Future Technology Devices International Ltd

VMusic SPI Application Note

Document Reference No.: FT_000029

Version 1.01

Issue Date: 2008-06-02

Future Technology Devices International Ltd (FTDI)

373 Scotland Street, Glasgow G5 8QB United Kingdom

Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758

E-Mail (Support): vinculum.support@ftdichip.com Web: http://www.ftdichip.com

Vinculum is part of Future Technology Devices International Ltd. Neither the whole nor any part of the information contained in, or the product described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Ltd will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Ltd, 373 Scotland Street, Glasgow G5 8QB United Kingdom. Scotland Registered Number: SC136640

© Copyright 2008 Future Technology Devices International Ltd

Table of Contents

© Copyright 2008 Future Technology Devices International Ltd

List of Tables

© Copyright 2008 Future Technology Devices International Ltd

List of Figures

© Copyright 2008 Future Technology Devices International Ltd

1 Introduction

The Vinculum VNC1L IC and firmware libraries provided by FTDI allow embedded systems to easily communicate with USB devices. Using the VNC1L device, microcontrollers can now communicate with a range of USB devices including Bulk Only Mass Storage Class (BOMS), Communication Device Class (CDC), Printer Class, Human Interface Device (HID) class devices and USB hubs.

To provide a visual demonstration of the capabilities of the VNC1L using the VMusic2 module, a sample application using a PIC microcontroller demonstration board has been created. This example shows how to use a PICDEM 2 PLUS Demo Board to issue the desired commands to the VNC1L in order to control a VMusic2 module.

This application note shows example methods of using the SPI interface to communicate with the VNC1L; sending and receiving data from the firmware command monitor; managing and decoding responses from the command monitor.

The application provides simple media player functionality using buttons, potentiometer and the LCD on the PICDEM 2 PLUS Demo Board. The buttons send commands to the VNC1L to control playback and select tracks; the LCD shows elapsed time and file metadata obtained by the VNC1L firmware; the potentiometer adjusts the playback volume. The buttons may perform different functions depending on the length of time they are held. Table 1.1 lists the functions of the buttons.

State	Button	Action
Stopped	**S3**	**Play**
	S2	**Play Random**
Playing	**S3 (Long)**	**Next Track**
	S2 (Long)	**Previous Track**
	S3 (Short)	**Next Track**
	S2 (Short)	**Previous Track**
	S3 and S2 (Long)	**Stop**
	S3 and S2 (Short)	**Pause**
Paused	**S3 and S2 (Short)**	**Play**
	S3 and S2 (Long)	**Stop**

Table 1.1 Button Functions

Implementation details for programming the SPI bus and communicating with the command monitor are provided. Other functions of the application are not specifically related to the function of the VNC1L device and are left to the user to interpret.

© Copyright 2008 Future Technology Devices International Ltd

2 Requirements

2.1 Hardware

This application requires a VMusic2 module, a Microchip PICDEM 2 PLUS Demo Board and a cable to connect the VMusic2 and PICDEM 2 PLUS board. A PIC18F452 or PIC18F4520 microcontroller is needed to run the provided code.

