Jupiter Instruments

Programmer's Interface Document

JI-820 Incremental Encoder Emulator Software (SW820)

Document No: PID820
Revision:1.1
Date: April 10, 2018

Paul Miller – Software Engineer - Prepared By

Date

Jupiter Instruments
Mission Viejo, CA 92692

Record Revision Sheet			
Rev.	**Reason/Description**	**Approvals**	**Date**
-	Initial release		

Table of Contents

Section **Page**

List of Tables

Table **Page**

1.0 INTRODUCTION

The JI-820 is a flexible, easy-to-use, PC controlled instrument designed to precisely emulate the function of a wide variety of incremental encoders. It provides the design, system, and test engineer with a tool to accurately emulate encoder signals generated by motion control and industrial monitoring systems. Variable encoder parameter available to the user include cycles per revolution, cycle frequency, A/B signal phase, Z signal position and polarity, signal amplitude, and selectable signal interface. An intuitive Windows application manages instrument setup and control. Communications and unit power is all provided via a USB 2.0 interface.

This document defines the software interface to the JI-820 Incremental Encoder Emulator. The interface defined herein is valid over the USB interface.

2.0 HARDWARE INTERFACE (USB)

Communication with the JI-820 is by way of a USB connection. The JI-820 incorporates a USB IC (FT245R from FTDI) that handles both the physical interface and USB protocol. Drivers and DLLs for this device support operation with several programming language types (C++, C#, LabVIEW, etc.) and operating systems (Wndows 7, 8, 10, XP, Linux, etc.) Additionally, a Virtual Com Port (VCP) driver is available that allows communication via a terminal emulator program such as Microsoft HyperTerminal. In this mode, commands can be rapidly tested using either keystroke entry or script file. Detailed information on the operating systems supported and programming examples can be found at the FTDI website (www.ftdichip.com). Specifically, driver downloads are available at http://www.ftdichip.com/FTDrivers.htm and the API for the FTD2XX.dll is available at http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf

2.1 Hardware

- FT245R USB Interface IC from FTDI
  http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT245R.pdf
- Data Transfer Rate:
  - 1Mbit/sec – D2XX Direct Driver
  - 1Mbit/sec - VCP

3.0 SOFTWARE INTERFACE

Commands shall be text strings. Responses shall be text strings.

3.1 Commands

Commands shall be case-insensitive. However, the mixed case specified below is recommended for readability.

Arguments shall be decimal.

All commands shall be terminated by a "\r". For example, a "GetHWVer" command will be sent as "GetHWVer\r"

3.2 Responses

Responses shall be either '!' or '?'.

Successful "Set" commands shall return "!\r". Successful "Get" commands shall return requested value terminated by "!\r".

For example, a successful "Set" command will return the following:

```
!
```

While a successful "Get" command might returned the following:

```
1234!
```

Return values shall always be formatted as decimal numbers.

If an error occurs, a "?\r" is returned.

```
?
```

4.0 COMMAND LIST

Command	Description	Example Command	Example Response
Identification			
GetHWVer	Returns hardware version	`GetHWVer`	`A!`
GetSWVer	Returns software version	`GetSWVer`	`B!`
GetVHDLVer	Returns VHDL version	`GetVHDLVer`	`C!`

Hardware Configuration			
SetVoltageSource	Set Signal Voltage Source • Valid arguments: 0 = Internal, 1 = External • Return: ! = success, ? = error	`SetVoltageSource 0`	!
SetDriveType	Set Output Driver type • Valid arguments: 0 = Push-Pull, 1 = Open-Drain • Return: ! = success, ? = error	`SetDriveType 0`	!
SetSigVoltage	Sets Output signal voltage in mV • Only valid if internal voltage source is selected. • Range: 5000 mV to 18000 mV Increments: 10mV • Return: ! = success, ? = error	`SetSigVoltage 10000`	!
EnableSignals	Enable/Disable Quad Signals (A, B, and Z) • Valid arguments: 0 = float, 1 = on • Return: ! = success, ? = error	`EnableSignals 1`	!

Quadrature Configuration			
SetQuadDirection	Set quadrature signal rotation direction. • Valid arguments: 0 = CW, 1 = CCW • Return: ! = success, ? = error	`SetQuadDirection 1`	!
SetCyclesPerRev	Set Encoder Cycles Per Revolution • Range: 1 to 2^{24} -1 • Return: ! = success, ? = error	`SetCyclesPerRev 1024`	!
SetCycleTimeMult	Set Cycle Time Multiplier (CTM) • This setting effectively sets the Cycle Time. • Argument Range: 1 to 2^{24} -1 • Cycle Time = (1/40 MHz)*(360/ Phase Resolution)*(CTM +1) • Return: ! = success, ? = error	`SetCycleTimeMult 100667`	!
SetQuadMode	Set Quadrature Operational Mode. • One of 4 modes is selected: 1. Free Run 2. Move to an absolute position 3. N/A 4. Move Pulse Count • Valid Arguments: 1, 2, 4 • Return: ! = success, ? = error	`SetQuadMode 2`	!

