



Application Note

AN_368

VM800P4350 RS232 Sample

Version 1.0

Issue Date: 2015-03-24

This document provides a guide for using the RS232 sample code to transfer data with a VM800P43 and RS232 plug in accessory card VI800A-232U.

Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold FTDI harmless from any and all damages, claims, suits or expense resulting from such use.

Future Technology Devices International Limited (FTDI)

Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom

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

Web Site: <http://ftdichip.com>

Copyright © 2015 Future Technology Devices International Limited

Table of Contents

1	Introduction	4
1.1	Overview	4
1.2	Features	4
1.3	Library dependencies.....	4
1.4	Limitations.....	4
1.5	Scope.....	4
2	RS232 Demo Hardware.....	5
3	RS232 Demo Firmware	6
3.1	Overview	6
3.2	Images.....	6
3.2.1	Image Loading.....	6
3.3	Display layout.....	7
3.4	RS232 Image Transfer	8
3.4.1	General Operation	8
3.4.2	Image Data Transfer (Arduino <----> FT800).....	8
3.4.3	Image Transfer Animation	8
3.5	Swipe Lists.....	9
3.5.1	Image Selection.....	9
3.5.2	Baudrate Selection.....	9
3.6	RS232 Routines	10
3.6.1	Transmit Functions.....	10
3.6.2	Receive Functions	11
4	Possible Improvements.....	12
5	Contact Information.....	13
	Appendix A – References	14
	Document References.....	14
	Acronyms and Abbreviations.....	14
	Appendix B – List of Tables & Figures	15
5.1	List of Tables	15



5.2	List of Figures	15
	Appendix C – Revision History	16

1 Introduction

This application note documents an example firmware project for the VM800P43A. The source code is available in the "examples\FT_VI800A_232U\FT_VM800P43_50\Image_Transfer_Master" and "examples\FT_VI800A_232U\FT_VM800P43_50\Image_Transfer_Slave" folders of the FTDI Arduino library.

Refer to the following document for further details of the Arduino libraries: [AN_318 Arduino Library For FT800 Series](#) and [AN_330 VI800A Modules Library](#).

1.1 Overview

The application implements a demonstration of data transfer, via RS232, and has been written to run on a pair of VM800P43A evaluation modules, with attached VI800A-232U modules:

- ATMEGA328P MCU (Arduino PRO 5V, 16MHz) processor
- FT800 processor + 4.3" touch panel display
- SD card connector
- RTC with battery backup
- IO expandability via daughter board (VI800A-232U)

1.2 Features

The RS232 example has the following features:

- Open source firmware.
- Touch input for control and selection (images and baudrate).
- Displays graphics output on the FT800 enabled display via SPI.
- Transfers data between two modules via RS232.

1.3 Library dependencies

The following FTDI libraries and the [Arduion IDE](#) must be installed before running the application. Refer to the respective library application note in each of download location for instructions on how to install the library.

- [Arduion Specific library for FT800](#) (library for the VM800P43A and examples)

- [Arduion Library Sample App\(V1.1\)](#) (library for the VI800A-232U and examples)

1.4 Limitations

Code has only been tested on the VM800P43A / VI800A-232U modules.

1.5 Scope

The guide is intended for developers who are creating applications, extending FTDI provided applications or implementing example applications for the VM800P43A / VI800A-232U modules.

The sample application consists of the source code.

2 RS232 Demo Hardware

The hardware required by the RS232 example code is as follows:

- 2 x VM800P4350 modules
- 2 x VI800A-232U modules

The VI800A-232U modules should be plugged into the rear of the VM800P4350 modules, and should then be connected as follows (colour of wires is not important):

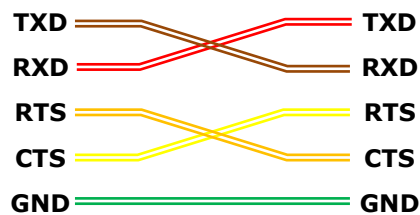


Figure 1: RS232 Wiring

The following is a photograph of the rear of the two VM800P4350 modules fitted with VI800A-232U daughter cards, with wiring connected as shown above.

