Introduction

The purpose of the project was to implement a character recognition system for a robotic rover using Xilinx's Virtex-II Multimedia Board and Xilinx's development software platforms, ISE and EDK. An image of a character is downloaded onto the multimedia board, where image processing algorithms decipher the image and output the correct character in simple text format. Our plan was to interface this character recognition system with our ECE final-year design project (ECE496), for which we developed a robotic rover that is capable of gathering and processing visual and audio clues in predefined formats. The character recognition system would augment the robot's existing image processing and autonomous capabilities by enabling it to recognize and interpret characters within the images of clues it captures with its webcam.

The robot was supposed to take images of visual clues using its webcam and download them onto the multimedia board's memory for processing. The character recognition system will store and compare images of clues taken with reference characters stored on the multimedia board's ZBT memory. The stored reference character that best matches with the character on the image of a clue (above a certain probability) will be output to the UART, in plain text. A simplified functional system block diagram of our project is shown below. (For a more structural system diagram, please refer to our group report).

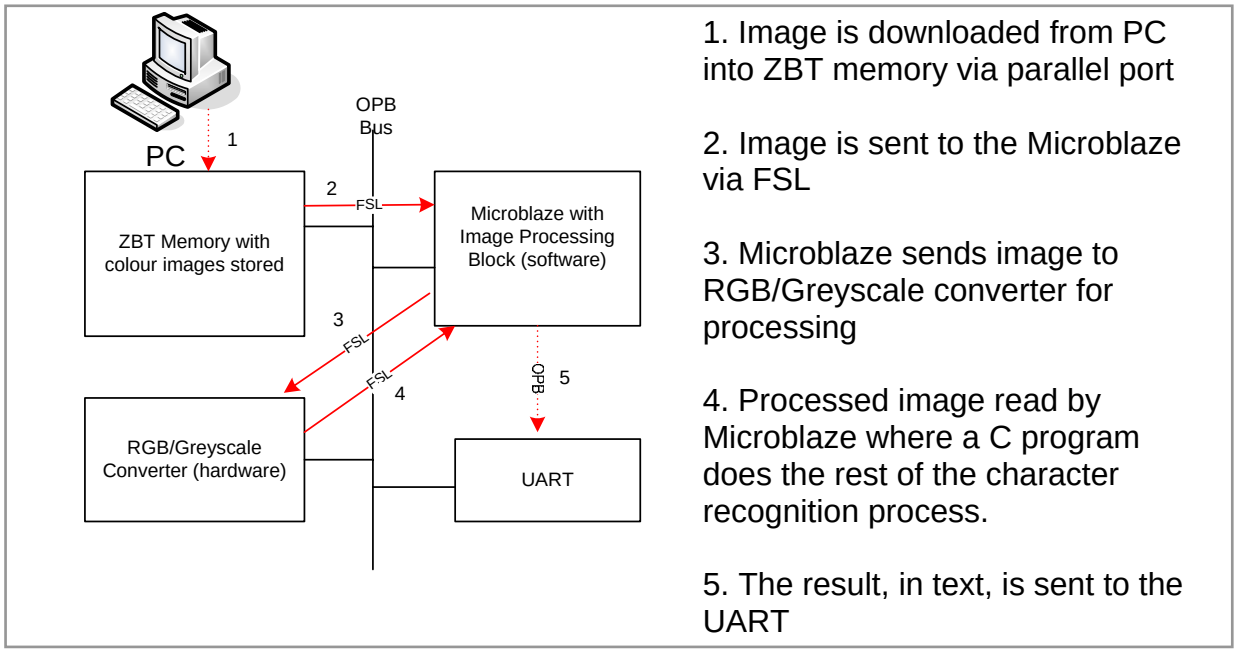

Fig. 1: System block diagram of character recognition system.

As shown in the system block diagram above, our character recognition system begins with a computer (presumably our robot) inputting an image of a visual clue that it has taken, over a parallel cable connection, onto the Xilinx multimedia board's ***ZBT memory***. ZBT memory uses an external memory core (EMC) (not shown in Fig.1), which acts as a controller to enable the image data to be downloaded onto ZBT memory.

The image data is then sent over a ***fast simplex link (FSL) bus*** to the ***Microblaze processor***, which then relays it to the ***RGB/greyscale converter*** to change the image from a colour

image to a greyscale image. Since the visual clues are in the form of black text against a white background, converting an image of a clue to greyscale serves reduce the amount of detail and, therefore, noise in the image to be processed. The RGB/greyscale converter was implemented in hardware using the FPGA and VHDL.

