 1 mode screen setup for global control.

Figure 29 shows the motor two setup screen. It is the same as motor one except for the gearing which in this case has the value 30. Because in this master/slave setup the **BIT_GLOBAL_SPEED** bit has been

set for both Hyperdrive10s, both will drive their respective transmission outputs at the master speed via their respective gearing ratios. In this case the only slave is motor two and it is now slaved to motor one for speed. Because motor one has a gearing of 40 and motor two a gearing of 30, motor two need only rotate at ¾ of the speed of motor one to have the same output speed.

These gearing values of 40 and 30 are for illustration. Normally they would be the result of a machine analysis for each motor's transmission as described in *Some Examples of Setting the Hyperdrive10 Gearing* on page 12-59.

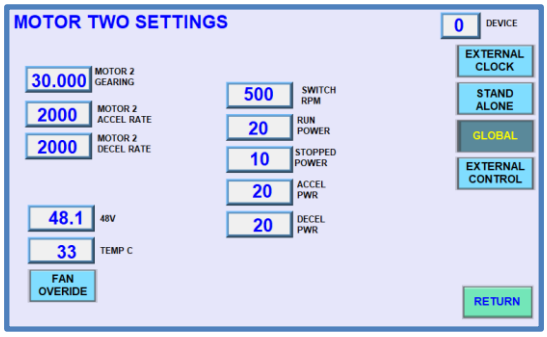

Figure 29. The device 2 setup screen.

Note that in the motor two settings screen of Figure 29 the GLOBAL button is also pressed and pressing the MOTOR 2 MODES button in Figure 25 will show the same screen as Figure 28, and the BIT_MATCH_ACCEL_TIME bit will also need to be set manually.

For the screen in Figure 28, what exactly has been setup?

- Both motors have their **BIT_GLOBAL_START** bits set, so they will not start until both their **BIT_DEVICE_ON** and the global **BIT_MASTER_ON** bits are set.
- Both have their **BIT_GLOBAL_SPEED** bits set, so both motors will use the **GF_ARBITRARY_SPEED** value (set in the Home screen of Figure 24) along with their own gearing (set respectively in Figure 27 and Figure 29) to make their output speeds match the global speed.
- Both have their **BIT_GLOBAL_ACCEL** bits set, so the intent is for the slave motors (just motor two in this case) to match the global accelerations set in Figure 26 (which are, of course, the motor one settings).

The effect of setting the **BIT_MATCH_ACCEL_TIME** has been described earlier in *Motor Acceleration Matching* on page 7-24 and in **Error! Reference source not found.** It forces that particular slave to match the global **accelerations**. So, no matter what the various gearings (within reason) all motors will take *exactly the same time* to start, stop, or change speed.

You should experiment with different speeds, accelerations, and gearings.

10.6 Using Distance to Specify Time

While using distance to specify time is available for a stand-alone situation, it would mostly be used in a master/slave situation where several Hyperdrive10s are working together. This example combines the master/slave configured in *A Master/Slave HMI Configuration* on page 10-46 and the input control configured in *An HMI Configuration for External Control* on page 10-39 to demonstrate

a two-motor machine where device one (the master) drives a conveyor and device two drives a stop start mechanism that must be synchronized to the conveyor speed.

The example machine uses two Hyperdrive10s in a master slave configuration very similar to the previous case. Device one is as usual the master and drives a standard product conveyor. Device two drives a mechanism that operates only for a moment as the product passes part of its mechanism. It is armed by a sensor registering that the product is approaching on the conveyor and has reached a specific position. The device two motor is started an exact delay time later when the product has moved on to be in the correct position for the device two mechanism. The delay time elapsing starts the motor and it is stopped by sensing some event in its mechanism, signalling the job done.

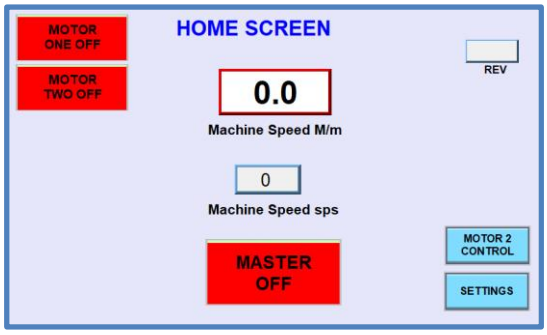

Figure 30. The home screen for the distance example.

