SPORT232-ST User's Guide

Table of Contents

#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174

TERMS AND CONDITIONS

LEGAL

DISCLAIMER: *RMV ELECTRONICS INC. does not assume any liability arising from the application and/or use of* the *product/s described herein, nor does it convey any license. RMV ELECTRONICS INC. products are not authorized for use as components in medical, life support or military devices without written permission from RMV ELECTRONICS INC.*
The material endosed in this package may not be copied, reproduced or imitated in any way, shape or form without the written consent of RMV-ELECTRONICS INC. This limitation also applies to the firmware that the Integrated Circuits in this package might contain.

PRODUCT WARRANTY: RMV warrants that the Product shall conform to the RMV functional specifications for a period of one (1) year after shipment of the Product (the "Warranty Period"). During the Warranty Period, RMV will repair or replace (at its sole discretion) any faulty equipment or Software that fails to perform or meet the technical and functional product specifications, provided that the Customer must promptly notify RMV in writing of any warranty claim during such Warranty Period. This warranty does not apply to any product which has been:
a) subjected to misuse, neglect, accident, abuse or unusual hazard;
b) repaired or altered by someone who is not authorized by RMV to perform such repairs or alterations;
c) modified with use of replacement parts not furnished by RMV; or
d) not paid for in full.
If RMV determines a warranty claim is valid, RMV shall, at its option, (a) repair the Product at the Seller's location; or (b) accept the return of the product to RMV's British Columbia facility for repair or replacement, with all shipping and insurance charges to be borne by the Customer.

THE CUSTOMER AGREES THAT EXCEPT AS PROVIDED IN ABOVE PARAGRAPH THERE ARE NO OTHER WARRANTIES, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. RMV'S RESPONSIBILITY FOR WARRANTY CLAIMS IS LIMITED TO REPAIR AND REPLACEMENT AS SET FORTH IN THIS AGREEMENT.

RETURN. The Customer may only return the Product with the prior written consent of RMV, which consent may be unreasonably refused. If the Product is accepted for return, the Customer shall pay RMV its restocking charge then in effect.

SOFTWARE LICENSE. For the purpose of this Agreement, "Software" is defined as operating systems and/or firmware supplied by RMV contained on a compact flash disk, semi-conductor device or other memory device, or system memory including hard wired logic instructions and microcode, and documentation used to describe, maintain, and use the Software. The Customer is hereby granted a non-exclusive, fully paid perpetual license to use the Software, but only in conjunction with the hardware component of the Product purchased under this Agreement. This Agreement does not confer upon the Customer any title of ownership rights to such Software. The Customer agrees to take all necessary precautions to protect the confidential nature of the Software and to prevent its disclosure to unauthorized personnel.

Chapter 2 INTRODUCTION

INTRODUCTION

The SPORT232-ST is a Data Acquisition Control and One Axis Stepping Motor Ready Board. With eight analog independent channels for different voltage range, the SPORT232-SP is ready to connect to sensors such as thermocouples, pH probes, photo cells, strain gauges, and piezoelectric sensors for data acquisition applications. In addition, to the Data Acquisition capabilities 4 channels feature low off-set signal conditioning which allows connection to almost any analog device without having to undergo the time and expense of developing an amplifier for the external device.

The SPORT232-ST also offers 8 digitals I/O that are TTL compatible. Each line can source/sink up to 5mA.

The SPORT232-ST has been implemented with one industrial PC driven stepper motor axis controlled, on board by the RMV856-ETX Controller. Acceleration and deceleration modes are supported. A velocity-profiling feature is also available for complex motion parameters. Inputs for external control such as abort, limit and home allow easy interfacing with mechanical system. Analog and digital inputs give the user a variety of options for sensor reading and actuator control. Networking through the RS232 port gives the system the ability to control up to 16 stepper motors, by daisy-chaining up to 4 boards in a multi-drop configuration. A 32-bit DLL allows easy interfacing to Windows, Linux, programming languages, moreover a demo software example is also included.

FEATURES

- Control up to 16 stepper motors independently.
- Biphasic, monophasic and halfstep modes.
- Current mode driver (chopper) for unipolar or bipolar motors up to 2 Amp / 40 V; operating current is set trough software commands.
- Drives 4,5,6 and 8 wire steppers. Automatic Power Saving timer.
- 16-million step position register can be read on the fly.
- First Rate, Slew Rate and Acceleration parameters.
- Velocity profiling mode for complex motion schemes using internal 128 byte FIFO.
- Abort, Home and Limit, Hardware and Software Trigger inputs for external control.
- Multiple boards in a network may be controlled through a single PC RS232 port.
- Up to 8 digital I/O lines for general purpose.
- 8 channel AD Converter with 12 bits of resolution, 2.500 V precision reference, and 4 channels with signal conditioning for inputs of 0-2.5V, 0-5V, 0-10V range.
- 32-bit DLL interface for Windows Operative System and Embedded Linux.
- Full compatibility with LabView Interface.
- Extremely easy to interface with any software for PLC

