# Communication - Using the USBUART User Module

## AN2401

**Author**: Svyatoslav Paliy, Vadym Grygorenko
**Associated Project**: Yes
**Associated Part Family**: CY8C24x94
GET FREE SAMPLES HERE
**Software Version**: PSoC Designer™ 4.3, SP1/SP2 Recommended
**Associated Application Notes**: None
PSoC Application Notes Index

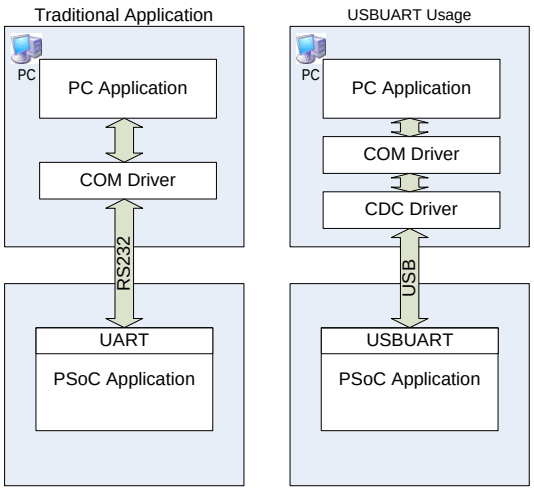
## Application Note Abstract

This Application Note describes how to use the USBUART User Module in PSoC Designer to quickly migrate from UART to USB.

## Introduction

Many embedded applications use the RS232 interface to communicate with external systems, such as PCs, especially for debugging. The RS232 COM port is rapidly disappearing from most new computers, replaced by USB. The simplest way to migrate to USB is to emulate RS232 over the USB bus. An advantage of this method is that PC applications see the USB connection as an RS232 COM port connection and thus it is very simple to use for debugging. This method uses a standard Windows® driver that is included with all versions of Microsoft® Windows from 98SE on.

Figure 1. USBUART vs. Traditional UART

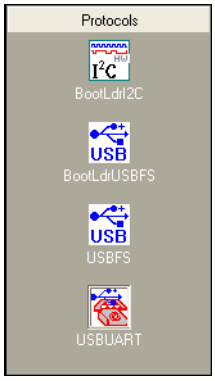



A Windows application detects a physically connected USB device as a COM port connected device and communicates with it using the standard WinAPI CreateFile, ReadFile, and WriteFile functions. No modifications of the existing PC software are required to support a USBUART-featured device.

## A Simple Echo Project

In PSoC Designer Device Editor, select the USBUART User Module from the Protocols tab. Switch to the Interconnect View, place the user module, and rename it to USBUART.

Figure 2. USBUART User Module Location



Once the user module is placed you can select the user module parameters.

- **Vendor ID** and **Product ID**. The VendorID (VID) and ProductID (PID) are used by the operating system to identify the USB device and assign it a driver. The VID is a 16-bit number unique to each manufacturer and assigned by the USB Implementers Forum. The VID can be purchased directly from USB Implementers Forum at www.usb.org/developers/vendor. Each VID comes with 65536 different PIDs. The PID is also a 16-bit number and you assign a unique product ID to each USB product you create.
- **VendorString**. A string that describes the product vendor.
- **ProductString**. A string that describes the product.
- **SerialNumberString**. Each USB device can have a unique serial number. The serial number is used to differentiate between two devices with the same VID and PID connected to the PC at the same time.
- **DevicePower**. The USB device can be Bus Powered, meaning that it uses the USB bus as its power source or Self Powered meaning that it uses an external power source and uses the USB bus only for data exchange.
- **MaxPower**. This is the maximum amount of power that your device will require from the USB port. If the DevicePower is set to Self Powered, this parameter is ignored. If you set this value higher than 100 mA, it cannot be connected to a low-power USB hub. The maximum power that can be supplied by the USB is 500 mA. If the device needs draws more than 500 mA it must use an external power source.