The greyscale image is sent back to the Microblaze over another FSL to perform the rest of the character recognition process. The ***image processing block*** used by the Microblaze is a software component written in C that compares the reference characters stored on ZBT memory with the image of the clue using various image processing algorithms to determine the reference character(s) that best matches the character(s) within the clue.

The once a reference character has been chosen by the image processing block as the one that best matches image of the visual clue taken, the plain text value associated with that reference character is displayed using the ***UART***.

My Contributions to the Project

Converting MATLAB algorithms to C

When my partner, Yasir Mirza, and I started on this project we both worked on the character recognition software of this project because the software was the heart of our project and it was our top priority to have a C version of it working on the Xilinx board as quickly as possible.

The original character recognition algorithms were written using MATLAB. I tried converting small portions of it into C using a compiler that was available in MATLAB and then tried to compile the C code snippets on XPS. We had originally hoped to use MATLAB's built-in compiler (mcc) to help us with the conversion process. However, this proved to be fruitless because the compiler used by XPS (mb-gcc) was a much lighter compiler without all the libraries needed to run the converted MATLAB code. This oversight on our part made conversion of the MATLAB code into C code to be the most time consuming part of this project.

We had hoped to finish the conversion of out MATLAB code early so that we could spend the remaining time turning our software into hardware. But, after a couple of weeks of slow progress with the MATLAB-to-C conversion process, we decided to start implementing the hardware portion of this project in parallel. Yasir continued to work on converting the MATLAB code to C, while I started implementing the EMC core for ZBT memory and the RGB/greyscale converter.

To ensure that my part would work with my partner's part, we made sure that the interfaces of our individual parts were in common; that is my part would read and process an image from ZBT memory using the same C functions that my partner's part would use to read and process the information from my part. We also tested portions of our parts together to ensure that one part can read what was written onto a bus by the other. However, we didn't have time to do a final test of all the components together.

Implementing ZBT Memory

Implementation of ZBT memory was relatively painless, since the tutorial from the course website was fairly explanatory. I was even able to increase the microprocessor clock speed to 50MHz so that it would operate at the same speed as the FSL for the RGB/greyscale converter. However, I did encounter some connection problems using while XMD. The error message that

would come up when connecting to the MDM was that the Microblaze was in reset and was unable to connect. I checked to make sure that switch SW0 was not in reset. I checked to make sure that my ports for my cores were connected correctly and that the parameters were the right values. However, the problem went away when I tried implementing ZBT memory a second time.

More problems came up when I tried implementing an FSL link between the Microblaze and ZBT memory (via OPB-MDM). I wasn't noticing any difference in performance in downloading images onto ZBT memory with and without the FSL. The problem turned out to due to the C_DEBUG_ENABLED parameter in the Microblaze instance being set to 0. This prevented the use of the on-chip hardware debugger, which was needed in order to use the FSL. After changing the parameter value for C_DEBUG_ENABLED to 1 and setting the connection type in the XMD Debug Options dialog box to hardware, I was able to download images onto ZBT memory about six times faster than before.

I also tried to connecting to multiple ZBT memory banks, which gave me errors which told me that certain ports had multiple drivers. This problem was solved when I realized that 1 ZBT memory bank was sufficient for out project.

I tested ZBT memory using TestApp.c, which is automatically generated by Base System Builder, test to see if ZBT memory is working properly. Then, I used Test_Mem.c to read and write values off of ZBT memory. Both programs showed that ZBT memory was working properly

Implementing RGB/Greyscale Converter

We had originally planned to implement the RGB/greyscale converter using Verilog because it was more widely used in industry. However, after spending some time researching Verilog designs, I realized that the clk_align and FSL cores were implemented using VHDL and, since each project can only use one HDL, I had to do some research into VHDL designs as well.

The images stored on ZBT memory were in 24-bit BMP format. I used Xilinx's application notes XAPP637 and XAPP529 (both are available on Xilinx's website under Application Notes) as my references on how to implement a RGB/YCbCr converter using FSLs to connect the core to the rest of the project.

One of the biggest hurdles that I encountered was in trying to decipher the complex VHDL codes for the examples from the application notes. There were numerous signals and functions from the examples for which I did not understand their purpose. The codes themselves didn't have any useful comments to guide you through the logic. I was hoping to find simpler, more generic cores that I could adapt for my purposes. But after spending a good deal of time studying the cores, was able to extract the basic functions that I needed to implement my relatively simple RGB/greyscale converter.