Chapter 3 : PARTS AND LOCATIONS

BOARD DIAGRAM

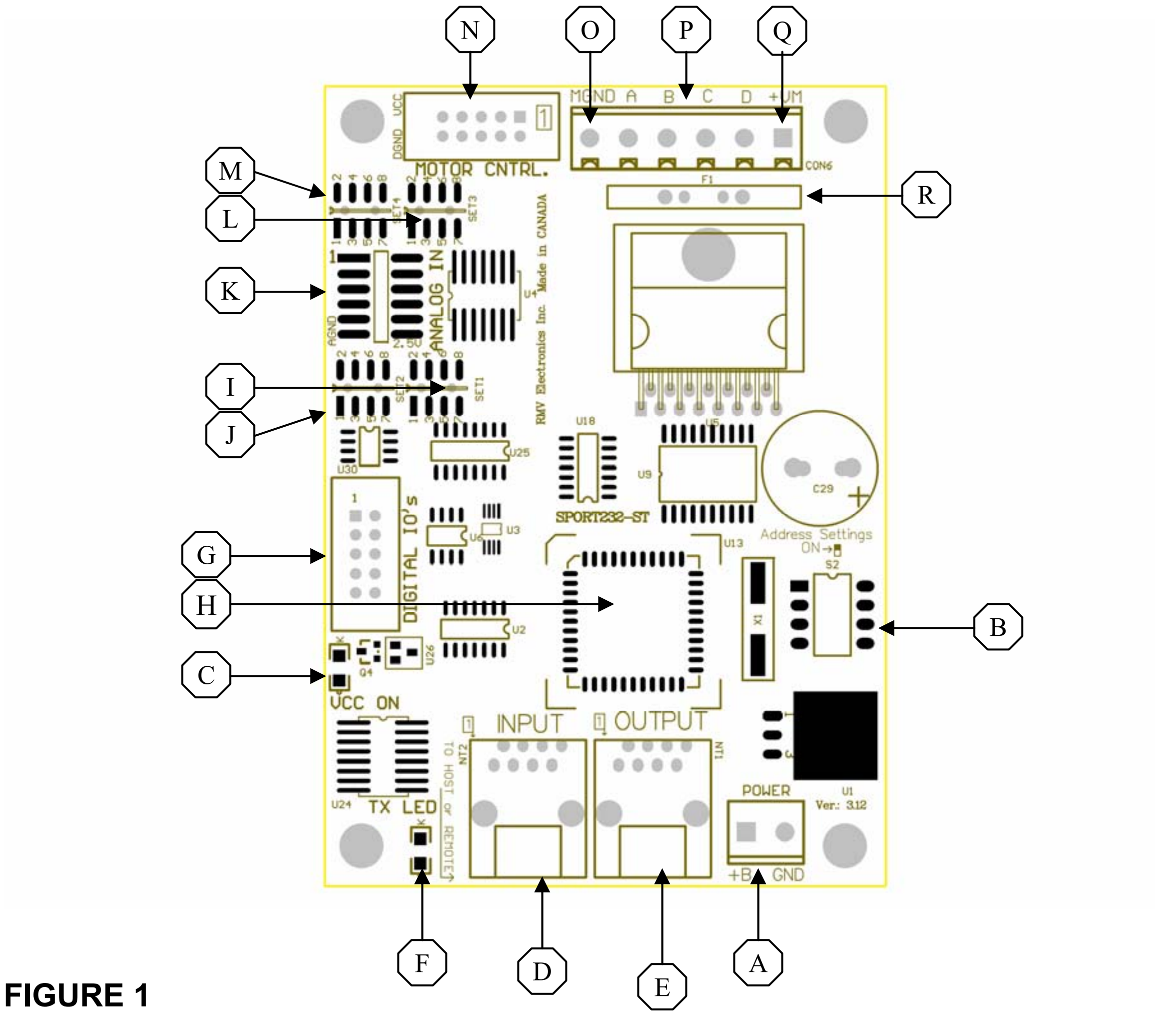


FIGURE 1

A	Logic Power Input	**J**	Ch1 Set Input Range
B	Address Setting	**K**	Analog Inputs Connector
C	Power On LED	**L**	Ch2 Set Input Range
D	Serial Connection to HOST	**M**	Ch3 Set Input Range
E	Serial Connection to Remote	**N**	Motor Control Signals
F	TX LED Indicator	**O**	Motor Ground
G	Digital IOs	**P**	Motor Phases
H	RMV856 Controller	**Q**	Motor Power
I	Ch0 Set Input Range	**R**	Motor Poly Switch

Chapter 3 GETTING STARTED

I- Connect SPORT232-ST Board to a Computer

Due to the variety of connectors used from one computer manufacturer to another, the serial cable is provided with the SPORT232-ST board.
The serial cable consist in the following parts: a RJ485 cable, plus a DB9 female to RJ45 D-Sub connector.
Table: 1 shows different connections in a Null-Straight-RJ45 cable, for DB25 or DB9 D-SUB connectors.
Connect one side of the provided serial cable to the board serial port RJ45 connector identified as INPUT in the Figure 1 as letter #D. Using the RJ45 to DB9 female adapter connect the other end of the RJ45 cable, and then to an available serial port of your PC. Use a DB25-male to DB9-male adapter if applicable.

TABLE 1 - SERIAL CABLE CONNECTIONS

SIGNAL	NULL MODEM CABLE		STRAIGHT CABLE		RJ45 CONNECTOR
	DB25-FEMALE	DB9-MALE	DB25-FEMALE	DB9-MALE	
GND	PIN 7	PIN 5	PIN7	PIN 5	PIN 4
RxD	PIN 3	PIN 3	PIN 3	PIN 2	PIN 5
TxD	PIN 2	PIN 2	PIN 2	PIN 3	PIN 6

