



XAPP699 (v1.0) March 3, 2004

A Software UART for the UltraController GPIO Interface

Author: Glenn C. Steiner

Summary

This application note describes how to implement a Software UART using a few I/O lines of the Xilinx UltraController™ GPIO interface. The Software UART offers many functions of a fabric-instantiated UART without the associated overhead logic. This design provides an effective method of low-cost, external data communications for development, debug, production, and monitoring in the field. Implementation and usage details for the Software UART design are provided. The reference design files contain example source files for both Verilog and VHDL implementations of the Software UART, C source files for the Software UART drivers, and a simple example demonstrating the function of the Software UART.

Introduction

The UltraController embedded processor solution is described in [XAPP672](#): "The UltraController Solution: A Lightweight PowerPC™ Microcontroller" as a complete reference design to be utilized as a lightweight PowerPC microcontroller. The implementation includes a GPIO interface with a 32-bit input port and a 32-bit output port. [Figure 1](#) is a block diagram of the UltraController GPIO interface.

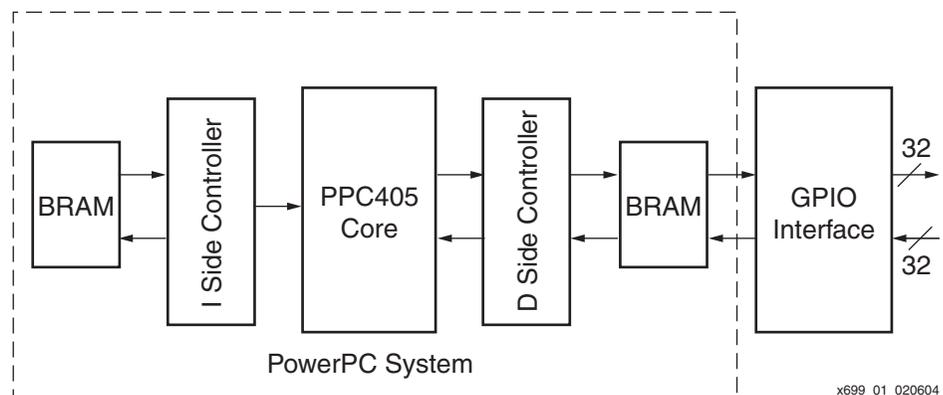


Figure 1: UltraController Block Diagram

[Figure 2](#) shows the Software UART configuration using an UltraController embedded processor with one GPIO input and one or two GPIO outputs.

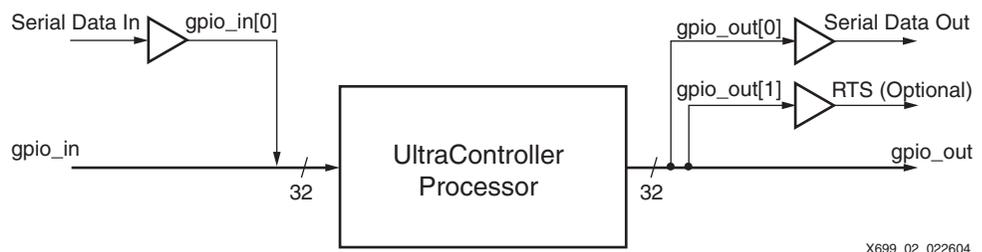


Figure 2: UltraController Software UART Diagram

© 2004 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

The Software UART implements a basic RS232 serial communications protocol, transmitting and receiving data words consisting of a start bit and eight data bits. There are two stop bits for transmit operations and one stop bit for receive operations. The UART is software configurable for any rate up to 115,200 baud when the PowerPC processor is clocked at 100 MHz. Bit transmit and receive operations are implemented entirely via software functions, and timing is based upon the internal PowerPC hardware timer. The Software UART is implemented as a polled device. Thus, it requires the full attention of the PowerPC processor while transmitting or receiving.

Advantages and Characteristics of the Software UART

Technical Advantages

Typically, serial I/O is used with a processor during three phases of a product life cycle:

1. System development and debug.
2. Production testing.
3. System monitoring or control.

Processor serial I/O communication enables status information to be quickly and easily displayed during software debug. By simply “printing” status information, a user can log the software task states and critical variable information. In many situations this procedure is more effective than setting breakpoints and viewing variables with a debugger.

