Model JI-4040
USB Multifunction Digital I/O Module - Isolated

Programmer's Interface Document

Version 1.2

Jupiter Instruments

5/6/2011 Edition

www.jupiteri.com

TABLE OF CONTENTS

1. INTRODUCTION

This document details the programming interface of the JI-4040 for application developers. The USB interface hardware, JI-4040 command set, and code examples are presented.

2. HOST COMMUNICATION

2.0 USB Interface

Communication with the JI-4040 is by way of a USB connection. The JI-4040 incorporates a USB IC (FT232R 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 (Vista, XP, 2000, 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

- FT232R USB Interface IC from FTDI
  http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
- Data Transfer Rate:
  - 1Mbit/sec – D2XX Direct Driver
  - 1Mbit/sec - VCP

2.2 JI-4040 Commands

No USB-specific programming is required to control the JI-4040. Communication with the USB port is accomplished via a VCP or manufacture supplied DLL: FTD2xx.dll or FTD2xx_NET.dll. In either case, the command and response messages are comprised of simple ASCII strings. A straightforward command/response type protocol ensures that all transmissions to the JI-4040 are acknowledged. A "!" response indicates a valid command message, and "?" indicates invalid.

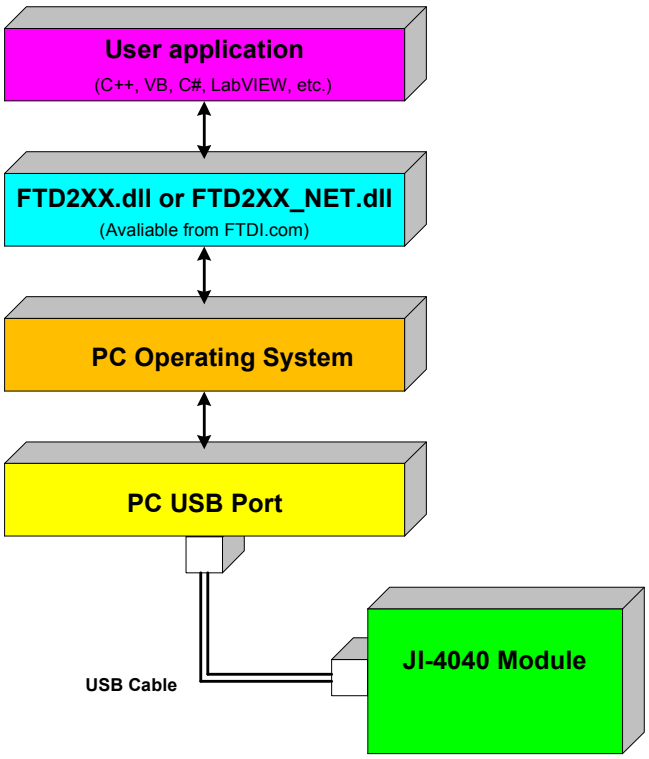

Figure 1. Communication Flow Diagram

2.2.1 **Syntax**

Command structure:

[$][Command Character][Port Character][Data Argument][Carriage Return]

Each command begins with a start delimiter '$' followed by a command character and port character. Both are upper-case alpha characters. Following is the Data Argument. The data length can range from 0 to 8 bytes, depending on the command type. Note that not all commands have arguments. A carriage return terminates all commands.

Response structure:

[Argument string][!]

A response is generated for all valid or invalid commands sent to the JI-4040 module. A '!' character is returned for all valid commands, and a single '?' response indicates an invalid command has been received. Depending on the command type, a single character or character string may precede the '!'.

Command/Response Examples:

Command	Response	Description
$WB55<CR>	!	Write data 55h to port B
$RC<CR>	3b!	Read data 3bh from port C
$YY3f6b9af1<CR>	!	Write 3fh to port D, 6bh to port C, 9ah to port B, and f1h to port A.
$R5<CR>	?	Invalid command

2.2.2 **Initialization**

The JI-4040 must be initialized once at the beginning of a session following a USB PC attachment. Typically the JI-4040 is initialized following a JI-4040 **Open()** call. See the example below for details.

C# Code Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();

        //----------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init()
        {
            if (JI4040_Device.Open() != 0)
            {
                // Error: Open //
                return false;
            }
            else if(JI4040_Device.Init() != true)
            {
                // Error: Init //
                return false;
            }
            else
            {
                // Success: Open and Init //
                // JI-4040 Ready for use //
                return true;
            }
        }
        //----------------------------------------------------------------------//

