

## More Serial Ports! SPI-to-Serial for .NET Micro Framework

By John R. Malin and Sean D. Liming  
SJJ Embedded Micro Solutions

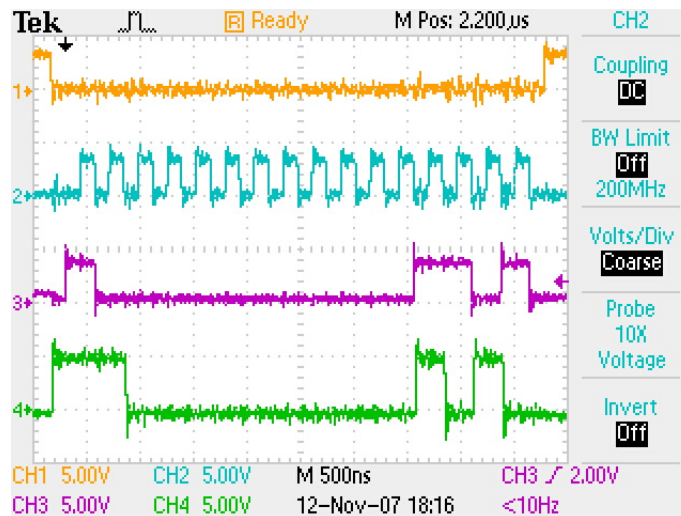
November 2007

RS-232 has been around since the beginning of modern computing. In 1969, a standard was introduced, and since then RS-232 can be found in almost every computing system. Even modern system-on-chips (SOC's) offer a few serial ports. With many embedded systems moving to SOC solutions, finding the right mix of features and capabilities can make it difficult to support all desired hardware IO configurations directly from the SOC.

After listening to customer feedback on different features needed in .NET Micro Framework, serial support is one of the most requested. "Need more serial ports!" Since SOCs limit the number of serial ports because of the pins available on the chip, another serial port, SPI, offers a way to add more RS-232 ports to the system. SPI is an open bus architecture that can support a number of devices such as Analog-to-Digital converters, Real-Time controllers (RTC), SPI-to-Ethernet modules, GPIO expanders, 7-segment LED drivers, various PIC controllers, etc. Best of all you can interface a number of devices from a single SPI port and make the devices address selectable. In the end you can have virtually as many devices as there are GPIO lines to enable them.

Of particular interest, is a SPI-to-RS-232 solution. A brief search of the Internet results in a solution from Maxim Integrated Products (<http://www.maxim-ic.com/>). Maxim provides a couple of IC solutions – MAX3110E (5V) and MAX3111E (3V). Both chips come in DIP packaging, which makes it easy to prototype. Figure 1 shows the circuit for the MAX3110E connected to EMAC's iPac-9302 board. This solution is for a single SPI-to-RS-232, but by adding an OR gate to OR a GPIO line and processor /CS line to control the /CS line of the MAX3110E/3111E, you can add other SPI devices to the SPI bus. For this exercise, we will keep the solution simple.





**Figure 2 SPI Interface Traces of the MAX3110E**

Figure 2 above shows a basic 16-bit data exchange between the iPac-9302 master-host and the MAX3110 client device. The orange signal (1) is the chip select. The aqua signal (2) is the clock. The purple signal (3) is the SSPTX1 (SPI Serial Output) data to the MAX3110. The green signal (4) is the SSPRX1 (SPI Serial Input) data, which is data sent back from the MAX3110. The MAX3110's SPI interface can run at the maximum SPI clock frequency of the iPac-9302, which is 4MHz.

Using the SPI-to-RS-232 circuit, we can write an application to communicate with the SJJCOMM Lite application found in the Embedded Development Kit for the .NET Micro Framework. This simple application for the iPac-9302 receives incoming RS-232 data on interrupt and echoes it back to the SJJCOMM Lite application. The source code for this example application is as follows:

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Hardware.SJJ;
using System.Threading;

namespace MaximS2RInt
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(
                Resources.GetString(Resources.StringResources.String1));
            pgmMaximSpi2RS232 app = new pgmMaximSpi2RS232();
            app.Run();
        }
    }

    public class pgmMaximSpi2RS232
    {
        static SPI.Configuration myMaximSPIPortConfig = new
        SPI.Configuration(Cpu.Pin.GPIO_NONE, false, 0, 0, false, true, 4000,
        SPI.SPI_module.SPI1);
        SPI mySerialSPIPort = new SPI(myMaximSPIPortConfig);

        const UInt16 uiMaximConfig = 0xC001; // 115,200, 8, N, 1; no interrupts
        const UInt16 uiMaximConfigRM = 0x0400; // Enable receive interrupt
        const UInt16 uiMaximWritePrefix = 0x8000; // OR with 8-bit data on lower byte
        const UInt16 uiMaximReadCommand = 0x0000;
        const UInt16 uiMaximReadDataValid = 0x8000; // AND with DOUT and
        //> 0 if data valid

        const UInt16 uiMaximTxFull = 0x4000; // AND with DOUT and if 0 Tx full
    }
}
```



It should be noted that this is a bare-bones test program. RS-232 flow control is not implemented, receive buffer overflow is not monitored, and minimal transmit buffer checking is done in this simple example.

*Windows is a registered trademark of Microsoft Corporation.*