1. Overview

Goals

The purpose of the project was to implement a character recognition system using Xilinx's Virtex-II Multimedia Board and Xilinx's development software platforms, ISE and EDK. The project is an offshoot of our ECE496 Design Project where we had to develop a robotic rover 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.

Description of IPs

Shown below in Fig. 1 is the system block diagram for our project. 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) , 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***.

This was our original design and is the final design that we have implemented. We had hoped to convert more of the software implementation to hardware but due to the complex manipulation of floating-point values that our software is required to do and time constraints, we had to settle for a system that was implemented mostly in software.

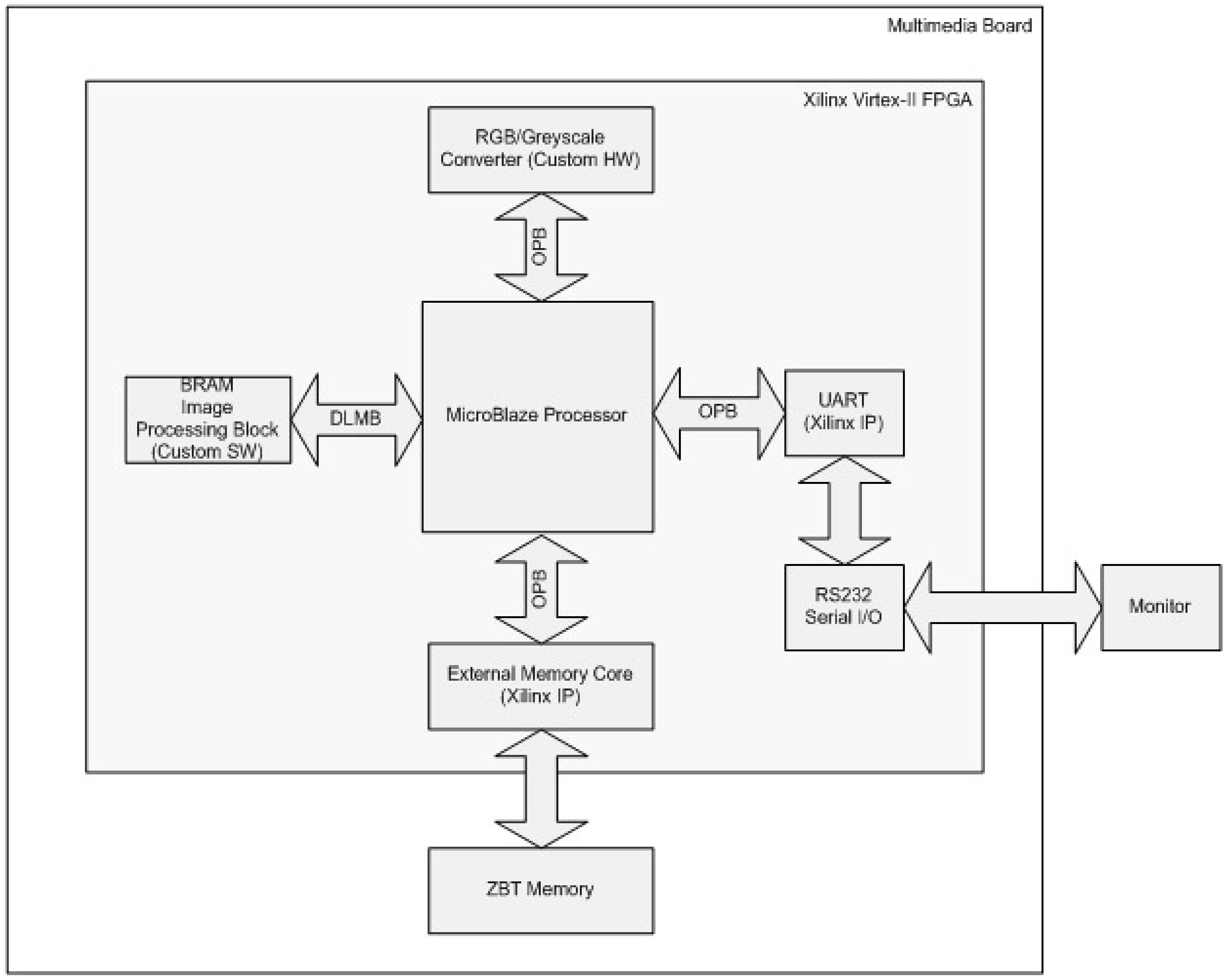


Fig. 1: System block diagram of character recognition system.