        //----------------------------------------------------------------------//
        private void openButton1_Click(object sender, EventArgs e)
        {
            if (USBopen_Init() == true)
            {
                statusTextBox1.Text = "Open";
            }
            else
            {
                statusTextBox1.Text = "Error: Open";
            }
        }
        //----------------------------------------------------------------------//
    }
}
```

2.2.3 **Command Set**

This section includes the complete command set for the JI-4040. Code examples make use of the `JI_4040_NET` class. A copy of this file can be found in Apendix A.

Legend	
aa or aa	8-bits ASCII HEX value – lower case
aaaa or aaaa	16-bit ASCII HEX value – lower case
aaaaaaaa or aaaaaaaa	32-bit ASCII HEX value – lower case
x	Port Character – upper case
<CR>	Carriage Return – control character "\r"
aa or aa	ASCII HEX value of a text character
. . . .	Hidden Code

2.2.3.1 Data Direction Port A (DA*aa*)
2.2.3.2 Data Direction Port B (DB*aa*)
2.2.3.3 Data Direction Port C (DC*aa*)
2.2.3.4 Data Direction Port D (DD*aa*)
2.2.3.5 Data Direction Port E (DE*aa*)
2.2.3.6 Data Direction Port F (DF*aa*)

Syntax: $Dx*aa*<CR> (x refers to selected port A, B, C, D, E, or F)

Direction: Write

Argument: *aa* 8-bits (ASCII HEX value)

Reset Value: 00h (read direction)

Response: ! = success
? = syntax error

Description: This command sets the Read/Write direction of the four 8-bit ports (A, B, C, D) and two 2-bit ports (E and F). Argument data 00h = Read, and ffh = Write.

Command Example:

Command	Response	Description
$DAff<CR>	!	Set port A to write
$DA00<CR>	!	Set port A to read
$DEff<CR>	!	Set port E to write
$DFff<CR>	!	Set port F to write

C# Code Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1() . . . .

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();


        //-------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init(). . . .
        //-------------------------------------------------------------------//

        //-------------------------------------------------------------------//
        // USB Open - Button click
        //
        private void openButton1_Click(object sender, EventArgs e) . . . .
        //-------------------------------------------------------------------//

        //-------------------------------------------------------------------//
        // Set I/O Direction and Read/Write Port Data
        // Return: True if successful
        //
        private bool setIODir_RWData()
        {
            const string DIR_INPUT = "00";
            const string DIR_OUTPUT = "ff";

            // Set Ports A, C, and E as Inputs //
            if (JI4040_Device.Write("DA" + DIR_INPUT) != true ||
                JI4040_Device.Write("DC" + DIR_INPUT) != true ||
                JI4040_Device.Write("DE" + DIR_INPUT) != true  )
            {
                // Error //
                return false;
            }
            // Set Ports B, D, and F as Outputs //
            if (JI4040_Device.Write("DB" + DIR_OUTPUT) != true ||
                JI4040_Device.Write("DD" + DIR_OUTPUT) != true ||
                JI4040_Device.Write("DF" + DIR_OUTPUT) != true)
            {
                // Error //
                return false;
            }

            string strPortA, strPortC, strPortE;

            // Read Ports A, C, and E //
            if (JI4040_Device.Read("RA", out strPortA) != true ||
                JI4040_Device.Read("RC", out strPortC) != true ||
                JI4040_Device.Read("RE", out strPortE) != true)
            {
                // Error //
                return false;
            }

            // Write data:
```

```
                    //              Port A -> Port B
                    //              Port C -> Port D
                    //              Port E -> PortF
                    if (JI4040_Device.Write("WB" + strPortA) != true ||
                        JI4040_Device.Write("WD" + strPortC) != true ||
                        JI4040_Device.Write("WF" + strPortE) != true)
                    {
                        // Error //
                        return false;
                    }

                    return true;
                }
                //-------------------------------------------------------------------//

                //-------------------------------------------------------------------//
                // R/W Port - Button click
                //
                private void rwPortsButton_Click(object sender, EventArgs e)
                {
                    if (setIODir_RWData() == true)
                    {
                        statusTextBox1.Text = "Success: R/W Port";
                    }
                    else
                    {
                        statusTextBox1.Text = "Error: R/W Port";
                    }
                }
                //-------------------------------------------------------------------//
            }
        }
    //-----------------------------------------------------------------------------//
```

2.2.3.7 Write Port A (WA*aa*)

2.2.3.8 Write Port B (WB*aa*)

2.2.3.9 Write Port C (WC*aa*)

2.2.3.10 Write Port D (WD*aa*)

2.2.3.11 Write Port E (WE*aa*)

2.2.3.12 Write Port F (WF*aa*)

Syntax: $W*x**aa*<CR> (*x* refers to selected port A, B, C, D, E, or F)

Direction: Write

Argument: *aa* 8-bits (ASCII HEX value)

Reset. Value: 00h

Response: ! = success
? = syntax error

Description: This command writes 8-bits of data to individual ports A, B, C and D, and 2-bits of data to ports E and F. For ports E and F, the lower two bits of the data byte map to the lower two port pins. Data bits 7 – 2 are ignored.

8-bit argument data is mapped to the port pins as follows:

Bit 7 = Port pin 7
Bit 6 = Port pin 6
Bit 5 = Port pin 5

Bit 4 = Port pin 4

Bit 3 = Port pin 3
Bit 2 = Port pin 2
Bit 1 = Port pin 1
Bit 0 = Port pin 0

Command Example:

Command	Response	Description
$WB55<CR>	!	Write 55h to port B
$WCc7<CR>	!	Write c7h to port C
$WDf3<CR>	!	Write f3h to port D
$WF03<CR>	!	Write 3h to port F (port F is a 2-bit port)
$WFff<CR>	!	Write ffh to port F (upper bits (7 – 2) ignored)

C# Code Example:

See code example in section 2.2.3.1 - 2.2.3.6 (above)

2.2.3.13 Write Ports ABCD (YY*aaaaaaaa*)

Syntax: $YY*aaaaaaaa*<CR>

Direction: Write

Argument: *aaaaaaaa* 32-bits (ASCII HEX value)

Reset. Value: 00000000h

Response: ! = success
? = syntax error

Description: This command writes 32-bits of data concurrently to ports A, B, C, and D.