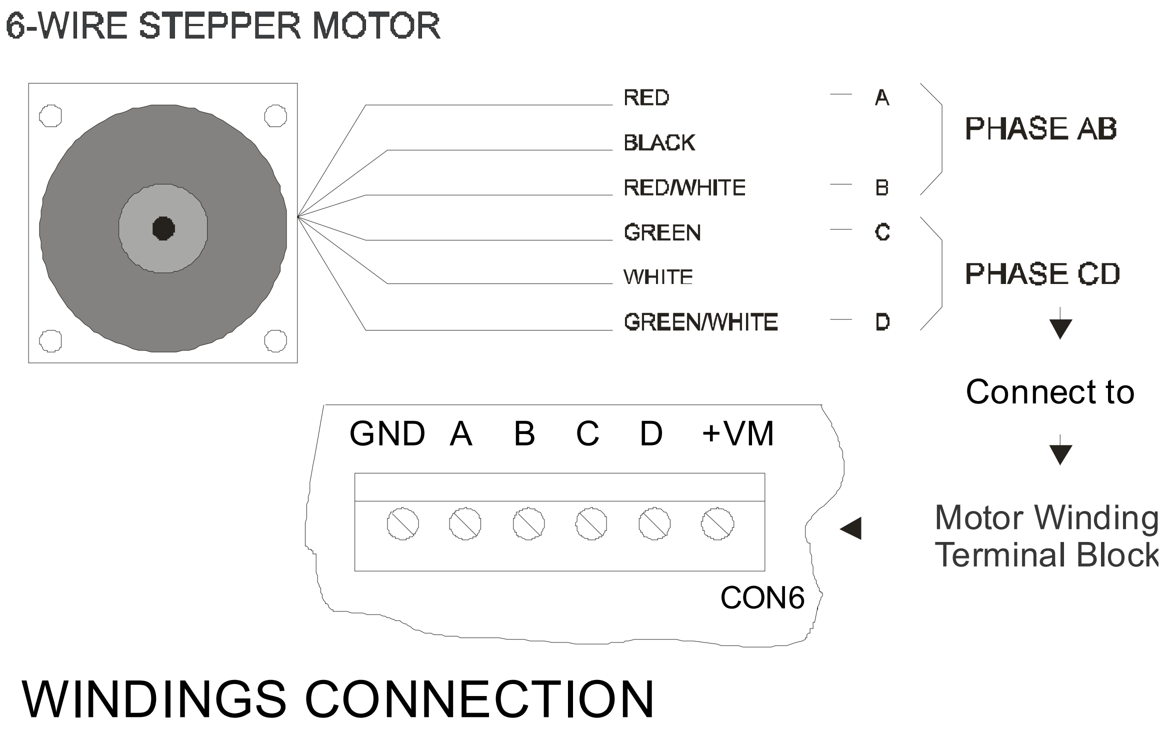
II- Power ON SPORT232-ST

In order to operate the SPORT232-ST Board, connect a power supply or battery between 7 to 12 VDC to the Power terminal block called "POWER" located at letter A in the Figure 1. This can be provided through a wall AC/DC transformer (500 to 1000mA is standard for most applications), a battery (at least 9V), or any other DC power supply.
In order to avoid reversed connection please connect *positive* (power supply) to *+B*, and *negative* (power supply) to *GND*.
Note: If the connection has been connected properly the LED **"VCC ON"** will glow. If the LED is off please review the connections and try again.

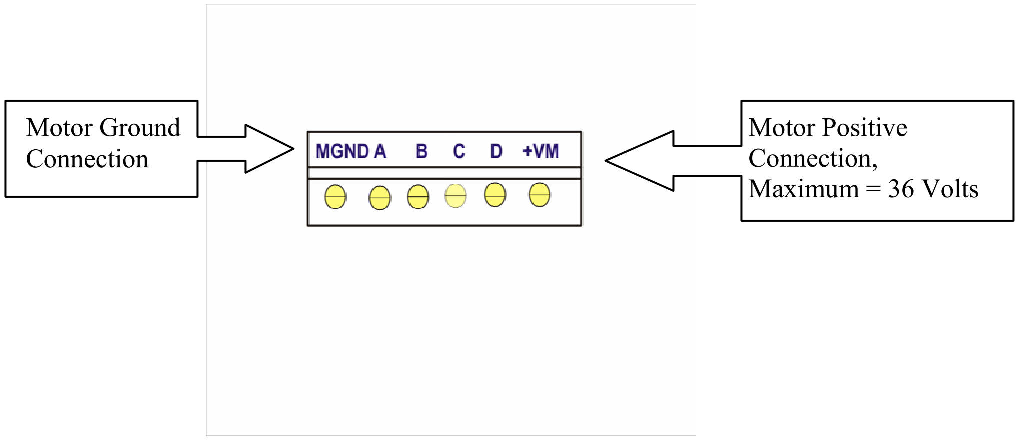
If however the input voltage drops to the point that the linear regulator decrease the VCC line lower than 4.66 Volts, *the RMV856 controller will go to the stage of "Hardware RESET". In addition, the LED "VCC ON" will go to the OFF stage.*

III- MOTOR POWER and WINDING CONNECTIONS

In order to connect the motors, winding terminals need to be identified first, and connected to terminal blocks in CON6 in A,B,C, and D as shown as follows:



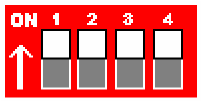
The picture above shows the standard wire colors for a 6-wire stepper motor. Since this kind of motor is intended for use with unipolar drivers, they have a centered tap connection that is no needed when using a bipolar driver, like the one found on the SPORT232-ST. If your motor doesn't match the above depicted color configuration, and easy way of identifying phase wires is using an ohmmeter. Look for a pair of wires that have continuity and measure its resistance. You will find 2 sets of three wires, on each of them two wires will have a higher resistance value. Identify those and mark them as A and B (Phase AB). Repeat this operation for the second set of three wires and mark them as C and D (Phase CD). Once phases have been identify, you can connect the wires to the motor winding terminal block (#1 in the board overview) as marked on board: terminals A, B, C and D. The next issue is to provide a power supply for the motor.



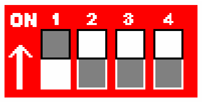
In order to have a better performance on the current mode driver the voltage used for driving the motor should be at least twice as high the nominal voltage for the windings (you may find this data available somewhere on the motor's body). For instance, a 36 VDC / 16 Amps power supply should be enough for a 4 motor application with each drawing 2 Amp per coil, and connecting all of them with a common power supply. The operating current can be set later on in software. Immediately after powering on the board, the operating current on the choppers is set to zero. Connect the positive terminal of the power supply to terminal **VM+** on the motor winding terminal block (#1) and the negative to **MGND**. Observe polarity carefully

IV- SETTING SPORT232-ST ADDRESS

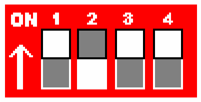
The SPORT23-St has a four bits SMT switch in order to set up each SPORT232-ST address. It is very important to set properly each address so until 16 SPORT232-ST can be operated over one serial port.
Each address is represented in binary from 0 to 15, for example the address Zero can be programmed as “0000”, and the address 15 can be programmed as “1111”.
following table shows



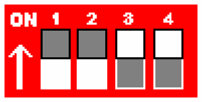
ADDRESS “0”



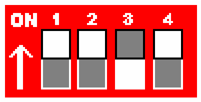
ADDRESS “1”



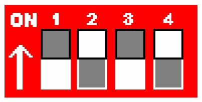
ADDRESS “2”



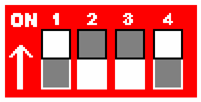
ADDRESS “3”



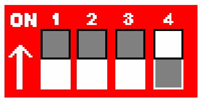
ADDRESS “4”



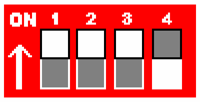
ADDRESS “5”



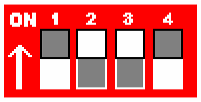
ADDRESS “6”



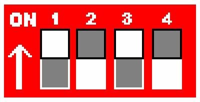
ADDRESS “7”