Of particular concern for this machine is the conveyor can be driven at various speeds, so the time between sensing the product and starting the motor will be variable. However, the distance is fixed and so we can use the Hyperdrive10 **BIT_USE_DISTANCE** bit to set the actual distance measurement and so make the setting independent of speed. With this specification we can put together an HMI starting with the home screen shown in Figure 30.

The drive one motor is a standard 200 Steps/rev motor driving a 2:1 chain drive and then the conveyor drive wheel which has an effective diameter of 152 mm. The machine speed is measured in Metres/minute and we know from Equation (4) (see page 12-60) that the master gearing in this case is:

$$Gearing = \frac{1000 * 200 * 2}{60 * \pi * 152} = 13.961$$

The slave also uses a 200 Steps/rev motor that direct drives a 48 mm diameter roller that dispenses a label onto the product. The motor two mechanism is setup so the label is in the correct position to be applied to the product as it passes the beforementioned position. Once the motor two starts it runs until another sensor indicates the label is applied and the next label is in position. From Equation **Error! Reference source not found.** (see page **Error! Bookmark not defined.**) we know the gearing for this roller case is:

$$Gearing = \frac{1000 * 200}{60 * \pi * 48} = 22.105$$

The screens for motor one are similar to the earlier master/slave example. The master Hyperdrive10 is configured as a standard master/slave configuration in Figure 31 and Figure 32.

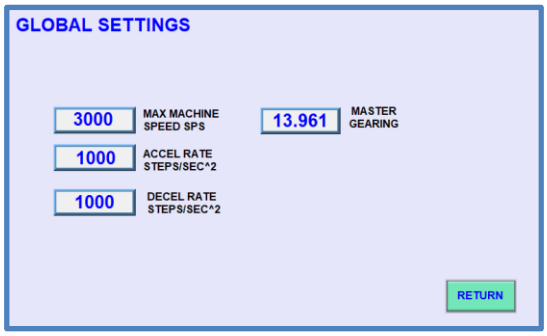

Figure 31. The global setting screen.

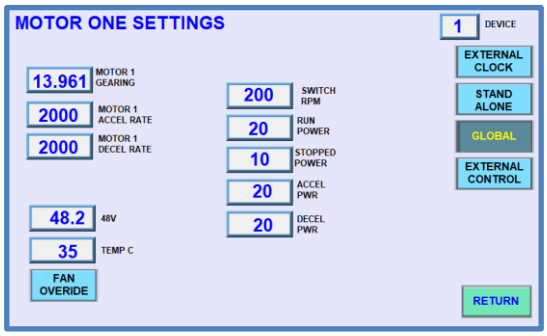

Figure 32. The motor one settings screen.

The motor two is also configured as a standard slave (Figure 33) except for the two quite high acceleration rates.

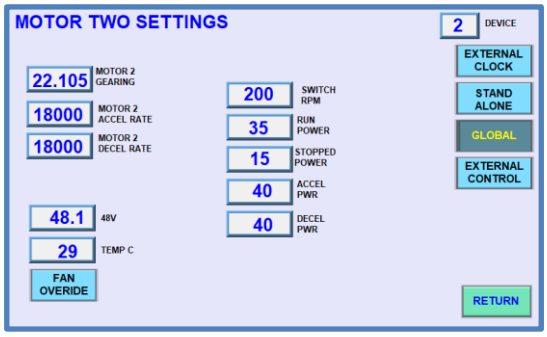

Figure 33. The motor two settings screen.

The **DB_CONTROLLER_MODE** settings are a little different for both Hyperdrives. The master settings in Figure 34 are configured for global start, speed and acceleration settings (although the latter is not actually necessary).

