18-551, Spring 2004
Group 8, Final Report
Hand Vein Recognition Portal

Bo Chen, bochen@andrew.cmu.edu
Richard Chen, rcchen@andrew.cmu.edu
Joanna Ye, jye@andrew.cmu.edu
Ling Zhai, lingz@andrew.cmu.edu

Table of Contents

1. Introduction

Security requiring verification and identification of individual is a constantly changing and improving field. These biometric systems that are currently available in the market are fingerprints, iris and hand shape devices. During biometric fingerprints, direct contact of the finger with the finger print image extracting sensor causes degradation in performance since good quality fingerprints are difficult to obtain due to oil from the finger, and dirt from the environment in which it is exposed to. During biometric iris scanners, users will feel discomfort when having to place the eye close to the scanner. And finally, biometric hand-shape recognizers are rarely used because problems may arise with people who suffer from arthritis.

We have proposed a new method of biometric identification that resolves the problems that exist in the fingerprints, iris and hand shape devices known as the hand-vein recognition system. Since this method requires a vein pattern image without direct contact with the hand or with the vein-pattern extracting sensor, there exists no contamination. And compared with iris recognition, hand vein recognition is non-intrusive and user friendly. Moreover, hand veins provide large, robust, stable, and hidden biometric features that even identical twins have different and distinct IR absorption patterns. It only requires low resolution IP to train and classify each person's unique vein. Therefore, hand vein recognition can be the next successful and cost-efficient biometric identification method.

2. Solution

Our solution consists of three main parts. The first part is obtaining a database of hand vein pictures that would be sufficient for preprocessing. The second part of our project is to use preprocessing methods to make the images into binary vein images. Lastly we need to train, classify, and test these images.

The solution of obtaining a database of hand vein pictures was solved by creating our own hand vein recognition box that would take pictures of a subject's hand. Creating this box allows us to have the flexibility of taking as many pictures of as many users as we want.

We used 4 different preprocessing tools to convert the raw image into a binary vein image. The first step was cropping the image so that only the subject's veins would show. Afterwards, we used illumination equalization to make the lighting throughout the image equal. Directional filters were used to enhance the vein image, and histogram thresholding converted the image into a binary vein image.

Classification and training in our project was done by a neural network. One of the primary reasons for choosing to use neural networks is that the testing portion would be extremely fast and easy.

3. Database

The first step that we had to take in this project was trying to obtain a database. At first we attempted to contact different research groups and companies in order to use a more professional database for our project. However, since hand vein recognition is still very much an emerging technology, very few groups had a database, and none were willingly to share their pictures with us.

Our next step then had to be to purchase a camera capable of taking images with good enough quality so we could run preprocessing on it. We decided upon the purchase of a black/white CCD camera with 6 infrared LEDs which would be used to accentuate the veins of a person. We also designed a hand vein recognition box that was designed to block out all outside light, and in addition keep a test subject's hand in place so the camera was able to take images at the exact right part of a subject's hand. Since our infrared camera only came with a video out input, we purchased a video out to USB hub so we could hook up the camera to a computer. We used Coffeecup Webcam software to take images at 100*100 resolution.

One of the main problems we encountered in taking images was trying to figure out a method of trying to keep a subject's hand in place without movement. We originally decided upon using a wooden stick with grooves so the test subject could grasp it at the same place. After taking a database with a set of 20 people, we decided to throw it out because the back of the subject's hand would be slanted, and not straight, so the infrared light from the camera to the test subject's hand would be very uneven, and the vein clarity after preprocessing wouldn't be very clear.

We decided to scratch the idea of using a stick, because we realized that we needed to lay a subject's hand flat against the ground. We taped down a cap in the position where the hand should go, and told test subjects to put their hand over the cap. Although variance in this method of taking images would increase, preprocessing clarity improved a lot.

When taking images of a subject's hand, we told the test subject to move their fingers around in order to vary up the images. Sometimes, a test subject's hand would also shift. Slightly to the right or left. It is important to note that different people's veins show up differently. Some people's veins definitely show up clearer than others. This could be due to a variety of reasons, including skin tone, amount of hair, and more protruding veins.

Our final database had 17 people, each person with 40 images. 20 images are suppose to be used for training, and 20 for testing. Specific images were not screened, and the first 40 images taken per person were used.

