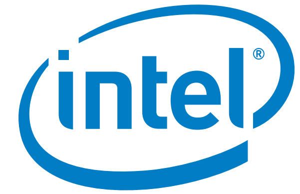
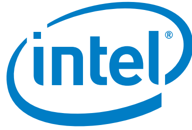


Enabling Multi-COM Port for Microsoft Windows OS 8.1 & 10 / IoT Core

White Paper

October 2016

Document Number: 335148-001



You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

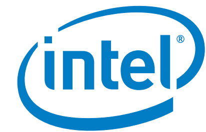
The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2016, Intel Corporation. All rights reserved.



Contents

1.0 Executive Summary 5

2.0 Background 6

2.1 Differences between SerCx2.sys and Serial.sys 6

3.0 SerCx2 Framework Overview 8

4.0 Implement Multi-COM Port 10

4.1 Create COM Port Peripheral Device at BIOS 10
4.2 Build Peripheral Driver for COM Port 12
4.2.1 File Manifest 12
4.2.2 Driver Interfaces 12
4.2.3 System Requirements for Compiling Driver 13
4.2.4 Driver Installation 14

5.0 Serial Communication with COM Port 15

5.1 RS-232 Serial COM Protocol 15
5.2 USB Serial Port - FT232R 16
5.3 User Application for Serial Communication 17
5.3.1 ExtraPutty 18

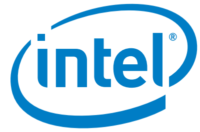
6.0 References 19

Figures

Figure 1. SerCx2 Serial Framework Diagram 8
Figure 2. Device Manager View of New COM Port 14
Figure 3. HSUART Level Shift Circuit from 1.8V to 3.3V 16
Figure 4. RS-232 Voltage Level Convert Circuit 16
Figure 5. FT232 Module Picture 17
Figure 6. Putty Configuration 17

Tables

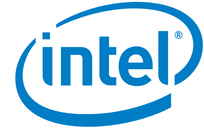
Table 1. Terminology 7
Table 2. File Manifest 12
Table 3. Implementation for I/O Request Callback Function 13
Table 4. Support IOCTL Command List 13
Table 5. RS-232 Logic and Voltage Levels 15
Table 6. Intel HSUART Interface Signals 15



Revision History

Date	Revision	Description
October 2016	1.0	Initial release.

§

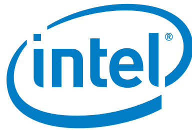


1.0 Executive Summary

COM port is a popular communication interface with customers and end-users. Although an Intel LPSS IO processor component includes 2-4 HSUART devices, HSUART has not been enumerated as a COM port device since Windows 8.

This paper presents a complete solution for enumerating a HSUART device as a typical RS-232 COM port device on Windows 8.1, Windows 10 and Windows 10 IoT Core. It addresses software and hardware changes, providing key information to end-users to improve their products.

§



2.0 Background

A Serial (COM) port is a hardware communication interface on a serial controller, which is a 16550 UART or compatible device. Through COM port, a serial controller communicates with a serially connected peripheral device. All versions of Windows provide driver support for serial controller devices. Windows includes the Serial.sys and Serenum.sys, and SerCx and SerCx2.

Starting with Windows 8.1, (included Windows 10), Versions 2 of serial framework extension (SerCx2) are used to assist an extension-based serial controller driver by handling many of the processing tasks that are common to serial controllers. However, this framework makes a serial controller driver unable to enumerate UART or HSUART as a typical COM port device, even though these ports conform to the RS-232 standard.

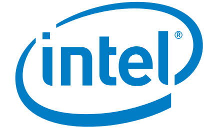
2.1 Differences between SerCx2.sys and Serial.sys

Serial.sys is designed to control named COM ports that are driven by 16550A or similar UARTs. External peripheral devices can be dynamically plugged into and removed from these ports. In principle, Serial.sys could use the system DMA controller, but, in a PC, this controller is an 8237 device with limited capabilities. A multiport board, which implements several serial ports, might contain a master DMA controller, but the hardware vendor for the board must write a custom serial driver to exploit these DMA capabilities.

Serial interfaces are now widely used to provide low-pin-count communication between integrated circuits on a printed circuit board. Data transmission rates through these interfaces can be relatively high due to the low impedance and short path lengths involved.