2. Outcome

The each individual block works on its own. However, we haven't had the time to put everything together because the conversion of the image processing block from MATLAB to C was completed only recently and was too big to fit onto the BRAMs of the multimedia board.

The character recognition software was first implemented in MATLAB, then the code was transferred to C on ugsparc machines (with gcc), and finally it was transferred to the Xilinx board using XPS. Currently it is running successfully on all three platforms. The program is fairly robust as it was able to make reasonable judgments regarding the degree of similarity between two images even in the presence of noise artificially added to images.

If more time was available to us, we would put all the IPs together and test the overall result. As described in the overview the image is supposed to be converted to greyscale and then passed on to the character recognition block. Right now, both character recognition and greyscale converter work on their own but do not communicate with each other.

3. Description of the Blocks

ZBT Memory: One 2MB memory bank was used for this project. The memory occupies addresses 0x80600000 to 0x807fffff. The design provided by Tutorial M08 worked, so no changes were necessary. For this project, the memory has been set up to operate at 50MHz in order to match the clock speed of the FSLs used in the RGB/greyscale converter. The system was tested using C application to write and read data to and from the ZBT memory.

clk_align_v1_00_a: The clk_align core was used ensure that the clock to the ZBT memory is properly aligned to the internal clk. The design provided by Tutorial M08 worked, so no changes were necessary.

fsl_v20_v1_00_b: Three instances of fast simplex link (FSL) were used in this project. The FSL instance, *download_link*, was used to download images quickly from the hard drive onto ZBT memory. The other 2 instances, *mb_to_hw* and *hw_to_mb* , were used relay image information from ZBT memory to the RGB/greyscale converter hardware (using the Microblaze as an intermediary) and to send the output of the RGB/greyscale converter to the Microblaze for further processing, respectively. The standard interfaces that came with fsl_v20_v1_00_b worked, so no changes were necessary. *download_link* was tested by downloading images to memory and then timing the difference between ZBT with FSL and ZAB without FSL. It turned out that ZBT with FSL was about 6 times faster than without.

RGB/Greyscale Converter (idct.vhd): This converter is used convert a colour image into a greyscale image to reduce the amount of information in the image that the Image Processing software block is required to process. The design was adapted from Xilinx's Application Notes XAPP529 and XAPP637. Both are available on the Xilinx's website, under Application Notes. The FSL wrapper code for XAPP529's was used to integrate the RGB/greyscale converter into the project using FSLs (specifically using FSL instances *mb_to_hw* and *hw_to_mb*). The convert takes in 24-bit pixel data and masks the red and blue components of the pixel, leaving only the green component. The resulting image is a greyscale. A test bench for this block was created using ISE's Test Bench Waveform tool. The output of the simulation is shown below.