4. Algorithm Analysis for Preprocessing

Illumination Equalization: The infrared light source emits spherical waves. When this light projects on a nearby object, the reflected light will be unevenly illuminated. This is defined as an illumination problem. The simple solution is to take an image of a flat background at a desired distance. Then subtract this image from raw hand images.

Cropping: For each illumination-equalized hand image, we crop the concentrated vein region the hand to a size 100*100. The cropping algorithm is described in the following steps: 1) the illumination of the infrared light source is spherical, meaning intensity is large in the center of an illuminated area and gradually diminishes moving away from the center; 2) the first step of the cropping algorithm is to locate this center based on pixel intensity; 3) then crop a 100*100 region around this center.

Directional Filter: Of all edge enhancing techniques tried on vein images, directional filter produces the best result. The purpose for applying a directional filter is to enhance the veins in the image. The design of the directional filter by Subhasis Chaudhuri in 1989 is based on three properties of blood vessels. 1) Blood vessels usually have small curvature, so anti-parallel pairs may be approximated by piecewise linear segments. 2) Vessels have low reflectance of light, and the light intensity of vessels may be approximated by a Gaussian curve. 3) The width of blood vessel decreases as blood vessel is farther away from a branching point. Chaudhuri provide the following sets of equations for creating the direction filter, denoted by K’.

$$\bar{r}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}$$

$$\bar{p}_i = [u\ v] = \bar{p}\bar{r}_i^T.$$

$$N = \{(u, v) \mid |u| \le 3\sigma, |v| \le L/2\}$$

$$K_i(x, y) = -\exp(-u^2/2\sigma^2) \quad \forall \bar{p}_i \in N.$$

$$m_i = \sum_{\bar{p}_i \in N} K_i(x, y)/A.$$

$$K_i'(x, y) = K_i(x, y) - m_i \ \forall \bar{p}_i \in N.$$

We used a directional filter oriented at 12 different orientations, each of size 15*15. The filter coefficients are multiplied by a scale factor of 10 and truncated to an integer in the range -128 to 127. The filter is convolved with each hand image; each pixel of a filtered image corresponds to a maximal response to a specific orientation. The directional filter works very well for enhancing blood vessels.

Histogram Threshold: After applying directional filter, an automatic thresholding algorithm is applied to segment the veins. The algorithm used in this project is called histogram thresholding. The purpose of this method is to create a histogram of the pixel intensities of an image, ranging from 0 to 255. The thresholding is then applied to keep 1000 white pixels out of 72*72 pixels, intending for normalized binary images. This normalization handles the unbalanced case of a hand with plenty of veins and a hand with little veins. The threshold output is a binary vein image. This algorithm is simple and suffices the task of automatically thresholding veins.

5. Preprocessing Discussion

Cropping: The cropping algorithm has flaws. Sometimes veins visible to eye is not in cropped images. One reason is the cropped region is small compared to the size of a hand. In the future, a more complete segmentation of hand vein patterns may result from first segmenting the hand from a raw image.

Illumination Equalization: Preprocessing of hand images simply failed without some sort of illumination equalization. We devoted a lot of time in setting up appropriate lighting conditions and also positioning the hand in a manner that fewer illumination problems occur. The infrared LEDs were too bright and needed to be dimmed by placing rice papers over the light. The illumination equalization technique by subtracting a background image is simple but greatly aid preprocessing.

Directional Filter: We have tried several edge enhancement techniques including Sobel filter and high pass filters. The result of apply Gaussian directional filter is far more superior. The size of a Gaussian directional kernel is 15*15, a rather large size but worked better than smaller sized kernels for the following reasons. 1) The image has noticeable illumination problems even after illumination equalization. 2) The hand has lots of noise from hair on hand as well as other entities underneath the skin. For these reasons, a small kernel size would respond strongly to the differently illumination areas as well as noise on the hand.

Automatic Threshold: We chose to use a simple automatic thresholding algorithm. The histogram threshold is good enough for most hand images. Sometimes this algorithm has noticeable flaws, such as missing sections of veins, and broken vein branch points. More advanced automatic thresholding algorithm such as fuzzy vein tracking and vein probing is known to work well with retinal blood vessels.

6. Classification and Neural Network

Purpose: Classification is a necessary step in recognition. A good method of accomplishing classification is neural network. The goal of a neural network is to learn a large set of input vein patterns, and then classify these vein patterns based on features extracted through the network's learning of the input patterns.

