Debugging with System Console Over TCP/IP

AN624-1.0 Application Note

This application note describes software that supports debugging your custom SOPC Builder component with the Altera® System Console over a TCP/IP communication channel. The application note provides a software application that reads System Console commands from a TCP/IP socket and converts them from the Avalon Streaming (Avalon-ST) packet protocol format in which they arrive at the FPGA, to the appropriate Avalon Memory-Mapped (Avalon-MM) commands. It can pass those commands to any Avalon-MM slave component in the SOPC Builder system that is connected to the Nios II processor data master port. The software application converts the Avalon-MM responses to the Avalon-ST packet protocol format and passes them back through the TCP/IP communication channel.

You can use this design example as a basis for testing your own custom SOPC Builder component using the System Console over a TCP/IP connection, by replacing the component under test in the design example with your own custom component. The only requirement is that the designated SOPC Builder component have an Avalon-MM slave port connected to the Nios II processor data master port.

Introduction

The System Console is an interactive console for low-level system debug of SOPC Builder based systems over various communication channels, including JTAG and TCP/IP. The System Console provides read and write access to the IP cores instantiated in your SOPC Builder system. The System Console is useful for low-level scripted or interactive testing and debugging of IP cores.

This application note demonstrates the use of TCP/IP as a communications channel for the System Console. Using an already established System Console packet protocol, this solution provides a flexible TCP/IP service for use with the System Console. The solution leverages full System Console support on the host to access this TCP/IP channel. The pluggable architecture of System Console supports this additional host-target communications channel. The System Console has a simple set of Tcl-based commands for communicating with various parts of your SOPC Builder system.

The `sctcp` application provides a channel for communication over TCP/IP sockets from the System Console running on a development host to an SOPC Builder system running on an FPGA. The `sctcp` application is a sockets-based application for the Nios II processor that leverages the NicheStack TCP/IP networking stack.

The example hardware design includes an Avalon-MM SOPC Builder on-chip memory component that the host controls through the TCP/IP networking stack. You can substitute your own custom SOPC Builder component with an Avalon-MM slave port and debug it using this methodology.

101 Innovation Drive
San Jose, CA 95134
www.altera.com

After completing this document, you have the knowledge to use the System Console to debug Avalon-MM SOPC Builder peripheral component hardware through a TCP/IP connection.

Prerequisites

This application note assumes that you are familiar with reading and writing embedded software and that you have read and followed the step-by-step procedures for building a microprocessor system in the *Nios II Hardware Development Tutorial*.

This application note also assumes that you are familiar with networking setup requirements. If a DHCP server is available, the software uses it to obtain an IP address. Otherwise, the software uses a predefined IP address specified in the **sctcp.h** software application source file. To gain additional familiarity with network setup, refer to the *Using the NicheStack TCP/IP Stack – Nios II Edition Tutorial*.

For complete details of the MicroC/OS-II real-time operating system for the Nios II processor, refer to the *MicroC/OS-II Real-Time Operating System* chapter of the *Nios II Software Developer's Handbook*.

For complete details of NicheStack TCP/IP Networking Stack initialization and configuration for the Nios II processor, refer to the *Ethernet and the NicheStack TCP/IP Stack – Nios II Edition* chapter of the *Nios II Software Developer's Handbook*.

System Console tcp_master TCP/IP service

The example design uses the tcp_master service of the System Console, available in the Quartus II software v10.0. This service enables the System Console to control any Avalon-MM slave. The example design does not demonstrate the tcp_bytestream service provided by the System Console to control Avalon-ST components.

For more information about System Console service masters and the System Console commands that use a service master, refer to the *Analyzing and Debugging Designs with System Console* chapter in volume 3 of the *Quartus II Handbook*.

