POPi

Design Document

Adam Lehenbauer
Alexander Robertson
Javier Coca
Yashket Gupta

Embedded Systems Design

March 26, 2006

1. OVERVIEW

POPi project main goal is to display a message (SMS) in the video screen of the XESS board, sent from a cellular telephone to an e-mail address. project designed to take advantage of both hardware and software resources. In consequence the design has been deviled in two equally important parts.

The Hardware components that will be required will include the FPGA and its peripherals. The FPGA will be implemented as the controller of the Video Terminal that will display the text messages. Also, as the controller of the SRAM peripherals. Finally the Ethernet peripheral that will give the communication with the server where the messages will be broadcasted. All the communication between the FPGA and the peripherals will be hosted by the OPB.

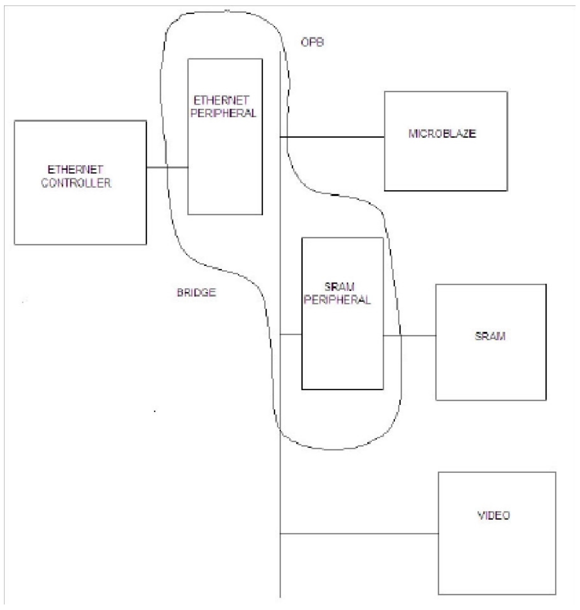


Figure 1. Hardware block diagram.

The Software that the project will require is a very significant part of its functionality. The two major "subsystems," the video controller and the Ethernet controller, and the software is the communication between them. The SRAM will be transparent to the software and appear only as a base address.

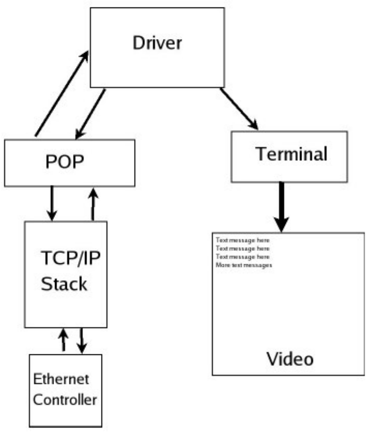


Figure 2. Software block diagram.

2. HARDWARE.

As mentioned, there are four major parts for the Hardware Part of our Design.

1. Ethernet Peripheral – It is basically a Protocol converter between the Ethernet and the OPB
   Figure 3a. Ethernet Controller time diagram.
   Figure 3b. Ethernet Controller pin diagram.

2. SRAM Peripheral – The same hardware we wrote for the Lab 6
   Figure 4a. Ethernet Time diagram.
   Figure 4b. Ethernet pin diagram.

3. BRIDGE – The two peripherals connected to the OPB form a BRIDGE that allows the data from the Ethernet Controller to be written on the SRAM where it is stripped of all the Headers and the Text Msg is extracted.

4. Video – We plan to use the Design of Lab 2 for our video with certain modifications.

2.1 Data path

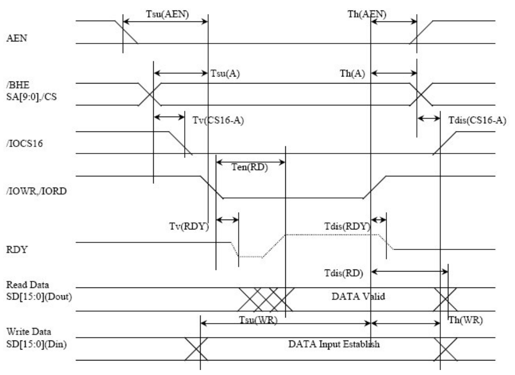
A. Data is written in the SRAM of the Ethernet controller by implementing a circular buffer – We need to define the optimal size for our buffer.

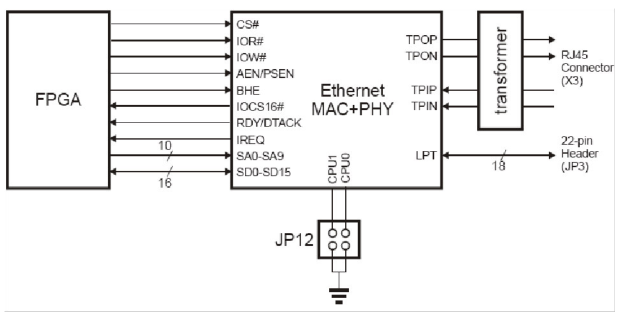
B. Once the Ethernet has the data it has to tell the Microblaze to get the data.

C. The Ethernet Peripheral should grab the data and write give it to the off chip SRAM.

D. Microblaze should run the Software written on the SRAM which strips the data from all the Headers and Derive the Text Msg and gives it to the Microblaze.