32-bit argument data is mapped to the port pins as follows:

Bit 31 = Port D, pin 8
Bit 30 = Port D, pin 7
Bit 29 = Port D, pin 6
Bit 28 = Port D, pin 5
Bit 27 = Port D, pin 4
Bit 26 = Port D, pin 3
Bit 25 = Port D, pin 2
Bit 24 = Port D, pin 1

Bit 23 = Port C, pin 8
Bit 22 = Port C, pin 7
Bit 21 = Port C, pin 6
Bit 20 = Port C, pin 5
Bit 19 = Port C, pin 4
Bit 18 = Port C, pin 3
Bit 17 = Port C, pin 2
Bit 16 = Port C, pin 1

Bit 15 = Port B, pin 8
Bit 14 = Port B, pin 7
Bit 13 = Port B, pin 6
Bit 12 = Port B, pin 5
Bit 11 = Port B, pin 4
Bit 10 = Port B, pin 3
Bit 9 = Port B, pin 2
Bit 8 = Port B, pin 1

Bit 7 = Port A, pin 8
Bit 6 = Port A, pin 7
Bit 5 = Port A, pin 6
Bit 4 = Port A, pin 5
Bit 3 = Port A, pin 4
Bit 2 = Port A, pin 3
Bit 1 = Port A, pin 2
Bit 0 = Port A, pin 1

Command Example:

Command	Response	Description
$YY3f6b9af1<CR>	!	Write 3fh to port D, 6bh to port C, 9ah to port B, f1h to port A
$YYb6734cd6<CR>	!	Write b6h to port D, 73h to port C, 4ch to port B, d6h to port A

C# Code Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1() . . . .

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();


        //------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init(). . . .
        //------------------------------------------------------------------//

        //------------------------------------------------------------------//
        // USB Open - Button click
        //
        private void openButton1_Click(object sender, EventArgs e) . . . .
        //------------------------------------------------------------------//

        //------------------------------------------------------------------//
        // Set I/O Direction and Read/Write Port 32-bit Data
        // Return: True if successful
        //
        private bool setIODir_RWData32bit()
        {
            const string DIR_INPUT = "00";
            const string DIR_OUTPUT = "ff";

            // Set Ports A & C as Inputs //
            if (JI4040_Device.Write("DA" + DIR_INPUT) != true ||
                JI4040_Device.Write("DC" + DIR_INPUT) != true)
            {
                // Error //
                return false;
            }
            // Set Ports B & D as Outputs //
            if (JI4040_Device.Write("DB" + DIR_OUTPUT) != true ||
                JI4040_Device.Write("DD" + DIR_OUTPUT) != true )
            {
                // Error //
                return false;
            }

            string strPortABCD;

            // Read Ports A, B, C, and D //
            if (JI4040_Device.Read("ZZ", out strPortABCD) != true)
            {
                // Error //
                return false;
```

```
                    }

                    // Write data:
                    //             Port A -> Port B
                    //             Port C -> Port D
                    if (JI4040_Device.Write("YY" + "00" + strPortABCD.Substring(0, 2) +
                        "00" + strPortABCD.Substring(4, 2)) != true)
                    {
                        // Error //
                        return false;
                    }
                    return true;
                }
                //---------------------------------------------------------------------//