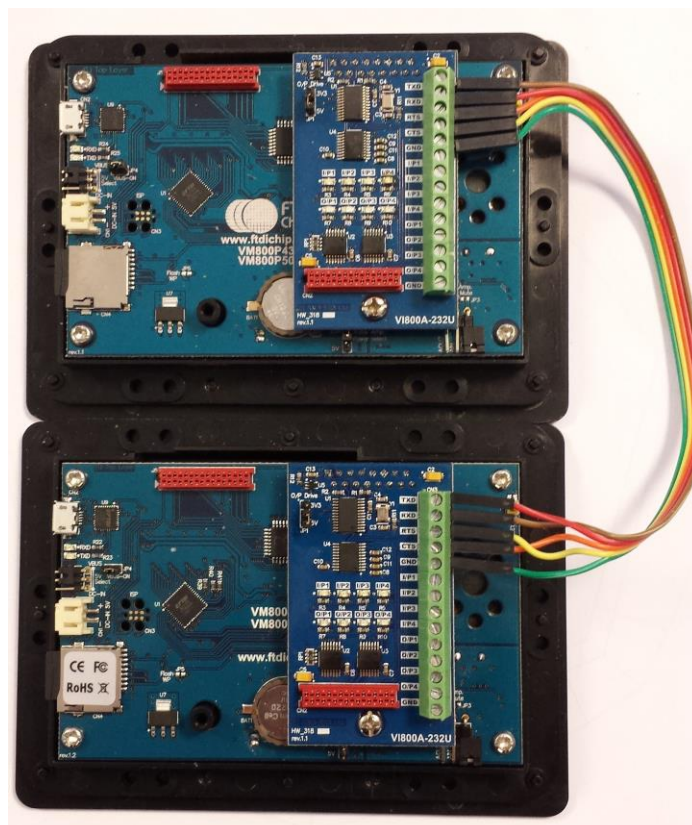


Figure 2: Two modules connected

3 RS232 Demo Firmware

3.1 Overview

The firmware included in the example code demonstrates transferring data from one VM800P4350 to another via the attached VI800A-232U daughter cards. In this particular case the data consists of a series of images, which demonstrates the data being transferred both physically and visually.

The firmware consists of two parts and is as follows:

- **Image_Transfer_Master.ino** - firmware loaded onto 1st module, referred to as 'MASTER'
 - The 'SDcard' folder contains images which should be copied onto the SD card of the 'MASTER' module at the root level.
- **Image_Transfer_Slave.ino** - firmware loaded onto 2nd module, referred to as 'SLAVE'

3.2 Images

The images used in this RS232 example are stored on an SD card attached to the MASTER, and are loaded one at a time into processor ram before being transferred to the FT800 for inflation and eventual storage in the FT800 graphics ram.

The images that have been selected cover a number of different physical sizes, in order to demonstrate the effect on transfer rates related to image size.

The images have also been selected so as to ensure that once inflated, they will all actually fit in the graphics ram of the FT800 (256Kb), ensuring reasonably fast access to the raw image data for transfer via RS232. *Note: The image filenames are hard coded and are stored in the `imageName[][]` array, meaning the user can easily change the images simply by supplying new files and filenames (current images leave 908 bytes free in the FT800 graphics ram).*

3.2.1 Image Loading

Loading of the images is performed by two functions as follows:

- **loadJpegImage()** which performs the following operations:
 - Checks that the file for the image to be loaded actually exists
 - Issues a **CMD_LOADIMAGE** command to the FT800 to load a JPEG image and inflate it
 - Calls the **loadJpegBytes()** function to transfer file data from SD card to FT800, supplying the data required by the **CMD_LOADIMAGE** command, in order that the FT800 can inflate the JPEG data and store it in the FT800 graphics ram
 - Issues a command to the FT800 to return information about the JPEG, which consists of image format, size and address of beginning of data in FT800 ram
 - Stores returned information about the image for subsequent use

Note: Stores 0 for image size if file does not exist or is invalid for whatever reason

- **loadJpegBytes()** which performs the following operations:
 - Loads and transfers 512 byte blocks at a time of the JPEG file and transfers it to the FT800, until the entire file has been processed (supplies the data for the load image command sent previously)

3.3 Display layout

The display layout for the MASTER and SLAVE are as follows:

