AN_330

VI800A TTLU 232U N485U Arduino Library Sample App

Document Reference No.:FT_001067

Version: 1.0

Issue Date: 2014_10_23

The objective of this document is to enable users to become familiar with the usage of the VM800P add-on modules Arduino library.

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
E-Mail (Support): **support1@ftdichip.com** Web: **http://www.ftdichip.com**

Table of Contents

1 Introduction

This application demonstrates sample applications for VM800P add-on-modules. The application gives a basic understanding of the serial add-on-board's features - transmit/receive, LED toggling. The sample application has been written for the Arduino Pro platform.

Users can review the source code of the sample application first, and then run the code to observe the effects. Editing the code is also encouraged to help learn the features of the VM800P add-on-board modules.

This document introduces how to set up and use the sample application with the FTDI VM800P development kits in relation to an Arduino Pro platform.

For VM800P add-on board development board details please refer to datasheets:

For TTLU : DS_VI800A-TTLU.pdf

For 232U : DS_VI800A-232U.pdf

For N485U : DS_VI800A-N485U.pdf

To learn more about Arduino Pro and its IDE, please check http://www.arduino.cc

1.1 Audience

This document assumes the audience has read the datasheet of the relevant VM800P add-on-board. In addition, familiarity with the C/C++programming language is necessary to understand the sample application source code. To understand the SPI interface of the Arduino Pro Platform, knowledge of the Arduino Pro hardware and IDE is required.

1.2 Scope

The Sample Application referenced in this document is created in the Arduino IDE and runs on the Arduino Pro built into the VM800P along with a specific add-on-board. A full project containing source code and compiled objects is provided.

1.3 Overview

Hardware

The VM800P plug-in modules are capable of several communication setups: PC to module, simple loopback, same module to module, and module to other devices using the same communication standard. The diagram below gives the basic hardware setup for PC to module communication and loopback self-test.

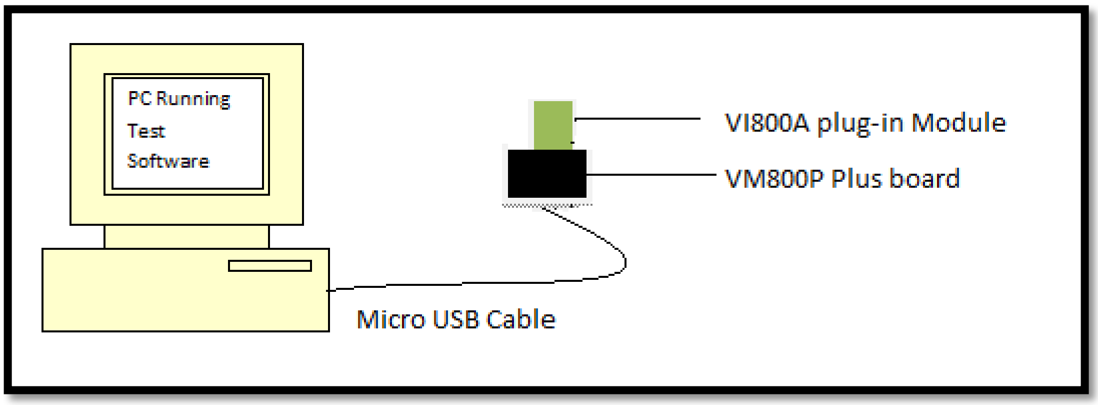


Figure 1-1 : Block Diagram of Setup

Architecture

The VM800P plug-in module library encapsulates the hardware communication, therefore, in-depth knowledge of the plug-in modules are not required.

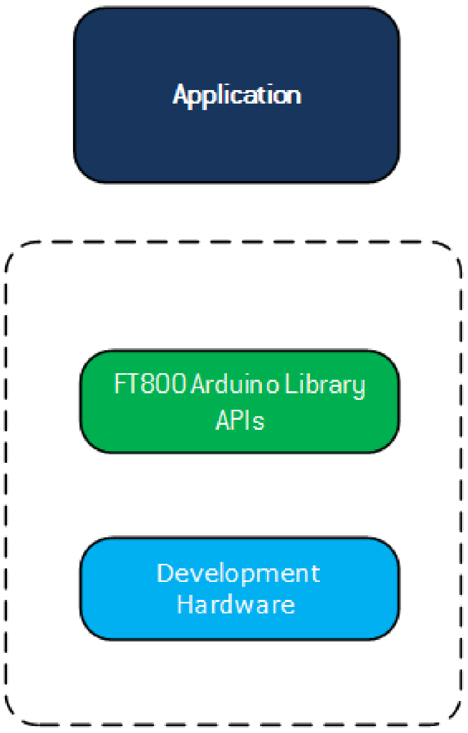


Figure 1-2 : Architecture diagram

1.4 Hardware requirement

The examples presented in this document use the following hardware, refer to the individual example for the specific items required.

- One of each of the FTDI VM800P Boards, two boards are required for same module communication.
- Two MicroB USB cables to provide power to the VM800P board
- Two of each module: VI800A-N485U, VI800A-232U, and VI800A-TTLU
- One TTL-232R cable(USB to TTL Serial Cable 5.0V)
- One RS232-USB cable

1.5 Software requirement

- Arduino IDE version 1.0.5 or later
- VM800P plug-in module Arduino Library release package.
- The open source virtual terminal Tera Term is used for plug-in module to PC communication.

2 Install Library

2.1 Another FTDI library is present

If there is an existing FTDI library in the library folder, simply extract the library files and put the new files/folders to their respective location in the existing library directory.

2.2 Automatic Installation

This method will install the library by the IDE and the library itself can be a ZIP file or unzipped folder. This method will install the library to the location specified in the IDE's Sketchbook location. The default directory, on OSX, would be at "~/Documents/Arduino/". On Windows, it would be at "My Documents\Arduino\". For more information on library installation, please refer to http://arduino.cc/en/Guide/Libraries.

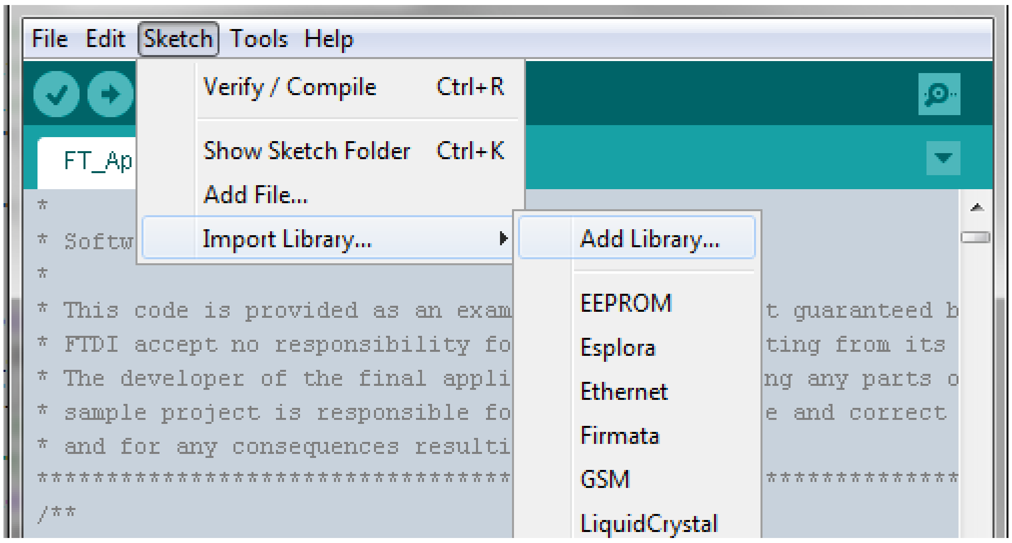


Figure 2-1 : Arduino IDE – add library option

In the Arduino IDE, click "Sketch", hover over the "Import Library" option in the drop down menu, and then click the "Add Library…" item.

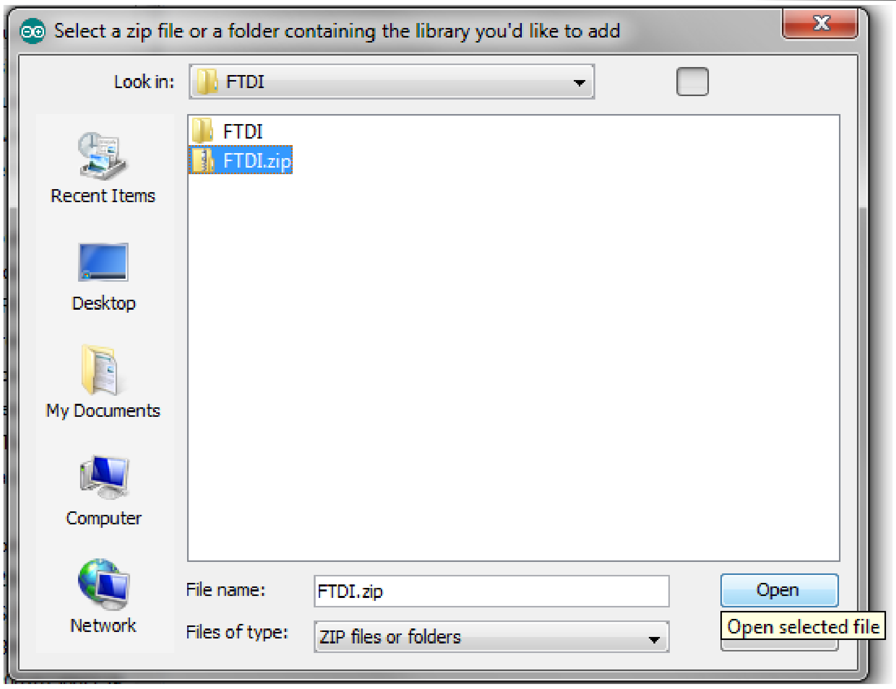


Figure 2-2 : Arduino IDE - library selection browser

Navigate to the downloaded FTDI library file or unzipped folder and open it. The IDE will automatically install the files to the library folder of the Sketchbook location.

2.3 Manual Installation

To manually install the library, simply unzip the library and put the FTDI folder in the library folder of the IDE's Sketchbook location. The Sketchbook location is in the preferences window accessed by choosing *File->Preferences*. Users will need to make a folder called "*libraries*" in the Sketchbook location if it doesn't contain one.

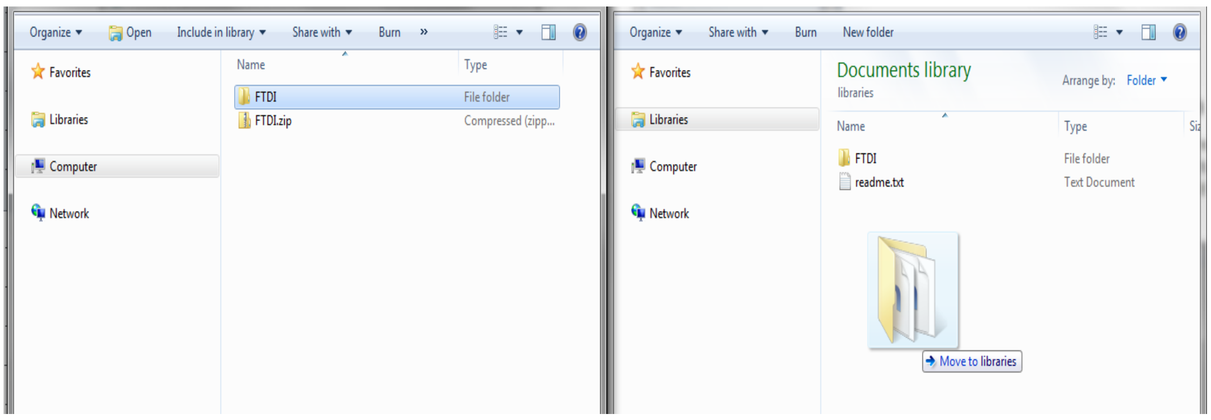
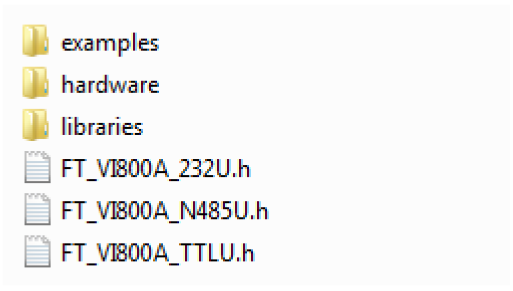


Figure 2-3 : Library Installation - Manual Installation

3 Library Folder Structure

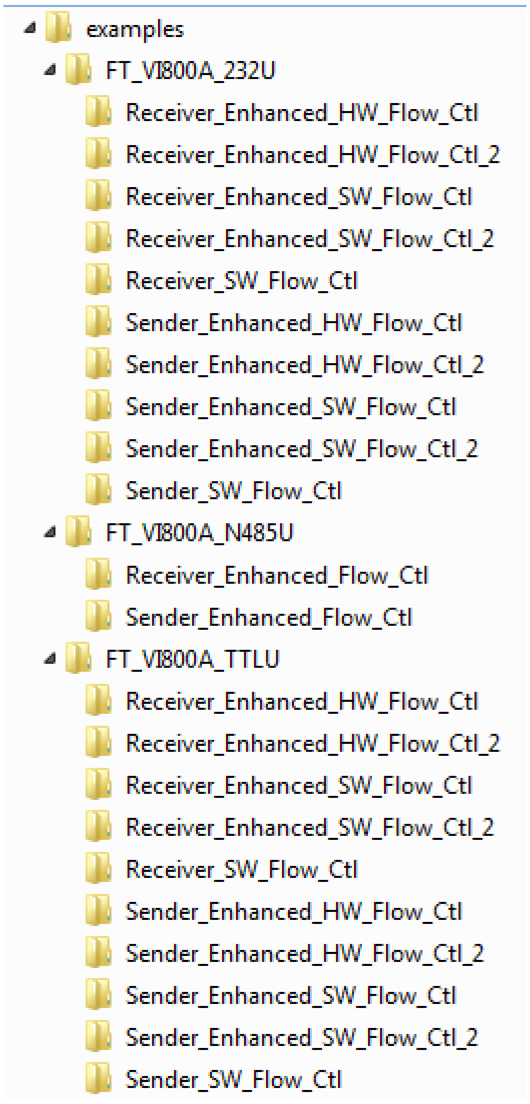
The below contents can be found in the release package:



examples
hardware
libraries
FT_VI800A_232U.h
FT_VI800A_N485U.h
FT_VI800A_TTLU.h

Figure 3-1 : FTDI Library folder contents

Example Folder: The examples folder contains example sketches in each of the respective platform sub folder. TTLU module examples are presented in FT_VI800A_TTLU folder, 232U module examples are presented in FT_VI800A_232U and N485U module examples are presented in FT_VI800A_N485U folder.



```
examples
  FT_VI800A_232U
    Receiver_Enhanced_HW_Flow_Ctl
    Receiver_Enhanced_HW_Flow_Ctl_2
    Receiver_Enhanced_SW_Flow_Ctl
    Receiver_Enhanced_SW_Flow_Ctl_2
    Receiver_SW_Flow_Ctl
    Sender_Enhanced_HW_Flow_Ctl
    Sender_Enhanced_HW_Flow_Ctl_2
    Sender_Enhanced_SW_Flow_Ctl
    Sender_Enhanced_SW_Flow_Ctl_2
    Sender_SW_Flow_Ctl
  FT_VI800A_N485U
    Receiver_Enhanced_Flow_Ctl
    Sender_Enhanced_Flow_Ctl
  FT_VI800A_TTLU
    Receiver_Enhanced_HW_Flow_Ctl
    Receiver_Enhanced_HW_Flow_Ctl_2
    Receiver_Enhanced_SW_Flow_Ctl
    Receiver_Enhanced_SW_Flow_Ctl_2
    Receiver_SW_Flow_Ctl
    Sender_Enhanced_HW_Flow_Ctl
    Sender_Enhanced_HW_Flow_Ctl_2
    Sender_Enhanced_SW_Flow_Ctl
    Sender_Enhanced_SW_Flow_Ctl_2
    Sender_SW_Flow_Ctl
```

Figure 3-2 : FTDI Library - examples folder

Hardware Folder: The SC16IS760IBS folder contains hardware specific macros used by the library.

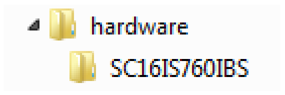


Figure 3-3 : FTDI Library - hardware folder

Libraries folder: The FT_VI800A folder contains the FT_VI800A class which implements APIs to access the hardware functionalities of the VI800A plug-in modules.