Neural Network Architecture: A two layer neural network performing gradient descent learning is sufficient for classification of input patterns with no translation or rotation effects. The first layer has 72*72 input units, denoted by x, corresponding to a binary vein image. The output layer has 17 units, corresponding to 17 users. The weights connecting input and output units characterize the network's learning of the input patterns. These weights, denoted by w, are updated by gradient descent. Sigmoid activation of output units is denoted by g. y corresponds to the target output of an input pattern. Target outputs aid the gradient descent learning process.

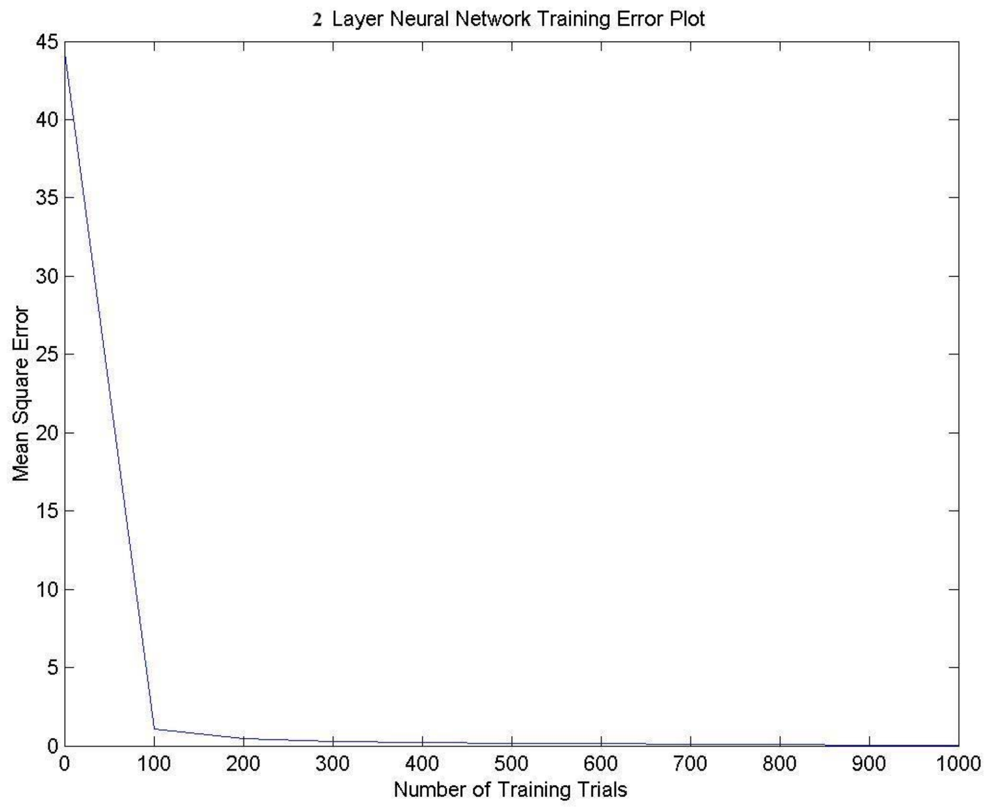
$$w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i g_i (1 - g_i) x_{ij}$$

$$g_i = g\left(\sum_{j=1}^{m} w_j x_{ij} \right)$$

$$\delta_i = y_i - g_i$$

$$g(h) = \frac{1}{1 + \exp(-h)}$$

Training of the neural network on PC took over half an hour. The neural network learns gradually across 1000 trials the classification of the input vein patterns. The mean square error is reported after every 100 trials and approaches zero near the end of training. Zero error indicates that the neural network has learned to generalize across various input patterns. In another words, the network has learned to classify the input vein patterns in training set.



Testing is done in EVM. Testing on each image has linear complexity. During verification testing, the weight matrix corresponding to an input user identity is retrieved and tested on a set of input vein images belonging to the correct user, other known users, and imposters. During identification testing, all weight matrices are retrieved and tested on a set of input vein images belonging to known users and imposters. Statistical accuracy of recognition during testing is recorded below.