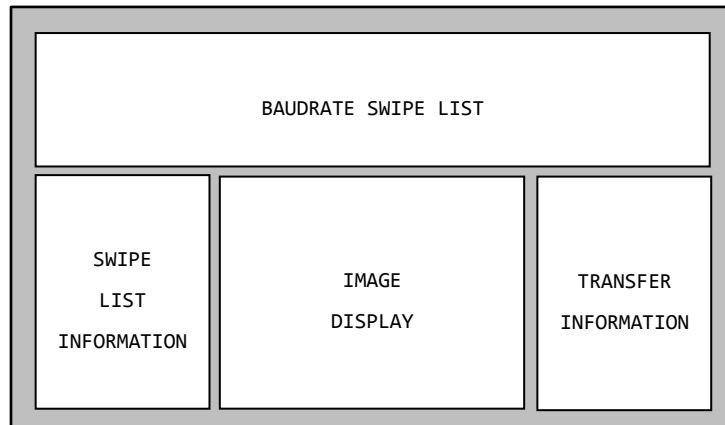


Figure 3: Slave Display Layout

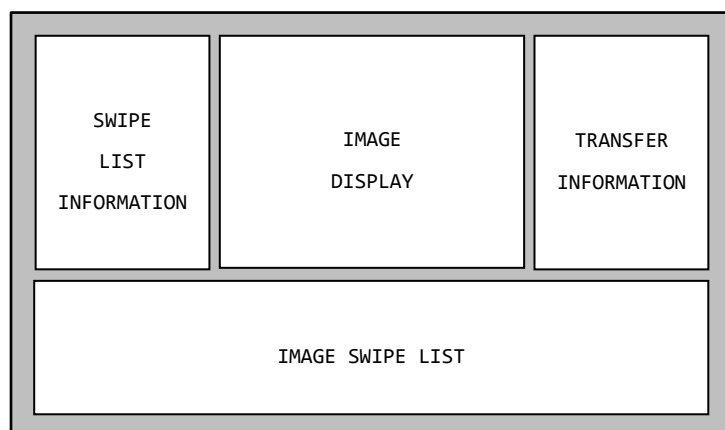


Figure 4: Master Display Layout

The SWIPE LIST areas are used to both display and allow the selection of the available baudrates on the SLAVE and images on the MASTER, enabling the baudrate and image to be changed at any time the user wishes.

The IMAGE DISPLAY area is used to visually represent the image data being transferred from the MASTER to the SLAVE, with the image moving up and disappearing line by line on the MASTER and then reappearing line by line on the SLAVE.

The SWIPE LIST INFORMATION area describes how to operate the swipe lists, although the vast majority of people are used to this sort of operation due to the ubiquity of mobile phones and tablets.

The TRANSFER INFORMATION area provides much of the same information on both the MASTER and SLAVE, i.e. image size (X, Y and bytes) together with the baudrate. The MASTER also displays the filename of the image selected, with the SLAVE displaying the total bytes transferred and the total time taken so far.

3.4 RS232 Image Transfer

3.4.1 General Operation

The general mode of operation is as follows:

- MASTER to SLAVE: **COMMAND_HELLO_THERE** is sent to establish communications
- SLAVE to MASTER: **COMMAND_HELLO_THERE_CONFIRMED** is returned indicating that the SLAVE is listening.
- MASTER to SLAVE: **COMMAND_IMAGE_START** is sent to initiate the start of an image transfer (command includes size of the image)
- SLAVE to MASTER: **COMMAND_IMAGE_START_CONFIRMED** is returned indicating that the SLAVE has initialised itself and is ready for image transfer
- For each line of image:
 - MASTER to SLAVE: **COMMAND_IMAGE_SLICE** is sent, containing the line number of the image being sent, together with the image data which has been read from the FT800 graphics ram
 - SLAVE to MASTER: **COMMAND_IMAGE_SLICE_CONFIRMED** is returned indicating that the SLAVE has received the image data and has transferred it to the FT800 graphics ram and is ready to accept the next command
- MASTER to SLAVE: **COMMAND_IMAGE_DONE** is sent, indicating that the image transfer has been completed
- SLAVE to MASTER: **COMMAND_IMAGE_DONE_CONFIRMED** is returned, indicating that the SLAVE has completed image transfer
- Both MASTER and SLAVE now sit idle for a short time, before starting the whole process again.

3.4.2 Image Data Transfer (Arduino <-----> FT800)

Transfer of data to and from the FT800 graphics ram is performed by the following functions:

- **copyDataFromRamG()** which copies the specified line of data, of the currently selected image, from the FT800 graphics ram to the Arduino ram (only used by the MASTER)
- **copyDataIntoRamG()** which copies the data from the Arduino ram into the FT800 graphics ram at the specified line (only used by the SLAVE)

3.4.3 Image Transfer Animation

Animation of the transfer of the image data gives a visual indication to the user of the amount of data transferred and the speed at which it is being transferred.

