



APPLICATION NOTE 3904

How to Use the MAXQ® Serial Driver to Develop Applications

Abstract: The application note demonstrates how to configure the MAXQ2000 microcontroller EV kit and MAXQ development environment for serial communication. Sample code in C demonstrates the usage and limitations of the MAXQ serial driver, and how to implement RTS/CTS flow control using the MAXQ serial driver.

Introduction

This application note describes how to develop applications using the MAXQ serial driver library on the [MAXQ2000 microcontroller evaluation \(EV\) kit](#). The MAXQ serial driver is available for the [IAR Embedded Workbench® for MAXQ and Rowley Associates' CrossWorks for MAXQ](#), two integrated development and debugging environments for MAXQ applications using C and assembly languages. Sample code in C demonstrates MAXQ serial-driver usage and limitations, and how to implement RTS/CTS flow control using the MAXQ serial driver.

MAXQ Serial-Driver Overview

The MAXQ serial driver provides a set of C functions which enable the user to configure the MAXQ UART for serial communication. These functions are listed in the source file, `maxq_serial.h`, supplied in the MAXQ2000 EV kit.

The MAXQ serial driver provides the following features.

1. Even, odd, and none parity.
2. Separate transmit and receive internal buffers. The size of these buffers is defined at compile time.
3. Non-blocking `serial_write` and `serial_read` routines.
4. A `serial_isr()` function that should be called as part of the interrupt handler by the user application. It computes the parity as "per configuration" and transmits/receives the data on the serial line.
5. Support for the flow control at the application level. The function `serial_stopTx()` stops the transmission of serial data, and `serial_restartTx()` restarts the serial transmission.

The source code for the MAXQ2000 serial driver is available [here](#). The driver can be ported for other MAXQ devices by modifying a few configuration parameters as follows.

1. `maxq_config.h` contains compiler-specific configuration parameters. Modify the following lines in the file to suit your device and compiler:

```
// user configurable options here
#define COMPILER_IAR
#define DEVICE_MAXQ2000
```

The above two lines specify that the driver will be compiled for IAR and MAXQ2000. To compile the driver for CrossWorks for MAXQ, modify the lines as below:

```
// user configurable options here
#define COMPILER_ROWLEY
#define DEVICE_MAXQ2000
```

2. The above two lines let the user include device-specific files for either the IAR or CrossWorks tool chains. `maxq_serial.h` contains driver-specific configuration parameters: driver buffer size, serial-port number, and crystal frequency in Hz. Change the following code to change the driver buffer size:

```
#define RX_RNG_SIZE 256
#define TX_RNG_SIZE 256
```

Modify the following code to specify the serial port (`SERIAL_PORT0` for serial port 0, `SERIAL_PORT1` for serial port 1) for which you want the driver configured:

```
// specify which serial port you are using
#define SERIAL_PORT0
```

Modify the following code to change the crystal clock used for the EV kit. The MAXQ2000 EV kit uses a 16MHz crystal:

```
#define XTAL_CLK 16000000 // 16,000,000Hz
```

Hardware and Software Requirements

To run the application provided in this application note, you need a MAXQ2000 EV kit, a +5V power supply with minimally 200mA capacity, a PC with two serial ports (one for downloading the application onto the EV kit, and the other for communication between the MAXQ application and PC application), a straight-through serial cable, the MAXQ development tool set (IAR Embedded Workbench or CrossWorks 1.0), and the Java™ runtime environment on your PC (including the `commApi` and `BlackBox` example).

Hardware Setup

The MAXQ2000 EV kit should be set up for serial communication with RTS/CTS flow control. The jumper and DIP switch settings follow:

1. Set switch SW3 1-8 to the **off** position.
2. Set switch SW1 1, 2, 5, and 6 to the **off** position and SW3, 4, 7, and 8 to the **on** position.
3. Set switch SW6, 3, and 8 to the **on** position, and rest to the **off** position.

