G9 Web Server App Note Appendum

Andrew Maier (amaier@ualberta.ca)
Sila Luckanachai (luckanac@ualberta.ca)

Follow the application note written by Group 8 Alex Newcomb (2012 winter term).
http://www.ece.ualberta.ca/~elliott/ece492/appnotes/2012w/Webserver/

With respect to his app note appendum, the NiosII fast type processor is the cpu match for a reliable and fast ethernet connection. The best configuration (given by Alex) is with 8KB Instruction and Data Caches, 32 byte data cache line size and Burst Mode enabled for both Instruction and Data Caches.

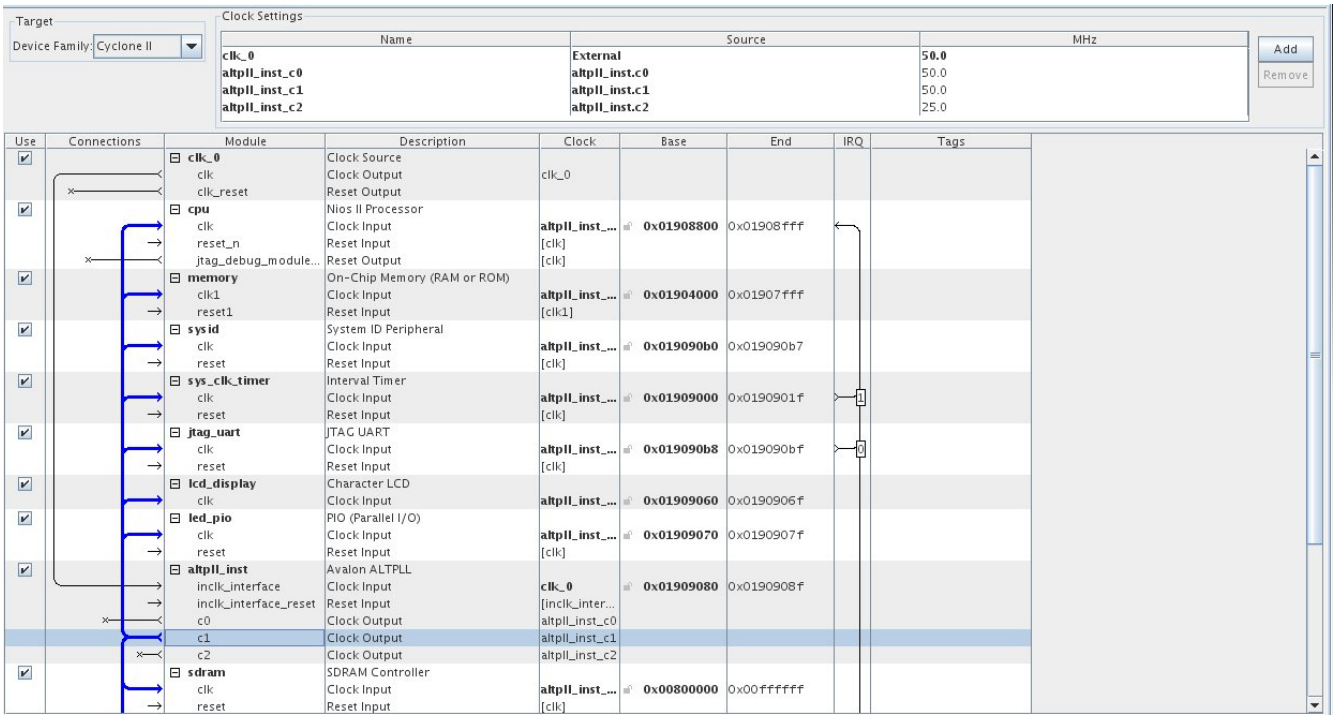
To compile the system with the fast type processor
Make the above noted changes to the processor type.

