IMPLEMENTING A SPI DRIVER IN UCLINUX

FOR CAL POLY SUPER PROJECT

By

Matt Staniszewski

Senior Project

COMPUTER ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2009

Table of Contents

List of Figures

Acknowledgments

I would like to first thank Dr. Harris for getting me interested in the SuPER project and helping me understand the importance of both sustainable energy and the need to bring affordable energy to everyone. It has been great working with everyone on the SuPER team who are just as dedicated. Also, I'd like to thank my fellow computer engineers Khanh Nguyen and Frank Scarfo for being a great team and letting me bounce ideas off of them at our meetings. Your suggestions and ideas helped me greatly in advancing the SPI driver development part of our project. To Dr. John Williams, Michal Simek, Carsten Bartsch, and all the other contributors to the microblaze-uclinux mailing list, thank you for your guidance and help. Your hints, tips, and examples helped me overcome all the hurdles in developing the SPI driver and I thank you. I hope that my comments and updates on the mailing list will be helpful in developing future releases of uClinux as well as helpful to those seeking to implement a SPI driver using Microblaze. Finally, thanks Tom Hickok for your vast knowledge of Microblaze and Xilinx. Thanks for your help implementing components on the Spartan 3E Starter Kit in the EDK as well as your design ideas for an IP core that helped me with the last hurdle in my project.

I. Introduction

The Sustainable Power for Electronic Resources (SuPER) project is aimed at bringing affordable and sustainable energy to the third world. Currently, 2 billion people in the world do not have electricity (US Department of Energy). However, the costs and emissions of a traditional grid system can be very expensive to implement, especially in the third world, where no infrastructure exists. An alternate approach would be to set up small localized grids in different areas. This idea has already been applied to phone systems. Instead of setting up a traditional phone network, cell phone towers are deployed in the third world. This allows people to communicate with phones while eliminating the need to the infrastructure. The SuPER project aims at applying the same local grid idea for power in the third world. [1] A SuPER module would operate a micro grid. A micro grid, as the name suggests, is a much smaller grid, such as the size of a small village in Africa. Using the SuPER as the main power source in this grid, an entire village could be powered using power from renewable sources, such as solar and wind energy.

Currently, the initial SuPER system is operational and the next phase of development is in progress. Part of the next phase includes replacing the laptop that controls the inputs and outputs to the sensors and controls to the SuPER system. It currently consumes around 60W of instantaneous power. To lower power consumption to under 3W, a Spartan3E FPGA will be used to control the system and read inputs and outputs. Brian Estrada and Patrick Mariano have already developed and tested a working Linux system known as uClinux on the Spartan3E. [2] However, while the base Linux system is running on the FPGA, there currently is no driver to control the SPI devices such as the digital-to-analog converter (DAC) and analog-to-digital converter (ADC). These devices are essential to reading the sensor input and sending out the control signals and PWM output voltage to the SuPER system.

This paper focuses on the next iteration of development on using uClinux on the Spartan3E Starter Kit with a focus on the development of a SPI driver. It first gives an overview of the project and the objectives to be met. Next, it discusses design using Xilnx's ISE and EDK software. Then, the paper describes setup of the development environment as well as setup and usage of the SPI driver in uClinux. After, it discusses the implementation of the DAC and ADC in uClinux. Finally, the testing applications used for this project are presented as well as recommendations for future development. The appendices contain references, the SPIDEV kernel patch, a sample UCF File, and the SPI user application for Petalinux.

Naming Conventions

In this paper, several terms are used to represent the same object or concept. Although Petalinux is a specific version of uClinux, the terms 'uClinux' and 'Petalinux' are synonymous with each other in this paper. Also, the Spartan 3E Starter Kit is given a variety of names throughout this paper. Some examples include 'Spartan 3E Starter Kit', 'Spartan 3E FPGA', 'Spartan 3E', 'FPGA', and 'board'. Please bear in mind that all these terms describe the exact same board that is being used to implement the SuPER control system.

II. Background

Currently, the SuPER system uses a laptop to gather sensor data from various inputs and send out the PWM output to the PIC microcontroller. However, the laptop consumes a large amount of power (60W instantaneous). An alternative solution is to use a Spartan 3E FPGA implemented on a Spartan 3E Starter Kit to read the sensor inputs and send out output signals to the MOSFET switches as well as the PWM output signal. This would eliminate the need for the PIC and National Semiconductor devices currently in use on the SuPER system as well as reduce power consumption to less than 3W using the Spartan 3E FPGA. To do so requires an operating system to be installed on the FPGA to operate the SuPER system. For this project, a variant of uClinux called Petalinux has been chosen to develop the user environment to operate the next phase of the SuPER project.

In their paper, Brian and Patrick [2] discuss the installation and development of both development environment for Petalinux as well as configuration and installation of the Petalinux kernel on a Spartan 3E Starter Kit. This paper provides an initial road map for starting to use uClinux on the Spartan 3E for the purposes of the SuPER project.

Once the Petalinux operating system was implemented on the starter kit board, the next iteration of this project will focus on implementing the necessary devices in the Petalinux environment. Several drivers are already available in Petalinux to access many of the devices that can be added to the Spartan 3E Start Kit in the Xilinx EDK, such as General Purpose IO (GPIO) devices. However, to use this FPGA with the SuPER project, a SPI driver is needed. This driver would allow several important devices necessary for the SuPER to work correctly in uClinux. First, the digital-to-analog converter (DAC), which would be used to send out an analog signal to the MOSFET switches as well as the PWM control signal. Second, the analog-to-digital converter (ADC) would be used to take analog inputs from the sensors on the SuPER system and convert them to digital signals that could be saved for logging purposes. Alex Diaz began work on this next iteration and discusses the use of SPI with Petalinux in his paper. [3] However, he was unable to find a driver that worked correctly in Petalinux, which proved to be quite difficult due to the lack of information regarding SPI driver development in uClinux.

III. Requirements

This project will develop the Petalinux development environment further as well as a SPI driver that will interface SPI devices using Petalinux.

First, the project will focus on the finding the best environment to develop hardware and software for Petalinux. This includes previous approaches discussed in both Brian and Patrick's [2] and Alex's papers [3]. The optimal operating system environment will be described as well as setup and installation of Xilinx EDK and ISE, as well as the other tools necessary to develop the Petalinux environment. Faster methods of file transfer to the Spartan 3E Start Kit will also be examined, such as USB and Ethernet. The optimal interface will be chosen for its ability to transfer the uClinux kernel files over to the board at the fastest rate.

Next, this project will develop a SPI driver for uClinux. It will continue the work of Alex [3] in SPI driver development, considering his ideas as well as new ones developed by myself as well as suggestions from the uclinux-microblaze mailing list. The SPI driver will be implemented in the Petalinux kernel to allow a user to write an application to easily interface to a SPI device inside the uClinux environment. If possible, the code will be developed in such a way that it is as similar to software written in courses such as CPE 329 that interface with SPI devices without the use of Petalinux.

Finally, SPI devices necessary for the operation of the SuPER project will be implemented using the new driver. First, digital-to-analog converters (DACs) will be implemented that take digital values and send out analog outputs, such as the PWM control signal to the PIC. Second, analog-to-digital converters (ADCs) will be implemented that take analog inputs from sensors on the SuPER system and convert them to digital values that can be stored for logging data.

IV. Design

Xilinx EDK Software

The initial design work regarding Petalinux has been carried out and explained by Brian and Patrick in their paper [2]. Similarly, this project will use the Xilinx EDK software to develop the hardware and software necessary to implement a SPI driver in Petalinux. The SPI core will be implemented in the EDK using a SPI IP core. Please see the section entitled *SPI IP* Core in Alex's paper [3] for a design description regarding adding the SPI core in the Xilinx EDK software. However, unlike Alex's paper, this paper will leave the Ethernet IP core installed so that it may be used to speed up development in Petalinux by providing a faster transfer of the kernel image.

SPI User Application

To develop the user application, the C programming language is used. The application to interface with the SPI devices will be created in a similar fashion to that described in Brian and Patrick's paper [2]. This application will provide a user interface in Petalinux that will be used to access the various SPI devices that were implemented in the Xilinx EDK. This includes both the DACs and ADCs that are necessary for the SuPER project to function properly. The user application will provide the access to the SPI devices that the SuPER's software program will use once the code is ported from the laptop currently in the system to Petalinux on the Spartan 3E Starter Kit.

V. Setting up the Desktop Linux Environment

This section describes how to setup the Linux environment on your computer. In order to develop in uClinux, a system needs to be set up to communicate with the Spartan 3E Starter Kit Board. Previous papers have suggested using a virtual machine (VM) of Linux running on top of Windows. While this allows the user to continue to use Windows for development, it also causes several issues that can cause serious delays. As Alex suggests in the section entitled *Environment Setup* [3], running a VM can make serial operations slower. The VM buffers data before sending it to the host operating system port. This can cause delays between sending and receiving serial data. This slows down the process of loading a new kernel image on to the Spartan 3E dramatically. In some cases, such as with VirtualBox, the serial communication is delayed so much by the VM software that data is not sent in the correct order, which can corrupt file downloads to the Spartan 3E FPGA. Therefore, it is suggested that you use a standalone installation of Linux to develop in instead of a VM, whether it is on a dual-boot system or by itself. This eliminates the serial communication issues as well as gives a slight performance boost when downloading files to the Spartan 3E Starter Kit.

Installing Linux on your Computer

For this project, CentOS has been chosen as the best Linux operating system to develop Petalinux. This distribution of Linux is recommended and supported by Xilinx, and thus used as the main development platform. Install the current version of CentOS from their website at `http://www.centos.org`. At the time of the writing this paper, CentOS 5 was the most current version. More detailed instructions regarding the installation of CentOS as well as initial setup can be found on the SuPER wiki [4] and in Alex's paper [3].

Once CentOS is installed, be sure to install the libcurl and libmotif packages. These are required by the Xilinx ISE and EDK software. Otherwise, the programs may not open after installation is complete. After logging in, go to Add/Remove Software, search for libcurl and libmotif, and install the development libraries.

Disabling Security-Enabled Linux (SELinux)

By default, CentOS enables Security-Enabled Linux (SELinux), which enables additional security features in Linux as well as a firewall. However, SELinux causes issues with both Xilinx's software and file transfers over serial, USB, and Ethernet. To prevent these issues, disable SELinux on your CentOS installation.

In CentOS, go to the menu bar at the top and click System > Administration > Security Level and Firewall**.** Enter the root password at the prompt. In the Security Level Configuration Window, click on the SELinux tab and change the SELinux setting to Disabled. Apply the settings and restart your computer to allow the changes to take effect. If you still have trouble with transfers (especially over Ethernet), try disabling the firewall as well. This ensures that all protocols are sent through directly to your computer (you may also wish to try turning on specific ports). In CentOS, go to the menu bar at the top and click System > Administration > Security Level and Firewall. Enter the root password at the prompt. In the Security Level Configuration Window, click on the Firewall Options tab and change the Firewall setting to Disabled. Apply the settings to disable the firewall. With the firewall disabled, communication over the interface you are using (serial, USB, Ethernet) should work correctly now.

Installing Xilinx ISE and EDK

Please see the section in their paper entitled *Installing the Xilinx ISE & EDK on a Linux Workstation* in Brian and Patrick's paper [2] or the SuPER wiki [4] for more information regarding the installation of Xilinx ISE and EDK.

Once you have completed installing ISE and EDK, copy each program’s settings.sh file over to /etc/profile.d. This forces the Xilinx environment variables to be applied every time the computer starts up. This allows you to load the program without the need to source the settings.sh files every time.

Use the following commands to do this:

```
[root@localhost ise9.1]# cp settings.sh /etc/profile.d/ise_settings.sh
[root@localhost edk9.1]# cp settings.sh /etc/profile.d/edk_settings.sh
```

The next time you start the computer, your Xilinx environment variables will be automatically set.

Installing the USB Driver

Once you have installed and setup Xilinx ISE and EDK correctly, you must install the drivers to communicate with the Spartan 3E Starter Kit in order to download hardware bit files. Previous papers have discussed using Xilinx’s JTAG cable with Digilent Export to program the Spartan3E board. However, there is no Export for Linux. Instead, we must use another tool in Linux to program the board. There are a variety of open source tools available to program Spartan-based boards. This paper focuses on using the libusb driver in conjunction with Xilinx’s iMPACT program to download bit files onto the starter kit with a regular USB cable. More information about the USB driver installation can also be found on the SuPER wiki [4] as well as the libusb project website at `http://www.rmdir.de/~michael/xilinx/`.

To install the USB driver, obtain a copy of usb-driver-head.tar.gz from http://www.rmdir.de/~michael/xilinx/. Extract the files in the archive and follow the instructions in the included README to install the driver. Once the driver is compiled and installed, copy the usb-driver folder to the /opt/pkg/xilinx directory. Before running iMPACT, initialize the driver with the following command:

```
[root@localhost]# export LD_PRELOAD=/opt/pkg/xilinx/usb-driver/libusb-
driver.so
[root@localhost ~]# impact -port auto
```

To avoid having to run this command everytime you run iMPACT, copy the export command above into a file and save it as a bash script under /etc/profile.d:

```
#!/bin/bash
export LD_PRELOAD=/opt/pkg/xilinx/usb-driver/libusb-driver.so
```

Be sure to save the file with a .sh extension. Once iMPACT is started, you can program the Spartan 3E Starter Kit as follows. In the main window, go to File > Initialize Chain. An open dialog box will pop up for the first device on the Spartan 3E chain. This is the chip we wish to program. Select the .bit file by browsing to the appropriate directory the file is under, selecting the .bit file, and hitting 'Open'. Next, hit 'Bypass' twice to skip programming the other two devices on the board. Once you are ready to program, right-click on the xc3s500e device in the main window and hit 'Program'. Hit 'OK' in the Programming Properties window to program the board. If successful, a 'Programming Succeeded' message will appear. Otherwise, you will get a 'Programming Failed' message.

If iMPACT says the connection has failed, be sure to check that your cable is plugged in correctly and the board is powered on. To verify the cable is plugged in, open a console window, type 'sudo /sbin/lsusb' and hit enter. If the cable is connected and the Spartan 3E is working properly, an entry for the Xilinx device should appear such as below:

```
[user@localhost ~]$ sudo /sbin/lsusb
Bus 003 Device 001: ID 0000:0000
Bus 004 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 005 Device 001: ID 0000:0000
Bus 001 Device 017: ID 03fd:0008 Xilinx, Inc.
Bus 001 Device 001: ID 0000:0000
Bus 001 Device 002: ID 413c:a005 Dell Computer Corp. Internal 2.0 Hub
```

Once you have verified the board is connected correctly, try detecting the cable manually instead of using auto-detect. Sometimes, auto-detect will fail to run when using libusb. To do so, in iMPACT go to Output > Cable Setup. In the Cable Communication Setup window, click 'Xilinx USB Cable' as the Communication Mode and hit 'OK'. Try to initialize the chain once more in iMPACT. If this fails to work, try power cycling your board, reseating the USB cable, and then reopening iMPACT.

Downloading Petalinux

Petalinux is a distribution of uClinux that was created by Petalogix. For this paper, the latest testing or SVN release was used for the SPI driver development. Please remember that the words uClinux and Petalinux are used interchangeably throughout this paper.

To download the latest SVN build, follow the instructions on Petalinux's website at `http://devloper.petalogix.com`. After the file has been downloaded, extract it to a directory under your home directory. For this paper, the Petalinux directory was extracted to ~/petalinux-svn and this directory will be referred to throughout this paper. The Petalinux archive contains a lot of files, so the extraction process may take a long time.

Creating a Test Development EDK Project

While the Ethernet module makes updating the kernel faster, it is still a slow process to update Petalinux if you are just testing a small hardware change or software bug. Therefore, it is recommended to create a second EDK project based on the reference design. This design can be used to program the Spartan 3E without the need to recompile a kernel and boot into uClinux. That way, you can test design changes, such as adding a new SPI device or making a software change to your user application.

To do so, copy over the entire hardware reference design in the ~/petalinux-svn/hardware/reference-designs/Xilinx-Spartan3E500-RevD-edk91/ directory. Rename the folder to something else such as ‘test-dev’ so that it doesn’t conflict with the reference design. Open the project in EDK. It will open an exact copy of the reference design from Petalinux. To develop software, the Petalinux kernel and U-boot software information must first be removed. Go to the Applications tab, right-click on the fs-boot project, and hit 'Delete Project'. Add a new Software Project by double-clicking 'Add Software Application Project', typing a name in the 'Project Name' field and hitting 'OK'. Be sure to right-click on it and hit 'Mark to Initialize BRAMs', otherwise the project will not compile into the download.bit file. Add your headers and source files to the new project to develop your software code. If you do not at source files, you will receive an error about your .elf file during compile time. Once you are ready to test your code, compile your EDK design and download the .bit file to your Spartan 3E Starter Kit via iMPACT to test your design.

Converting from a Test EDK Project to Petalinux

When you are ready to test your code in Petalinux, copy the source and header files into a directory located under ~/petalinux-svn/software/user-apps and compile it as a user application. Please see the section entitled *Creating User Applications for uClinux* in Brian and Patrick's paper for information regarding this topic [2]. Be aware that you will need to convert the constants and some of the header files in your software code. Remove any of the following header references:

```
#include <xparameters.h>
#include <xio.h>
```

Add the following include to your files instead:

```
#include <linux/autoconf.h>
```

Rename all the Xilinx you used in your code with those in Petalinux. Most of these may be found under ~/petalinux-svn/software/linux-2.6.x-petalogix/include/linux/autoconf.h. The following examples are based on the code in Appendix D. For example, the constant `XPAR_OPB_SPI_0_BASEADDR` converts to `CONFIG_XILINX_SPI_0_BASEADDR` in Petalinux using the constant from autoconf.h. In some cases, there will not be an entry for constant in your code in autoconf.h. For these, use the hard coded hardware address for the IP core. These values can be found in the EDK under the System Assembly View when the Addresses Filter applied. For example, the constant `XPAR_FLASH_ADC_SEL_BASEADDR` converts to `0x42000000` in Petalinux. Once these changes are applied, you should be able to compile your project in Petalinux as a user application.

VI. Setting up uClinux on a Spartan 3E Board

This topic is discussed in depth in Brian and Patrick's paper [2]. Please see the section in their paper entitled *Installing uClinux on a Spartan 3E Starter Board*. This will give you a Spartan 3E Starter Kit with a booting base Petalinux system.

A Note on Kermit

Whenever you use Kermit, the following command must be run in a console window each time you boot your computer:

```
[user@localhost ~]$ sudo chmod a+rw /dev/ttyS0
```

This ensures that all users are given read and write privileges to the serial port. This allows Kermit to work properly. There may be an easier way to implement this using a script; I ran it once every time I started my system. Also note that you may be using a different serial port from ttyS0, so change the command above accordingly depending on your serial port.

VII. Setting up Ethernet File Transfer

By default, the Petalinux EDK reference design for the Spartan 3E Start Kit includes an Ethernet Lite IP core that is implemented in the hardware. Xilinx doesn't include the core as part of its suite of IP cores. However, the license file for the Ethernet Lite core can be downloaded for free from their website. Once this core is licensed, you can compile the hardware reference design successfully with the Ethernet hardware block included. If you wish to develop without the Ethernet core, please see the section entitled *Removing Ethernet IP Core (If license is expired)* in Alex's paper [3].

