Little Brother Surveillance: Viewing Surveillance Camera Images on a Cell Phone

Nathaniel Brown, Chris Mordue
Department of Computer Science and Engineering
University of Washington, Seattle, WA
{cmordue, ngb}@cs.washington.edu

ABSTRACT

Surveillance cameras have become a pervasive element of today's retail stores. Hidden from public view they are used to monitor costumers and employees to prevent theft. Privacy advocates worry that these hidden cameras could be misused and want to know how they are being watched. These camera images could be put to many other uses such as finding a lost child or friend. Even more prevalent than surveillance cameras, are the public's use of cell phones. We propose a system that combines these two technologies and allows anyone with a cell phone to view store surveillance camera images. In this paper we describe the design and implementation of a prototype Little Brother Surveillance system.

1. INTRODUCTION

A mother is shopping at a department store with her young child. While the mother is trying on clothes the child wanders off out of sight. Today the mother must wander around the store looking for her child, or ask store personnel to help. As with all department stores this one is equipped with an extensive array of surveillance cameras throughout the store. If the parent just had access to the camera images she would be able to quickly see where her kid is and go retrieve them.

Most new cell phones today come equipped to handle multiple wireless communication protocols such as Bluetooth, infrared and 802.11 wi-fi. The goal of Little Brother Surveillance is to use the phone's wireless technologies to send surveillance camera images to the phone. Now when the child wanders off mom has an easy way to find him. She can pull out her cell phone and open Little Brother Surveillance. An image of the surveillance camera nearest her will appear on her screen and she will be able to scroll through the store's network of cameras until she finds her child. Using the description of the camera's location, as well as the image itself, she can go retrieve her kid.

The outline of the rest of the paper is as follows. Section 2 compares our work to what others have done before. Section 3 describes our approach and implementation strategy. Section 4 evaluates what we did. Section 5 is a discussion of possible future work. Section 6 is a conclusion and is followed by the reference section.

2. RELATED WORK

There are several products on the market that are similar to Little Brother Surveillance. Nokia makes a camera that can send images over the cellular network. It is basically a camera with a phone number that you can dial into to see images. This is very good for offsite surveillance situations but a little bit overkill and expensive for our scenario.

There also are a number of wireless cameras on the market that send their images back to a computer which can then post them to a website. A web equipped phone could then load this website to view camera images. This again is very good for offsite surveillance such as keeping an eye on your home while on vacation. For our project's idea to work with this system, each department store would need to have a web page and publicize it to its customers.

FPGAs are widely used for development work and therefore there are a number of implementations to communicate with memory banks via several communication channels (i.e. RS-232, RS-485, telephone lines, etc). Many of the newer proto-boards have USB ports built into them. Microcontroller makes an FPGA proto-board with USB and serial port connections on it for easy USB development without requiring extensive knowledge of the USB protocol[1]. The USB protocol is based upon a host-device system. All of the USB controllers we found made the FPGA a USB device assuming the FPGA would be connected to a host. FPGAs are often used for real time imaging. Xilinx boasts itself as a chip maker that produces chips with image processing in mind[2].

While there are some similar products on the market, none of them integrate the components all together in the way that our project does.

3. Approach and Implementation

Parts List

- PC
- Logitech Quickcam Express USB web camera
- Xess XSV proto-board with Xilinx XCV300 chip
- ECS-2100 48.000 MHz oscillator
- 3COM USB Bluetooth dongle
- Nokia 6600 cell phone

Original Implementation Plan

The original goal of the project was to connect a web camera directly to a FPGA and to have the FPGA act as an USB host. The FGPA would communicate directly with the web camera through the web camera's USB communication protocol. Once a web camera image frame had been transferred onto the FPGA board, a resizing algorithm would shrink the image to a predefined size (suitable for a cell phone). The image would then be broadcast via Bluetooth to a Nokia 6600 cell phone. Initially, we intended to use the Bluetooth capabilities of an Intel personal server to communicate between the FPGA and the phone. Once the image was broadcast on Bluetooth, an application on the phone would buffer and display the image to the user.