Software Design Overview

The software creates a MicroC/OS-II task for communication with System Console using NicheStack TCP/IP Networking Stack sockets. The task reads System Console commands in the form of bytes from a TCP/IP socket. These bytes represent Avalon streaming packet data. The task then extracts, interprets and executes each System Console command by reading and writing to the memory mapped interface of the SOPC Builder component under test. Next, the task converts the System Console command results to Avalon streaming packet data bytes and writes the bytes to the TCP/IP socket for transmission back to the System Console tool running on the development host.

Communicating over the TCP/IP channel, System Console uses the tcp_master master service to control any Avalon-MM slave SOPC Builder component. An on-chip memory serves as an example SOPC Builder hardware component under test in this design. Connect the slave port for any Avalon-MM SOPC Builder component that you desire to test to the Nios II processor's data master port in the SOPC Builder connection panel.

Hardware and Software Requirements

This tutorial requires the following hardware and software:

- Quartus® II software version 10.0 or later
- Nios II Embedded Design Suite (EDS) version 10.0 or later
- Altera USB-Blaster™ cable
- RJ-45 connected Ethernet cable on the same network as the PC development host

This application note uses as a reference the following Altera development kit board:

- Nios II Embedded Evaluation Kit, Cyclone® III Edition (NEEK)

To complete this tutorial, your Altera development board must have the following connections:

- Connected through a USB-Blaster connection to the host computer
- Connected through an Ethernet cable to the same network as the PC development host

Design Example Files

The software design files for this application note are available from the Debugging with System Console over TCP/IP Design Example web page of the Altera website. The hardware design files for the design walkthrough are available from the Nios II Ethernet Standard Design Example web page.

The following sections describe the files.

Hardware Design Files

The Nios II Ethernet Standard Design Example web page of the Altera website contains the hardware design files that correspond to this design example walkthrough. The system is available in several different **.zip** files that target different Altera development boards. On the web page, locate the Nios II Ethernet Standard Design Example **.zip** file that corresponds to your board. For example, the correct file for the NEEK is **niosii-ethernet-standard-3c25.zip**.

For the initial hardware design, you need not use a design example from the Nios II Ethernet Standard Design Example web page. You can use any SOPC Builder hardware design with a Nios II processor that is NicheStack capable. The design example walkthrough provides the information to add a component to be tested with the software application. The software application works correctly with any qualifying initial hardware design with the addition of the new component under test.

The design example **.zip** file, described in "Application Files", contains prebuilt versions of the Quartus II project and hardware image file that you generate in this walkthrough starting from the NEEK version of the Nios II Ethernet Standard Design Example.

To make the hardware design files available for the design example walkthough, download and unzip the file in a new working directory, <*Nios_II_Ethernet_Standard*>. Make sure the path name has no spaces.

Application Files

The **AN624_Debugging_with_System_Console_over_TCPIP.zip** file you can download from the Debugging with System Console over TCP/IP Design Example web page contains the following design example files:

- Software program files and board support package (BSP) creation script for the software application
- Pre-generated Quartus II project files and an SRAM Object File (**.sof**) for a specific final hardware design

Downloading the Application Files

To make the application files available for the design example walkthrough, follow these steps:

1. Download and unzip the file in a new working directory, <*sctcp*>. Make sure the path name has no spaces.
2. Copy the <*sctcp*>**/software_examples** directory to your <*Nios_II_Ethernet_Standard*> directory.

Application File Descriptions

The application files provide a complete software application and BSP to communicate with System Console over NicheStack TCP/IP Networking Stack sockets. Irrespective of the hardware design you use, the software is located in the <*sctcp*>**/software_examples** directory. Figure 1 shows the **AN624_Debugging_with_System_Console_over_TCPIP.zip** directory structure.

Figure 1. Application Software Files Directory Structure

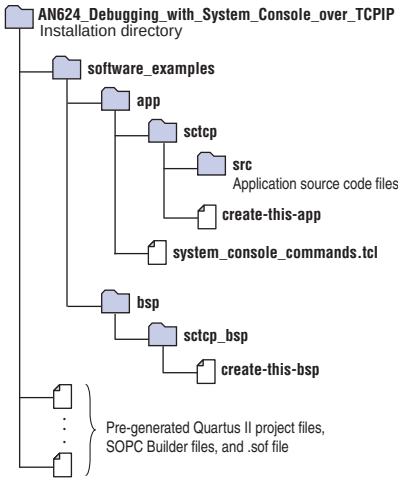


Table 1 lists the application source files located in the **src** directory in Figure 1.