● 2 Layer Neural Network:

– False rejection rate:	0.05
– False acceptance rate:	0.00
– Correct identification rate:	0.95
– Correct verification rate:	0.95

● Hamming Distance:

– False rejection rate:	0.20
– False acceptance rate:	0.20
– Correct identification rate:	0.80
– Correct verification rate:	0.80

● Imposter:

– Acceptance rate:	0.00

7. Classification and Neural Network Discussion

- Rotation and Translation: Our two layer neural network can not generalize to rotated or translated hand images. The reason is that the gradient descent learning rule does not work in the presence of translation and rotation effect. Currently there is a four layer competitive neural network capable of recognizing rotated and translated faces, built by E.T. Rolls and G. Wallis in 1997. Rather than building a more complex neural network, we ignored rotation and translation problems by carefully positioning the hand at the designated spot for each image taken.

- Training Set Size & Number of Features: Based on the rule of thumb that the size of training set should be at least ten times larger than the number of features, the architecture of our neural network clearly violated this rule of thumb. We used an input layer with 72*72 units, in which each unit represents a feature corresponding to a binary pixel. Our training set size is only 17*20 images. A solution to this issue is to introduce a hidden layer with a few units, maybe between 3 and 7. The purpose of this hidden layer is to extract features from an input image. And then these features will be used to classify input patterns. However, if a neural network can generalize correctly in two layers, adding another layer will not affect neural network's learning of the input patterns. So the features need not be explicitly present in a two layer network.

- Training Result: The neural network learns nearly perfectly the various input patterns, as seen in the error plot. One reason for such flawless training is that there is not much variation among training images.

- Testing Result: There tends to be at least one outlier in the training and testing sets of each user. The outlier is usually due to excessive hand motion, translation, or rotation effects. Except for these outliers, the neural network correctly recognizes known users during identification and verification. Tests on imposters show that the neural network is ambivalent about the identity of imposters, as seen in output activity levels of 0.5 for all output units. The reason is that the neural network does not know that the imposter is a known user, nor does it know that the imposter is not a known user.

- Hamming Distance: The Hamming distance test is a quick classification technique which is not as reliable as neural network. The results of Hamming distance test is still very high, indicating that there is not much variations among training images of a user.

8. EVM Signal Flow

Preprocessing:

1. PC to EVM: direction filter at 12 orientations, each of size 15 * 15.
2. PC to EVM: illumination equalized and cropped image, size 100*100
3. EVM: convolve image with filter at each orientation. For each pixel, choose maximal response of 12 orientations.
4. Repeat Step 2.

Testing:

1. PC to EVM: preprocessed binary image, size 72*72
2. PC to EVM: weight profile for 1 user
3. Repeat step 2 for all other weight profiles.
4. EVM to PC: testing result

Memory:

- directional filter at 12 orientations on-chip: 12 * 15 * 15 = 2700 bytes
- illumination equalized and cropped image on-chip: 100 * 100 = 10000 bytes
- After convolution, cut image to 72*72=5184 short int (2 bytes).
- Total = 27 kB + miscellaneous local memory < 32 kB
- No need for off chip storage.

Speed:

- Directional filter at 12 orientations are convolved with image simultaneously. Before unrolling, 70,000,000 cycles are required. Optimization is done by placing image and filter on-chip. After unrolling and parallelization by compiler, 17,000,000 cycles are required.
- HPI transfer 1.7 mB.

EVM Discussion:

- Directional Filter: Using FFT would have been faster for the convolution, and a second output space would be needed for storage in order to choose the maximal orientation response for each pixel.

- Testing is extremely fast. 72*72*17 multiplications and 72*72*17 additions are needed to test each image.

9. Demo

For our demo, we decided upon a variety of demonstrations to show that our project does, indeed, work. For the preprocessing section of our project, we allowed the tester to choose any image he wanted out of our training database, which had 17 users with 20 images per user. These images are raw. We then sent the images to the EVM and preprocessed the images, and then showed the result of our preprocessing on those images. We did not choose to demo preprocessing on all the images because of time constraints.

The second part of our demo consists of the tester randomly choosing an image from a testing set of 17 users with 20 testing images per user. We then tested the images compared to the other images on the EVM. The following is a sample output of what would be generated for identification.

- Identification output: <u>0.98</u> 0.01 0.02 0.00 0.01 0.01 0.02 0.02 0.01 0.01 0.01 0.01 0.01 0.00 0.01 0.02 0.01

Based on the results, we can see here that User 1, who is represented by the first number in the list has a 98% percent recognition rate. Therefore, the neural network is telling us that User 1 is person it is identifying based on the testing image inputted.