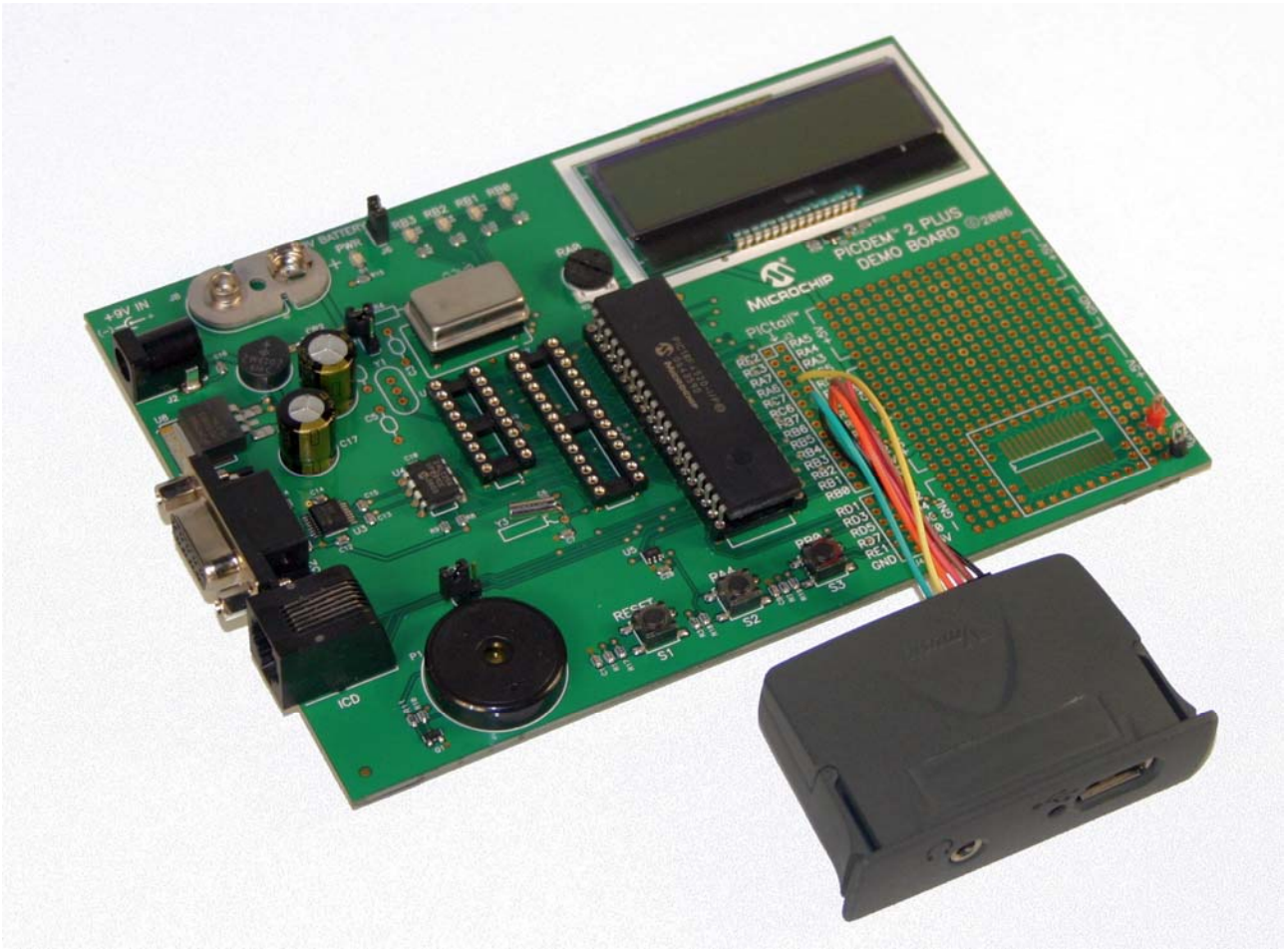

Figure 2.1 VMusic SPI Application Demo Hardware

The connecting cable comprises four I/O port signals and +5V and GND signals. This carries power and the SPI interface signals - SCLK, CS, SDI and SDO. Table 2.1 shows the connection between the CN2 (monitor interface) on the VMusic2 module and the 2mm ribbon cable supplied with the module. In Table 2.2 the port connections where the cable should be connected to on the PICDEM 2 PLUS board are listed.

.

© Copyright 2008 Future Technology Devices International Ltd

Pin Number	Signal Name	Colour
8	N/A	Blue
7	N/C	
6	CS	Green
5	SCLK	Yellow
4	SDI	Orange
3	+5V	Red
2	SDO	Brown
1	GND	Black

Table 2.1 VMusic2 Connections

Pin Number	Signal Name	Colour
RC2	CS	Green
RC5	SCLK	Yellow
RC1	SDI	Orange
+5V	+5V	Red
RC0	SDO	Brown
GND	GND	Black

Table 2.2 PICDEM 2 PLUS Connections

The VMusic2 Module must be set to SPI mode using jumper CN3 to connect the UART/SPI pin to GND.

J7 and J9 on the PICDEM 2 PLUS board must be unpopulated and J6 populated. The 4MHz canned oscillator, supplied with the board, is required.

2.2 Firmware

The VNC1L device on the VMusic2 Module must be programmed with VMSC1 firmware version 3.63 or later for this sample project. All of the VMSC1 firmware commands are specified in the Vinculum Firmware User Manual.

2.3 Software

Sample C code for the PIC18F452 or PIC18F4520 microcontroller is available as a free download from the FTDI web site. The code for the PIC microcontroller must be compiled with SourceBoost C or ported to an equivalent compiler before being programmed onto the device.