Table 1. sctcp Application Source Files

File	Description
alt_2_wire.c	Contains utilities that provide a low-level interface to the EEPROM devices
alt_2_wire.h	Defines utilities that provide a low-level interface to the EEPROM devices
alt_eeprom.c	Contains utilities that read, write, dump, and fill the contents of the EEPROM devices
alt_eeprom.h	Defines utilities that read, write, dump, and fill the contents of the EEPROM devices
alt_error_handler.c	Contains three error handlers, one each for the Nios II `sctcp` application, NicheStack TCP/IP Stack, and MicroC/OS-II
alt_error_handler.h	Contains definitions and function prototypes for the three software component-specific error handlers
iniche_init.c	Defines `main()`, which initializes MicroC/OS-II and the NicheStack TCP/IP stack and processes the MAC and IP addresses; contains the PHY management tasks; defines function prototypes
sctcp.c	Defines the tasks and functions that utilize the NicheStack TCP/IP Stack sockets interface to interpret System Console commands received on a socket and compose responses to send back over the socket to System Console host tool
sctcp.h	Defines the task prototypes and task priorities used in the `sctcp` application
tcp_channel_master.c	Defines the `TCPChannelMaster` function, which processes commands coded in the Avalon Streaming Packet Protocol

The **alt_2_wire.c** and **.h** and the **alt_eeprom.c** and **.h** application source files implement the `alt_eeprom` software component, used for retrieval of the MAC address from EEPROM on the NEEK. These source files serve no other function for the `sctcp` application. The remaining files implement the `sctcp` application.

Design Example Construction

The following exercise shows you how to construct hardware and software with which the System Console can communicate using the Avalon-ST packet protocol to and from an SOPC Builder hardware component under test. This design example includes software to communicate with SOPC Builder hardware components through the System Console over NicheStack TCP/IP Networking Stack sockets. This application note uses the Nios II Ethernet Standard Example Design for the NEEK as the reference starting point.

Additionally, the <*sctcp*> directory provides a completed reference design for the NEEK. If you have a NEEK, you can skip the hardware system creation stage, and use this hardware design.

Creating the Hardware System

In the following steps you create a system capable of communicating with System Console over TCP/IP, with an on-chip memory used to represent any SOPC Builder component IP hardware under test, by starting with the Nios II Ethernet Standard Design Example hardware design.

Getting Started with Creation of the Example Design

To begin building the hardware system, follow these steps:

1. Change directory to <*Nios_II_Ethernet_Standard*>.
2. Open the Quartus II software.
3. On the File menu, click **Open Project** (not **Open**).
4. Browse and load the Quartus II Project File (**.qpf**) from the newly-created directory.
5. On the Tools menu, click **SOPC Builder**.

Adding an On-Chip RAM Memory

In this section, you add an on-chip memory to the system. This memory component is representative of any Avalon-MM SOPC Builder component that you wish to test with the System Console.

To add an on-chip memory, perform the following steps:

1. In the Component Library, expand **Memories and Memory Controllers**, expand **On-Chip**, and then click **On-Chip Memory (RAM or ROM)**.
2. Click **Add**. The On-Chip Memory (RAM or ROM) wizard interface appears.