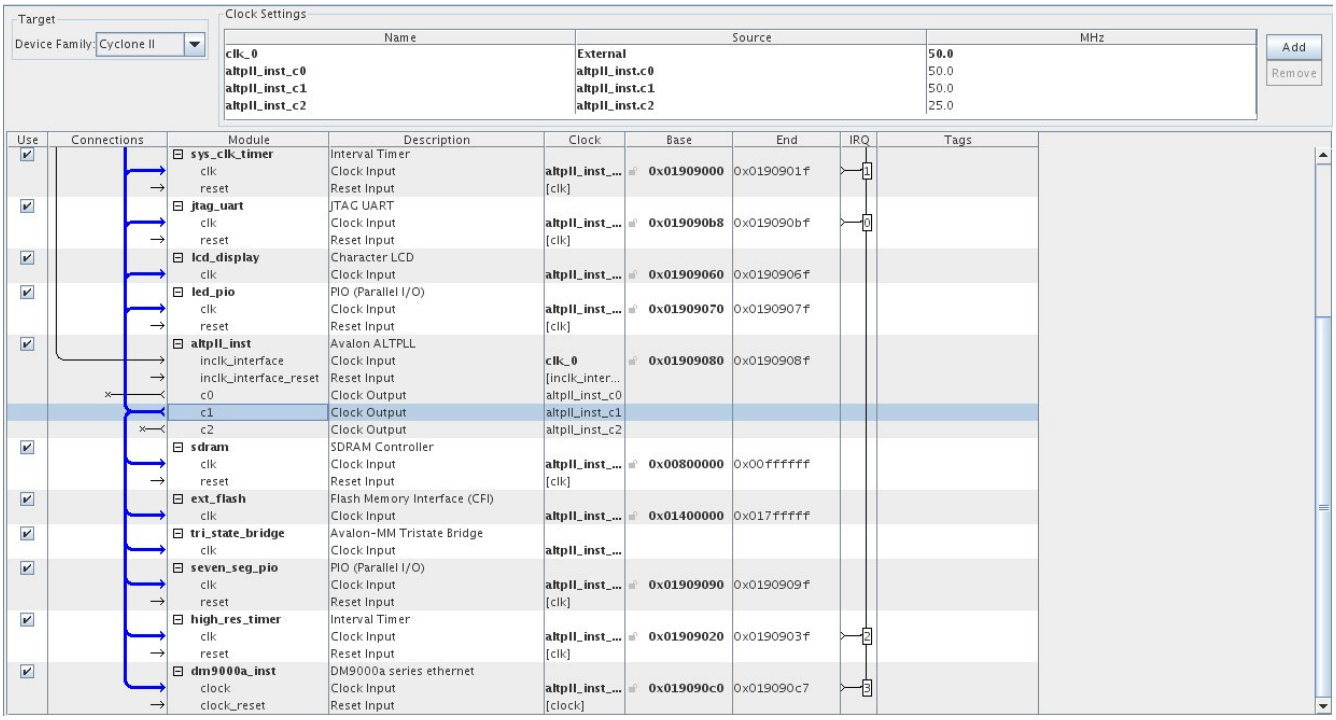
Note there is a typo in his initial web server application note, the dm9000a component should be called **dm9000a_inst NOT dm900a_inst (this will cause a compile error with the the software interface later).

In order to ensure synchronization of components, you must use a pll to provide a clock for all components. Recall that with your **altpll_inst** component there are three output clocks: c0, c1, c2. C0 outputs a 50 MHz -3.00ns phase shifted clock for the SDRAM, C2 outputs a 25 MHz clock which will be used to drive the Ethernet, however C1 is an unused 50MHz clock output.

In SOPC Builder you will need to connect every component to this C1 clock.

To do this right-click in SOPC builder on any component and click on “Clocks”. This will change your display and show how the clocks are connected. Connect the clocks like the following screenshot:





Now generate the system. After it is complete, back in Quartus you will want to add all of the files to your project MAKE SURE THAT YOU INCLUDE THE **dm9000a.vhd, dm9000a_inst.vhd, and dm9000a.hw.tcl**.