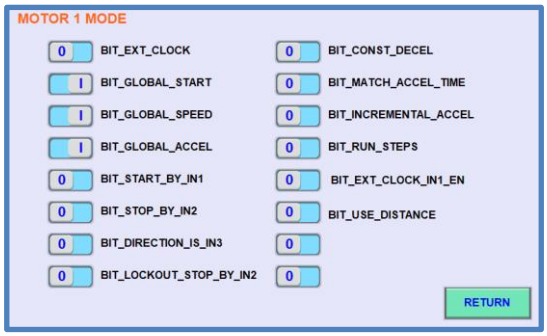

Figure 34. The motor one controller mode settings.

Motor two however has a considerably different configuration. We need overall master on/off and speed control but also input controlled start and stop. The direction is assumed to be preset. We need to be able to specify the distance between where the product is sensed and where the label is applied. There is also the possibility that because of the machine setup, the label may need to travel some distance after the stop signal is sensed. The **DB_CONTROLLER_MODE** register settings for motor two implementing these requirements is shown in Figure 35.

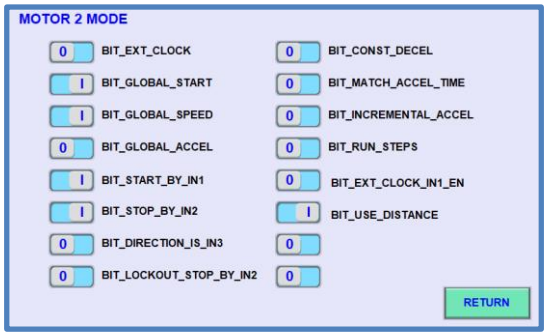

Figure 35. The motor two controller mode setting screen.

The MOTOR 2 CONTROL button in Figure 30 displays the screen shown in Figure 36. This looks very similar to the earlier external control screen in Figure 18, and has much the same functionality except we are now dealing in *distances* not time. The PRERUN DISTANCE and POSTRUN DISTANCE numerics addresses the **D16_PRERUN_DISTANCE** and **D16_POSTRUN_DISTANCE** registers respectively. Input 1 is the product sensor and the PRERUN DISTANCE numeric specifies the distance the product will travel on the conveyor from the Input 1 event before the motor 2 must start. In a real machine this could be a simple manual measurement of the setup distance using a standard tape measure.

The sensor connected to Input 2 indicates when the label feed must be halted after the current label has been fed and the next label is now in position. Note that this need not be the exact mechanical position, just an exact and relative position that can be conveniently sensed. Where this occurs the postrun distance can be used to extend the motor runtime by a precise distance past this sensed position, so the label is moved into the correct position.

Once the setup screens have been configured, to begin we enter an arbitrary speed such as 24 M/m into the Machine Speed M/m numeric in the home screen. The corresponding speed of 335 Steps/sec should appear in the Machine Speed sps numeric (24 * 13.961 = 335).

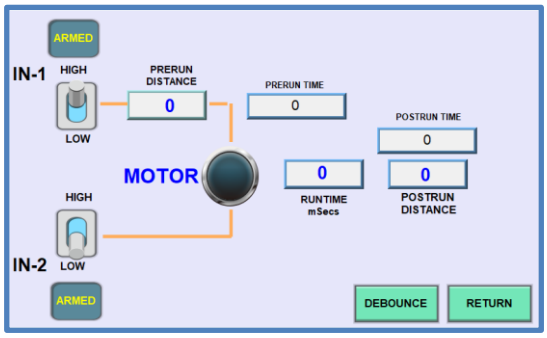

Figure 36. The motor 2 control screen.

We are simulating both inputs by providing the two sensor signals from a function generator instrument, and as shown in Figure 36, the active edges are high-going for Input 1 and low-going for Input 2. For now we leave the PRERUN DISTANCE and POSTRUN DISTANCE numerics at zero. Recall from Figure 35 the motor runtime has been set in mSecs (the **BIT_RUN_STEPS** bit is clear), and because a runtime of zero means the motor will run forever, we leave that numeric at zero. You would normally enter a value here that exceeds any possible feed time so motor 2 will only run for that time if the feed fails to be stopped. An alternative is to set the **BIT_RUN_STEPS** bit and enter the number of steps that provide the same safeguard.