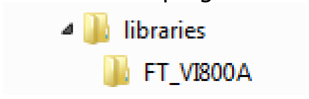


Figure 3-4 : FTDI Library - libraries folder

Platform headers: The following header files include platform specific macros. An application needs to include one of the respective platform header files.

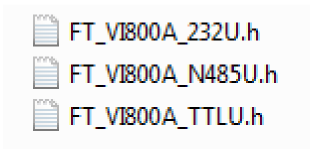


Figure 3-5 : FTDI Library - platform headers

3.1 FT_VI800A class

The FT_VI800A.h in FTDI\Libraries\FT_VI800A folder contains the FT_VI800A class which implements APIs to access and utilize the functionalities for the TTLU, 232U and N485U plug-in modules. The following table has the entire public accessible API in the library and their condensed description. Please refer to the FT_VI800A.h for the complete description of respective API along with any limitations, remarks and implications.

3.2 APIs

Prototype	Description
void GetVersion(uint8_t &Major, uint8_t &Minor, uint8_t &Build);	Get the Major, Minor and Build version of the library.
FT_Status Init(uint8_t SSpin);	Overloaded initialization API with default values.
FT_Status Init(uint8_t SSpin, uint32_t BaudRate, FT_ Parity_Type parityType, FT_ Stop_Bit_Length stopBitLength, FT_Word_Length wordLength);	Overloaded Initialization API with user specified values.
uint8_t Read();	Read a single byte from receive FIFO.
FT_Status Read(uint8_t* array, uint8_t maxBytesToRead, uint32_t &numOfBytesRead);	Read n numbers of bytes from receive FIFO.
uint8_t Write(uint8_t data);	Write a byte of data to the transmit FIFO.
uint8_t Write(uint8_t* array, uint32_t size);	Write n numbers of bytes to the transmit FIFO.
void SetDataFormat(FT_Parity_Type parityType, FT_Stop_Bit_Length stopBitLength, FT_Word_Length wordLength);	Set the communication data format.
void SetBaudRate(uint32_t baudrate);	Set the communication baud rate.
uint8_t GetTransmitFIFOFreeSpace();	Get how much free space in the transmit FIFO.
uint8_t GetReceiveFIFOFillLevel();	Get the fill level of the receive FIFO.
void SetInterrupts(uint8_t interrupts);	Enable or disable IRQ interrupts.
void EnableSleepMode(FT_OPTIONS option);	Enable or disable sleep mode.
void SWReset();	Perform a device reset.
uint8_t GetInterrupts(void);	Get all interrupts from the device.

Prototype	Description
FT_Status SetFlowControl(FT_FLOW_TYPE FlowControlType, uint8_t param1, uint8_t param2, uint8_t param3, uint8_t param4, uint8_t param5, uint8_t param6, uint8_t param7, uint8_t param8);	Initialize and use an enhanced flow control.
void SetInternalLoopBack(FT_OPTIONS option);	Enable or disable internal loop back mode.
void EnableFIFO(FT_OPTIONS option);	Enable or disable transmit and receive FIFO.
void EnableTransmitter(FT_OPTIONS option);	Enable or disable transmitter.
void EnableReceiver(FT_OPTIONS option);	Enable or disable receiver.
uint8_t GetReceiveFIFOStatus(void);	Get the status of the FIFO.
void SendXON1SpecialCharacter();	Send a XON1 special character for enhanced software flow control.
void SendXON2SpecialCharacter();	Send a XON2 special character for enhanced software flow control.
void SendXOFF1SpecialCharacter();	Send a XOFF1 special character for enhanced software flow control.
void SendXOFF2SpecialCharacter();	Send a XOFF2 special character for enhanced hardware and software flow control.
uint8_t ReadScratchPad(void);	Read the scratch pad register.
void EnableRTSOutput(FT_OPTIONS option);	Enable/disable the RTS signal pin.
bool isCTSActive(void);	Check the status of the CTS pin.
void WriteToScratchPad(uint8_t value);	Write to the scratch pad register.
FT_Status SetFIFOInterruptTriggerLevel(uint8_t TransmitFIFOInterruptLevel, uint8_t ReceiveFIFOInterruptLevel);	Set the FIFO interrupt trigger level.
void SetPresetFIFOInterruptTriggerLevel(FT_Preset_RX_FIFO_Trigger_Level ReceiveFIFOInterruptLevel, FT_Preset_TX_FIFO_Trigger_Level TransmitFIFOInterruptLevel);	Set the FIFO interrupt trigger level with pre-set value

Prototype	Description
void SetIOPinDirection(uint8_t Direction);	Set the GPIO pin direction.
uint8_t GetIOPinDirection(void);	Get the GPIO pin direction.
void SetIOPinState(uint8_t State);	Set the output GPIO pin state.
void StartTransmitLineBreak(void);	Continuously send a line break to the recipient.
void StopTransmitLineBreak(void);	Stop sending line breaks to the recipient.
bool isDataInReceiver(void);	Check to see if receiver FIFO has data in it.
bool isTransmitFIFOandShiftRegisterEmpty(void);	Check to see if both of the transmit FIFO and the transmit shift register are empty.
bool isTransmitFIFOEmpty(void);	Check to see if the transmit FIFO is empty or not.
void clearFIFO(void);	Clear the receive and transmit FIFO.
void Exit(void);	End the SPI

Table 3-1 - APIs description

4 Library Usage

4.1 Overview

The VM800P plug-in module Arduino Library has greatly reduced the amount of hardware specifics that the user needs to know. The following sections describe the simple setup for the hardware-assisted enhanced flow controls.

4.2 Initialization

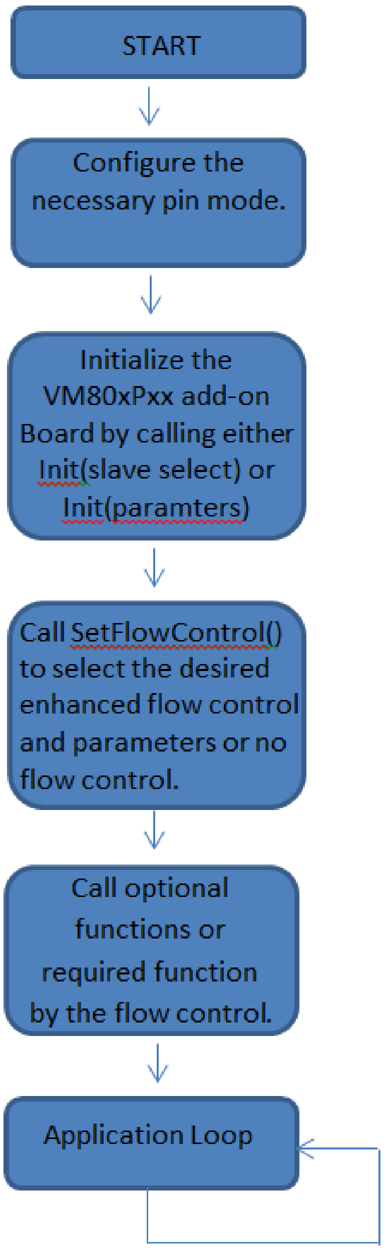

START
Configure the necessary pin mode.
Initialize the VM80xPxx add-on Board by calling either Init(slave select) or Init(paramters)
Call SetFlowControl() to select the desired enhanced flow control and parameters or no flow control.
Call optional functions or required function by the flow control.
Application Loop

Figure 4-1 : Library Initialization

The following setup is the same for the TTLU, N485U and 232U modules.

Library Include:

```
/* Arduino standard includes */
#include "SPI.h"
#include "Wire.h"

/* VM8xxpxx plug-in library includes. Include the appropriate header file*/
#include "FT_VI800A_TTLU.h" or   #include "FT_VI800A_232U.h" or #include "FT_VI800A_N485U.h"
```

Plug-in module Global Object instantiation:

```
FT_VI800A VI800A;
```

Arduino SPI pin configuration:

```
void setup()
{
            /* Initialize any pin mode*/

            /* Initialize serial print related functionality, optional*/
            Serial.begin(9600);

            VI800A.Init(Slave pin); or VI800A.Init(/*baud rate and data format*/);

            VI800A.SetFlowControl(…); //Configure flow control

            //Optional functions or required functions as described in the following sections.

}
```

The general VM800P plug-in module initialization consists of the global library object instantiation, call one of the overloaded *Init*() functions, setup flow control using the *SetFlowControl*(…) function, and any optional or required functions for the flow control.

4.3 Enhanced Flow Control

The VM800P plug-in modules can use one or more hardware-assisted, enhanced flow controls. The following table shows which enhanced flow controls can be used by the specific plug-in module. Specific enhanced flow control setup will be described with details in the following sections.

	Software manages flow control	**Enhanced Software Flow Control**	**Enhanced Hardware Flow Control**	**Enhanced Auto RS485 Flow Control**
VI800A-232U	x	x	x	-
VI800A-TTLU	x	x	x	-
VI800A-N485U	x	-	-	x

Table 4-1 - Modules and Flow Controls

2.1.1 4.3.1 Enhanced Software Flow Control

With enhanced software flow control, if the receiver is instructed to control the data flow with a certain special character(s) then the receiving device will automatically compare the incoming data with the XOFF1 and/or XOFF2 special character(s). Matching special character(s) are automatically discarded. When the correct combination of XOFF character(s) are detected, the data transmission is halted after the completion of receiving the current character till the correct combination of XON special character(s) are received.

If the transmitter is instructed to send a certain combination of special character(s) then the device will automatically monitors the receive FIFO according to the trigger levels set by the 7th and/or 8th parameter of the initialization function. The correct combination of XON1 and/or XON2 special character(s) are sent when the FIFO level is below the resume trigger level and the XOFF1 and/or XOFF2 special character(s) are sent when the FIFO level is above the halt trigger level.

To use enhanced software flow control, specify the following parameters in the *SetFlowControl* function:

Parameter 1: *FT_SW_Flow*,
Parameter 2: one group in the *FT_SW_Flow_Combination* table,
Parameter 3: an 8-bit value of the XON1 special character,
Parameter 4: an 8-bit value of the XON2 special character,
Parameter 5: an 8-bit value of the XOFF1 special character,
Parameter 6: an 8-bit value of the XOFF2 special character,
Parameter 7: one entry in *FT_Flow_Trigger* table,
Parameter 8: the halt level if *FT_Use_TCR_RX_Trigger* option was used for the 7th parameter or the pre-set trigger level in the *FT_Preset_RX_FIFO_Trigger_Level* table if the 7th parameter was *FT_Use_FCR_RX_Trigger*.
Parameter 9: the resume level if the *FT_Use_TCR_RX_Trigger* option was used for 7th parameter.

The valid values of the trigger levels for FT_Use_TCR_RX_Trigger option are from 0 to 60, with a granularity of 4. The halt level has to be greater than the resume level else the *SetFlowControl* will return an error.

4.3.1.1 Enhanced Software Flow Control Flow Combination

```
typedef enum FT_SW_Flow_Combination{
        NO_TX_FLOW = 0x00,  //no transmit flow control
        TX_XON1_XOFF1 = 0x08, //transmit xon1, xoff1
        TX_XON2_XOFF2 = 0x04, //transmit xon2, xoff2
        TX_XON1_XON2_XOFF1_XOFF2 = 0x0c, //transmit xon1 and xon2, xoff1 and xoff2
        NO_RX_FLOW = 0x00, //no receive flow control
        RX_XON1_XOFF1 = 0x02, //receiver compares xon1, xoff1
        RX_XON2_XOFF2 = 0x01, //receiver compares xon2, xoff2
        TX_XON1_XOFF1_RX_XON1_OR_XON2_XOFF1_OR_XOFF2 = 0x0b,//transmit xon1,
                xoff1 and receiver compares xon1 or xon2, xoff1 or xoff2
        TX_XON2_XOFF2_RX_XON1_OR_XON2_XOFF1_OR_XOFF2 = 0x07,//transmit xon2,
                xoff2 and receiver compares xon1 or xon2, xoff1 or xoff2
        TX_XON1_XON2_XOFF1_XOFF2_RX_XON1_XON2_XOFF1_XOFF2 = 0x0f,//transmit xon1
                and xon2, xoff1 and xoff2 and receiver compares xon1 and xon2, xoff1 and xoff2
        NO_TX_FLOW_RX_XON1_XON2_XOFF1_XOFF2 = 0x03,//no transmit flow control but
                receiver compares xon1 and xon2, xoff1 and xff2
}FT_SW_Flow_Combination;
```

The software flow control combination parameter specifies the special character(s) that the device will automatically send to the receiver when the FIFO has reached a FIFO trigger level and/or stop data transmission when the correct combination of special character(s) are received. There are two flow groups in the *FT_SW_Flow_Combination* table:

Group 1: The first 4 options in the table, *FT_SW_Flow_Combination*[0:3], are intended for the transmit flow control and the next 3 options, *FT_SW_Flow_Combination*[4:6], are intended for the receive flow control. Users can bitwise AND one in each of these two subgroups as the parameter value.

Group 2: The pre-set flow combinations of *FT_SW_Flow_Combination*[7:10].

4.3.1.2 Receive FIFO Trigger Level

```
typedef enum FT_Flow_Trigger{
        FT_Use_No_RX_Trigger=0,
        FT_Use_TCR_RX_Trigger=1,
        FT_Use_FCR_RX_Trigger=2,
}FT_Flow_Trigger;
```

If FIFO is enabled and one of the flow combinations referred to in the previous section is used then the special characters are sent based on the receive FIFO level set by the 8th and/or 9th parameter.

FT_Use_No_RX_Trigger option specifies no FIFO trigger will be used. This option is useful if the device will act purely as a sender or only a small amount of data will be received. The transmit and receive flow controls are not mutually exclusive so FT_SW_Flow_Combination[4:6] options can still be used to instruct the receiver to check for the incoming special character(s) to stop or start the data transmission.

FT_Use_TCR_RX_Trigger option instructs the device to use the TCR register which requires two trigger levels: halt level as the 7th parameter, and resume level as the 8th parameter. The valid value range for the trigger levels is 0 to 60 with a granularity of 4. The device will send the XOFF special character(s) when the receive FIFO is above the halt level and the XON special character(s) is sent when the FIFO level is below the resume level.

FT_Use_FCR_RX_Trigger option instructs the device to use the FCR register and one of the values in the *FT_Preset_RX_FIFO_Trigger_Level* table should be the 7th parameter. The value specifies the halt level and the resume level is the next lower value.

NOTE 1: If either *FT_Use_TCR_RX_Trigger* or *FT_Use_FCR_RX_Trigger* option is used then the library will also attempts to enable the transmit and receive FIFO. The FIFO should be enabled under the assumption that the lowest receive trigger level is 4 bytes.

2.1.2 4.3.2 Enhanced Hardware Flow control

Enhanced Hardware flow control is comprised of auto CTS and auto RTS. The RTS output pin automatically activates when there is enough room, below the specified halt trigger level, in the FIFO to receive data and de-activates the RTS output when the receive FIFO is sufficiently full, above the halt trigger level.

To use Hardware flow control, specify the following parameters in the *SetFlowControl* function:

Parameter 1: *FT_HW_Flow*
Parameter 2: One or more entries from the *FT_HW_FLOW_OPTIONS* table to use auto CTS and/or RTS.
Parameter 3: One entries from the *FT_Flow_Trigger* table, same trigger level options as described in the software flow control.
Parameter 4: the halt level if *FT_Use_TCR_RX_Trigger* option is used for the 3rd parameter or the pre-set trigger level in the *FT_Preset_RX_FIFO_Trigger_Level* table if the 3rd parameter is *FT_Use_FCR_RX_Trigger*.
Parameter 5: the resume level if the *FT_Use_TCR_RX_Trigger* option was used for 3rd parameter.

2.1.3 4.3.3 Enhanced RS485 Flow Control

To use RS485 flow control, specify the following parameters in the *SetFlowControl* function:

Parameter 1: *FT_RS485_Flow*
Parameter 2: One entry from the *FT_RS485_FLOW_OPTIONS* table
Parameter 3: The value of the address byte, if *FT_Auto_RS485_Address_Detection* option was specified for the 2nd parameter.

In RS-485 mode, a 'master' station transmits an address character followed by data characters for the addressed 'slave' stations. The VI800A-N485U supports two RS485 modes: normal multidrop mode and auto address detection mode.

In normal multidrop mode, *FT_RS485_Multidrop_Mode* option was used for 2nd parameter, the receiver is initially disabled and all address byte broadcasted by the 'master' will get pushed to the receive FIFO. The application needs to check the address byte and enables the receiver for the subsequent data from the 'master' if the address byte addresses its ID address. The application can disable the receiver upon receiving another address byte that's not for itself or a completion message from the 'master'.

