Building 3c120 Simple Socket Server Plus

Last Updated

May 26, 2009

The Simple Socket Server Plus (SSS Plus) example is an enhanced version of the Simple Socket Server (SSS) example that Altera has shipped for the past few years. The primary enhancements to this example are as follows:

- Enabled the InterNiche FTP server.
- Enabled the InterNiche TFTP server.
- Enabled the InterNiche FTP Client.
- Enabled the InterNiche TFTP Client.
- Enabled the InterNiche Telnet server.
- Enabled the InterNiche console.
- Enabled the InterNiche VFS file system.
- Enabled the Altera ROZIPFS file system.

These are all basic capabilities that are available in the InterNiche TCP/IP stack as it ships with the Altera Nios II development environment.

This example is built on the Software Build Tools flow as described in the Nios II Software Developers Handbook. The software project directory contains an app subdirectory and a bsp subdirectory, each of which contain a shell script that creates the project makefiles and bsp settings, etc. To create and build the project you should refer to the “readme.txt” file located in the “app” subdirectory of the software project directory. The “readme.txt” file will provide you with the specific command line to execute to build the application.

There are substantial modifications that this example employs from the default Simple Socket Server application and associated default uCOS-II BSP. These differences can be seen in the BSP configuration, BSP patches, and application sources.

BSP configuration changes

If you look at the “create-this-bsp” script for this application, you can see that we are building a “ucosii” aware BSP with the “altera_iniche” and “altera_ro_zipfs” software packages enabled.

The rozipfs package is configured in pretty typical fashion with the rozipfs expected to be located at offset 0x0 of the CFI flash. The hardware design that this is ported to has allocated the lower 40MB of the CFI flash for user data, so we have plenty of room to store an rozipfs file there.

The iniche package is configured to NOT use the dhcp client for IP address acquisition. The software application statically assigns an IP address and MAC address in the application which are assumed to be used by default. If you wish to use the DHCP client service for IP address acquisition, you should change this setting and rebuild the BSP.

The sysclk and timestamp timer services are assigned to timers in pretty standard fashion.

The ucosii environment is expanded to allow up to 30 tasks, instead of the 20 tasks that SSS would request.

The compiler optimization level of the BSP is set to –O2.

The program and data sections for the application are all configured to run out of the low latency DDR RAM that is provided in the hardware system.

BSP Patches

If you look in the “bsp_patches” directory you will see a number of patches that are applied to the default BSP files prior to compiling the library. You can see this process driven by the “create-this-bsp” script which invokes the “apply_patches.sh” script in the “bsp_patches” directory. These patches accomplish the following results:

altera_avalon_tse.c.patch – this patch makes a call to the Marvell PHY configuration routine that activates an internal phase delay on the TX and RX clocks of the PHY device which is a requirement of the RGMII hardware interface that is running in the 3C120 FPGA design.

brdutils.c.patch – fixes a race condition in the irq_Mask() function of the InterNiche stack.

ipport.h.patch – the changes to this file are essentially to enable all of the extra services that this example wishes to demonstrate. You should make changes to the patched version of ipport.h if you wish to modify any of the services that this example has configured.

nrmenus.c.patch – fixes a conditional compilation issue that arises due to the enabling of the in_menus option in ipport.h

tftpcli.c.patch – fixes an issue where the InterNiche TFTP client does not properly form the file name field that is passed during a file transfer request.

timeouts.c.patch – this patch essentially wakes up the InterNiche FTP client more frequently than the one second interval that is used by default. This significantly improves the file transfer performance of the FTP client.

tk_crnos.c.patch – this patch corrects two issues, one surrounding the way that wakeup events are implemented for calls to select() which resulted in only a couple of tasks pending on select() calls to be awakened. The second issue is with the way tk_yield() was implemented, this is simplified and more predictable than the original implementation.

vfsfiles.c.patch – this patch increases the minimum amount of memory that is allocated each time the VFS file system requests a block of memory. This substantially improves the performance of the TFTP and FTP clients and servers that use the VFS, at the expense of frivolous memory usage.

Application Sources

Please refer to the “create-this-app” script for details of the specific makefile settings that are directed by that script when generating the application makefiles. The application source files for Simple Socket Server Plus are derived from the original Simple Socket Server application; however these substantial changes have been applied:

network_utilities.c – the get_mac_addr() and get_ip_addr() functions that are required to provide implementation specific results when called by the InterNiche TCP/IP stack have been hard coded to return a MAC and IP address that are simply hard coded constants in the application source. All of the flash manipulation code that was part of the original SSS implementation has been commented out. For users who wish to make a deployable implementation out of this example, attention will need to be paid here and some deployable implementation of get_mac_addr() and get_ip_addr() will need to be implemented here. This file contains the hard coded MAC address settings.

simple_socket_server.h – this file contains the hard coded IP address settings.

displays.c – this file has been added to the project to provide access to the displays that are available on the 3C120 development board.

demo_control.c – this file has been added to the project to provide the additional commands that are accessed thru the InterNiche menu system.

led.c – has been modified to combine the LED and seven segment functionality onto the one set of LEDs available on the 3C120 board. This makes the intended operation of the original SSS functionality a bit clumsy, but workable.

lrozipfs.zip and srozipfs.zip – are provided with the project sources as example ROZIPFS file systems that can be programmed into flash using this application. The lrozipfs.zip file provides a lager multi level hierarchy directory structure in it’s file system, whereas the srozipfs.zip file provides a simple small flat directory structure in it’s file system.