Return to the home screen and start the motors. Motor 1 should start immediately. Motor 2 should start on the Input 1 high-going transition and stop on the Input 2 low-going transition. Figure 37 shows what is happening. It is an oscilloscope plot that records both inputs and the motor state with the major vertical timelines indicating 0.1 second intervals. Obviously not what we want but used here to show the situation when both prerun and postrun registers are zero – the motor starts on the Input 1 high-going transition and stops on the Input 2 low-going transition as programmed.

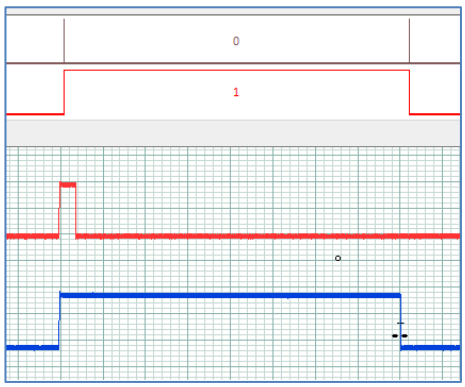

Figure 37. The top trace shows the input 1 and 2 transition events, the next trace is the motor running, the third trace is the product sensor high-going pulse, and the bottom trace is the low-going stop sensor.

For this case, internally the product sensor event is sensed by Input 1. The check for a prerun value is zero so the motor is started immediately. The motor runs until the Input 2 event occurs. The check for a postrun value is zero so the motor is stopped immediately.

We now stop the machine and enter both prerun and postrun values as shown in Figure 38.

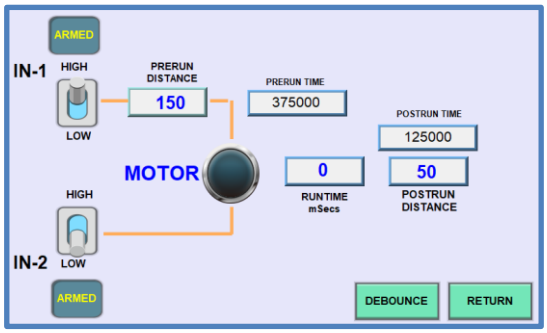

Figure 38. The Motor 2 control screen setup to demonstrate the use of pre- and post-run distances.

Because we are using M/m we can use the default **DF_DISTANCE_NUMERATOR** (see *How to Derive the Distance Measuring Multiplier* on Page 13-61).

A speed of 24 M/m is 0.4 Metres/sec or 400 mm/sec. It will therefore take $\frac{150}{400}$ = 0.375 secs to travel the 150 mm at that speed. The PRERUN TIME is this time in uSecs. The POSTRUN TIME likewise shows $\frac{50}{400}$ = 0.125 seconds. The default **DF_DISTANCE_NUMERATOR** value is 60,000 which the Hyperdrive10 uses to derive the register times in microseconds shown in Figure 38, based on the input M/m value. For example:

$$Time_{uSecs} = \frac{150 * 60000}{24} = 375000$$

If we start the machine again and check our plot of what is happening with motor 2, it will now look like Figure 39. The product sensor is again the high-going pulse in the third trace. This time when the **D32_PRERUN_TIME** register is checked is has a non-zero value so that is loaded into a counter which is shown starting in the top trace and begins counting down. When it reaches zero (trace one going low for the first time) the motor is started, indicated by trace two going high. Note the interval is 0.375 seconds and so the product would have travelled the required 150 mm at 24 M/m.

> **Note**: There is a product/label dependant minimum time between the label motor stopping after one product label application and the start signal for the next arriving. The time depends on the product and label dimensions, the label application time, and the mechanical capabilities of the machine. A typical recommended minimum would be 0.1 second.