In auto address detection mode, *FT_Auto_RS485_Address_Detection* option is used in the 2nd parameter, the device constantly monitors the receiving data for the address byte. Upon receiving an address byte that matches the ID address, as specified by the 3rd parameter, the device will automatically enables the receiver and push the address byte to the receive FIFO. The receiver will then receive the subsequent data from the 'master' station until disabled by the application after having received a completion message from the 'master' station or automatically disabled if another address byte is received and it does not match the address byte specified in the 3rd parameter.

All RS485 address bytes are assumed to have a parity type of 1 so the library *Init* fucntion must use *FT_Forced_Parity_Zero* option for the parity type parameter. If the master is another plug-in module then it must change the data format by using the *SetDataFormat()* function with *FT_Forced_Parity_One* parity type to send an address byte and revert back the data format for data characters.

4.4 Receive FIFO Status

```
typedef enum FT_FIFO_Status{
        FT_FIFO_OK=0,
        FT_Overrun_Error=2,
        FT_Parity_Error=4,
        FT_Framing_Error=8,
        FT_Break_Interrupt=16,
}FT_FIFO_Status;
```

Sometimes errors might occur in the receive FIFO such as data overrun, data frame error, data parity error, and line break condition. Calling the *GetReceiveFIFOStatus* function and comparing the entries in the *FT_FIFO_Status* table can determine the status of the receive FIFO. Character errors, FT_Parity_Error, FT_Framing_Error, and FT_Break_Interrupt, are identified by reading the FIFO because those three errors are indicating the character status on the top of the receive FIFO (next character to be read). More than one character error can occur and the character errors persist till the character is read from the FIFO. FT_Overrun_Error indicates whether a FIFO overrun condition has occurred. The *GetReceiveFIFOStatus* function returns FT_FIFO_OK if no errors were detected. NOTE: Polling the status from the FIFO should not enable the following interrupts (no interrupts are enabled by default): *FT_Modem_Status_Interrupt*, *FT_Receive_Line_Status_Interrupt*, *FT_Transmit_Holding_Register_Interrupt*, and *FT_Receive_Holding_Register_Interrupt*.

4.5 Interrupts

```
typedef enum FT_Enableable_Interrupts{
        FT_Enable_All_Interrupts=239,
        FT_CTS_Interrupt=128,
        FT_RTS_Interrupt=64,
        FT_Xoff_Interrupt=32,
        FT_Modem_Status_Interrupt=8,
        FT_Receive_Line_Status_Interrupt=4,
        FT_Transmit_Holding_Register_Interrupt=2,
        FT_Receive_Holding_Register_Interrupt=1,
        FT_Enable_No_Interrupts=0,
}FT_Enableable_Interrupts;
```

Interrupts can be used in some situations instead of excessively polling the device. The *FT_Enableable_Interrupts* table has all the available device interrupts. Interrupts are disabled/enabled by the *SetInterrupts* function. *FT_Enable_All_Interrupts* and

FT_Enable_No_Interrupts entries are not interrupts but they can be used as a quick way to enable all interrupts or none at all.

```
typedef enum FT_Interrupts_Status{
          FT_No_Interrupts = 255
          FT_Receiver_Line_Status_Error = 6,
          FT_Receiver_Time_Out_Interrupt = 12,
          FT_RHR_Interrupt=4,
          FT_THR_Interrupt=2,
          FT_Modem_Interrupt=0,
          FT_Input_Pin_State_Change=48
          FT_Received_Xoff_Signal=16,
          FT_CTS_RTS_Inactive=32,
}FT_Interrupts_Status;
```

After the desired interrupt(s) and an ISR have been set, as shown in Section 8.1.2, the application should compare the result from the *GetInterrupts* function against the entries in the *FT_Interrupts_Status* table. Interrupts are prioritized and only the highest priority interrupt is reported so the application should check the interrupts in the order presented in the *FT_Interrupts_Status* table. The *GetInterrupts* function returns the *FT_No_Interrupts* status if there is no interrupt.

NOTE: The *SetPresetFIFOInterruptTriggerLevel* function uses the predefined trigger levels in the FCR register for FIFO interrupts which is the same register that the *SetFlowControl* function sets if the application specifies *FT_Use_FCR_RX_Trigger* flow control trigger levels. Both of them can’t be used at the same time.

4.6 Baud Rate

The desired baud rate is one of the parameter values for the overloaded *Init()* library function. The valid range of the baud rate is 15 to 921600. NOTE: not all baud rates within the range are "supported" because of the rounding error on some "non-standard" baud rates. The final baud rate uses the following equation to calculate a divisor for the internal baud rate generator logic unit: divisor = 14745600 / (baud rate*16). The 16-bit divisor can only take integer values, so if the division did not yield an integer value the actual baud rate will be a little different to the desired rate. The following table shows the desired baud rates, actual device baud rate and the percentage difference.

Desired baud rate:	Internal baud rate:	Percentage error difference between desired and actual:
15	15	0
50	50	0
75	75	0
110	110.0023872	0.002170158
134	134.0119238	0.008897962
150	150	0
300	300	0
600	600	0
1200	1200	0
1800	1800	0
2000	2003.478261	0.173761953
2400	2400	0
3600	3600	0
4800	4800	0
7200	7200	0
9600	9600	0
19200	19200	0
38400	38400	0
56000	57600	2.816901408
57600	57600	0
115200	115200	0
230400	230400	0
460800	460800	0
921600	921600	0

Table 4-2 - Baud Rates

5 Setup steps

5.1 Hardware Setup and Pin Connections

There are two 'Comm Daughter' connectors on the back of the VM800P board and the TTLU, 232U, and N485U plug-in modules should only be connected on the right side connector (J6).

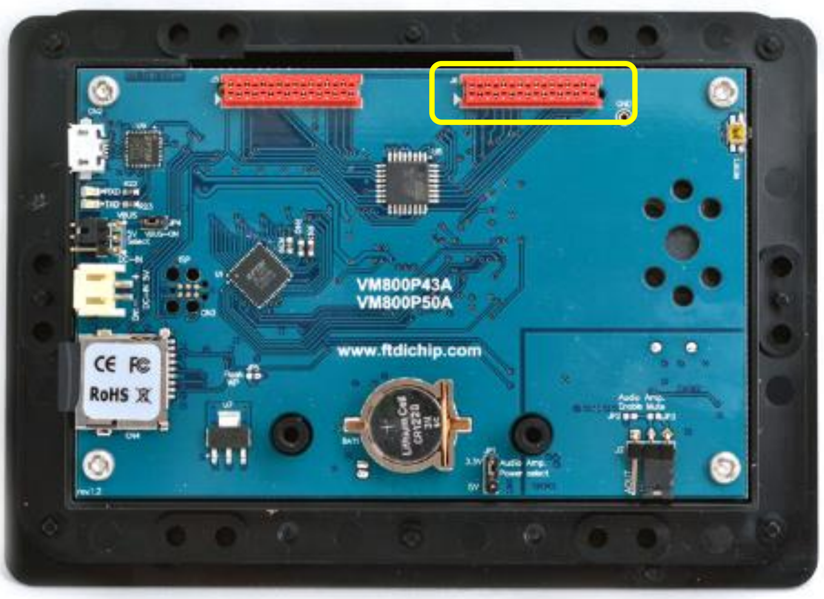


Figure 5-1 : VM800P Board

The VI800A-TTLU plug-in module is capable of using the enhanced software and hardware flow control or application managed flow control. The following tables listed out which pins must be connected for the particular flow control:

Enhanced Software Flow Control	
From:	To:
TXD	RXD
RXD	TXD
GND	GND

Table 5-1 - TTLU Enhanced Software Flow Control connections

Hardware Flow Control	
From:	To:
TXD	RXD
RXD	TXD
RTS#	CTS#
CTS#	RTS#
GND	GND

Table 5-2 - TTLU Hardware Flow Control connections

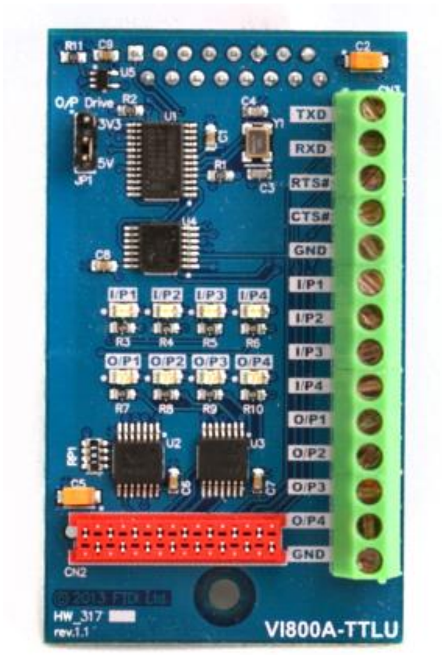


Figure 5-2 : VI800A-TTLU module

The VI800A-232U plug-in modules are capable of using the enhanced software and hardware flow control or application managed flow control. The following tables list which pins must be connected for the particular flow control:

Enhanced Software Flow Control	
From:	To:
TXD	RXD
RXD	TXD
GND	GND

Table 5-3 - 232U Enhanced Software Flow Control connections

Hardware Flow Control

From:	To:
TXD	RXD
RXD	TXD
RTS	CTS
CTS	RTS
GND	GND

Table 5-4 - 232U Hardware Flow Control connections

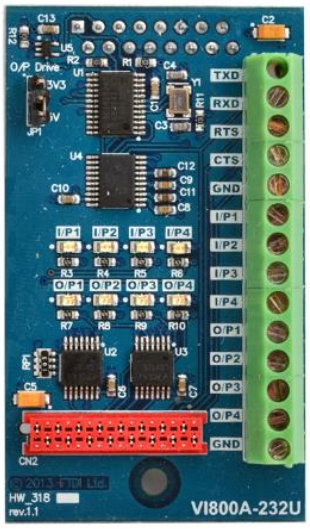


Figure 5-3 : VI800A-232U module

The VI800A-N485U plug-in modules are capable of using enhanced RS-485 flow control or application managed flow control. The following tables list which pins must be connected for the enhanced RS-485 flow control:

Enhanced RS-485 Flow Control	
From:	To:
Rx+	TX+
Rx-	TX-
Tx+	Rx+
Tx-	Rx-
GND	GND

Table 5-5 - N485U Flow Control connections

It's important to make sure the jumper JP2 is connected to the ON position. Connecting the module loop back jumpers have the same effect as connecting the module's output pins to the input pins of itself and the receiver unit. Termination resistor jumpers end the RS485 line.

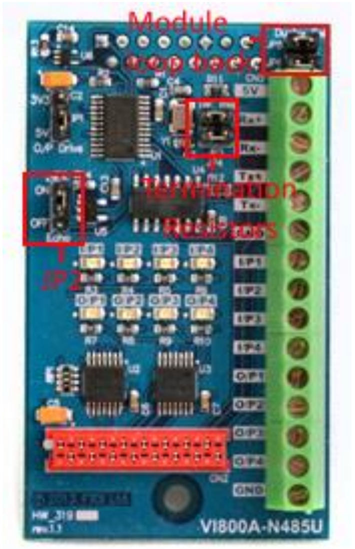


Figure 5-4 : VI800A-N485U Module

5.2 Library Sketches

This section demonstrates how to access the sample Arduino sketches of the VM800P plug-in module Arduino library and upload the sketch to the VM800P board.

A successfully installed library will contain "FTDI" as one of the items in the Examples list menu and hovering over it will reveal the available sketch for the particular platform as shown below. If the plug-in library was added to an existing FTDI library then the "FTDI" folder will also contain all the other sample sketches.

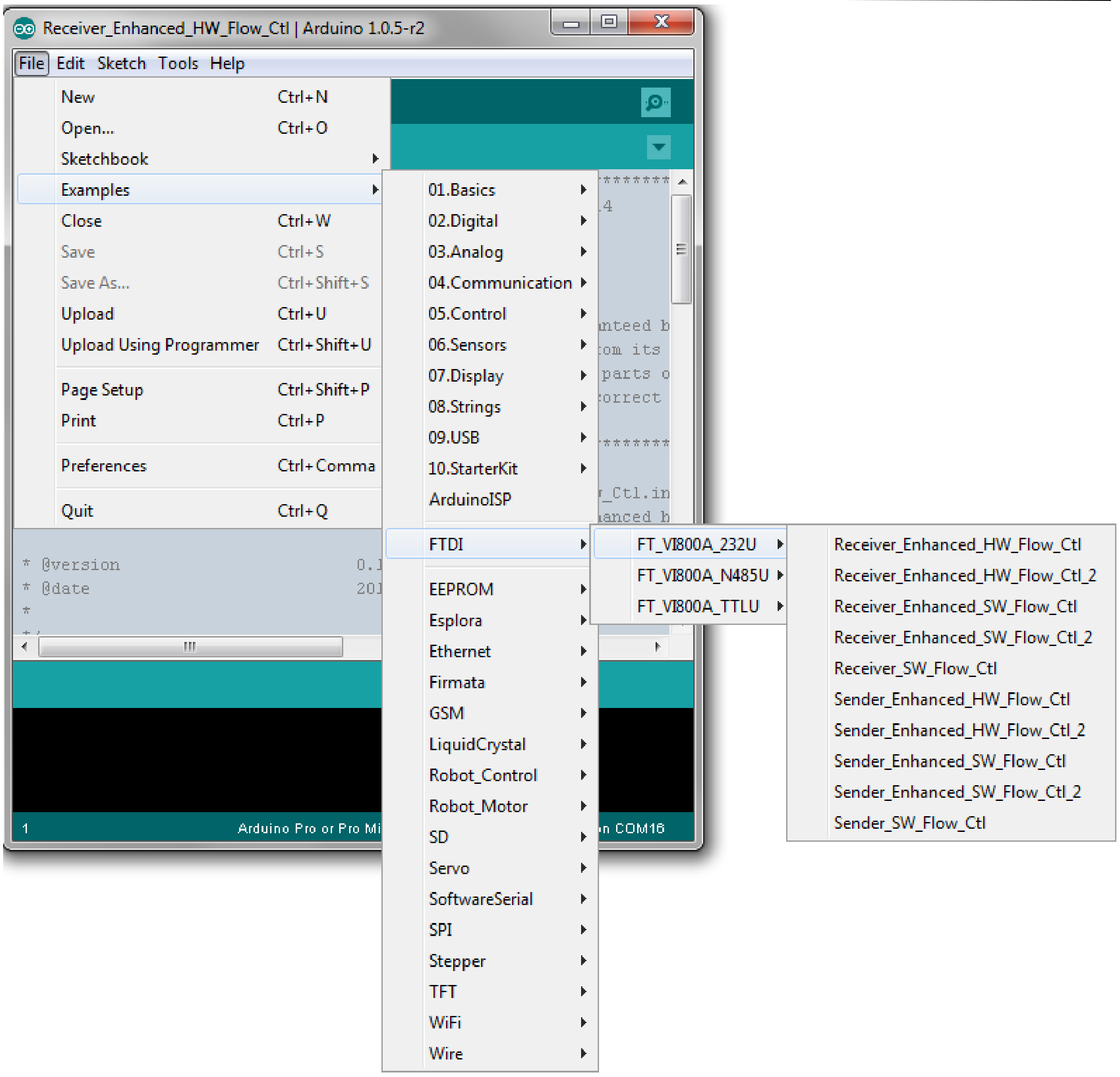
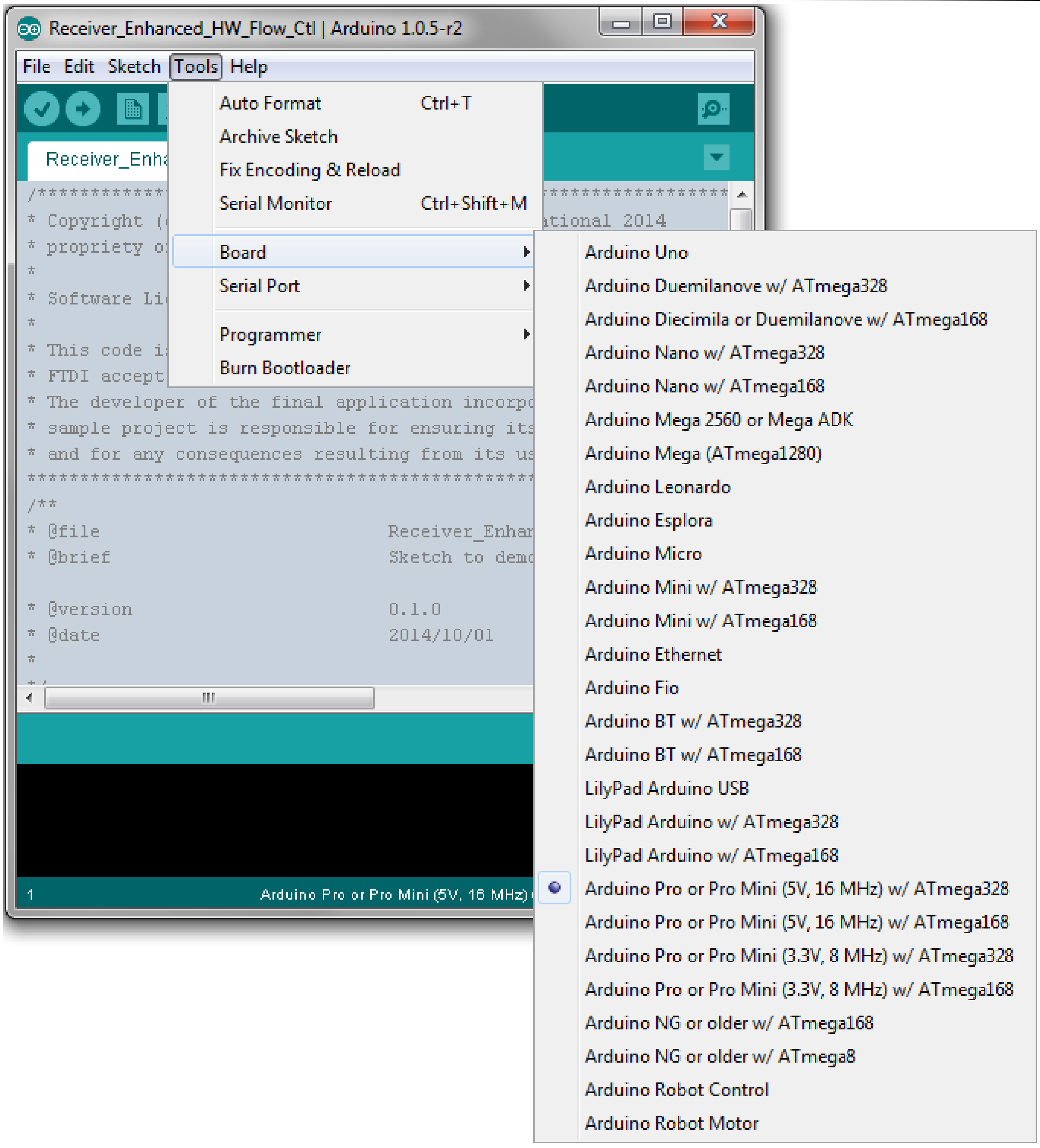