With the Ethernet Lite core implemented in Petalinux's reference design, you can configure the uClinux kernel to include Ethernet drivers. This will allow you to communicate over the built-in Ethernet port on the Spartan 3E. Specifically, you will be able to use the Trivial File Transfer Protocol (TFTP) to upload files to your board via Ethernet. Using TFTP, the kernel image can be downloaded in about ten seconds rather than ten minutes with a serial connection. This dramatically reduces the wait time while doing development and can be very useful.

Installing the Ethernet Core License

Before using TFTP, the Ethernet Core License must be installed first. The following section provides information regarding installation of the core license. For more information, please see the guide on the SuPER wiki [5]. What follows is a summary of that guide.

Go to the Ethernet Lite IP Core website on Xilinx's website at `http://www.xilinx.com/products/ipcenter/xps_ethernetlite.htm` and click on 'Access Lounge'. Log on to the site with your Xilinx user name and password. If you do not have a logon, register for free at Xilinx's website. Once you enter your user name and password, you will be logged into the Access Lounge site for the Ethernet Lite IP Core. Scroll down to the bottom of the page and select 'Generate the OPB 10/100 Ethernet MAC Lite License Key'. Fill out the form with the requested information. If you need help obtaining your Host ID, please follow the link on the form to the help site. Submit the form when you are finished filling it out. A confirmation page will appear telling you to check your email. If you do not receive one after several minutes, try clicking on 'Send me download instructions'**.** Once you receive the confirmation email, download the core_licenses_full.zip attachment and follow the instructions in the email to set up your license. The Ethernet Lite OPB Core should now be successfully installed. To test this, try to build the hardware for the Spartan 3E RevD reference design for Petalinux located under ~/petalinux-svn/hardware/reference-designs/hardware/reference-designs/Xilinx-Spartan3E500-RevD-edk91/system.xmp. If you receive errors such as the following, the IP core license is not installed correctly.<IMAGE> Please reread this section, as well as the guide from the SuPER wiki [5] for more information.

Configuring TFTP in CentOS

Once the core is successfully installed and tested on the computer, the next step is to configure CentOS as the TFTP server so that files can be uploaded to the FPGA.

On your CentOS computer, install TFTP by running the following command in the Console:

```
[root@localhost ~]# yum install tftp tftp-server
```

Install the software packages along with any dependencies it prompts to install. After TFTP is installed, open up /etc/xinetd.d/tftp with your favorite text editor. If the file doesn’t exit, create a new file with this name. Be sure to edit this file as root, either directly or using the ‘sudo’ command. Set up the file with the following information:

```
service tftp
{
   disable = no
   socket_type = dgram
   protocol = udp
   wait = yes
   user = root
   server = /usr/sbin/in.tftpd
   server_args = -s /tftpboot
   per_source = 11
   cps = 100 2
   flags = IPv4
}
```

Be sure to verify that all the entries are correct. The ‘user = root’ line is the same regardless of whether you are logged in as root or not. Once the file has been verified, restart the xinetd service by going to System > Administration > Services and entering the root password. In the Service Configuration window, scroll down, select the xinetd service and hit 'Restart'. Hit 'OK' to the success message and close the Service Configuration window.

Run the following commands to set up the tftpboot directory (if it hasn’t already been created) to use with TFTP:

```
[root@localhost ~]# mkdir /tftpboot
[root@localhost ~]# chown –R nobody:nobody /tftpboot
[root@localhost ~]# chmod –R 777 /tftpboot
```

You may also wish to chown the /tftpboot directory to your user name instead, otherwise you will have to use the ‘sudo’ command when running ‘make all’ under the ~/petalinux-svn/software/petalinux-dist directory.

To test TFTP, copy or create a file under /tftpboot and then try to download it locally to your computer using TFTP:

```
[root@localhost ~]# touch /tftpboot/some_file
[root@localhost ~]# tftp localhost
tftp> get some_file
tftp> quit
```

If successful, and run ‘ls’ to view your current directory. The file you placed in /tftpboot and downloaded with TFTP should be in the current working directory. If not, go back and verify the settings in the section above. Also, restart your computer to make sure all the settings have taken effect.

Configuring TFTP in Petalinux

After TFTP has been installed and test in CentOS, Petalinux must be configured to use Ethernet and TFTP for file transfer. By default, the Petalinux kernel is already set up to use and implement Ethernet, so no additional software settings are needed. Only the hardware reference design must be rerun with the core license installed.

Please note that while this method is much faster, serial is still used primarily for uploading the SREC file and monitoring the console output from the Spartan 3E board. The TFTP transfer is used for uploading the ub.config.img, u-boot-s.bin, and image.ub files, which take a long time to upload via serial connection.

After the hardware reference design has been recompiled, rerun ‘petalinux-copy-autoconfig’ under ~/petalinux-svn/hardware/reference-designs/hardware/reference-designs/Xilinx-Spartan3E500-RevD-edk91 and run ‘make all’ under the ~/petalinux-svn/software/petalinux-dist directory to recompile the kernel. Program the board with the new .bit file and upload the SREC file as per instructions in Brian and Patrick’s paper [2]. Once the SREC file is loaded, press any key in the Kermit serial console to stop U-boot from automatically booting. To use TFTP, configure the following environment variables to enable Ethernet:

```
U-Boot> setenv ipaddr <IP Address>
U-Boot> setenv serverip <Server IP>
U-Boot> saveenv
```

Where `<IP Address>` is a valid IP address on your network and `<Server IP>` is the IP of your CentOS workstation. Once these settings are saved, run the following to download the auto-configuration script for Petalinux:

```
U-Boot> tftp $(clobstart) ub.config.img
U-Boot> autoscr $(fileaddr)
```

The auto-configuration will update all the U-boot settings for Petalinux. Please note that this will also reset your IP settings. In the future, these IP settings should be written into the auto-configuration script. You may also wish to use DHCP to get an IP address rather than using static addresses. For more information regarding these topics, see Petalinux website regarding customizing U-Boot [6] and setting up DHCP [7].

Rerun the environment variable settings above again. From this point forward, these IP settings will be saved until the next time your load a new ub.config.img file. Load the u-boot-s.bin file next using the following commands:

```
U-Boot> tftp $(clobstart) u-boot-s.bin
U-Boot> protect off $(bootstart) +$(bootsize)
U-Boot> erase $(bootstart) +$(bootsize)
U-Boot> cp.b $(clobstart) $(bootstart) $(filesize)
```

Next, load the kernel image.ub file using the following commands:

```
U-Boot> tftp $(clobstart) image.ub
U-Boot> protect off $(kernstart) +$(kernsize)
U-Boot> erase $(kernstart) +$(kernsize)
U-Boot> cp.b $(clobstart) $(kernstart) $(filesize)
```

All of the files have now been successfully loaded into the flash on the Spartan 3E Starter Kit. Reset your board be downloading the download.bit file again. Petalinux should now successfully boot to the logon prompt as it did previously.

Whenever you make kernel changes or software changes, such as adding user applications, the image.ub file is the only file that needs to be uploaded again. For this, you can use TFTP, dramatically cutting down the development time for software in Petalinux. However, if you need to modify the hardware, then you must go through the entire process outlined in Brian and Patrick's paper [2] again and upload all the files again via TFTP. Remember, the IP environment variables will be reset once you run a new auto-configuration script.

VIII. Implementing the SPIDEV Driver in uClinux

The Petalinux kernel already includes SPI drivers that can be configured. These include a bit banging driver and SPI character driver. These drivers are quite old and do not work correctly with the Spartan 3E Starter Kit. Instead, another driver is needed to access the SPI bus configured on the FPGA.

The solution is a SPI driver known as SPIDEV. This driver implements each SPI bus interface as a block device rather than a character device, assigning each SPI interface a name /dev/spi-x where ‘x’ denotes the bus. For example, if you implement two SPI buses in your Xilinx EDK hardware design and use SPIDEV, you will use /dev/spi-0 and /dev/spi-1 in Petalinux to access devices on these buses.

Once the SPIDEV driver is properly set up and initialized, accessing SPI devices such as DACs and ADCs can be done using code written for a Xilinx EDK project, such as in CPE 329. For more information on interfacing with a specific SPI device, see the section entitled *Implementing SPI Devices in uClinux*.

Adding the SPI IP Core to Xilinx EDK

Please see the section entitled *SPI IP Core* in Alex’s paper [3] for information about adding the SPI core in EDK. Once added, please also verify that the OPB_Clk signal is connected to the sys_clk_s net, otherwise your SPI devices will not operate correctly. To view all signal connections, select the System Assembly View in EDK. Change Filters at the top to ‘Ports’ and then select Connection Filters > All to the right of that. Expand the port view of the

SPI module by click the plus (+) sign next to the SPI device. Check that the clock signal is correctly set up.

Once all bus, port, and address settings are connected correctly, configure the SPI module for the number of devices you wish to use. In the System Assembly View, under the Ports filter, right-click on the SPI module and hit ‘Configure IP’ to change the core’s settings. Change ‘Total Number of Slave Select Bits in SS Vector’ to the number of SPI devices you plan on having connected to this bus. For example, if you want to use the DAC and ADC, you would set this value to 2. Depending on the SPI device connected to the bus, you may also wish to set the ‘Ratio of OPB Clock Frequency to SCK Frequency’ setting. As the name implies, this is the ratio of $\frac{OPB\_Clk}{SCK}$ and is 32 by default. This means that if you’re using the default 50 MHz clock, the SCK frequency will be: $\frac{50MHz}{32} = 1.5625MHz$. The default setting is fine for the DAC and ADC, but if you connect a slower device to the SPI bus, such as a SD card module, you may need to increase this ratio to lower the SCK frequency. All other settings should be left the same in the SPI IP core properties window. Click ‘OK’ to close the window to save any changes and save the project. Recompile your EDK project to include the SPI IP core.

Installing SPIDEV in Petalinux

To use SPIDEV, the correct kernel files need to first be updated or patched to integrate the driver into the default uClinux kernel.

The latest SPI driver from Xilinx is used to implement the SPIDEV driver and must first be downloaded and copied over to your Petalinux directory. Xilinx uses git repositories to allow users to download the latest drivers for their devices. Install git and obtain the Xilinx Linux repository using the following commands:

```
[root@localhost ~]# yum install git
[root@localhost ~]# git clone git://git.xilinx.com/linux-2.6-xlnx.git
```

Once the Xilinx git repository has been downloaded, copy the following files into your Petalinux directory:

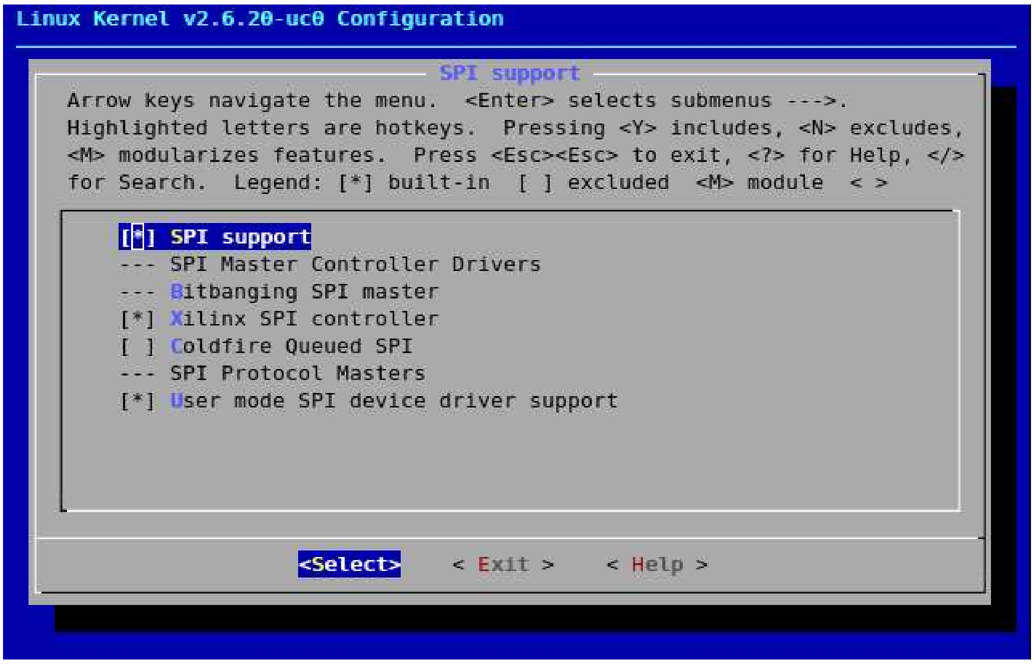
```
[root#localhost ~]# cp ~/linux-2.6-xlnx/drivers/spi/xilinx_spi.c ~/petalinux-
svn/software/linux-2.6.x-petalogix/drivers/spi/xilinx_spi.c
[root#localhost ~]# cp ~/linux-2.6-xlnx/drivers/spi/spidev.c ~/petalinux-
svn/software/linux-2.6.x-petalogix/drivers/spi/spidev.c
[root#localhost ~]# cp ~/linux-2.6-xlnx/include/linux/spi/spidev.h
~/petalinux-svn/software/linux-2.6.x-petalogix/include/linux/spi/spidev.h
```

After you have copied the correct files from the Xilinx git directory, the SPIDEV patch needs to be applied to Petalinux. Please see Appendix B for the patch file. Take note of the areas in bold within the patch. These are to emphasize specific sections of the patch that need to be configured. First, be sure to check the 'CONFIG_XILINX_SPI...' entries in the script against the constants in the generated ~/petalinux-svn/software/linux-2.6.x-petalogix/include/linux/autoconf.h file. Second, update the SPI board info initdata entries for the setup.c file with the SPI settings for your SPI bus, such as speed and bus number. Comment out any of the dummy structures in this section that you are not using. Finally, I'd recommend adding some print statements to the ~/petalinux-svn/software/linux-2.6.x-petalogix/drivers/spi/spidev.c file for debugging purposes. These will show up during the Petalinux boot process and make it easy to tell if your SPI device is setup correctly. The kernel uses the 'printk' command instead of a 'printf', but the formatting is exactly the same. I'd add the printk's to the spidev_probe() function specifically, since this is the function that initializes the device. Once you have modified the patch, install it (patch.diff) by copying it into your base petalinux directory and running the following command:

```
[root#localhost ~/petalinux-svn]# patch –p1 < patch.diff
```

All hunks that are patched should display ‘succeeded’ messages. If you see a ‘FAILED’ message, the failed portion of the patch will be saved into a file with the same name as the file to be patched with a .rej file extension appended. If this occurs, apply the changes in the .rej file manually in the Petalinux kernel.

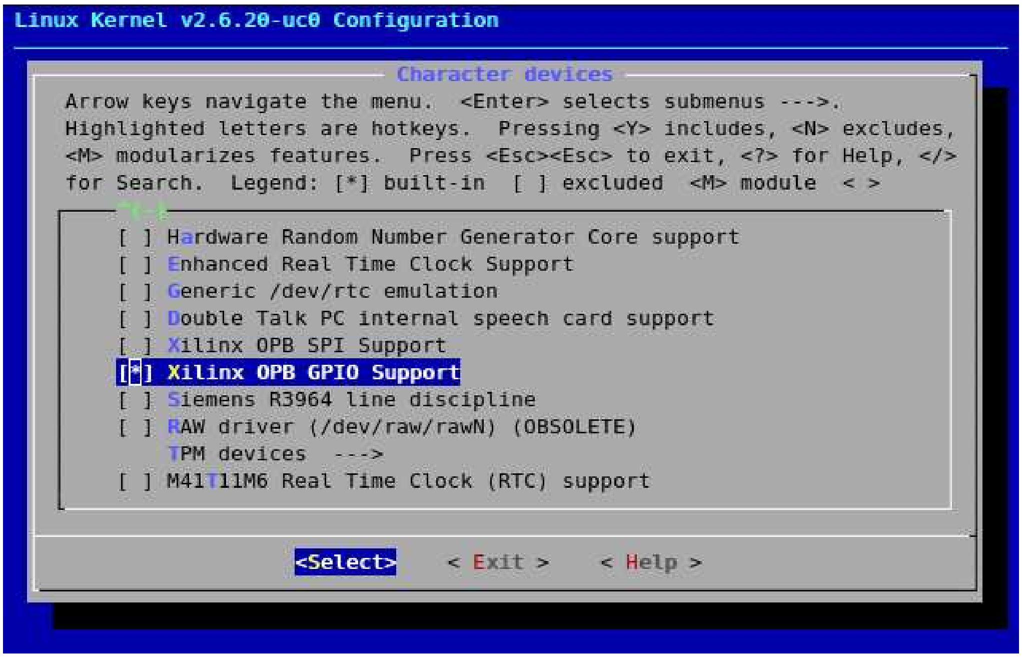
Once the patch has been applied, set the SPI settings in the Petalinux kernel. Open a console window and change into the ~/petalinux-svn/software/petalinux-dist directory and type 'make menuconfig' to open up the kernel configuration. In the uClinux Configuration window, select 'Kernel/Library/Defaults Selection' and check 'Customize Kernel Settings'. Hit 'Exit' twice and then 'Yes' to save the configuration. The Linux Kernel Configuration window will load. Select 'Device Drivers' and then 'SPI Support' Be sure to check 'SPI Support', 'Xilinx SPI Controller', and 'User mode SPI device driver support'. Everything else should be unchecked. If one or more of these options does not appear, please exit and reapply the patch properly.



```
Linux Kernel v2.6.20-uc0 Configuration

                          SPI support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] SPI support
      --- SPI Master Controller Drivers
      --- Bitbanging SPI master
      [*] Xilinx SPI controller
      [ ] Coldfire Queued SPI
      --- SPI Protocol Masters
      [*] User mode SPI device driver support

                 <Select>    < Exit >    < Help >
```

Figure 1 – Settings for the SPI device driver section of the uClinux kernel.

Exit out of the current screen and select 'Character Devices'. Be sure that 'Xilinx OPB SPI Support' is unchecked. Also, check 'Xilinx OPB GPIO Support' since GPIOs are used for many devices in the EDK design, including the custom IP core discussed in the section entitled *Adding the Custom Flash ADC Control IP Core*.