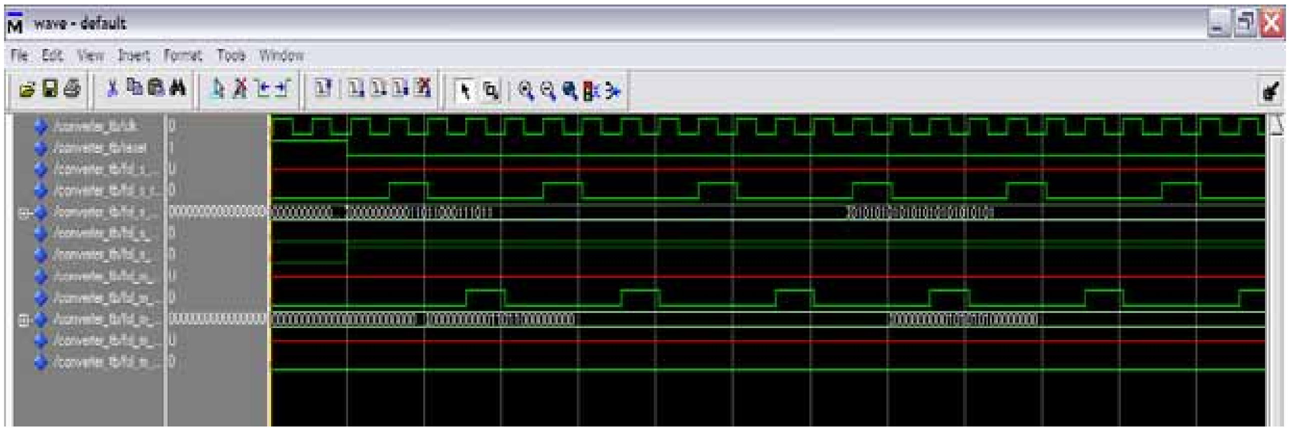


Fig. 2: Simulation of RGB/greyscale converter.

Character Recognition Software:

As discussed earlier, the character recognition software was intended to be used on an autonomous rover which gathers visual clues (consists of black text printed on white background). The ability to recognize characters in a clue is critical for giving maximum autonomy to the rover.

The algorithm for character recognition was not developed by us. In fact, some research was done on the web to find out a good way of solving this problem. Many different algorithm exist but the one we used is described in the paper *"Text Character Recognition: Identification of Hebrew Letters"* by Lavi Zmstein of University of Florida. (http://www.mil.ufl.edu/publications/fcrar03/%7BHebrew%7D%20Letter%20Recognition.pdf)

In this histogram method of character recognition we first find the location of all black pixels in the image. Once that is done, the centroid of each character is calculated. Then the distance vectors are calculated for each pixel from the centroid. These distances are then normalized by the maximum distance from the centroid to get some values between 0 and 1. Then 21 bins are created between 0 and 1 with 0.05 increments. All normalized distance values are then put into these bins according to their magnitude. These bins are normalized again by the total number of bins. Finally this procedure is repeated for the input image which has to be classified. Once these profiles are created for the training images and the input image, each training profile is compared to input image profile to produce a number which represent the likelihood that two images are same. If the number is closer to zero, it represent a higher likelihood.

Now each block of this algorithm will be described for both Matlab and C code respectively.

classify_mod.m: A Matlab function which reads in images of numbers 1-10 from a local directory and calls blah and blah1 functions to create profiles of training images. Once that is done it calls blah and blah1 again to create a profile for input image which has to be classified. Finally it calls kl function to compare the profiles and prints out the result of comparison on screen.

blah.m: This function finds out the locations of all black pixels and then finds the centroid of the character. Then it calculates the distance vector of each pixel to centroid and normalizes all distances with max distance. After normalization it call hist fuction which bins all the values into 21 bins from 0 to 1 with 0.05 increment. Finally it normalizes all bin values again and returns it to classify_mod.

blah1.m: This has a similar functionality as blah except that it after its finds the location of black pixels, it bins the rows and columns separately into 5 bin each without calculating a distance vector. It returns the normalized bin values to classify_mod.

kl_mod.m: This role of this function is simply to compare the bin profiles which were generated earlier using blah and blah1. It returns a number which represent the likelihood that the characters are similar.

nosie_test.m: This function can be used to add Gaussian noise to images and then compare them to see noise immunity of this algorithm

runtest_mod.m: This function can be used to compare all characters with each other. It produces a color map in which the similarity of two characters is represented by shades of gray. Completely white means 100% match and black means that the characters are completely different.

Now the description of C functions:

small_main.c: This file just has the same functionality as classif_mod. It can read in pixels values from the ZBT or alternatively you can just define a pattern yourself in a two dimensional array. So basically it compares two patterns (in the file I call them sample and input image). It calls smaller() function to create the profiles and compare the result. Finally it prints out the result on screen.