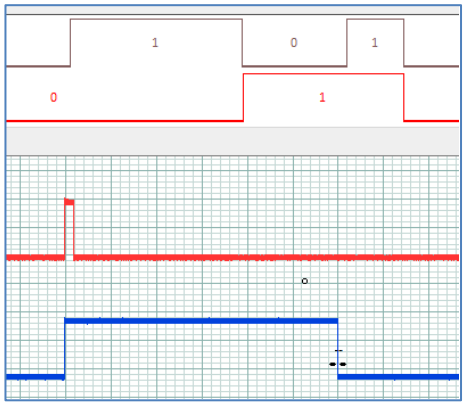

Figure 39. The motor 2 running with values for both the prerun and postrun registers.

The simulated label is fed until the inter-label gap or some similar event is detected – the bottom trace going low. This time the **D32_POSTRUN_TIME** register is also non-zero and is loaded into a counter and counted down. The top trace going high for the second time indicates this event. When it reaches zero trace one goes low and the motor is stopped as shown by trace two also going low.

In a configuration like this the requirement for a constant deceleration time is very important to ensure the motor's deceleration time is constant no matter what the speed. If we return to Figure 35 and set the **BIT_CONST_DECEL** bit, what does that do to the motor run profile? Figure 40 shows what happens.

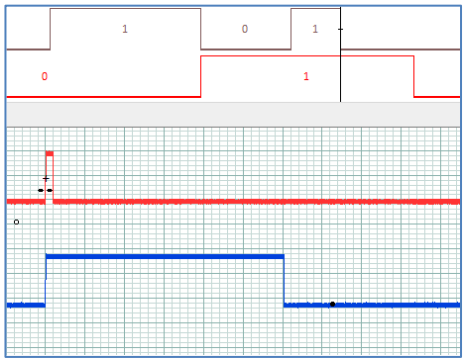

Figure 40. Showing the additional inverse speed dependant time added after the stop signal when constant deceleration is selected.

Now the postrun counter terminating initiates the inverse speed dependant deceleration sequence as can be seen by how it now takes time after this event before the motor stops (trace 2).

10.7 An Example Using Incremental Acceleration

We can use the same test setup as for *Using Distance to Specify Time* on page 10-49 to setup a test of incremental acceleration. It is important to note the difference between Incremental Acceleration and Acceleration Matching. The latter is used where two or more motors must have their acceleration times matched regardless of their gearing, so they remain synchronised throughout an acceleration event, all arriving at the new speed at exactly the same time. Incremental acceleration applies where a motor must run at the machine speed but is using much higher acceleration rates than the machine in general, is in a start/stop cyclical mode, and could potentially perform several cycles during the time it takes the whole machine to progress through an acceleration event. Without incremental acceleration this motor would be instantly at the machine destination speed and would not match the machine speed until the acceleration event concluded.

We will use the same machine settings as before only configured for a much slower machine acceleration time. Figure 41 shows the machine global setup and the very slow acceleration rates.

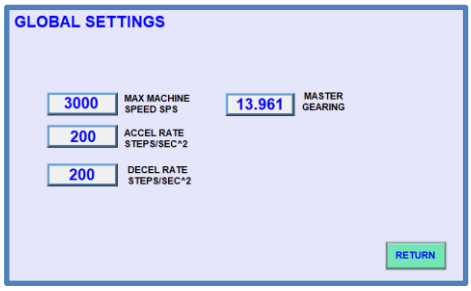

Figure 41. The basic machine parameters.

Motor 1 is setup as before in Figure 32 and Figure 34. However, motor 2 has different setting, modes, and gearing. We will add a cogged belt drive to the motor 2 roller giving a 2:1 gearbox ratio. Including this in the Equation (4) (see page 12-60) template the gearing for this roller case is:

$$Gearing = \frac{1000 * 200 * 2}{60 * \pi * 48} = 44.210$$

For this example the screens for the motor two settings are shown in Figure 42 and the modes in Figure 43.

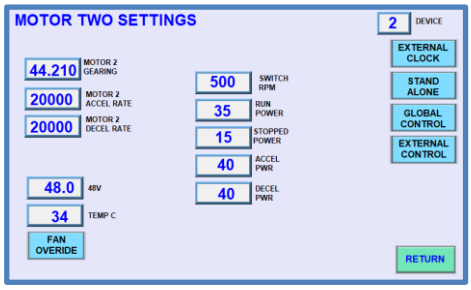

Figure 42. The motor 2 settings for ab Incremental Acceleration test.