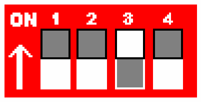
ADDRESS “8”



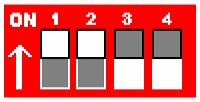
ADDRESS “9”



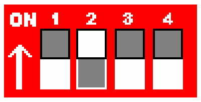
ADDRESS “10”



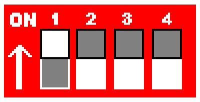
ADDRESS“11”



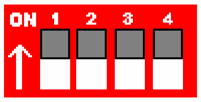
ADDRESS “12”



ADDRESS “13”

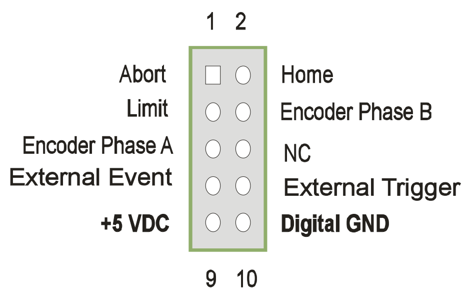


ADDRESS “14”



ADDRESS “15”

V- EXTERNAL MOTOR CONTROL SIGNALS



External Motor Control Connections

Inputs for home, limit switch and abort are available on the Connector Called “Motor CNTRL”. This 10 pins IDC Connector is showed in Part and Location N

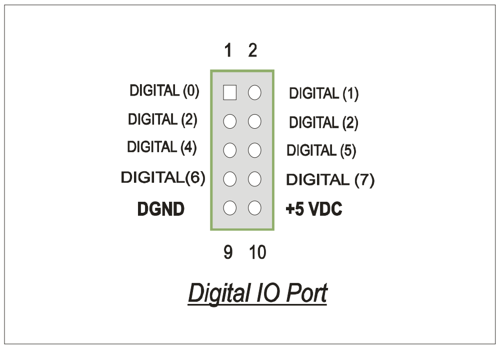
Also the signals necessary for connecting to a shaft encoder, or an external power driver can be found on this connector as shown on the picture. *Abort* input (when low) causes the motor to stop at once and clears any remaining operations stored on the FIFO memory.

The *Limit* signal (when low) will stop the motion and set a flag according to the moving direction (CW or CCW).
The *Home* signal (when high) will also stop the motion and set a flag, provided that the SEEK_HOME mode has been previously enabled on the MOTORCONFIG register.
The *External Trigger* signal (when Low) will trigger the motor, only if this pin will have validity its state, if the flag TRIGGER_EXTRN has been enabled in MOTORCONFIG register.
The *External Event* signal (when Low) will trigger the event command, only if this pin will have validity its state, if the flag EVENT_EXTRN has been enabled in MOTORCONFIG register.
Reading the MOTORSTATUS register can monitor the state of these three flags.

The following flags can be programmed in the MOTORCONFIG 16 bits register from the RMV856-ETX controller:

HALF_STEP	1
BIPHASIC	2
MONOPHASIC	4
EXT_DRIVER	8
ENCODER_FDBK	16
POWER_SAVE	32
VEL_PROFILE	64
SEEK_HOME	128
TRIGGER_EXTRN	256
EVENT_EXTRN	512

VI- DIGITALS INPUT/OUTPUT



The SPORT232-ST has 8 TTL compatible Input - output lines are available for control of external devices or reading any kind of switches. Each line can sink or source up to 5 mA.
Before a digital IO can be used, the bus needs to be configured as Input “0” or Output “1”.

The RMV856 kernel has a series of commands for write-read-and configure a 8 bits digital h. For more information in this topic please refer to the Software Section.

VII- Setting Analog Channels Attenuation

Analog Connector Description

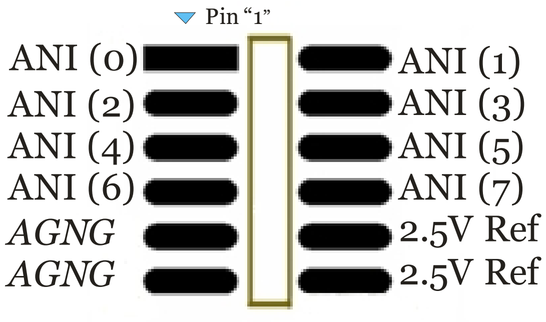
The SPORT232-ST has been design with an analog to digital converter of 12 bits resolution and eight independent analog channels. The analog input signals can be connected trough a 2mm shrouded header (12 pins), this connector is showed in the **BOARD DIAGRAM** **, refer Figure 1 in the letter (#K).**

Four channels ANI(0), ANI(1), ANI(2), ANI(3) have signal conditioning already embedded in the board. Each channel can be programmed to its dynamic range trough a jumper selection, allowing input signals which vary in voltage from : 0-2.5V, 0-5V, and 0-10V.

Pins 10 and 12 from the analog input connector has a 2.5 Volt precision voltage reference output, moreover these two pins can drain a maximum of 15 mA in total.

The analog to digital converter has an internal 2.5V voltage reference, and can be enabled or disabled by software. If model of the SPORT232-ST does not have 2,5V output capability, the ADC internal reference can not be disabled by software.

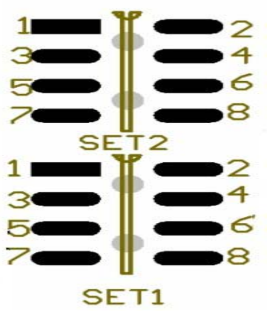
The following diagram shows the Analog Connector Pin Description:

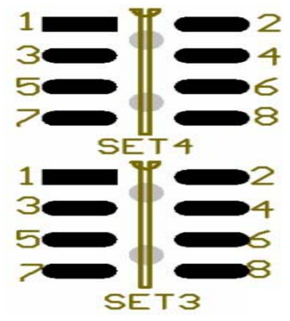


NOTE 1: Only SPORT232-STIxx Versions will have 2.5 Volt Reference Out capability.
NOTE 2: The part number for the shrouded 2mm male connector from FCI is as follow
Part Number :98424-S52-12u
The part number for the crimp-to-wire housing, double row, un-keyed, from FCI is as follow:
Part Number: 69307-012

Setting Jumper Channel Attenuation

The input attenuator scales the analog input according to the voltage reference for each channel as provided by the Analog to Digital Converter (2.5V). This prevents the input voltage of the selected input channel from exceeding the dynamic range of the input amplifier. The input attenuator must be set for each individual channel being used. On the SPORT232-ST the input attenuation may be set for up to four channels (**ANI(0), ANI(1), ANI(2), ANI(3)**).
This is done by setting the jumpers on Set1 for channel 1, Set2 for Channel 2, Set3 for Channel 3, and Set4 for Channel 4.
The following two figure shows the Connector Set1,Set2,Set3, and Set4





The following table, which is also printed on the SPORT232 Board PCB for easy reference, indicates the correct jumper settings for a given dynamic range for a channel.