Diagnostic routines created during the product development and debug phases are frequently made integral to the production test process. Serial communication can be used to initiate, control, and retrieve data from the diagnostic routines.

Finally, once a product has been installed, the serial port can be used for system monitoring or control. This usage might be for one-time events, such as when a technician periodically checks or monitors a piece of equipment. Alternatively, the process might be ongoing, where another computer system continuously monitors the system of interest.

Having serial I/O capability with a very small logic footprint is highly desirable. The Software UART implements a simple serial I/O function entirely in software, eliminating the need for an IP UART and its associated bus connections. In contrast, the On-chip Peripheral Bus (OPB) UART Lite, available with the Embedded Development Kit (EDK), occupies approximately 100 logic cells. In addition, the implementation requires both a Processor Local Bus (PLB) and an OPB to support the UART, necessitating additional FPGA logic. Thus, the UltraController Software UART presents a significantly smaller logic solution, using only two or three FPGA I/O pins for serial communication. [Figure 3](#) demonstrates the Software UART communicating with a PC.

```
115200Baud - HyperTerminal
File Edit View Call Transfer Help
UltraController
Serial I/O Demo
Baud-->8,None,1; 115200
The value of 'i' is: 0
The value of 'i' is: 1
The value of 'i' is: 2
The value of 'i' is: 3
The value of 'i' is: 4
The value of 'i' is: 5
Enter text to be echoed and displayed on LCD:
the quick brown fox jumped over the lazy dog
now is the time for all good men to come to the aid of their country
Connected 0:47:16 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
```

x699_03_030104

Figure 3: HyperTerminal Session Using the UltraController Software UART

Technical Characteristics and I/O Interface

Table 1 lists the characteristics of the Software UART.

Table 1: Software UART Characteristics

| Parameter | Value |
|-----------|--|
| Baud Rate | User-selectable rates from 110 to 115,200. Non-standard rates also are available |
| Data Bits | 8 |
| Parity | None |
| Stop Bits | 1 for Receiving 2 for Transmitting |

The communication signals from the FPGA must be connected to a device providing the appropriate set of line drivers and receivers, such as a MAX3223.

By modifying the C source code drivers, a user can change the number of data and stop bits, and add parity.

The Software UART has the following constraints:

- The drivers are equivalent to Level 0 drivers. The processor polls for input and waits for character transmission, and is 100% utilized when transmitting or receiving a character.
- There is no transmit FIFO. When a character is transmitted, the processor is 100% utilized for the full duration of the character transmission.
- There is no receive FIFO. The processor must loop while waiting to receive a character to receive it reliably.
- The PowerPC hardware timer controls the bit timing for transmit and receive operations. With a 100 MHz UltraController device, baud rates of up to 115,200 were tested. The design must be simulated to ensure that the processor can meet the required data rates.
- The RS232 Standard (EIA232 Standard) contains timing requirements, as well as data speed versus cable length specifications. The higher the data rate, the shorter the cable. Information about the RS232 standard is at: <http://www.eia.org/>.

Implementing and Using the Software UART

Reference Design Environment

The UltraController reference design is built and tested using the following Xilinx software products:

- ISE - Version 6.1.03i (Service Pack 3)
- EDK - Version 6.1.02i Build EDK_G.12+3 (Service Pack 3)

The reference design is demonstrated utilizing the following hardware:

- Memec Design (Insight) Virtex-II Pro™ P4 Development Board
- Xilinx Parallel Cable IV for bitstream download and code debug
- PC running a terminal emulation program
- Serial cable connected between development board and a PC

Refer to the ReadMe/QuickStart file (`readme_uart.txt`) for additional information on building the UltraController reference design, using the above products, update information, and for board configuration data.

HDL Implementation and Software Implementation

Supplied Files

The UltraController Software UART reference design consists of a completed and tested EDK PowerPC design. The HDL reference implementations provided with this application note includes code examples in Verilog and VHDL. The PowerPC software reference design example is in C. The UltraController Software UART is supplied as a ZIP archive. The archive must be unzipped on top of an UltraController reference design. [Table 2](#) lists the files contained within the UltraController Software UART archive.

