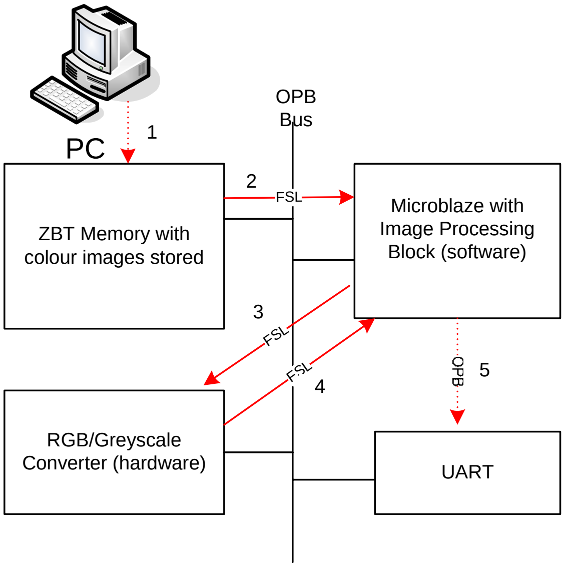
Introduction

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. This project was carried out to complement our final year design project (ECE496). That objective of that project was to control a robot from a remote location over the wireless internet. The goal was to make the rover as autonomous as possible. The robot in ECE496 project took part in a competition in which it had to gather visual and audio clues. The visual clues consisted of plain text printed on white background. That's where ECE532 project ties in. If we could design an algorithm for character recognition, it would complement our robots autonomous behavior and thus achieve the object of the project.

Another component of the design was RGB to greyscale converter. This block is important because most of the image processing algorithm that exist work on greyscale images only (like edge detection, histogram equalization, etc). This block was to be completely implemented in hardware. The complete system and the interaction between all the blocks can be seen in the diagram below.

The robot had a mounted USB webcam using which it can capture images of clues. Once those images are taken, they can be downloaded to the ZBT memory of multimedia board. Then the RGB to greyscale converter, implemented in hardware, makes the appropriate modification to the image and stores it back to a different location in memory. Once that is done, the character recognition software is used to compare it to a set of training images. Finally the output of character recognition is printed on screen using RS232 interface.


PC
1
OPB
Bus
2
FSL
ZBT Memory with colour images stored
Microblaze with Image Processing Block (software)
3
FSL
FSL
4
OPB
5
RGB/Greyscale Converter (hardware)
UART

1. Image is downloaded from PC into ZBT memory via parallel port

2. Image is sent to the Microblaze via FSL

3. Microblaze sends image to RGB/Greyscale converter for processing

4. Processed image read by Microblaze where a C program does the rest of the character recognition process.

5. The result, in text, is sent to the UART

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

In the above diagram the ZBT memory uses an external memory core (EMC), which is controller that allows the images to be downloaded to memory from pc. The core is not shown in the picture above.

The communication between the Microblaze and ZBT memory as well as the converter is done over a FSL bus. This makes downloads and data transfer faster and also reduced the complexity of design in our case.

Contributions to the Project

Code for Character recognition

In this project I have mostly worked on the software parts of the code and my partner, Eric, worked on the hardware side of things. However, these are not mutually exclusive and both of us helped each other also.

First part of the design procedure was to do some research on character recognition algorithms. This took some time as there are many different algorithms for doing this sort of work. Choosing the best algorithm that suits our requirement was not easy. Several algorithms were tried and codes for those algorithms were generated in Matlab. After quite a bit of testing we settled on the histogram based character recognition algorithm. The details of this algorithm and the paper describing it can be found in the group report.

One might ask, why would we use Matlab to implement the algorithm when it is not compatible with Xilinx tools anyway. The answer is that it is much easier to implement these algorithms in Matlab. It has an Image Processing Toolbox that provides many builtin functions for image manipulation and processing. When we were testing many different algorithms in the beginning it was much more appropriate to do it in Matlab so that we could quickly compare the results of all the algorithms.

So, when we settled on histogram based algorithm for character recognition we tried to convert it to C. My original plan was to use Matlab Compiler (mcc) to generate the C code. I had never used mcc before. Little did I know that what a big mess it will make. The way it works is that goes through all the functions and tries to convert them to C. If you call another function whiting that then it tries to convert it to C as well. For example, imread() function simply read a specified image file. However, it calls many other functions inside its body for error checking and other purposes. Those functions further call many other Matlab built-in functions. As you can imagine, it quickly builds up to a large number of files. When I compiled my code using mcc it generated over 250 C files and their headers. I imported those into XPS and sure enough it didn’t work. Chris Comis explained to us that mb-gcc is a much lighter compiler and doesn’t have all the necessary libraries. Therefore, we had to switch gear and try to convert the Matlab code to C manually. This resulted in significant wastage of time.