Figure 3. Parameter Values Used In This Demo. Project

| User Module Parameters | Value |
|---|---|
| VendorID | 04B4 |
| ProductID | F235 |
| VendorString | Cypress Corp |
| ProductString | Demo USB-UART |
| SerialNumberString | 0001 |
| DevicePower | Self Powered |
| MaxPower | 100 |

Add the following code to *main.c*:

Code 1. Contents of *main.c*

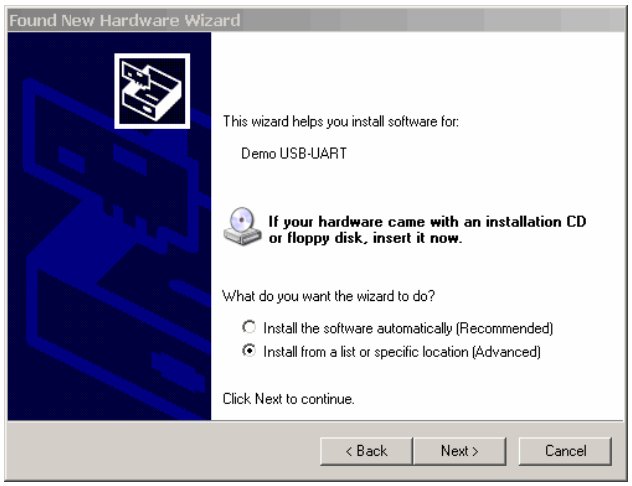
```
BYTE Len;
BYTE pData[32];

void main() {
     //Enable Global Interrupts
   M8C_EnableGInt;
     //Start USBUART 5V operation
   USBUART_Start(USBUART_5V_OPERATION);
     //Wait for Device to initialize
   while(!USBUART_Init());

       // Main loop
 while(1) {
       //Get count of ready data
   Len = USBUART_bGetRxCount();
   if (Len) {
        //Read all data rom RX
     USBUART_ReadAll(pData);
        //Wait for TX ready
     while(!USBUART_bTxIsReady());
        //Send received data back
     USBUART_Write(pData, Len);
}} }
```

Compile your project, download it to the PSoC® device and connect the board to the USB port on your PC. When the device is first connected to the PC the New Hardware Wizard will start.

Figure 4. New Hardware Wizard – Step 1

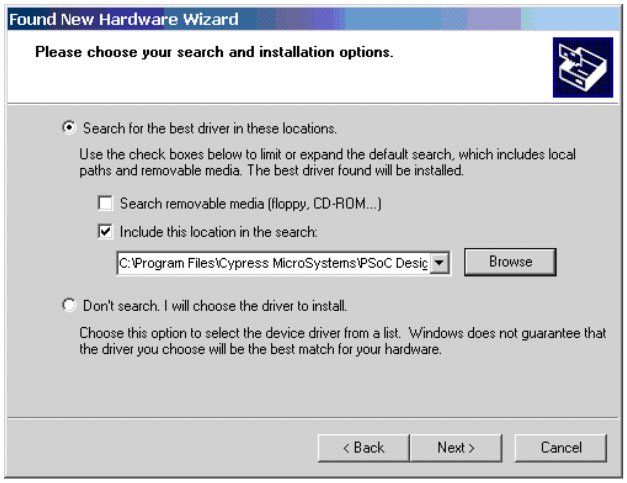

Select **No, not this time** and click the **Next** button.

Figure 5. New Hardware Wizard – Step 2



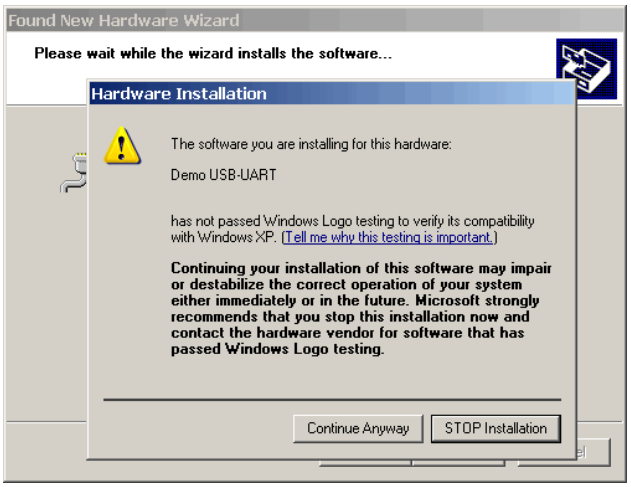

Select **Install from a list or specific location (Advanced)** and click the **Next** button.