Table 2: Source Files Included in the UltraController Software UART Archive

| File Name | Description |
|---|--|
| ..\read_me_uart.txt | Read Me file for installation and configuration of the Software UART. |
| ..\system_4i_4d_uart_vhdl.xmp, ..\system_4i_4d_uart_vlog.xmp, ..\system_8i_8d_uart_vhdl.xmp, ..\system_8i_8d_uart_vlog.xmp | Platform Studio configuration files for 4i_4d / 8i_8d and Verilog / VHDL configurations. |
| ..\sw\gpio.c | This file contains the GPIO code library supplied with the UltraController reference design with the addition of single bit input and output routines for the serial port, and a routine to set the RTS signal. |
| ..\sw\soft_uart.c | This file contains the mid-level drivers for UART initialization, character input and output, and string output. |
| ..\sw\display_uart_io.c | This file contains functions that output a string and an integer to either the LCD or the Software UART. |
| ..\sw\uart_demo.c | This file contains the top-level demonstration of the Software UART. Includes examples of sending and receiving characters, sending strings, sending strings with integers, and sending strings via the Xilinx <code>print</code> function, which utilizes <code>STDOUT</code> . |
| ..\19200Baud.ht, 38400Baud.ht, 57600Baud.ht, 115200Baud.ht | Configuration files for the Microsoft® HyperTerminal program. Set for COM-1. |
| ..\projnav\uc_4i_4d.v, ..\projnav\uc_4i_4d.vhd, ..\projnav\uc_8i_8d.v, ..\projnav\uc_8i_8d.vhd | Top-level Verilog and VHDL modules instantiating the UltraController with Software UART. |
| ..\projnav\uc_4i_4d.ucf, ..\projnav\uc_8i_8d.ucf | User constraint files for the design. Each file adds to the original UltraController constraint file pins for Serial Input, Serial Output, and RTS. |
| ..\projnav\tf_uc.tf, ..\projnav\tb_tf.vhd | Verilog or VHDL Testbench for the UltraController Software UART. |

The UltraController Software UART is delivered in two tested archives. The first archive is used with single PowerPC processor FPGAs (XC2VP4 or XC2VP7), and the second is used with dual PowerPC processor FPGAs (XC2VP20 through XC2VP100). All configurations were simulated and tested in hardware. The single processor version was tested with the Memec Virtex-II Pro Development Board containing an XC2VP4-FG456. Serial communication with the dual processor version was tested with the Xilinx HW-AFX-FF1152-300 board containing an XC2VP20-FF1152.

HDL Implementation

The UltraController Software UART is delivered with a sample top-level module: UltraController_Demo. The UltraController_Demo module instantiates the following modules: uc_4i_4d or uc_8i_8d.

The uc_4i_4d or uc_8i_8d module contains the PowerPC core, JTAG unit, reset controller, block RAM interfaces, block RAM, and the GPIO interface module. Both Verilog and VHDL designs are available in the reference design files. The HDL code is similar to the code provided for the original UltraController reference design. The design uses gpio_in[0] for serial data input and gpio_out[0] for serial data output. The design ties the RTS signal line to logic zero on the development board for proper functioning of many PC-connected UARTs. The UltraController_Demo module contains the following example Verilog code:

```

module UltraController_Demo (sys_clk, nsys_rst, gpio_in, gpio_out, ser_in,
ser_out, RTS);
input    sys_clk;
input    nsys_rst;
input    [29:31]gpio_in;           // 3 inputs for Push Buttons
output   [16:31]gpio_out;         // 15 outputs for LCD and LEDs
input    ser_in;                  // serial input
output   ser_out;                 // serial output
output   RTS;                    // RTS control line for Serial Port

// Wire the RTS line low
assign RTS = 0;

//Instantiating BUFGP on Input Clock
wire uc_sys_clk;
BUFGP U1 (.I(sys_clk), .O(uc_sys_clk));

//Wire the UltraController pins to the outside
wire [0:31] gpio_in_s;           // gpio port wires
wire [0:31] gpio_out_s;
assign gpio_in_s = gpio_in | ser_in << 31;
assign gpio_out = gpio_out_s;
assign ser_out = gpio_out_s[0];

// Instantiate the UltraController Core
system uc_4i_4d (
    .sys_rst      (~nsys_rst)    // I - Active High reset
    .sys_clk      (uc_sys_clk) , // I
    .gpio_out     (gpio_out_s),  // O
    .gpio_in      (gpio_in_s)    // I
);

endmodule

```

External Port Connections

Refer to [XAPP672](#) for a detailed description of the UltraController port connections. [Table 3](#) lists the externally available pins as described in XAPP672.