Seeking to tackle the largest problems first, we began work on understanding the USB web camera communication protocol and also the FPGA USB controller. The communication protocols for the all USB web cameras were unpublished. This had not stopped a number of Linux users from reverse engineering part of the protocol for Logitech's Quickcam Express to produce a scaled-down Linux driver for it. We used their work and USB tracing software from HHDSoftware to try and understand the protocol ourselves enough to implement something on the FPGA[3]. This proved to be a much more difficult task than we had anticipated and forced us to give up on connecting the USB web camera to the FPGA, after spending way too much time on it.

The time spent on the USB controller, on the other hand, proved more successful. Realizing that we could not write our own USB controller in the time allotted, we searched the web for free Verilog or VHDL implementations. The only free softcore USB controller we could find for the FPGA was produced by Trenz Electronic[4]. The package from Trenz included sample software for the PC to write data over USB and have it displayed on the FPGA board's seven-segment display. This appeared to be a reasonable starting point for our USB communication needs. This module required the FPGA to have an external 48 MHz clock. We connected an ECS-2100 48.000MHz clock chip to pin J27 on the proto-board.

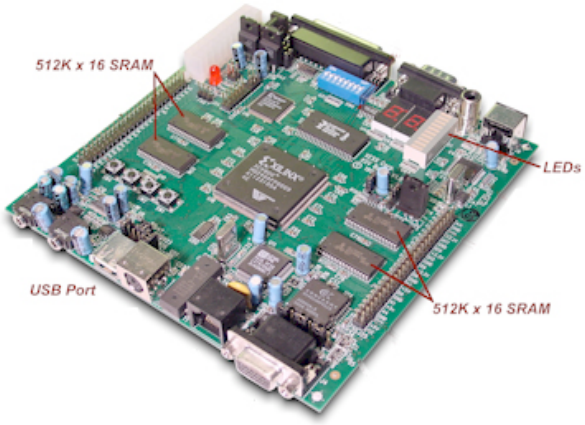

Figure 1. Xilinx XCV300 FPGA on a Xess XSV proto-board.

Unable to communicate directly with the web camera, the overall design needed to be changed. The web camera would no longer be plugged into the FPGA board, but would instead be routed through a PC. We then could use the PC to grab the image and send it to the FPGA in a format the FPGA could understand. Now that we had decided to include the PC in the implementation, we decided to have the PC perform the Bluetooth communication as well. The personal server was then removed from the implementation plan.

Final Implementation Plan

The final plan had been settled upon. The USB web camera, a Logitech Quickcam Express, is connected to a PC. The PC is running a Java application using the Java Media Framework to capture the web camera's image. We modified an example piece of software to meet our needs[5]. This code converts the image pixels to gray-scale to simplify the implementation on the FPGA. Using the host software example from Trenz Electronics that came with the USB controller, we developed an application to write images via USB to the FPGA. The FPGA receives the pixel values for the image and writes them to one of the proto-boards SRAM banks. Once a frame is entirely loaded into the SRAM, the FPGA begins resizing the image and writing it back to the SRAM. Upon completion, the

FPGA resends the resized image back to the computer over the USB port. The final step is to send the image via a Bluetooth dongle to the cell phone. The cell phone will look for the computer broadcasting the camera images. An application on the cell phone will capture the image and display it for the user.