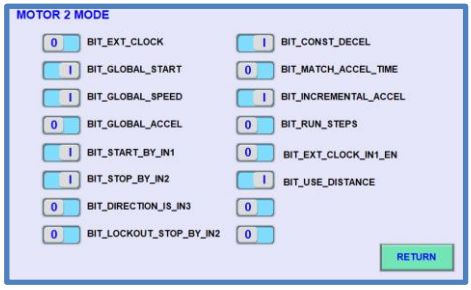

Figure 43. The motor 2 Mode settings.

Mostly the events demonstrated in Figure 11 happen too quickly for direct observation. Using such a low global acceleration rate here allows a test setup where with a fast enough motor 2 cycle time, a machine speed increase from, for example, 10 M/m to 50 M/m will include several motor 2 cycles and the incremental speed increase can be observed.

10.8 Setting Input Debounce

When the input signal to inputs 1 or 2 is a mechanical switch, it is likely the switch transition will include “bounce”. This means that the switch closing (and sometimes the opening) will actually open and close several times as the mechanical contacts bounce off each other. It will happen quite quickly

of course, usually within 2 to 10 mSecs, but to the microprocessor watching this event it will see each individual event.

As this will usually upset what is expected, the Hyperdrive10 includes debounce control. This works by detecting the very first switch transition but then ignoring any further transitions for a given time. When that delay concludes the switch is again examined to see if its current state reflects the initial state. Only if this is true, in other words the switch really did change state, will the Hyperdrive10 take the required action.

The debounce delays are set in the two **D16_INPUT_x_DEBOUNCE** registers and represent milliseconds. The minimum value is 2 and the largest 500. Typical values will be 2-10. Remember that this figure will delay the action required by the event.

The best way to characterise a debounce delay is to observe the switch transitions with an oscilloscope. You can then visually see the switch bounce time boundaries.

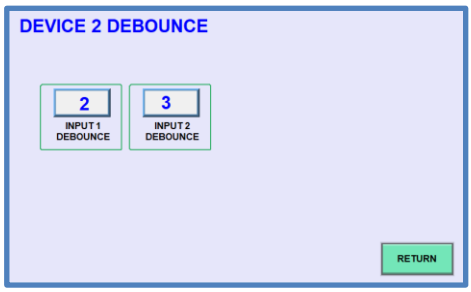

Figure 44. A typical switch debounce setting screen.

Figure 44 shows a typical screen where debounce values have been set. The numeric objects involved are shown in Table 19.

OBJECT FUNCTION	MODBUS NAME	FORMAT	TYPE	ADDRESS
INPUT 1 DEBOUNCE TIME	**D16_INPUT_1_DEBOUNCE**	16-Bit Unsigned	4x_MAX1W	201
INPUT 2 DEBOUNCE TIME	**D16_INPUT_2_DEBOUNCE**	16-Bit Unsigned	4x_MAX1W	202

Table 19. Typical HMI object debounce settings.

11 Trouble Shooting

11.1 Unexpected response to switch 1 and 2 inputs

This will nearly always be a result of switch bounce within the external switch. Check you have configured the particular input with a debounce value. Try 10 as an initial test value. The best method is to watch the switch input electrically with an oscilloscope.

12 Some Examples of Setting the Hyperdrive10 Gearing

The following shows the general method for developing the gearing value for a number of common speed measurement conversions using typical machine configurations. A gear train can be described by a ratio $\Re$ and a linear translation by an additional diameter ***D***.

12.1.1 A Linear Metres/Minute Example

Here we develop the gearing value for a machine where the motor drives a transmission mechanism that produces a linear output. Examples would be conveyor or roller drive systems. The mechanism could be a gearbox or geared drive belt system connected to a conveyor drive wheel or a roller drive of some sort.