Figure 5-5 : Arduino IDE – example sketch

Clicking on any one of the example sketches will open it up with a new sketch window. It can be examined and/or uploaded to the Arduino board by selecting the correct board type and Serial Port first then choosing Upload. For the VM800P plug-in example sketches, the board choice must be the "Arduino Pro or Pro Mini (5V, 16MHz) w/ Atmega328".

After making the wiring connections for the respective module as described in Section 5.1, the sketch can be uploaded to the board.


Receiver_Enhanced_HW_Flow_Ctl | Arduino 1.0.5-r2
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
Receiver_Enha
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Robot Control
Arduino Robot Motor

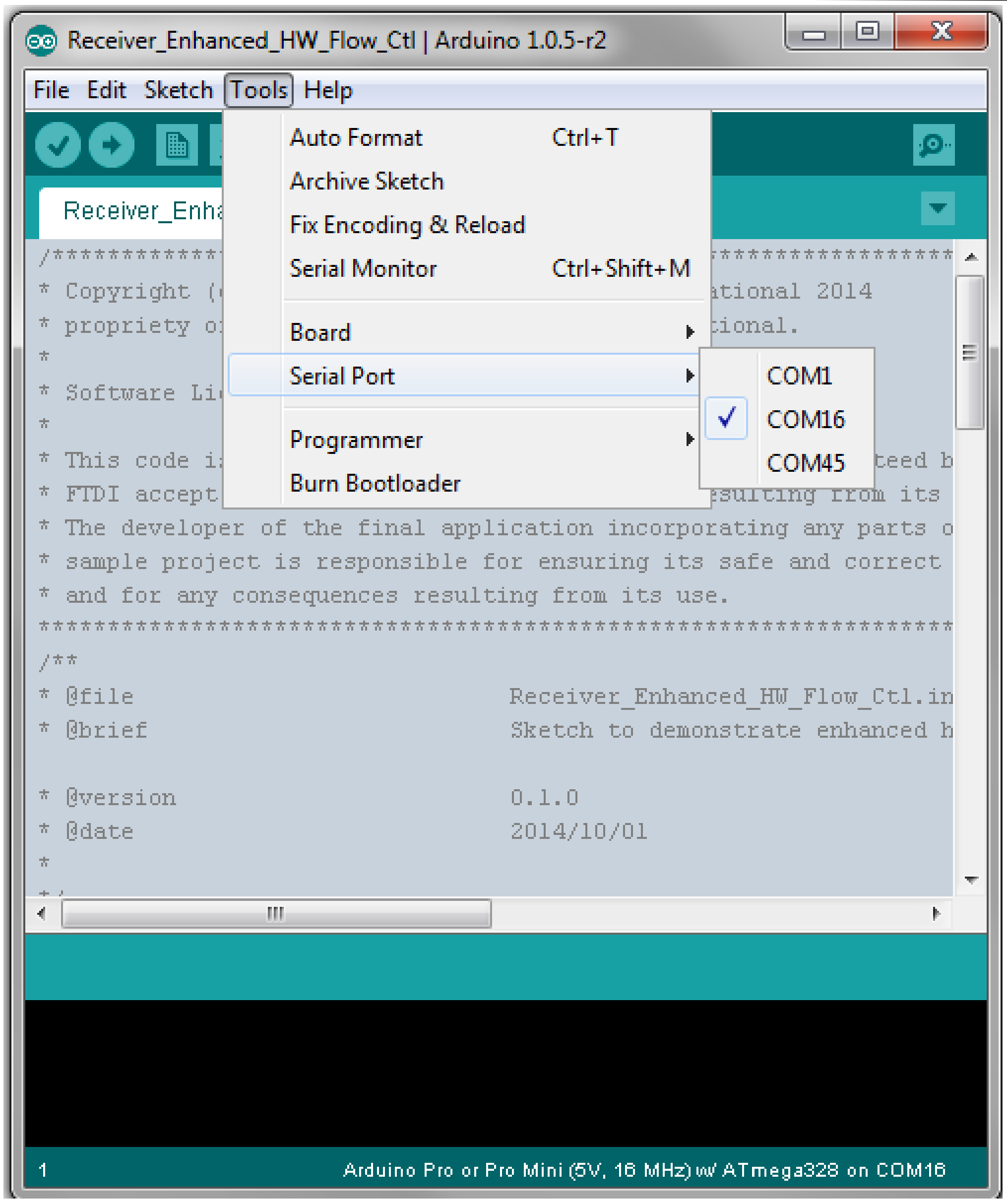
Board: ● Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328

```
/************                                  ******************
* Copyright (                                  ational 2014
* propriety o
*
* Software Li
*
* This code i
* FTDI accept
* The developer of the final application incorpo
* sample project is responsible for ensuring its
* and for any consequences resulting from its us
*************************************************
/**
* @file                             Receiver_Enhar
* @brief                            Sketch to demo

* @version                          0.1.0
* @date                             2014/10/01
*
```

```
1                                   Arduino Pro or Pro Mini (5V, 16 MHz)
```

Figure 5-6 : Arduino IDE -Board selection



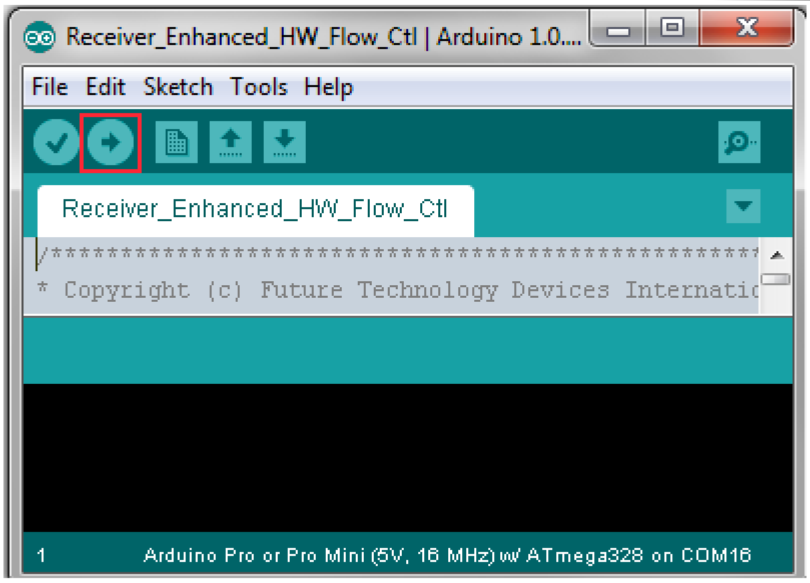
[✓ COM16]

```
/*************                    ******************
* Copyright (                      ational 2014
* propriety o                      tional.
*
* Software Li
*
* This code i                                  teed b
* FDTI accept                      esulting from its
* The developer of the final application incorporating any parts o
* sample project is responsible for ensuring its safe and correct
* and for any consequences resulting from its use.
*******************************************************************
/**
* @file                          Receiver_Enhanced_HW_Flow_Ctl.in
* @brief                         Sketch to demonstrate enhanced h

* @version                       0.1.0
* @date                          2014/10/01
*
```

```
1          Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328 on COM16
```

Figure 5-7 : Arduino IDE – Serial port selection


Receiver_Enhanced_HW_Flow_Ctl | Arduino 1.0....
File Edit Sketch Tools Help
Receiver_Enhanced_HW_Flow_Ctl

```
/*************************************************
* Copyright (c) Future Technology Devices Internatio
```

1 Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328 on COM16

Figure 5-8 : Arduino IDE – Upload Button

Once the sample sketch has been uploaded, the user can observe the sketch print-outs by clicking the Serial Monitor button, as indicated by the following figure.

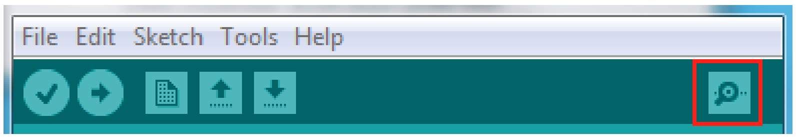


Figure 5-9 : Arduino IDE - Serial Monitor

6 VI800A-232U Examples

This example uses example sketches in the FT_VI800A_232U folder. The following example sketches demonstrate two scenarios –VI800A-232U to VI800A-232U and VI800A-232U to RS-232 cable communication.

6.1 Setup and Run

After making the pin connection as described in Section 5.1, it should look something like the following figure. NOTE: The 232U example sketches are using the enhanced software flow control so the RTS and CTS pins are not required to be connected but the two pins are required for hardware flow control.

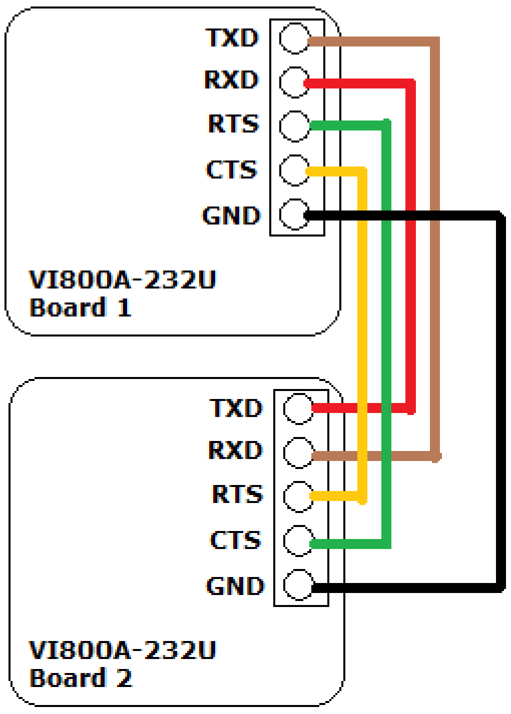


Figure 6-1 : VI800A-232U to VI800A-232U

Once the hardware setup is done and the example sketches have been uploaded to one of each board, the Serial Monitor should be enabled as explained in Library Sketches section. The software flow control example sketches will not start by themselves unless the user has sent a letter 'p' to the sender sketch's Serial Monitor, as indicated in the following figure.

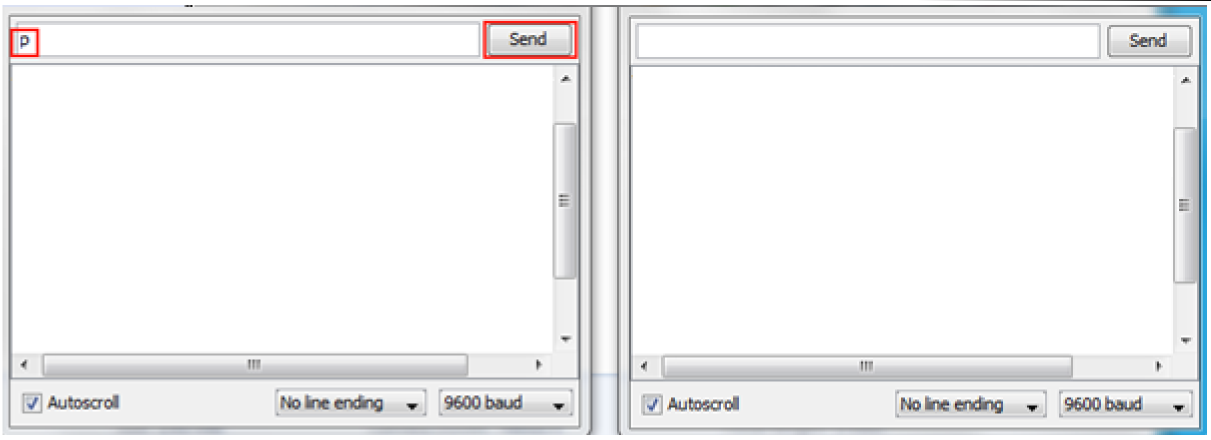


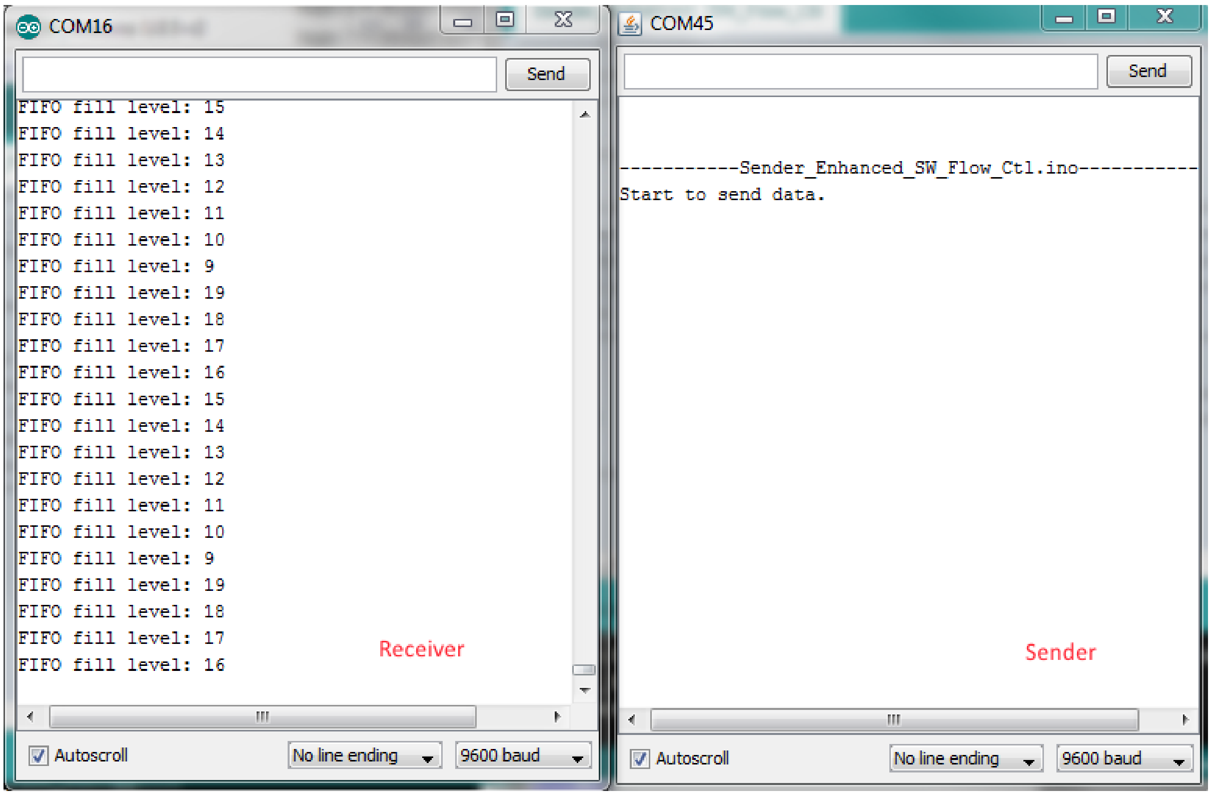
Figure 6-2 : VI800A-232U Serial Monitor #1

6.2 VI800A-232U to VI800A-232U Software Flow Control

6.2.1 Example 1

For this example, two VM800P boards and VI800A-232U modules, Receiver_Enhanced_SW_Flow_Ctl and Sender_Enhanced_SW_Flow_Ctl example sketches in the FT_VI800A_232U folder are needed.

