Model JI-300
I2C Host Adapter

Programmer's Interface Document

Version 1.2a

Jupiter Instruments

2/29/2008 Edition

TABLE OF CONTENTS

1. INTRODUCTION

This document provides details on the functional and operational requirements for both hardware and software interfaces between the Model JI-300 I2C Host Adapter and a host computer.

2. HOST COMMUNICATION

2.1 USB Interface

Communication with the JI-300 is by way of a USB interface. The JI-300 incorporates a USB IC (FT245R from FTDI) that handles both the physical interface and the entire USB protocol. Drivers and DLLs available for this device support operation with several programming language types (C++, C#, LabVIEW, etc.) and operating systems (Vista, XP, 2000, Linux, etc.). Additionally, a Virtual Com Port (VCP) driver is provided 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)

No USB-specific programming is required to control the JI-300. Commands (and responses) are comprised of simple text string constructs. A straightforward command/response type protocol ensures that commands (either read or write) are accepted (or rejected) by the JI-300.

2.2 Hardware

- FT245R USB Interface IC from FTDI (www.ftdichip.com)
- Data Transfer Rate:
  - 1Mbyte/sec – D2XX Direct Driver
  - 300Kbytes/sec - VCP

2.3 Command Set

A simple set of ASCII commands is used to control the operation of the host adapter. All commands, command responses, and data are transmitted and received as ASCII character strings.

2.3.1 Syntax

Command structure:

[$][Command character][Argument string][Carriage return]

Each command begins with a start delimiter '$' followed by a single command character. All commands are lower-case alpha characters. Following the command character is the argument string that contains from 0 to 510 characters depending on the command type. Note that not all commands have arguments. A carriage return terminates all commands.

Response structure:

[Argument string][!]

The JI-300 responds to all received commands. A '!' character is returned for all valid commands. Depending on the command, a character or character string may precede the '!' character. A '?' response indicates an invalid command has been received. No character string precedes the invalid command response.

2.3.2 Commands – Write

Note: AA = Hexadecimal text string

<CR> = Carriage Return (\r)

Syntax	R/W	Description	Valid Response
$wAAAAAAA.. ….AA<CR>	W	Write (Tx) I2C Data – with Stop Command This command is used to transmit data to an I2C slave device. A minimum of 0 bytes and maximum of 255 bytes can be transmitted in a single command. The first two characters following the command character indicates the Tx byte count. The next two characters specify the Slave Address. The remaining characters constitute the data payload. • Tx Byte Count: Range = 00h – FFh. When counting Tx bytes, the slave address counts as 1. • Slave Address: Range = even addresses from 00h to FEh. The upper 7-bits of the byte are the address code and the LSB is the direction (0= write). • Data Field: Maximum of 255 bytes Response A valid response includes a status byte followed by a "valid command" delimiter (!). This response will be returned within 10ms, message complete or not. The status byte bit definitions are as follows: Bit 7 Bit 6 0 0 Tx in process (timer time-out) -- bit 5 = > MM mode busy 1= true -- bit 4 = > TBD -- bit 3 = > Clock Stretch 1*= true -- bit 2 = > TBD -- bit 1 => TBD -- bit 0 => Bus is not free 1= true Bit 7 Bit 6 0 1 Done with error (Tx SM) -- bit 5 = > TBD -- bit 4 = > No ACK response 1* = true -- bit 3 = > Clock Stretch 1*= true -- bit 2 = > Bus contention 1* = true MM mode - Loss of Arbitration (bit Tx)	AA!