Dynamic Range	Jumper Settings
0 to 2.5 VDC	8-7
0 to 5 VDC	6-5 and 4-3
0 to 10 VDC	6-5 and 2-1

Note 3: Input impedance is approximately 1 MOhms.

Voltage Calculation

All voltage readings obtained through the analog input channels do not give true voltage. These readings are relative to the on-board voltage reference which is 2.5V. In order to calculate the reading value in volts (V_r), with a 12 bit ADC, the following formula must be used:

$$V_r = \frac{reading}{4096} X\ 2.5V$$

In order to determine the actual voltage (V_a) for each channel the resultant voltage figure must be multiplied by a constant, depending on the dynamic range of the input selected for that particular channel.

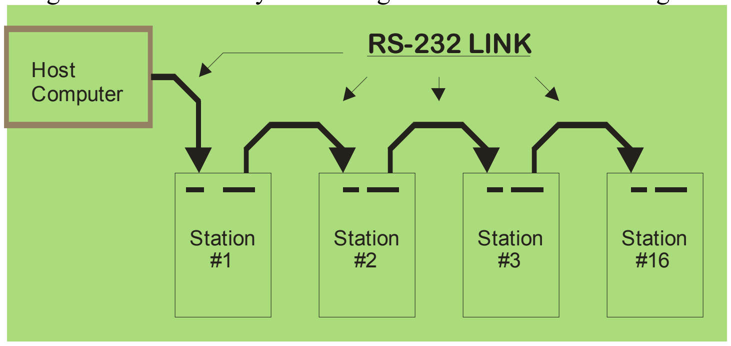
Different Dynamic Range

0 to 10 V	V_r x 4 = V_a
0 to 2.5V	V_r = V_a
0 to 5V	V_r X 2 = V_a

CHAPTER: 5 MODE OF OPERATION

Host Interface and Communication with PC's

Interfacing is accomplished by using a standard capacitive charge-pump RS232 IC, which generates the voltage sources necessary for driving the RS232 TX DATA signal.



Two driver-receiver pairs handle the RS232 interface. The first one handles communication with the host (or remote board) while the remaining one allows multidrop operation.

Networking Operation

Up to sixteen SPORT232-ST can be networked, making a 16 stepper motor system to be controlled by a single computer. The RS232 data flows from the host computer to the 1st board and from there it "daisy-chains" to the 2nd board. In this way up to 16 board can share the same RS232 line. On every station (an SPORT232-ST board) the signal gets repeated and sent to the next station.

RMV856 Stepper Motor Controller

The heart of the stepper motor control is the RMV856 IC. This is a CMOS custom microcontroller that takes care of all the functions necessary to control the stepper motor, digital input-output and analog inputs. It has been designed using a network-oriented concept, which allows easy interaction and programming of several controllers at the same time. An embedded UART on this microcontroller allows asynchronous communication using any standard speed between 9600 and 115200 Bauds. The RMV856 can work together with a 4 phase standard driver (*phases A, B, C and D*), or generate the signals required for and external driver (*step, direction and power control*), depending on the configuration settings.

Power Driver Section

All the stepper motors are driven by an H-bridge IC, the L298. This is a high voltage, high current, dual-full-bridge driver. It can handle up to a 2 Amp current and is controlled to work as current mode chopper. Winding currents can be tightly controlled according to the value set by the on-board Digital to Analog Converter IC . The user has the ability of setting this current to any value between 50 mA and 2 Amps. When half step mode is selected, a torque compensation technique shapes the driving current to follow a pseudo-sinusoidal waveform. Current shaping greatly reduces resonance associated with full step driving, while improving torque characteristic of half step driving. Free wheel diodes are connected to the H-bridge legs, so that a very fast turn off time is achieved, allowing high speed motor stepping.

Winding Current Setting and Power Saving

Operating winding current can be set to any value between 0.05 and 2.00 Amps, by using the corresponding software function on the DLL. When the Power Saving is enabled, if the motor is idle for a period longer than 1 second, the winding currents will be set to half the programmed value. No sooner does the motor restart its motion than the power control is taken over by the stepping procedure, and the current returns to the initially set value.

Connections of Stepper Motors

8, 6, 5 or 4 lead stepper motors can be connected to the board using the terminal blocks Motor1 to Motor4 (#1). A separated power supply connection for each motor is provided on each of them. A common ground connection between all motors and the board power supply is arranged so that the ground terminal is the most negative point in all the connections. Unipolar motors are connected in bipolar mode, which leads to a better utilization of the windings and avoids unnecessary heating. This means that the centered taps must not be connected. The best thing to do is to keep them isolated by using a piece of shrinking tube and avoid any short circuit.

Power-Supply for the Board

The board power terminal block called “POWER” provides a connection for board power supply. Power supply voltage ranges from a minimum of 8 VDC to a maximum of 12 VDC. An inexpensive 12 VDC/500 mA wall transformer is a good option for satisfying that requirement.

Power Supply for the Stepper Motors

This power supply must meet stepper motors requirements in terms of torque, driving method and speed. Depending on whether the motors are going to be operated in constant current mode or not, the voltage must meet the requirement of the chopping circuit. That means that in order to achieve a high stepping rate, the motor's power supply voltage should be high enough for decreasing the turn on time on the windings. A typical voltage for most NEMA23 motors is 36 VDC. This power supply must also be able to provide the peak current at which the motor is rated. For instance, suppose the motor to be driven requires 3.3 V/2 Amps per winding to provide the rated static torque. The equivalent winding resistance is:

Equivalent Resistance = 3.3V/2 A = 1.15 Ohms

Increasing the voltage from the nominal value of 3.3 VDC to 36 VDC will make the time constant to decreased about ten times, since the resistance value has been incremented by the same amount. This reduction in the electric time constant will allow the motor to reach a pull in rate very much higher than using a 3.3 VDC power (which is the supply voltage for providing the static torque current). That means that the acceleration rate and pullout torque at high stepping rate will also benefit from this situation.