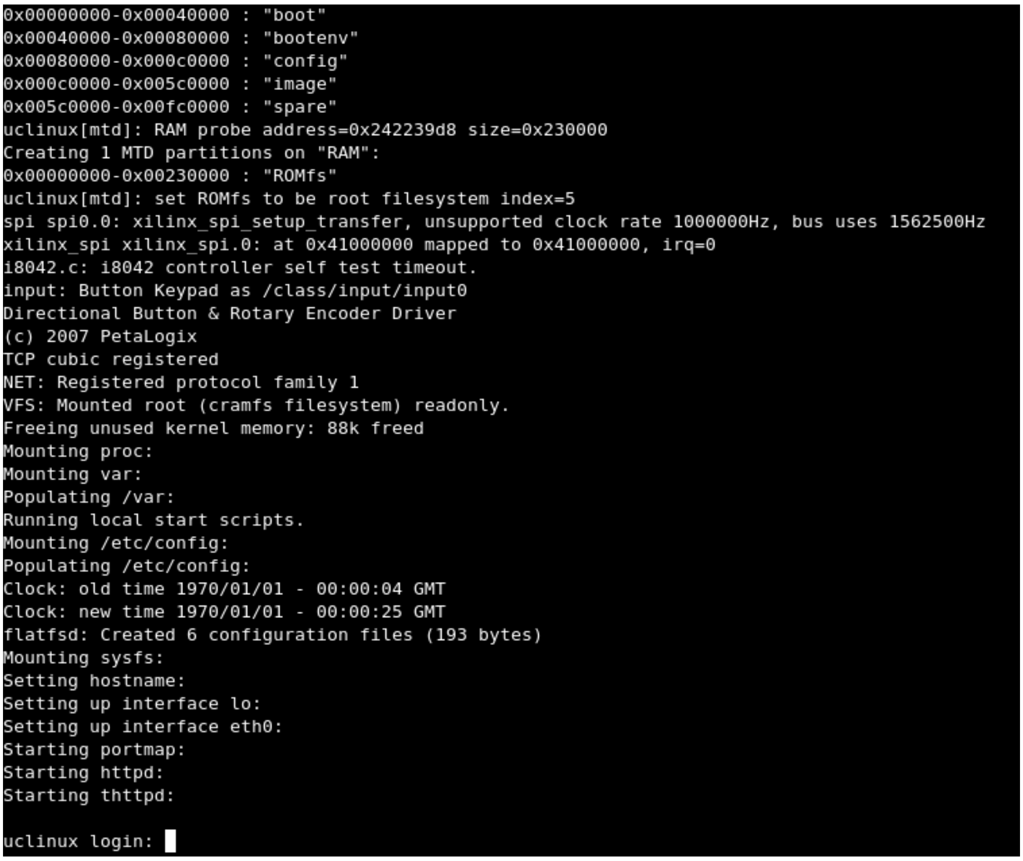


```
Linux Kernel v2.6.20-uc0 Configuration

                        Character devices
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [ ] Hardware Random Number Generator Core support
      [ ] Enhanced Real Time Clock Support
      [ ] Generic /dev/rtc emulation
      [ ] Double Talk PC internal speech card support
      [ ] Xilinx OPB SPI Support
      [*] Xilinx OPB GPIO Support
      [ ] Siemens R3964 line discipline
      [ ] RAW driver (/dev/raw/rawN) (OBSOLETE)
          TPM devices  --->
      [ ] M41T11M6 Real Time Clock (RTC) support

                <Select>    < Exit >    < Help >
```

Figure 2 - Settings for the character device driver section of the uClinux kernel.

Hit 'Exit' three times and hit 'Yes' to save the new configuration. Type 'make all' to build the new kernel.

When Petalinux boots up, a message for ‘spi’ and ‘xilinx_spi’ should be listed, as shown in the figure below:



```
0x00000000-0x00040000 : "boot"
0x00040000-0x00080000 : "bootenv"
0x00080000-0x000c0000 : "config"
0x000c0000-0x005c0000 : "image"
0x005c0000-0x00fc0000 : "spare"
uclinux[mtd]: RAM probe address=0x242239d8 size=0x230000
Creating 1 MTD partitions on "RAM":
0x00000000-0x00230000 : "ROMfs"
uclinux[mtd]: set ROMfs to be root filesystem index=5
spi spi0.0: xilinx_spi_setup_transfer, unsupported clock rate 1000000Hz, bus uses 1562500Hz
xilinx_spi xilinx_spi.0: at 0x41000000 mapped to 0x41000000, irq=0
i8042.c: i8042 controller self test timeout.
input: Button Keypad as /class/input/input0
Directional Button & Rotary Encoder Driver
(c) 2007 PetaLogix
TCP cubic registered
NET: Registered protocol family 1
VFS: Mounted root (cramfs filesystem) readonly.
Freeing unused kernel memory: 88k freed
Mounting proc:
Mounting var:
Populating /var:
Running local start scripts.
Mounting /etc/config:
Populating /etc/config:
Clock: old time 1970/01/01 - 00:00:04 GMT
Clock: new time 1970/01/01 - 00:00:25 GMT
flatfsd: Created 6 configuration files (193 bytes)
Mounting sysfs:
Setting hostname:
Setting up interface lo:
Setting up interface eth0:
Starting portmap:
Starting httpd:
Starting thttpd:

uclinux login: █
```

Figure 3 - A typical boot screen in Kermit showing the SPIDEV drive initialization.

This message verifies that you have set up the SPIDEV driver correctly in Petalinux.

IX. Implementing SPI Devices in uClinux

Once Petalinux has been configured to use the SPIDEV, devices that use the SPI bus, such as the digital-to-analog converter (DAC) and analog-to-digital converter (ADC), can be easily implemented in the EDK by adding the correct modules and connecting the proper pins in the UCF file. Once the hardware reference design has been recompiled, a user application needs to be created to access the device and compiled into the Petalinux kernel. Finally, boot the Spartan 3E Starter Kit with the new hardware .bit file and load the flash with the new U-boot configuration and kernel images. The SPI device will now be accessible using the user application you created.

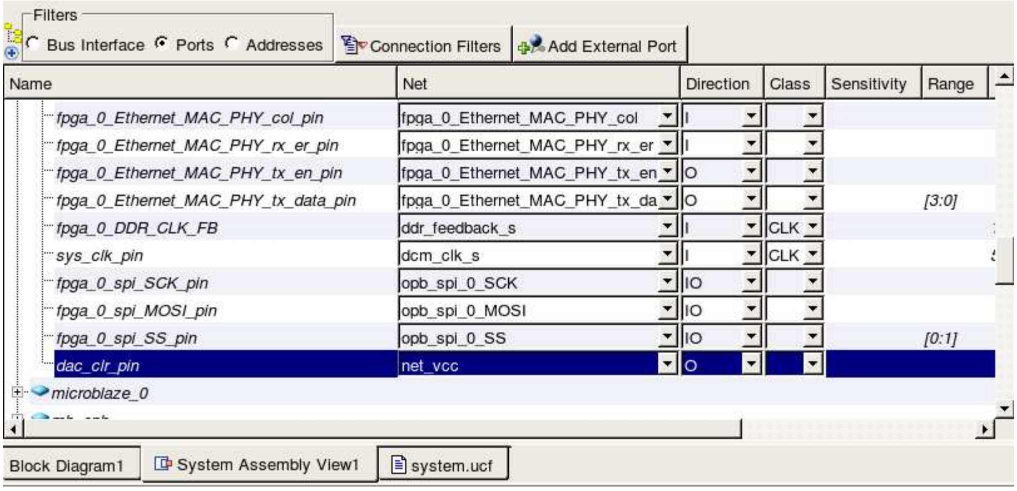
Implementing the Digital to Analog Converter (DAC) in uClinux

The onboard DAC chip of the Spartan 3E Starter Kit provides 4 DACs, each with an 12-bit output. For more specifications regarding the setup and communication with the DAC, please see the User's Guide [8].

For testing, you may wish to try building the DAC into your test development EDK project first as suggested in the section entitled *Creating a Test Development EDK Project* above. That way, you can program your board directly and test the DAC and your software without needing to recompile and load Petalinux onto the board.

To connect the DAC, add the lines to the UCF for the DAC as specified in the sample UCF file in Appendix C. Please note that in the sample file, the DACs chip-select (CS) is connected to bit 0 of the SPI Slave-Select (SS) Vector. Both the DAC CS and SS Vector are active low signals, so it is possible to use the SS vector to enable the DAC device directly.

In the EDK, open System Assembly View and switch to the Ports Filter. Hit ‘Add External Port’ to add a new external port to the bottom of the list. Rename the new port ‘dac_clr_pin’ to match the sample UCF file, set its direction to output (O) and connect it to the ‘net_vcc’ net, since we wish the DAC to always be on. Clearing the DAC can also be performed in software using a GPIO connected to the DAC CLR pin and sending a '1' to clear the DAC. Add the correct entry into the UCF file. Also, there are no issues with bus contention, since the SS will control when the DAC has access to the SPI bus.



Name	Net	Direction	Class	Sensitivity	Range
fpga_0_Ethernet_MAC_PHY_col_pin	fpga_0_Ethernet_MAC_PHY_col	I			
fpga_0_Ethernet_MAC_PHY_rx_er_pin	fpga_0_Ethernet_MAC_PHY_rx_er	I			
fpga_0_Ethernet_MAC_PHY_tx_en_pin	fpga_0_Ethernet_MAC_PHY_tx_en	O			
fpga_0_Ethernet_MAC_PHY_tx_data_pin	fpga_0_Ethernet_MAC_PHY_tx_da	O			[3:0]
fpga_0_DDR_CLK_FB	ddr_feedback_s	I	CLK		
sys_clk_pin	dcm_clk_s	I	CLK		
fpga_0_spi_SCK_pin	opb_spi_0_SCK	IO			
fpga_0_spi_MOSI_pin	opb_spi_0_MOSI	IO			
fpga_0_spi_SS_pin	opb_spi_0_SS	IO			[0:1]
dac_clr_pin	net_vcc	O			
microblaze_0					

Figure 4 - The DAC CLR external port configuration in Xilinx EDK.

Next, write your user application to interface to the DAC. I’d recommend that you first write it as a software project in your EDK test development project and then translate it to a Petalinux user application. To do so, open your software project under the Applications tab in EDK. If you have not created a test project or software project yet, please see the section entitled *Creating a Test Development EDK Project* above. Write software to interface with the DAC using C or C++. A sample user application for the SPI interface using the DAC and ADC has already been provided for use in Appendix D. Please note that the copy of the program provided is for use as a user application for Petalinux. To convert this project to your test development

EDK project, the constants need to be changed as described in the section entitled *Creating a Test Development EDK Project*.

Once you have written the software, recompile the hardware design and load the download.bit file onto your Spartan board. If you are using a test EDK project, load the Spartan 3E Starter Kit through iMPACT to verify that the DAC is working using your code. Once your are satisfied with the results, copy your hardware configuration settings and UCF file changes to the Petalinux reference design and translate your software project into a user application as discussed above in the section entitled *Creating a Test Development EDK Project*.

If you are using Petalinux or are translating your changes over to Petalinux, be sure to create and compile your code as a user application as discussed in the section entitled *Creating User Applications for uClinux* in Brian and Patrick's paper [2]. Program your board with the .bit file, reload the U-boot configuration and kernel image, and then boot Petalinux to test your user application and communication with the DAC.

Implementing the Analog to Digital Converter (ADC) in uClinux

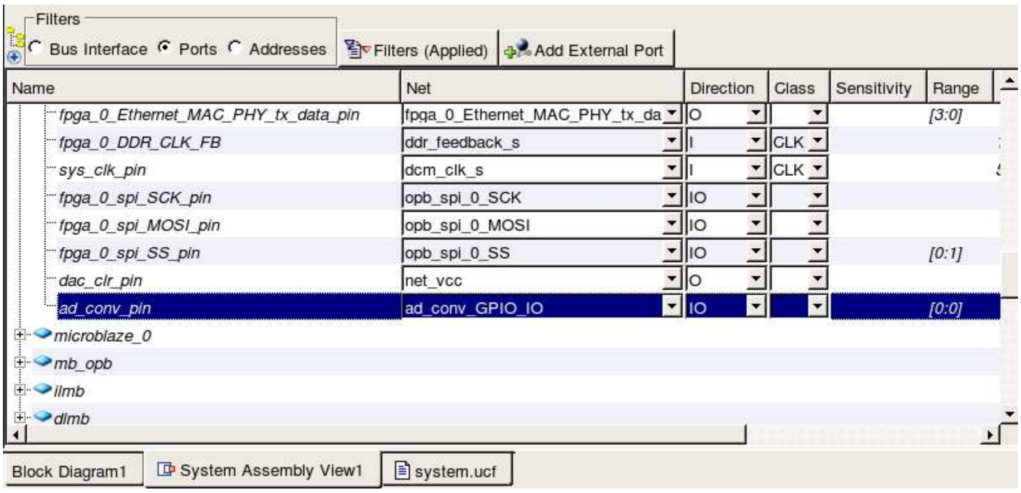
The onboard ADC chip of the Spartan 3E Starter Kit provides 2 ADCs connected to an onboard preamplifier, each with a different analog range depending on the gain settings. For more specifications regarding the setup and communication with the ADC, please see the User's Guide [8].

For testing, you may wish to try building the ADC into your test development EDK project first as suggested in the section entitled *Creating a Test Development EDK Project* above. That way, you can program your board directly and test the ADC and your software without needing to recompile and load Petalinux onto the board.

To connect the ADC, add the lines to the UCF for the ADC as specified in the sample UCF file in Appendix C. Please note that in the sample file, the Preamp chip-select (CS) is connected to bit 1 of the SPI Slave-Select (SS) Vector. Both the Preamp CS and SS Vector are active low signals, so it is possible to using the SS vector to enable the Preamp device directly. Controlling this signal also controls the ADC, since the ADC is connected directly to the preamp internally.

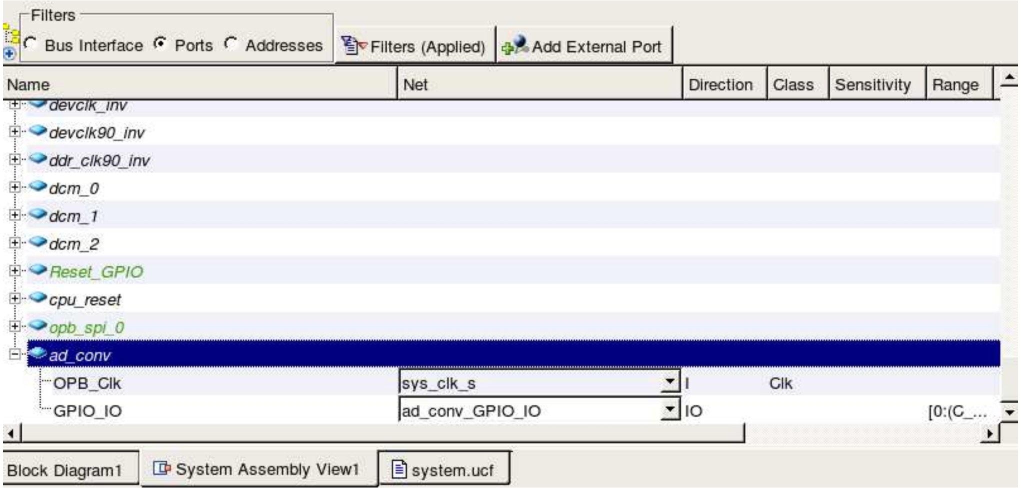
The ADC is selected using the AD_CONV signal, which is an active-high control signal. Since it is active-high and due to the nature of how the AD_CONV signal must be triggered, it cannot be connected to the SS vector directly. Please see the User’s Guide for more information regarding the nature and operation of the AD_CONV signal [8]. To control the ADC with the AD_CONV signal, we must create a GPIO that controls this pin and then set it manually in the software code.

In the EDK, open System Assembly View and switch to the Ports Filter. Add a new General Purpose IO (GPIO) IP Core from the IP Catalog on the left-hand side by double-clicking the core. Rename it to 'ad_conv' to match the UCF file. Right-click on the new IP core and hit 'Configure IP'. In the Properties window, change 'GPIO Data Bus Width' to 1 and hit 'OK' to close the Properties window. In the Ports view, connect OPB_Clk to 'sys_clk_s' and click 'Make External' under 'GPIO_IO'. Scroll up and rename the External Port to 'ad_conv_pin' to match the UCF file. Change the Filter to 'Bus Interface' and hook up the new ad_conv GPIO to the OPB bus by clicking the green circle to the left of the 'SOPB' connection. Change the Filter to 'Addresses' and assign the ad_conv GPIO 512 bytes starting at 0x41900000 to match the code in Appendix D.



Name	Net	Direction	Class	Sensitivity	Range
fpga_0_Ethernet_MAC_PHY_tx_data_pin	fpga_0_Ethernet_MAC_PHY_tx_da	O			[3:0]
fpga_0_DDR_CLK_FB	ddr_feedback_s	I	CLK		
sys_clk_pin	dcm_clk_s	I	CLK		
fpga_0_spi_SCK_pin	opb_spi_0_SCK	IO			
fpga_0_spi_MOSI_pin	opb_spi_0_MOSI	IO			
fpga_0_spi_SS_pin	opb_spi_0_SS	IO			[0:1]
dac_clr_pin	net_vcc	O			
ad_conv_pin	ad_conv_GPIO_IO	IO			[0:0]
microblaze_0					
mb_opb					
ilmb					
dlmb					

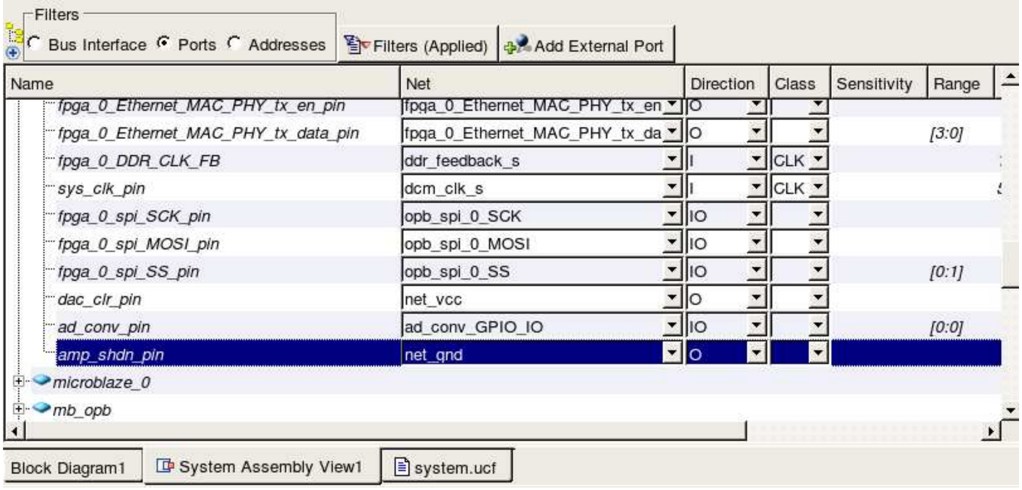
Figure 5 - The AD CONV external port configuration in Xilinx EDK.



Name	Net	Direction	Class	Sensitivity	Range
devclk_inv					
devclk90_inv					
ddr_clk90_inv					
dcm_0					
dcm_1					
dcm_2					
Reset_GPIO					
cpu_reset					
opb_spi_0					
ad_conv					
OPB_Clk	sys_clk_s	I	Clk		
GPIO_IO	ad_conv_GPIO_IO	IO			[0:(C_...

Figure 6 - The AD CONV GPIO configuration in Xilinx EDK.

Next, we need to add an external connection to the AMP SHDN net. Hit ‘Add External Port’ to add a new external port to the bottom of the list. Rename the new port ‘amp_shdn_pin’ to match the sample UCF file, set its direction to output (O) and connect it to the ‘net_gnd’ net, since we wish the Preamp (and ADC) to always be on. Add the correct entry into the UCF file. Shutting down the Preamp can also be performed in software using a GPIO connected to the AMP SHDN pin and sending a '0' to shut off the Preamp. There are no issues with bus contention, since the SS will control when the amp (and thus ADC) has access to the SPI bus.



Name	Net	Direction	Class	Sensitivity	Range
fpga_0_Ethernet_MAC_PHY_tx_en_pin	fpga_0_Ethernet_MAC_PHY_tx_en	O			
fpga_0_Ethernet_MAC_PHY_tx_data_pin	fpga_0_Ethernet_MAC_PHY_tx_da	O			[3:0]
fpga_0_DDR_CLK_FB	ddr_feedback_s	I	CLK		
sys_clk_pin	dcm_clk_s	I	CLK		
fpga_0_spi_SCK_pin	opb_spi_0_SCK	IO			
fpga_0_spi_MOSI_pin	opb_spi_0_MOSI	IO			
fpga_0_spi_SS_pin	opb_spi_0_SS	IO			[0:1]
dac_clr_pin	net_vcc	O			
ad_conv_pin	ad_conv_GPIO_IO	IO			[0:0]
amp_shdn_pin	net_gnd	O			
microblaze_0					
mb_opb					

Figure 7 - The AMP SHDN external port configuration in Xilinx EDK.