These switch settings connect the RTS, TXD0, RXD0 and CTS signals of the MAXQ2000 EV kit's serial connector to the microcontroller, and enable SW4 DPST to pull the microcontroller's INT11 signal to ground. See **Figure 1**.

```
JU1: connect Pins 1 and 2
JU2: connect Pins 1 and 2
JU3: connect Pins 1 and 2
JU4: open
JU5: closed
JU6: closed
JU7: closed
JU8: closed
JU9: closed
JU10: open
JU11: closed (The MAXQ2000 EV kit is powered by a JTAG interface
board which is powered by a +5V power supply.)
```

4. Connect the serial cable between the MAXQ2000 EV kit and a PC serial port.

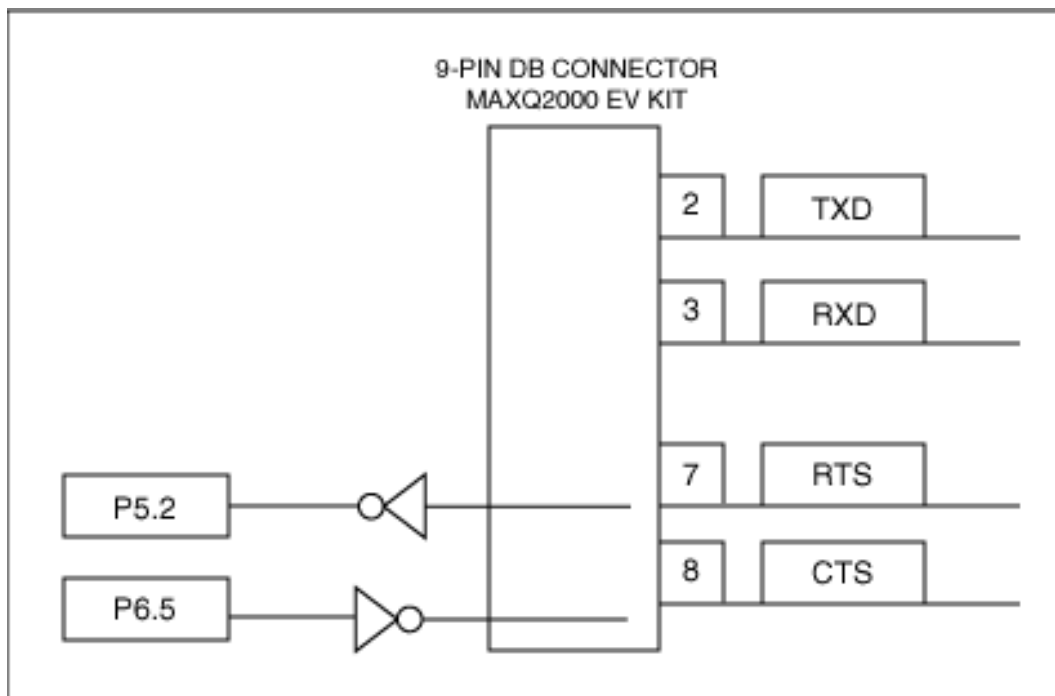


Figure 1. This setup of the MAXQ2000 EV kit's switch settings enables serial communication with RTS/CTS flow control.

Software Setup

1. Download the [demo software](#).
2. Install the development tool set ([IAR Embedded Workbench for MAXQ](#), [CrossWorks for MAXQ](#)).
3. Download and install both the [Java Runtime Environment](#) and the [Java communications API](#).
4. Run the BlackBox example in the Java communications API distribution. If this works correctly, then you should be able to see your host serial ports.

Getting Started with the MAXQ Serial Driver

To begin using the development tools, please review either of two application notes. For the Rowley CrossWorks for the MAXQ, see application note 3698, "[Getting Started with Rowley CrossWorks and the MAXQ2000 Evaluation Kit](#)", or for the IAR Embedded Workbench for MAXQ, see application note 3378, "[Getting Started with the IAR Compiler and the MAXQ2000 Evaluation Kit](#)".