Shaft Encoder Operation

A quadrature two channels shaft encoder can be connected to the External Control for reading back the shaft position. A 24-bit register is available for this purpose and it can be read at any time. Another use of the shaft encoder is the motor stall detection. When enabled, this feature will stop the motor if the controller detects that there is no position confirmation from the encoder when a step has been taken. If you are using a motor with a shaft encoder ready, be aware that in order to use the stall detection feature the number of steps per revolution on the motor must be equal to the encoder's pulse count per revolution. Also keep in mind that the encoder phases should be connected in a way that when the motor turns CW the encoder position register is incremented. This can be done by properly connecting the encoder's wires to Encoder Phase A and B inputs on the External Control Connector for that particular motor. If the result indicates a situation that opposes to the one above mentioned, the encoder's wires should be switched. You can use the DEMO program in order to read the encoder counter register and verify the operation above described.

Chapter 6: SOFTWARE DESCRIPTION

The SPORT232-ST is distributed with a DEMO software written for Windows XX operative system, and an interface Dynamic Link Library called "ST400NT.dll". This library is also used in the ST400NT Series. This DLL has specific functions to be addressed only to SPORT232-ST board. The specific functions correspond to the following topics: Initialization Process, Setting winding current, Trigger ON or OFF, Pause ON-OFF, and Read-Configure-Write to a Digital Port, Read an analog input .

The below descriptions shows the functions for the ST400NT.dll in order to program the SPORT232-ST board. When an application program is written, the following explanation shows the order of the ST400NT.DLL functions must be called, in order to initialize the SPORT232-ST.

1) PortOpen(int cPort, int cBaud), or PortConfigure(void)
2) GetConnectedControllers(char* StepperControllers)
3) SPORT232ST_Init (void)
4) SPORT232ST_SetCurrent(char* address, int current)

   After doing 1 to 4 the SPORT232-ST has been initialized, and the winding current from the stepping motor has been programmed.

Functions Description

Communication Functions

Serial Port Functions

Int PortOpen(int cport, int cBaud)

This function open the Serial Port of the computer for the port specified and the speed programmed.
Parameters : int cPort { 1,2,3,4 }
int cBaud { 9600, 19200, 38400, 57600, 115200 }
return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int PortClose(void)
This function close the Serial Port of the computer.
Parameters : NONE
return : 1 Not Error
-1 Error , Please call the function char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int PortConfigure(void)
This function open the Serial Port of the computer, when is called will appear the
windows and the user can chose the serial port and the baud rate.

Parameters : int cPort { 1,2,3,4 }
int cBaud { 9600, 19200, 38400, 57600, 115200 }
return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int RMVGetPort(void)
This function "RETURN " which Serial Port of the computer has been open.

Parameters : NONE
return : { 1, 2, 3, 4 }
If the Serial Port has not been open this function will send the default value =2.

Int RMVGetBaudRate(void)

This function "RETURN " which "BAUD RATE " the Serial Port of the computer has been
open.

Parameters : NONE
return : { 9600, 19200, 38400, 57600, 115200 }
If the Serial Port has not been open this function will send the deafault value =
9600.

Int RMVChangeBaudRate(long NewSpeed)

This function change the "BAUD RATE" for the Serial Port of the computer, at the
speed programmed in NewSpeed.

Parameters : int cBaud { 9600, 19200, 38400, 57600, 115200 }
return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

SPORT232-ST Addressing

SPORT232-ST Addressing Functions

Int CheckControllerAddress(char * ICAddress)
This function Check if the Address sent to processor respond, if return -1 the address
selected is wrong

Arguments: char * ICAddress = Address for the Controller
return : 1 Address is OK
-1 Address is Not OK

Int GetConnectedControllers(char * StepperControllers)

This function return all the Stepper Motor Controller are connected ,

Parameters : StepperControllers this variable return a string with all the RMV856-ETx Stepping Motor
Connected to the serial port.
for example 0,1,2,3,4,5,6,7, N,N,N,N,N,N,N,N
return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

SPORT232-ST Initialization

This function will initialize the SPORT232-ST, resetting all the RMV856-ETx controllers connected in daisy chain.

int SPORT232ST_Init (void);

This function will initialize automatically all the SPORT232-ST connected in the network. Before this function is called *GetConnectedControllers(char * StepperControllers)* need to be call.
NOTE: Is very important to call this function in the beginning of application.

Parameters : NONE (internally all the address will be setup)
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Setting Motor Current

Set Winding Current

Int SPORT232ST_SetCurrent(char* address, int current);

This function will set the current in mA for specified address.

```
int  ITCAPI  SPORT232SR_TriggerON( char *addrlist);
 int  ITCAPI  SPORT232ST_TriggerOFF (char *addrlist);
 int  ITCAPI  SPORT232ST_PauseON  (char *addlist);
 int  ITCAPI  SPORT232ST_PauseOFF( char* adrdrlist);Example:

if ( MotorSetCurrent(ADDRESS1, current ) == -1)
{
      strcpy(s1," ERROR in Set Current Motor Number :" );
       strcat(s1,s);
       MessageBox (NULL,RMVGetErrorMessage(), " ERROR in Set Current Motor"
,MB_OK|MB_ICONERROR);
}
```

Motor Configuration Mode

Motion Confuguration Mode

Int MotorConfigureMode(char *address,int Mode)
This Function will set Mode operand for the Motor Controller

FLAGS :

HALF_STEP = 1
BIPHASIC = 2
MONOPHASIC = 4
EXT_DRIVER = DISABLE
ENCODER_FDBK = 16
POWER_SAVE = 32
VEL_PROFILE = 64
SEEK_HOME = 128
TRIGGER_EXTRN = 256
EVENT_EXTRN = 512

HALF_STEP
When enabled the motor will step in Half Step Mode

BIPHASIC
When enabled the motor will step in Full Step Mode, (two phases ON at a time)

MONOPHASIC
When enabled the motor will step in Full Step Mode, (one phase ON at a time)

EXT_DRIVER
Disabled for the SPORT232-ST

ENCODER_FDBK
When enabled the RMV856 Controller will check for a valid encoder change in position
and if "TWO CONSECUTIVE STEPS ARE TAKEN, THE MOTION WILL STOP AT ONCE". A corresponding
flag will be set on the motor Status Register, please refer to MotorGetStatus.