SerCx2.sys is designed to work with dedicated serial ports that are permanently connected to peripheral devices and that support high data rates. SerCx2.sys is flexible in its support for DMA. Complex data transfers that use system DMA are fully supported. In addition, SerCx2.sys provides an optional custom transfer mode to support a serial controller that has built-in bus-master DMA capability.

A COM port controlled by Serial.sys is assigned with a device name. A user-mode application can open this port by name, and then send I/O requests directly to the port. In contrast, a serial port controlled by SerCx2.sys and a serial controller driver is unnamed. Typically, only the slave driver can send I/O requests directly to the port. An application that needs to configure the port or to transfer data through the port sends I/O requests to the slave driver. Then, acting as intermediary, this driver sends the corresponding I/O requests to the port.



Another difference is that Serial.sys implements software flow control, but SerCx2.sys does not. Both Serial.sys and SerCx2.sys support hardware flow control using the RTS and CTS signals.

A final difference is that Serial.sys can work in conjunction with Serenum.sys, but SerCx2.sys cannot. Serenum.sys is a filter driver that enumerates devices that are connected to serial ports.

Table 1. Terminology

Term	Description
(HS)UART	(High Speed) Universal Asynchronous Receiver/Transmitter
RS-232	A standard for serial communication transmission of data
IOCTL	I/O Control Request
COM	Serial Communication Port
TTL	Transistor-transistor Logic
ACPI	Advanced Configuration and Power Interface
ASL	ACPI Source Language
SerCx2	Version 2 of Serial Framework Extension
RTS/CTS	Request To Send / Clear To Send
DTR/DSR	Data Terminal Ready / Data Set Ready
TXD/RXD	Transmit Data / Receive Data

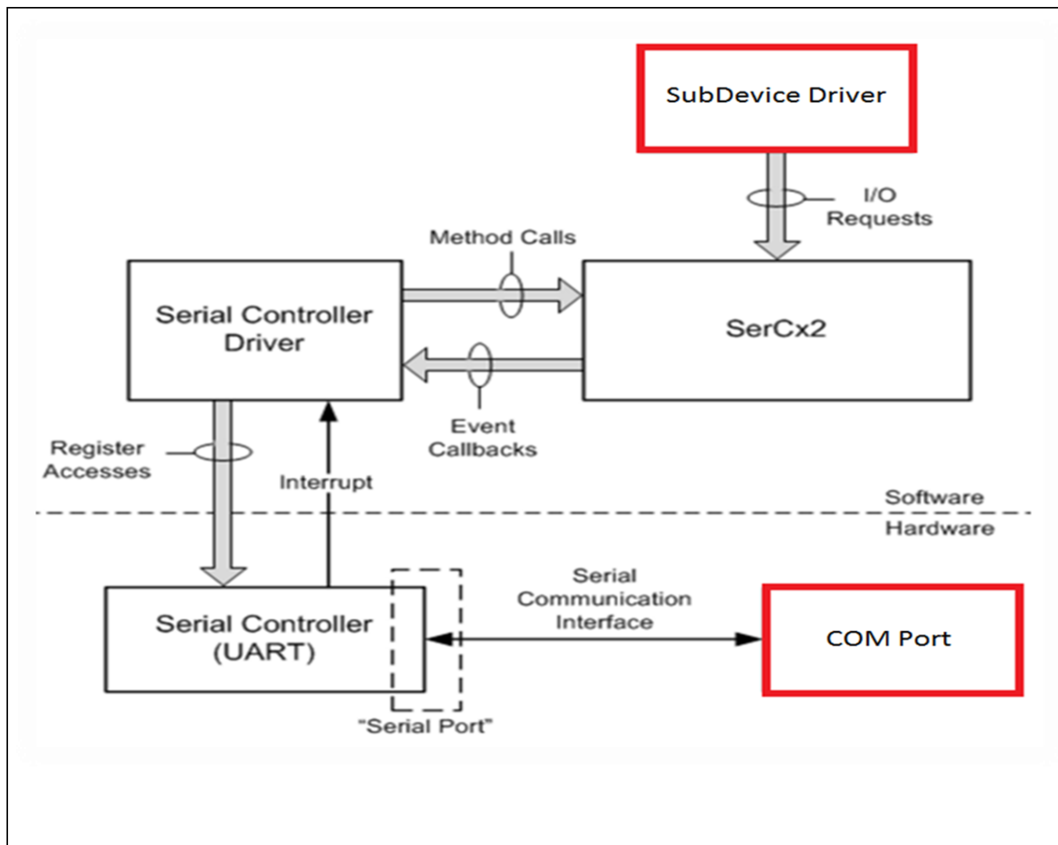
§

3.0 SerCx2 Framework Overview

SerCx2 works together with a serial controller driver to enable communication between a peripheral driver and a serially connected peripheral device.