Execute the Java BlackBox sample application on your PC and configure it for 115200 baud, 8 data bits, 1 stop bit, and none parity. Enable the RTC/CTS flow control by clicking on the 'Xmt' check box for the serial port of your choice. The BlackBox setup is shown in **Figure 2**. Note the RTS and CTS lines are green indicating that the hardware flow control is off.

From the demonstration software downloaded earlier, open the corresponding project (.EWW for IAR and .hzip for CrossWorks). Compile the application and download it into the MAXQ EV kit. For detailed instructions on downloading applications to the MAXQ, see the application notes mentioned above. Power on the MAXQ EV kit and you should see serial output from the MAXQ printed onto the BlackBox GUI. Enter some characters in the input window, and observe that the same characters are printed in the BlackBox output window. The keyed characters are transmitted from the PC to the MAXQ EV kit, and in turn, echoed back to the PC. The MAXQ application reports the number of bytes received, which should match the BlackBox report of bytes sent.

Signaling the MAXQ EV kit to Stop Sending the Characters

Click on the RTS text in the BlackBox GUI. This event signals the EV kit to stop transmitting the characters. Notice that the printing in the BlackBox stops. Click again on the RTS text to enable the MAXQ to send characters, and see

that the test string continues to update.

Signaling the PC to Stop Sending the Characters

Press the SW4 switch on the MAXQ EV kit. This signals the PC (BlackBox application) to stop sending the characters. The CTS text on the GUI should turn from green to black. Enter some characters in the BlackBox application and see that the characters are not echoed back. Press the SW4 switch again which turns the CTS signal back to green. The characters that were typed before should echo back.