3. Under **Memory type**, perform the following parameter value selections:
   - Select **RAM (Writable)**
   - Turn off **Dual-port access**
   - Set **Block type** to **Auto**
   - Turn on **Initialize memory content**
4. For **Data width**, select **32**.
5. In the **Total memory size** box, type `4096` and select **Bytes** to specify a memory size of 4 KBytes.
6. Under **Read latency**, for **Slave s1**, select **1**.
7. Click **Finish**. You return to the SOPC Builder **System Contents** tab, and an instance of the on-chip memory named **onchip_memory2_0** now appears at the bottom of the System Contents description.
8. In the **Clock** column, double-click and select **sdram_sysclk**.
9. In the **Base** column, double-click and type `0x00001000`.
10. Connect the slave port for **onchip_memory2_0** to **cpu/data_master**.
11. Ensure no other connections target the slave port for **onchip_memory2_0**.

The modified Nios II Ethernet Standard Design Example hardware design is also available in the **AN624_Debugging_with_System_Console_over_TCPIP.zip** file. You can compare your completed system to the predefined system located in <*sctcp*>**.**

Generating and Compiling the Hardware System Design

In this section, you generate HDL for the SOPC Builder system, and then compile the project in the Quartus II software to produce an SRAM Object File (**.sof**) for programming the FPGA.

To generate and compile the system, perform the following steps:

1. In SOPC Builder, click the **System Generation** tab.
2. Turn off **Simulation. Create project simulator files**. System generation requires less time when this option is off.
3. Click **Generate**. This might take a few moments. A **Stop** button replaces the **Generate** button, indicating generation is taking place.

   When generation is complete, the **Generate** button replaces the **Stop** button, and a **SUCCESS: SYSTEM GENERATION COMPLETED** message displays.
4. After generation completes, click **Exit** in SOPC Builder to return to the Quartus II software.
5. On the Processing menu, click **Start Compilation** to compile the project in the Quartus II software.
6. When compilation completes and displays the **Full compilation was successful** message box, click **OK**.
7. On the Tools menu, click **Programmer**.

8. Turn on the **Program/Configure** checkbox for the **.sof** file in the Quartus II Programmer.
9. Click **Start** to download the FPGA configuration data to your target hardware.

Creating Software for the System

In the following steps, you build an application and a BSP at the command line in a Nios II Command Shell. If any errors require debugging at run-time, you can use the Nios II Software Build Tools for Eclipse to import the application and BSP projects, and debug the software interaction with the hardware.

To build the application and BSP projects for this application note, follow these steps:

1. Start a Nios II Command Shell.
2. Change directory to the hardware design directory that contains your Quartus II project file. If you use the prebuilt hardware design that accompanies this application note, the directory is <*sctcp*>. If you construct the hardware from the Nios II Ethernet Standard design, the directory is <*Nios_II_Ethernet_Standard*>.
3. Change directory to **software_examples/app/sctcp**.
4. Build the project by typing the following command:
   `./create-this-app` ↵

The `sctcp` application software project and BSP project are created and built, and the software image file **sctcp.elf** appears in the directory.

After you make any hardware design changes in SOPC Builder, you can most easily regenerate the application and BSP by performing the following steps:

1. Delete all folders and files except the **create-this-bsp** script from **software_examples/bsp/sctcp_bsp**.
2. Delete the file **software_examples/app/sctcp/Makefile**.
3. Repeat the preceding set of steps to build the application and BSP projects.