Having created the converter, my next task was to find a way to connect the core to the rest of our project. I initially looked into putting the core onto the OPB bus. However, that required creating an OPB bus interface for the converter as well as memory-mapping it so that it can be used. To simplify matters, I decided to use FSLs to act as the communications link between the Microblaze and the converter. With FSLs, the interface is already provided for me; all I needed to do was to connect the FSLs to the right slave and master ports for reading and writing.

The next hurdle was to simulate the core, to make sure the reads and writes were occurring at the right times. Getting to know all the FSL signals, to know which ones are needed and in what order, took quite a bit of time. I also tried to create a VHDL test bench for the converter, but I kept on getting compilation errors. Luckily, ISE was available to help me create a test bench for my converter. Using Test Bench Waveform, I was able to produce the simulated output shown below.

Initially, I had wanted to modify the RGB/YCbCr converter from XAPP637 so that only information on Y will be outputted because Y contained information on an image's brightness. However, the complexity of the existing code made it difficult to modify. To solve this problem, I took a simpler approach. My research showed me that the colour green in an image contained the majority of the brightness information for that image. I also learned that for a 24-bit bitmap, the colour information for each pixel is stored in the order of red, green, and blue, with 8 bits out of the 24 dedicated to each colour. Therefore, for every 24-bit pixel value that is read from memory, my converter filters out the red and the blue components to leave only the green, as shown in the simulation in Fig.2. The resulting image would be a greyscale, as shown in Fig.3.

Implementing the rest of our character recognition system in hardware proved to be difficult due to all the computations that our algorithm does on using floating-point numbers, which is difficult to practically convert to a fix-point implementation. This was one of the major hurdles to implementing our system in hardware that we weren't able to overcome in time.

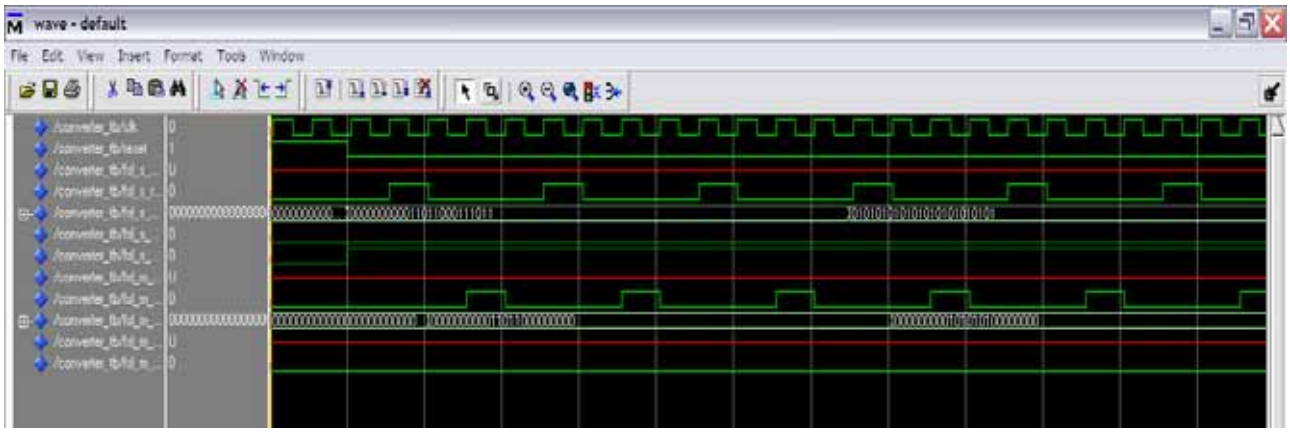

Fig. 2: Simulation of RGB/Greyscale converter.

Fig.3: Left: Before filtering red and blue components. Right: After filtering. (Images produced using MATLAB).

Looking back…

In retrospect, we underestimated the difficulty involved in converting a MATLAB code to C. This proved to be the one thing that held everything back. If we had the chance to do things differently, we would have started by compiling a simple program using MATLAB to C and downloading it onto XPS to see what problems might arise. We would also have started on the hardware portion of the project a little earlier rather than spending so much time on the software part of the project. At least the chances of having something working in time would be better. It might also have been better if we started by implementing our system in hardware and then use snippets of C code to do the things that were not implementable.

Course Feedback

I like the labs in this course but I would have liked to have gotten a little more help in some of them. While it may be good for us to figure out how to solve some problems on our own, I felt, especially during the first few labs, while people are still learning to use the tools, people should be entitled to little more help. Answers for the questions in the labs, for example, should be posted some where or least answered during a lab session as well.