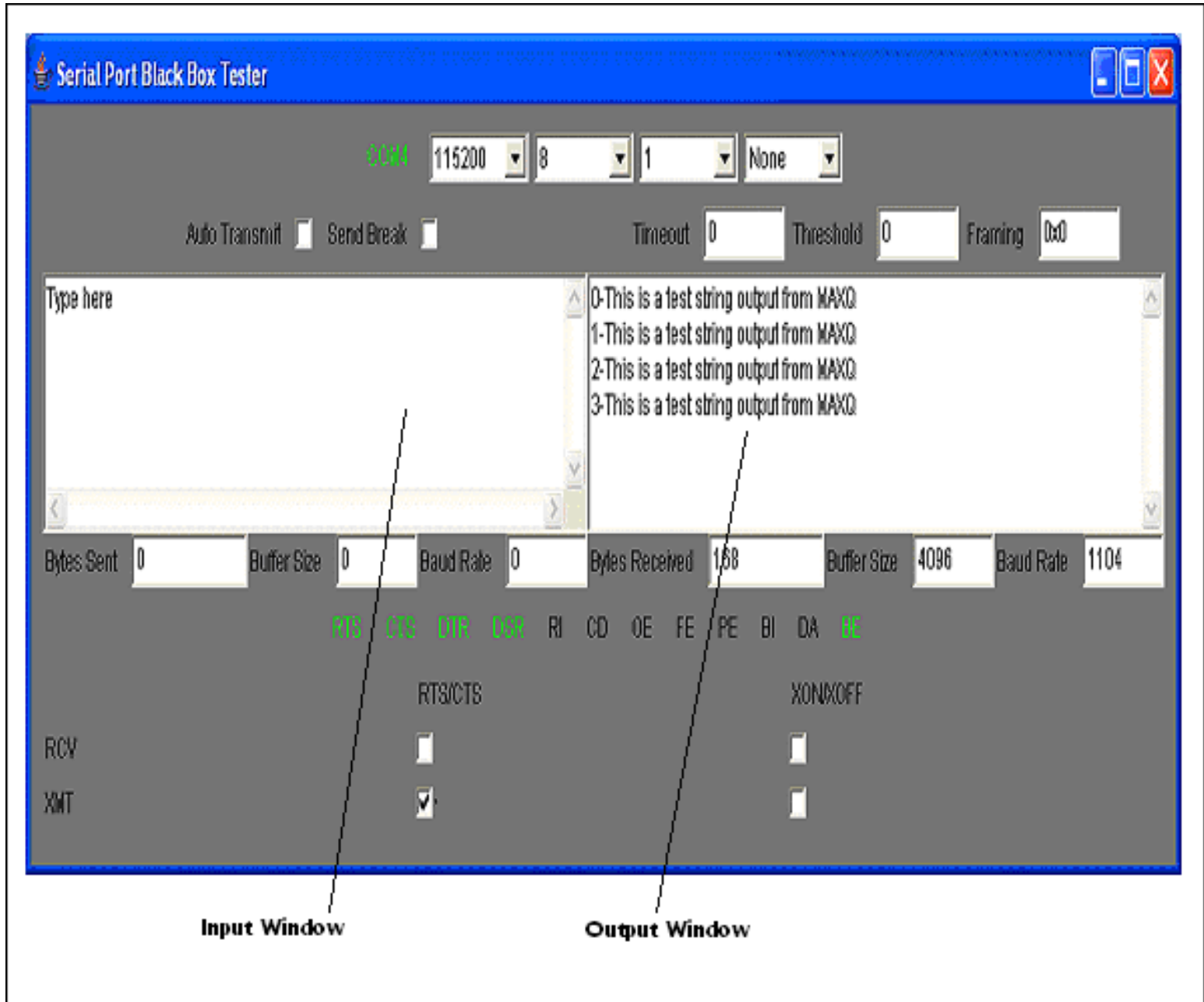


Figure 2. The BlackBox sample application to run on your PC.

Application Overview

The file `sample_serial.c` contains the application's main routine and interrupt service routines (ISR) for servicing the serial interrupts and external interrupts. The firmware does the following:

1. `serial_init()` initializes the serial driver.
2. `serial_setparameters()` configures the serial driver for 115200/8/1/N. You can configure these parameters

according to your application needs.

3. Configures the external interrupts 10 (Int10 is connected to RTS line) and 11 (SW6 #3 is ON).
4. Enables the interrupts for module 1 (external interrupts) and module 2 (UART).
5. Enables global interrupts.
6. Reads and writes from/to the serial port in a loop through `serial_read()` and `serial_write()`.
7. The interrupt service routine handles RTS/CTS control flow. (Pressing the SW4 button asserts or deasserts the RTS line from the EV kit; the RTS line from the PC stops/restarts the serial transmission.)

Limitations and Development Issues

1. The MAXQ serial driver does not support flow control (software/hardware). It is the responsibility of the application to implement the flow control. The sample application provided in this application note describes the hardware (RTS/CTS) flow control.
2. The MAXQ serial driver provided is compiled for MAXQ2000 and 16MHz crystal configurations. Changing the crystal frequency will alter the driver's baud rate computation.
3. The MAXQ serial-driver internal transmit-and-receive buffers' sizes are fixed at compile time as 256 bytes.

Conclusion

The MAXQ compiler provided by IAR and Rowley Associates and the libraries provided by Maxim allow applications written in C to access the power and functionality of MAXQ devices. The MAXQ serial will enable users to deploy applications that use the MAXQ UART.

IAR Embedded Workbench is a registered trademark of IAR Systems AB.

Java is a trademark of Sun Microsystems, Inc.

MAXQ is a registered trademark of Maxim Integrated Products, Inc.

Application Note 3904: www.maxim-ic.com/an3904

More Information

For technical support: www.maxim-ic.com/support

For samples: www.maxim-ic.com/samples

Other questions and comments: www.maxim-ic.com/contact

Automatic Updates

Would you like to be automatically notified when new application notes are published in your areas of interest? [Sign up for EE-Mail™](#).

AN3904, AN 3904, APP3904, Appnote3904, Appnote 3904

Copyright © by Maxim Integrated Products

Additional legal notices: www.maxim-ic.com/legal