		-- bit 1 => Bus contention (SDA) 1 = true MM mode - Loss of Arbitration (Start) -- bit 0 => Bus is not free 1= true Bit 7 Bit 6 1 0 Done without error (Tx SM) -- bit 5 – bit 0 => N/A Bit 7 Bit 6 1 1 Message syntax error -- bit 5 – bit 0 => N/A * Use the Error Byte Count register to determine the data byte involved with this error.	
$yAAAAAA…….AA<CR>	W	Write (Tx) I2C Data – without Stop Note that this command is the same as above except that the I2C transmission is not terminated by a Stop. This command is used to create a Repeated Start event.	AA!
$s<CR>	W	Halt Before beginning a new session, send this command to ensure all JI-300 processes have stopped.	!
$mAA<CR>	W	Configuration register -- bit 7 => LED configuration 0 = Direct control 1 = Monitor SCL/SDA activity -- bit 6 => LED direct ctrl 0 = off, 1= on -- bit 5 => Stop after loss of arbitration 1 = enabled -- bit 4 => Multi-Master Mode 1 = enabled -- bit 3 => External Bus Voltage 1 = enabled -- bit 2 => TBD -- bit 1 => Clock Stretch 1 = Infinite period 0 = Fixed period (tCLK_Str) -- bit 0 => Bus-Free Wait 1 = Infinite period 0 = Fixed period (tBUF_Wait)	!
$gAAAA<CR>	W	SCL High time - tHIGH This is the high period of the SCL signal. Argument is 16-bits. t = N*20ns + 120ns. N has a minimum value of 0. t (range) = 120nS to 1310.8uS	!

$uAAAA<CR>	W	SDA Setup time – tSU;DAT This is the SDA time period prior to SCL going high. Argument is 16-bits. t = N*20ns + 60ns. N has a minimum value of 0. t (range) = 120nS to 1310.8uS	!
$hAAAA<CR>	W	SDA Hold time – tHD;DAT This is the SDA time period after SCL goes low. Argument is 16-bits. t = N*20ns + 60ns. N has a minimum value of 0. t (range) = 120nS to 1310.8uS	!
$kAAAA<CR>	W	Bus-Free time – tBUF Bus free time between a STOP and START condition. Argument is 16-bits. t = N*20ns + 60ns. t has a minimum value of 0.	!
$nAA<CR>	W	Bus-Free Wait time – Maximum time host adapter will wait for a Bus-free condition. Argument is 8-bits. t = N*1.6384ms + 1.6384ms. t has a minimum value of 0.	!
$pAAAA<CR>	W	Start Hold & Stop Setup time – tHD;STA & tSU;STO Argument is 16-bits. This value should normally be set to the SCL High time value. t = N*20ns + 120ns. N has a minimum value of 0. t (range) = 120nS to 1310.8uS	!
$jAA<CR>	W	SCL Rise time– tr Maximum rise time before a SCL stretch condition is acknowledged. Argument is 8-bits. t = N*20ns + 60ns. N has a minimum value of 0.	!
$xAA<CR>	W	SCL Stretch time Maximum time before a clock stretch error condition is acknowledged. Argument is 8-bits. t = N*20uS + 20uS. N has a minimum value of 0.	!

$iAAAA<CR>	W	I2C Bus Voltage The bus voltage is adjustable form 1.20V to 5.30V in 1mV increments. The valid voltage range (to ensure that the I2C receivers work properly) is from 1.50V to 5.25V. Vbus = (1.20 + N*0.001)V. N valid range: 0000h to 0FFFh.	!
$zAA<CR>	W	Pull-up Resistors Four switched pull-up resistors are available on both the SDA and SCL signals. Argument is 4-bits. The bit definitions are as follows: -- bit 7 – bit 4 => N/A -- bit 3 => 4.99K 1= enabled -- bit 2 => 2.21K 1= enabled -- bit 1 => 1.00K 1= enabled -- bit 0 => 499 ohms 1= enabled	!