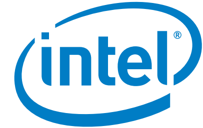
The following diagram shows the working framework of SerCx2.

Figure 1. SerCx2 Serial Framework Diagram



This peripheral driver runs in either kernel mode or user mode, and sends I/O requests to the serial port to which the peripheral device is connected. The COM port device is a peripheral device, and the related peripheral driver is sub-device driver of HSUART. Once the sub-device driver is implemented, additional COM port devices based on HS-UART can be used. These COM port devices are connected by wire to the HS-UART port of SOC.

SerCx2 and the HSUART controller driver both run in kernel mode, and communicate with each other through the SerCx2 device-driver interface (DDI). Typically, only drivers send I/O requests directly to the HSUART controller. When a user-mode application, such as Putty or Extra Putty, needs to communicate with a serially connected peripheral device, the peripheral driver for the device acts as intermediary between the

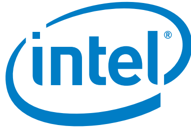


application and the device. If the application needs to transfer data to or from the peripheral device, the application sends a write request or read request to the peripheral driver, and the peripheral driver responds by sending a corresponding write and read request to the HSUART controller. In addition, the peripheral driver can send device I/O control requests (IOCTLs) to configure the serial port.

The peripheral device is our sub-device for HSUART; the peripheral driver is sub-device driver for this COM port device.

The next chapter introduces sub-device and sub-device driver for HSUART, which band together to implement multi-COM port on Windows OS 8.1 and 10 IoT Core.

§



4.0 Implement Multi-COM Port

This section presents the complete solution for enabling multi-COM port with HSUART. As described in the previous chapter, two important parts must be implemented. First, create a COM port peripheral device. Even though this device is not a real physical device, the ACPI description for this COM port device must be included in the BIOS ACPI table. Second, on the software side, the related driver for this serial peripheral device should support the basic interface of SerCx2.

4.1 Create COM Port Peripheral Device at BIOS

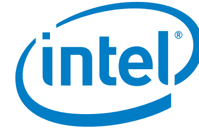
The peripheral device for COM port is virtual hardware. The BIOS must assign a unique ACPI Hardware ID for this special peripheral device and store it in its ACPI table.

The default Hardware IDs (INT3511 and INT3512) used here are Intel HS-UART COM port peripheral device IDs.

The following sample ASL code is used to define the description of the ACPI table for "COM port for HSUART1":

```
Device (VUT0)
{
     Name (_HID, "INT3511")  // _HID: Hardware ID
     Method (_STA, 0, NotSerialized)  // _STA: Status
                {
                    If ((BDID == CHRB))
                    {
                        If (_OSI ("Android"))
                        {
                            Return (Zero)
                        }
                        Else
                        {
                            Return (0x0F)
                        }
                    }
                    Else
                    {
                        Return (Zero)
                    }
                }

      Method (_CRS, 0, NotSerialized)  // _CRS: Current
      Resource Settings
                {
                    Name (BBUF, ResourceTemplate ()
                    {
                        UartSerialBus (0x0001C200,
```

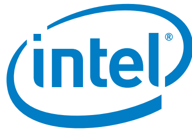


```
                              DataBitsEight, StopBitsOne,
                              0xFC, LittleEndian, ParityTypeNone,
                              FlowControlNone,
                              0x0020, 0x0020, "\\_SB.PCI0.URT1",
                              0x00, ResourceConsumer, ,
                              )
                          })
                Return (BBUF) /* \_SB_.PCI0.URT1.VUT0._CRS.BBUF */
                  }
}
```

