EE 371 Lab 5

Extending a Simple Microprocessor – Adding an Application

University of Washington - Department of Electrical Engineering

James K. Peckol

Introduction:

Life in the undersea habitat opens up unbelievable new vistas and unseen wonders to explore; but on rare occasions can also get a wee bit boring. What to do, what to do, what to do? Going for a walk is a bit of a challenge…Wait, we have a large fiber optic network down here and a way to connect to it that we can use when it is idle. Thank you very much Dr. Delaney, UW oceanography professor, this is brilliant.

Check it out it is totally awesome:

https://ooinet.oceanobservatories.org and

https://ooinet.oceanobservatories.org/cameras

Now we can communicate with other habitats down here and cooperate to make incredible undersea observations and discoveries….then in our leisure time….  Hey, let's do it.

In this fifth and final project, we will design and implement an application that will run on our microprocessor and will communicate and interact with an application on another microprocessor in another habitat. That application will implement a network-based game that is played between two or more opponents, each on their own system. The opposing teams will choose the game and the rules of play. Possible games can be a variant on Mines or Mine Sweeper, Space Invaders, Battleship, Chasing through a maze, or an approved game of your team's choice.

Dr. Delany is that you in there?

In this fifth project, we will continue work with the Qsys tools and concepts as we extend our basic communications system for the bathysphere and ocean bottom habitat system that we began in an earlier project. To that end, we will develop and test a distributed application that will support interactive communication over our asynchronous serial network.

Again, of note, the processor that we are building and working with is very similar to that which we have in our home or lab PCs.

Prerequisites:

Familiarity with the Quartus II, Qsys development environment, and the Signal Tap (nope still ain't *Spinal Tap*) logic analyzer will definitely help (note, this is not *Q* from Star Trek nor the Game of Thrones…nor the Greek god of the sky). A continued willingness to learn and to explore is good…keep that meat computer active. No beer until the project is completed – it confuses the meat computer – and you can turn on a several LEDs using switches on the processor, print the requisite *hello world* as you run several C programs on our new microprocessor, talk with the outside world, create a new form of indie rock, and have fun. Work with the fish, they're still in school too. Hey, these are all still cool.



Your Turn

Several images from

http://www.interactiveoceans.washington.edu/gallery

Cautions and Warnings:

Once again, 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 because by that point, the program is already at its destination. Is it true that if your program is a few bits short or your logic has a few bad bits that you can go to engineering stores and can buy some new ones using bit coins? Will WD40 help you loosen up any stuck bits?

Background:

Have some knowledge of the C language and ability to develop simple C programs. Have a working understanding of the Verilog HDL modeling and synthesis, intra and inter system timing and timing constraints, and basic methods of system I/O. Have a basic understanding of network fundamentals.

When debugging digital circuits, it is helpful to understand what each part of the circuit is doing and exactly when certain events are occurring. The amount of information can be huge, and often we would like to filter out as much of the unnecessary data as possible. Also, sometimes we are interested in timing measurements, other times we are more concerned with comparing the different states of various parts of the circuit. As we've learned, a logic analyzer can be a great help with all of this, and much more.

Microcomputer communications is a rapidly growing field with an ever-increasing number of applications, ranging from intra and inter exchanges on SOCs (system on a chip) to local PC networks and on to large-scale communication systems between cities and countries. Familiar schemes include SPI, I^2C, Bluetooth (developed in Professor Sahr's lab at the UW) CAN, MAP, USB….

Central to any communications between electronic devices is a *protocol* for transmitting and receiving information. A key point to understand is that a protocol is *not* simply a collection of binary 0's and 1's expressed by different voltage levels or the presence or absence of light or radio waves. A protocol is the *meaning* placed on those signals and that different protocols can apply at different levels of a network hierarchy.

Within a microprocessor, data is usually transferred in parallel form. While parallel communication is far more efficient than serial, it is not always as practical, and thus most communications between computers and other electronic devices not in the immediate vicinity of each other usually occurs in serial form. Of course having two different formats for communicating data can generate many occasions when data have to be converted from one form to the other.

Another important aspect of communication is ensuring that the data given to the user, following reception, contains no errors arising from such a transmission. Note, we do not guarantee no transmission errors, these happen; rather, at the end of the day, we guarantee the data to be correct. There are a variety of schemes by which this is accomplished: all begin with recognizing that a transmission error has occurred. We'll examine one, simple parity checking.

You decide to design and build the first prototype in two phases; sort of practicing selling into the real world – staged deliverables and all that. At the end of the first phase, you will have a stand-alone version with interface to a PC – your Eclipse console; at the end of the second, the game will connect over a serial network and interact with a second game.

Project:

General Description

The game can be a portable (with network support) modern version of one of the classic games such as Mines or Mine Sweeper (designed and written by former UW grad Tom Anderson), Space Invaders, Battleship, Chasing through a maze, or an approved game of your team's choice. Your new version is played in outer space between defenders of the Earth and invaders from the hidden undersea world, Xcz3oid (pronounced zoid – the X and the 3 are silent and the c and z are pronounced together somewhat like a sleeping snake).

The project is to be developed in two phases. The first phase deliverable will be the core game that will provide a text based display of weapons, positions, battle status, and other information.

The second phase will support connection to another game via a high-speed serial network and thereby permit play with a second (remote) player.

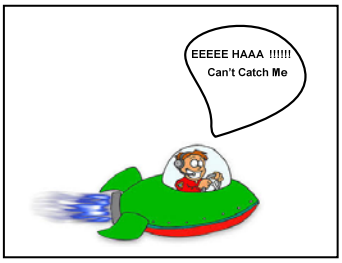
Creativity, low cost, flexibility, and compatibility with other systems are important goals.

General Requirements

You are building the prototype for a two-player game. Because play must involve multiple players or teams, you must ensure that another team(s) is/are designing and building the same game with the same communications protocol.

You have begun by doing a market survey/study to determine potential requirements for the new game. Your research shows that network based games are becoming increasingly popular.

Thus, in your game, each move will be transmitted over a network and the consequences of the move (such as destroying an opponent's weapon or spaceship) are returned as appropriate. For your first design, you decide that the link between games should operate at 9600 baud. The state of the game is to be held and displayed locally. Information to be displayed should include the player's turn, win or lose status, the current game status, intermediate states, etc.



Your research shows that a particularly popular feature is the ability to collect and store statistics on the game…shortest time, amount of enemy materiel destroyed, player's name, score, etc. All of this data should be stored in a RAM outside of the microprocessor.

To ensure compatibility between different games, of course, it is important that, at a minimum, all have the same display layout, and format. Further, all should support the same set of commands and communications protocol, interpret them the same way, and that all should support the same board layout(s). In the remote mode, it is important to ensure that the states of all of the boards are consistent at the start, end, and during each game.

Deliverables:

1. A full project report, from each team, as described on the class web page under *Workload and Grading*.
2. The annotated Verilog and C source code for the final applications both on the DE1-SoC board and on the PC.
3. Each project will have an associated creativity/complexity score that will range between 1-10. It is expected that your project will rate at least a 5. The complexity measure will be determined at the time of your demo and will be a multiplier on your demo score.
4. Answers to any questions above.
5. Other things that you deem to be important.
6. Anything that we haven't thought of.
7. Signature page, signed and stating that the report and its contents are solely your team's work and cites outside references for work that is not your own.