POWER_SAVE
When enabled, after motion is completed (motor is in idle state), the controller
will WAIT for the specified POWER SAVE TIME (refer to MotorPowerON), and then will
decrease the operating current to half the programmed value.

VEL_PROFILE
When enabled, the controller will step starting at a rate determined by the latest

FIRST SPEED entered. From that initial value the rate will be increased or decreased (depending on sign and magnitude of the latest SLOPE entered) until the target position is reached (instructed by FIFOMotorGoStepRel or FIFOMotorGoAbsPos functions).

SEEK_HOME
When is enabled, the controller will wait a valid high level signal on the HOME input. If this signal is received the motion will stop at once and a corresponding flag will be set on the MOTOR STATUS REGISTER.

TRIGGER_EXTRN
Enable the Trigger External signal, the kernel for the RMV856 will hold the motion programmed until External Signal goes low.

EVENT_EXTRN
This flag is disabled at this moment.

Int MotorConfigureMode(char *address,int Mode)

Parameters: The broadcasting mode is NOT allowed
char * adrlist = { "0" or "1" or "2" or ....or "15"} address of RMV856 Controllers
int Mode = (for enable) => HALF_STEP and POWER_SAVE and ENCODER_FDBK ,
(for Disable)=>HALF_STEP and POWER_SAVE and (NOT)ENCODER_FDBK
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Software TRIGGER and PAUSE

Int SPORT232SR_TriggerON(char *addrlist);
This function will trigger ON the motor via a software command.

Parameters: The broadcasting mode is allowed
char * adrlist = { "0" or/and "1" or/and "2" or ....or/and "15"} address of RMV856 Controllers

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int SPORT232ST_TriggerOFF (char *addrlist);
This function will trigger OFF the motor via a software command.

Parameters: The broadcasting mode is allowed
char * adrlist = { "0" or/and "1" or/and "2" or ....or/and "15"} address of RMV856 Controllers

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int SPORT232ST_PauseON (char *addlist);
This function will pause ON the motor via a software command.

Parameters: The broadcasting mode is allowed
char * adrlist = { "0" or/and "1" or/and "2" or ....or/and "15"} address of RMV856 Controllers

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int SPORT232ST_PauseOFF(char* adrdrlist);
This function will pause OFF the motor via a software command.

Parameters: The broadcasting mode is allowed
char * adrlist = { "0" or/and "1" or/and "2" or ....or/and "15"} address of RMV856 Controllers

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Analog Input Functions

Int SPORT232ST_GetADCchannel(char* address, int channel, int* adc_value)

Reads a channel from an 12 bit Analog to Digital converter installed into
the SPORT232-ST board.
Parameters: The broadcasting mode is NOT allowed
char * *adrlist* = { "0" or "1" or "2" or ....or "15"} address of RMV856 Controller
int *channel* : 0 to 7 {ANI(0)…. ANI(7) }
*int * adc_value* : 0 to 4095

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Digital I/O Functions

The SPORT232-ST board has 8 bits as digital IO TTL , and each bit can be configured as input or output interpedently.

Int SPORT232ST_ConfigureDigitalUserIO(char* address , int USerCnf)
Parameters: The broadcasting mode is allowed
char * *adrlist* = { "0" or/and "1" or/and "2" or ....or/and "15} address of RMV856 Controller
int USerCnf : 0 = All Configured as Inputs
255 = All Configured as Outptu
"0" Configure as input
"1" Configure as output
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int SPORT232ST_WriteDigitalUserIO(char* address, int gUserData)
Parameters: The broadcasting mode is allowed
char * adrlist = { "0" or/and "1" or/and "2" or ....or/and "15"} address of RMV856 Controllers
int gUserData 0 TO 255
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int SPORT232ST_ReadDigitalUserIO(char* address, int* value)
Parameters: The broadcasting mode is NOT allowed
char * *adrlist* = { "0" or "1" or "2" or ....or "15"} address of RMV856 Controller
*int * value* : 0 to 255

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Motion Related Commands

Here are all the Functions respecting to movement of the " Stepper Motor Controller
" this functions are stored in the internal FIFO

Int MotorWait(char * address, unsigned int Delay)
This Function will insert a Delay in milliseconds
Parameters: The broadcasting mode is NOT allowed
*char * address*= { "0" or "1" or "2" or ....or "15"} address for the Controllers addressed
unsigned int Delay = from 1 msec. To 65535 msec.

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorSpeed(char * address, int Slew_Rate, int First_Rate, int Accel)
This function set all the trapezoidal movements, the parameters for this function
will not be storage into the internal FIFO. The broadcasting mode is NOT allowed.

Parameters : char * address = { "0" or "1" or "2" or ....or "15"} address for the Controllers wanted
int Slew_Rate = from 16 to 8500 Steps/sec
int First_Rate = from 16 to 8500 Steps/sec
int Accel = from 0 to 255 (minimum 0 , maximum 255)
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorGetPosition(char * address, long * position)
This function return " The Counter Position Register" from the RMV856 Controller.
trough (*position). Broadcasting is not allowed.

Parameters : *char * address* = { "0" or "1" or "2", ...,or "15"} only one address is allowed
*long *position* = position returned
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorSetPosition(char *adrlist, long NewPosition)
This function set "The Counter Register from the RMV856 Controller" to the value
specified in NewPosition, another use of this function is for "RESETING" the Counter Register in this case NewPosition = 0.

Parameters :. Broadcasting is allowed.
*char * adrlist* = { "0", "1", "2", ..., "15"} address for the Controllers wanted
long NewPosition = from -8388607 to +8388607
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorPowerON(char * adrlist, int PowerDownTime)
This function turn "ON" the power for the Stepper Motor, and set the timing for Power Down in idle mode.
In order to enable the "Power Save" feature in the "MotorConfigureMode" function the flag POWER_SAVE must be set. broadcasting is allowed.

Parameters: char * adrlist = { "0", "1", "2", ..., "15"} address for the Controllers wanted
int PowerDownTime = from 1 to 10 sec.
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorPowerOFF(char * adrlist)
This function turn "OFF" the power for the Stepper Motor.