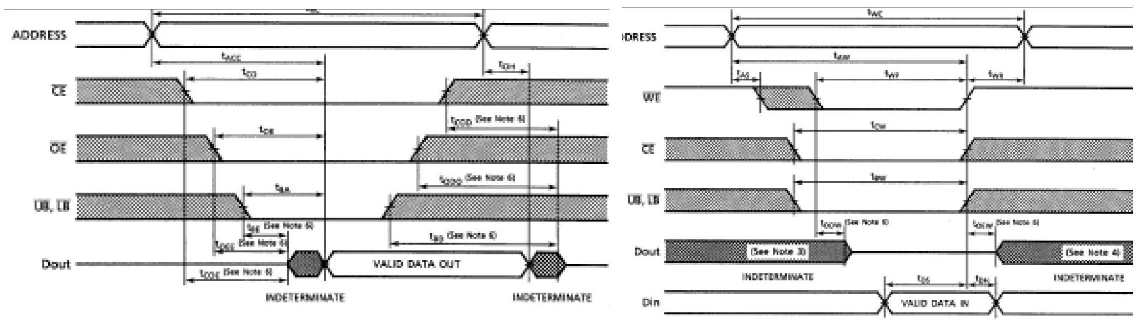
E. The Microblaze then displays the data to the Video.





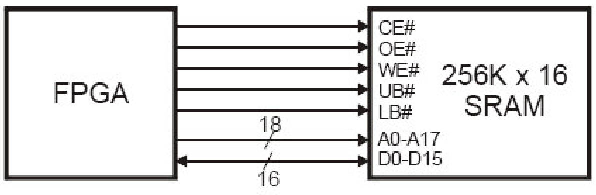
(a) *(b)*

Figure 3. Ethernet Controller diagram.



(a)

Read *Write*



(b)

Figure 3. SRAM diagram.

3. SOFTWARE

The software for this system, as mentioned, will consist of four distinct parts. For some of these parts we hope to be able to drop in third party software if it fits our needs. If not, we anticipate that our requirements can be satisfied without too much difficulty with something we can write ourselves.

The software pieces we will require are:

1. A driver.

2. POP client.

3. A TCP/IP stack.

4. A terminal.

3.1 Program Flow

A. The driver will run the whole system and communicate between the video subsystem and the network subsystem.

B. The POP client will poll the external mail server periodically to check for a new message.

C. The TCP/IP stack will communicate between the Ethernet controller and the POP client.

A. On the video side of the system, the terminal will handle displaying the messages on the screen.

3.2 Driver

The driver is the top level software construct in this system. It will call the POP client and wait for a response in the form of an ASCII character string (our message). When the driver receives a response from the POP client, it will call the terminal with that string as an argument and allow the terminal to display the string on the monitor.

Since the driver's job is essential a bridge between the two very different subsystems, it will not be very complex.

3.3 POP client

The POP client will be responsible for polling the off-site mail server at a fixed interval to retrieve new messages. The POP protocol itself is very simple and will probably require more work on the mail server end than on the client end.

When the POP client does find and fetch a new message, it will be responsible for handling and removing the e-mail headers. We do not plan to support any e-mail features by reporting the sender, time received, etc. that will be found in the headers. The POP client will return the headers to the driver and will be called again after the message is displayed.

3.4 TCP/IP stack

The TCP/IP stack is responsible for taking the POP client's requests and wrapping them in the proper TCP and IP headers and sending this packet to the Ethernet controller for transmission. It is also responsible for handling, checking and stripping the TCP and IP headers from incoming Ethernet frames.

We hope to find a suitable stack that we can drop into our system with minimal adjustments. Currently, we do not know what kind of compilation challenges the board will provide. If no suitable stack can be found, we will implement a barebones stack, probably with the help of "TCP/IP Lean" by Jeremy Bentham. Since we will not need to support many features of TCP or IP, we do not anticipate this to be too difficult. However, if it is necessary, it will probably be the hardest software to write for the system.

The IP packet that this stack creates will be taken from the SRAM in hardware and sent to the Ethernet controller for frame headers and transmission. Likewise, frames received by the controller will be written into SRAM and passed to the stack for handling.

3.5 Terminal

The terminal will be a slightly special purpose terminal similar to the one we implemented in lab 2. In fact, the hardware used in lab 2 will probably stay the same, and but the software will need some adjustments. Instead of accepting input from the keyboard, the terminal will only be taking input from the driver. The carriage return and line feeds that were available in lab 2 will be combined into a return. We plan to display one message on a line, wrapping if necessary. We will probably scroll up by one line when all rows are used, but this behavior has not been fully determined yet.

4. DESIGN ISSUES

The main Design issue in the Hardware part will obviously be the handling of the OPB which is shared between the Microblaze, the Ethernet Peripheral, the SRAM Peripheral and the Video.

We also comprehend that it is necessary to implement some sort of Protocol, may be an interrupt handler which prevents a new data from being lost or over written when the system is processing the Current Data from the Ethernet. This again boils down to how we store the intermediate data while the previous data is being handled.

The Design issues we find by designing the software part are:

Offsite Mail Server
Although the specifics of this are not finalized, we do know what we need this server to do. The mail server will have to run a simple POP server that allows plaintext passwords, if any passwords need to be used. Since the Columbia mail servers use encryption for authentication, we will not be able to use them. In fact, if the POP server allows it, we will not need any authentication at all.
We definitely plan to use third party software for this part of the project.

The machine we install this software on will probably be a barebones Linux machine. One of our group members has a working Linux web server that is available, or Professor Edwards has volunteered a machine to use.

Network Issues
Since our system will need an IP address to communicate with the mail server, we recognize two possibilities for obtaining one. We can either implement a very lightweight DHCP client by ourselves (unless we can find a third party one), or we can arrange for one with the lab's network administrators and hard-code it into the system.

Cellular Network
Fortunately, this end of the system is already taken care of. Major cellular networks allow SMS messages to be sent to arbitrary e-mail addresses. We have already demonstrated this end of the project by sending SMS messages to our Columbia e-mail addresses.