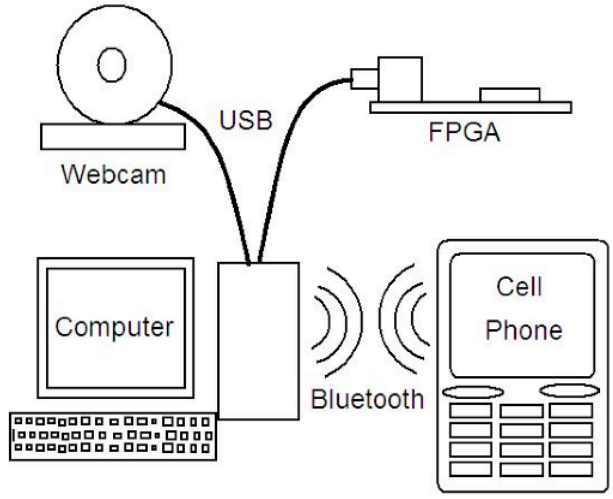

Figure 2. Overall hardware architecture.

As mentioned earlier, the USB controller came set up as an example to write values to the FPGA proto-board's two 7-segement displays. To do this, there was simply a 16-bit data in and 16-bit data out bus coming from the USB controller. According to the documentation for the USB Function Controller, we should be able to connect it to multiple endpoints. However, after further investigation, we realized that the free version of the USB controller was an unsythesizeable net list, pre-wired to a 16-bit input and 16-bit output endpoint. With no other available USB controller options, we were forced to develop a communication protocol using only these 16-bit data lines.

The host software on the PC is written to send and receive 16-bit data packets via USB. These packets are sent at a slower rate than the 48 Mhz clock speed that our Verilog module is operating at. Therefore we can sample the data lines every clock cycle to determine when the lines have stabilized, and then register the values. This works effectively but requires that consecutive data packets be different, which is enforced by the host software.

Each pixel value sent over USB is immediately stored in one of the FPGA's SRAM. The FPGA's proto-board contains two SRAM banks with each bank having 2^{18}, 16-bit locations. Reading and writing of the SRAM is done through a VHDL module from the University of Queensland[6]. A read or a write takes two 48 MHz clock cycles.

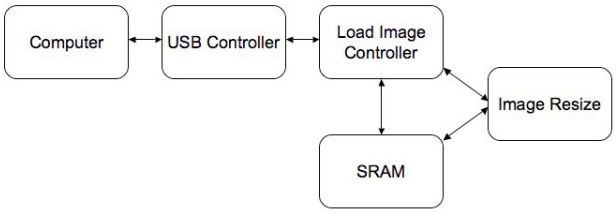

Figure 3. FPGA Architecture.

To resize the image, a simple algorithm was devised. Conceptually, the algorithm inflates the pixels of the resized algorithm and overlays the grids together. Then the algorithm takes the pixel value of the old image's pixel whose borders surround the center of the new pixel. Figure 4 shows an example of the resize algorithm.

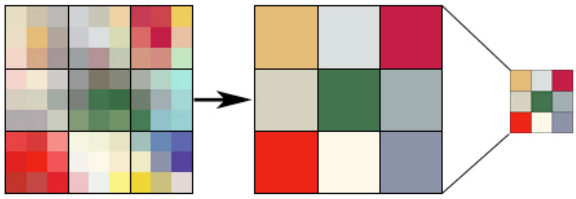

Figure 4. An example of the resize algorithm resizing an image by a factor of three.

Using C++, the algorithm was implemented and tested using a graphics package (gp_142) from the Spring 2001 CSE 143 class at the University of Washington. Then, the algorithm was optimized to enable an easy implementation in Verilog. All divisions and multiplications were eliminated. The C++ code was then converted to a Verilog finite state machine and was integrated into the FPGA's project code. To integrate the Verilog code, we muxed the data line and read/write lines so that the code to write the initial image to the SRAM and the resizing code would work on the same SRAM.

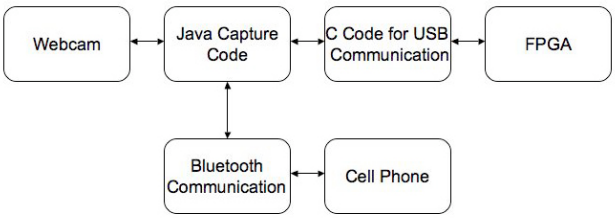

Figure 5. Software Architecture.