                //---------------------------------------------------------------------//
                // R/W Port - 32-Bit Data - Button click
                //
                private void rwPorts32Button_Click(object sender, EventArgs e)
                {
                    if (setIODir_RWData32bit() == true)
                    {
                        statusTextBox1.Text = "Success: R/W Port 32-bit";
                    }
                    else
                    {
                        statusTextBox1.Text = "Error: R/W Port 32-bit";
                    }
                }
                //---------------------------------------------------------------------//
            }
        }
        //-------------------------------------------------------------------------------//
```

2.2.3.14 Read Port A (RA)

2.2.3.15 Read Port B (RB)

2.2.3.16 Read Port C (RC)

2.2.3.17 Read Port D (RD)

2.2.3.18 Read Port E (RE)

2.2.3.19 Read Port F (RF)

Syntax: $Rx<CR> (x refers to selected port A, B, C, D, E, or F)

Direction: Read

Argument: None

Reset. Value: None

Response: *aa*! = success, 8 bits
? = syntax error

Description: This command reads byte data ports A, B, C and D, and 2- bit data from ports E and F. For ports E and F, the lower two bits of the data byte map to the lower two port pins. Data bits 7 – 2 are ignored.

Port pins mapping is as follows:

Bit 7 = Port pin 7
Bit 6 = Port pin 6
Bit 5 = Port pin 5
Bit 4 = Port pin 4

Bit 3 = Port pin 3
Bit 2 = Port pin 2
Bit 1 = Port pin 1
Bit 0 = Port pin 0

Command Example:

Command	Response	Description
$RB<CR>	5c!	Read 8-bits from port B. (port B = 5ch)
$RC<CR>	63!	Read 8-bits from port C. (port C = 63h)
$RD<CR>	d7!	Read 8-bits from port D. (port D = d7h)
$RE<CR>	02!	Read 2-bits from port E. (port E = 2h)

C# Code Example:

See code example in section 2.2.3.1 - 2.2.3.6 (above)

2.2.3.20 Read Ports ABCD (ZZ)

Syntax: $ZZ<CR>

Direction: Read

Argument: None

Reset Value: None

Response: *aaaaaaaa*! = success, 32 bits
? = syntax error

Description: This command reads 32-bits of data concurrently from ports A, B, C, and D

Port pins mapping is as follows:

Bit 31 = Port D, pin 8
Bit 30 = Port D, pin 7
Bit 29 = Port D, pin 6
Bit 28 = Port D, pin 5
Bit 27 = Port D, pin 4
Bit 26 = Port D, pin 3
Bit 25 = Port D, pin 2
Bit 24 = Port D, pin 1

Bit 23 = Port C, pin 8
Bit 22 = Port C, pin 7
Bit 21 = Port C, pin 6
Bit 20 = Port C, pin 5
Bit 19 = Port C, pin 4
Bit 18 = Port C, pin 3
Bit 17 = Port C, pin 2
Bit 16 = Port C, pin 1

Bit 15 = Port B, pin 8
Bit 14 = Port B, pin 7
Bit 13 = Port B, pin 6
Bit 12 = Port B, pin 5
Bit 11 = Port B, pin 4
Bit 10 = Port B, pin 3
Bit 9 = Port B, pin 2
Bit 8 = Port B, pin 1

Bit 7 = Port A, pin 8
Bit 6 = Port A, pin 7
Bit 5 = Port A, pin 6
Bit 4 = Port A, pin 5
Bit 3 = Port A, pin 4
Bit 2 = Port A, pin 3
Bit 1 = Port A, pin 2
Bit 0 = Port A, pin 1

Command Example:

Command	Response	Description
$ZZ<CR>	2c31635c!	Read 32-bits from ports A – D (port A = 2ch, port B = 31h, port C = 63h, port D = 5ch)
$ZZ<CR>	cdf21367!	Read 32-bits from ports A – D (port A = cdh, port B = f2h, port C = 13h, port D = 67h)

C# Code Example:

See code example in section 2.2.3.13 (above)

2.2.3.21 Clock Prescalar - Port G (KG*aa*)

2.2.3.22 Clock Prescalar - Port H (KH*aa*)

Syntax: $K*x**aa*<CR> (*x* refers to selected port G or H)

Direction: Write

Argument: *aa* 8-bits

Reset Value: 00h

Response: ! = success
? = syntax error

Description: This register sets the frequency of the clock that drives the 16-bit TH and TL counters and the clock tick for the Pulse/Period timer. The frequency of the clock is as follows:

N_{PRE} = Divisor Count, Valid range 00h to ffh
F_{PRE} = Pre-scalar Clock Frequency

$F_{PRE} = 10*(1/(N_{PRE} + 1))$ MHz,
F_{PRE} Range: 10MHz to 39.06KHz

Command Example:

Command	Response	Description
$KH00<CR>	!	Sets port H F_{PRE} = 10 MHz
$KG09<CR>	!	Sets port G F_{PRE} = 1 MHz
$KG63<CR>	!	Sets port G F_{PRE} = 100 KHz

C# Code Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1() . . . .

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();


        //-------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init(). . . .
        //-------------------------------------------------------------------//

        //-------------------------------------------------------------------//
        // USB Open - Button click
        //
        private void openButton1_Click(object sender, EventArgs e) . . . .
        //-------------------------------------------------------------------//


        //-------------------------------------------------------------------//
        // Configure Port G as a 10KHz, 50% duty cycle, Clock source
        // Return: True if successful
        //
        private bool setPortG_10KHz_CLK()
        {
             // Configure Ports G as a CLK source //
            if (JI4040_Device.Write("CG" + "20") != true)
            {
                // Error //
                return false;
            }

            // Set G Prescalar to divide-by 10. Resolution is 1 us (1MHz).  //
            if (JI4040_Device.Write("KG" + "09") != true)
            {
```

```
                    // Error //
                    return false;
                }

                // Set Port G TH = TL = 50.
                // Signal period = 100uS @ 50% duty cycle.  //
                if (JI4040_Device.Write("HG" + "0031") != true ||
                    JI4040_Device.Write("NG" + "0031") != true )
                {
                    // Error //
                    return false;
                }

                // Start CLK //
                if (JI4040_Device.Write("GG") != true)
                {
                    // Error //
                    return false;
                }

                return true;
            }
            //-----------------------------------------------------------------//

            //-----------------------------------------------------------------//
            // 10KHz CLK Source Port G - Button click
            //
            private void PortG_10KHzCLK_Button_Click(object sender, EventArgs e)
            {
                if (setPortG_10KHz_CLK() == true)
                {
                    statusTextBox1.Text = "Success: 10KHz CLK Port G";
                }
                else
                {
                    statusTextBox1.Text = "Error: 10KHz CLK Port G";
                }
            }
            //-----------------------------------------------------------------//

        }
    }
//---------------------------------------------------------------------------//
```

2.2.3.23 Timer High (TH) - Port G (HG*aaaa*)

2.2.3.24 Timer High (TH) - Port H (HH*aaaa*)

Syntax: $H**x***aaaa*<CR> (**x** refers to selected port G or H)

Direction: Write

Argument: ***aaaa*** 16-bits

Reset Value: 0000h

Response: ! = success
? = syntax error

Description: This register sets the high period of the clock signal in Clock Generator mode and sets the period of the one-shot in Pulse Generator mode. Period is calculated as follows:

N_{PRE} = Clock Pre-Scalar
N_{TH} = TH Register Count, Valid range 0000h to ffffh

$TH = (N_{TH} + 1) * (N_{PRE} + 1) * (0.1)uS,$

Command Example:

Command	Response	Description
$HG0000<CR>	!	Sets port G TH = 0.1us (if port G F_{PRE} = 10 MHz)
$HH0009<CR>	!	Sets port H TH = 1us (if port H F_{PRE} = 10 MHz)
$HH0063<CR>	!	Sets port H TH = 10us (if port H F_{PRE} = 10 MHz)
$HH0063<CR>	!	Sets port H TH = 100us (if port H F_{PRE} = 1 MHz)

C# Code Example:

See code example in section 2.2.3.22 (above)

2.2.3.25 Timer Low (TL) - Port G (NG*aaaa*)

2.2.3.26 Timer Low (TL) - Port H (NH*aaaa*)

Syntax: $Nx*aaaa*<CR> (x refers to selected port G or H)

Direction: Write

Argument: *aaaa* 16-bits

Reset Value: 0000h

Response: ! = success
? = syntax error

Description: This register sets the Low period of the clock signal in Clock Generator mode. Period is calculated as follows:

N_{PRE} = Clock Pre-Scalar
N_{TL} = TL Register Count, Valid range 0000h to ffffh

$TL = (N_{TL} + 1) * (N_{PRE} + 1) * (0.1)uS,$

Command Example:

Command	Response	Description
$NG0000<CR>	!	Sets port G TL = 0.1us (if port G F_{PRE} = 10 MHz)
$NH0009<CR>	!	Sets port H TL = 1us (if port H F_{PRE} = 10 MHz)
$NH0063<CR>	!	Sets port H TL = 10us (if port H F_{PRE} = 10 MHz)
$NH0063<CR>	!	Sets port H TL = 100us (if port H F_{PRE} = 1 MHz)

C# Code Example:

See code example in section 2.2.3.22 (above)

2.2.3.27 Configuration Register - Port G (CG*aa*)

2.2.3.28 Configuration Register - Port H (CH*aa*)

Syntax: $C*x**aa*<CR> (*x* refers to selected port G or H)

Direction: Write

Argument: *aa* 8-bits

Reset Value: 00h

Response: ! = success
? = syntax error

Description: This register selects the functional mode of the Special Function Port (SFP). Functions include:
1) Clock or Pulse Generator
2) Timer to measure CLK periods or pulse widths
3) Event counter

See the table below for configuration details.

Argument Byte	Selected Function	Operation Details
00h	Read	
10h	Write	
20h	CLK Gen	CLK source
21h	Pulse Gen	One-Shot Pulse
30h	Period Timer	Positive, rising-edge to rising-edge
31h	Period Timer	Period, falling-edge to falling-edge
32h	Pulse Timer	Pulse, rising-edge to falling-edge
33h	Pulse Timer	Pulse, falling-edge to rising-edge
40h	Event Counter	Rising-edge, D0 = X
41h	Event Counter	Falling-edge, D0 = X
44h	Event Counter	Rising-edge, Enable when D0 = 0
45h	Event Counter	Falling-edge, Enable when D0 = 0
46h	Event Counter	Rising-edge, Enable when D0 = 1
47h	Event Counter	Falling-edge, Enable when D0 = 1

Command Example:

Command	Response	Description
$CG20<CR>	!	Configure SFP G as a CLK source
$CH41<CR >	!	Configure SFP H as an Event Counter that counts falling edge events.
$CH46<CR>	!	Configures SFP H as an Event Counter that counts rising edge events when D0 =1.

C# Code
Example:

See code example in section 2.2.3.22 (above)

2.2.3.29 Start - Port G (GG)

2.2.3.30 Start - Port H (GH)

Syntax: $Gx<CR> (x refers to selected port G or H)

Direction: Write

Argument: None

Reset Value: N/A

Response: ! = success
? = syntax error

Description: This command starts the function selected by the Configuration Register.

Command
Example:

Command	Response	Description
$GG<CR >	!	Start Port G function
$GH<CR >	!	Start Port H function

C# Code
Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1() . . . .

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();


        //------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init(). . . .
        //------------------------------------------------------------------//

        //------------------------------------------------------------------//
        // USB Open - Button click
        //
        private void openButton1_Click(object sender, EventArgs e) . . . .
        //------------------------------------------------------------------//
```

```
//-----------------------------------------------------------------//
// Configure Port H as a positive-edge pulse timer with a 1uS resolution
// Return: True if successful
//
private bool configPortH_PosEdgePulseTimer()
{
    // Configure Port H as a positive-edge pulse timer //
    if (JI4040_Device.Write("CH" + "32") != true)
    {
        // Error //
        statusTextBox1.Text = "Error: Port H Write!";
        return false;
    }

    // Set H Prescalar to divide-by 10. Resolution is 1 us (1MHz).  //
    if (JI4040_Device.Write("KH" + "09") != true)
    {
        // Error //
        statusTextBox1.Text = "Error: Port H Write!";
        return false;
    }

    return true;
}
//-----------------------------------------------------------------//

//-----------------------------------------------------------------//
// Configure Port H as a positive-edge pulse timer - Button click
//
private void configPortG_PTimerButton_Click(object sender, EventArgs e)
{
    if (configPortH_PosEdgePulseTimer() == true)
    {
        statusTextBox1.Text = "Port G Pulse Timer configured";
    }
    else
    {
        // Error //
        statusTextBox1.Text = "Error: Port H Write!";
    }
}
//-----------------------------------------------------------------//

 //-----------------------------------------------------------------//
// Start Measurement Port H Pulse Timer - Button click
//
private void run_portG_PulseTimerButton_Click(object sender, EventArgs e)
{
    // Send Start Command //
    if (JI4040_Device.Write("GH") != true)
    {
        // Error //
        statusTextBox1.Text = "Error: Port H Write!";
        return;
    }
    else
    {
        statusTextBox1.Text = "Measurement in Progress";
        this.Refresh();

        // Check for measurement done //
        string statusStr;
        for (int i = 0; i < 200; ++i)
        {
            // Sleep for 1 ms //
            Thread.Sleep(1);
            if(JI4040_Device.Read("UH", out statusStr) != true)
            {
                // Error //
                statusTextBox1.Text = "Error: Port H Read!";
                return;
            }
            // Check Status:  Bit 2 = Timer Overflow   1 = Overflow
            //                Bit 1 = Timer Data RDY   1 = Data ready
```

```
                                statusStr = statusStr.Substring(1, 1);
                                int j = Convert.ToByte(statusStr, 16);
                                j = j & 0x06;