© Copyright 2008 Future Technology Devices International Ltd

3 SPI Interface Module

The SPI Interface module controls the application's communication with the SPI interface on the VNC1L. The files SPI.c and SPI.h manage the interface as an SPI Master, controlling the SCLK (clock), CS (chip select) and SDI (data in) signals. The SDO (data out) signal is controlled by the VNC1L.

The direction of data is encoded in the SPI transaction as either a read or a write. The definitions are shown in Figure 3.1.

```
#define DIR_SPIWRITE 0
#define DIR SPIREAD  1
```

Figure 3.1 SPI Data Direction Definition Code

Each callable function in the SPI Interface Module is aliased using a pre-processor definition to allow the entire module to be replaced by a functionally equivalent module for handling communications. This may be a UART or Parallel FIFO interface. The module is called the Monitor Interface. The definitions are included in the SPI.h file and are shown in Figure 3.2. The calling application should only use the generic interface routine names and not the SPI specific ones.

```
#define monInit spiInit
#define monRead spiRead
#define monReadWait spiReadWait
#define monWrite spiWrite
#define monInterrupt
```

Figure 3.2 Monitor Callable Functions for SPI Interface

3.1 Pin Definitions

The SPI Interface between the PICDEM 2 PLUS and the VNC1L on the VMusic2 Module utilises 4 signals. All signal names given here are relative to the VNC1L device. The pin definitions for the signals used by the SPI interface are shown in Figure 3.3.

```
#define PORT_SDI  portc.1        // SDI (on VMusic) is RC1 (Port C bit 1)
#define TRIS_SDI  trisc.1

#define PORT_SDO  portc.0        // SDO (on VMusic) is RC0 (Port C bit 0)
#define TRIS_SDO  trisc.0

#define PORT_SCLK portc.5        // SCLK is RC5 (Port C bit 5)
#define TRIS_SCLK trisc.5

#define PORT_CS   portc.2        // CS is RC2 (Port C bit 2)
#define TRIS_CS   trisc.2
```

Figure 3.3 SPI Pin Definition Code

3.2 External Functions

The SPI Interface code presents 4 subroutines externally:

- spiInit – initialise the SPI interface I/O pins.
- spiReadWait – read continually from the SPI interface until a byte is received.
- spiRead – check SPI interface for available byte of data.
- spiWrite – write a byte of data to the SPI interface.

© Copyright 2008 Future Technology Devices International Ltd

These commands can be used to allow a program to access the VNC1L firmware command monitor.

3.2.1 spiInit

Parameters:

None.

Returns:

None.

Calls:

None.

This routine sets up the 4 interface pins SCLK (output), CS (output), SDI (output) and SDO (input). It will also initialise the 3 output signal to a logic low.

3.2.2 spiReadWait

Parameters:

None.

Returns:

char – data read from SPI Data Read operation.

Calls:

spiXfer

Repeatedly calls spiXfer to generate an SPI Data Read operation until a byte of data has been received successfully.

3.2.3 spiRead

Parameters:

char *pSpiData – pointer to variable to receive data read from SPI Data Read operation.

Returns:

char – zero if data read successful, 1 if unsuccessful.

Calls:

spiXfer

Calls spiXfer subroutine once to generate an SPI Data Read operation. If a byte of data has been received successfully it will return zero, otherwise it will return 1. The calling parameter pSpiData is updated with data read from the SPI interface.

3.2.4 spiWrite

Parameters:

char spiData – data to write with SPI Data Write operation.

© Copyright 2008 Future Technology Devices International Ltd

Returns:

None.

Calls:

spiXfer

Calls spiXfer subroutine once to generate an SPI Data Write operation. If the byte of data has been transmitted successfully it will return, otherwise it will retry.

3.3 Internal Functions

The following subroutines are not available externally to the SPI interface code.

3.3.1 spiXfer

Parameters:

int spiDirection – specifies an SPI read or write operation.
char *pSpiData – pointer to variable to receive a byte read from an SPI read operation or pointer to data to transmit with an SPI write operation.

Returns:

char – zero if data write successful, 1 if unsuccessful.

Calls:

None.

This subroutine generates a state machine to both read and write data on the SPI interface. The direction of data flow is set using the spiDirection parameter using the values in Figure 3.1.

The state machine flow is as follows:

Create start condition on SPI interface – CS high, SDI high, SCLK rising edge.
Set data direction – CS high, SDI set to spiDirection, SCLK rising edge.
Set data operation – CS high, SDI set to low, SCLK rising edge.
For each 8 bits of data (starting at most significant bit) –
    If read set bit in data variable to value from SDO signal.
    If write set SDI to bit value from data variable.
    Toggle SCLK signal.
Read status bit value from SDO signal. Toggle SCLK signal.
Set finish condition – CS low, SCLK rising edge.
Return status bit value.