Using System Console with the TCP/IP Channel

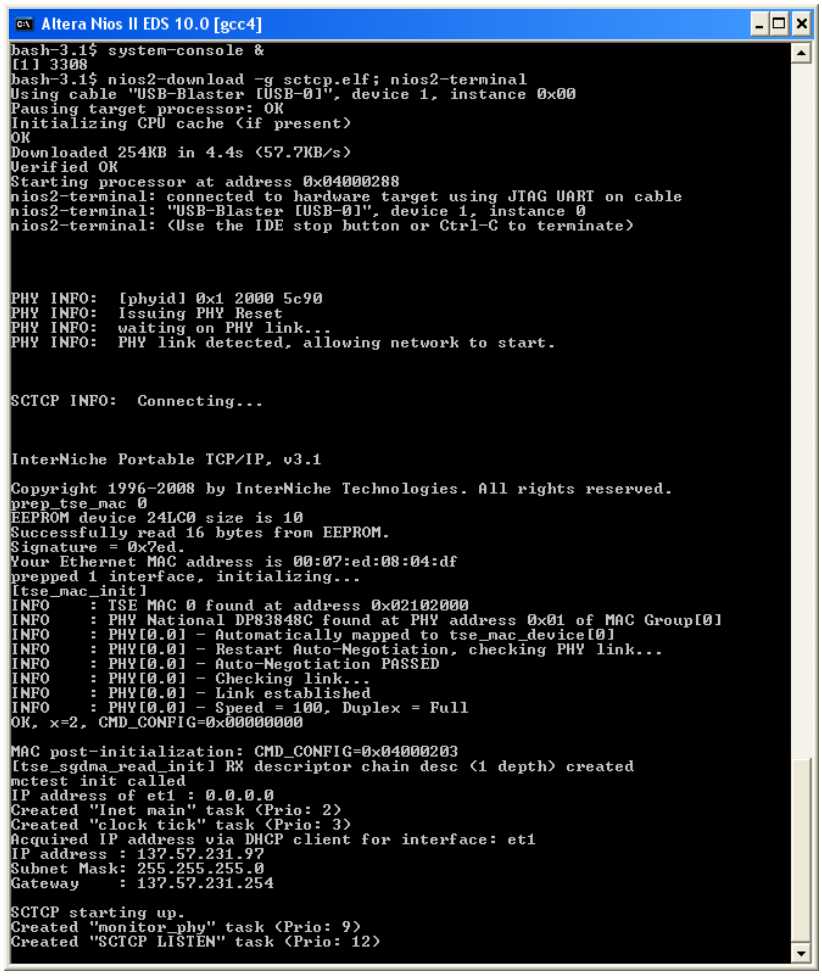
To use the System Console with the `sctcp` application, follow these steps:

1. To start the System Console, from a Nios II Command Shell, type the following command:
   `system-console &` ↵
2. To start the `sctcp` application, follow these steps:
   a. In a Nios II Command Shell, change directory to the application build directory, <*sctcp*>**/software_examples/app/sctcp**.
   b. Type the following command:
      `nios2-download -g sctcp.elf; nios2-terminal` ↵

      Note the IP address displayed in the nios2-terminal startup messages.

Figure 2 shows an example display following the download of **sctcp.elf**.

Figure 2. sctcp.elf Download Display



3. Use the `add_service` command in the System Console to add a tcp_master service at port 30 at the nios2-terminal IP address. For example, if the IP address displayed in the nios2-terminal startup messages is 137.57.235.55, in the system console, type the following command:
   `add_service tcp_master my_service 137.57.235.55 30` ↵

   The System Console responds with the following path for the new service:
   `/connections/tcp_master/my_service`

4. In the System Console, set a variable <*tcp_master_path*> to the path to the new tcp_master, by typing the following command:
   `set tcp_master_path [ lindex [ get_service_paths master ] 0 ]` ↵