Figure 6. New Hardware Wizard – Step 3



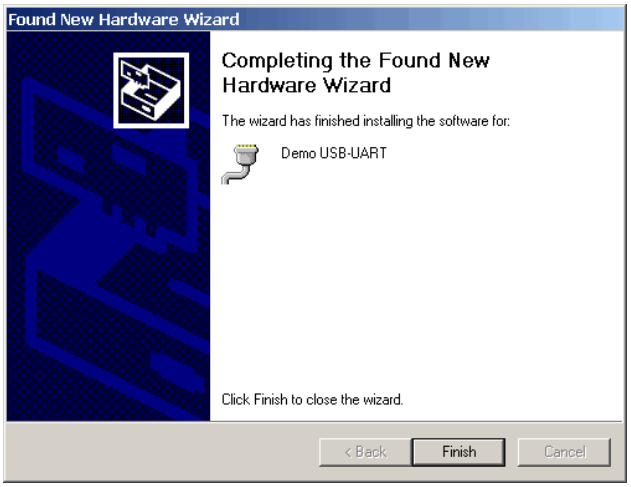

Select **Search for the best driver in these locations**, and **Include this location in the search** and set the path to the `\lib` subfolder of the PSoC USBUART demonstration project. PSoC Designer automatically generates an `.inf` file with the driver in the `\lib` subfolder of the project that contains the USBUART User Module.

Figure 7. New Hardware Wizard – Step 4



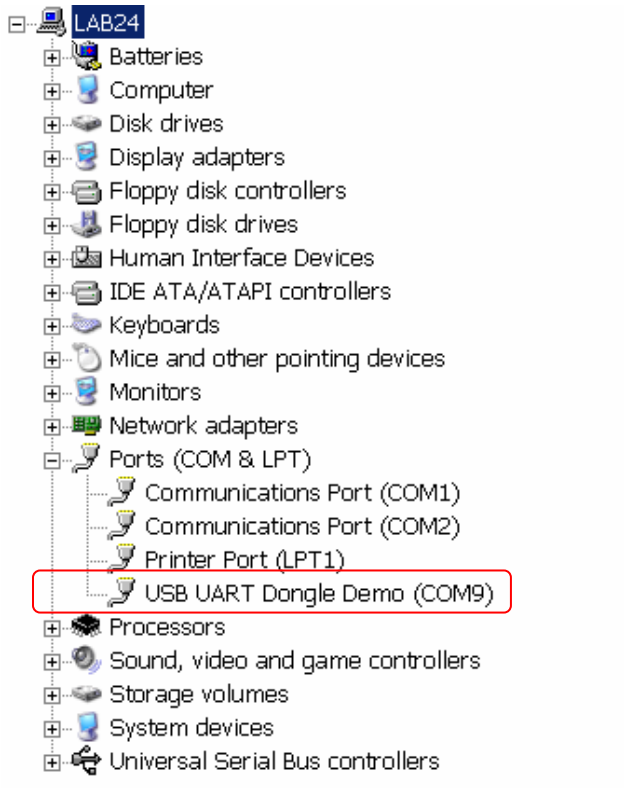

Click **Continue Anyway**.

Figure 8. New Hardware Wizard – Step 5




Click **Finish**.

Figure 9. Device Manager Tree




Now that the USBUART is connected, you can open HyperTerminal, set parameters, and communicate with the board. The system simply echoes back anything that you type in the terminal window. For this echo project the port settings (speed, parity and so on) are ignored, as the communication is handled using USB protocols. However, if you are creating an interface for an existing project and the software that you use to communicate with the device or the device itself requires these settings, the terminal settings can be accessed using USBUART API calls. See the USBUART User Module data sheet for details.

The demonstration project *echo.zip* echoes the text that you type in the HyperTerminal program and also sends the text to the LCD display. The project is designed for the CY3214-PSoCEvalUSB board and uses port 4 to connect to the LCD. The project is included in the project file with this Application Note.

## RX8-to-USB Dongle Project

The second project is the RX8-to-USB dongle. This project demonstrates how to correctly handle data synchronization between the RX8 and USB.

To begin, you place and configure the USBUART User Module as you did in the previous project. In addition, place an RX8 User Module, connect input to the desired pin and set the correct clock speed according to chosen communication speed. The clock rate must be set to eight times the desired bit receive rate.

Add the following code to *main.c*:

Code 2. *ain.c* for RX8-to-USB Dongle Project

```
BYTE baBuf0[32], baBuf1[32];
BYTE bPointer, bActiveBuf, bfBufferFull;

void main() {
  M8C_EnableGInt;

//Start USBUART Operation with 5V operation
  USBUART_Start(USBUART_5V_OPERATION);
//Wait for Device to initialise
  while(!USBUART_Init());

  bPointer = 0;
  bActiveBuf = 0;
  bfBufferFull = 0;

//Start RX8
  RX8_Start(0);
  RX8_EnableInt();

// mail loop
  while (1) {
    while (bfBufferFull == 0);
    bfBufferFull = 0;
 //If TX is ready
    while (!USBUART_bTxIsReady());
 // choose active buffer
    if (bActiveBuf == 0)
      USBUART_Write(baBuf1, 32);
    else
      USBUART_Write(baBuf0, 32);
    //
    while (!USBUART_bTxIsReady());
    // flush the CDC driver buffer
    USBUART_Write(0, 0);
}  }
```

Two buffers, baBuf0 and baBuf1, are used to synchronize the RX8 and USBUART functions. The double buffer avoids data corruption in case a USB transaction takes place at the same time as a UART transaction. The *bActiveBuf* flag indicates disposition of the buffers.