The spiXfer subroutine is not interrupt driven and does not use the built-in SPI functionality of the microcontroller. It is solely intended to demonstrate an efficient method of reading and writing to the SPI Interface on the VNC1L.

© Copyright 2008 Future Technology Devices International Ltd

3.3.2 spiDelay

Parameters:

None.

Returns:

None.

Calls:

None.

A delay function may be added to reduce the speed of the bus operation by defining the spiDelay routine to perform a delay function if it is desired to reduce the SPI interface speed. This is set to produce a nop operation in this example.

© Copyright 2008 Future Technology Devices International Ltd

4 Command Monitor Module

The firmware command monitor on the VNC1L interprets commands from an application and reports the status of commands. There are several error responses which can come from each command.

The Command Monitor Module controls the sending of commands to the VNC1L, receiving the responses and interpreting the responses. The appropriate subroutines in the Monitor Interface Module are called to communicate with the VNC1L. Figure 4.1 lists the enumeration of responses from the VNC1L firmware command monitor.

```
// Prompts and messages returned by VNC1L in Short Command Set
enum vResponse {
       // Prompts returned by all VNC1L firmware
    Resp_Prompt_OK, // >  (Success)
    Resp_Prompt_ND, // ND (Success)
    Resp_Prompt_UE, // E echo
    Resp_Prompt_LE, // e echo
    Resp_Prompt_CF, // CF (Command Failed)
    Resp_Prompt_BC, // BC (Bad Command)
    Resp_Prompt_DF, // DF (Disk Full)
    Resp_Prompt_FI, // FI (File Invalid)
    Resp_Prompt_RO, // RO (Read Only)
    Resp_Prompt_FO, // FO (File Open)
    Resp_Prompt_NE, // NE (Dir Not Empty)
    Resp_Prompt_FN, // FN (Filename Invalid)
    Resp_Prompt_End,
       // Asynchronous messages returned by VMSC1 firmware
    Resp_Message_P,         // P / Playing
    Resp_Message_S,         // S / Stopped
    Resp_Message_T,         // T / Time
       // Asynchronous messages returned by all VNC1L firmware
    Resp_Message_NU,              // NU / No Upgrade
    Resp_Message_DD1,       // DD1 / Device Detected USB Port 1)
    Resp_Message_DD2,       // DD2 / Device Detected USB Port 2)
    Resp_Message_DR1,       // DR1 / Device Removed USB Port 1)
    Resp_Message_DR2,       // DR2 / Device Removed USB Port 2)
    Resp_Message_Splash,    // Ver ...
    Resp_None = 0xff,
};
```

Figure 4.1 Monitor Response Enumeration

4.1 External Functions

4.1.1 monSendByte

Parameters:

char monData – byte of data to write to interface.

Returns:

None.

Calls:

monWrite()

This command simply transfers a byte of data to the Monitor Interface Module. It has an inline specifier due to the simplicity of the function.

© Copyright 2008 Future Technology Devices International Ltd

4.1.2 monCmdSend

Parameters:

char monCmd – command byte to write to interface.

Returns:

None.

Calls:

monWrite()

This command writes the command byte passed as a parameter to the Monitor Interface Module. It follows the command byte with a carriage return character if required.

4.1.3 monCmdSendByteParam

Parameters:

char monCmd – command byte to write to interface.
unsigned char monParam – single byte to be sent as parameter.

Returns:

None.

Calls:

monWrite()

This command writes the command byte passed as a parameter to the Monitor Interface Module. It will then send a space character to delimit the parameter. The single-byte parameter monParam is sent followed by a carriage return.

4.1.4 monCmdSendParam

Parameters:

char monCmd – command byte to write to interface.
unsigned char monCount – count of bytes to send as parameters to interface.
unsigned char *pmonParam – pointer to array of bytes to be sent as parameters.

Returns:

None.

Calls:

monWrite()

This command writes the command byte passed as a parameter to the Monitor Interface Module. It will then send a space character to delimit the parameters. The parameters consist of monCount bytes, the data is passed in the pmonParam array. The command and parameters are followed by a carriage return.

4.1.5 monResponse

Parameters:

None.

Returns:

© Copyright 2008 Future Technology Devices International Ltd

enum vResponse – enumeration of prompt or error response command returns.