                                // Data Ready? //
                                if ((j & 0x02) > 0)
                                {   // Yes. Check for Overflow //
                                    if ((j & 0x04) > 0)
                                    {
                                        statusTextBox1.Text = "Measurement Overflow!";
                                        return;
                                    }
                                    else
                                    {
                                        // Read Timer Data //
                                        string dataStr;
                                        if (JI4040_Device.Read("JH", out dataStr) != true)
                                        {
                                            // Error //
                                            statusTextBox1.Text = "Error: Port H Read!";
                                            return;
                                        }
                                        j = Convert.ToUInt16(dataStr, 16);
                                        // Print Measurement //
                                        statusTextBox1.Text = "Pulse Width = " +
                                                        j.ToString() + "uS";
                                        return;
                                    }
                                }
                            }
                            // Measurement Timeout //
                            statusTextBox1.Text = "Measurement Time-out!";

                            // Send Stop Command //
                            if (JI4040_Device.Write("PH") != true)
                            {
                                // Error //
                                statusTextBox1.Text = "Error: Port H Write! ";
                                return;
                            }

                        }
                    }
                    //-------------------------------------------------------------------//

                }
            }
        //-----------------------------------------------------------------------//
```

2.2.3.31 Stop - Port G (PG)

2.2.3.32 Stop - Port H (PH)

Syntax: $Px<CR> (x refers to selected port G or H)

Direction: Write

Argument: None

Reset Value: N/A

Response: ! = success
? = syntax error

Description: This command halts all activity on selected port.

Command Example:

Command	Response	Description
$PG<CR >	!	Stop Port G function
$PH<CR >	!	Stop Port H function

C# Code Example:

See code example in section 2.2.3.30 (above)

2.2.3.33 Timer/Event Register - Port G (JG)

2.2.3.34 Timer/Event Register - Port H (JH)

Syntax: $Jx<CR> (x refers to selected port G or H)

Direction: Read

Argument: None

Reset Value: 0000h

Response: *aaaa*! = success, 16 bits
? = syntax error

Description: This command reads the contents of the SFP Timer/Event register. The count data is 16-bits in length.

Command Example:

Command	Response	Description
$JG<CR>	635c!	Read 16-bit data from Timer/Event register, port G.
$JH<CR>	c389!	Read 16-bit data from Timer/Event register, port H.

C# Code Example:

See code example in section 2.2.3.30 (above)

2.2.3.35 Port Status Register - Port G (UG)

2.2.3.36 Port Status Register - Port H (UH)

Syntax: $Ux<CR> (x refers to selected port G or H)

Direction: Read

Argument: None

Reset Value: N/A

Response: *aa*! = success, 8 bits
? = syntax error

Description: This register indicates the status of the selected SFP.

Status mapping is as follows:

Bit 7 = TBD	
Bit 6 = TBD	
Bit 5 = TBD	
Bit 4 = Event Counter Overflow	1 = Overflow
Bit 3 = Event Counter Data RDY	1 = Data ready
Bit 2 = Pulse/Period Timer Overflow	1 = Overflow
Bit 1 = Pulse/Period Timer Data RDY	1 = Data ready
Bit 0 = CLK/One-Shot Stop	0 = Stopped/Done, 1 = Running

Command Example:

Command	Response	Description
$UG<CR>	01!	One-shot done for port G.
$UH<CR>	02!	Period timer done for port H.

C# Code Example:

See code example in section 2.2.3.30 (above)

2.2.3.37 Version Register (VV)

Syntax: $VV<CR>

Direction: Read

Argument: None.

Response: *aaaa*! = success
? = syntax error

aa = HW revision
aa = VHDL version

Description: This command returns the hardware and VHDL versions of the JI-4040 module. The first two return characters represent the hexadecimal value of the HW revision and the last two represent the VHDL version.

Command Example:

Command	Response	Description
$WW<CR>	3133!	Translation: HW Ver 1, VHDL Ver 3
$WW<CR>	4139!	Translation: HW Ver A, VHDL Ver 9

C# Code Example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    public partial class Form1 : Form
    {
        public Form1() . . . .

        // Create new instance of the JI_4040_NET device class //
        JI_4040_NET JI4040_Device = new JI_4040_NET();


        //------------------------------------------------------------------//
        // Open USB Port and Initilize JI-4040
        // Return: True if successful
        //
        private bool USBopen_Init(). . . .
        //------------------------------------------------------------------//

        //------------------------------------------------------------------//
        // USB Open - Button click
        //
        private void openButton1_Click(object sender, EventArgs e) . . . .
        //------------------------------------------------------------------//

        //------------------------------------------------------------------//
        // About - Button click
        //
        private void aboutButton_Click(object sender, EventArgs e)
        {
            //Print HW Rev and VHDL Ver //

            string str1;
            if (JI4040_Device.Read("VV", out str1) != true)
            {
                // Error //
                statusTextBox1.Text = "Error: About Read!";
                return;
            }

            string HWchar = str1.Substring(0, 2);
            int i = Convert.ToUInt16(HWchar, 16);
            char c = (char)i;
            HWchar = c.ToString();

            string VHDLchar = str1.Substring(2, 2);
            i = Convert.ToUInt16(VHDLchar, 16);
            c = (char)i;
            VHDLchar = c.ToString();

            statusTextBox1.Text = "HW Rev: " + HWchar +
                                  ", VHDL Ver: " + VHDLchar;
        }
        //------------------------------------------------------------------//


    }
}
//----------------------------------------------------------------------//
```

APPENDIX A

1. JI_4040_NET Class File

The latest version of this file can be found on the Jupiter Instruments website (www.jupiteri.com).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Threading;
using FTD2XX_NET;