The following sample ASL code is used to define the description of the ACPI table for "COM port for HSUART2":

```
Device (VUT1)
{
     Name (_HID, "INT3512")  // _HID: Hardware ID
     Method (_STA, 0, NotSerialized)  // _STA: Status
                {
                    If ((BDID == CHRB))
                    {
                        If (_OSI ("Android"))
                        {
                            Return (Zero)
                        }
                        Else
                        {
                            Return (0x0F)
                        }
                    }
                    Else
                    {
                        Return (Zero)
                    }
                }

      Method (_CRS, 0, NotSerialized)  // _CRS: Current
      Resource Settings
                {
                    Name (BBUF, ResourceTemplate ()
                    {
                        UartSerialBus (0x0001C200,
                        DataBitsEight, StopBitsOne,
                        0xFC, LittleEndian, ParityTypeNone,
                        FlowControlHardware,
                        0x0020, 0x0020, "\\_SB.PCI0.URT2",
                        0x00, ResourceConsumer, ,
                        )
                    })
```




```
                    Return (BBUF) /* \_SB_.PCI0.URT2.VUT1._CRS.BBUF */
                }
}
```

Note: When HSUART controller is in PCI mode, the parameter 10 of UartSerialBus should be "_SB.PCI0.URT2". For ACPI mode, it should be "_SB.URT2".

Note: For more information about UartSerialBus functionality, refer to Section 6.0 References.

4.2 Build Peripheral Driver for COM Port

4.2.1 File Manifest

The source files listed in Table 2 are located in the "Sub Device Driver Sample Code\UartSample" folder. These files are used to build the UART sub-device driver.

Table 2. File Manifest

File	Description
Device.c & Device.h	WDFDEVICE related functionality and callbacks.
Driver.c & Driver.h	DriverEntry and WDFDRIVER related functionality and callbacks.
Public.h	Header file to be shared with applications.
Queue.c & Queue.h	WDFQUEUE related functionality and callbacks.
Trace.h	Definitions for WPP tracing.
UartSample.inf	Sample INF file that contains installation information for this sub-device driver.

4.2.2 Driver Interfaces

This driver uses the Microsoft Kernel Mode Driver Framework. It provides functions for the user application and help application to send I/O requests to the SerCx2 framework. The requests are handled by the HSUART controller.

For example, ReadFile calls the EvtIoRead request; WriteFile calls EvtIoWrite; DeviceIoControl calls EvtIoDeviceControl, and so on.

Table 3 lists callback functions which are implemented at this HSUART Sub-device driver.

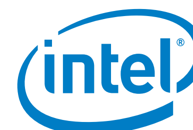


Table 3. Implementation for I/O Request Callback Function

I/O Request Callback API	Callback Implementation
PFN_WDF_IO_QUEUE_IO_WRITE EvtIoWrite	UartSampleEvtIoWrite
PFN_WDF_IO_QUEUE_IO_READ EvtIoRead	UartSampleEvtIoRead
PFN_WDF_IO_QUEUE_IO_DEVICE_CONTROL EvtIoDeviceControl	UartSampleEvtIoDeviceControl
PFN_WDF_IO_QUEUE_IO_STOP EvtIoStop	UartSampleEvtIoStop
PFN_WDF_IO_QUEUE_IO_CANCELED_ON_QUEUE EvtIoCanceledOnQueue	UartSampleEvtIoCanceledOnQueue

Note: For more information about the serial I/O request interface, refer to: https://msdn.microsoft.com/en-us/library/ff552359(VS.85).aspx

During data transfer by the serial communication interface, some IOCTL command are very important to configure HSUART working parameters. This serial communication interface is implemented at the UartSampleEvtIoDeviceControl callback function.

In the current Sub-device driver, we only implement part of SerCx2 serial interface. Table 4 list the IOCTLs.

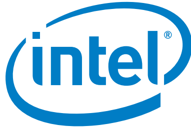
Table 4. Support IOCTL Command List

IOCTL_UARTTESTTOOL_OPEN	IOCTL_UARTTESTTOOL_CLOSE
IOCTL_SERIAL_SET_BAUD_RATE	IOCTL_SERIAL_GET_BAUD_RATE
IOCTL_SERIAL_SET_MODEM_CONTROL	IOCTL_SERIAL_GET_MODEM_CONTROL
IOCTL_SERIAL_SET_LINE_CONTROL	IOCTL_SERIAL_GET_LINE_CONTROL
IOCTL_SERIAL_SET_CHARS	IOCTL_SERIAL_GET_CHARS
IOCTL_SERIAL_SET_HANDFLOW	IOCTL_SERIAL_GET_HANDFLOW
IOCTL_SERIAL_GET_MODEMSTATUS	IOCTL_SERIAL_GET_DTRRTS
IOCTL_SERIAL_GET_MODEMSTATUS	IOCTL_SERIAL_GET_COMMSTATUS
IOCTL_SERIAL_GET_PROPERTIES	IOCTL_SERIAL_SET_FIFO_CONTROL
IOCTL_SERIAL_GET_STATS	IOCTL_SERIAL_CLEAR_STATS
IOCTL_SERIAL_PURGE	IOCTL_SERIAL_SET_TIMEOUTS