Unlike the DAC, the ADC has special bus contention issues. The ADC uses the SPI MISO signal, which shares a pin with bit 0 of the flash's data (DQ) bus. The flash is used to by Petalinux to hold the U-Boot and kernel information. To avoid this issue, the correct devices must be selected as described in the User's Manual [8]. However, in the EDK, there is no way to select between the flash and ADC for this single pin. To solve this issue, a new IP core was developed.

The Flash ADC Control IP Core sends the information on the shared pin to the correct location based on both the flash chip-enable (CEN) and a manual CEN controlled by a GPIO as shown in the diagram in Figure 8 below.

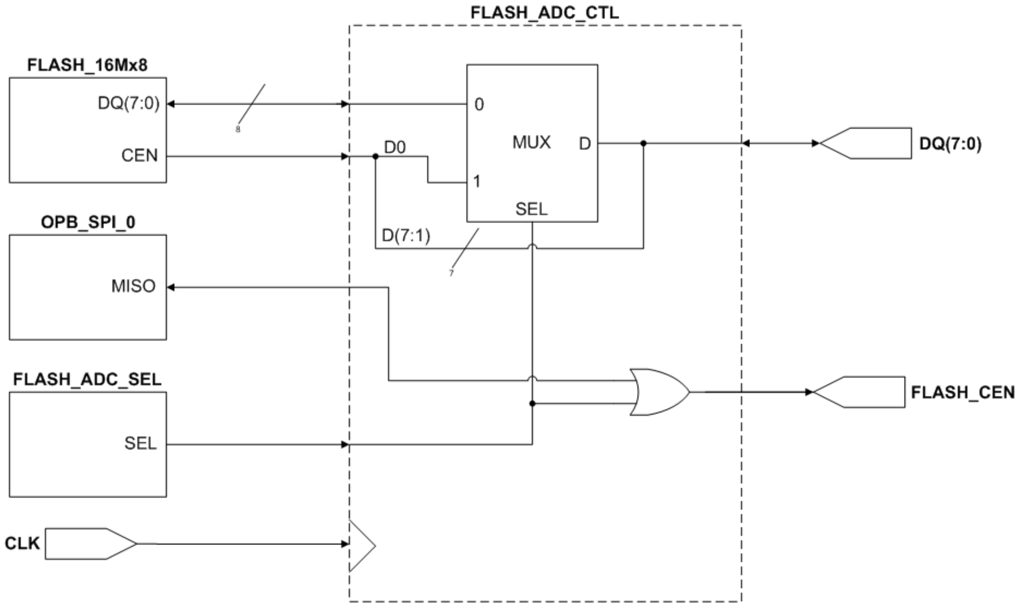
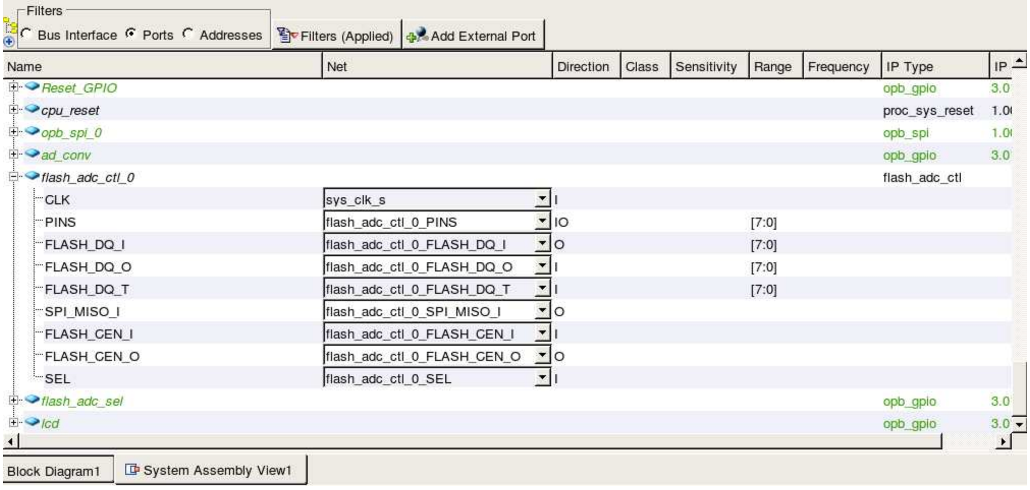


Figure 8 - A diagram of the Flash ADC Control IP Core implementation.

The IP core was developed in VHDL using the Xilinx ISE. If changes need to be made to this code, open the flash_adc_ctl ISE project, edit the VHDL code, and run 'Synthesize' to verify the code compiles correctly.

The Flash ADC Control IP Core needs to be added to the reference design (and test development EDK project if you are using it to run the ADC properly in Petalinux. In EDK, go to Hardware > Create or Import Peripheral. In the Create and Import peripheral wizard, hit 'Next', select 'Import Existing Peripheral' and hit 'Next' again. Select 'To an XPS Project' and hit 'Next'. Enter 'flash_adc_ctrl' as the VHDL top entity name and hit 'Next'. Check 'HDL Source Files' and hit 'Next'. Select 'Browse to your HDL source and dependent library files in next step' and hit 'Next'. Click 'Add Files', browse to the flash_adc_ctl.vhd file, hit 'Open' and then hit 'Next'. Uncheck 'Select Bus Interface(s)' and hit 'Next'. Uncheck 'Select and Configure Interrupt(s)' and hit 'Next'. Hit 'Next' and 'Finish' to complete the import of the IP Core.

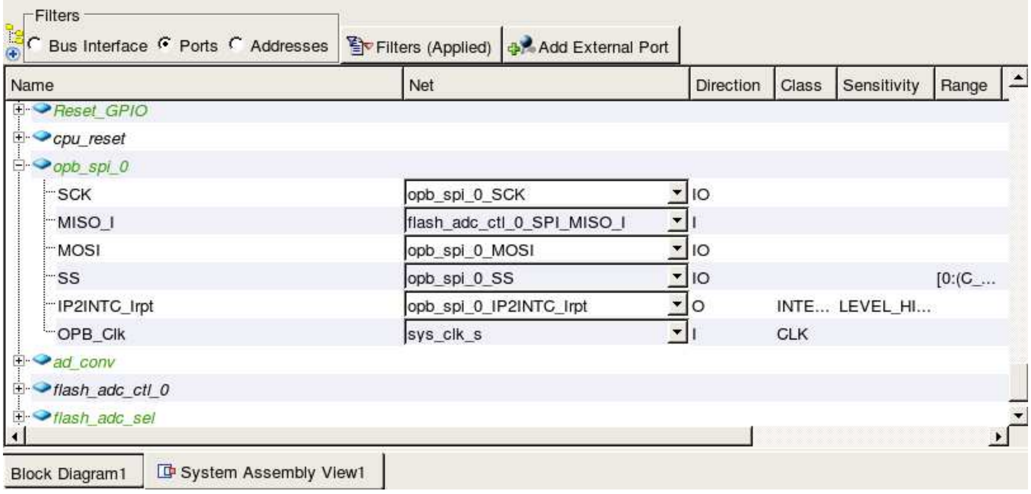
Next, add and configure the custom IP core in the reference design (or test design) EDK project. In the System Assembly View, add the flash_adc_ctl IP Core from the IP Catalog by double-clicking on it under Project local pcores. Configure the Flash ADC Control, SPI, and Flash IP Cores as shown below. Be sure to pay careful attention to all connections and bus widths (7:0 and 0:7 are different, so set it up as shown).



Filters: ○ Bus Interface ● Ports ○ Addresses

Name	Net	Direction	Class	Sensitivity	Range	Frequency	IP Type	IP
Reset_GPIO							opb_gpio	3.0
cpu_reset							proc_sys_reset	1.0
opb_spi_0							opb_spi	1.0
ad_conv							opb_gpio	3.0
flash_adc_ctl_0							flash_adc_ctl	
CLK	sys_clk_s	I						
PINS	flash_adc_ctl_0_PINS	IO			[7:0]			
FLASH_DQ_I	flash_adc_ctl_0_FLASH_DQ_I	O			[7:0]			
FLASH_DQ_O	flash_adc_ctl_0_FLASH_DQ_O	I			[7:0]			
FLASH_DQ_T	flash_adc_ctl_0_FLASH_DQ_T	I			[7:0]			
SPI_MISO_I	flash_adc_ctl_0_SPI_MISO_I	O						
FLASH_CEN_I	flash_adc_ctl_0_FLASH_CEN_I	I						
FLASH_CEN_O	flash_adc_ctl_0_FLASH_CEN_O	O						
SEL	flash_adc_ctl_0_SEL	I						
flash_adc_sel							opb_gpio	3.0
lcd							opb_gpio	3.0

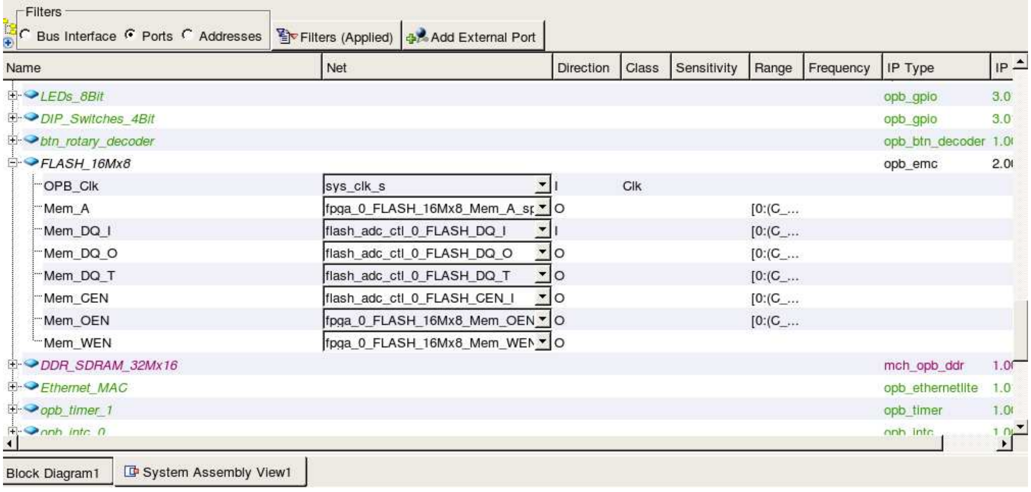
Figure 9 - The Flash ADC Control IP core configuration in Xilinx EDK.



Filters: ○ Bus Interface ● Ports ○ Addresses

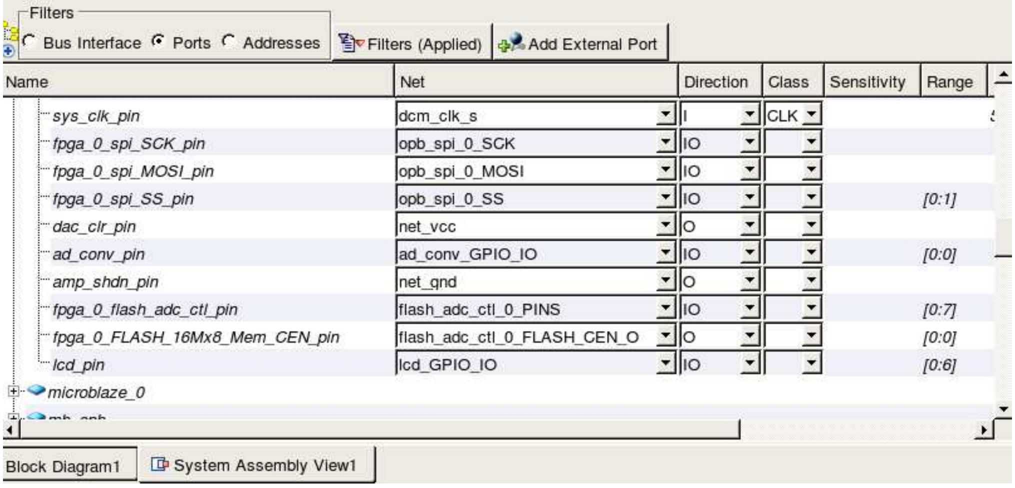
Name	Net	Direction	Class	Sensitivity	Range
Reset_GPIO					
cpu_reset					
opb_spi_0					
SCK	opb_spi_0_SCK	IO			
MISO_I	flash_adc_ctl_0_SPI_MISO_I	I			
MOSI	opb_spi_0_MOSI	IO			
SS	opb_spi_0_SS	IO			[0:(C_...
IP2INTC_Irpt	opb_spi_0_IP2INTC_Irpt	O	INTE...	LEVEL_HI...	
OPB_Clk	sys_clk_s	I	CLK		
ad_conv					
flash_adc_ctl_0					
flash_adc_sel					

Figure 10 - The SPI IP core configuration in Xilinx EDK.



Name	Net	Direction	Class	Sensitivity	Range	Frequency	IP Type	IP
LEDs_8Bit							opb_gpio	3.0
DIP_Switches_4Bit							opb_gpio	3.0
btn_rotary_decoder							opb_btn_decoder	1.0
FLASH_16Mx8							opb_emc	2.0
OPB_Clk	sys_clk_s	I	Clk					
Mem_A	fpga_0_FLASH_16Mx8_Mem_A_sp	O			[0:(C_...			
Mem_DQ_I	flash_adc_ctl_0_FLASH_DQ_I	I			[0:(C_...			
Mem_DQ_O	flash_adc_ctl_0_FLASH_DQ_O	O			[0:(C_...			
Mem_DQ_T	flash_adc_ctl_0_FLASH_DQ_T	O			[0:(C_...			
Mem_CEN	flash_adc_ctl_0_FLASH_CEN_I	O			[0:(C_...			
Mem_OEN	fpga_0_FLASH_16Mx8_Mem_OEN	O			[0:(C_...			
Mem_WEN	fpga_0_FLASH_16Mx8_Mem_WEN	O						
DDR_SDRAM_32Mx16							mch_opb_ddr	1.0
Ethernet_MAC							opb_ethernetlite	1.0
opb_timer_1							opb_timer	1.0
opb_intc_0							opb_intc	1.0

Figure 11 - The Flash IP core configuration in Xilinx EDK.



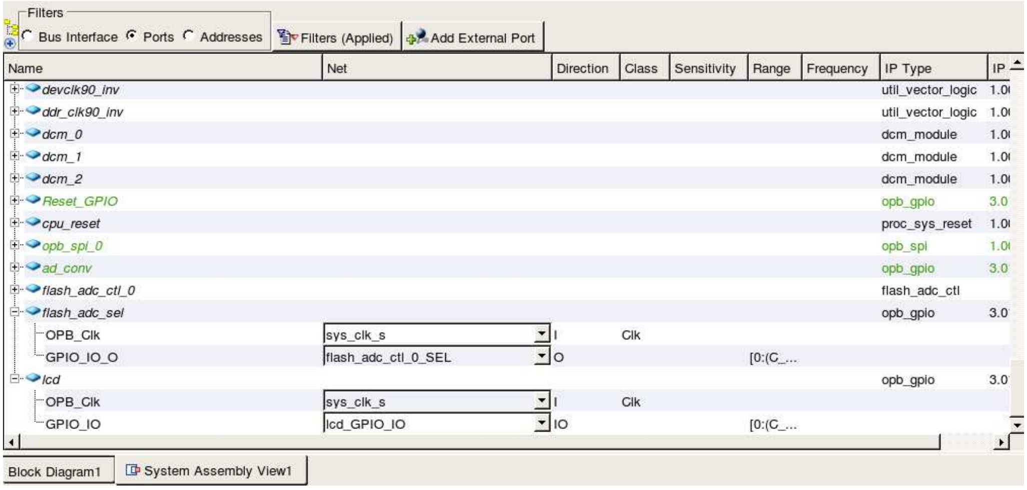
Name	Net	Direction	Class	Sensitivity	Range
sys_clk_pin	dcm_clk_s	I	CLK		
fpga_0_spi_SCK_pin	opb_spi_0_SCK	IO			
fpga_0_spi_MOSI_pin	opb_spi_0_MOSI	IO			
fpga_0_spi_SS_pin	opb_spi_0_SS	IO			[0:1]
dac_clr_pin	net_vcc	O			
ad_conv_pin	ad_conv_GPIO_IO	IO			[0:0]
amp_shdn_pin	net_gnd	O			
fpga_0_flash_adc_ctl_pin	flash_adc_ctl_0_PINS	IO			[0:7]
fpga_0_FLASH_16Mx8_Mem_CEN_pin	flash_adc_ctl_0_FLASH_CEN_O	O			[0:0]
lcd_pin	lcd_GPIO_IO	IO			[0:6]
microblaze_0					
mb_opb					

Figure 12 - The external port configuration for all the devices needed to implement the DAC and ADC in Xilinx EDK.

Please note that a LCD GPIO has also been added to display the input of the ADC on the LCD screen. This is not required, but it is a nice feature for testing. The code and UCF file provided in Appendices C and D include the LCD module. This module is a standard GPIO with a 7-bit IO bus. Please see Figure 14 below for pin settings and memory mappings.

The Flash ADC Control IP Core does not need to be added to a bus or have address space allocated to it. Add the appropriate external port connections to the UCF file as shown in Appendix C.

In addition to the Flash CEN signal, an external CEN must be added that is controlled manually. This signal is ORed with the Flash CEN, overriding it in the case we want to disable the flash (flash CEN set to '1'). We must create a GPIO that controls this pin and then set it manually in the software code. In the EDK, open System Assembly View and switch to the Ports Filter. Add a new General Purpose IO (GPIO) IP Core from the IP Catalog on the left-hand side by double-clicking the core. Rename it to 'flash_adc_sel' to match the UCF file. Right-click on the new IP core and hit 'Configure IP'. In the Properties window, change 'GPIO Data Bus Width' to 1 and hit 'OK' to close the Properties window.   In the Ports view, connect the IP core as shown in the following figure.

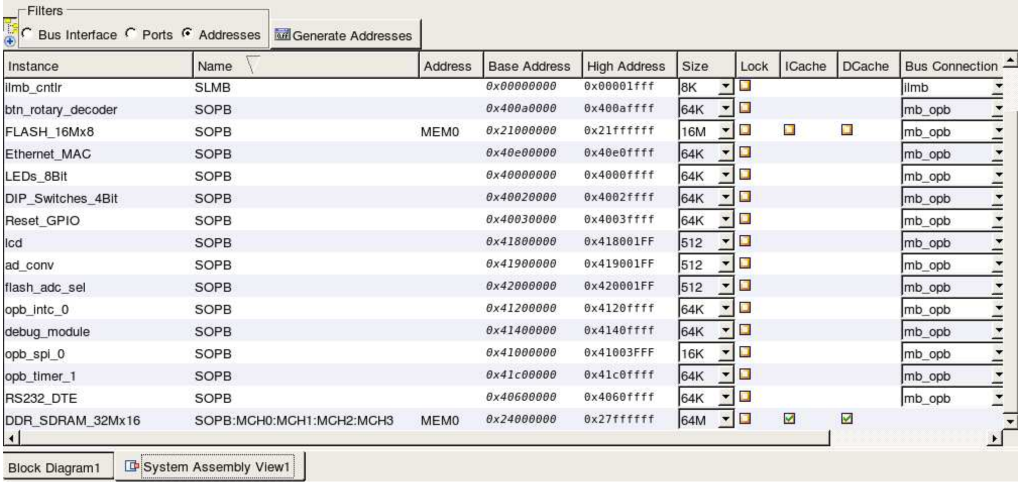


Name	Net	Direction	Class	Sensitivity	Range	Frequency	IP Type	IP
devclk90_inv							util_vector_logic	1.0
ddr_clk90_inv							util_vector_logic	1.0
dcm_0							dcm_module	1.0
dcm_1							dcm_module	1.0
dcm_2							dcm_module	1.0
Reset_GPIO							opb_gpio	3.0
cpu_reset							proc_sys_reset	1.0
opb_spi_0							opb_spi	1.0
ad_conv							opb_gpio	3.0
flash_adc_ctl_0							flash_adc_ctl	
flash_adc_sel							opb_gpio	3.0
OPB_Clk	sys_clk_s	I	Clk					
GPIO_IO_O	flash_adc_ctl_0_SEL	O			[0:(C_...			
lcd							opb_gpio	3.0
OPB_Clk	sys_clk_s	I	Clk					
GPIO_IO	lcd_GPIO_IO	IO			[0:(C_...			