//-----------------------------------------------------//
// 3/10/2011 Initial release
//-----------------------------------------------------//
// 4/2/2011
//         1. Several general modifications
//         2. Clean-up comments
//-----------------------------------------------------//

namespace C_sharp_Test_Code_JI_4040_Rev1
{
    class JI_4040_NET
    {
        // Create new instance of the FTDI device class
        FTDI FTDIDevice = new FTDI();

        UInt32 FTDIDeviceCount = 0;
        FTDI.FT_STATUS ftStatus = FTDI.FT_STATUS.FT_OK;



        //-------------------------------------------------------------------------------//
        public bool setPortDirection(string port, string dData)
        {
            // Ensure correct case //
            port = port.ToUpper();     // Port Data upper case
            dData = dData.ToLower();   // Direction data lower case

            // Write Data //
            string DIRData = "D" + port + dData;

            // Execute and return
            return Write(DIRData);
        }
        //-------------------------------------------------------------------------------//


        //-------------------------------------------------------------------------------//
        public bool writePortData(string port, string pData)
        {
            // Ensure correct case //
            port = port.ToUpper();     // Port upper case
            pData = pData.ToLower();   // Data lower case

            // Write Data //
            string writeData = "W" + port + pData;

            // Execute and return
            return Write(writeData);
        }
        //-------------------------------------------------------------------------------//


        //-------------------------------------------------------------------------------//
        public bool writePortData32(string pData)
        {
```

```
        // Ensure correct case //
        pData = pData.ToLower();   // Direction data lower case

        // Write Data //
        string writeData = "$" + "YY" + pData + "\r";

        // Execute and return
        return Write(writeData);
    }
    //-------------------------------------------------------------------------------//


    //-------------------------------------------------------------------------------//
    public bool readPortData(string port, out string rxData)
    {
        // Ensure correct case //
        port = port.ToUpper();     // Port upper case

        // Write Data //
        string dataStr = "$" + "R" + port + "\r";

        // Execute and return status and received data //
        return Read(dataStr, out rxData);

    }
    //-------------------------------------------------------------------------------//


    //*******************************************************************************//
    // Open USB Port
    //
    // Return value:
    //  0  = Success
    //  -1 = Unknown error.
    //  -2 = No un-opened JI-4040 devices found
    //
    // Updated: 3/10/2011
    //*******************************************************************************//
    public int Open()
    {
        // Determine the number of FTDI devices connected//

        ftStatus = FTDIDevice.GetNumberOfDevices(ref FTDIDeviceCount);
        // Verify command execution  //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return -1; ;
        }

        // If no USB devices avialable, return //
        if (FTDIDeviceCount == 0)
        {
            return -2;
        }

        // Allocate storage for device info list
        FTDI.FT_DEVICE_INFO_NODE[] FTDIDeviceList = new
              FTDI.FT_DEVICE_INFO_NODE[FTDIDeviceCount];

        // Populate device list
        ftStatus = FTDIDevice.GetDeviceList(FTDIDeviceList);
        // Verify command execution  //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return -1; ;
        }

        // Find the first un-opened JI-4040 device and then open //
        string str1;
        for (uint i = 0; i < FTDIDeviceCount; ++i)
        {
            str1 = FTDIDeviceList[i].Description.ToString();
            // Check for JI-4040 device //
            if (str1.Contains("JI-4040") == true)
            {
```

```
                // JI-4040 device found //
                // Open device if it's un-opened //
                if (0 == (int)(FTDIDeviceList[i].ftHandle))
                {
                    str1 = FTDIDeviceList[i].SerialNumber;
                    ftStatus = FTDIDevice.OpenBySerialNumber(str1); // Open USB
                    if (ftStatus != FTDI.FT_STATUS.FT_OK)
                    {
                        // Error! //
                        return -1; ;
                    }
                    else
                    {
                        // Success, Port is open!!
                        return 0;
                    }
                }

            }
        }

        // No un-opened JI-4040 devices found//
        return -2;
    }
    //*****************************************************************************//


    //*****************************************************************************//
    // Close USB Port
    //
    // Return value:
    //  0  = Success
    //  -1 = Error.
    //
    // Updated: 3/10/2011
    //*****************************************************************************//
    public int Close()
    {
        ftStatus = FTDIDevice.Close();
        if (ftStatus == FTDI.FT_STATUS.FT_OK)
        {
            return 0;  // Success: Port closed
        }
        else
        {
            return -1;  // Error
        }
    }
    //*****************************************************************************//


    //*****************************************************************************//
    // Initilize JI-4040
    //
    // Return value:
    //  True  = Success
    //  False = Error.
    //
    // Updated: 3/10/2011
    //*****************************************************************************//
    public bool Init()
    {
        // Set BAUD rate to 1M //
        ftStatus = FTDIDevice.SetBaudRate(1000000);
        if (ftStatus == FTDI.FT_STATUS.FT_OK)
        {
            // Configure for 8-bit data, 2 stop bits, no parity //
            FTDIDevice.SetDataCharacteristics(8, 2, 0);
            if (ftStatus == FTDI.FT_STATUS.FT_OK)
            {
                return true;  // Success
            }
            return false;  // Error
        }
        else
```

```
        {
            return false;  // Error
        }
    }
    //*****************************************************************************//


    //*****************************************************************************//
    // General Purpose Write Command
    //
    // 1. A "$" prefix and "\r" sufix are added to the command string.
    // 2. Returns a TRUE if command executes successfully
    //
    // New: 1/13/2011
    //*****************************************************************************//
    public bool Write(string str1)
    {
        // Write Data //
        string dataBuffer = "$" + str1 + "\r";
        uint numBytesWritten = 0;
        ftStatus = FTDIDevice.Write(dataBuffer, dataBuffer.Length, ref numBytesWritten);