Table 3: HDL Module Port Connections

| Port | I/O | Description |
|-----------------|-----|--|
| Nsys_rst | I | UltraController module reset (active Low) |
| sys_clk | I | CPU clock. Clock signal for the PowerPC core |
| gpio_out[16:31] | O | UltraController output port |
| gpio_in[29:31] | I | UltraController input port |

Table 4 lists the GPIO pins that are used for the Software UART.

Table 4: HDL Module Pins Used for Serial Communication

| Port | I/O | Description |
|---------|-----|---|
| ser_out | O | Serial data output |
| RTS | O | Optional RTS. Tested, but not used in this reference design. Not connected as provided. |
| ser_in | I | Serial data input |

Table 5 shows the gpio_out and gpio_in pins connected in the Software UART design. Where noted, several pins from the original UltraController reference design were left connected but unused.

Table 5: HDL Module GPIO Pin Connections

| Bit Number | Code Bit Number | Description | Comments |
|--------------------|-----------------|-------------|---------------------------------------|
| Output Port | | | |
| gpio_out[31] | 0 | LCD D0 | |
| gpio_out[30] | 1 | LCD D1 | |
| gpio_out[29] | 2 | LCD D2 | |
| gpio_out[28] | 3 | LCD D3 | |
| gpio_out[27] | 4 | LCD D4 | |
| gpio_out[26] | 5 | LCD D5 | |
| gpio_out[25] | 6 | LCD D6 | |
| gpio_out[24] | 7 | LCD D7 | |
| gpio_out[23] | 8 | LCD RS | |
| gpio_out[22] | 9 | LCD EN | |
| gpio_out[21] | 10 | -- | |
| gpio_out[20] | 11 | Sound | Left connected but not used in design |
| gpio_out[19] | 12 | LED 1 | Left connected but not used in design |
| gpio_out[18] | 13 | LED 2 | Left connected but not used in design |
| gpio_out[17] | 14 | LED 3 | Left connected but not used in design |
| gpio_out[16] | 15 | LED 4 | Left connected but not used in design |
| Input Port | | | |
| gpio_in[31] | 0 | SW 1 | Left connected but not used in design |
| gpio_in[30] | 1 | SW 2 | Left connected but not used in design |
| gpio_in[29] | 2 | SW 3 | Left connected but not used in design |

Software Implementation

The reference design provides a library of functions for performing I/O via the Software UART. The functions encompass character input and output, string output, string followed by integer output, and control of the RTS line. Standard input (STDIN) and standard output (STDOUT)

have also been mapped to the Software UART via implementation of the `inbyte` and `outbyte` functions.

Table 6 lists the functions in `soft_uart.c` used by a calling software program.

Table 6: Software UART I/O Functions from `soft_uart.c`

| Function | Input | Returns | Description |
|-----------------------------------|-------------------------------|-------------------|--|
| <code>UART_Init</code> | <code>int baudrate</code> | | Sets the UART baud rate. The baud rate can be any value, standard or non-standard. |
| <code>UART_Char_in</code> | | <code>char</code> | Returns a character read from the serial input pin. |
| <code>UART_Char_out</code> | <code>char character</code> | | Outputs the passed character via the serial output pin. |
| <code>UART_String_out</code> | <code>char *string_out</code> | | Outputs the string for which a pointer has been passed via the serial output pin. Calls <code>UART_Char_out</code> . |
| <code>UART_String_out_crlf</code> | <code>char *string_out</code> | | Outputs the string for which a pointer has been passed via the serial output pin. Then outputs carriage return and line feed characters. Calls <code>UART_String_out</code> and <code>UART_Char_out</code> . |
| <code>inbyte</code> | | <code>char</code> | Implements the STDIN function for a single byte. Calls <code>UART_Char_in</code> . |
| <code>outbyte</code> | <code>char character</code> | | Implements the STDOUT function for a single byte. Calls <code>UART_Char_out</code> . |

The following sample code segment demonstrates initializing the UART, reading a character, and echoing it back:

```
#define BAUDRATE 9600 // define UART Baud rate
char character; // variable for received character
// initialize the soft UART baud rate for defined rate
UART_Init (BAUDRATE);
// fetch a character using the UART char in function
character = UART_Char_in();
// transmit a character using the UART char out function
UART_Char_out(character);
```