Note: For more information about the serial I/O request interface, refer to: https://msdn.microsoft.com/en-us/library/windows/hardware/dn265347(v=vs.85).aspx

4.2.3 System Requirements for Compiling Driver

- Microsoft Visual Studio* 2012 or higher



- Microsoft Windows* Driver Kit (WDK) Version 8.0 or higher

4.2.4 Driver Installation

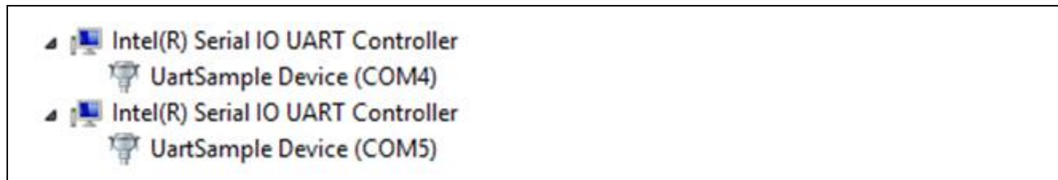
For probing the COM Port peripheral device in the driver, the Hardware IDs shown in the driver's .inf file (INT3511 and INT3512) should be identical with the IDs defined in the BIOS device description of the ACPI table.

```
%UartSample.DeviceDesc%=UartSample_Device, ACPI\INT3511; TODO:
edit hw-id
%UartSample.DeviceDesc%=UartSample_Device, ACPI\INT3512; TODO:
edit hw-id
```

Because the sub-device driver depends on the HSUART controller driver before installing the sub-device driver, ensure that the HSUART host controller driver has already been installed successfully.

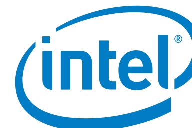
Figure 2 illustrates how the new COM port devices are available and displayed under the HSUART controller in Device Manager, after installing the HSUART sub-device driver successfully.

Figure 2. Device Manager View of New COM Port



Note: After clicking "View" and selecting "Devices by Connection", the Device Manager will display above view.

§



5.0 Serial Communication with COM Port

When transferring data via the serial communication port between two ends, the important rule is that the identical signal protocol will be used at two serial communication ends. The most common serial communication protocol is RS-232. A more popular and widely used chip, the FT232R, is the converter between the USB and UART interface. This chip can also implement serial communication.

5.1 RS-232 Serial COM Protocol

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are either in the range of +3 to +15 volts or the range -3 to -15 volts with respect to the "Common Ground" GND pin. Consequently, the range between -3 to +3 volts is not a valid RS-232 level. For data transmission lines (TXD, RXD), logic one is defined as a negative voltage. The signal condition is called "mark". Logic zero is positive and the signal condition is termed "space". Control signals have the opposite polarity: the asserted or active state is positive voltage and de-asserted or inactive state is negative voltage.

Table 5. RS-232 Logic and Voltage Levels

Data Circuits	Control Signals	Voltage
0 (space)	Asserted/Active	+3 to +15 V
1 (mark)	Deasserted/Inactive	-15 to -3 V