        // Ensure successful USB Write //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return false; // USB Error! //
        }

        uint numBytesRead = 0;
        uint RxQueue = 0;
        string rDataBuffer = "";

        // Wait for successful HW response "!" //
        for (int loop = 0; loop < 100; ++loop)
        {
            Thread.Sleep(1);

            FTDIDevice.GetRxBytesAvailable(ref RxQueue);
            if (RxQueue > 0)
            {
                ftStatus = FTDIDevice.Read(out rDataBuffer, RxQueue, ref numBytesRead);
                if (ftStatus != FTDI.FT_STATUS.FT_OK)
                {
                    return false; // USB Error!
                }
                if (rDataBuffer.Contains('!'))
                {
                    return true; // Done
                }
                else if (rDataBuffer.Contains('?'))
                {
                    return false; // HW Error!
                }
            }

        }
        return false; // Error!
    }
    //*****************************************************************************//


    //*****************************************************************************//
    // Clear UART
    //
    //
    // New: 2/7/2011
    //*****************************************************************************//
    public bool clrUART()
    {
        // Write Data //
        string dataBuffer = "\r";
        uint numBytesWritten = 0;

        ftStatus = FTDIDevice.Write(dataBuffer, dataBuffer.Length, ref numBytesWritten);
```

```
        // Ensure successful USB Write //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return false; // USB Error! //
        }

        ftStatus = FTDIDevice.Write(dataBuffer, dataBuffer.Length, ref numBytesWritten);

        // Ensure successful USB Write //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return false; // USB Error! //
        }

        uint numBytesRead = 0;
        uint RxQueue = 0;
        string rDataBuffer = "";

        // Clear Rx Buffer //
        for (int loop = 0; loop < 100; ++loop)
        {
            Thread.Sleep(1);

            FTDIDevice.GetRxBytesAvailable(ref RxQueue);
            if (RxQueue > 0)
            {
                ftStatus = FTDIDevice.Read(out rDataBuffer, RxQueue, ref numBytesRead);
                if (ftStatus != FTDI.FT_STATUS.FT_OK)
                {
                    return false; // USB Error!
                }
             }
        }
        return true;
    }
    //*************************************************************************//


    //*************************************************************************//
    // General Purpose Read Command
    //
    // 1. A "$" prefix and "\r" sufix are added to the command string.
    // 2. A TRUE return indicates sucessful execution and that recDataStr is valid
    // 3. Received data returned via recDataStr
    //
    // New: 1/23/2011
    //*************************************************************************//
    public bool Read(string CMDstr1, out string recDataStr)
    {
        recDataStr = "";

        // Write Command //
        string dataBuffer = "$" + CMDstr1 + "\r";
        uint numBytesWritten = 0;
        ftStatus = FTDIDevice.Write(dataBuffer, dataBuffer.Length, ref numBytesWritten);
        // Ensure successful USB Write //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
           return false; // USB Error! //
        }

        uint numBytesRead = 0;
        uint RxQueue = 0;
        string rDataBuffer = "";

        // Read Response//

        // Wait for successful HW response "!" //
        for (int loop = 0; loop < 100; ++loop)
        {
            // Wait //
            Thread.Sleep(1);

            // Check Buffer for data //
            FTDIDevice.GetRxBytesAvailable(ref RxQueue);
```

```
            if (RxQueue > 0)
            {
                // Yes, data is avaliable //
                // Read Buffer//
                ftStatus = FTDIDevice.Read(out rDataBuffer, RxQueue, ref numBytesRead);
                if (ftStatus != FTDI.FT_STATUS.FT_OK)
                {
                    return false; // USB Error!
                }
                else
                {
                    recDataStr += rDataBuffer;
                }

                if (recDataStr.Contains('!'))
                {
                    // Remove '!' mark //
                    recDataStr = recDataStr.TrimEnd('!');
                    return true; // Done
                }
                else if (recDataStr.Contains('?'))
                {
                   return false; // HW Error!
                }
            }

        }
        return false; // Error!
    }
    //*************************************************************************//


    //*************************************************************************//
    // Write a single character
    //
    //
    // New: 2/20/2011
    //*************************************************************************//
    public bool writeOneChar(string str1)
    {
        // Write Data //
        string dataBuffer = str1;
        uint numBytesWritten = 0;

        ftStatus = FTDIDevice.Write(dataBuffer, dataBuffer.Length, ref numBytesWritten);

        // Ensure successful USB Write //
        if (ftStatus != FTDI.FT_STATUS.FT_OK)
        {
            return false; // USB Error! //
        }
        return true;
    }
    //*************************************************************************//


    //*************************************************************************//
    // Read USB Rx Buffer
    //
    //
    // New: 2/20/2011
    //*************************************************************************//
    public bool readUSBRx(out string recDataStr)
    {
        uint numBytesRead = 0;
        uint RxQueue = 0;
        string rDataBuffer = "";
        recDataStr = "";

        FTDIDevice.GetRxBytesAvailable(ref RxQueue);
        if (RxQueue > 0)
        {
            ftStatus = FTDIDevice.Read(out rDataBuffer, RxQueue, ref numBytesRead);
            if (ftStatus != FTDI.FT_STATUS.FT_OK)
            {
```

```
                        return false; // USB Error!
                    }
                }
                recDataStr = rDataBuffer;
                return true;
            }
            //*****************************************************************************//

        }
    }
        //*****************************************************************************//
```

APPENDIX B

1. C_sharp_Test_Code_JI_4040_Rev1 File

The latest version of this file can be found on the Jupiter Instruments website (www.jupiteri.com).