Figure 13 - The Flash ADC Select GPIO and LCD GPIO configuration in Xilinx EDK.

Change the Filter to 'Bus Interface' and hook up the new flash_adc_sel GPIO to the OPB bus by clicking the green circle to the left of the 'SOPB' connection. Change the Filter to

'Addresses' and assign the flash_adc_sel GPIO 512 bytes starting at 0x42000000 to match the code in Appendix D.



Instance	Name	Address	Base Address	High Address	Size	Lock	ICache	DCache	Bus Connection
ilmb_cntlr	SLMB		0x00000000	0x00001fff	8K	□			ilmb
btn_rotary_decoder	SOPB		0x400a0000	0x400affff	64K	□			mb_opb
FLASH_16Mx8	SOPB	MEM0	0x21000000	0x21ffffff	16M	□	□	□	mb_opb
Ethernet_MAC	SOPB		0x40e00000	0x40e0ffff	64K	□			mb_opb
LEDs_8Bit	SOPB		0x40000000	0x4000ffff	64K	□			mb_opb
DIP_Switches_4Bit	SOPB		0x40020000	0x4002ffff	64K	□			mb_opb
Reset_GPIO	SOPB		0x40030000	0x4003ffff	64K	□			mb_opb
lcd	SOPB		0x41800000	0x418001FF	512	□			mb_opb
ad_conv	SOPB		0x41900000	0x419001FF	512	□			mb_opb
flash_adc_sel	SOPB		0x42000000	0x420001FF	512	□			mb_opb
opb_intc_0	SOPB		0x41200000	0x4120ffff	64K	□			mb_opb
debug_module	SOPB		0x41400000	0x4140ffff	64K	□			mb_opb
opb_spi_0	SOPB		0x41000000	0x41003FFF	16K	□			mb_opb
opb_timer_1	SOPB		0x41c00000	0x41c0ffff	64K	□			mb_opb
RS232_DTE	SOPB		0x40600000	0x4060ffff	64K	□			mb_opb
DDR_SDRAM_32Mx16	SOPB:MCH0:MCH1:MCH2:MCH3	MEM0	0x24000000	0x27ffffff	64M	□	☑	☑	

Figure 14 - The memory mappings for all the devices needed to implement the DAC and ADC in Xilinx EDK.

Next, write your user application to interface to the ADC. I'd recommend that you first write it as a software project in your EDK test development project and then translate it to a Petalinux user application. To do so, open your software project under the Applications tab in EDK. If you have not created a test project or software project yet, please see the section entitled *Creating a Test Development EDK Project* above.

Write software to interface with the ADC using C or C++. A sample user application for the SPI interface using the DAC and ADC has already been provided for use in Appendix D. Please note that the copy of the program provided is for use as a user application for Petalinux. This code also includes the LCD code to display values of the ADC to the LCD screen. If you wish to use the LCD, be sure to add an LCD GPIO to your project and add the correct pins in the UCF file. Otherwise, remove the references to the LCD functions. To convert this project to your test development EDK project, the constants need to be changed as described in the section entitled *Creating a Test Development EDK Project*.

Once you have written the software, recompile the hardware design and load the download.bit file onto your Spartan board. If you are using a test EDK project, load the Spartan 3E Starter Kit through iMPACT to verify that the ADC is working using your code. Once your are satisfied with the results, copy your hardware configuration settings and UCF file changes to the Petalinux reference design and translate your software project into a user application as discussed above in the section entitled *Creating a Test Development EDK Project*.

If you are using Petalinux or are translating your changes over to Petalinux, be sure to create and compile in your code as a user application as discussed in the section entitled *Creating User Applications for uClinux* in Brian and Patrick's paper [2]. Program your board with the .bit file, reload the U-boot configuration and kernel image, and then boot Petalinux to test your user application and communication with the ADC.

X. Testing

To test SPIDEV, there are several test scripts that can be implemented to test the SPIDEV driver. The default test script for SPI is located in the Xilinx git directory under linux-2.6-xlnx/Documentation/spi/spidev_test.c. Another test script was provided by the uclinux-microblaze mailing list and is available at `http://www.itee.uq.edu.au/~listarch/microblaze-uclinux/archive/2009/03/msg00000.html`. Install these scripts into Petalinux as user applications. The process for doing so can be found in the section entitled *Creating User Applications for uClinux* in Brian and Patrick's paper [2].

XI. Recommendations

Now that the SPIDEV driver has been successfully tested and implemented using the instructions included in this paper, there are several improvements and modifications that can be made to both improve the functionality of the driver as well as fix some issues that I faced while working on the driver.

Custom U-Boot Auto Configuration Script

As suggested in the section entitled *Configure TFTP in Petalinux* above, the current implementation of Ethernet in the U-Boot environment requires the user to set static IP addresses. In addition, these settings must be redone every time a new U-Boot auto configuration script is run (ub.config.img). Some improvements include using DHCP instead of static IP addresses and customizing the configuration script so that your settings will not be overwritten. Please see the section suggested for links to information regarding these modifications.

Upgrade Petalinux from SVN Build

The latest build at the onset of my project was petalinux-v0.30-rc1. However, since then the uclinux-microblaze mailing list has updated the code to petalinux-v0.40-rc3. Although I have not confirmed this, many of the developers on the mailing list suggested that the SPIDEV driver be added to the latest release. As of petalinux-v0.40-rc3, which I used to test the procedures I've outlined in this paper, there is still no SPIDEV support in the kernel by default. If newer releases emerge, it may be worth looking into them, since then it would be unnecessary to run the patch and make manual modifications to the uClinux kernel. This is merely suggestion; the SVN build, which is the latest, will be based off of petalinux-v0.40-rc3 and will also include the latest software updates, so either build may already include the SPIDEV driver now. The code I have used and developed was provided to Dr. Harris on a CD to allow the next student working on the uClinux part of the SuPER project to continue development from where I have left of, so the best suggestion regarding the software would be to use the version on the CD to develop your changes, since it is where I left off in my work.

User-Friendly SPI Interface

Although the code provided in Appendix D provides a good framework for accessing the SPI devices in Petalinux, a much better program could be written to provide more functionality and easier user access. The user application used in this paper was merely a simple interface to verify that a person could communicate with and use the SPI devices on the Spartan 3E Starter Kit using the console in Petalinux. Future implementations might also include a wider range of devices besides the DAC and ADC. They may also expand the number of command line arguments and make it easier for the user to understand. Ultimately, the code from the SuPER project will need to be ported from the laptop to the Spartan 3E, so a friendly user command for accessing the SPI devices would be very helpful and make updating the code much easier.

XII. Conclusion

This project built on the previous successes of Brian, Patrick, and Alex, who worked on the initial development of Petalinux for the SuPER project. The reason we wish to develop a working system in Petalinux is to allow the SuPER team to port the program that collects data from the SuPER and sends out control signals on to the Spartan 3E Starter Kit board. This would eliminate the laptop and other microcontrollers that are currently used on the SuPER and would cut the power consumption down greatly.

At the start of this project, I focused on finding the best development environment for working with uClinux, which turned out to be a standalone CentOS installation. Next, I implemented and tested the Ethernet interface on the Spartan 3E and used it to update the Petalinux kernel on the board much faster than the old serial interface. Next, I worked on the development of the SPIDEV driver in uClinux and implemented the driver with help from the uclinux-microblaze mailing list. Once the interface was up and testing well, I added the DAC into the Petalinux user application for SPI access. Finally, I worked on the addition of the ADC to the SPI interface in Petalinux including development of a new IP core that would handle the contention on the pin that is shared between the flash memory and ADC.

The main hurdles in this project were the documentation regarding SPI drivers for Petalinux and the bus contention issues with the ADC. Thanks to the mailing list, I was able to develop and implement the SPIDEV driver successfully. The ADC works correctly under the test environment, and Xilinx’s lack of information regarding use of the ADC and flash together presented a considerable roadblock which I was able to overcome with a custom IP core. The uclinux-microblaze mailing list was very helpful for learning about previous implementations as well as providing support and advice regarding pieces of this project such as the SPIDEV driver. All the previous work done by Brian, Patrick, and Alex also helped greatly in understanding Petalinux and SPI devices. Without their work, this project would not have been possible in the twenty weeks that it lasted.

Appendix A – Bibliography

[1] James G. Harris, “White Paper for Sustainable Power for Electrical Resources - SuPER”. Available: http://courseware.ee.calpoly.edu/~jharris/research/super_project/white_paper_susper.pdf

[2] Brian Estrada and Patrick Mariano, “Development of uClinux Platform for Cal Poly SuPER Project”. Available: http://courseware.ee.calpoly.edu/~jharris/research/super_project/be_pm_sp.pdf

[3] Alex Diaz, “Development of uClinux SPI interface On Spartan 3e Development board For Cal Poly Super Project”. Available: http://courseware.ee.calpoly.edu/~jharris/research/super_project/ad_sp.pdf

[4] Matt Staniszewski, “HOWTO: Setup the CentOS and Xilinx Development Environment”. Available: http://super.ceng.calpoly.edu/doku.php?id=howto_setup_centos_xilinx

[5] Matt Staniszewski, “HOWTO: Obtain and Install the Ethernet Lite OPB Core”. Available: http://super.ceng.calpoly.edu/doku.php?id=howto_ethernet_lite

[6] Petalogix, “Customizing U-Boot”. Available: http://developer.petalogix.com/wiki/UserGuide/Bootloaders/UBoot/CustUBoot

[7] Petalogix, “System Setting Configuration Menu”. Available: http://developer.petalogix.com/wiki/UserGuide/Customising/SystemConfigMenu

[8] Xilinx, “Spartan-3E FPGA Starter Kit Board User Guide”. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