SetPhase	Set phase resolution and phase value for all 4 phases. • First argument is Phase Resolution followed by phases 1, 2, 3, and 4. • Valid phase resolution values: 1°, 5°, 10°, 45°, 90°. • Each Phase (1,2,3,4) is individually set. • Nominal setting: 90°. • Sum of all phases must = 360° • Phase values must be multiples of Phase resolution setting. • Phase Range: 90° +/- 80° (not to exceed) • Return: ! = success, ? = error • Use GetConfigErrors command to check for setting errors. •	`SetPhase 1 90 90 90 90`	!
SetZModeTiming	Set Z Operational Mode and Signal Timing. • SetZModeTiming is a 4 argument command. First argument is Z Mode, next is phase resolution, next is signal rising-edge phase, and last is signal falling- edge phase. • One of 4 modes is selected: 0. Z signal disabled 1. Z signal resides in last Cycle 2. Z signal resides in first Cycle 3. Z signal resides in both the first and last Cycle • Valid phase resolution values: 1°, 5°, 10°, 45°, 90°. • Rise/Fall phase values range from 0 to 359 (in increments set by SetPhase command) and must be multiples of	`SetZModeTiming 3 10 180 180`	!

	Phase resolution setting. • Rules for Phase settings are as follows: 1. For Modes 1 & 2, rising edge phase < falling-edge phase. 2. For Mode 3, all phase values are valid. • Return: ! = success, ? = error • ? Use GetQuadStatus command to check for setting errors.		
SetZPolarity	Set Z signal Polarity. • Valid arguments: 0 = Pulse High, 1 = Pulse Low • Return: ! = success, ? = error	`SetZPolarity 1`	!
EnableZ	Enable Z Signal. • Valid arguments: 0 = off, 1 = on • Return: ! = success, ? = error	`EnableZ 1`	!
EnableZSS	Enable Single-Step on Z Signal • Valid arguments: 0 = disable, 1 = enable • Return: ! = success, ? = error	`EnableZSS 1`	!

SetPulseMoveCnt	Set Pulse Move Count • Range: 1 to 2^{24} -1 • Moves quad signal in CW or CCW direction N counts. • To Execute: 1) set Mode = "Pulse Move", 2) set Run = 1. • Execution complete when GetQuadStatus CMD, 1 byte, bit 3 = 0. • Return: ! = success, ? = error	`SetPulseMoveCnt 12345`	!

Commands			
ResetQuad	Stop Quarature signal process, clear all counters, over-flow errors, and Reset Position. • Reset position: Rev = 0, Cycle = 1, Phase = 1. • Return: ! = success, ? = error	`ResetQuad`	!
Run	Run/Stop selected Mode • Valid arguments: 0 = Stop, 1 = Run • Return: ! = success, ? = error	`Run 1`	!
SingleStepCW	Single Step CW Command • All other modes must be halted prior to execution (Run = 0). • Function Complete when valid Return received. • Return: ! = success, ? = error.	`SingleStepCW`	!
SingleStepCCW	Single Step CCW Command • All other modes must be halted prior to execution (Run = 0). • Function Complete when valid Return received. • Return: ! = success, ? = error.	`SingleStepCCW`	!

Read			

GetQuadStatus	Get Quadrature status
	• 2 bytes are returned: 1st Processing Status, 2nd Configurtion Status
	
	• 1st Byte: Processing Status
	  • Bit 0 = Mode "Free Run" 1 = Running
	  • Bit 1 =
	  • Bit 2 = Mode "Single Step" 1= Running
	  • Bit 3 = Mode "Pulse Move" 1 = Running
	  • Bit 4 = 0
	  • Bit 5 = 0
	  • Bit 6 = 0
	  • Bit 7 = Rev Count Roll-Over Error
	
	• 2nd Byte: Configurtion Status
	  • Bit 0 = Z Configuration Error
	  • Bit 1 = Quadrature Resolution Setting Error
	  • Bit 2 = Phase Setting Error 1
	  • Bit 3 = Phase Setting Error 2
	  • Bit 4 =
	  • Bit 5 = "Run To" Setting Error 2
	  • Bit 6 = 0
	  • Bit 7 = 0
		`GetQuadStatus`	`01 21!`

SampleQuadPos	Samples and holds quadrature signal position information
	• Captured position information is retrieved using the following commands:
	  1. GetRevCount
	  2. GetCycleCount
	  3. GetPhase
	• Return: ! = success, ? = error
		`SampleQuadPos`	`!`

GetRevCount	Get Quadrature Rev Count • Comamnd used to retrieve captured quadrature rev count. • Count size: 0x7fff ffff (half of 32-bit) • Use SampleQuadPosition command to capture data. • Return: ! = success, ? = error	`GetRevCount`	`123456!`
GetCycleCount	Get Quadrature Cycle Count • Comamnd used to retrieve captured quadrature cycle count. • Count size: 24-bits • Use SampleQuadPosition command to capture data. • Return: ! = success, ? = error	`GetCycleCount`	`54321!`
GetPhase	Get Phase Position • Comamnd used to retrieve captured quadrature phase position. • Count size: 3-bits • Use SampleQuadPosition command to capture data. • Return: ! = success, ? = error	`GetPhase`	`2!`
GetZState	Get Z State • Comamnd used to retrieve state of Z signal. • Data size: 1-bit (0 or 1) • Use SampleQuadPosition command to capture data. • Return: ! = success, ? = error	`GetZState`	`1!`

GetOvrIstatus	Get Z State • Comamnd used to retrieve Power Supply Over-current state. • Return data size: 1-bit (0 or 1) • Return: ! = success, ? = error	`GetOvrIStatus`	`1!`

Table 4-1 Command List

5.0 SOFTWARE PROBLEM REPORTS

There are no Software Problem Reports as this is the initial release.

SPR	Description

Table 54-1: Software Problem Reports