It is achieved by simple use of a scissor rectangle, set to the size of the image and positioned centrally within the display area, and the knowledge of which line of image is currently being transferred:

- On the Master, the image is positioned to only show lines that have still to be transferred
 - Y display position = top of scissor rectangle - lines transmitted
- On the Slave, the image is positioned to only show lines that have been received
 - Y display position = bottom of scissor rectangle - lines transmitted

The scissor rectangle, together with the positioning of the image, ensures that only the parts of the image we wish to show are seen.

The addition of a flickering 'white scanning light' suggests to the user that the image is being 'scanned', a simple but effective addition to the overall animation.

3.5 Swipe Lists

Refer to the following document for further in-depth details of the operation of swipe lists:

AN_367_VM800P4350_Swipe_Lists.pdf

Note: The user can select images and baudrates at any time during operation.

3.5.1 Image Selection

Image selection on the MASTER operates in the following way (in response to the user selecting an image):

- Current command will be completed, including waiting for any response which is expected
- New image is selected and all relevant variables are initialised
- General mode of operation is restarted (**COMMAND_HELLO_THERE**), at which point the transfer of the newly selected image will begin

3.5.2 Baudrate Selection

Baudrate selection on the SLAVE operates in the following way (in response to the user selecting a baudrate):

- The SLAVE performs the following operations (before processing the latest command from the MASTER):
 - **COMMAND_BAUDRATE_CHANGE** is sent, together with the selected baudrate, to initiate baudrate change on the MASTER (no response is required).
 - All bytes in the Tx queue are transmitted
 - Baudrate is then changed to the selected value
 - A small delay is then made (50ms)
 - Transfer of the image continues at the new baudrate
- The MASTER performs the following operations:
 - Baudrate is changed to the supplied value
 - No response is sent
 - Transfer of the image continues at the new baudrate

Notes:

- 1) *The swipe list routines are called used the same method as described in the application note AN_367_VM800P4350_Swipe_Lists, i.e. a call to the `milliTimer()` function from within the main while loop (which does not exit).*
- 2) *The `milliTimer()` function is responsible for calling the display rendering function, which itself calls the touch handling and render functions.*

3.6 RS232 Routines

Both the MASTER and SLAVE use the same core routines to implement a simple command / response system, only ever processing one command and response at a time.

Commands are sent and received by the functions described below, and are called from within the main while loop (which does not exit).

3.6.1 Transmit Functions

3.6.1.1 *commandStateHandlerTx()*

This is the main command handling function, and is responsible for determining what happens next, with regards command execution and transmission.

This function operates in one of two states (*dataTxState*):

- STATE_TX_IDLE:
 - If *dataTxCommand* is set to **COMMAND_WAITING_FOR_REPLY**:
 - Does nothing (waiting for a response)
 - Otherwise:
 - *dataTxCommand* may be processed internally within the function or it may result in a command being sent via RS232, in which case the *commandBlockHandlerTx()* function will be called (see below), resulting in a change of state to STATE_TX_DATA
- STATE_TX_DATA:
 - Calls the *commandBytesHandlerTx()* function (see below) to transmit bytes of command to be sent via RS232, and will automatically switch state to STATE_TX_IDLE once data has been transmitted

3.6.1.2 *commandBlockHandlerTx()*

The command block handler is supplied with a number of variables and is responsible for the following operations:

- Inserts the provided variable values into the transmit buffer.
 - **command** is the command to be sent.
 - **parameter** is the parameter to be sent
 - **datasize** is the size of the command block + any extra data payload to be sent with the command
- Additional data payload should have been inserted into the transmit buffer before this routine is called
- Initialises the transmit variables detailing how many bytes of information have to be sent
- Sets *dataTxCommand* to **COMMAND_WAITING_FOR_REPLY**
- Sets *dataTxState* to STATE_TX_DATA

3.6.1.3 *commandBytesHandlerTx()*

The command bytes handler is responsible for the following operations:

- Checks to see if there is any data to be sent via RS232
- Checks to see if there is free space in the RS232 transmit buffer
- If the above are true:
 - Next byte of data is added to the RS232 transmit buffer
 - Number of bytes left to transmit is decremented
 - If all bytes have been transmitted:
 - Sets *dataTxState* to STATE_TX_IDLE ready to process the next command

3.6.2 Receive Functions

3.6.2.1 *commandStateHandlerRx()*

This is the main command receiving handling function, and is responsible for receiving commands (and data) before passing to the transmit routines for processing.