It is important to note that we did not demo the training/classification part of our project. This is because training for the neural networks takes over 30 minutes on the PC, which takes too long to demo. Because of this reasoning, we also did not implement training on the EVM because that would take much longer.

10. Prior 551 Work

There has been no prior work done on hand vein recognition in 551. Hand vein recognition is a new and emerging technology, with currently very few companies on the market utilizing this technology. However, hand vein recognition can be related to other types of identification and verification methods, such as iris recognition or fingerprint recognition.

11. References

1. S. Chaudhuri, “Detection of Blood Vessels in Retinal Images Using Two-Dimensional Matched Filters,” IEEE Transactions on Medical Imaging, VOL. 8, NO. 3, September 1989.

In this reference, Chaudhuri discusses the use of directional filters in retinal images. We used this reference learn how directional filters work and how it would fit into our project. This reference also provides the equations for the directional filters we used in the preprocessing part of our project.

2. A.W. Moore, “Regression and Classification with Neural Networks,” Carnegie Mellon University, September 25, 2001.

This reference is provided by A.W. Moore, who is an AI professor at Carnegie Mellon University. In the slides that he provides, we learn about how neural networks work, and how it could provide the answer to the how classification should be done in our project. Moore provides equations for sigmoid activation and gradient descent learning in 2 layer neural network, which we used in the classification/testing aspect of our project.

12. Conclusion

The hand vein recognition portal proposed in our solution has worked. Although the algorithms and implementation of our project may seem simple, it is important to realize that this technology is extremely new, and that few companies are selling the product on the market currently. We do realize that improvements may be made to our project but in the limited time constraints of one semester, it is extremely encouraging to see a functional hand vein recognition portal, with 2/3 of the project done on the EVM.

Improvements, perhaps implemented in a future 551 class, can include some of the following

- Using branch angles as a means of classifying an individual's veins
- Developing a more sophisticated preprocessing method such as more accurate cropping method and a more sophisticated method of making a binary vein image rather than the histogram method of thresholding we used.
- Develop some sort of training algorithm that allows for rotation and translation of the hand. This would make making a database extremely fast and efficient.

Our preprocessing

It was exciting to work on a project that had many possibilities and paths with no 'right' way of dong things. For example, using neural networks for classification was something that we never even thought about until midway through the project. Working on this project has shown that hand vein recognition is a valid method of identification and verification and that in the future, we can imagine its use increasing as more secure means of security is needed.