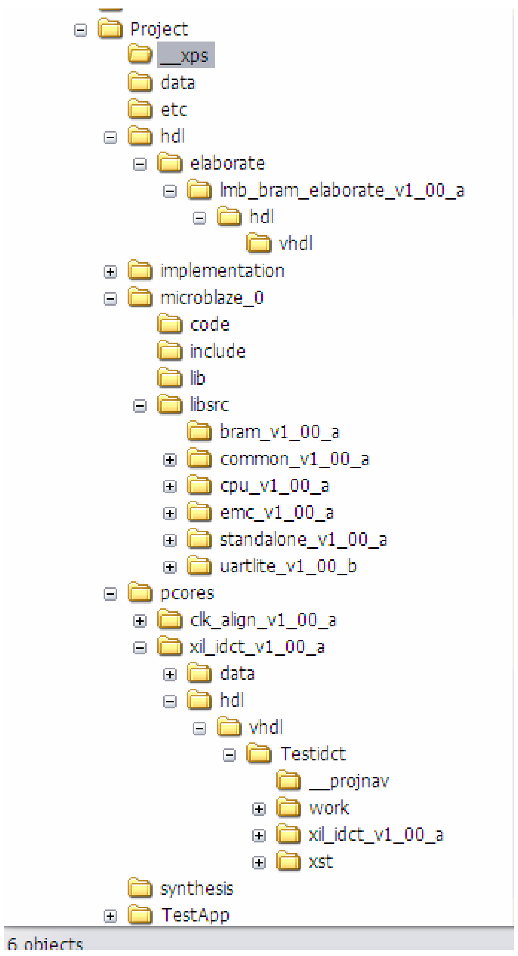
smaller.c: This fuction implements the functionality of blah and blah1. First it finds out the locations of all black pixels and then finds the centroid of the character. Then it calculates the combined magnitude of x & y pixel locations and normalizes them. After normalization it call bins all the values into 21 bins from 0 to 1 with 0.05 increment. Finally it normalizes all bin values again. After that it implements the functionality of blah by creating 5 bins each (calls hist1.c) for row and column locations and normalize these bins with total number of elements in bins. Finally it calls kl1 and kl2 to compare the two profiles created

hist1.c: creates 5 bins and allocates the values into appropriate bins.

sqrt.c: Finds the square root of an integer using newton’s method of approximation using several iteration this was created to avoid using math.h library which is quite large and overflows bram.

kl.c: same function as kl_mod. It basically just compares the bin profiles created by smaller().