The following sample code segment demonstrates sending a line of text, separate carriage return and line feed, and then a line of text with a carriage return and line feed:

```
char line1[17], line2[17]; // Text Output buffers
strcpy (line1, "UltraController");
strcpy (line2, "Serial I/O Demo");

UART_String_out (line1); // Send line of text to UART
UART_String_out_crlf (""); // Send a CRLF
UART_String_out_crlf (line2); // Send second line of text to UART with a
CRLF
```

The following sample code segment demonstrates sending a line of text using the Xilinx `print` function, which uses `STDOUT`. The text is preceded and followed by a carriage return and line feed.

```
// following call is to the Xilinx print function using STDOUT
print ("\r\nEnter text to be echoed and displayed on LCD:\r\n");
```

[Table 7](#) identifies the `gpio.c` function used by a calling software program.

Table 7: Software UART Function from `gpio.c`

| Function | Input | Returns | Description |
|----------------------------|-----------|---------|--|
| <code>GPIO_UART_RTS</code> | char data | | Bit zero of "data" sets the RTS signal line High or Low. |

Frequently the RTS line must be placed at logic zero to enable remote serial transmission. The RTS signal is used in some situations to perform a hardware handshake flow control for data being received. The RTS function has been tested for correct signaling via the RTS signal line. The use of RTS flow control might not work with all hardware UARTs. The following code segment demonstrates control of the RTS line:

```
// Optional setup and control of UART RTS line -- Following has been tested
// but commented out to minimize usage of GPIO port bits
GPIO_UART_RTS ( 0 ); // set RTS enabling terminal to send to soft UART
```

[Table 8](#) lists the functions from `display_uart_io.c` used by a calling software program.

Table 8: I/O Functions from `display_uart_io.c`

| Function | Input | Returns | Description |
|---------------------------|---|-------------|--|
| <code>UART_int_out</code> | char *message, int value | | Outputs the string for which a pointer has been passed via the serial output pin followed by the unsigned integer passed. Calls <code>UART_String_out</code> , <code>UART_String_out_crlf</code> , and <code>Int_Str</code> . |
| <code>LCD_int_out</code> | char *message, int value | | Calls the LCD display routine included with the UltraController reference design, displays the string for which a pointer has been passed on Line 1, and displays the unsigned integer passed on Line 2. Calls <code>LCD_Write</code> and <code>Int_Str</code> . |
| <code>Int_Str</code> | unsigned int value, int width, int trim | *asc_string | Returns a pointer to a string of the ASCII representation of the unsigned integer number passed. The <code>width</code> parameter indicates the number of digits to convert. If <code>trim</code> is 1, leading zeros are removed from the string. |

The functions in [Table 8](#) are useful for diagnostic code in the display of an integer variable.

The following sample code segment demonstrates sending the integer "i" to the UART with an accompanying message:

```
int i;
for (i=0; i<5; i++)
    UART_int_out ("The value of 'i' is: ",i);
```

System Performance and Timing Considerations

The Software UART uses the hardware system timer to control the output bit rate and the input bit timing.

For serial output, the hardware system timer is read and then each data bit is output at the appropriate bit period referenced to the original start time. Thus, the bit rate is adjusted continuously each bit period for timer and processor overhead variations. As shown in ["Appendix A: Simulation Examples"](#), the eighth bit has a maximum timing error of only 0.3% with a 115,200 baud output.

For serial input, the processor loops while waiting for the start bit. Once the start bit is detected, the timer is read, and the processor reads the first data bit one and one-quarter bit periods later. One quarter of a clock period was chosen to accommodate software overhead at high baud rates. Each subsequent bit is read at the appropriate bit period referenced to the original start time. Thus, the bit read rate is continuously adjusted each bit period for timer and processor overhead variations. As shown in [“Appendix A: Simulation Examples”](#), the eighth bit has a maximum timing error of 0.5% with a 115,200 baud input.

The reference design was simulated using ModelSim with a 100 MHz processor clock rate, compiler optimization Level 0 (Off) and at Level 2, and create symbols for debugging enabled. The design also was tested in hardware with processor clock rates of 50 MHz and 100 MHz.

At 100 MHz and compiler optimization Level 0, the software overhead for transmission of a single bit is approximately 2.2 μ s. The bit time is 8.68 μ s at 115,200 baud. As verified via simulation, [Table 9](#) shows the maximum baud rate versus clock frequency with compiler optimization Level 0.