The following figure is a snapshot of the data transmission output in the Arduino serial monitors. Notice the receiver's FIFO fill level is always within a certain range due to the receive trigger levels.



```
FIFO fill level: 15
FIFO fill level: 14
FIFO fill level: 13
FIFO fill level: 12
FIFO fill level: 11
FIFO fill level: 10
FIFO fill level: 9
FIFO fill level: 19
FIFO fill level: 18
FIFO fill level: 17
FIFO fill level: 16
FIFO fill level: 15
FIFO fill level: 14
FIFO fill level: 13
FIFO fill level: 12
FIFO fill level: 11
FIFO fill level: 10
FIFO fill level: 9
FIFO fill level: 19
FIFO fill level: 18
FIFO fill level: 17
FIFO fill level: 16
```

```
-----------Sender_Enhanced_SW_Flow_Ctl.ino-----------
Start to send data.
```

Figure 6-3 : VI800A-232U Serial Monitor #2

6.2.1.1 Code Explanation

In this example, if the user sent a letter 'p' in the sender's serial monitor, the sender, Sender_Enhanced_SW_Flow_Ctl.ino, will keep sending the alphabet letters till the receiver, Receiver_Enhanced_SW_Flow_Ctl.ino, has reached its halt trigger level and resumes data transmission after FIFO is below the resume level.

6.2.1.1.1 Sender

```
VI800A.Init(FT_232U_CS);

VI800A.SetFlowControl(FT_SW_Flow,NO_TX_FLOW_RX_XON1_XON2_XOFF1_XOFF2,48,48,49,4
9, FT_Use_No_RX_Trigger);

//enable receive FIFO
VI800A.EnableFIFO(FT_Enable);
VI800A.clearFIFO();
```

The sender has the above setup sequence which performs the following:

- Use the overloaded Init() function to initialize the device with the default values of: baud rate – 9600, parity type – none, stop bits – 1, data bits – 8.
- In the SetFlowControl function, the sender will use enhanced software flow control, *FT_SW_Flow*, with a flow combination that will only check the incoming data for the XON1 special character followed by XON2 special character to resume data transmission and XOFF1 followed by XOFF2 special character for stop data transmission, *NO_TX_FLOW_RX_XON1_XON2_XOFF1_XOFF2*. The receiver in this example will not send any kind of data to the sender except the special characters, so it doesn't make sense for the sender to have any transmit flow combination [1]. The following four integer parameter values are for XON1, XON2, XOFF1, and XOFF2 special characters, in respective order, and they can be any 8-bit value integer. It's important that the same special characters are being used by the sender and the receiver. NOTE: two special characters for resume and halt are recommended if the transferring data is in hex format, otherwise non-printable ASCII value are sufficient if the transferring data consist only of printable characters. The final parameter, *FT_Use_No_RX_Trigger,* indicates the sender will not use any receiving FIFO trigger level to send special characters to the receiver because the receiver will not send any data to the sender.
- The *EnableFIFO(FT_Enable)* library function explicitly enables the FIFO because the *SetFlowControl* function attempts to enable the FIFO if the receive trigger level is specified, but since no receive trigger level is specified then the FIFO is not enabled. The configuration in this example requires the sender to enable the FIFO because only one byte is used if the FIFO is not enabled. The software flow combination of the receiver will send two special characters for the signal to halt and resume data transmission so the FIFO must be enabled in order to prevent data overflow.

It's strongly recommended to enable the FIFO whenever possible, even if the software flow combination consists of one single signaling character. Situations such as the special character sequence of: resume followed by the halt character but the device might not be able to detect the halt character if the special characters are being sent in quick succession and the device was busy.

[1] Whenever enhanced software flow control is used, the device will automatically discard the special character(s) and they wouldn't appear in the receiving FIFO.

```
        if((VI800A.GetTransmitFIFOFreeSpace()) && manualStart){
                VI800A.Write(value[valueIndex]);
                valueIndex=(valueIndex+1)%26;
        }
```

The above code snippet is all the sender needs to send the data to the receiver. The *GetTransmitFIFOFreeSpace()* function is used to make sure no transmit FIFO overflow.

6.2.1.1.2 Receiver

```
…
VI800A.Init(FT_232U_CS);
…
VI800A.SetFlowControl(FT_SW_Flow,TX_XON1_XON2_XOFF1_XOFF2,48,48,49,49,
        FT_Use_FCR_RX_Trigger, FT_RX_Sixteen_Characters);
VI800A.clearFIFO();
```

The Receiver has the above initialization setup and it performs the following:

- Initialize the device with default baud rate and data format.
- In the *SetFlowControl* function, the receiver will use the enhanced software flow control, *FT_SW_Flow*. The software flow combination, *TX_XON1_XON2_XOFF1_XOFF2*, specifies the device will only send XON1 and XON2 to signal the sender to resume and XOFF1 followed by XOFF2 to halt the data transmission, the special characters are the same as the ones used by the sender. The receiver will use FCR receive FIFO trigger levels and the pre-set level of 16 characters.

In the above figure, notice the receive FIFO level is always around 9 to 19 but the halt trigger level is set to 16 characters and the resume level is 8, next lower pre-set level. Due to the overhead to send the special characters, the actual FIFO trigger levels might not be exactly as the specified level and it could be a couple of characters passed the trigger levels.

```
incomingData = VI800A.Read();
        if(g_var != incomingData)
        {
                Serial.print(g_var);
                Serial.print(' ');
                Serial.print(incomingData);
                Serial.print(' ');
        }
        delay(100);
        g_var++;
        Serial.print("FIFO fill level: ");
        Serial.println(VI800A.GetReceiveFIFOFillLevel());
        if(g_var > 'Z')
        {
                g_var = 'A';
        }
```

The receiver ensures the correctness of the flow control with its own check on the incoming data against its own expected value and the alphabet characters should match up even if running the two sketches for a prolonged period of time.

6.2.2 Example 2

This example uses the Receiver_Enhanced_SW_Flow_Ctl_2 and Sender_Enhanced_SW_Flow_Ctl_2 examples in the FT_VI800A_232U folder and FIFO status functionality is being demonstrated.

6.2.2.1 Code Explanation

6.2.2.1.1 Sender

The sender, Sender_Enhanced_SW_Flow_Ctl_2.ino, has almost the same code as the sender of the previous example and the only major differences are:

```
else if(command == 'q'){
        readData^=1;
        if(readData){
                Serial.println("Read data from FIFO.");
                VI800A.EnableFIFO(FT_Enable);
                VI800A.clearFIFO();
        }
        else{
                Serial.println("Stack data in FIFO");
                VI800A.EnableFIFO(FT_Disable);
                VI800A.clearFIFO();
        }
}
```

The above code section toggles the FIFO every time the user presses the letter 'q'. If reading the FIFO is disabled and if the user sends more than 1 character to the sender from the receiver's serial output then the device will raise an overflow error, reported by the *GetReceiveFIOFStatus()* function.

```
void GetReceiveFIOFStatus(){
        fifoStatus = VI800A.GetReceiveFIFOStatus();
        if(fifoStatus==FT_FIFO_OK){
                return;
        }
        else{
                if((fifoStatus&(uint8_t)FT_Overrun_Error)==FT_Overrun_Error)
                        Serial.println("Data Overflow occurred.");
                else if((fifoStatus&(uint8_t)FT_Parity_Error)==FT_Parity_Error)
                        Serial.println("Data Parity Error occurred.");
                else if((fifoStatus&(uint8_t)FT_Framing_Error)==FT_Framing_Error)
                        Serial.println("Data Framing Error occurred.");
                else if((fifoStatus&(uint8_t)FT_Break_Interrupt)==FT_Break_Interrupt)
                        Serial.println("Line Break detected.");
        }
}
```

The *GetReceiveFIFOStatus* library function reports any errors for the whole receive FIFO as well as the next character to be read, as described in Section 4.4. FT_Parity_Error, FT_Framing_Error, and FT_Break_Interrupt FIFO errors represent the top of the FIFO, the next character to be read, and FT_Overrun_Error indicates whether an overrun event has occurred or not.

6.2.2.1.2 Receiver

The only place where the receiver, Receiver_Enhanced_SW_Flow_Ctl_2.ino, is different from the previous receiver is the following:

```
if(command == 'l'){ //send line break condition
        VI800A.StartTransmitLineBreak();
        delay(5);
        VI800A.StopTransmitLineBreak();
}
```

The above code snippet sends out the line break condition to the sender whenever the user pressed the letter 'l'. Line break condition is represented by the value of 0x00 in the FIFO as well as the *FT_Break_Interrupt* FIFO status. *StartTransmitLineBreak* library function continuously sends out the line break condition and the *StopTransmitLineBreak* function must be called to stop the transmission of the line break condition.

6.2.3 Example 3

This example uses the Receiver_SW_Flow_Ctl.ino and Sender_SW_Flow_Ctl.ino example sketch. Instead of using enhanced flow control, this section demonstrates the scenario where either the sender or the receiver is unable to use enhanced software flow control. This example also demonstrates how to set the GPIO pin state to enable the on-board LEDs.

The sender will start to send data to the receiver whenever the user presses the letter 'p' in the serial monitor.

6.2.3.1 Sender

```
//set data format, same as calling VI800A.Ini(FT_232U_CS);
VI800A.Init(FT_232U_CS,9600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);

//use hardware flow control but no FIFO trigger
VI800A.SetFlowControl(FT_No_Flow);
VI800A.EnableFIFO(FT_Enable);
VI800A.SetIOPinDirection(0xF0);
VI800A.SetIOPinState(0xFF);
```

The sender has the above setup section. It's almost the same as all the other examples except the *SetIOPinDirection* and *SetIOPinState* functions are used to set the direction of the GPIO pins and the state of the output pins. Unless the output pins are connected to the respective input pin, the input GPIO pins will not light up. Also, the device has a fixed direction for the GPIO pins so the pin direction value should always be 0xF0.

```
        if(VI800A.isDataInReceiver()){
                incomingData = VI800A.Read();
                if(incomingData==XON){  //signal for data transmission
                        transmitData=1;
                        VI800A.SetIOPinState(IOPinStates[IOPinStatesIndex]);  //enable the next
LED in the sequence
                        IOPinStatesIndex=(IOPinStatesIndex+1)&0x03;
                }
                else if(incomingData==XOFF) //signal to halt data transmission
                        transmitData=0;
                else{
                        Serial.print("Received: ");
                        Serial.println(incomingData);
                }
        }
```

Whenever enhanced software flow control is used, the device automatically detects, performs the appropriate functions and discards the special character(s). In a pure application controlled software flow, a similar function in the above code snippet is required to manually detect the special character(s) to halt and resume data transmission. In this example sketch, XON special character has the value of 0x01 and XOFF special character has the value of 0x02. The *SetIOPinState* function lights up the next LED in the sequence when the sender received a XON special character.

6.2.3.2 Receiver

```
        /* Initialize serial print related functionality */
        Serial.begin(9600);
   Serial.println("\n\r\n\r-----------Sample Application for 232U--------------");

        VI800A.Init(FT_232U_CS,9600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);
        VI800A.SetFlowControl(FT_No_Flow);
        VI800A.EnableFIFO(FT_Enable);
```

The receiver has the above setup section.

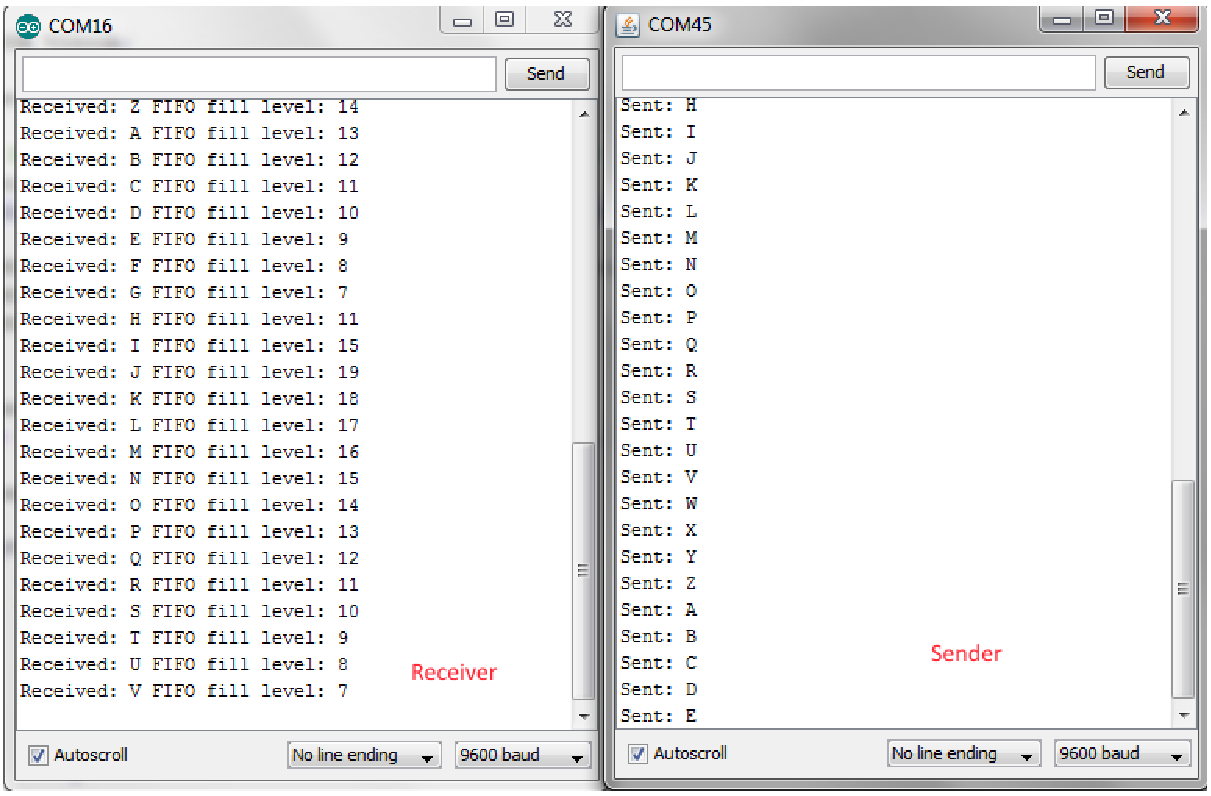
```
char  XON=0x01,XOFF=0x02;

uint8_t haltlvl=16, resumelvl=8, curFIFOlvl, XONSent=0, XOFFSent=0;
```

The receiver has its own halt and resume FIFO trigger level and the state of the data transmission. The halt and resume levels are the same as the previous examples.

```
curFIFOlvl = VI800A.GetReceiveFIFOFillLevel();
if(curFIFOlvl>haltlvl){
        if(XOFFSent==0){
                VI800A.Write(XOFF);
                XOFFSent=1;
                XONSent=0;
        }
}
else if(curFIFOlvl<resumelvl){
        if(XONSent==0){
                VI800A.Write(XON);
                XONSent=1;
                XOFFSent=0;
        }
}
```