If you already have your top level created you can try and compile the system, however the provided example from the previous application note did not correctly map their clocks. You may use my provided top level example to get it running **webserver_top.vhd** .

You can now import the pin assignments and compile the system.

**Note if you get a critical warning stating that it could not find a “webserver.sdc” or “cpu.sdc” file this is because it is not always generated by default with the fast type processor configuration. Your program should be able to work by ignoring this warning, however I used the default .sdc file that is generated with the economy processor. You can do this by changing the processor type back to the economy type, re-generating, saving the .sdc file, and importing it into your fast type processor project. If you want to optimize your implementation you can also create your own timing constraints file through TimeQuest.

You can now continue with the original application note to create the software.

**Note that the first time you run the task you may be prompted to enter the serial number of the board (9 digit number). Just enter any 9 digit number, “123456789” works just fine. This will happen any time that you erase the flash on the board.

How to write the web server to flash

YOU MUST HAVE RUN THE WEB SERVER ON THE BOARD PREVIOUSLY ON THE RAM OF THE BOARD SUCCESSFULLY TO DO THIS. BECAUSE THE ETHERNET DRIVER REQUIRES A 9 DIGIT NUMBER INPUT, IF YOU HAVEN'T SUCCESSFULLY RAN THE WEB SERVER TASK PREVIOUSLY **IT WILL NOT RUN ON FLASH.

Do the same system compilation and generation as described in the previous application note, the appendum, and the above listed steps. **Except** you will need to change the cpu reset vector.

Right-Click on the cpu component in SOPC and click edit. Then change the “Reset Vector” to “ext_flash” (as with any flash project).

Generate the system and go back to Quartus upon completion.

Compile the project in Quartus.

Turn your board OFF and flick the switch from “RUN” to “PROG” (located just to the left of the lcd).

Turn the board back ON and run the “./scripts/reconnect_jtag.sh” script.

Open the programmer from Quartus and change the mode from “JTAG” to “Active Serial Programming”. Note that these steps are also outlined in more detail in the document provided by Nancy Minderman.

Now click on “Add File...” and select the .pof file in your directory (likely “webserver.pof”).

Check the “Program/Configure” section and click “Start”.

Now you can turn off the board, flick the switch from “PROG” back to “RUN”

Once complete, open up the “Nios II Software Build Tools for Eclipse” IDE.

Follow the original application note for details on how to create a new project.

Once you have your webserver project in the IDE you will need to program the ro_zipfs.zip file onto the flash of the board. The previous application note will tell you to program this at an offset of 0x100000 however this may be overwritten by your code when it is programmed. Therefore you need to choose an offset that will allow sufficient room for your code; we know that the code will be written to the base address of the flash without offset. I chose 0x200000 for the demo web server code.

Open up the Flash programmer Nios II -> Flash Programmer

File -> New... and then provide the .sopc file when prompted. This will provide the flash with the addresses and names of the components.

Click Add... on the right and find the ro_zipfs.zip (in the project “system” folder) and select it.

Change the offset to “0x200000” and click Start.

Click Exit.

Now we need to edit the BSP before we generate it, this was outlined in the previous application note as well.

Right-click in the project folder ending in “_bsp”.

Click Nios II -> BSP Editor...

Then click on “Software Packages” and change the ro_zipfs_offset to “0x200000” to match what where we flashed the archive.

Ensure that the correct base is given to your flash in ro_zipfs_base. Mine is 0x1400000. Click Generate and exit.