Table 9: UltraController Clock Rate versus Maximum Baud Rate

| UltraController Clock Rate | Maximum Baud Rate |
|----------------------------|-------------------|
| 100 MHz | 115,200 |
| 50 MHz | 57,600 |
| 10 MHz | 19,200 |

Because individual designs vary, simulation timing is recommended.

As an additional consideration, there is no buffering for the serial input. The processor must begin execution of the serial input routine prior to the receipt of a start bit. Not doing so might result in loss or misinterpretation of the received character.

Refer to [“Appendix A: Simulation Examples,”](#) for additional timing analysis and timing considerations.

Conclusion

For simple control, system monitoring, or I/O functions, the UltraController reference design is an ideal solution. The inclusion of a Software UART function with the UltraController provides a mechanism to easily display and capture valuable information about the functioning of the processor code as well as the FPGA during debug, factory test, and in the field. The Software UART provides many of the capabilities and functions of a fabric-instantiated UART without the associated logic overhead. The Software UART provides reliable serial communication up to 115,200 baud.

Appendix A: Simulation Examples

The UltraController Software UART was simulated in Verilog and VHDL with the LCD display routines and several UART string output routines bypassed. The following subsections provide examples of the simulation results with the UART operating at 115,200 baud. The examples show serial data input and output, and demonstrate the software overhead observed and compensated for.

Post Place and Route Simulation – Serial Output Example

Processor Clock Rate: 100 MHz

Compiler Optimization: Level 0 (Off)

Create Symbols for Debugging (-g option): On

Bypass LCD for Simulation: Enabled

The simulation screen capture in [Figure 4](#) shows the Carriage Return character (0D h) being output followed by the Line Feed character (0A h).

Between the two cursor lines is the data character (CR = 0D h) in reverse bit order. The time for the eight bits is 69.4 μ s, or 8.68 μ s per bit. The desired bit period for 115,200 baud is 8.68 μ s per bit, or 69.4 μ s for eight bits. The timing error at the eighth bit is 0.00%.

The start bit is observed to the left of the first cursor line.

The two stop bits are observed to the right of the second cursor line.