4. Description of the Design Tree

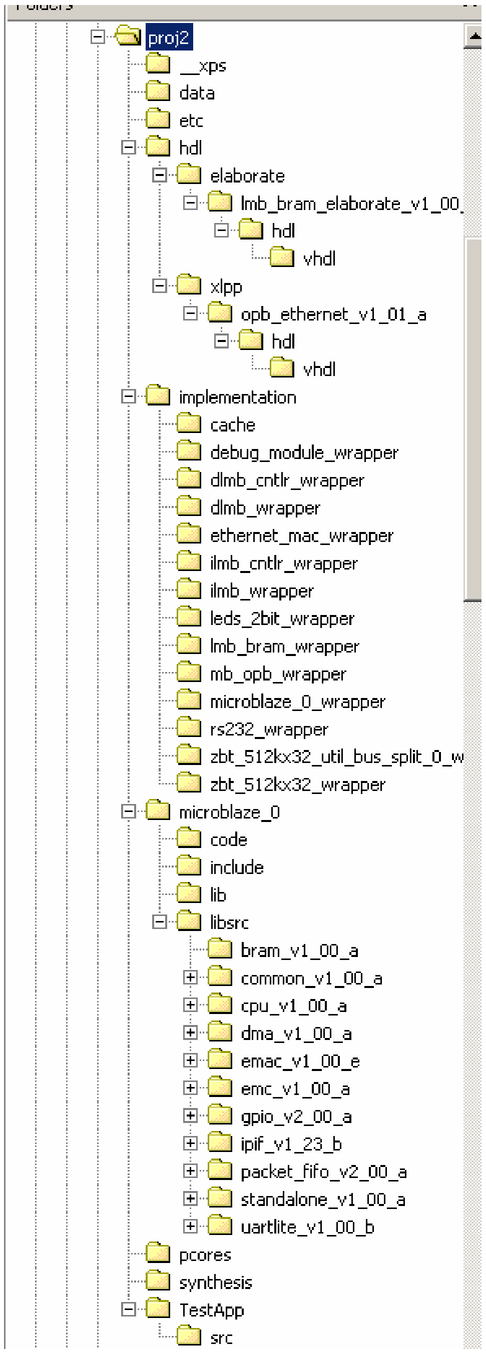


```
⊟ Project
    __xps
    data
    etc
  ⊟ hdl
    ⊟ elaborate
      ⊟ lmb_bram_elaborate_v1_00_a
        ⊟ hdl
            vhdl
  ⊞ implementation
  ⊟ microblaze_0
      code
      include
      lib
    ⊟ libsrc
        bram_v1_00_a
      ⊞ common_v1_00_a
      ⊞ cpu_v1_00_a
      ⊞ emc_v1_00_a
      ⊞ standalone_v1_00_a
      ⊞ uartlite_v1_00_b
  ⊟ pcores
    ⊞ clk_align_v1_00_a
    ⊟ xil_idct_v1_00_a
      ⊞ data
      ⊟ hdl
        ⊟ vhdl
          ⊟ Testidct
              __projnav
            ⊞ work
            ⊞ xil_idct_v1_00_a
            ⊞ xst
    synthesis
  ⊞ TestApp
6 objects
```

Main Directory: Project

File	Directory	Purpose	Origin
Idct.vhd	Xil_idct_v1_00_a	RGB/greyscale converter VHDL file	XAPP529, modified by Eric Wang
Idct_core.vhd Xil_Idct_v1_00_a.vhd	Xil_idct_v1_00_a	Wrappers for RGB/greyscale converter	XAPP529, modified by Eric Wang
Clk_align.vhd	Clk_align_v1_00_a	Ensure clock alignment between ZBT memory and internal clk	Tutorial M08
Testidct.npl	Testidct	ISE Project file for simulating RGB/Greyscale converter	ISE
Converter_tb.tbw	Testidct	Test bench used to simulate RGB/Greyscale converter.	ISE

Converter.c	TestApp	C file used to write and read from converter	XAPP529, modified by Eric Wang
TestApp.c	TestApp	Used to test ZBT memory	Base System Builder
Test_Mem.c	TestApp	Used to read from and write to ZBT memory	Tutorial M08



```
proj2
    __xps
    data
    etc
    hdl
        elaborate
            lmb_bram_elaborate_v1_00_
                hdl
                    vhdl
        xlpp
            opb_ethernet_v1_01_a
                hdl
                    vhdl
    implementation
        cache
        debug_module_wrapper
        dlmb_cntlr_wrapper
        dlmb_wrapper
        ethernet_mac_wrapper
        ilmb_cntlr_wrapper
        ilmb_wrapper
        leds_2bit_wrapper
        lmb_bram_wrapper
        mb_opb_wrapper
        microblaze_0_wrapper
        rs232_wrapper
        zbt_512kx32_util_bus_split_0_w
        zbt_512kx32_wrapper
    microblaze_0
        code
        include
        lib
        libsrc
            bram_v1_00_a
            common_v1_00_a
            cpu_v1_00_a
            dma_v1_00_a
            emac_v1_00_e
            emc_v1_00_a
            gpio_v2_00_a
            ipif_v1_23_b
            packet_fifo_v2_00_a
            standalone_v1_00_a
            uartlite_v1_00_b
    pcores
    synthesis
    TestApp
        src
```

The decription of C files in src directory is

small_main.c: The main function which calls smaller() to generate a profile for image and compare
smaller.c:
hist1.c: creates 5 bins and allocates the values into appropriate bins.
sqrt.c: Finds the square root of an integer using newton’s method of approximation.
kl.c: It basically just compares the bin profiles created by smaller().