In the above code snippet, the receiver is required to constantly monitor the receive FIFO fill level. The appropriate special character was sent whenever a trigger level has been reached.



```
Received: Z FIFO fill level: 14
Received: A FIFO fill level: 13
Received: B FIFO fill level: 12
Received: C FIFO fill level: 11
Received: D FIFO fill level: 10
Received: E FIFO fill level: 9
Received: F FIFO fill level: 8
Received: G FIFO fill level: 7
Received: H FIFO fill level: 11
Received: I FIFO fill level: 15
Received: J FIFO fill level: 19
Received: K FIFO fill level: 18
Received: L FIFO fill level: 17
Received: M FIFO fill level: 16
Received: N FIFO fill level: 15
Received: O FIFO fill level: 14
Received: P FIFO fill level: 13
Received: Q FIFO fill level: 12
Received: R FIFO fill level: 11
Received: S FIFO fill level: 10
Received: T FIFO fill level: 9
Received: U FIFO fill level: 8
Received: V FIFO fill level: 7
```

```
Sent: H
Sent: I
Sent: J
Sent: K
Sent: L
Sent: M
Sent: N
Sent: O
Sent: P
Sent: Q
Sent: R
Sent: S
Sent: T
Sent: U
Sent: V
Sent: W
Sent: X
Sent: Y
Sent: Z
Sent: A
Sent: B
Sent: C
Sent: D
Sent: E
```

Figure 6-4 : Software Flow Control output

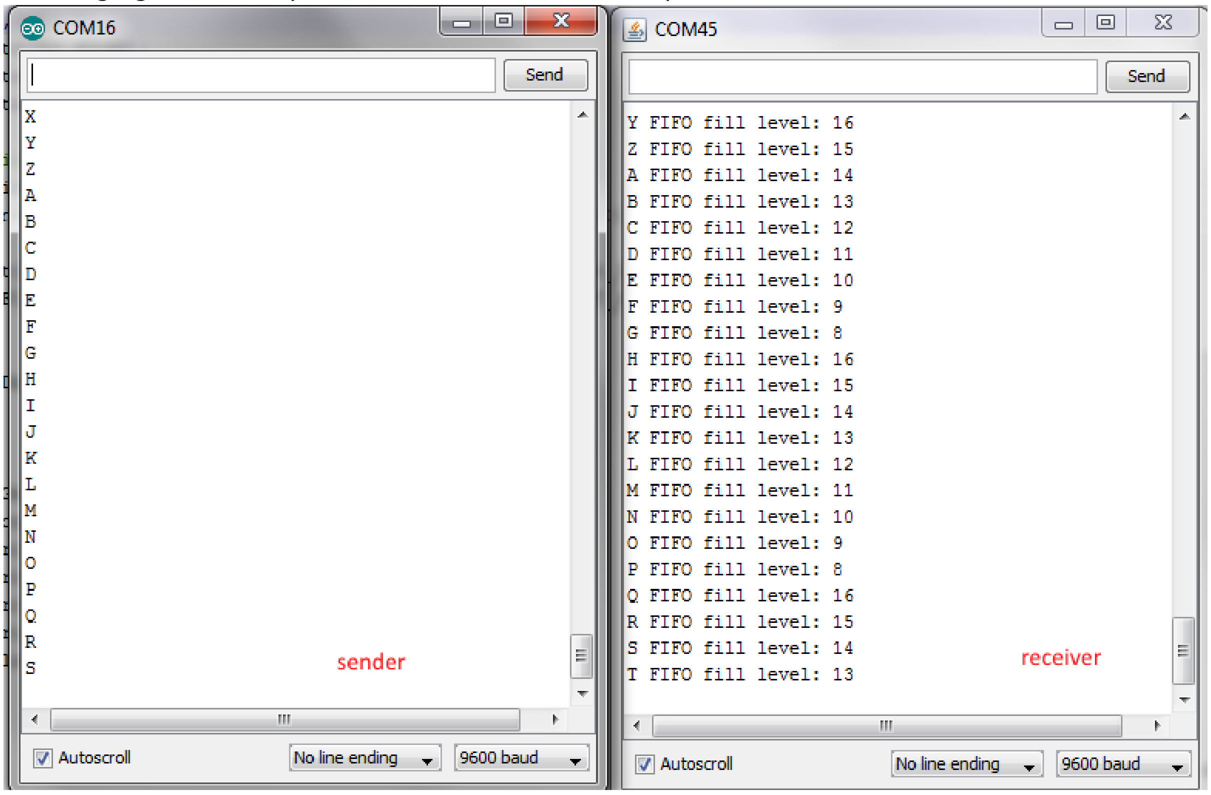
One major difference between application controlled software flow and enhanced software control is the amount of overhead. As seen in the serial outputs of the previous examples, the minimum and maximum numbers of the receiver's fill level are only 2 or 3 characters beyond the trigger levels. In the above figure, the FIFO fill level reached 19 characters before the sender halted data transmission and that's only when the sender has a 20 milliseconds time delay because the example wouldn't work if there is no delay or the delay is too short.

6.3 VI800A-232U to VI800A-232U Hardware Flow Control

This example uses the same hardware setup as the VI800A-232U to VI800A-232U Software Flow control.

6.3.1 Example 1

This example uses the Sender_Enhanced_HW_Flow_Ctl.ino and Receiver_Enhanced_HW_Flow_Ctl.ino example sketches in the FT_VI800A_232U folder. The data transmission starts whenever the user sends a letter 'p' to the sender's serial monitor. The following figure is a snapshot of the serial terminal outputs.



```
X
Y
Z
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
```

```
Y FIFO fill level: 16
Z FIFO fill level: 15
A FIFO fill level: 14
B FIFO fill level: 13
C FIFO fill level: 12
D FIFO fill level: 11
E FIFO fill level: 10
F FIFO fill level: 9
G FIFO fill level: 8
H FIFO fill level: 16
I FIFO fill level: 15
J FIFO fill level: 14
K FIFO fill level: 13
L FIFO fill level: 12
M FIFO fill level: 11
N FIFO fill level: 10
O FIFO fill level: 9
P FIFO fill level: 8
Q FIFO fill level: 16
R FIFO fill level: 15
S FIFO fill level: 14
T FIFO fill level: 13
```

Figure 6-5 : VI800A-232U Serial Monitor #3

6.3.1.1 Code Explanation

6.3.1.1.1 Sender

```
VI800A.Init(FT_232U_CS,9600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);

VI800A.SetFlowControl(FT_HW_Flow, FT_Use_No_RX_Trigger);
VI800A.EnableFIFO(FT_Enable);
```

The sender has the above setup to initialize the device with default values. Similar to software flow control parameters, if *FT_Use_No_RX_Trigger* option was specified then the library will not enable the FIFO so the FIFO must be enabled explicitly. The sender will not receive large amounts of data in this example so no FIFO trigger levels are specified.

6.3.1.1.2 Receiver

```
VI800A.Init(FT_232U_CS,9600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);
VI800A.SetFlowControl(FT_HW_Flow, FT_Use_FCR_RX_Trigger, FT_RX_Sixteen_Characters);
```

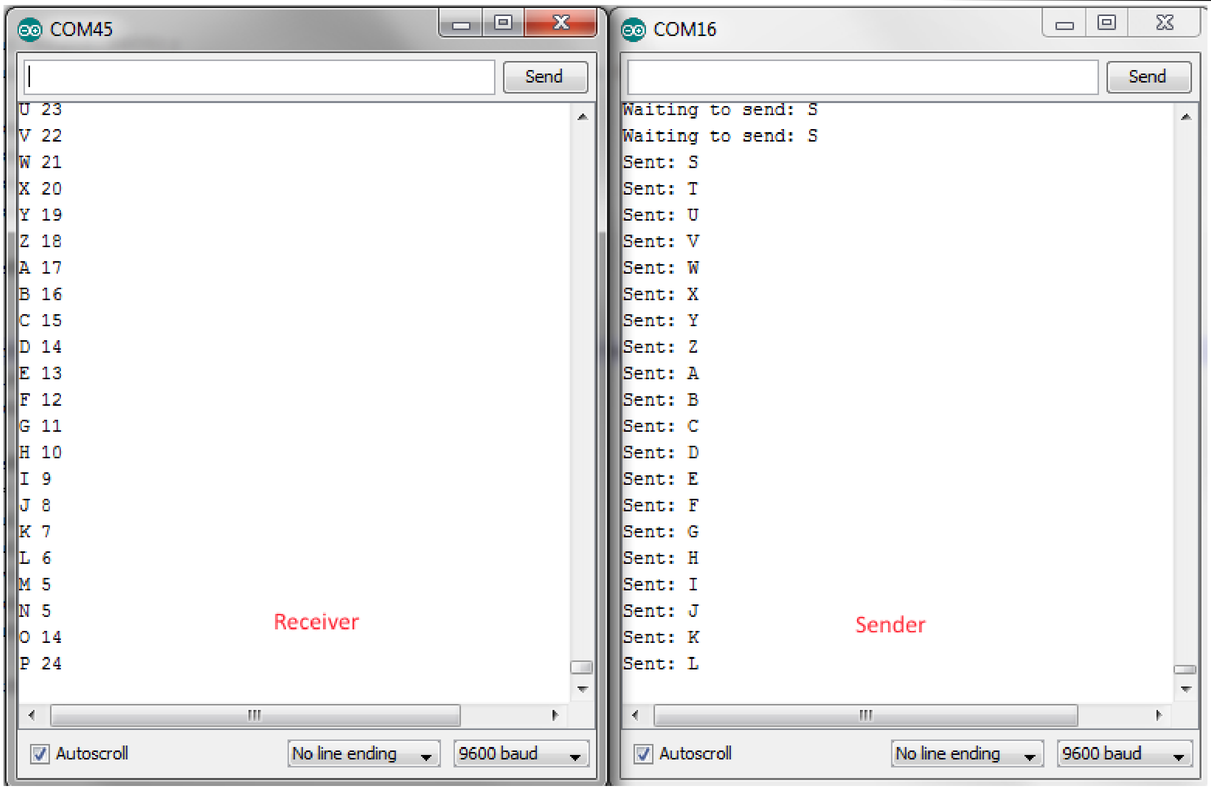
The receiver has the above setup. Hardware flow control doesn't use special characters but it uses the same parameters for FIFO trigger levels. In this example the receiver uses the FCR FIFO trigger levels, *FT_Use_FCR_RX_Trigger,* and the pre-set halt level of 16 characters, *FT_RX_Sixteen_Characters*. The resume level is 8 characters.

In this example, both of the sender and receiver are using similar code sections as the enhanced software flow control examples to send and receive data. However, there are a couple of differences between enhanced software and enhanced hardware flow control. Hardware flow control has much lower overhead than software flow control, as shown in the above figure. The receiver's FIFO fill level stops and starts at the exact level as the specified trigger levels, even though same trigger levels are used in both software and hardware flow control examples. However, the trade-off for hardware flow control is that it requires a 4 wires interface as oppose to only 2 wires for software flow control.

6.3.2 Example 2

This example uses the same hardware setup as the previous example and the required example sketches are: Sender_Enhanced_HW_Flow_Ctl_2.ino and Receiver_Enhanced_HW_Flow_Ctl_2.ino.

This example shows similar functionalities as the previous example and a couple of library functions to make sure the data in the FIFO have actually been sent to the receiver. The following is a snapshot of the serial monitors.



```
U 23
V 22
W 21
X 20
Y 19
Z 18
A 17
B 16
C 15
D 14
E 13
F 12
G 11
H 10
I 9
J 8
K 7
L 6
M 5
N 5
O 14
P 24
```

```
Waiting to send: S
Waiting to send: S
Sent: S
Sent: T
Sent: U
Sent: V
Sent: W
Sent: X
Sent: Y
Sent: Z
Sent: A
Sent: B
Sent: C
Sent: D
Sent: E
Sent: F
Sent: G
Sent: H
Sent: I
Sent: J
Sent: K
Sent: L
```

Figure 6-6 : Hardware Flow Control Example 2 Serial Monitors Output

6.3.2.1 Sender

```
//initialize function, same as calling
VI800A.Init(FT_232U_CS,921600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);
VI800A.SetFlowControl(FT_HW_Flow, FT_Use_No_RX_Trigger);
```

The sender has the above initialization codes. Instead of the default baud rate and data format, this example uses the maximum supported baud rate of 921600.

```
if(manualStart && (VI800A.GetTransmitFIFOFreeSpace())){
        if(VI800A.Write(value[valueIndex])==0)
            Serial.println("Data not in FIFO");
        delay(5);
        while(!VI800A.isTransmitFIFOandShiftRegisterEmpty()){
                Serial.print("Waiting to send: ");
                Serial.println((char)value[valueIndex]);
        }
        Serial.print("Sent: ");
        Serial.println(value[valueIndex]);
        valueIndex=(valueIndex+1)%26;
}
```

The sender in this example didn't enable the FIFO and it checks each character has actually been sent before sending the next character. The overloaded *Write* function returns how many bytes of data are sent to the transmit FIFO and the *isTransmitFIFOandShiftRegisterEmpty* library function guarantees all the data in the FIFO has been pushed out to the receiver by checking the emptiness of the transmit FIFO as well as the transmit shift register.

6.3.2.2 Receiver

```
VI800A.Init(FT_232U_CS,921600,FT_No_Parity, FT_One_Stop_Bit, FT_Eight_Data_Bits);
VI800A.SetFlowControl(FT_HW_Flow, FT_Use_TCR_RX_Trigger, 40, 4);
```

Instead of the FCR receive trigger levels used by the previous examples, the receiver in this example uses the TCR register. The TCR register takes two parameters for the halt and resume level, in this case the halt level is 40 characters and the resume level is 4 characters.

6.4 VI800A-232U to PC

For this scenario, one VI800A-232U module and one RS232-USB cable are needed. The connection should be as follows (again CTS and RTS lines are not needed for software flow control):

Connect (232U)	Connect TO (RS232-USB cable)
GND	GND
TXD	RXD
RXD	TXD
RTS#	CTS
CTS#	RTS

Table 6-1 - VI800A-232U & RS232-USB connections

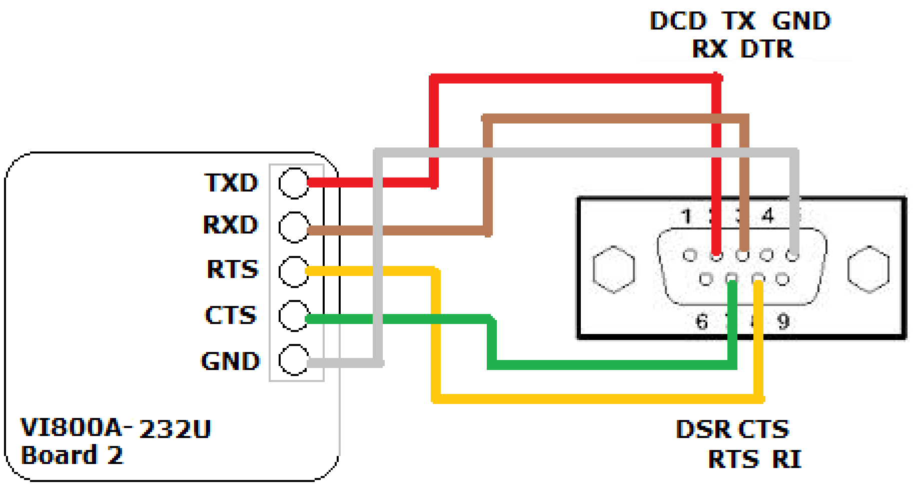
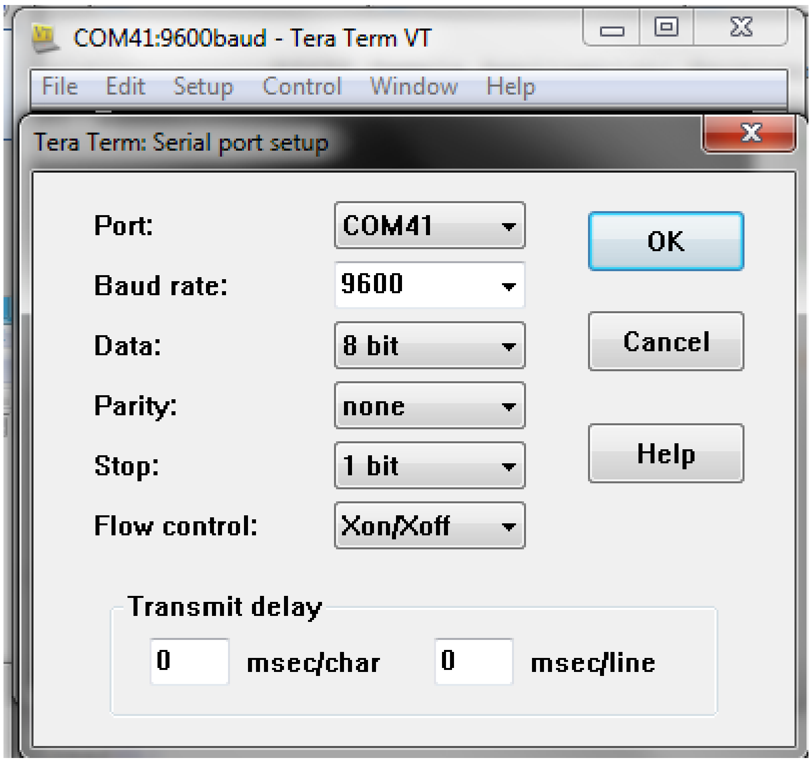