13. Code

```
/****************************************************************
 * Course:  ECE 18-551
 * File:    filterevm.c
 * Purpose: Performs the image filtering
 * Name:    Group 8, 2004
 ****************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include <common.h>
#include <board.h>          /* EVM library */
#include <pci.h>            /* PCI communication library */
#include <dma.h>

/* Defines for the size of the image */
#define X_SIZE 72
#define Y_SIZE 72
#define F_SIZE 100 * 100
#define KM_SIZE 15 * 15 * 12
#define OUTF_SIZE 72 * 72

char Km[KM_SIZE];
char frame[F_SIZE];
short int result[OUTF_SIZE];

/* Function prototypes */
int request_transfer(void *buf, int size, int command);
int wait_transfer();
int dma_copy_block(void *src, void *dest, int numBytes, int chan);

/* Version 1.  Quadruple *for* loop, out of external memory */
void do_comp1(void)
{
	int x,y,i,j,km_num;
	int max;
	int max_pixel = -(1 << 16), min_pixel = 1 << 16;
	int temp;
	int histo[256] = {0};
	// this does the max(correlation) and finds max_pixel and min_pixel
	for (x = 14; x < 86; x++)
	{
		for (y = 14; y < 86; y++)
		{
			max = -(1 << 16);
			for (km_num = 0; km_num < 12; km_num ++)
			{
				temp = 0;
				for (i = 0; i < 15; i++)
				{
					for(j = 0; j < 15; j++)
					{
						temp += frame[x - 7 + i + (y - 7 + j) * 100] *
```

```
                                                                Km[i + j * 15 + (15 * 15 * km_num)];
                                                }
                                        }
                                        temp = temp / 10;
                                        if (temp > max) max = temp;
                                }
                                result[(x - 14) + (y - 14) * 72] = max;
                                if (max > max_pixel) max_pixel = max;
                                if (max < min_pixel) min_pixel = max;
                        }
                }

                //now we linearly interpolate between 0 and 255
                // and build the histogram

                // clean the memory
                memset(histo, 0, 4 * 256);

                for (x = 0; x < 72 * 72; x++)
                {
                   temp = (255 * (result[x] - min_pixel))  / (max_pixel - min_pixel);
                   if (temp > 0) result[x] = temp;
                   else result[x] = 0;
                   histo[result[x]]++; // increase histogram
                }

                //we want 1000 white pixels
                temp = 0;
                for (i = 255; i >= 0 && temp < 750; i--) temp += histo[i];

                // threshold to what we wanted
                for (x = 0; x < 72 * 72; x++)
                {
                   if ( result[x] < i) result[x] = 0;
                   else result[x] = 255;
                }
}

int main(void)
{
  evm_init();              /* Initialize the board */
  pci_driver_init();       /* Call before using any PCI code */

  DMA_AUXCR = 0x00000010;     /* Set priority of HPI over CPU to avoid crashing */

  /* Get Km off the PC */
  request_transfer(Km, KM_SIZE, 0x00);
  wait_transfer();

  while( wait_transfer() == 0x03) // keep processing until the PC wants to quit
  {
         request_transfer(frame, F_SIZE, 0x01);
         wait_transfer();   /* Wait until it's done */

         do_comp1(); //   Compute first version of correlation
```

```
        request_transfer(result, OUTF_SIZE * 2, 0x02);//          Ask PC to fetch the results
        wait_transfer();                              //          Wait until it's done

        /* A command of 0x04 means evm is ready for next pixture*/
        request_transfer(NULL, 0, 0x04);
 }
}

/* Use mailbox 1 for address, 2 for size, and 3 for command */
int request_transfer(void *buf, int size, int command)
{
  amcc_mailbox_write(2, size);
  amcc_mailbox_write(3, command);
        pci_message_sync_send((unsigned int)buf, FALSE);
        return(0);
}

/* The PC will send a message when the transfer is complete.  Wait
        for that to happen */
int wait_transfer()
{
        unsigned int value;
        pci_message_sync_retrieve(&value);
        return(value);
}

/* Use the request_transfer framework to send data to be output to
        the screen.  The PC uses command 0x03 to indicate data to be
        output */
int pc_printf(char *message) {
 int len;

 len = strlen(message);
 while(len%4) len++;      /* Must be multiple of 4 */

 request_transfer(message, len, 0x03);
 wait_transfer();
 return(0);
}
```

```
/*****************************************************/
/* 18-551 Lab 3                                                                                          */
/* filterpc.c: Communication routines for filtering  */
/*                       (PC side)
*/
/* Author: <pwb@andrew.cmu.edu>                                                              */
/*  Last Modified: '2/16/00 10:51:46 AM'                               */
/*****************************************************/

#include <stdio.h>
#include <windows.h>
#include <evm6xdll.h>

/* Image size */
#define X_SIZE 280
#define Y_SIZE 280
#define F_SIZE X_SIZE*Y_SIZE


#undef LOAD_FILE
//#define LOAD_FILE    /* Uncomment to automatically load program */

/* Define these to match your setup */
#define INFILE "zeros.raw"
#define OUTFILE "results"
#define EVM_FILE "filter.out"

/* Structure for holding transfer information */
struct transfer_s {
            void *buffer;
            unsigned long size;
            unsigned long command;