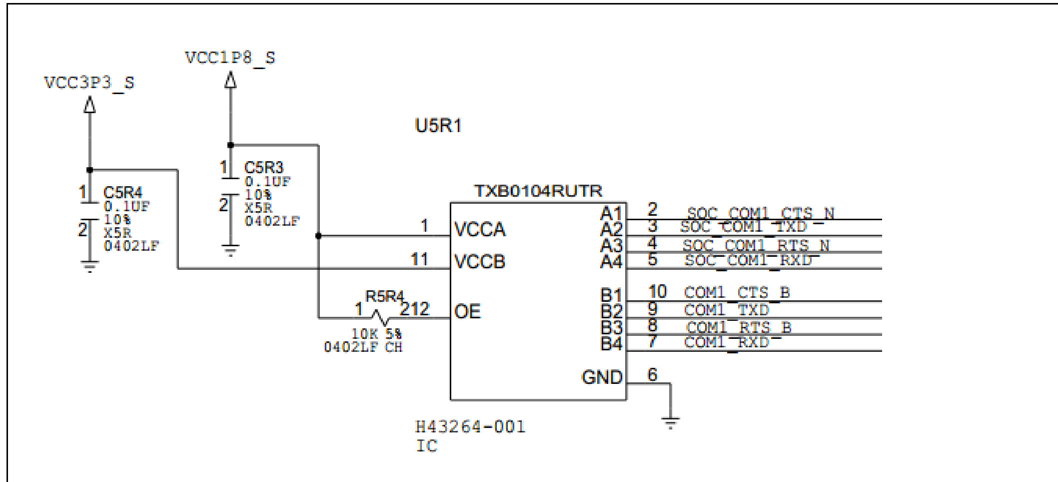
However, the HS-UART signal from the Intel SOC chip belongs to the TTL signal. The HSUART interface signal output voltage should be 1.8 V.

Table 6. Intel HSUART Interface Signals

Signal Name	Direction / Type	Voltage	Description
Hsuart_rxd	I	1.8 v	High-Speed UART receive data input
Hsuart_txd	O	1.8 v	High-Speed UART transmit data output
Hsuart_rts	O	1.8 v	High-Speed UART request to send

Consequently, when we need to connect to the RS-232 serial peripheral device via our new COM port base on HSUART, there are two important circuits that should be wired to the Intel HSUART output pin. The first is Voltage Level Shift Circuit, whose responsibility is to convert output voltage from 1.8 Volts to 3.3 Volts. See Figure 3 for an example of this circuit.

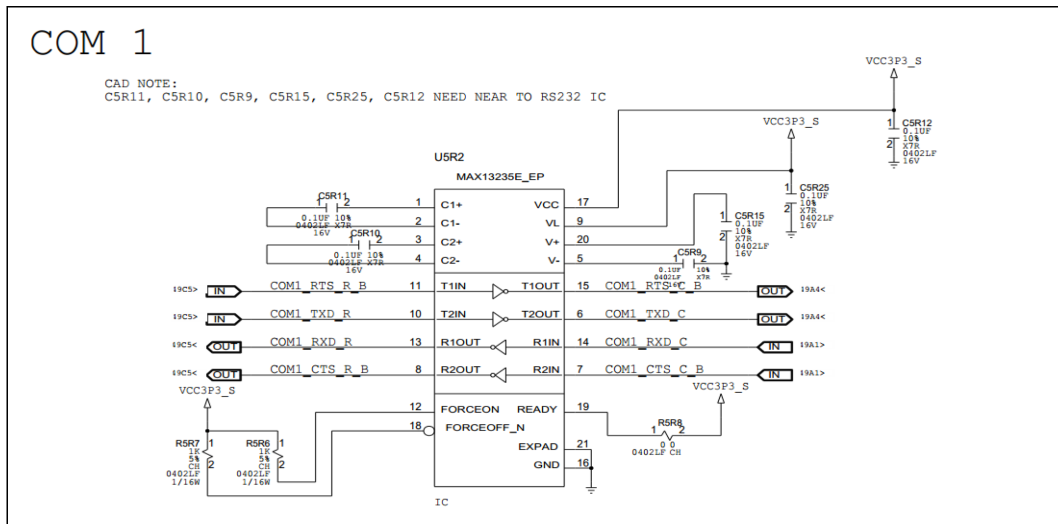
Figure 3. HSUART Level Shift Circuit from 1.8V to 3.3V



Second circuit should convert TTL voltage to the RS-232 voltage level to meet the RS-232 standard. See Figure 4 for an example of this circuit.

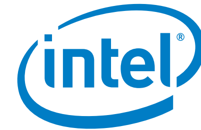
However, when serial communication is between two identical TTL voltage devices, the RS-232 converter circuit can be left out. When we connect by loop two SOC's HS-UART interfaces without a converter circuit, the serial communication can still work well.