Calls:

monRead()

After a command has been sent to the VNC1L, it will always return a response indicating success or an error condition. The monResponse subroutine compares the data returned from the command monitor to a list of expected responses and returns a matching value from an enumeration list. This can be used by the calling application to check for the expected behaviour of the device.

4.1.6 monPrompt

Parameters:

None.

Returns:

enum vResponse – enumeration of prompt or error response command returns.

Calls:

monResponse()

This will poll the monResponse subroutine until a valid prompt or error message is returned. In the enumeration vResponse, Resp_Prompt_End marks the last valid prompt or error message in the list. Subsequent values indicate asynchronous messages which may be sent by the VNC1L.

4.2 Internal Functions

There are no internal functions in the Command Monitor Module.

© Copyright 2008 Future Technology Devices International Ltd

5 Program Control

This section describes the sequence of events used by the application to communicate with the VNC1L firmware command monitor.

5.1 Initialisation

The first task of the application is to establish a connection to and synchronise with the command monitor. This is best accomplished by sending and receiving echo commands until the state of the command monitor is known. The code in Figure 5.1 repeatedly sends out a capital 'E' character followed by a lower-case 'e' character until the command monitor echoes the characters back in the same order.

```
// ensure Vinculum is synchronised
// ignore all messages and prompts until sync operation completes
while (1)
{
	clear_wdt();
	monCmdSend('E');
	while (monPrompt() != Resp_Prompt_UE);
	monCmdSend('e');
	if (monPrompt() == Resp_Prompt_LE)
	break;
};
```

Figure 5.1 Synchronisation with Echo Commands

Next, the application changes the command monitor to use the Short Command Set. This reduces the length of commands and responses making sending commands and parsing responses easier. The code in Figure 5.2 shows how this is done. An error handling routine is called if the command monitor call is not successful.

```
// change to Short Command Set Mode
monCmdSend(CMD_SCS);
resp = monPrompt();
if (resp != Resp_Prompt_OK)
{
	opError(resp);
}
```

Figure 5.2 Change to Short Command Set

After the Short Command Set mode has been selected the code enters the main loop, see Figure 5.3, where it checks for button presses and updates the LCD based on the asynchronous messages returned by the command monitor. It remains in this loop until the disk containing the media files is removed.

© Copyright 2008 Future Technology Devices International Ltd

```
do
{
	clear_wdt();
	resp = monResponse();
	if (resp == Resp_Message_P)
	{
		rspPlaying();
		opStatus.STATUS_PLAY = 1;
		opStatus.STATUS_WAITPROMPT = 0;
		opStatus.STATUS_RESET = 1;
	}
	else if (resp == Resp_Message_T)
	{
		rspTime();
		opStatus.STATUS_PLAY = 1;
	}
	else if (resp == Resp_Message_S)
	{
		rspStopped();
		opStatus.STATUS_PLAY = 0;
	}
	else if (resp == Resp_Prompt_ND)
	{
		// device removed - no action
	}
	else if (resp == Resp_Prompt_OK)
	{
		// no idle processing until prompt received
		opStatus.STATUS_WAITPROMPT = 0;
	}
	else if (resp == Resp_None)
	{
		opControl();
	}
	opStatus();

	// if disk is removed detect this and loop back
	// to No Disk message
} while (resp != Resp Prompt ND);
```

Figure 5.3 Main Control Loop

© Copyright 2008 Future Technology Devices International Ltd

6 Summary

The provided C code for a PIC microcontroller demonstrates how to use the SPI Interface and Command Monitor to interface with a VMusic2 Module.

The techniques used in this application note are applicable to the whole range of VNC1L-based modules and applications.

© Copyright 2008 Future Technology Devices International Ltd

7 Copyright and Disclaimer

© Copyright 2008 Future Technology Devices International Limited

Version 1.01 Initial Datasheet Created May 2008

This document provides preliminary information that may be subject to change without notice. Vinculum is part of Future Technology Devices International Limited. Neither the whole nor any part of the information contained in, or the product described in this datasheet, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Limited will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Limited, 373 Scotland Street, Glasgow G5 8QB, United Kingdom. Scotland Registered Number: SC136640.

8 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
373 Scotland Street
Glasgow G5 8QB
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) vinculum.sales@ftdichip.com
E-mail (Support) vinculum.support@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL http://www.ftdichip.com
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
4F, No 18-3, Sec. 6 Mincyuan East Road
Neihu District
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

© Copyright 2008 Future Technology Devices International Ltd