If we want to work time in seconds and measurements in millimetres, we first need to convert the linear speed in metres per minute to millimetres per second:

$$Speed_{MMpS} = \frac{1000 * Speed_{MpM}}{60} \quad (1)$$

There are one or most often two steps in translating the linear speed to the motor's rotational speed. The first step converts a rotary motion to a linear motion via some sort of drive wheel. This could be a cogged wheel or a drive drum, but the important dimension is the effective outer diameter ***D*** of the wheel. Assuming there is no slippage, knowing the drive wheel circumference gives us the distance the conveyor will travel during one revolution of the wheel

$$Circumference = \pi D$$

So if the motor is driving the conveyor wheel directly and including the conversion from (1), the required motor speed in revolutions per second will be:

$$RotationsPerSecond_{MOTOR} = \frac{1000 * ConveyorSpeed_{MpM}}{60 * \pi * D} \quad (2)$$

As an example, if our conveyor drive wheel effective diameter is 127.324 mm and we need a conveyor speed of 24 metres per minute (MpM), the relation is:

$$MotorSpeed_{RpS} = \frac{1000 * 24}{60 * \pi * 127.324} = 1\ RpS$$

If there is a gearbox this will increase the motor speed by the gearbox ratio $\Re$, and adding this to equation (2) it becomes:

$$MotorSpeed_{RpS} = \frac{1000 * \Re * ConveyorSpeed_{MpM}}{60 * \pi * D} \quad (3)$$

Using our previous example and a gearbox ratio of 5, the motor revolutions per second will be:

$$MotorSpeed_{RpS} = \frac{1000 * 5 * 24}{60 * \pi * 127.324} = 5\ RpS$$

While this gives a motor speed in revolutions per second, the Hyperdrive10 needs a motor speed in steps per second (SpS), and so the final speed term must include the motor manufacturer's basic Steps per Revolution (SpR).

We can combine all these terms into a single value that expresses the motor ***Gearing***:

$$Gearing_{MpM} = \frac{1000 * StepsPerRev * \Re}{60\ \pi\ D} \quad (4)$$

For the example we have been working through, the figures for a standard 200 steps/rev motor are:

$$Gearing_{MpM} = \frac{1000 * 200 * 5}{60 * \pi * 127.324} = \quad 41.667$$

The Hyperdrive10 re-computes this term every time the conveyor drive wheel diameter or the gearbox ratio is changed. From then on the result is used whenever the user alters the required drive speed to produce the necessary motor speed in steps per second:

$$Speed_{SpS} = LinearSpeed_{MpM} * Gearing = 24 * 41.667 = 1000\ \ SpS$$

12.1.2 A Revolutions per Minute Example

Translating an input requirement in RPM to produce a rotation speed in steps per second is very similar. Knowing the speed, the motor's steps per revolution, and the gearbox ratio, the relationship is:

$$Speed_{SpS} = \frac{Speed_{RPM} * StepsPerRev * \Re}{60} \quad (5)$$

For a 60 RpM example and a 1:1 gearbox ratio:

$$Speed_{SpS} = \frac{60 * 200 * 1}{60} = 200\ SpS$$

The Gearing equation for the RpM case is very simple:

$$Gearing_{RpM} = \frac{StepsPerRev * \Re}{60} \quad (6)$$

So for a 200 steps per revolution motor and a 5:1 gearbox, the *Gearing* term is computed from:

$$Gearing_{RpM} = \frac{200 * 5}{60} = 16.667$$

Note The Gearing term is always used as a multiplier to the **DF_OUTPUT_SPEED** term to derive Steps/Second, and inversely to derive Steps/Sec from an arbitrary speed.

13 How to Derive the Distance Measuring Multiplier

When the **BIT_USE_DISTANCE** bit is used to allow for specifying the time it takes to travel a given distance, the user must provide a conversion factor that converts the machine's speed readout in a measurement like Metres/min or Feet/min (the **DF_ARBITRARY_SPEED** or **GF_ARBITRARY_SPEED**) into a time/distance. For the linear case, the Hyperdrive10 arbitrary speed is just the linear output speed, a measure of how much distance is travelled in a set time. The inverse of that is how much time for a given distance, and here we show how a constant can be developed to provide this time from the machine's speed and distance terms.