Appendix B - SPIDEV Kernel Patch

```
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/Makefile petalinux-
svn/software/linux-2.6.x-petalogix/arch/microblaze/platform/common/Makefile
--- petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/Makefile    2009-01-30
12:24:12.000000000 +0100
+++ petalinux-svn/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/Makefile    2009-03-02
10:38:46.000000000 +0100
@@ -14,7 +14,7 @@ EXTRA_CFLAGS      += -I$(TOPDIR)/drivers/xili

 platobj-$(CONFIG_MTD_PHYSMAP) += physmap-flash.o
 platobj-$(CONFIG_XILINX_GPIO) += xgpio.o
-platobj-$(CONFIG_XILINX_SPI)  += xspi.o
+platobj-$(CONFIG_SPI_XILINX)  += xspi.o
 platobj-$(CONFIG_SERIAL_UARTLITE)  += xuartlite.o
 platobj-$(CONFIG_SERIAL_8250)  += x16550.o
 platobj-$(CONFIG_XILINX_SYSACE)  += xsysace.o
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/xspi.c petalinux-
svn/software/linux-2.6.x-petalogix/arch/microblaze/platform/common/xspi.c
--- petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/xspi.c      2007-04-01
22:54:51.000000000 -0700
+++ petalinux-svn/software/linux-2.6.x-
petalogix/arch/microblaze/platform/common/xspi.c      2009-05-18
23:44:51.000000000 -0700
@@ -1,83 +1,80 @@
-/*
- * arch/microblaze/platform/common/xspi.c
- *
- * platform device initialisation for Xilinx SPI devices
- *
- * Copyright 2007 PetaLogix
- *
- * based on original kernel/platform.c which was
- * Copyright 2007 LynuxWorks
- *
- * This file is licensed under the terms of the GNU General Public License
- * version 2.  This program is licensed "as is" without any warranty of any
- * kind, whether express or implied.
- */
-
-#include <linux/autoconf.h>
-#include <linux/init.h>
-#include <linux/resource.h>
-#include <linux/xilinx_devices.h>
-#include <linux/serial_8250.h>
-
-#define XSPI_PLATFORM_DATA_INITIALISER(n)            \
-{                                        \
-     .device_flags = (CONFIG_XILINX_SPI_##n##_FIFO_EXIST ? XSPI_HAS_FIFOS :
0) | \
-           (CONFIG_XILINX_SPI_##n##_SPI_SLAVE_ONLY ? XSPI_SLAVE_ONLY : 0), \
-     .num_slave_bits = CONFIG_XILINX_SPI_##n##_NUM_SS_BITS \
-}
-
-#define XSPI_PLATFORM_DEVICE_INITIALISER(n)          \
-{                                        \
-     .name = "xilinx_spi",                     \
-     .id = (n),                         \
-     .dev.platform_data = &xspi_pdata[n],             \
-     .num_resources = 2,                        \
-     .resource = (struct resource[]) {          \
-           {                            \
-                 .start      = CONFIG_XILINX_SPI_##n##_BASEADDR, \
-                 .end  = CONFIG_XILINX_SPI_##n##_HIGHADDR, \
-                 .flags      = IORESOURCE_MEM              \
-           },                                       \
-           {                                        \
-                 .start      = CONFIG_XILINX_SPI_##n##_IRQ,            \
-                 .end  = CONFIG_XILINX_SPI_##n##_IRQ,            \
-                 .flags      = IORESOURCE_IRQ              \
-           }                            \
-     }                                  \
-}
-
-static struct xspi_platform_data xspi_pdata[] = {
-#ifdef CONFIG_XILINX_SPI_0_INSTANCE
-XSPI_PLATFORM_DATA_INITIALISER(0),
-#endif
-#ifdef CONFIG_XILINX_SPI_1_INSTANCE
-XSPI_PLATFORM_DATA_INITIALISER(1),
-#endif
-#ifdef CONFIG_XILINX_SPI_2_INSTANCE
-XSPI_PLATFORM_DATA_INITIALISER(2),
-#endif
-};
-
-static struct platform_device xilinx_spi_device[] = {
-#ifdef CONFIG_XILINX_SPI_0_INSTANCE
-XSPI_PLATFORM_DEVICE_INITIALISER(0),
-#endif
-#ifdef CONFIG_XILINX_SPI_1_INSTANCE
-XSPI_PLATFORM_DEVICE_INITIALISER(1),
-#endif
-#ifdef CONFIG_XILINX_SPI_2_INSTANCE
-XSPI_PLATFORM_DEVICE_INITIALISER(2),
-#endif
-};
-
-static int __init xspi_platform_init(void)
-{
-     int i;
-
-     for(i=0;i<ARRAY_SIZE(xilinx_spi_device); i++)
-           platform_device_register(&xilinx_spi_device[i]);
-
```

```
-     return 0;
-}
-
-device_initcall(xspi_platform_init);
+/*
+* arch/microblaze/platform/common/xspi.c
+*
+* platform device initialisation for Xilinx SPI devices
+*
+* Based on arch/microblaze/platform/common/xspi.c and
+* xilspi.c from someone posting to the uclinux mailing list.
+*
+* Modified for the Xilinx xilinx_spi driver!
+*/
+
+#include <linux/autoconf.h>
+#include <linux/init.h>
+#include <linux/resource.h>
+#include <linux/xilinx_devices.h>
+#include <linux/serial_8250.h>
+
+
+#define XSPI_PLATFORM_DATA_INITIALISER(n)            \
+{                                          \
+     .bus_num = n, \
+     .num_chipselect = CONFIG_XILINX_SPI_##n##_NUM_SS_BITS,      \
+     .speed_hz = CONFIG_XILINX_CPU_CLOCK_FREQ /
CONFIG_XILINX_SPI_##n##_OPB_SCK_RATIO \
+}
+
+#define XSPI_PLATFORM_DEVICE_INITIALISER(n)                \
+{                                                        \
+        .name = "xilinx_spi",                               \
+        .id = (n),                                        \
+        .dev.platform_data = &xspi_pdata[n],               \
+        .num_resources = 2,                                \
+        .resource = (struct resource[]) {                \
+                {                                        \
+                        .start        = CONFIG_XILINX_SPI_##n##_BASEADDR,
\
+                        .end        = CONFIG_XILINX_SPI_##n##_HIGHADDR,
\
+                        .flags        = IORESOURCE_MEM
\
+                },                                                      \
+                {                                                     \
+                        .start        = CONFIG_XILINX_SPI_##n##_IRQ,
\
+                        .end        = CONFIG_XILINX_SPI_##n##_IRQ,
\
+                        .flags        = IORESOURCE_IRQ
\
+                }                                        \
+        }                                                \
+}
+
+static struct xspi_platform_data xspi_pdata[] = {
```

```
+#ifdef CONFIG_XILINX_SPI_0_INSTANCE
+XSPI_PLATFORM_DATA_INITIALISER(0),
+#endif
+#ifdef CONFIG_XILINX_SPI_1_INSTANCE
+XSPI_PLATFORM_DATA_INITIALISER(1),
+#endif
+#ifdef CONFIG_XILINX_SPI_2_INSTANCE
+XSPI_PLATFORM_DATA_INITIALISER(2),
+#endif
+};
+
+static struct platform_device xilinx_spi_device[] = {
+#ifdef CONFIG_XILINX_SPI_0_INSTANCE
+XSPI_PLATFORM_DEVICE_INITIALISER(0),
+#endif
+#ifdef CONFIG_XILINX_SPI_1_INSTANCE
+XSPI_PLATFORM_DEVICE_INITIALISER(1),
+#endif
+#ifdef CONFIG_XILINX_SPI_2_INSTANCE
+XSPI_PLATFORM_DEVICE_INITIALISER(2),
+#endif
+};
+
+static int __init xspi_platform_init(void)
+{
+        int i;
+
+        for(i=0;i<ARRAY_SIZE(xilinx_spi_device); i++)
+                platform_device_register(&xilinx_spi_device[i]);
+
+     return 0;
+}
+
+device_initcall(xspi_platform_init);

diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/Xilinx-Spartan3E500-RevD/setup.c
petalinux-svn/software/linux-2.6.x-petalogix/arch/microblaze/platform/Xilinx-
Spartan3E500-RevD/setup.c
--- petalinux-svn.orig/software/linux-2.6.x-
petalogix/arch/microblaze/platform/Xilinx-Spartan3E500-RevD/setup.c     2009-
01-14 10:16:06.000000000 +0100
+++ petalinux-svn/software/linux-2.6.x-
petalogix/arch/microblaze/platform/Xilinx-Spartan3E500-RevD/setup.c     2009-
03-02 11:22:45.000000000 +0100
@@ -7,4 +7,76 @@
  */

 /* Nothing to do, but we need this C file to keep kbuild happy */
+/* This is not true. We are now registering SPI slaves here. */

+#include <linux/device.h>
+#include <linux/spi/spi.h>
+
+static struct spi_board_info spi_board_info[] __initdata = {
+     /* (order of busses here has no influence on numbering) */
+     /* first bus on EVM (only 4 devices) */
+     {
+           /* SPI flash on the Spartan EVM --> spi-0 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 0,
+           .chip_select      = 0,
+     },
+     {
+           /* dummy --> spi-1 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 0,
+           .chip_select      = 1,
+     },
+     {
+           /* dummy --> spi-2 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 0,
+           .chip_select      = 2,
+     },
+     {
+           /* dummy --> spi-3 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 0,
+           .chip_select      = 3,
+     },
+     /* second bus on expansion board (only 4 devices) */
+     {
+           /* dummy --> spi-4 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 1,
+           .chip_select      = 0,
+     },
+     {
+           /* dummy --> spi-5 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 1,
+           .chip_select      = 1,
+     },
+     {
+           /* dummy --> spi-6 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 1,
+           .chip_select      = 2,
+     },
+     {
+           /* dummy --> spi-7 */
+           .modalias   = "spidev",
+           .max_speed_hz     = 1000000,
+           .bus_num    = 1,
+           .chip_select      = 3,
+     },
+};
+
+static int board_init_spi(void)
+{
+     spi_register_board_info(spi_board_info, ARRAY_SIZE(spi_board_info));
+     return 0;
+}
+arch_initcall(board_init_spi);
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/drivers/spi/Kconfig petalinux-svn/software/linux-2.6.x-
petalogix/drivers/spi/Kconfig
--- petalinux-svn.orig/software/linux-2.6.x-petalogix/drivers/spi/Kconfig
      2007-03-30 07:49:34.000000000 +0200
+++ petalinux-svn/software/linux-2.6.x-petalogix/drivers/spi/Kconfig    2009-
03-02 10:37:45.000000000 +0100
@@ -103,6 +103,14 @@ config SPI_S3C24XX_GPIO
        GPIO lines to provide the SPI bus. This can be used where
        the inbuilt hardware cannot provide the transfer mode, or
        where the board is using non hardware connected pins.
+
+config SPI_XILINX
+     tristate "Xilinx SPI controller"
+     depends on SPI_MASTER
+     select SPI_BITBANG
+     help
+     This exposes the SPI controller IP from the Xilinx EDK.
+     Experimental.
 #
 # Add new SPI master controllers in alphabetical order above this line
 #
@@ -143,6 +151,13 @@ config SPI_S3C24XX
 comment "SPI Protocol Masters"
      depends on SPI_MASTER

+config SPI_SPIDEV
+     tristate "User mode SPI device driver support"
+     help
+       This supports user mode SPI protocol drivers.
+
+       Note that this application programming interface is EXPERIMENTAL
+       and hence SUBJECT TO CHANGE WITHOUT NOTICE while it stabilizes.

 #
 # Add new SPI protocol masters in alphabetical order above this line
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/drivers/spi/Makefile petalinux-svn/software/linux-2.6.x-
petalogix/drivers/spi/Makefile
--- petalinux-svn.orig/software/linux-2.6.x-petalogix/drivers/spi/Makefile
      2007-03-30 07:49:34.000000000 +0200
+++ petalinux-svn/software/linux-2.6.x-petalogix/drivers/spi/Makefile   2009-
03-02 10:38:05.000000000 +0100
@@ -19,9 +19,11 @@ obj-$(CONFIG_SPI_S3C24XX_GPIO)           += spi_s
 obj-$(CONFIG_SPI_S3C24XX)         += spi_s3c24xx.o
 obj-$(CONFIG_MCFQSPI)             += mcf_qspi.o
 obj-$(CONFIG_DS1305)              += DS1305RTC.o
+obj-$(CONFIG_SPI_XILINX)               += xilinx_spi.o
 #    ... add above this line ...
 # SPI protocol drivers (device/link on bus)
+obj-$(CONFIG_SPI_SPIDEV)                += spidev.o
 #    ... add above this line ...

 # SPI slave controller drivers (upstream link)
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/drivers/spi/spidev.c petalinux-svn/software/linux-2.6.x-
petalogix/drivers/spi/spidev.c
--- petalinux-svn.orig/software/linux-2.6.x-petalogix/drivers/spi/spidev.c
      2009-03-02 11:09:02.000000000 +0100
+++ petalinux-svn/software/linux-2.6.x-petalogix/drivers/spi/spidev.c   2009-
03-02 11:20:39.000000000 +0100
@@ -37,6 +37,7 @@

 #include <asm/uaccess.h>

+typedef unsigned int* uintptr_t;

 /*
  * This supports acccess to SPI devices using normal userspace I/O calls.
@@ -52,9 +53,11 @@
  * particular SPI bus or device.
  */
 #define SPIDEV_MAJOR               153   /* assigned */
-#define N_SPI_MINORS               32    /* ... up to 256 */
+//#define N_SPI_MINORS             32    /* ... up to 256 */
+#define N_SPI_MINORS               8     /* ... up to 256 */

-static unsigned long   minors[N_SPI_MINORS / BITS_PER_LONG];
+//static unsigned long minors[N_SPI_MINORS / BITS_PER_LONG];
+static unsigned long   minors[1];


 /* Bit masks for spi_device.mode management.  Note that incorrect
@@ -561,7 +564,7 @@ static int spidev_probe(struct spi_devic
      struct spidev_data      *spidev;
      int               status;
      unsigned long           minor;
-
+
      /* Allocate driver data */
      spidev = kzalloc(sizeof(*spidev), GFP_KERNEL);
      if (!spidev)
@@ -584,9 +587,10 @@ static int spidev_probe(struct spi_devic

            spidev->devt = MKDEV(SPIDEV_MAJOR, minor);
            dev = device_create(spidev_class, &spi->dev, spidev->devt,
-                           spidev, "spidev%d.%d",
+                           /*spidev,*/ "spidev%d.%d",
                            spi->master->bus_num, spi->chip_select);
            status = IS_ERR(dev) ? PTR_ERR(dev) : 0;
+           dev_dbg(&spi->dev, "spidev%d.%d created\n", spi->master->bus_num,
spi->chip_select);
      } else {
            dev_dbg(&spi->dev, "no minor number available!\n");
            status = -ENODEV;
@@ -668,6 +672,7 @@ static int __init spidev_init(void)
            class_destroy(spidev_class);
            unregister_chrdev(SPIDEV_MAJOR, spidev_spi.driver.name);
      }
+
      return status;
 }
 module_init(spidev_init);
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/drivers/spi/xilinx_spi.c petalinux-svn/software/linux-2.6.x-
petalogix/drivers/spi/xilinx_spi.c
--- petalinux-svn.orig/software/linux-2.6.x-
petalogix/drivers/spi/xilinx_spi.c  2009-05-22 14:13:52.000000000 -0700
+++ petalinux-svn/software/linux-2.6.x-petalogix/drivers/spi/xilinx_spi.c
      2009-03-04 22:18:58.000000000 -0800
@@ -15,15 +15,12 @@
 #include <linux/init.h>
 #include <linux/interrupt.h>
 #include <linux/platform_device.h>
-
-#include <linux/of_platform.h>
-#include <linux/of_device.h>
-#include <linux/of_spi.h>
-
 #include <linux/spi/spi.h>
 #include <linux/spi/spi_bitbang.h>
 #include <linux/io.h>

+#include <linux/xilinx_devices.h>
+
 #define XILINX_SPI_NAME "xilinx_spi"

 /* Register definitions as per "OPB Serial Peripheral Interface (SPI)
(v1.00e)
@@ -147,11 +144,20 @@
            struct spi_transfer *t)
 {
      u8 bits_per_word;
+     u32 hz;
+     struct xilinx_spi *xspi = spi_master_get_devdata(spi->master);

      bits_per_word = (t) ? t->bits_per_word : spi->bits_per_word;
+     hz = (t) ? t->speed_hz : spi->max_speed_hz;
      if (bits_per_word != 8) {
-           dev_err(&spi->dev, "%s, unsupported bits_per_word=%d\n",
-                 __func__, bits_per_word);
+           dev_err(&spi->dev, "%s, unsupported bits_per_word=%d, bus
supports 8\n",
+                 __func__, bits_per_word);
+           return -EINVAL;
+     }
+
+     if (hz && xspi->speed_hz > hz) {
+           dev_err(&spi->dev, "%s, unsupported clock rate %uHz, bus uses
%uHz\n",
+                 __func__, hz, xspi->speed_hz);
            return -EINVAL;
      }

@@ -299,38 +304,32 @@
      return IRQ_HANDLED;
 }

-static int __init xilinx_spi_of_probe(struct of_device *ofdev,
-                             const struct of_device_id *match)
+static int __init xilinx_spi_probe(struct platform_device *dev)
 {
+     int ret = 0;
      struct spi_master *master;
      struct xilinx_spi *xspi;
-     struct resource r_irq_struct;
-     struct resource r_mem_struct;
-
-     struct resource *r_irq = &r_irq_struct;
-     struct resource *r_mem = &r_mem_struct;
-     int rc = 0;
-     const u32 *prop;
-     int len;
+     struct xspi_platform_data *pdata;
+     struct resource *r;

      /* Get resources(memory, IRQ) associated with the device */
-     master = spi_alloc_master(&ofdev->dev, sizeof(struct xilinx_spi));
+     master = spi_alloc_master(&dev->dev, sizeof(struct xilinx_spi));

      if (master == NULL) {
            return -ENOMEM;
      }

-     dev_set_drvdata(&ofdev->dev, master);
+     platform_set_drvdata(dev, master);
+     pdata = dev->dev.platform_data;

-     rc = of_address_to_resource(ofdev->node, 0, r_mem);
-     if (rc) {
-           dev_warn(&ofdev->dev, "invalid address\n");
+     if (pdata == NULL) {
+           ret = -ENODEV;
            goto put_master;
      }

-     rc = of_irq_to_resource(ofdev->node, 0, r_irq);
-     if (rc == NO_IRQ) {
-           dev_warn(&ofdev->dev, "no IRQ found\n");
+     r = platform_get_resource(dev, IORESOURCE_MEM, 0);
+     if (r == NULL) {
+           ret = -ENODEV;
            goto put_master;
      }

@@ -342,84 +341,69 @@
      xspi->bitbang.master->setup = xilinx_spi_setup;
      init_completion(&xspi->done);
```

```
-       xspi->irq = r_irq->start;
-
-       if (!request_mem_region(r_mem->start,
-                       r_mem->end - r_mem->start + 1, XILINX_SPI_NAME)) {
-               rc = -ENXIO;
-               dev_warn(&ofdev->dev, "memory request failure\n");
+       if (!request_mem_region(r->start,
+                       r->end - r->start + 1, XILINX_SPI_NAME)) {
+               ret = -ENXIO;
                goto put_master;
        }

-       xspi->regs = ioremap(r_mem->start, r_mem->end - r_mem->start + 1);
+       xspi->regs = ioremap(r->start, r->end - r->start + 1);
        if (xspi->regs == NULL) {
-               rc = -ENOMEM;
-               dev_warn(&ofdev->dev, "ioremap failure\n");
-               goto release_mem;
-       }
-       xspi->irq = r_irq->start;
-
-       /* dynamic bus assignment */
-       master->bus_num = -1;
-
-       /* number of slave select bits is required */
-       prop = of_get_property(ofdev->node, "xlnx,num-ss-bits", &len);
-       if (!prop || len < sizeof(*prop)) {
-               dev_warn(&ofdev->dev, "no 'xlnx,num-ss-bits' property\n");
+               ret = -ENOMEM;
+               goto put_master;
+       }
+
+       ret = platform_get_irq(dev, 0);
+       if (ret < 0) {
+               ret = -ENXIO;
                goto unmap_io;
        }
-       master->num_chipselect = *prop;
+       xspi->irq = ret;
+
+       master->bus_num = pdata->bus_num;
+       master->num_chipselect = pdata->num_chipselect;
+       xspi->speed_hz = pdata->speed_hz;

        /* SPI controller initializations */
        xspi_init_hw(xspi->regs);

        /* Register for SPI Interrupt */
-       rc = request_irq(xspi->irq, xilinx_spi_irq, 0, XILINX_SPI_NAME, xspi);
-       if (rc != 0) {
-               dev_warn(&ofdev->dev, "irq request failure: %d\n", xspi->irq);
+       ret = request_irq(xspi->irq, xilinx_spi_irq, 0, XILINX_SPI_NAME, xspi);
+       if (ret != 0)
                goto unmap_io;
-       }

-       rc = spi_bitbang_start(&xspi->bitbang);
-     if (rc != 0) {
-           dev_err(&ofdev->dev, "spi_bitbang_start FAILED\n");
+     ret = spi_bitbang_start(&xspi->bitbang);
+     if (ret != 0) {
+           dev_err(&dev->dev, "spi_bitbang_start FAILED\n");
            goto free_irq;
      }

-     dev_info(&ofdev->dev, "at 0x%08X mapped to 0x%08X, irq=%d\n",
-                 (unsigned int)r_mem->start, (u32)xspi->regs, xspi->irq);
-
-     /* Add any subnodes on the SPI bus */
-     of_register_spi_devices(master, ofdev->node);
+     dev_info(&dev->dev, "at 0x%08X mapped to 0x%08X, irq=%d\n",
+                 r->start, (u32)xspi->regs, xspi->irq);

-     return rc;
+     return ret;

 free_irq:
      free_irq(xspi->irq, xspi);
 unmap_io:
      iounmap(xspi->regs);
-release_mem:
-     release_mem_region(r_mem->start, resource_size(r_mem));
 put_master:
      spi_master_put(master);
-     return rc;
+     return ret;
 }

-static int __devexit xilinx_spi_remove(struct of_device *ofdev)
+static int __devexit xilinx_spi_remove(struct platform_device *dev)
 {
      struct xilinx_spi *xspi;
      struct spi_master *master;
-     struct resource r_mem;

-     master = platform_get_drvdata(ofdev);
+     master = platform_get_drvdata(dev);
      xspi = spi_master_get_devdata(master);

      spi_bitbang_stop(&xspi->bitbang);
      free_irq(xspi->irq, xspi);
      iounmap(xspi->regs);
-     if (!of_address_to_resource(ofdev->node, 0, &r_mem))
-           release_mem_region(r_mem.start, resource_size(&r_mem));
-     dev_set_drvdata(&ofdev->dev, 0);
+     platform_set_drvdata(dev, 0);
      spi_master_put(xspi->bitbang.master);

      return 0;
@@ -428,42 +412,27 @@
 /* work with hotplug and coldplug */
 MODULE_ALIAS("platform:" XILINX_SPI_NAME);

-static int __exit xilinx_spi_of_remove(struct of_device *op)
```

```
-{
-     return xilinx_spi_remove(op);
-}
-
-static struct of_device_id xilinx_spi_of_match[] = {
-     { .compatible = "xlnx,xps-spi-2.00.a", },
-     { .compatible = "xlnx,xps-spi-2.00.b", },
-     {}
-};
-
-MODULE_DEVICE_TABLE(of, xilinx_spi_of_match);
-
-static struct of_platform_driver xilinx_spi_of_driver = {
-     .owner = THIS_MODULE,
-     .name = "xilinx-xps-spi",
-     .match_table = xilinx_spi_of_match,
-     .probe = xilinx_spi_of_probe,
-     .remove = __exit_p(xilinx_spi_of_remove),
+static struct platform_driver xilinx_spi_driver = {
+     .probe      = xilinx_spi_probe,
+     .remove     = __devexit_p(xilinx_spi_remove),
      .driver = {
-           .name = "xilinx-xps-spi",
+           .name = XILINX_SPI_NAME,
            .owner = THIS_MODULE,
      },
 };

 static int __init xilinx_spi_init(void)
 {
-     return of_register_platform_driver(&xilinx_spi_of_driver);
+     return platform_driver_register(&xilinx_spi_driver);
 }
 module_init(xilinx_spi_init);

 static void __exit xilinx_spi_exit(void)
 {
-     of_unregister_platform_driver(&xilinx_spi_of_driver);
+     platform_driver_unregister(&xilinx_spi_driver);
 }
 module_exit(xilinx_spi_exit);
+
 MODULE_AUTHOR("MontaVista Software, Inc. <source@mvista.com>");
 MODULE_DESCRIPTION("Xilinx SPI driver");
 MODULE_LICENSE("GPL");
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/include/linux/spi/spi.h petalinux-svn/software/linux-2.6.x-
petalogix/include/linux/spi/spi.h
--- petalinux-svn.orig/software/linux-2.6.x-petalogix/include/linux/spi/spi.h
      2007-03-30 04:57:33.000000000 +0200
+++ petalinux-svn/software/linux-2.6.x-petalogix/include/linux/spi/spi.h
      2009-03-02 10:43:22.000000000 +0100
@@ -680,4 +680,15 @@ spi_unregister_device(struct spi_device
            device_unregister(&spi->dev);
 }

+/* device driver data */
```

```
+static inline void spi_set_drvdata(struct spi_device *spi, void *data)
+{
+       dev_set_drvdata(&spi->dev, data);
+}
+
+static inline void *spi_get_drvdata(struct spi_device *spi)
+{
+       return dev_get_drvdata(&spi->dev);
+}
+
 #endif /* __LINUX_SPI_H */
diff -rupN petalinux-svn.orig/software/linux-2.6.x-
petalogix/include/linux/xilinx_devices.h petalinux-svn/software/linux-2.6.x-
petalogix/include/linux/xilinx_devices.h
--- petalinux-svn.orig/software/linux-2.6.x-
petalogix/include/linux/xilinx_devices.h  2009-05-22 14:13:12.000000000 -0700
+++ petalinux-svn/software/linux-2.6.x-
petalogix/include/linux/xilinx_devices.h  2009-05-18 23:44:56.000000000 -0700
@@ -18,11 +18,7 @@

 #include <linux/types.h>
 #include <linux/version.h>
-#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,15)
-#include <linux/device.h>
-#else
 #include <linux/platform_device.h>
-#endif

 /*- 10/100 Mb Ethernet Controller IP (XEMAC) -*/

@@ -51,14 +47,23 @@
 #define XEMAC_DMA_SIMPLE     2     /* simple 2 channel DMA */
 #define XEMAC_DMA_SGDMA            3     /* scatter gather DMA */

+/*- 10/100 Mb Ethernet Controller IP (XEMACLITE) -*/
+struct xemaclite_platform_data {
+     u32 tx_ping_pong;
+     u32 rx_ping_pong;
+     u8 mac_addr[6];
+};
+
 /*- 10/100/1000 Mb Ethernet Controller IP (XTEMAC) -*/

 struct xtemac_platform_data {
+#ifdef XPAR_TEMAC_0_INCLUDE_RX_CSUM
      u8 tx_dre;
      u8 rx_dre;
      u8 tx_csum;
      u8 rx_csum;
      u8 phy_type;
+#endif
      u8 dma_mode;
      u32 rx_pkt_fifo_depth;
      u32 tx_pkt_fifo_depth;
@@ -74,6 +79,7 @@
 #define XTEMAC_DMA_SIMPLE    2     /* simple 2 channel DMA */
 #define XTEMAC_DMA_SGDMA     3     /* scatter gather DMA */
+/* LLTEMAC platform data */
 struct xlltemac_platform_data {
      u8 tx_csum;
      u8 rx_csum;
@@ -86,23 +92,22 @@
      u32 ll_dev_fifo_irq;

      u8 mac_addr[6];
-};
+ };
+

 /* LocalLink TYPE Enumerations */
 #define XPAR_LL_FIFO    1
 #define XPAR_LL_DMA     2

 /*- SPI -*/
-
+
+/* SPI Controller IP */
 struct xspi_platform_data {
-     u32 device_flags;
-     u8 num_slave_bits;
+     s16 bus_num;
+     u16 num_chipselect;
+     u32 speed_hz;
 };

-/* Flags related to XSPI device features */
-#define XSPI_HAS_FIFOS       0x00000001
-#define XSPI_SLAVE_ONLY            0x00000002
-
 /*- GPIO -*/

 /* Flags related to XGPIO device features */

diff -rupN petalinux-svn.orig/software/petalinux-
dist/vendors/PetaLogix/common/common.mak petalinux-svn/software/petalinux-
dist/vendors/PetaLogix/common/common.mak
--- petalinux-svn.orig/software/petalinux-
dist/vendors/PetaLogix/common/common.mak  2009-01-15 18:08:10.000000000 +0100
+++ petalinux-svn/software/petalinux-dist/vendors/PetaLogix/common/common.mak
      2009-03-02 11:08:21.000000000 +0100
@@ -180,6 +180,7 @@ DEVICES +=      \
 endif

 I2C_MAJOR = 89
+SPIDEV_MAJOR = 153

 ifndef CONFIG_SYSTEM_ROOT_PASSWD
 CONFIG_SYSTEM_ROOT_PASSWD := "root"
@@ -231,6 +232,15 @@ ifdef CONFIG_I2C
      done
 endif #IIC

+# make SPI nodes if necessary
+# SPIDEV_MAJOR taken from spidev.c (see definition above),
+# max number of devices matches spidev.c
+ifdef CONFIG_SPI_SPIDEV
+     for i in 0 1 2 3 4 5 6 7; do \
+           touch $(ROMFSDIR)/dev/@spi-$$i,c,$(SPIDEV_MAJOR),$$i; \
+     done
+endif #SPI_SPIDEV
+
 else
      $(ROMFSINST) $(COMMON)/etc/rc/checkroot /etc/init.d/checkroot
      $(ROMFSINST) -s /etc/init.d/checkroot /etc/rc.d/S01checkroot
diff -rupN petalinux-svn.orig/software/petalinux-
dist/vendors/Xilinx/Spartan3E500-RevD/config.linux-2.6.x petalinux-
svn/software/petalinux-dist/vendors/Xilinx/Spartan3E500-RevD/config.linux-
2.6.x
--- petalinux-svn.orig/software/petalinux-dist/vendors/Xilinx/Spartan3E500-
RevD/config.linux-2.6.x 2009-02-02 11:55:59.000000000 +0100
+++ petalinux-svn/software/petalinux-dist/vendors/Xilinx/Spartan3E500-
RevD/config.linux-2.6.x 2009-02-05 19:31:09.000000000 +0100
@@ -1541,8 +1541,20 @@ CONFIG_XILINX_IIC=y
 #
 # SPI support
 #
-# CONFIG_SPI is not set
-# CONFIG_SPI_MASTER is not set
+CONFIG_SPI=y
+CONFIG_SPI_MASTER=y
+
+#
+# SPI Master Controller Drivers
+#
+CONFIG_SPI_BITBANG=y
+CONFIG_SPI_XILINX=y
+# CONFIG_MCFQSPI is not set
+
+#
+# SPI Protocol Masters
+#
+CONFIG_SPI_SPIDEV=y

 #
 # Dallas's 1-wire bus
```