At this point I and my partner decided to split the work so that I mainly worked on the software and he worked on the hardware. The manual conversion of Matlab code presented a lot of problems. The reason is that Matlab is built for matrix manipulation and you can do a complex task with simple calls to built-in functions. For example, creating a given number of bins and distributing the values from a matrix into those bins takes just one line. However in C, I had to write a separate function for this specific algorithm.

I decided to first create the C code on ugsparc machines and use gcc to compile it since it is much easier to debug without constantly building the system, downloading it on board, and running gdb.

The first big problem with C code was how to read pixel values from image files. Again Matlab had a one line solution for this (i.e. imread() function) but in C it was much harder. Originally we were dealing with Jpeg images, but since the Jpeg format encodes the values with Discrete Cosine Transform (DCT), it was hard to read the pixel values. Therefore, I decided to use bitmap images instead.

The next big problem had to do with my rusty C, I was trying to pass image data contained in a two dimensional array between different functions. However, I didn't know that C actually passes a pointer to the array instead of the actual array itself. Thus, I was always reading in zeros from the image array. Once I figured out the proper way of passing arrays, the next problem had to do with printf() statements. I though that I would be able to print double values using %d in printf but that wasn't the case. So for sometime, printf() kept giving me weird numbers on the screen which looked like memory addresses rather than actual vales of variables. I figured out two problems with my code, first I was not properly casting some variables and secondly %f gave the correct output for doubles.

Once my code worked on ugsparc, I tried to put it on the board using XPS. However, it immediately gave me error due to size of code. Virtex-II boards has only 64 KB of memory for instructions. Since I was using sqrt and log functions in my algorithm, so I had included math.h library. It turned out to be a bad idea since all the functions in math.h were included in the memory. Therefore, I had to come up with some other way of including these functions. I created a Newton's method based algorithm for square root approximation.

Finally after solving all these problems, I was able to successfully run it on the board. I did some testing to make sure that it was recognizing the patterns correctly and the output was similar to what Matlab code was producing.

In order to make sure that my code worked correctly with my partner's. I made sure that his code for reading values out of memory worked with my source code since ZBT memory is the way through which images are passed between hardware and software blocks.

Through this project I was able to learn a lot of new skills and better understood how busses work, how microprocessor's work, etc. Also, this taught me the lesson that you have to spend a lot of time familiarizing yourself with a new tools. Although, it seems like you can build your project and get every thing working with a few button clicks but it is never that simple. This also taught me not to rely too much on "shortcut tools" i.e. mcc. Had we not lost time in the beginning due to all mcc problems, we would have a much easier time with finishing our project.

Other Work Related to Project

As I stated earlier, the hardware and software components of this project were not mutually exclusive and we helped each other to do the work.

I helped Eric with debugging the ZBT memory stuff after he couldn't get it to work by going through the tutorial. Similarly, I did some research on the web to find out a way to implement RGB to greyscale conversion. I was able to find YCrCb to RGB conversion block on Xilinx website. Eric was able to modify that block appropriately to get some of his stuff working.

Other than that, obviously a lot of time was spent on ECE496 project which is a full-year design project. Many other image processing algorithms were designed to provide further autonomy to the robot.

Feedback to Xilinx

There is one bug which I found with Xilinx tools. Lets say you are working with your C code and you open multiple files in the window of XPS then every time you close a file it switches to "System" tab for no apparent reason and you have to go back to "Applications" tab to access you code again. This doesn't happen if you have a single line available.

Also they should provide a command line interface in xps so that you can have much more control on what you do. For example if you want to compile you code with several different options, currently it is not very easy. It could be made much easier with a command line.

Course Feedback

I think the course was very nice and rewarding and made us learn a lot. The way all the modules were marked was also good because it allowed us to work on module according to our schedule and not have a threat of submitting a lab every week hanging over our head. The grading structure is also fine since it allows us to skip a final exam in a tough semester and gives us a much more hands on experience by doing a project. However, I would like to have better information as to what to expect in the quizzes. Currently the course doesn't see to have a strict focus, so it would be nice to develop a syllabus and follow it throughout. Lastly signing out xilinx boards to students for the whole semester would also help a lot.