2.3.3 Commands – Read

Note: AA = Hexadecimal text string

<CR> = Carriage Return (\r)

Syntax	R/W	Description	Valid Response
$qAAAA<CR>	W	Read (Rx) I2C Data Command This command is used to read data from an I2C slave device. The command can read from 0 to 255 bytes. The first two characters following the command character specifies the number of bytes to read. The next two characters specifies the Slave Address. Captured I2C data is stored in the Rx Buffer and read out using the Read Rx Buffer command. • Read Byte Count: Range = 00h – FFh. • Slave Address: Range = Odd addresses from 01h to FFh. The upper 7-bits of the byte are the address code and the LSB is the direction (1= read). Response A valid response includes a status byte followed by a "valid command" delimiter (!). This response will be returned within 10ms, message complete or not. The status byte bit definitions are as follows: Bit 7 Bit 6 0 0 Rx in process (timer time-out) -- bit 5 = > MM mode busy 1= true -- bit 4 = > TBD -- bit 3 = > Clock Stretch 1*= true -- bit 2 = > TBD -- bit 1 => TBD -- bit 0 => Bus is not free 1= true Bit 7 Bit 6 0 1 Done with error (Rx SM) -- bit 5 = > TBD -- bit 4 = > No ACK response 1* = true -- bit 3 = > Clock Stretch 1*= true -- bit 2 = > Loss of Arbitration (bit Tx) 1* = true -- bit 1 => Loss of Arbitration (Start) 1 = true -- bit 0 => Bus is not free 1= true	AA!

		Bit 7 Bit 6 1 0 Done without error (Tx/Rx SM) -- bit 5 – bit 0 => N/A Bit 7 Bit 6 1 1 Message syntax error -- bit 5 – bit 0 => N/A * Use the Error Byte Count register to determine the data byte involved with this error.	
$dAAAA<CR>	W	Read(Rx) I2C Data – without Stop Note that this command is the same as above except that the I2C transmission is not terminated by a Stop. This command is used to create a Repeated Start event .	AA!
$r<CR>	R	Read Rx Buffer Read data stored in Rx buffer. Data stored in this buffer is that read after the Read I2C Data command. This Buffer has a maximum depth of 255 bytes.	…AAAAA!
$c<CR>	R	Rx Buffer Byte Count This register indicates the number of bytes stored in the Rx Buffer. Value range is from 00h to ffh.	AA!
$t<CR>	R	General Status Register This 8-bit register contains current host adapter status. Register bit definitions are as follows: -- bit 7 => Tx/Rx SM idle -- bit 6 => bus is not free -- bit 5 => MM mode busy 1= true -- bit 4 => Rx CLK stretch -- bit 3 => Stop CLK stretch -- bit 2 => Bus Voltage over-current 1 = Over-current condition* -- bit 1 => SDA State 1 = high -- bit 0 => SCL State 1 = high * Note: When an over-current condition has occurred, the event is latched and the external bus voltage is disabled. Reading the General Status register clears the event flag.	AA!

$b<CR>	R	Transaction Status register This 8-bit register contains the results of the last Tx or Rx command. Register bit definitions are as follows: -- bit 7 => Tx/Rx transaction complete -- bit 6 => Tx Memory Overflow -- bit 5 => TBD -- bit 4 => No ACK -- bit 3 => CLK stretch error -- bit 2 => 1) Normal Mode : Bus Contention during Tx 2) MM Mode : Loss of arbitration during Tx -- bit 1 => 1) Normal Mode: Bus Contention at Start 2) MM Mode : Loss of arbitration at Start -- bit 0 => Bus is not free	AA!
$e<CR>	R	Error Byte Count This register indicates the byte count when an error occurs during either an I2C bus Tx or Rx operation. This register is read when the Tx/Rx Status Register indicates that an error has occurred due to either 1) a loss of arbitration during bit Tx (bit 4); 2) Clock Stretch (bit 5); or 3) No ACK response (bit 6).	AA!
$v<CR>	R	Version Register 1st two characters : HW version 2nd two characters : VHDL version	AAAA!

2.4 Programming Example using HyperTerminal

In this simple example, the JI-300 will be programmed to read and write data to a target I2C device (M24C04 EEPROM) using Microsoft's HyperTerminal, a terminal emulator program.