Figure 4. RS-232 Voltage Level Convert Circuit



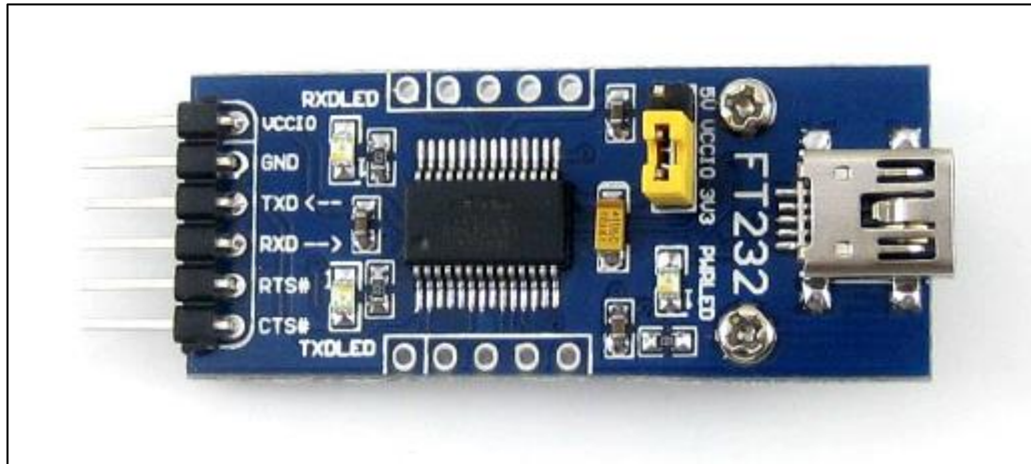
5.2 USB Serial Port - FT232R

The FT232R is a USB to serial UART interface, as shown Figure 5. The left side interfaces, such VCC/GND/TXD/RXD/RTS/CTS, could be 5V/3.3V/2.8V/1.8V CMOS drive output and TTL input. Consequently, Intel SOC chip HSUART interfaces are able to



connect to these pins. The USB port is connected to the other host device. This connection method is the common circuit to design for a USB debug port on a platform layout. The FT232 driver should be installed successfully to the host machine. After that, the debug message can be outputted and displayed via the debug COM port.

Figure 5. FT232 Module Picture



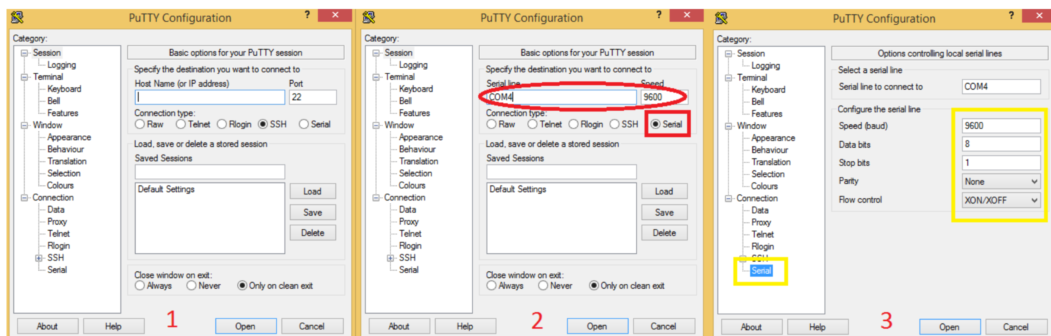
5.3 User Application for Serial Communication

In this section, the user applications' usage examples are introduced, such as Putty and Extra-Putty.

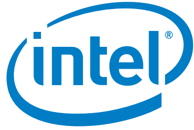
For more information about Putty, see http://www.putty.org/.

The following represents a basic Putty configuration that could be used to operate a user application.

Figure 6. Putty Configuration



As shown in the figure above, when opening a COM port for serial communication, parameters such as transfer speed, data bits, stop bits, parity and flow control must be set.

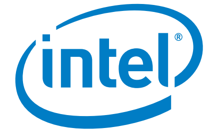


5.3.1 ExtraPutty

For complete information about ExtraPutty, go to http://www.extraputty.com/.

Extra Putty has the same configuration as Putty. Compared to Putty, ExtraPutty is able to transfer a file by following the XMODEM/YMODEM/ZMODEM rule.

§



6.0 References

https://msdn.microsoft.com/en-us/library/windows/hardware/ff546939(v=vs.85).aspx

http://www.acpi.info/spec.htm

https://msdn.microsoft.com/en-us/library/windows/hardware/dn265347(v=vs.85).aspx