Figure 6-7 : 232U & RS232-USB connected

6.4.1 Software Flow with Tera Term

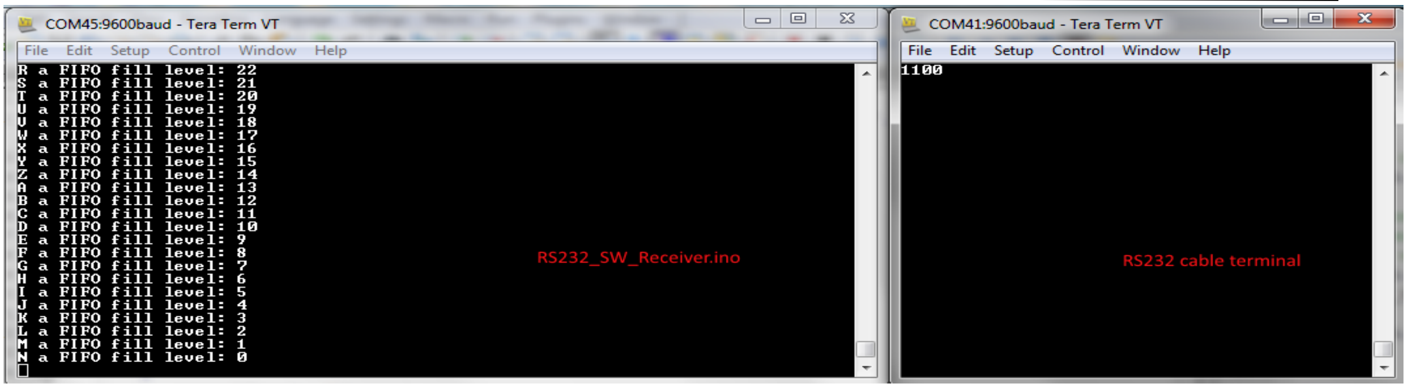
After connecting the VI800A-232U with the RS232 cable as defined in the above table. The Receiver_Enhanced_SW_Flow_Ctl.ino should be uploaded to the VM800P board. Open two instances of Tera Term and connect one to the VM800P board and the other one to the RS232 cable's serial port. The Tera Term terminal for the RS232 cable needs to have the same data configuration as used in the example sketch and the Flow control option should be "Xon/Xoff". The Tera Term Serial port configuration can be accessed by clicking the Setup menu and then the "Serial port…" option.



Port: COM41
Baud rate: 9600
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: Xon/Xoff
Transmit delay
0 msec/char 0 msec/line

Figure 6-8 : Tera Term Serial Port setup

The following figure is the result of continuously sending characters from the RS232 terminal till the trigger level of 16 characters has been reached.


COM45:9600baud - Tera Term VT
File Edit Setup Control Window Help
RS232_SW_Receiver.ino
COM41:9600baud - Tera Term VT
File Edit Setup Control Window Help
RS232 cable terminal

```
R a FIFO fill level: 22
S a FIFO fill level: 21
T a FIFO fill level: 20
U a FIFO fill level: 19
V a FIFO fill level: 18
W a FIFO fill level: 17
X a FIFO fill level: 16
Y a FIFO fill level: 15
Z a FIFO fill level: 14
A a FIFO fill level: 13
B a FIFO fill level: 12
C a FIFO fill level: 11
D a FIFO fill level: 10
E a FIFO fill level: 9
F a FIFO fill level: 8
G a FIFO fill level: 7
H a FIFO fill level: 6
I a FIFO fill level: 5
J a FIFO fill level: 4
K a FIFO fill level: 3
L a FIFO fill level: 2
M a FIFO fill level: 1
N a FIFO fill level: 0
```

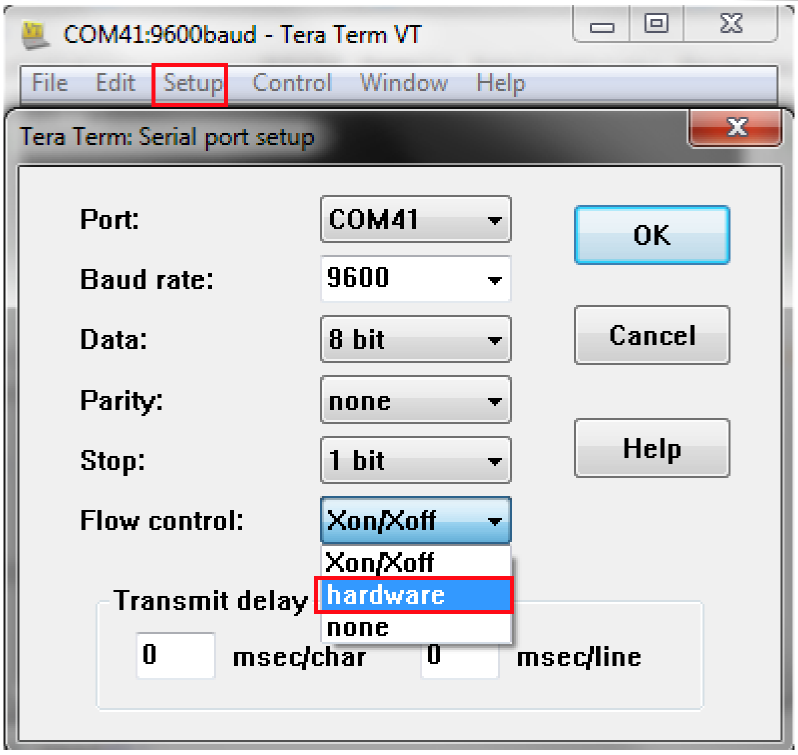
```
1100
```

Figure 6-9 : Software Flow with PC

The Receiver_Enhanced_SW_Flow_Ctl.ino uses ASCII value of number 1 for XOFF and number 0 for XON characters so they're being shown in the RS232 cable terminal when the FIFO level reached the trigger levels. The special characters are discarded by the plug-in modules so connecting the module to a PC can be used to test whether the special characters are actually sent.

6.4.2 Hardware Flow with Tera Term

For this example, the Sender_Enhanced_HW_Flow_Ctl.ino is being used. The example sketch should be uploaded to the VM800P board and the Tera Term terminal for the RS232 cable should use the hardware flow control option.


COM41:9600baud - Tera Term VT

Tera Term: Serial port setup

Port: COM41
Baud rate: 9600
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: Xon/Xoff
Transmit delay
0 msec/char 0 msec/line

Figure 6-10 : Tera Term Serial Port Hardware Flow Option

In the RS232 cable terminal, click File->Send file… and then choose a text file to be sent to the RS232_HW_Receiver.

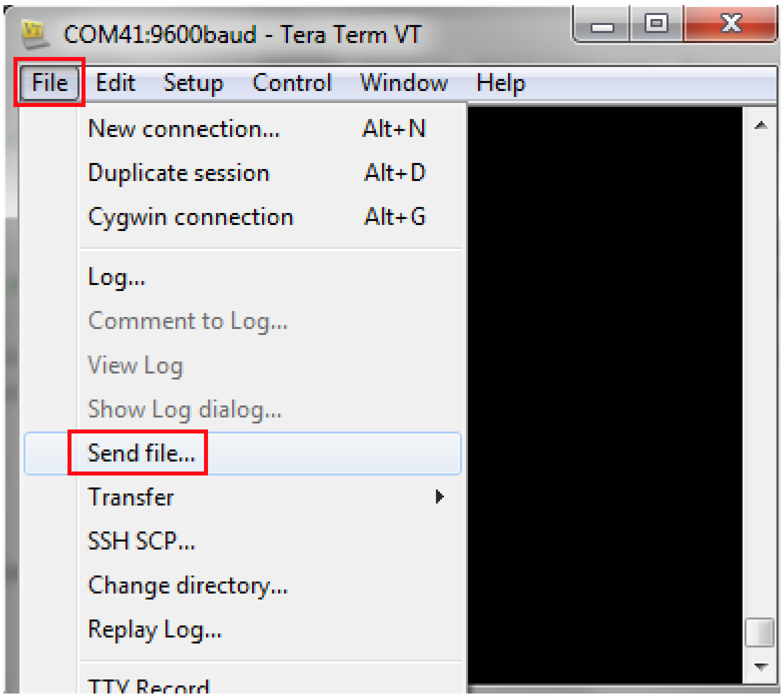
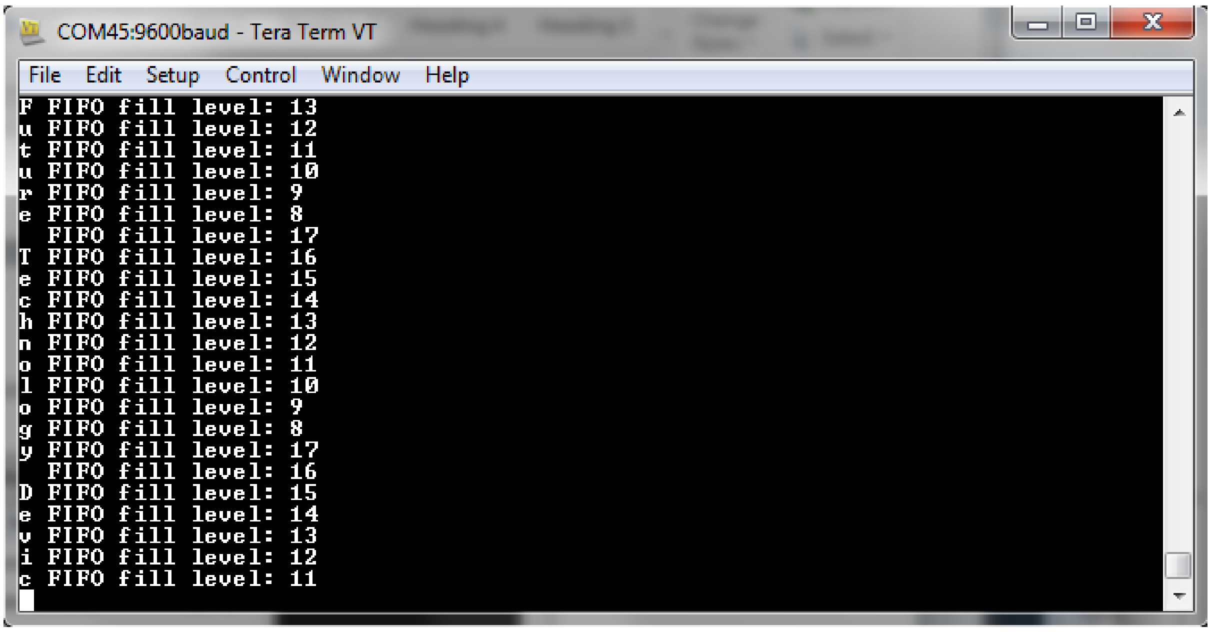

COM41:9600baud - Tera Term VT

Figure 6-11 : Tera Term Send File

The following figure is an instance of the RS232_HW_Receiver terminal during the transmission of a text file. Notice the FIFO trigger levels of halting at 17 characters and resume at 8 characters are still in effect.


COM45:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
F FIFO fill level: 13
u FIFO fill level: 12
t FIFO fill level: 11
u FIFO fill level: 10
r FIFO fill level: 9
e FIFO fill level: 8
  FIFO fill level: 17
T FIFO fill level: 16
e FIFO fill level: 15
c FIFO fill level: 14
h FIFO fill level: 13
n FIFO fill level: 12
o FIFO fill level: 11
l FIFO fill level: 10
o FIFO fill level: 9
g FIFO fill level: 8
y FIFO fill level: 17
  FIFO fill level: 16
D FIFO fill level: 15
e FIFO fill level: 14
v FIFO fill level: 13
i FIFO fill level: 12
c FIFO fill level: 11
```

Figure 6-12 : Tera Term Hardware Flow Data Transferring

7 VI800A-TTLU Examples

All the examples for VI800A-TTLU plug-in modules are almost the same as those in the FT_VI800A_232U folder for the VI800A_232U. The only differences are the initialization macro and the appropriate header file.

7.1 Setup and Run

The VI800A-TTLU plug-in boards should be connected as described in the TTLU Hardware Flow Control table.

The following figure shows the connected VI800A-TTLU plug-in boards.
NOTE: software flow control doesn't require the connection on the CTS# and RTS# pins. The pins are connected so the setup can be used in all example sketches in this section.

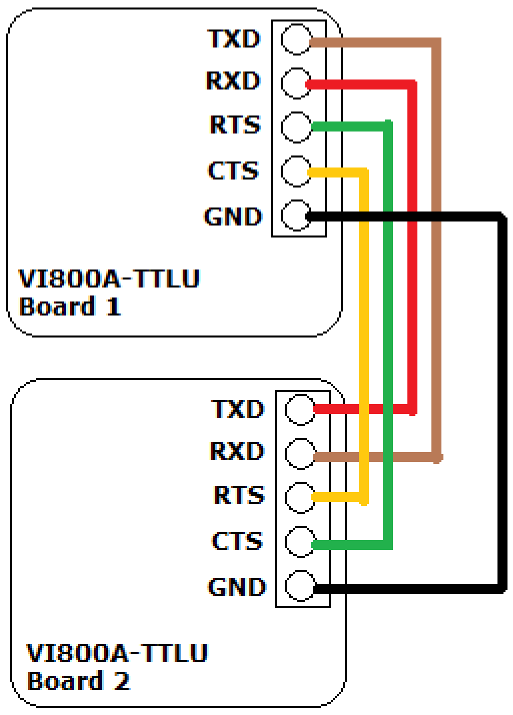


Figure 7-1 : TTLU to TTLU Connections

7.2 VI800A-TTLU to VI800A-TTLU Software Flow Control

Refer to Section 6.2 but the respective example sketches should be from the FT_VI800A_TTLU example folder.

7.3 VI800A-TTLU to VI800A-TTLU Hardware Flow Control

Refer to Section 6.3 but the respective example sketches should be from the FT_VI800A_TTLU example folder.

7.4 VI800A-TTLU to PC

Refer to Section 6.4 but the respective example sketches should be from the FT_VI800A_TTLU example folder.

8 VI800A-N485U Examples

In this example, two VI800A-N485U modules are used. After making the pin connection as described in Hardware Setup and Pin Connections, it should look like the following Figure.

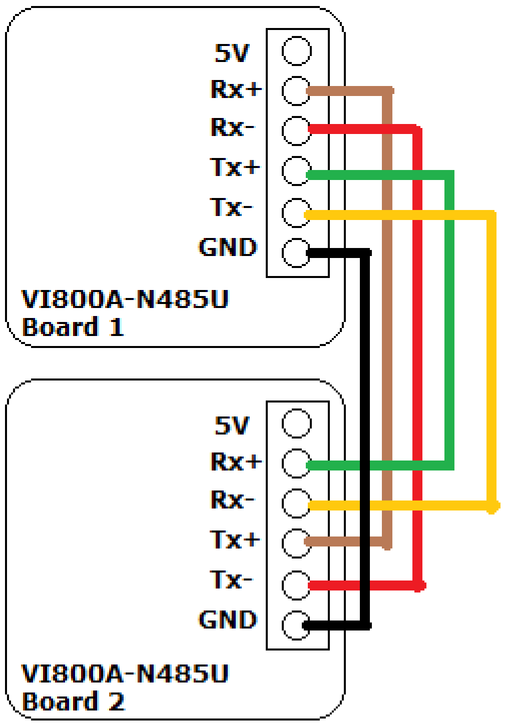


Figure 8-1 : N485U to N485U connection

8.1 Code Explanation

There are two example sketches in the "VI800A-N485U" FTDI Example subfolder: *Receiver_Enhanced_Flow_Ctl* and *Sender_Enhanced_Flow_Ctl*. The receiver example sketch contains both Auto Address Detection and Multidrop enhanced flow functionalities, accessible by a macro switch. The Receiver is also demonstrating the device interrupt functionality so the application doesn't have to constantly poll the receive FIFO for any incoming data. The auto address detection mode is enabled by default.

8.1.1 Sender

```
VI800A.Init(SS, 9600,FT_Forced_Parity_Zero, FT_One_Stop_Bit, FT_Eight_Data_Bits);

VI800A.SetFlowControl(FT_No_Flow);
VI800A.EnableFIFO(FT_Enable);
VI800A.clearFIFO();
```

The above code snippet is the sender's setup section.

NOTE: The parity type for both of the sender and receiver must be "Forced Zero" in order for the receiver device to properly detect the actual address byte.