The general I2C setup is as follows:

- SCL Frequency: 100KHz
- SCL Duty Cycle: 50%
- Bus Voltage: 5.0V
- Bus Pull-up Resistors: 2.2K
- Multi-Master Mode: Disabled
- Clock Stretch Time: Infinite
- Bus-Free Wait Time: Infinite

1. Connect the M24C04 EEPROM to the JI-300 as shown in Figure 1:

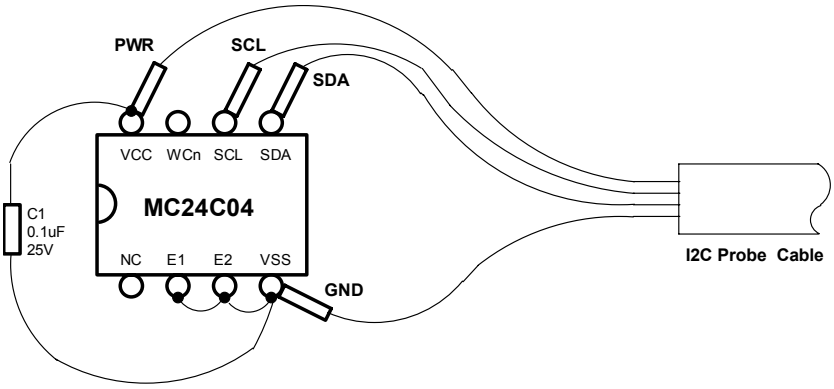

Figure 1. Connections to M24C04 I2C

2. Connect the JI-300 to the host PC using 6' USB cable.

3. After a few seconds, confirm that the JI-300 is powered by verifying that the front panel PWR LED is on. (If the LED does not illuminate, or flickers on and then goes out, begin trouble-shooting by verifying that the USB/Virtual COM port driver has been loaded on the host PC. Refer to appendix C of the JI-300 User's Manual for details.)

4. Open HyperTerminal and configure as follows:
   - BAUD rate: 115.2K
   - Data bits: 8
   - Parity: None
   - Stop bits: 1
   - Flow Control: None

- Emulation: VT100
- Connect using: COMx (JI-300 virtual com port)

5. Begin a session by lifting the receiver (call) and sending the following configuration commands. Typed commands will not be visible at the command line since the JI-300 does not echo transmitted data. Note that it is only necessary to send these commands once at the beginning of a session.

    a. Send a Halt command to ensure all JI-300 processes have stopped.
        - $s<CR> valid response = !

    b. Set SCL = 100Khz @ 50% duty cycle
        1. tHIGH = 5us
        2. tSU;DAT = 2.5uS
        3. tHD;DAT = 2.5uS
            - $g00f4<CR> valid response = !
            - $u007a<CR> valid response = !
            - $h007a<CR> valid response = !

    c. Set Bus Voltage = 5.00V
        - $i0ed8<CR> valid response = !

    d. Set bus pull-up resistors = 2.2K
        - $z04<CR> valid response = !

    e. Configure behavior (and turn-on external bus voltage)
        - $m8b<CR> valid response = !

6. Send I2C commands to the EEPROM:

    a. Write the ASCII data “Hello” (48h, 65h, 6ch, 6ch, 6fh) beginning at address 00h.
        - $w07a00048656c6c6f<CR> valid response = 80!

    b. Read 5 bytes from the EEPROM beginning at address 00h.
        1. Per EEPROM data sheet, send a dummy write without stop.
            - $y02a000<CR> valid response = 80!
        2. Read 5 bytes
            - $q05a1<CR> valid response = 80!
        3. Get data from buffer and verify data is an ASCII “Hello”
            - $r<CR> valid response = 48656c6c6f!

7. Done

2.5 I2C Command Execution Flowchart

The flowchart on the following page graphically details the steps required to execute an I2C command. Included are steps to configure the I2C bus (SCL clock rate, bus voltage, pull-up resistors, etc.), steps to handle an excessively long clock-stretch event, and details on processing a transaction anomaly or bus error.

I2C Command Execution Flowchart