Parameters : Broadcasting is allowed.
char * adrlist = { "0", "1", "2", ..., "15"} address for the Controllers wanted
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorDirection(char * address , int direction)
This function set the direction to the Stepper Motor, CW or CCW.

Parameters : char * adrlist = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int Direction = CKWISE =1, CCKWISE = 0

Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorNumberStepRel(char * address , long step_to_go)
This function set the number of the steps in which the motor will go, at the final speed, acceleration and the first rate previous set .

Parameters: char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
long step_to_go = from -8388607 to +8388607
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Example:
```
if (-1 ==(MotorSlewRate(ADDRESS1,slew_rate))
{
     MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
    return NOT_OK;
}
if ( -1== MotorFirstRate(ADDRESS1, first_rate) )
{
  MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
   return NOT_OK;
}
if( -1 == MotorAccelDecel(ADDRESS1, acceldecel))
{
   MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
   return NOT_OK;
}
if ( -1== MotorNumberStepRel( ADDRESS1, 100000)) // HERE the MOTOR STARTS MOVING
    // "PREVIOUSLY HAS BEEN TRIGGERED"
{
    MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
    return NOT_OK;
}
```

Int MotorGoAbsPos(char * address , long step_to_go)
This function set the position " Absolute " at which the motor will go, at the final speed, acceleration and the first rate previous set. Let suposse the current position is at +1000, the function MotorGoAbsPos(ADDRES1, -1000), will make the motor to move 2000 CCWISE direction until reaches position = -1000.

Arguments. Broadcasting is NOT allowed.

Parameters : char * adrlist = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
long step_to_go = from -8388607 to +8388607
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Example:
```
if ( -1== MotorGoAbsPos( ADDRESS1, -1000)) // HERE the MOTOR STARTS MOVING to the
// -1000 position
// "PREVIOUSLY HAS BEEN TRIGGERED"
{
  MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
  return NOT_OK;
}
```

Int MotorSlewRate(char * address, int slew_rate)

This function specify the slewing step rate for the movement. Please refer to the function GetFixedSpeed in order to know the real speed that the RMV856 controller will generate. This function will set the Slew Rate for TRAPEZOIDAL PROFILE MODE. Before calling this function, the function MotorConfigureMode MUST be called and disable VEL_PROFILE flag.

Arguments. Broadcasting is NOT allowed.
Parameters: char * address= { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int slew_rate = from 16 to 8500 Step/sec
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorFirstRate(char * address, int first_rate)
This function set the First Rate for the movement. Please refer to the function GetFixedSpeed in order to know the real speed that the RMV856 controller will generate. This function will set the First Rate for TRAPEZOIDAL PROFILE MODE. Before calling this function, the function MotorConfigureMode MUST be called and disable VEL_PROFILE flag.

Arguments. Broadcasting is NOT allowed.
Parameters : char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int first_rate = from 16 to 8500 Step/sec
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorAccelDecel(char * address, int acceldecel)
This function will set the Acceleration-Deceleration rate for TRAPEZOIDAL PROFILE MODE. Before calling this function, the function MotorConfigureMode MUST be called and disable VEL_PROFILE flag.

Arguments. Broadcasting is NOT allowed.
Parameters : char * address= { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int acceldecel = from 0 to 255 Step/ (sec**2)
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorRepeat(char * address)
This function will repeat the motion according to latest parameters sent.

Arguments. Broadcasting is NOT allowed.
Parameters : char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorJog (char * address, int direction)
This function will set the motor in JOG Mode.

Parameters : char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int direction = CKWISE =1, CCKWISE = 0
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorChangeJogSpeed (char * address, int direction)
This function will change the motor in JOG speed. Before use this function MotorJog
need to be called.

Parameters: char * address= { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int direction = CKWISE =1, CCKWISE = 0
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Example
```
// UpDown Control
   int Size3 = MDISJOG1->GetTextLen(); // Tmemo
  char *Cmd = new char[++Size3];
    MDISJOG1->GetTextBuf(Cmd,Size3);

      JogSpeed1 = atoi(Cmd);
if ( -1== MotorChangeJogSpeed(ADDRESS1,JogSpeed1))
{ MessageBox(NULL,RMVGetErrorMessage(),ERRORMSG1,MB_OK|MB_ICONERROR);
}
```

Int MotorStop(char * address)
This function Stop the motion independently which mode has been set up. In order
to continue with the motion the parameters need to be sent again.

Parameters: char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorSoftReset(char * address)
This function "RESET" the RMV856 Controller, all the parameters and the FIFO will be erased.
WARNING: the baud rate will be set at : 9600 baud, if the serial port for the computer was open at different speed an error will occur like "COM port timed out".
Arguments. Broadcasting is NOT allowed.

Parameters: char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Int MotorGetStatusRegister(char * address , int * Status)
This function return the Status Register flags from the RMV856 Controller in which the address has been selected.

The Flags are:

TIGGER_ON = 1	; set when motor has been triggered
PAUSED = 2	; set when motor is being hold (paused)
FULL_STOP = 4	; set when motor was stopped/abortted by hardware or software
RUNNING = 8	; set when motor is moving
SWITC_CCW = 16	; set when CCW limit has been reached
SWITCH_CW = 32	; set when CW limit has been reached
AT_HOME = 64	; set when home switch is on
MOTION_CMPLT = 128	; set when motion has been completed
CURRENT_DIR = 256	; "1" ;set for direction for CW, "0" set for direction for CCW
MOTOR_STALLED = 512	; "1" set when position error is bigger than two steps
MOTORTRIG_EXTERNAL= 2048	; "1"; External Trigger has been set
EVENTTRIG_EXTERNAL = 4096	; "1" External Event has been set

Arguments. Broadcasting is NOT allowed.

Parameters: char * address = { "0" or"1" or "2" or ....or "15"} address for the Controller wanted
int * Status = returned value
Return : 1 No Error
-1 Error , Please call char * RMVGetErrorMessage() or
RMVGetErrorMessageVb(char * Error), in order to know which kind of error happened.

Get Error Messages

Char * RMVGetErrorMessage (void);

This function return a pointer to character with the error
Arguments: NONE

return = Error String

RMVGetErrorMessageVb (char *Error)
This function return the String Error by reference.

Arguments : char *Error = string with the Error
return 1 allways