

Introduction

The Altera® RS-232 utility is designed as a quick and easy way to exchange data between a personal computer (PC) and the Stratix™ EP1S25 DSP development board or Stratix EP1S80 DSP development board. It consists of a Visual Basic application and a Quartus® II software project. The Visual Basic application selects one of three files and sends the contents of that file to the board via the PC’s serial port. It also monitors the serial port for incoming data and stores it in the “data_back” file. The hardware control logic consists of a UART, two FIFO buffers and three state machines. The utility includes the source code for both the hardware control logic and the Visual Basic Application.

Hardware Control Logic

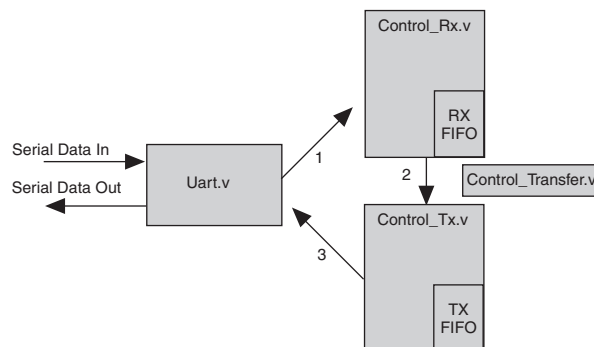
Control_RX.v waits for data to come in via the serial port, and then writes the received data into the RX FIFO buffer. **Control_Transfer.v** moves the data from the RX FIFO buffer into the TX FIFO buffer. **Control_TX.v** reads the data out of the TX FIFO buffer and sends it back to the PC. By default, the state machines expect to deal with data packets of 2,048 bytes. To use the utility in its current form the user must have exactly 2,048 bytes in the file to be transmitted to the board. Two switches on the board control the transfer of data from the RX FIFO buffer into the TX_FIFO buffer and then from the TX FIFO buffer back to the PC. The seven-segment display shows the current status of the utility and reports any errors:

- Zero (0) when the utility is ready for data to be sent from the PC to it.
- One (1) when 2,048 bytes have been received and written to the RX FIFO buffer.
- Two (2) when all data has been transferred from the RX FIFO buffer to the TX FIFO buffer.
- Three (3) when all data has been sent back to the PC.

For details on the operation of the hardware logic, see the comments contained within the respective Verilog HDL files.

Figure 1 shows a block diagram of the hardware logic.

Figure 1. RS-232 Utility Hardware Logic Block Diagram

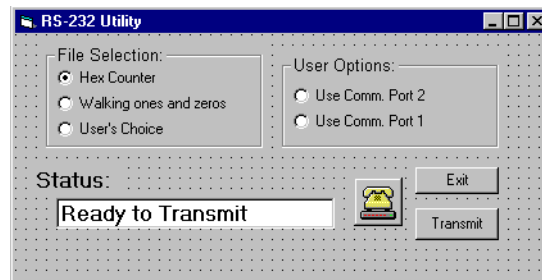


PC User Interface

The PC graphical user interface (GUI) is a Visual Basic Application that selects the desired file and sends it to the PC. Altera provides two files, a hex counter and walking ones and zeros; the user can also select a file. The hex counter consists of the hexadecimal values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, and it repeats that string 127 times for a total of 2,048 bytes. Walking ones and zeros consists of alternating ones and zeros, which repeat 1,024 times, for a total of 2,048 bytes. If the user selects a file, it must be named **My_File.txt** and be placed in the **c:\MegaCore\<Stratix_DSP_Kit-v[version] or Stratix_DSP_Kit_pro-v[version]>\utilities\rs-232_utility** directory. It also must contain exactly 2,048 bytes (unless the user modifies the hardware control logic and Visual Basic application to account for a different number of bytes).

Figure 2 shows the RS-232 utility graphical user interface.

Figure 2. RS-232 Utility User Interface



Setting up the RS-232 Utility

To use the RS-2323 utility, the user must set up the following items.

- Install the DSP Development Kit, Stratix Edition or DSP Development Kit, Stratix Professional Edition on the PC.
- Power up the DSP development board and configure it with the RS-232 **Utility.sof** file. (Refer to *AN 208: Configuring Stratix Devices*, for details on device configuration.)
- Connect a serial cable from one of the PC's communication ports to the RS-232 connector on the development board.

Using the RS-232 Utility

The utility was designed so that the user can replace the **Control_Transfer.v** file with some data processing function. The design must read the data out of the RX FIFO buffer, do whatever processing is desired, and then write the data back to the TX FIFO buffer. SW1 on the board controls the transfer of data from the RX FIFO buffer to the TX FIFO buffer. This switch can also be used as an enable for the user's data processing function. SW2 controls the transfer of data from the TX FIFO buffer to the PC.

In a typical application, the user would design a data processing function that begins by reading data from the RX FIFO buffer, and ends by writing data into the TX FIFO buffer. Then, the Visual Basic Application can send the data to the development board. The universal asynchronous receiver/transmitter (UART) and Control_RX logic stores that data in the RX_FIFO buffer. Pressing SW1 initializes the user's data processing function. When processing completes and all data resides in the TX_FIFO buffer, pressing SW2 sends the data back to the PC.

Modifying the RS-232 Utility

The power of this utility is in its ability to be modified and it was designed to ensure that users could adjust it to meet the needs of a specific application. This section describes some of the adjustments that can be made.