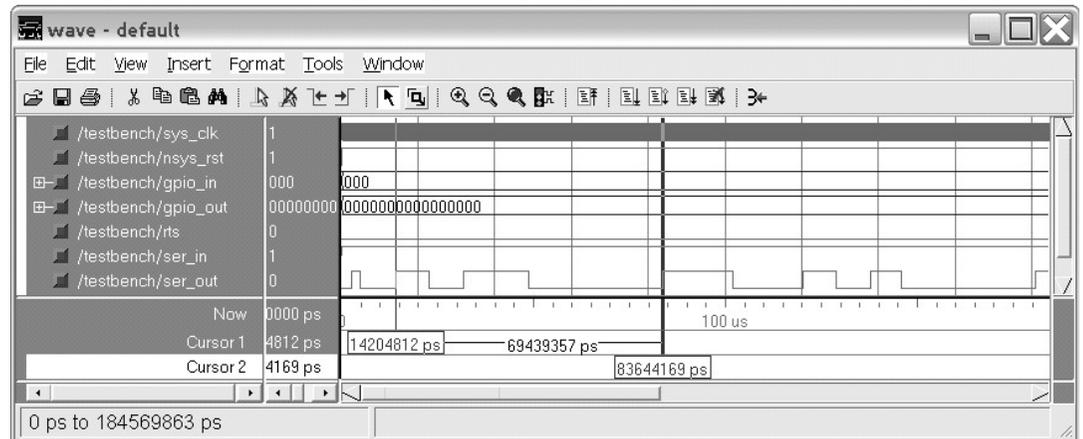


Figure 4: CR (0D h) and LF (0A h) Output

Post Place and Route Simulation – Serial Input and Output Example

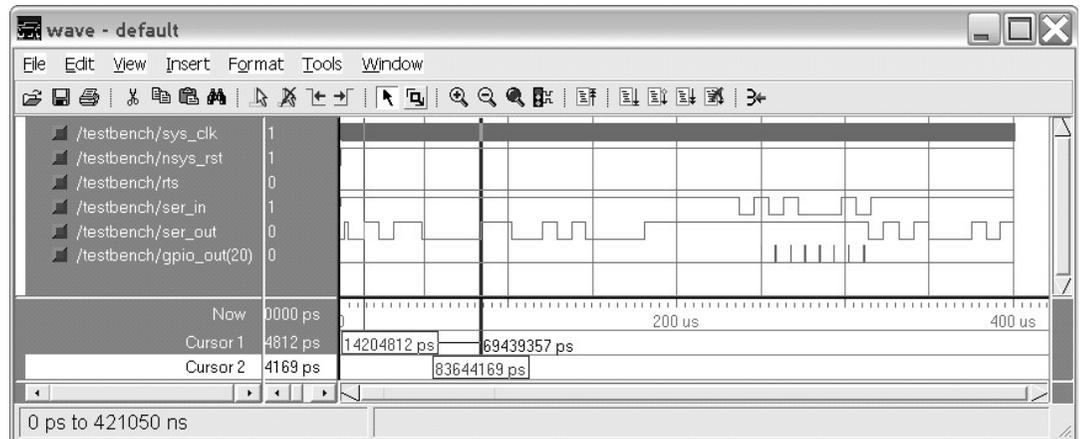
Processor Clock Rate: 100 MHz

Compiler Optimization: Level 0 (Off)

Create Symbols for Debugging (-g option): On

Bypass LCD for Simulation: Enabled

The simulation screen capture in [Figure 5](#) shows the Carriage Return character (0D h) being output followed by the Line Feed character (0A h). Next the Software UART reads the character E (45 h) and then writes it. Serial input is on gpio_in[0], and serial output is on gpio_out[0].

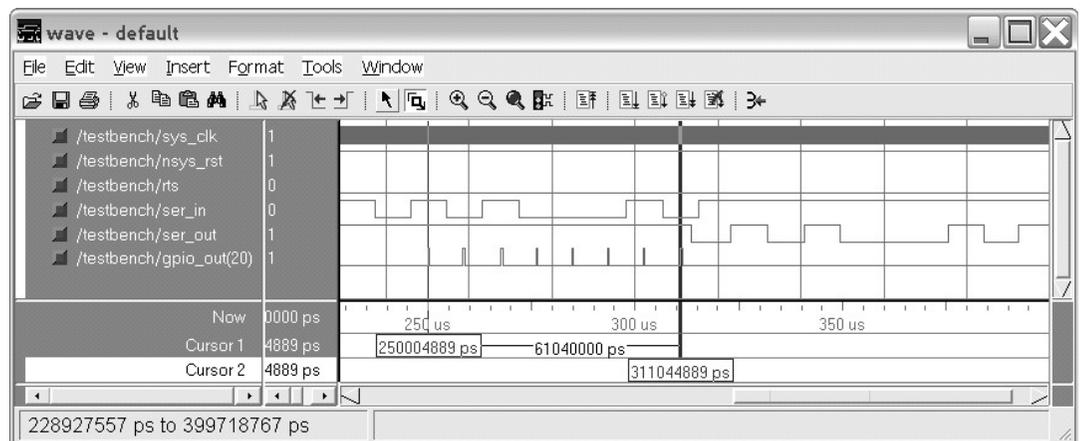


x699_05_022904

Figure 5: CR and LF Output Followed by “E” Input and “E” Output

[Figure 6](#) is an enlarged view of the character input and output sequence. The gpio_out[20] line was toggled at each bit sample period. The point of sampling should be 25% into each bit period. With the fast 115,200 baud rate and software overhead, the sample point is nearly centered for each bit period. Note that the start bit for the echoed character occurs slightly prior to the stop bit on input, which is an indicator that for simple character acquisition the Software UART successfully keeps up with a 115,200 baud input data stream.

The data character (E = 45 h) is located between the two cursor lines in reverse bit order. The time for the eight bits (seven sample periods) is 61.04 μ s or 8.72 μ s per bit. The desired bit period for 115,200 baud is 8.68 μ s per bit. The timing error at the eighth bit is 0.5%.