Appendix C – Sample UCF File

```
############################################################################
## This system.ucf file is generated by Base System Builder based on the
## settings in the selected Xilinx Board Definition file. Please add other
## user constraints to this file based on customer design specifications.
############################################################################

Net sys_clk_pin LOC=c9;
Net sys_clk_pin IOSTANDARD = LVCMOS33;

## System level constraints
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 20000 ps;

## IO Devices constraints

#### Module RS232_DTE constraints

Net fpga_0_RS232_DTE_RX_pin LOC=U8; //DTE connector (J10)
#AVNET Net fpga_0_RS232_DTE_RX_pin LOC=R7;  //DCE connector (J9)
Net fpga_0_RS232_DTE_RX_pin IOSTANDARD = LVCMOS33;
Net fpga_0_RS232_DTE_TX_pin LOC=M13; //DTE connector (J10)
#AVNET Net fpga_0_RS232_DTE_TX_pin LOC=M14;  //DCE connector (J9)
Net fpga_0_RS232_DTE_TX_pin IOSTANDARD = LVCMOS33;

#### Module LEDs_8Bit constraints

Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<0> LOC=F9;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<0> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<1> LOC=E9;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<1> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<2> LOC=D11;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<2> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<3> LOC=C11;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<3> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<4> LOC=F11;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<4> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<5> LOC=E11;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<5> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<6> LOC=E12;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<6> IOSTANDARD = LVCMOS33;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<7> LOC=F12;
Net fpga_0_LEDs_8Bit_GPIO_d_out_pin<7> IOSTANDARD = LVCMOS33;

#### Module DIP_Switches_4Bit constraints

Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<0> LOC=N17 | PULLDOWN;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<0> IOSTANDARD = LVCMOS33;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<1> LOC=H18 | PULLDOWN;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<1> IOSTANDARD = LVCMOS33;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<2> LOC=L14 | PULLDOWN;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<2> IOSTANDARD = LVCMOS33;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<3> LOC=L13 | PULLDOWN;
Net fpga_0_DIP_Switches_4Bit_GPIO_in_pin<3> IOSTANDARD = LVCMOS33;

#### Directional buttons and rotary encoder

Net fpga_0_Buttons_west_pin LOC=D18 | PULLDOWN;
Net fpga_0_Buttons_west_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Buttons_east_pin LOC=H13 | PULLDOWN;
Net fpga_0_Buttons_east_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Buttons_north_pin LOC=V4 | PULLDOWN;
Net fpga_0_Buttons_north_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Buttons_south_pin LOC=K17 | PULLDOWN;
Net fpga_0_Buttons_south_pin IOSTANDARD = LVCMOS33;

Net fpga_0_Rotary_A_pin LOC=K18 | PULLUP;
Net fpga_0_Rotary_A_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Rotary_B_pin LOC=G18 | PULLUP;
Net fpga_0_Rotary_B_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Rotary_center_pin LOC=V16 | PULLDOWN;
Net fpga_0_Rotary_center_pin IOSTANDARD = LVCMOS33;


#### Module FLASH_16Mx8 constraints

Net fpga_0_FLASH_16Mx8_Mem_A_pin<31> LOC=h17;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<31> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<30> LOC=j13;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<30> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<29> LOC=j12;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<29> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<28> LOC=j14;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<28> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<27> LOC=j15;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<27> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<26> LOC=j16;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<26> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<25> LOC=j17;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<25> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<24> LOC=k14;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<24> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<23> LOC=k15;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<23> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<22> LOC=k12;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<22> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<21> LOC=k13;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<21> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<20> LOC=l15;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<20> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<19> LOC=l16;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<19> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<18> LOC=t18;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<18> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<17> LOC=r18;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<17> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<16> LOC=t17;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<16> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<15> LOC=u18;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<15> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<14> LOC=t16;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<14> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<13> LOC=u15;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<13> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<12> LOC=v15;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<12> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<11> LOC=t12;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<11> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<10> LOC=v13;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<10> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<9> LOC=v12;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<9> IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<8> LOC=n11;
Net fpga_0_FLASH_16Mx8_Mem_A_pin<8> IOSTANDARD = LVCMOS33;

# Data (DQ) Pins for Flash go through ADC/Flash Control
Net fpga_0_flash_adc_ctl_pin<6> LOC=p10;
Net fpga_0_flash_adc_ctl_pin<6> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<5> LOC=r10;
Net fpga_0_flash_adc_ctl_pin<5> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<4> LOC=v9;
Net fpga_0_flash_adc_ctl_pin<4> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<3> LOC=u9;
Net fpga_0_flash_adc_ctl_pin<3> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<2> LOC=r9;
Net fpga_0_flash_adc_ctl_pin<2> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<1> LOC=m9;
Net fpga_0_flash_adc_ctl_pin<1> IOSTANDARD = LVCMOS33;
Net fpga_0_flash_adc_ctl_pin<0> LOC=n9;
Net fpga_0_flash_adc_ctl_pin<0> IOSTANDARD = LVCMOS33;

Net fpga_0_FLASH_16Mx8_Mem_OEN_pin LOC=c18;
Net fpga_0_FLASH_16Mx8_Mem_OEN_pin IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_WEN_pin LOC=d17;
Net fpga_0_FLASH_16Mx8_Mem_WEN_pin IOSTANDARD = LVCMOS33;
Net fpga_0_FLASH_16Mx8_Mem_CEN_pin<0> LOC=d16;
Net fpga_0_FLASH_16Mx8_Mem_CEN_pin<0> IOSTANDARD = LVCMOS33;

#Net flash_CEN_pin<0> LOC=d16;
#Net flash_CEN_pin<0> IOSTANDARD = LVCMOS33;


Net fpga_0_FLASH_16Mx8_emc_ben_gnd_pin LOC=c17;
Net fpga_0_FLASH_16Mx8_emc_ben_gnd_pin IOSTANDARD = LVCMOS33;

#### Module DDR_SDRAM_32Mx16 constraints

Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clk_pin LOC=J5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clk_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clkn_pin LOC=J4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clkn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<0> LOC=P2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<1> LOC=N5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<2> LOC=T2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<2> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<3> LOC=N4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<3> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<4> LOC=H2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<4> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<5> LOC=H1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<5> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<6> LOC=H3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<6> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<7> LOC=H4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<7> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<8> LOC=F4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<8> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<9> LOC=P1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<9> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<10> LOC=R2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<10> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<11> LOC=R3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<11> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<12> LOC=T1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<12> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_BankAddr_pin<0> LOC=K6;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_BankAddr_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_BankAddr_pin<1> LOC=K5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_BankAddr_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CASn_pin LOC=C2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CASn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CKE_pin LOC=K3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CKE_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CSn_pin LOC=K4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_CSn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_RASn_pin LOC=C1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_RASn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_WEn_pin LOC=D1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_WEn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DM_pin<0> LOC=J1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DM_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DM_pin<1> LOC=J2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DM_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQS_pin<0> LOC=G3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQS_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQS_pin<1> LOC=L6;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQS_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQS_pin<1> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<0> LOC=H5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<0> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<1> LOC=H6;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<1> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<2> LOC=G5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<2> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<2> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<3> LOC=G6;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<3> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<3> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<4> LOC=F2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<4> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<4> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<5> LOC=F1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<5> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<5> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<6> LOC=E1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<6> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<6> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<7> LOC=E2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<7> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<7> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<8> LOC=M6;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<8> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<8> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<9> LOC=M5;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<9> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<9> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<10> LOC=M4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<10> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<10> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<11> LOC=M3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<11> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<11> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<12> LOC=L4;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<12> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<12> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<13> LOC=L3;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<13> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<13> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<14> LOC=L1;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<14> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<14> PULLUP;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<15> LOC=L2;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<15> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_32Mx16_DDR_DQ_pin<15> PULLUP;

#### Module Ethernet_MAC constraints

Net fpga_0_Ethernet_MAC_PHY_tx_clk_pin LOC=T7;
Net fpga_0_Ethernet_MAC_PHY_tx_clk_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_clk_pin LOC=V3;
Net fpga_0_Ethernet_MAC_PHY_rx_clk_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_crs_pin LOC=U13;
Net fpga_0_Ethernet_MAC_PHY_crs_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_dv_pin LOC=V2;
Net fpga_0_Ethernet_MAC_PHY_dv_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<0> LOC=V8;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<0> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<1> LOC=T11;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<1> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<2> LOC=U11;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<2> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<3> LOC=V14;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<3> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_col_pin LOC=U6;
Net fpga_0_Ethernet_MAC_PHY_col_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_er_pin LOC=U14;
Net fpga_0_Ethernet_MAC_PHY_rx_er_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_en_pin LOC=P15;
Net fpga_0_Ethernet_MAC_PHY_tx_en_pin IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<0> LOC=R11;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<0> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<1> LOC=T15;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<1> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<2> LOC=R5;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<2> IOSTANDARD = LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<3> LOC=T5;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<3> IOSTANDARD = LVCMOS33;

Net fpga_0_DDR_CLK_FB LOC=B9;
Net fpga_0_DDR_CLK_FB IOSTANDARD = LVCMOS33;

### Avnet added begin
Net fpga_0_DDR_CLK_FB TNM_NET = fpga_0_DDR_CLK_FB;
TIMESPEC TS_fpga_0_DDR_CLK_FB = PERIOD fpga_0_DDR_CLK_FB 10000 ps;

NET ddr_clk_90_s         TNM_NET = DDR_FEEDBACK_CLOCK_PHASE_90;
NET ddr_clk_90_n_s       TNM_NET = DDR_FEEDBACK_CLOCK_PHASE_270;
NET dlmb_port_BRAM_Clk   TNM_NET = OPB_CLOCK;
NET ddr_dev_clk_s        TNM_NET = DDR_DEVICE_CLOCK;
NET ddr_dev_clk_90_s     TNM_NET = DDR_DEVICE_CLOCK_PHASE_90;

TIMESPEC TS_OPB_TO_Device   = FROM OPB_CLOCK                   TO
DDR_DEVICE_CLOCK             TIG;
TIMESPEC TS_OPB_TO_Device90 = FROM OPB_CLOCK                   TO
DDR_DEVICE_CLOCK_PHASE_90    TIG;
TIMESPEC TS_Device_TO_OPB   = FROM DDR_DEVICE_CLOCK            TO OPB_CLOCK
TIG;
TIMESPEC TS_Device_TO_OPB90 = FROM DDR_DEVICE_CLOCK_PHASE_90   TO OPB_CLOCK
TIG;
TIMESPEC TS_OPB_TO_DDR      = FROM OPB_CLOCK                   TO
DDR_FEEDBACK_CLOCK_PHASE_90  TIG;
TIMESPEC TS_DDR_TO_OPB      = FROM DDR_FEEDBACK_CLOCK_PHASE_90 TO OPB_CLOCK
TIG;

# OFFSET constraints must be manually adjusted to account for the clock
phase.
OFFSET = IN -0.750 ns VALID 3.5 ns BEFORE fpga_0_DDR_CLK_FB TIMEGRP
DDR_FEEDBACK_CLOCK_PHASE_90 ;
#  clock cycle - 0.75 ns -  clock cycle, valid  clock cycle - DDR Tac
uncertainty (+/- 0.75 ns = 1.5ns)
OFFSET = IN -5.75 ns VALID 3.5 ns BEFORE fpga_0_DDR_CLK_FB TIMEGRP
DDR_FEEDBACK_CLOCK_PHASE_270 ;
#  clock cycle - 0.75 ns -  -  clock cycle, valid  clock cycle - DDR Tac
uncertainty (+/- 0.75 ns)

NET "*/DDR_DQ_I*"  IOBDELAY=NONE;
NET "*/DDR_DQ_O*"  IOBDELAY=NONE;
NET "*/DDR_DQ_T*"  IOBDELAY=NONE;
NET "*/DDR_DQS_I*" IOBDELAY=NONE;
NET "*/DDR_DQS_O*" IOBDELAY=NONE;
NET "*/DDR_DQS_T*" IOBDELAY=NONE;
### Avnet added end

NET "lcd_pin<6>" LOC = "R15" | IOSTANDARD = LVCMOS33;       # LCD Data 0
NET "lcd_pin<5>" LOC = "R16" | IOSTANDARD = LVCMOS33;       # LCD Data 1
NET "lcd_pin<4>" LOC = "P17" | IOSTANDARD = LVCMOS33;       # LCD Data 2
NET "lcd_pin<3>" LOC = "M15" | IOSTANDARD = LVCMOS33;       # LCD Data 3
NET "lcd_pin<2>" LOC = "M18" | IOSTANDARD = LVCMOS33;       # LCD E
NET "lcd_pin<1>" LOC = "L18" | IOSTANDARD = LVCMOS33;       # LCD RS
NET "lcd_pin<0>" LOC = "L17" | IOSTANDARD = LVCMOS33;       # LCD RW

# SPI Devices
# Added by: Matt Staniszewski, Spring 2009

# SPI Main Connections
#NET "fpga_0_spi_MISO_pin"    LOC = "N10" | IOSTANDARD = LVCMOS33;      #
MISO
NET "fpga_0_spi_MOSI_pin"     LOC = "T4"  | IOSTANDARD = LVCMOS33;      #
MOSI
NET "fpga_0_spi_SCK_pin"      LOC = "U16" | IOSTANDARD = LVCMOS33;      # SCK

# SPI Device Control Connections
NET "dac_clr_pin"       LOC = "P8"  | IOSTANDARD = LVCMOS33;      # DAC CLR
NET "amp_shdn_pin"      LOC = "P7"  | IOSTANDARD = LVCMOS33;      # Preamp
shutdown pin
#NET "flash_CEN_pin<0>" LOC = "D16" | IOSTANDARD = LVCMOS33;      # Flash
Chip Enable

# SPI DAC Connections
NET "fpga_0_spi_SS_pin<0>"    LOC = "N8"  | IOSTANDARD = LVCMOS33;      # DAC
CS

# SPI Preamp Connections
NET "fpga_0_spi_SS_pin<1>" LOC = "N7"     | IOSTANDARD = LVCMOS33;      #
Preamp CS
#NET "fpga_0_amp_dout_pin<0>" LOC = "E18" | IOSTANDARD = LVCMOS33;      #
Preamp data out

# SPI ADC Connections
NET "ad_conv_pin<0>"    LOC = "P11" | IOSTANDARD = LVCMOS33;      # ADC CS

#Flash/ADC Shared Pin
NET fpga_0_flash_adc_ctl_pin<7> LOC = "N10" | IOSTANDARD = LVCMOS33;
```