- bActiveBuf == 0 – Indicates that baBuf0 is used to receive data from RX8 and baBuf1 is used to transmit data by USBUART. The data in baBuf1 was previously received via RX8.
- bActiveBuf == 1 – Indicates that baBuf1 is used to receive data from RX8 and baBuf0 is used to transmit previously received data via USBUART.

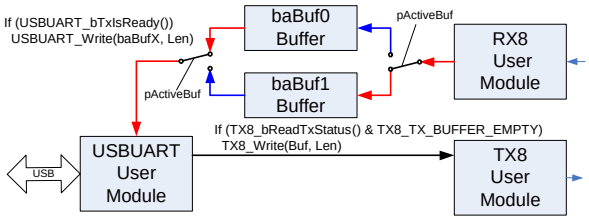
Data is transmitted via USBUART in the main loop and read from TX8 in the RX8 interrupt. You can see the interrupt handler in the user section of *rx8int.asm* located in the *usb_uart_rx.zip* archive.

When the active receive buffer fills up, the RX8 interrupt handler sets the bfBufferFull flag and inverts the value of bActiveBuf.

The data is transmitted to the PC in 32-byte packets. You can decrease the buffer length to 1 if you want to immediately send each byte received by the RX8. This allows you to synchronize the fully asynchronous RX8 and virtually asynchronous USBUART.

The Windows CDC driver has an additional buffer. If the timing of received data is critical to your application, you must use the driver's buffer flush. To flush the buffer, you send a zero-length data packet with the USBUART_Write(0,0) API function. If the timing of received data is not critical to your application you can remove this line from the source code. This makes the transfer from USB to RX8 simpler because the intermediate buffer is not required.

Figure 10. Full UART to USB Block Diagram




The UART-to-USB dongle project, *usb_uart_bridge.zip,* is included in the project file included with this Application Note.

You can adjust the port parameters with the following API functions:

- USBUART_dwGetDTERate – Returns the data terminal rate set for this port in bits per second. You can use this API function to determine the port speed that was set in the terminal software. You can then adjust the clock for the TX8 and RX8 user modules to match this speed.
- USBUART_bGetCharFormat – Returns the number of stop bits.
- USBUART_bGetParityType – Returns the parity type.
- USBUART_bGetDataBits – Returns the number of data bits.

## PC Programming for USBUART

The PC program for the USBUART is the same as if the device is attached to a physical COM port. You can use the MSCOMM object or Win32 API functions. The following example uses Win32 API functions.

Code 3. C# Code Sample to Work With USBUART COM Port

```
//port open
hPortHandle = CreateFile("COM" + PortNum,
GENERIC_READ | GENERIC_WRITE, 0 , 0
OPEN_EXISTING, 0 , 0);

//port settings
DCB dcbCommPort = new DCB();
GetCommState(hPortHandle, ref dcbCommPort);
dcbCommPort.BaudRate = Baudrate;
dcbCommPort.Parity = Parity;
dcbCommPort.ByteSize = ByteSize;
dcbCommPort.StopBits = stopBits;

//read data
ReadFile(hPortHandle, BufBytes, NumBytes,
ref BytesRead, 0);

//write data
WriteFile(hPortHandle, BufBytes, NumBytes,
ref BytesWritten, 0);

//close port
CloseHandle(hPortHandle);
```

**Note.** To open a COM port COM10 and above you must add the prefix string "\\.\" to the COM port name. In C the slash character must be escaped, so the code looks like this:

```
CreateFile( "\\\\.\\COM10",...)
```

A simple C# project, *UsbUartRead.net.zip,* is included in the project file included with this Application Note.

## About the Authors

**Name**: Svyatoslav Paliy
**Title**: Application Engineer
**Background**: Svyatoslav earned a Master of Science degree from Lviv Polytechnic National University (Lviv, Ukraine) in 2000. His interests include the various aspects of embedded systems design, and Windows and Linux programming.

**Contact**: svt@isto.lviv.ua

**Name**: Vadym Grygorenko
**Title**: Sr. Application Engineer
**Background**: Ukraine Solution Center

**Contact**: vad_gr@ukr.net

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/