EE 371 Project 4

Building and Working with a Simple Microprocessor….

University of Washington - Department of Electrical Engineering

James K. Peckol

Introduction:

In this fourth project, we will work with some new tools and concepts as we design and incorporate a NIOS II microprocessor onto our gate array. Building on the core hardware, we will add support for general purpose I/O to enable information exchange between the processor and functionality both within and external to the FPGA. Our first piece of functionality will be the scanner subsystem we developed in the previous project. With the processor implemented, operating, and integrated with the scanner subsystem, we will put our knowledge of C to work by developing several simple programs, running them on our processor to generate commands to our scanner subsystem.

Prerequisites:

Familiarity with the Quartus development environment and the *Signal Tap* (nope still ain't *Spinal Tap*) logic analyzer will definitely help. A continued willingness to learn and to explore. No birra or Barbera del Monferrato until the project is completed and you can turn on a several LEDs using switches on the processor and print the requisite *hello world* and toggle a few lines as you run several C programs on our new microprocessor. Hey, these are all still good.

Cautions and Warnings:

Now that we are working with a microprocessor, always make certain that the cables connecting the PC to the DE1-SoC board are not twisted and don't have any knots that might get confused with 0's in your machine code. If they are twisted or tangled, the compiled C instructions might get reversed as they are downloaded into the target and your program will run backwards. Also, try to keep your DE1-SoC board lower than your PC thus making it easier for the electrons to get to your board and making the data transfer much faster. However, this will not affect the speed at which your program runs. Also be careful that you do not get the data moving at too high of a rate otherwise it may continue on past your target system. If that does occur, try adding code to indicate that the system is to perform no operation (NOPs) in critical places in the stream. These can be lost with little damage.

Background:

Have some knowledge of the C language and ability to develop simple C programs. Have a working understanding of basic methods of system I/O such as switches and LEDs.

Objectives:

The major objectives of this project include:

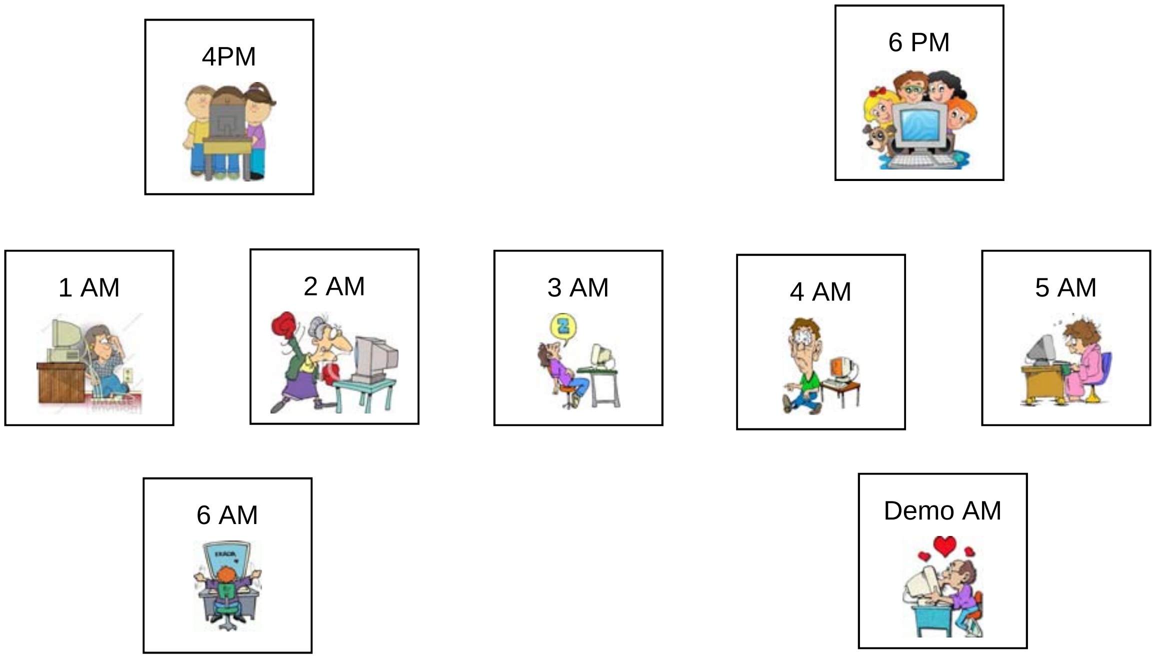
- Design, implement, and test a simple microprocessor on the DE1-SoC board FPGA.
- Design, implement, and test a fundamental I/O interface to the microprocessor.
- Add a basic data buffer to each scanner subsystem.

- Begin the integration our scanner subsystem with the microprocessor.
- Use our knowledge of the C language to develop and execute some simple C programs to put our new microprocessor to work.
- Build a foundation for the final project.

Designing and Building a Microprocessor and C Application:

For the first part of the project, we will work with a new Quartus tool called Qsys to design and implement a simple microprocessor based upon the Altera NIOS II core. We will then use the Eclipse based NIOS II IDE to develop several C applications that will run on that processor.

Basic Microprocessor Development Cycle



Building a Basic Microprocessor

For this first part of the project, we will focus on designing and implementing the core microprocessor system. Our primary tool will be the Altera Qsys system integration tool that is intended to support the design of digital hardware systems containing a variety of different digital components.

To facilitate getting started with this tool, we have placed several relevant documents on the class web page, under ***documentation/NIOS, Tutorials, UsersGuides, Handbooks***. Note that the Tutorials have nothing to do with King Tut or his tutor, although it may have had roots in Greek mythology.