This function operates in one of two states (*dataRxState*):

- **STATE_RX_IDLE:**
 - If *dataRxCommand* is set to **COMMAND_WAITING_FOR_REPLY**:
 - Sets *dataRxBytes* to **COMMAND_BLOCK_SIZE**, indicating size of command we expect to receive
 - Sets *dataRxReady* to true, indicating we are ready to receive data
 - Sets *dataRxState* to **STATE_TX_DATA**, to receive next command
 - Otherwise:
 - *dataTxCommand* is set to *dataRxCommand* (the command received)
 - *dataTxparameter* is set to *dataRxparameter* (the parameter to be received)
 - *dataTxdatasize* is set to *dataRxdatasize* (the size of the command block + data payload received)
 - *dataRxCommand* is set to **COMMAND_WAITING_FOR_REPLY**
- **STATE_RX_DATA:**
 - Calls the *commandBytesHandlerRx()* function (see below) to receive bytes of command sent via RS232, which will automatically switch state to **STATE_RX_IDLE** once a full command block and data payload has been received

3.6.2.2 *commandBlockHandlerRx()*

The command block handler is responsible for the following operations:

- Once a full command block and data payload (if any) has been received, the following operations take place:
 - *dataRxCommand* is set to the command received.
 - *dataRxparameter* is set to the parameter to be received.
 - *dataRxdatasize* is set to the size of the command block + data payload received
 - Sets *dataRxState* to **STATE_TX_IDLE** to process the command.
 - Sets *dataRxReady* to true, indicating we are ready to receive data as command has now been processed

3.6.2.3 *commandBytesHandlerRx()*

The command bytes handler is responsible for the following operations:

- Checks to see if we are ready to receive data (*dataRxReady* set to true)
- Checks to see if there is any data received via RS232
- If the above is true:
 - Next byte of data is read and added to RS232 receive buffer
 - Number of bytes received is incremented
 - If number of bytes received is equal to command block size:
 - Checks to see if a data payload has been combined with the command block, and if so, increases the required size of the command block so as to include the data payload size.
 - Once a full command block and data payload (if any) has been received:
 - Calls the *commandBlockHandlerRx()* function to process the command block
 - Sets *dataRxReady* to false, indicating we are not ready to receive data as command has still to be processed

4 Possible Improvements

This is purely a demonstration of transferring commands and data via RS232, with some visuals thrown in, feel free to experiment with it.

5 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com

Branch Office – Tigard, Oregon, USA

Future Technology Devices International Limited
(USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited
(Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
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

Branch Office – Shanghai, China

Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com

Web Site

<http://ftdichip.com>

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640

Appendix A – References

Document References

Arduino Library Guide:

http://www.ftdichip.com/Support/Documents/AppNotes/AN_318_Arduino_Library_For_FT800_Series.pdf

VI800A Module Library Guide:

http://www.ftdichip.com/Support/Documents/AppNotes/AN_330%20VI800A_TTLU_232U_N485U_ArduinoLibrary_SampleApp.pdf

FT800 Series Programmers Guide:

<http://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf>

EVE Product Page:

<http://www.ftdichip.com/EVE.htm>

Acronyms and Abbreviations

Terms	Description
API	Application Programming Interface
EVE	Embedded Video Engine
RTC	Real Time Clock
SD Card	Secure Digital Card
USB-IF	Universal Serial Bus

Appendix B – List of Tables & Figures

5.1 List of Tables

No tables defined.

5.2 List of Figures

Figure 1: RS232 Wiring.....	5
Figure 2: Two modules connected	5
Figure 3: Slave Display Layout	7
Figure 4: Master Display Layout.....	7

Appendix C – Revision History

Document Title: AN_368 VM800P4350 RS232 Sample
Document Reference No.: FT_001162
Clearance No.: FTDI# 445
Product Page: <http://www.ftdichip.com/FTProducts.htm>
Document Feedback: [Send Feedback](#)

Revision	Changes	Date
0.1	Initial Draft	2014-12-23
1.0	First Release	2015-03-24