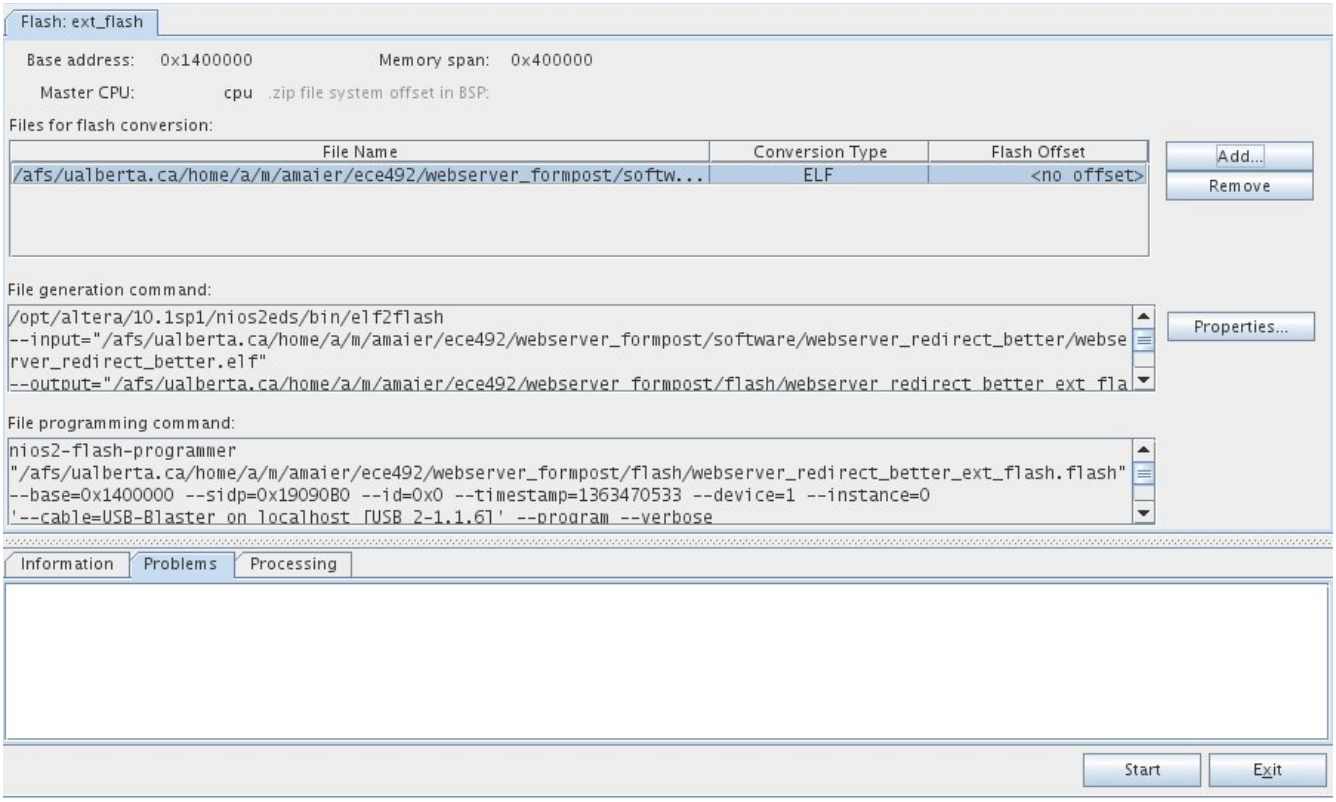
Now you will want to clean both of your projects. And then hit ctrl-b to build all projects.

Once it's built you want to flash the executable code onto your flash.

Click Nios II -> Flash Programmer

Click File -> New... and provide the .sopc file just like we did with the ro_zipfs.zip file.

Click Add... on the right and search for a file with the extension “.elf”. It will be in your software project folder and not the bsp project. See the screenshot:



Then click “Start”. Your code will now be written to the flash without an offset (ie at the base location of your flash). This corresponds with the reset vector address that we set in the SOPC builder.

When the flashing has completed you now need to restart the board. As long as you have previously entered the 9 digit serial number (or “123456789”) and did not erase the flash, your web server should hopefully be running successfully now.

Note that you cannot use the Jtag port for debugging now, however you can set the stdout to be the lcd display on the board.

If you have any questions, feel free to send me an email at amaier@ualberta.ca.