A standard machine speed is Metres/minute (MPM). A conversion is required that converts the user distance measurement into a time delay that can be measured at the given MPM speed.

To develop this we first convert the machine speed in MPM to the time it takes to travel 1 millimetre. The Hyperdrive10 internal clock ticks at a 1 microsecond rate, so the time it takes to travel 1 mm expressed in microseconds at the given MPM rate is:

$$OneMillimetreTravelTime_{uSecs} = \frac{60 * 10^6}{MPM * 1000} \tag{7}$$

We now simply multiply this by the user entered distance to give the required travel time in microseconds:

$$DelayTime_{uSecs} = \frac{Distance * 60000}{MPM} \tag{8}$$

> **Note**: The number 60,000 is the factory default number entered into the **DF_DISTANCE_NUMERATOR** register when Switch 4 is ON at boot time.

The Hyperdrive10 applies Equation (8) every time the distance or speed changes.

Another linear rate measurement in an imperial situation would be Feet/min (FPM). Like above, the conversion is first to some convenient measurement like inches and then to time. In this case the conversion is:

$$DelayTime_{uSecs} = \frac{Distance * 60 * 10^6}{FPM * 12}$$

$$= \frac{Distance * 5000000}{FPM}\ uSecs$$

An example is 20 F/m and a measured distance of 12 inches:

$$\frac{12 * 5000000}{20} = 3000000\ uSecs$$

Working it through again, the travel here is 12 inches every 3 seconds, or 1 foot every 3 seconds, or 20 feet in 60 seconds, or 20 F/m. So, the **DF_DISTANCE_NUMERATOR** number from F/m to uSec/inch is 3,000,000.

14 Using USB for Installing a Software Update

From time to time as a result of software improvements or a bug surfacing, Kremford will release software revisions for the Hyperdrive10. While upgrading your Hyperdrive10 to the latest software will need care, there is very little that can go wrong.

There is a link on the Hyperdrive10 printed circuit board (PCB) that needs to be positioned to configure the device for an upgrade.

You will also need a mini-USB lead and access to a Windows PC computer or lap-top containing the downloaded Hyperdrive10 file. The file will have a name like KR093-9-x-xx.bin where the “x-xx” will be a revision number.

1. Make sure the power is off.
2. Remove the four screws that hold the cover and fan in place and lay it aside. There should be enough fan cable that unplugging should not be required.
3. The PCB is now accessible. Find the link called LNK1. It is towards the middle of the board. It has a link mounted on just one pin.
4. Remove the link and replace it so it covers both LNK1 pins. You are ‘shorting’ LNK1.
5. Connect the mini-USB cable plug to the PCB mounted socket using access through the 12mm hole in the side of the Hypedrive10.
6. Plug the other end of the mini-USB lead into the computer.
7. Power up both the computer and the Hyperdrive10.
8. The Hyperdrive10 will appear as a new hard disc in Windows explorer with a name like CRPX_ENABLED. Navigate to that disc.
9. Only one file will be visible called **firmware.bin**. Delete this file.
10. Copy the new file to the Hyperdrive10 disc. When that completes the update is complete. While it will show the same file name at this point, after a power cycle it will have changed to the standard firmware.bin name.
11. Power down the PC and Hyperdrive10. Unplug the USB cable.

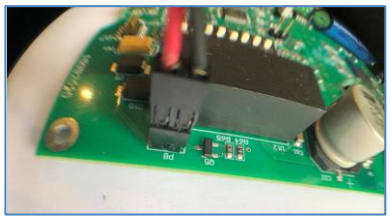

Figure 45. Observe the fan lead polarity.

12. Remove the link from LNK1 and replace it just sitting on one pin so it does not get lost.
13. Power up the Hyperdrive10 and check it starts correctly. If the HMI includes an ASCII display of the revision number (**DF_SOFTWARE_REVISION**), check it displays the new number.
14. Replace the cover. If the fan was unplugged, ensure the plug polarity is as per Figure 45.