Note: You do not have to download any additional Altera software for this project

Find the document *NIOSii_hardware_tutorial.pdf*. Read through it, then follow the instructions, step by step, to build and download a NIOS II based system to the DE1-SoC board. Pay attention to the note above this paragraph.

We have the following points of modification and clarification for the tutorial.

Page 1-9: Start the Quartus system and create a new project. Rather than working with a bsd file as the top-level module, we will begin with an empty Verilog file. Add the file to the project and ensure that the name of the project and the top-level Verilog module are the same. It is not necessary to use the names suggested by the tutorial.

Proceed with the tutorial.

Page 1-11: ***Specify Target FPGA and Clock Settings***: the ***Clock Settings*** and ***Project Settings*** tabs are in the main window

Page 1-14: step 1, Select ***Embedded Processors***.

Page 1-23: steps 1-4, ***Generate the Qsys System***

This step generates or creates a Verilog file for your NIOS II processor. If this step is successful, select the ***Generate*** tab again then ***HDL Example***. The example is a template for the interface to your processor. You will need this to specify the I/O in your top-level module and when you assign pins for your system.

Page 1-24: ***Integrate the Qsys System into the Quartus II Project***

a. Create an instance of the NIOS II processor in your top-level module.
b. Add your processor's IP file to your project.

   Browse to your working directory and find the file:
   <workingFilesDirectory>/<myProcessorName>/synthesis/<myProcessorName>.qip
c. Specify your top-level module I/O. Use the information from the ***HDL Example*** template above to identify your module I/O.
d. Compile your project.
e. Assign pins to your project
f. Compile again and program the Cyclone V.
g. Skip ahead to page 1-32.

Developing the Software

Once you have successfully built the NIOS II processor, assigned the pins, and downloaded it to the DE1-SoC board, it's time to do the software part.

Page 1-32: ***Develop Software Using the NIOS II SBT for Eclipse***

Follow the steps to create the firmware and software package that you then run on the microprocessor that you have created and built on the Cyclone V FPGA, You have now achieved a fundamental goals of electrical engineering. You have managed to flash an LED….in fact many of them.

If you ever get the message that the IDE can't find or identify your target system, elaborate the error and scroll to the far right hand side of that page to the 'refresh connection' button. Select that and all should be good. If not, there is another problem of some kind.

Developing Additional Applications

When you have the first application (*Count Binary*) successfully running, you are ready to explore the Qsys system further.

Adding General Purpose I/O

1. Using the tutorial: *Introduction_to_Qsys_Tool.pdf* from the class web page, work through the tutorial to develop the *lights and switches* system and application. This project illustrates how to develop a basic GPIO (General Purpose I/O) input and output system for the processor. We will use this interface in this and the next project.

Adding Textual I/O

2. From the same environment, and using the NIOS II processor, create a new application. This time, select the template, *hello world small*.
3. Complete the *hello world small* project just as you did the *Count Binary* project. You will now have two systems and two applications.

Adding Textual I/O

4. When that is working, make the following modifications to the program:
   a. After printing *hello world…* to the console, the program waits for an input from the user. When the user enters the letter 'G' from the console, the system enters an infinite loop enabling you to control the LEDs from the switches as you did in the *lights and switches* project. You will need to use the 'alt' form of stdio for this project.
   b. In the infinite loop, the system will read the state of the switches. If SW0 is a '1' the system will complement the bits corresponding to SW1..SW7 and then illuminate the corresponding LEDs. Otherwise, it will illuminate the corresponding LEDs based upon the state of the switches.

Integrating the Scanner Subsystem – The First Steps

High Level Requirements

Overview

As we work towards our final gondola borne canal data collection subsystem, our next steps will be to incorporate a basic data buffer into each of our scanner subsystems and to use the NIOS II microprocessor to control the subsystem.

Requirements

The required system additions are as follows:

- A 10 entry, byte wide data buffer is to be incorporated into each scanner subsystem.

  Look at the program hal0.v on the class web page for an idea as to how to design your data buffer.

  As test data, use the address of each location in the buffer as the data to be entered into that location, e.g. 0→0, 1→1, 2→2, …9→9.
- The microprocessor shall receive and interpret the following requests from the scanner subsystem:
  - ✓ *ReadyToTransfer* from the currently active scanner.
- The microprocessor shall generate the following commands to the scanner subsystem:
  - ✓ *StartScanning* to begin the data collection process.
  - ✓ *Transfer* to begin sending collected data to the canal collection station.

To design and build the interface between the scanner subsystem and the processor, review what you did in the *lights and switches* project.

Status and Data Display

- The state of all control and status lines for the active scanner shall be presented on the Red LEDs.
- For the current phase of the project, display the data being written to the data buffer during filming and the data being transmitted during transfer on the numeric LED display.
- For extra credit, read the data being transmitted during transfer and display it on the Eclipse console.

Deliverables:

A lab demo showing…

1. The *countBinary* design and working implementation that meets the specified requirements.
2. The *lights and switches* design and working implementation that meets the specified requirements.
3. The *hello world small* design and working implementation that prints the ‘hello world’ message to the console.
4. The *hello world small* design and working implementation extended to support, console I/O and the modified functionality of the *lights and switches* program as specified in the requirements.
5. An integrated and working NIOS II - Climate data collection subsystem design that meets all of the specified requirements.

A short lab report containing:

1. An abstract, introduction, general description of your hardware and software design, discussion of any problems that you had with the development of the project, a summary, and a conclusion.
2. The annotated Verilog and C source code for all applications both on the DE1-SoC board and on the NIOS II processor.