x699_06_022904

Figure 6: Enlarged View – “E” Input and “E” Output

Post Place and Route Simulation – Serial I/O with Compiler Optimization

Processor Clock Rate: 100 MHz

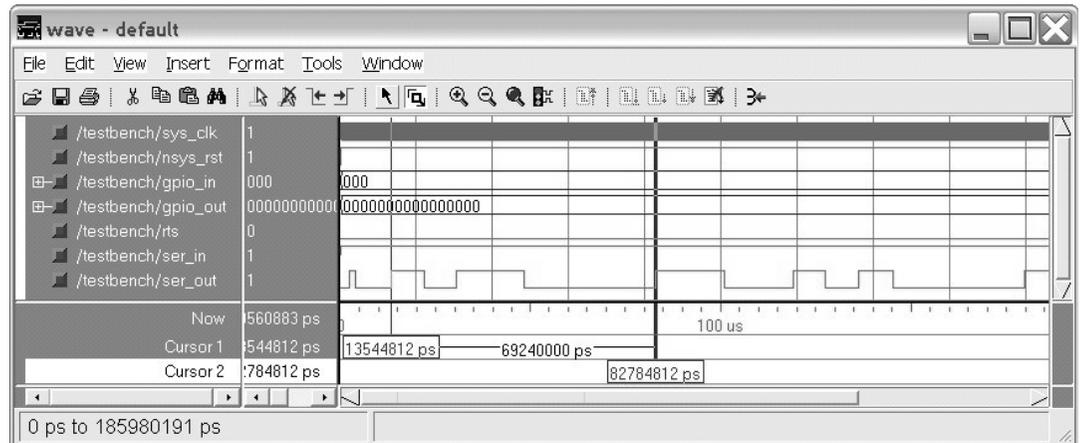
Compiler Optimization: Level 2

Create Symbols for Debugging (-g option): On

Bypass LCD for Simulation: Enabled

The previous example was compiled with optimization set to level 2 and simulated to demonstrate the stability of the design with changes in software overhead. In this example, critical timing remained essentially unchanged.

Figure 7 shows the data character (CR = 0D h) located between the two cursor lines in reverse bit order. The time for the eight bits is 69.24 μ s or 8.66 μ s per bit. The desired bit period for 115,200 baud is 8.68 μ s per bit, or 69.4 μ s for eight bits. The timing error at the eighth bit is 0.3%.

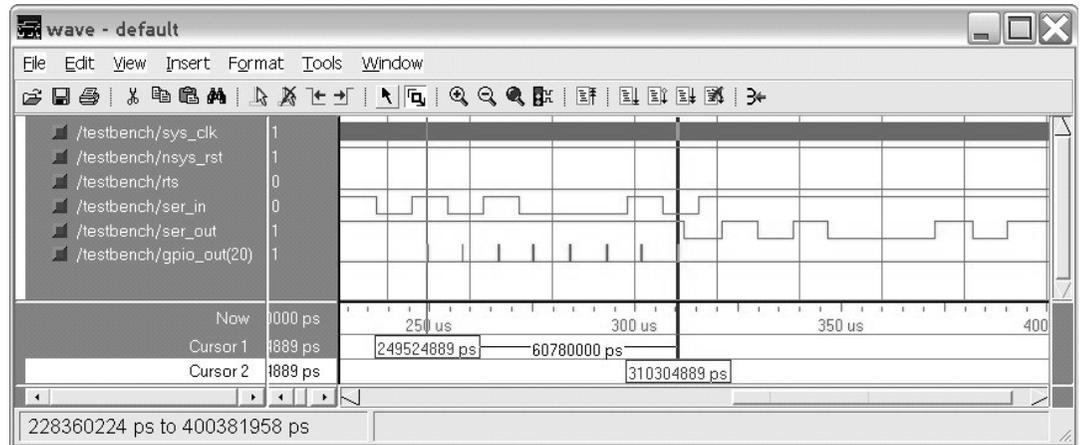


x699_07_022904

Figure 7: CR and LF Output

Figure 8 is an enlarged view of the character input and output sequence. The gpio_out[20] line was toggled at each bit sample period. The point of sampling should be 25% into each bit period. With compiler optimization turned on, software overhead is reduced and the sample point is closer to 25% into the bit period. Note that the start bit for the echoed character now occurs *significantly* prior to the input stop bit, which is an indicator that for simple character acquisition the Software UART successfully keeps up with a 115,200 baud input data stream.

The data character (E = 45 h) is located between the two cursor lines in reverse bit order. The time for the eight bits (seven sample periods) is 60.78 μ s, or 8.68 μ s per bit. The desired bit period for 115,200 baud is 8.68 μ s per bit. The timing error at the eighth bit is 0.00%.



x699_08_022904

Figure 8: Enlarged View – “E” Input and “E” Output

Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|----------|---------|-------------------------|
| 03/03/04 | 1.0 | Initial Xilinx release. |