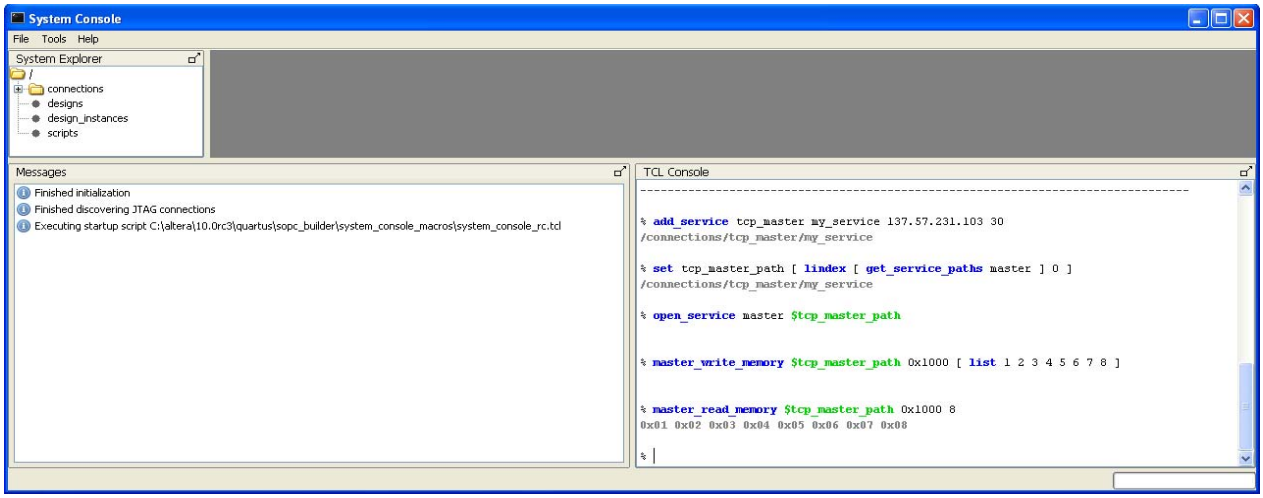
5. Open the tcp_master service using the <*tcp_master_path*> variable, by typing the following command:
   `open_service master $tcp_master_path ↵`
6. Use the tcp_master service to write values to the on-chip memory component, by typing the following command:
   `master_write_memory $tcp_master_path 0x1000 [ list 1 2 3 4 5 6 7 8 ] ↵`

   The command references the component by its base address. In Step 9 on page 7, you set the base address of the on-chip memory component to 0x1000.
7. Use the tcp_master service to read back those values, by typing the following command:
   `master_read_memory $tcp_master_path 0x1000 8 ↵`

   The System Console responds with the following values read from memory:
   `0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08`

Figure 3 shows the sequence of commands in the System Console.

Figure 3. System Console GUI with Commands Using the tcp_master Service



System Console TCP/IP Channel and sctcp Design Notes

The amount of data that can be exchanged with a single System Console command is limited by the amount of Java heap space available to the System Console on the development host. To avoid exhausting the Java heap, ensure that the size of each transaction is less than 100,000 bytes.

Most of the processing time is used for NicheStack TCP/IP stack packet processing. Many different optimization techniques are available to increase the system throughput. The Nios II processor runs at 50 MHz in this design, but it could run much faster, depending on the other requirements of your custom SOPC Builder system. The NicheStack networking stack has a zero-copy interface, which you can employ for greater efficiency. Other techniques for boosting performance include using an Altera Vectored Interrupt Controller (VIC), and locating interrupt service routines in single clock-cycle latency tightly coupled on-chip memory.

For information about accelerating a Nios II-based hardware system running the NicheStack TCP/IP Networking Stack, refer to *AN 440: Accelerating Nios II Networking Applications*.

You can replace the entire hardware system with a different processor and Ethernet hardware, if you adapt the `sctcp` software for the new hardware system. Specifically, the software that interprets the System Console packet-protocol commands received on a socket must run correctly with a TCP/IP stack for the new processor and Ethernet hardware.

For details on the packet protocol that the System Console uses, refer to the *Avalon-ST Bytes to Packets and Packets to Bytes Converter Cores* chapter and the *SPI Slave/JTAG to Avalon Master Bridge Cores* chapter in the *Embedded Peripherals IP User Guide*. Both of these chapters include a figure with an example that illustrates the relevant conversion.

Conclusion

In this application note you built software to support the System Console in communicating through a TCP/IP channel to an Avalon slave component, **onchip_memory2_0**, that you added to the Nios II Ethernet Standard Design Example SOPC Builder system. The software extracts System Console-generated read and write commands, encapsulated in an Avalon-ST packet protocol, from a socket. To test your own hardware, you can replace the on-chip RAM memory **onchip_memory2_0** in the hardware design with your own SOPC Builder hardware component.

Document Revision History

Table 2 shows the revision history for this document.

Table 2. Document Revision History

Date	Version	Changes
August 2010	1.0	Initial release.