Changing the Baud Rate

As shipped, the utility operates at 115,200 baud. However, the UART baud register is write-enabled, which allows the user to modify the transfer rate. In `Control_RX`, the first process that occurs is a write into this baud register of 346, which results in a baud rate of 115,200. Other common serial communication rates and their respective baud- divisor register value are given below.

| Baud Rate | Divisor Value |
|------------------|----------------------|
| 19,200 | 32 |
| 38,400 | 64 |
| 57,600 | 128 |
| 115,200 | 346 |

If the user change the baud setting in the hardware control logic, the user must modify the Visual Basic application to match it. The baud rates detailed above are commented in the Visual Basic source code and can be implemented by commenting the current setting and uncommenting the desired setting.

Changing the Number of Transmitted/Received Data Bytes

As shipped, the utility deals with exactly 2,048 bytes of data. Increasing or decreasing the number of bytes involves modifying all three state machines, as well as the Visual Basic Application. Each of the three state machines count the bytes in one state or another, and this is the necessary change to allow for more or less data to be processed. In `Control_RX.v` this counting occurs in State 7 (when `Cur_State = 0111`). To increase or decrease the byte quantity, change the value from 2,048 to the desired number of bytes. The user must make the same change for `Control_Transfer.v` in State 1 (`Cur_State = 0001`), and `Control_TX.v` in state 5 (`Cur_state = 0101`). Finally, the user must change the Visual Basic Application to send and receive the correct number of bytes. There are several modifications that must be made, including:

- `MSComm1.InBufferSize = 2048`
This setting specifies the number of bytes that are available for holding received data in the PC.
- `MSComm1.OutBufferSize = 2048`
This setting specifies the number of bytes that are available for holding data to be transmitted from the PC to the development board.
- `MSComm1.InputLen = 2048`
This setting specifies the number of characters that are read from the input buffer when a receive event is detected.
- `MSComm1.RThreshold = 2048`
This setting specifies the number of characters that must be received for a receive event to be detected.
- `MSComm1.SThreshold = 2048`
This setting specifies the number of characters that are transmitted when a transmit event is started.

Also, the variables `data2pc` and `data2board` are allotted 2048 bytes in memory. Change the `dim` statements to account for the new lengths of data.

UART Details

Advanced users may want to use only the UART and design custom control logic to handle the receiving/transmission of data. The UART (as shipped) must use no parity, no CTS/RTS control, and two stop bits. The UART was originally designed for use with a processor, and thus interrupts can be generated. The following sections provide details on the UART's operation.

UART Introduction

The UART used in the RS-232 utility implements simple RS-232 asynchronous transmit-and-receive logic inside an Altera device. The UART sends and receives serial data over two external pins (RxD and TxD). The UART uses a logic 0 for mark, and a logic 1 for space. The UART runs on a single synchronous clock input, `clk`.

Transmitter Logic

The UART transmitter consists of an 8-bit `txdata` holding register and an 8-bit transmit shift register. The `txdata` holding register is written by the `Control_Tx` state machine. The transmit shift register is automatically loaded from the `txdata` register when a serial transmit shift operation is not currently in process. The transmit shift register directly feeds the TxD data pin. Data is shifted out to TxD LSB first.

These two registers provide double buffering; so a new value can be written into the `txdata` register while the previously written character is being shifted out. The transmitter's status can be monitored by reading the `status` register's transmitter ready (`trdy`), transmitter shift register empty (`tmt`), and transmitter overrun error (`toe`) bits. The transmitter logic automatically inserts the correct number of start, stop, and parity bits in the serial TxD data stream as required by the RS-232 specification.

Receiver Logic

The UART receiver consists of an 8-bit receiver-shift register and an 8-bit `rxdata` holding register. The `rxdata` holding register can be read directly. The `rxdata` holding register is loaded from the receiver shift register automatically every time a new character is fully received. These two registers provide double buffering. The `rxdata` register can hold a previously-received character while the subsequent character is being shifted into the receiver shift register.

The receiver's status can be monitored by reading the `status` register's read-ready (`rrdy`), receiver-overrun error (`roe`), break detect (`brk`), parity error (`pe`), and framing error (`fe`) bits. The receiver logic automatically detects the correct number of start, stop, and parity bits in the serial RxD stream as required by the RS-232 specification. The receiver logic checks for four exceptional conditions in the received data, and sets corresponding `status` register bits (`fe`, `pe`, `roe`, or `brk`).

Baud Rate Generation

The UART's internal baud clock is derived from the UART's master clock input, and is generated by a clock divider. The divisor value can be written into the control register of the UART.

rxdata Register

When a new character is fully received via the RxD input, it is transferred into the `rxdata` register, and the `status` register's `rrdy` bit is set to 1. When a value is read from the `rxdata` register, the `status` register's `rrdy` bit is set to 0. If a character is transferred into the `rxdata` register when the `rrdy` bit is set (that is, the previous character has not been retrieved), a receiver-overrun error occurs and the `status` register's `roe` bit is set to 1. New characters are always transferred into the `rxdata` register, whether or not the previous character has been retrieved. Writing data to the `rxdata` register has no effect.

txdata Register

Characters to be transmitted are written directly into the txdata register. Characters should not be written to the txdata register until the transmitter is ready for a new character, as indicated by the trdy bit in the status register. If a character is written to the txdata register when trdy is 0, the results are undefined. The trdy bit is set to 0 when a character is written into the txdata register. The trdy bit is set to 1 when a character is transferred from the txdata register into the transmitter shift register.

For example, assume the UART is idle and a first character is written into the txdata register. The trdy bit is set to 0, then set to 1 when the character is transferred into the transmitter shift register. A second character can then be written into the txdata register, and the trdy bit is set to 0 again. However, this time the first character still occupies the transmitter shift register and is still in the slower process of being transmitted over the TxD output pin. The trdy bit is not set to 1 until the first character is fully shifted out and the second character is automatically transferred into the transmitter shift register. Reading data from the txdata register produces an undefined result.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

Copyright © 2003 Altera Corporation. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services. All rights reserved.