```
command = Serial.read();
if(command == 'q'){
        VI800A.SetDataFormat(FT_Forced_Parity_One, FT_One_Stop_Bit,
FT_Eight_Data_Bits);
        delay(3);
        VI800A.Write(command);
        Serial.print("Sent Address Byte: ");
        Serial.println((char)command);
        delay(3);
        VI800A.SetDataFormat(FT_Forced_Parity_Zero, FT_One_Stop_Bit,
FT_Eight_Data_Bits);
        }
else if(command == 'w'){
        VI800A.SetDataFormat(FT_Forced_Parity_One, FT_One_Stop_Bit,
FT_Eight_Data_Bits);
        delay(3);
        VI800A.Write(command);
        Serial.print("Sent Address Byte: ");
        Serial.println((char)command);
        delay(3);
        VI800A.SetDataFormat(FT_Forced_Parity_Zero, FT_One_Stop_Bit,
FT_Eight_Data_Bits);
}
else{
        VI800A.Write(command);

        //Uncomment the following two lines to print out the sent character.  However, the
overhead to print the following two message will most likely be longer than 4 character time
frame to hit the stale data interrupt
        //Serial.print("Sent: ");
        //Serial.println((char)command);
}
```

The letter 'q' and 'w' are configured as an address byte by explicitly changing the parity type to "*FT_Forced_Parity_One*" before sending the address byte then reverting back the data format for any subsequent data. The receiver sketch will only push data to the FIFO after receiving the 'q' address. The receiver will disable its receiving functionality for any other address byte that does not address its address ID, so pressing the 'q' key will enable subsequent data to get through and pressing the 'w' key will immediately disable the receiver, if it's not disabled. A time delay is required after setting the address data format and before reverting back to the data format because data written to the FIFO has to go through the shift register. The effect of the commented Serial print lines will be discussed in the receiver section.

8.1.2 Receiver

8.1.1.1 Auto Address Detection

As referenced in Section 4.3.3, the auto address detection mode automatically detects the incoming addresses, enables the device receiver, and disables the receiver after received an address byte that's not addressing itself. The default mode of the receiver is auto address detection and it uses the device interrupt for data retrieval.

```
pinMode(FT_N485U_INT,INPUT);
digitalWrite(FT_N485U_INT,HIGH);
```

To use external device interrupts, the FT_N485U_INT pin must be configured as an input pin.

```
VI800A.Init(FT_N485U_CS, 9600,FT_Forced_Parity_Zero, FT_One_Stop_Bit,
FT_Eight_Data_Bits);

#ifdef AUTO_ADDRESS_DETECTION
        VI800A.SetFlowControl(FT_RS485_Flow,FT_Auto_RS485_Address_Detection
_Mode,AddressByte);
#else
        VI800A.SetFlowControl(FT_RS485_Flow,FT_RS485_Multidrop_Mode,Address
Byte);
#endif

VI800A.SetInterrupts(FT_Xoff_Interrupt|FT_Receive_Holding_Register_Interrupt);

VI800A.EnableFIFO(FT_Enable);
VI800A.clearFIFO();

VI800A.SetFIFOInterruptTriggerLevel(8,8);
attachInterrupt(0,interruptHasHappened,FALLING);
```

The above code snippet is the receiver's initialization code with the parity type is set to "*FT_Forced_Parity_Zero*". The AUTO_ADDRESS_DETECTION macro is defined by default so auto address detection is used and the *AddressByte* variable contains the value of the letter 'q'. The *SetInterrupts* function enables the address byte special character and receive FIFO trigger level interrupts. The *SetFlowControl* function for the enhanced RS485 flow control doesn't enable the FIFO so the FIFO needs to be enabled explicitly and the *SetFIFOInterruptTriggerLevel* function sets the FIFO interrupt trigger level to 8 characters for both receive and transmit FIFO. Finally, the Arduino standard library *attachInterrupt* function is used to initializes the *interruptHasHappened* ISR with *FALLING* trigger mode.

```
void interruptHasHappened(void){
        InterruptOccured = 1;
}
```

The *interruptHasHappened* ISR function sets a volatile global variable which instructs the application to perform the appropriate operation according to the prioritized interrupt from the device.

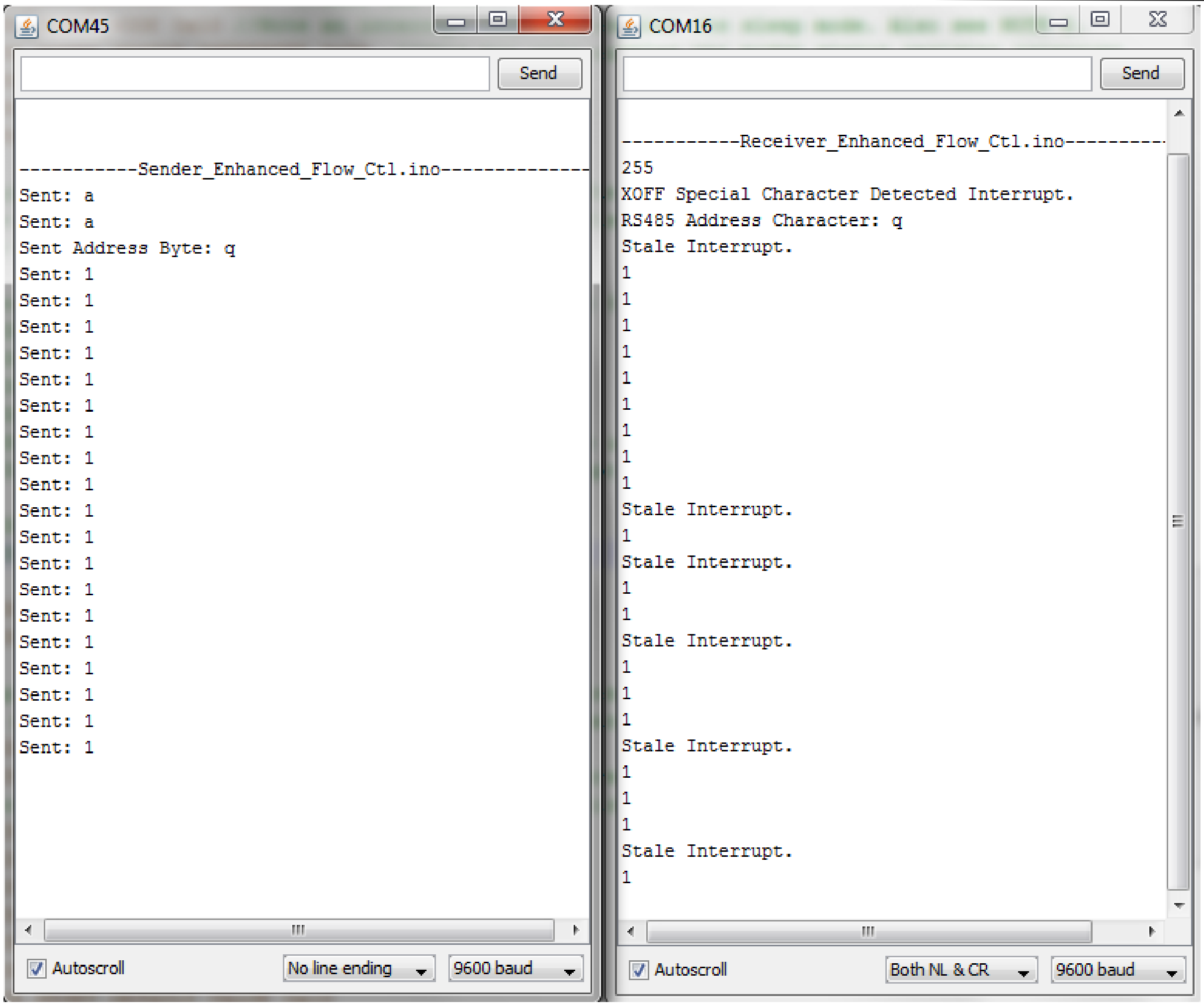
```
        if(status == FT_No_Interrupts)
                return;

        if(status == FT_Receiver_Time_Out_Interrupt){
                        Serial.println("Stale Interrupt.");
                        while(VI800A.GetReceiveFIFOFillLevel()){
                                Serial.println((char)VI800A.Read());
                        }
        }

        else if(status == FT_RHR_Interrupt){
                        Serial.println("Receive FIFO trigger level Interrupt: ");
                        while(VI800A.GetReceiveFIFOFillLevel()){
                                Serial.println((char)VI800A.Read());
                        }
        }

        else if(status == FT_Received_Xoff_Signal){
                        Serial.println("XOFF Special Character Detected Interrupt.");
                        Serial.print("RS485 Address Character: ");
                        Serial.println((char)VI800A.Read());
#ifndef AUTO_ADDRESS_DETECTION
                        if(VI800A.Read() == AddressByte)
                                VI800A.SetReceiver(FT_Enable);
                        else
                                VI800A.SetReceiver(FT_Disable);
#endif
        }
```

The *CheckInterruptStatus* function will be called if an interrupt has occurred. The device interrupts are prioritized and the type of interrupt should be checked in the order listed in the plug-in library's *FT_Interrupts_Status* table.

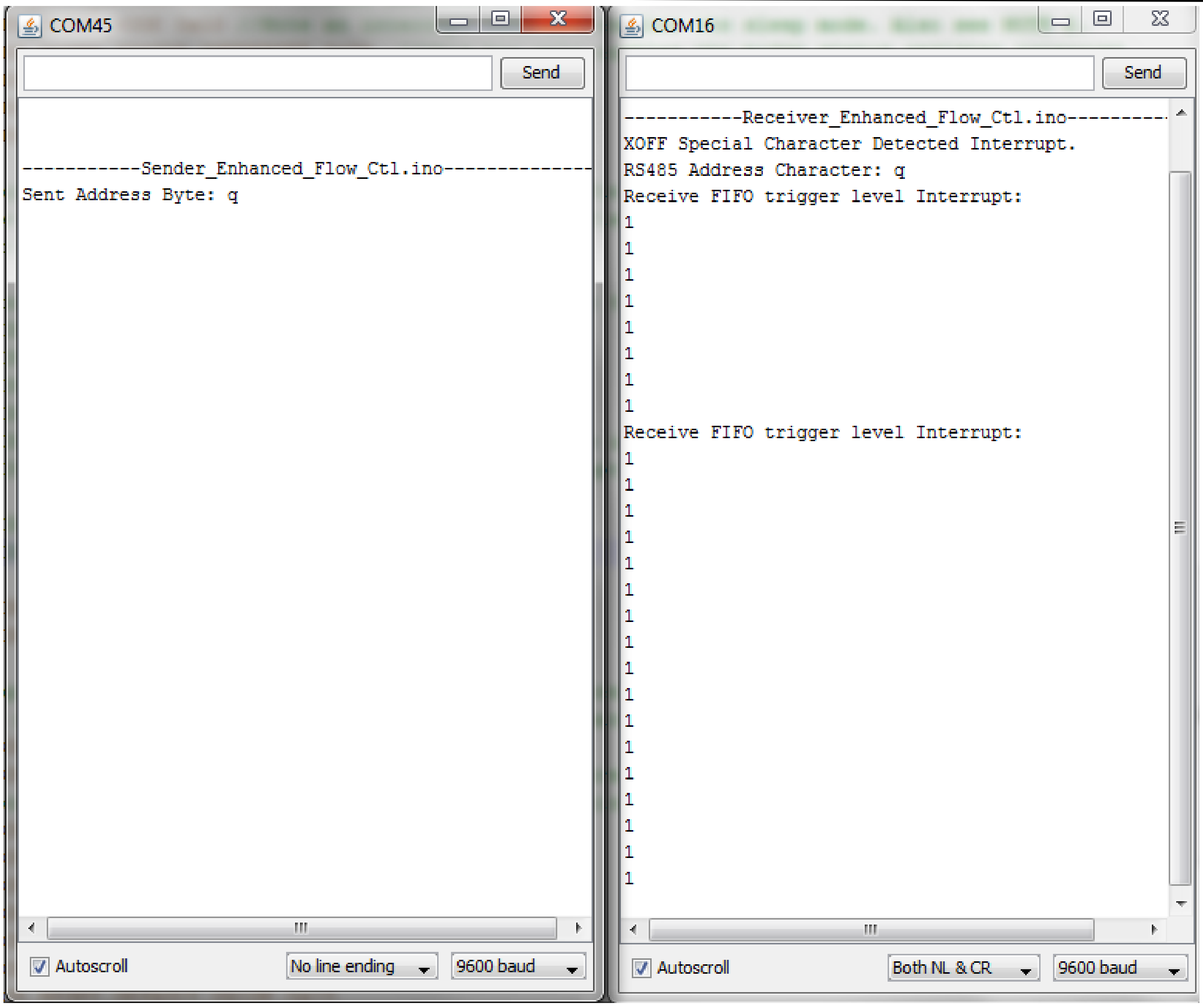


```
-----------Sender_Enhanced_Flow_Ctl.ino-------------
Sent: a
Sent: a
Sent Address Byte: q
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
Sent: 1
```

```
-----------Receiver_Enhanced_Flow_Ctl.ino---------
255
XOFF Special Character Detected Interrupt.
RS485 Address Character: q
Stale Interrupt.
1
1
1
1
1
1
1
1
1
Stale Interrupt.
1
Stale Interrupt.
1
1
Stale Interrupt.
1
1
1
Stale Interrupt.
1
1
1
Stale Interrupt.
1
```

Figure 8-2 : RS485 auto address detection with sender prints

The above figure shows the serial monitors output if the commented Serial print lines are enabled. Notice the receiver didn't print the first two characters from the sender due the correct address byte having not been sent yet so the receiver hasn't enabled the device's receiver. After the letter 'q' has been sent and received by the receiver, the device will generate a special character interrupt and the application prints out the character. The above scenario also shows the situation when a series of 1's are being sent at the same time. The application sets the trigger levels to 8 characters but the receive FIFO interrupt was not be triggered even after the FIFO has queued up 8 characters. There is only a stale data interrupt because the stale data interrupt has a higher priority than the FIFO interrupts and the serial print lines in the sender are enabled which created a time delay of more than 4 character time frame for the stale data interrupt.



```
-----------Sender_Enhanced_Flow_Ctl.ino--------------
Sent Address Byte: q
```

```
-----------Receiver_Enhanced_Flow_Ctl.ino---------
XOFF Special Character Detected Interrupt.
RS485 Address Character: q
Receive FIFO trigger level Interrupt:
1
1
1
1
1
1
1
1
Receive FIFO trigger level Interrupt:
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

Figure 8-3 : RS485 auto address detection without sender prints

The above figure shows the serial monitors output if the commented Serial print lines are **not** enabled. Without the serial prints, the sender can send every byte to the receiver within the 4 character time frame so no stale data interrupt was triggered. After detected a receive FIFO interrupt the example prints out all the data in the FIFO.

The example sketch is not an elegant way to retrieve the data from the FIFO but it's merely to demonstrate the device interrupt usage. Applications are not recommended to retrieve data in the way presented by this example because the complexity of how the sender sends out bulk data determines whether stale data or FIFO level interrupts will be trigged.

8.1.1.2 Multidrop mode

If the AUTO_ADDRESS_DETECTION macro in the RS485_Receiver.ino is commented out then the device will use multidrop mode. The differences between multidrop mode and auto address detection are: The application uses the multidrop mode needs to manually compare the received address character and enable or disable its receiving functionality accordingly. Auto address detection mode passed most of the operations to the devices but the trade-off is that only one address can be used while in Multidrop mode, applications can accept more than 1 address. The example application in multidrop mode enables/disables the receive functionality in the *CheckInterruptStatus* function, after the address byte was detected by the device.

9 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

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.

Appendix A– References

Document References

1. FTDI VM800P board
2. VI800-TTLU / VI800-232U / VI800-N485U

Acronyms and Abbreviations

Terms	Description
Arduino Pro	The open source platform variety based on ATMEL's ATMEGA chipset
EVE	Embedded Video Engine
SPI	Serial Peripheral Interface
UI	User Interface
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Appendix B – List of Tables & Figures

List of Tables

List of Figures

Appendix C– Revision History

Document Title:	AN_330 VI800A_TTLU/232U/N485U Sample Application On Arduino Library
Document Reference No.:	FT_001067
Clearance No.:	FTDI#417
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial release	2014-10-23