Appendix D - SPI User Application

```
/**********************************
 * spi.c
 **********************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/autoconf.h>
#include "spi.h"
#include "intc.h"
#include "lcd.h"

static unsigned int deviceMask;

int twosComplement(unsigned int num, int n)
{
   int num_2s = 0;
   int i;

   /* Function only works up to 32
      bits.  Return 0 if n is > 32 */
   if(n > 32)
      return 0;
   /* If the MSB of the number is a one
      then perform two's complement
      and negate value */
   else if(num & (0x1 << (n - 1)))
   {
      /* Take the complement of the number
       and add 1 to get the two's complement
       value */
      num_2s = ~num + 1;

      /* Zero out the upper bits from the
       nth to the 32nd bit */
      for(i = n; i < 32; i++)
       num_2s &= ~(0x1 << i);

      /* If the MSB of the number is a one
       then negate the two's complement
       value */
      if(num & (0x1 << (n - 1)))
       num_2s *= -1;
   }
   /* Positive number, which is the same as
      the two's complement equivalent */
   else
      num_2s = num;

   return num_2s;
}
```

```
void initSPI(void)
{
   /* Set all salve select lines high */
   SPISSR=~0;

   /* Set up SPI control register */
   SPICR=0x086;

   /* Enable SPI interrupts */
   SPIGIE=~0;

   /* Set up device mask, which is the status when all
      the SS lines are high */
   deviceMask=SPISSR;

   /* Enable the interrupt in Intc */
   /* SIE=XPAR_OPB_SPI_0_IP2INTC_IRPT_MASK; */
}

void clearSPI(void)
{
   /* Wait for the last transaction to finish */
   while(SPISSR != deviceMask);

   /* Set the "Tx FIFO Reset" and "Rx FIFO Reset" Bits */
   SPICR |= 0x60;

   /* Clear the "Tx FIFO Reset" and "Rx FIFO Reset" Bits */
   SPICR &= ~0x60;
}

void readADC(unsigned int adc[])
{
   unsigned int value = 0x00;

   /* Change the SPI control register clock phase (CPHA)
      bit to '1' to change the phase for the ADC */
   SPICR=0x08E;

   /* Manually trigger the AD_CONV signal to begin
      reading in data to the ADC*/
   AD_CONV_DD=0x0;
   AD_CONV=0x0;

   /* Wait for the last transaction to finish. Allows
      data to be converted. */
   while(SPISSR!=deviceMask);

   /* Manually trigger the AD_CONV signal again
      to begin output of converted data */
   AD_CONV=0x1;
   AD_CONV=0x0;

   /* Read in values from the ADC, seperate the
      two ADC A and B values and store them into
      an array */
   value = readSPI(4);
   adc[0] = ((value >> 16) & 0x00003FFF);
   adc[1] = ((value) & 0x00003FFF);

   /* Reset the clock phase for the SPI bus in
      case another device is being used, such as
      the DAC */
   SPICR=0x086;
}

int readSPI(unsigned int numBytes)
{
   int value = 0;

   /* Force the maximum send width to be
      four bytes */
   if(numBytes > 4)
      numBytes = 4;

   /* Wait for the last transaction to finish */
   while(SPISSR != deviceMask);

   /* Read in each byte from the Rx FIFO */
   for(; numBytes; --numBytes)

      /* Append the current byte to the LSB of the value */
      value = (value << 8) | (SPIDRR & 0x0FF);

   /* Return the final value read in from the SPI */
   return value;
}

void writeSPI(unsigned int device, unsigned int value, unsigned int numBytes)
{
   int i;

   /* Force the maximum send width to be
      four bytes */
   if(numBytes > 4)
      numBytes = 4;

   /* Wait for the last transaction to finish */
   while(SPISSR != deviceMask);

   /* Lower the SS line for the specified device */
   SPISSR &= ~device;

   /* Special case for writing to the amplifiers */
   if(device == SPI_AMP)
   {
      /* Send the same byte to the Tx FIFO four times
       to ensure the amps get the new settings */
      for(i = 0; i < 4; i++)
       SPIDTR = value;
   }
   /* Other cases */
   else
   {
      /* Put the bytes of the data into the Tx FIFO
       MSB first */
      for(; numBytes; --numBytes)
      {
       SPIDTR = (value >> ((numBytes - 1) * 8)) & 0xFF;
      }
   }

   /* Wait for data in the Tx FIFO to be transmitted
      by polling the status bit */
   for(;;)
   {
      if(SPISR & 0x04)
       break;
   }

   /* Raise the SS lines when finished */
   SPISSR=~0;
}

double getAdcVoltage(double adc_val, int gain)
{
   double vin;

   /* Based on the User's Guide on the ADC (p. 76) */
   double divisor = 8192.0;

   /* Set the divisor for calculating the ADC
      voltage based on the given gain setting */
   switch(gain)
   {
      case 0x1: divisor *= -1.0; break;
      case 0x2: divisor *= -2.0; break;
      case 0x3: divisor *= -5.0; break;
      case 0x4: divisor *= -10.0; break;
      case 0x5: divisor *= -20.0; break;
      case 0x6: divisor *= -50.0; break;
      case 0x7: divisor *= -100.0; break;
      default: break;
   }

   /* Calculate ADC input voltage based on
      the formula in the User's Guide (p. 76) */
   vin = ((adc_val / divisor) * 1.25) + 1.65;

   /* Send input voltage value back */
   return vin;
}

int main(int argc, char *argv[])
{
   unsigned int adc[2] = {0x00, 0x00};
   unsigned int gain_a, gain_b;

   /* Initialize the SPI and LCD devices and clear
      the SPI bus */
   initSPI();
   clearSPI();
   initLcd();

   /* Set the flash ADC control GPIO to an output
      and enable the ADC while disabling the flash by
      setting the value to '1' */
   FLASH_ADC_SEL_DD=0x00;
   FLASH_ADC_SEL=0x01;

   /* Print help message if incorrect number of arguments */
   if(argc < 3 || argc > 4)
   {
      printf("\nusage: spi <device> <options>\n");
      printf("\nCurrently supported devices:\n\n");
      printf("DAC_A <value>\nDAC_B <value>\nDAC_C <value>\nDAC_D
<value>\nDAC_ALL <value>\nADC_ALL <gain a> <gain b>\n\n");
      printf("Possible DAC <value> entries:\n\n0 - 4095\n\n");
      printf("Possible ADC <gain a> <gain b> entries:\n\nx1, x2, x5, x10,
x20, x50, x100 (all values negative!)\n\n");

   }
   else
   {
      /* Run command based on arguments */
      if(strcmp(argv[1], "DAC_A") == 0 && argc == 3)
       /* Send value to DAC A */
       sendDAC(DAC_A,atoi(argv[2]));
      else if(strcmp(argv[1], "DAC_B") == 0 && argc == 3)
       /* Send value to DAC B */
       sendDAC(DAC_B,atoi(argv[2]));
      else if(strcmp(argv[1], "DAC_C") == 0 && argc == 3)
       /* Send value to DAC C */
       sendDAC(DAC_C,atoi(argv[2]));
      else if(strcmp(argv[1], "DAC_D") == 0 && argc == 3)
       /* Send value to DAC D */
       sendDAC(DAC_D,atoi(argv[2]));
      else if(strcmp(argv[1], "DAC_ALL") == 0 && argc == 3)
       /* Send value to all DACs */
       sendDAC(DAC_ALL,atoi(argv[2]));
      else if(strcmp(argv[1], "ADC_ALL") == 0 && argc == 4)
      {
       /* ADC, set gain settings with those entered */
       if(strcmp(argv[2], "x1") == 0)
          gain_a = x1;
       else if(strcmp(argv[2], "x2") == 0)
          gain_a = x2;
       else if(strcmp(argv[2], "x5") == 0)
          gain_a = x5;
       else if(strcmp(argv[2], "x10") == 0)
          gain_a = x10;
       else if(strcmp(argv[2], "x20") == 0)
          gain_a = x20;
       else if(strcmp(argv[2], "x50") == 0)
          gain_a = x50;
       else if(strcmp(argv[2], "x100") == 0)
          gain_a = x100;
       else
```

```
            gain_a = x1;

         if(strcmp(argv[3], "x1") == 0)
            gain_b = x1;
         else if(strcmp(argv[3], "x2") == 0)
            gain_b = x2;
         else if(strcmp(argv[3], "x5") == 0)
            gain_b = x5;
         else if(strcmp(argv[3], "x10") == 0)
            gain_b = x10;
         else if(strcmp(argv[3], "x20") == 0)
            gain_b = x20;
         else if(strcmp(argv[3], "x50") == 0)
            gain_b = x50;
         else if(strcmp(argv[3], "x100") == 0)
            gain_b = x100;
         else
            gain_b = x1;

         /* For safety, assert the CPHA bit in the
            SPI Control Register before sending values
            to the amps */
         SPICR=0x08E;

         /* Send settings to amp */
         sendAmp(gain_a,gain_b);

         /* Reset SPI Control Register */
         SPICR=0x086;

         /* Read value in from the ADC */
         readADC(adc);

         /* Output ADC A information to LCD */
         writeLcd(LCD_CMD, 0x01);
         printLcd("A:");
         display_num(getAdcVoltage(twosComplement(adc[0], 14), 0x2));
         printLcd("v (");
         display_num(twosComplement(adc[0], 14));
         printLcd(")");

         /* Output ADC B information to LCD */
         writeLcd(LCD_CMD, 0xC0);
         printLcd("B:");
         display_num(getAdcVoltage(twosComplement(adc[1], 14), 0x3));
         printLcd("v (");
         display_num(twosComplement(adc[1], 14));
         printLcd(")");
        }
        /* Run command based on arguments */
        else
        {
         printf("\nusage: spi <device> <options>\n");
         printf("\nCurrently supported devices:\n\n");
         printf("DAC_A <value>\nDAC_B <value>\nDAC_C <value>\nDAC_D
<value>\nDAC_ALL <value>\nADC_ALL <gain a> <gain b>\n\n");
         printf("Possible DAC <value> entries:\n\n0 - 4095\n\n");
```

```
       printf("Possible ADC <gain a> <gain b> entries:\n\nx1, x2, x5, x10,
x20, x50, x100 (all values negative!)\n\n");
      }
   }

   /* Reset flash ADC control select to enable
      flash and disable ADC */
   FLASH_ADC_SEL=0x00;

   return 0;
}

/******************************************
 * spi.h
 ******************************************/

#ifndef SPI_H
#define SPI_H

/* Amplifier gain settings
   (all values are negative!) */
#define x1        0x01
#define x2        0x02
#define x5        0x03
#define x10       0x04
#define x20       0x05
#define x50       0x06
#define x100      0x07

/* Regiser settings */
#define FLASH_ADC_SEL_DD   (*(volatile unsigned long *)(0x42000000+0x04))
#define FLASH_ADC_SEL      (*(volatile unsigned long *)(0x42000000+0x00))

#define AD_CONV_DD         (*(volatile unsigned long *)(0x41900000+0x04))
#define AD_CONV            (*(volatile unsigned long *)(0x41900000+0x00))

#define     SPICR          (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x60))
#define     SPISR          (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x64))
#define     SPIDTR               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x68))
#define     SPIDRR               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x6C))
#define     SPISSR               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x70))
#define     SPIGIE               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x1C))
#define     SPIISR               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x20))
#define     SPIIER               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x28))
#define     SPIRXF               (*(volatile unsigned long
*)(CONFIG_XILINX_SPI_0_BASEADDR+0x78))

/* Defined constants and macros */
#define     SPI_DAC     0x01
```

```
#define     SPI_AMP     0x02
#define     SPI_ADC     0x04

#define DAC_A 0x00
#define DAC_B 0x01
#define DAC_C 0x02
#define DAC_D 0x03
#define DAC_ALL 0x0F

#define     sendDAC(n,val)          writeSPI(SPI_DAC, 0x00300000 |
(((n)&0x0F)<<16) | (((val)&0x0FFF)<<4), 4)
#define     sendAmp(a,b)            writeSPI(SPI_AMP, (((b)&0x07)<<4) |
((a)&0x07), 1)

/* Function prototypes */
void initSPI(void);
void clearSPI(void);
int readSPI(unsigned int numBytes);
void writeSPI(unsigned int device, unsigned int value, unsigned int
numBytes);
void readADC(unsigned int adc[]);
double getAdcVoltage(double adc_val, int gain);

#endif

/****************************
 * intc.h
 ****************************/

#ifndef INTC_H
#define INTC_H

/* Register Settings */
#define     ISR         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x00))
#define     IPR         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x04))
#define     IER         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x08))
#define     IAR         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x0C))
#define     SIE         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x10))
#define     CIE         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x14))
#define     IVR         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x18))
#define     MER         (*(volatile unsigned long
*)(XPAR_OPB_INTC_0_BASEADDR+0x1C))

/* Defined Macros */
#define     initIntc()  {MER=0x03;}
#define     pendingIntc(x)    ((IAR=IPR&(x)) | IPR&(x))     /* Check for
pending interrupt and acknowledge */

#endif
```

```
/*****************************************
 * lcd.c
 *****************************************/

#include <linux/autoconf.h>
#include "lcd.h"

int readLcdStatus()
{
   int i, r;

   /* Set LCD data lines to inputs */
   LCD_DD = LCD_DATA;                                       // Set data lines
to inputs

   /* First read cycle */
   LCD = LCD_RW;
   LCD = LCD_RW | LCD_E;   /* ~125ns */
   r = (LCD & LCD_DATA);   /* ~125ns (E high for ~250ns) */
   for(i = 0; i < 5; ++i)  /*  Wait ~750ns */
      LCD = LCD_RW;

   /* Second read cycle */
   r <<= 4;
   LCD = LCD_RW | LCD_E;   /* ~125ns */
   r |= (LCD & LCD_DATA);  /*  ~125ns (E high for ~250ns) */
   for(i = 0; i < 5; ++i)  /*  Wait ~750ns */
      LCD = LCD_RW;

   /* Set LCD back to output */
   LCD_DD = 0;

   /* Send back status byte */
   return r & 0xFF;
}

void sendLcd(int rs, int d)
{
   int i;

   /* First write cycle */
   LCD = rs;
   LCD = rs | LCD_E | d;   /* ~125ns */
   LCD = rs | LCD_E | d;   /* ~125ns total E high for ~250ns */
   LCD = rs | d;     /* ~125ns Hold time */
   for(i = 0; i < 5; ++i)  /*  Wait ~750ns */
      LCD = rs;
}

void returnLcdHome()
{
   /* Send command to return the LCD cursor to the
      home position */
   sendLcd(LCD_CMD, 0x02);
}

void writeLcd(int rs, int value)
```

```
{
   int i, d;

   if(rs)
      rs = LCD_RS;

   /* Wait for the LCD to not be busy */
   while(readLcdStatus() & LCD_BUSY);

   /* Send upper nibble */
   sendLcd(rs, (value >> 4) & LCD_DATA);

   /* Send lower nibble */
   sendLcd(rs, value & LCD_DATA);

   for(i = 0; i < 10; ++i) LCD = 0; /* Wait ~1000ns */
}

void initLcd()
{
   int i;

   /* Set LCD data lines to outputs */
   LCD_DD = 0;

   for(i = 0; i < 20 * 5500; ++i) LCD = 0;   /* ~20ms delay */

   /* Set to 8-bit operation */
   sendLcd(LCD_CMD, 0x03);
   for(i = 0; i < 500; ++i) LCD=0;  /*  ~91us delay */

   /* Set to 8-bit operation */
   sendLcd(LCD_CMD, 0x03);
   for(i = 0; i < 500; ++i) LCD=0;  /*  ~91us delay */

   /* Set to 8-bit operation */
   sendLcd(LCD_CMD, 0x03);
   for(i = 0; i < 500; ++i) LCD=0;  /*  ~91us delay */

   /* Set to 4-bit operation */
   sendLcd(LCD_CMD, 0x02);
   for(i = 0; i < 500; ++i) LCD=0;  /* ~91us delay */

   /* Function Set */
   writeLcd(LCD_CMD, 0x28);

   /* Display On */
   writeLcd(LCD_CMD, 0x0C);

   /* Clear Screen */
   writeLcd(LCD_CMD, 0x01);

   /* Set Entry Mode */
   writeLcd(LCD_CMD, 0x06);
}

void printLcd(char *msg)
```

```
{
   /* Write out the message a character
      at a time until msg points to NULL */
   while(*msg)
      writeLcd(LCD_CHAR, *(msg++));
}

/* display_num: displays the given number up to
 * 7 digits on the LCD */
void display_num(double n)
{
   int i;

   /* Write out negative sign if needed */
   if (n < 0)
   {
      writeLcd(LCD_CHAR, '-');
      n *= -1;
   }

   /* Convert each digit for up to 7 digits into
      a character to output */
   unsigned char millions = ((int)n / 1000000) + 48;
   unsigned char hun_thousands = (((int)n % 1000000) / 100000) + 48;
   unsigned char ten_thousands = (((int)n % 100000) / 10000) + 48;
   unsigned char thousands = (((int)n % 10000) / 1000) + 48;
   unsigned char hundreds = (((int)n % 1000) / 100) + 48;
   unsigned char tens = (((int)n % 100) / 10) + 48;
   unsigned char ones = ((int)n % 10) + 48;
   unsigned char tenths;
   unsigned char hundredths;

   /* Calculate the decimal remainder by subtracting
      the other places from the inputted number */
   double decimal = n - (millions - 48) - (hun_thousands - 48) -
(ten_thousands - 48)
      - (thousands - 48) - (hundreds - 48) - (ones - 48);

   /* If a decical portion exists, calculate
      the tenths and hundredths values */
   if(decimal > 0)
   {
      tenths = (int)(decimal * 10) + 48;
      hundredths = ((int)(decimal * 100) % 10) + 48;
   }

   /* Write out the millions digit if > 0 */
   if (n >= 1000000)
      writeLcd(LCD_CHAR, millions);

   /* Write out the hundred thousands digit if > 0 */
   if (n >= 100000)
      writeLcd(LCD_CHAR, hun_thousands);

   /* Write out the ten thousands digit if > 0 */
   if (n >= 10000)
      writeLcd(LCD_CHAR, ten_thousands);
   /* Write out the thousands digit if > 0 */
   if (n >= 1000)
      writeLcd(LCD_CHAR, thousands);

   /* Write out the hundreds digit if > 0 */
   if (n >= 100)
      writeLcd(LCD_CHAR, hundreds);

   /* Write out the tens digit > 0 */
   if (n >= 10)
      writeLcd(LCD_CHAR, tens);

   /* Write out the ones digit */
   writeLcd(LCD_CHAR, ones);

   /* Check if there's a decimal and that
      it is made up of valid number characters */
   if(decimal > 0 && tenths > 47 && tenths < 58)
   {
      /* Write out the decimal to the
       hundredths place */
      writeLcd(LCD_CHAR, '.');
      writeLcd(LCD_CHAR, tenths);
      writeLcd(LCD_CHAR, hundredths);
   }
}

/***************************************
 * lcd.h
 ***************************************/

#ifndef LCD_H
#define LCD_H

/* Register Settings */
#define     LCD         (*(volatile unsigned long *)(0x41800000+0x00))
#define     LCD_DD            (*(volatile unsigned long *)(0x41800000+0x04))
#define     LCD_DATA    0x0F
#define     LCD_E       0x10
#define     LCD_RS            0x20
#define     LCD_RW            0x40
#define     LCD_BUSY    0x80
#define     LCD_CMD           0x00
#define     LCD_CHAR    LCD_RS

/* Function Prototypes */
void initLcd();
void writeLcd(int rs, int value);
void printLcd(char *msg);
void display_num(double n);

#endif
```

Appendix E – Analysis of Senior Project Design

Summary of Functional Requirements

For this project, I first developed the best environment to work with uClinux under. Next, I developed a SPI driver for the Spartan 3E Starter Kit in uClinux. This driver will be used in future extensions of the SuPER project to convert the current control system from a laptop to a Spartan 3E FPGA to save power. Finally, I wrote a basic user application in uClinux to implement the DAC and ADC on the starter kit for testing.

Primary Constraints

For the project, the primary constraint was access to information. Much of the work surrounding SPI driver development in uClinux is very bare, and it can be hard to find a driver that works for your exact system. I was fortunate enough to have some great people on uclinux-microblaze mailing list to help me work on the SPIDEV driver throughout my project.

Another difficulty was finding a way around the shared pin between the flash data bus and the SPI core. This caused bus contention whenever the two devices were turned on. The solution was to create a custom IP core that forced Xilinx EDK to allow both devices to access the pin through a MUX. This MUX was controlled by a manual select controlled by a GPIO.

Economics

There was no cost for my project since the parts required for development had been purchased already for previous SuPER projects. Everything was provided and returned at the end of the project.

The development time for the project turned out to be quite long, since much of the time was spent researching possible solutions to problems that occurred. Overall, the project took about 10 hours a week to finish on time. This was the approximate estimate given at the beginning of the quarter.

Environmental and Sustainability

The SuPER project will have an enormous impact on the environment in the third world. The project was developed as a way to provide renewable energy to those who currently lack the infrastructure and access to energy. Each station will provide a supply of solar energy to power a small village that would otherwise not have power. The part I have worked on has helped increase the amount of energy available to the people who will use the system. By replacing the laptop with the Spartan 3E Starter Kit I developed my driver for, the consumption of the control system will drop to less than 3W, allowing the rest of the energy previously taken by the laptop to be distributed among the micro grid within the village.

Social and Political

The SuPER project will greatly affect the people in the third world. It will give them access to a resource they did not have before, providing energy to villages all over the world.

Development

During this project, I learned a great deal of new knowledge about the Linux kernel. Working with the uClinux kernel and patching the SPIDEV driver into my build taught me a great deal about how the kernel works and how drivers are written in Linux. I also expanded my knowledge of Xilinx ISE and EDK by writing my own IP core, which I had not done before.