Bluetooth communication from the PC is not very well supported. Several methods of Bluetooth communication for PCs were looked at. The current version of Windows does not contain a Bluetooth API. There exists a Java API for Bluetooth but the actually implementation for it is not available for free. One approach we explored was to open a socket connection, in Java, to the Bluetooth dongle. We also tried to use the Bluetooth dongle as a simulated serial connection to communicate with the phone. We were able to send characters from the phone and receive them in Hyperterminal on the computer. We were unable to send characters from Hyperterminal to the phone. The phone comes packaged with software that enables you to send it files from the computer. These files are received by the phone in its inbox. Thus we can send and display images to the phone, but not from within our own software.

4. Evaluation

USB and Bluetooth communication proved to be much more complicated than we had originally anticipated. The original design goal of producing a tightly packaged add-on for a USB web camera that would enable it to communicate with a cell phone via Bluetooth was not met.

Trying to reverse engineer the bit-level communication with the web camera was a heavy drain on time. We were forced to abandon this route and probably should have done so much sooner. The result of having to add the PC made us give up the tightly packaged model we had originally envisioned and set us several weeks behind schedule.

Development work on the FPGA was slower and more difficult than normal. The USB controller modules and SRAM module were written in VHDL. Our implementation was written in Verilog. Such mixed language designs can be synthesized onto the FPGA but can not be simulated in software together. This, along with the lack of output pins on the proto-board, made debugging FPGA code extremely difficult.

We have a working protocol to send and receive images to and from the FPGA via USB. Due to the limited capabilities of the free USB controller, the data rate is much too slow to be feasible. Sending a typical webcam image to the FPGA for resizing takes a mater of minutes when it should be done in less than a second.

5. Future Work

To make this product practical, the computer must be removed from the implementation. This would involve understanding the USB webcam protocol for the Logitech Quickcam Express. We would also need a fully featured USB controller to handle these communications efficiently. A serial Bluetooth dongle that could plug into the FPGA board is also required (or a USB one and another USB port on the proto-board).

There are several future extensions that would further the users enjoyment. Using color rather than black and white images would be the fairly easy to incorporate. Extending the system to handle multiple cameras and multiple cell phones is another possibility. And finally, a usability study of a working system would provide more feedback as to possible improvements.

6. Conclusion

Many valuable design lessons were learned throughout the project. Knowing when to give up on a method of implementation in favor of another due to time constraints is essential to the completion of a good finished product. Adjusting to different implementations, in our case, required familiarity with multiple programming languages. This was not an issue but often the most difficult parts dealt with integrating them together effectively. Integrating multiple independent systems together proved to be more difficult than we had anticipated. Developing or finding appropriate test beds to verify correct communication between different pieces is essential.

Despite the difficulty with the project, we feel we have learned a plethora of things about communication protocols. The big lesson to take home is that while Bluetooth and USB are industry standards, developing hardware and software that utilizes them is still very complicated. The up and coming Microsoft Bluetooth API should simplify future Bluetooth development. Utilizing this, along with a full-featured USB controller for the FPGA would enable this project to be taken to completion.

7. References

[1] Saelig Company website for Microcontoller Board Interfaces. http://www.industrialnewsroom.com/fullstory/19107.

[2] Image Scaling and Enhancement IP. http://www.xilinx.com/esp/mil_aero/collateral/presentations/image_scaling_enhance.pdf.

[3] HHD Software USB Monitor 2.36. http://www.hhdsoftware.com/.

[4] Trenz Electronic. http://www.trenz-electronic.de/.

[5] Using Logitech Webcam in Java. http://ltu164.ltu.edu/itseng/version.htm/.

[6] SRAM Interface in VHDL. http://www.itee.uq.edu.au/~peters/xsvboard/.