};

/* Function prototypes */
#ifdef LOAD_FILE
int load_file(HANDLE hBd, LPVOID hHpi);
#endif
int wait_request(struct transfer_s *ts);
int send_data(struct transfer_s *ts, void *local_buf);
int get_data(struct transfer_s *ts, void *local_buf);
void hpi_write_word(ULONG addr, ULONG data);

/* Global vars */
unsigned char frame[F_SIZE];        /* Input */
short int result[F_SIZE];                                     /* Output */

HANDLE hBd    = NULL;
LPVOID hHpi = NULL;
HANDLE h_event;


/* Writes 'result' to a file.  number is the integer to append to the
            end of the filename */
void data_to_file(int number)
{
            FILE *outfile;
            char fname[255];

            sprintf(fname, "%s%i.raw", OUTFILE, number);
            if ((outfile=fopen(fname, "wb")) != NULL) {
                    fwrite(result, F_SIZE, sizeof(short int), outfile);
                    fclose(outfile);
            }
            else {
                    fprintf(stderr, "Error! File %s could not be opened for write\n", fname);
            }
}

int main(int argc, char **argv) {
            int program_exit=0, i=0;
            char s_buffer[80];
            char temp[1024];
            struct transfer_s ts;
            FILE *infile;

            /* Read the image file into frame */
            if ((infile=fopen(INFILE, "rb")) != NULL) {
                    fread(frame, F_SIZE, 1, infile);
                    fclose(infile);
            }
            else {
                    fprintf(stderr, "Error! File %s could not be opened for read\n", INFILE);
            }


            /* Open the board */
#ifdef LOAD_FILE
            hBd = evm6x_open(0, 1);
#else
            hBd = evm6x_open(0, 0);
#endif

            if ( hBd == INVALID_HANDLE_VALUE ) {
                    fprintf(stderr, "Couldn't open board\n");
                    exit(1);
            }
            fprintf(stderr, "Opened connection to board...\n");

            /* Also open connection to HPI */
       hHpi = evm6x_hpi_open(hBd);
       if ( hHpi == NULL ) exit(4);

#ifdef LOAD_FILE
            load_file(hBd, hHpi);
            fprintf(stderr, "Loaded program...\n");
#else
            /* Set DMA AUX priority greater than CPU priority, so we
                    don't lock the PCI bus */
       hpi_write_word(0x01840070 /*Addr*/, 0x00000010 /*Data*/);

            /* Reset the mailboxes and FIFO */
            hpi_write_word(0x0170003C, 0x0e000000);
#endif

            /* Setup a windows event so we don't have to poll for incoming
                    messages */
            evm6x_clear_message_event(hBd);
            sprintf( s_buffer, "%s%d",EVM6X_GLOBAL_MESSAGE_EVENT_BASE_NAME, 0);
            h_event = OpenEvent( SYNCHRONIZE, FALSE, s_buffer );

            /* Main loop... Waits for messages from the EVM and does a transfer
                    depending on the value of the message received */
            while(!program_exit) {
                    wait_request(&ts);
                    fprintf(stderr, "Transfer request: CMD %x, SIZE %i, ADDRESS %x\n", ts.command,
                            ts.size, ts.buffer);

                    /* Now based on the command that was sent, do something */
                    switch(ts.command) {
                    case(0x01): /* Send frame */
                            if(ts.size != F_SIZE) fprintf(stderr, "Wrong size!!!\n");
                            send_data(&ts, frame);
                            break;
                    case(0x02): /* Get frame */
                            if(ts.size != F_SIZE*sizeof(short int))
                                    fprintf(stderr, "Wrong size\n");
                            get_data(&ts, result);
                            data_to_file(++i);/* Output frame to a new file */
                            break;
                    case(0x03): /* Print string */
                            get_data(&ts, temp);
                            printf("%s\n", temp);
                            break;
                    case(0x04): /* Exit program */
                            program_exit = 1;
                            break;
                    default:
                            fprintf(stderr, "Unknown command\n");
                            break;
                    }
            }

            /* Clean up and exit */
            if (!evm6x_hpi_close(hHpi)) exit(9);
            if (!evm6x_close(hBd)) exit(16);
            return(0);
}

/* Waits for windows event signalling incoming message.  Then reads
            address from mailbox 1, size from mailbox 2, and command from
            mailbox 3 */
int wait_request(struct transfer_s *ts)
{
            /* wait for event signaling a message from the DSP */
            WaitForSingleObject( h_event, INFINITE );

            if(!evm6x_retrieve_message(hBd, (unsigned long *)&ts->buffer))
```

```
                fprintf(stderr, "Error retrieving 1...\n");
        if(!evm6x_mailbox_read(hBd, 2, (unsigned long *)&ts->size))
                fprintf(stderr, "Error retrieving 2...\n");
        if(!evm6x_mailbox_read(hBd, 3, (unsigned long *)&ts->command))
                fprintf(stderr, "Error retrieving 3...\n");

        return(0);
}

/* Size and address are given in struct ts.  local_buf is the address of
        the PC buffer to send */
int send_data(struct transfer_s *ts, void *local_buf)
{
        unsigned long int ulLength = ts->size;

        /* Write the data to EVM memory*/
        if (!evm6x_hpi_write(hHpi, (PULONG)local_buf, (PULONG)&ulLength, (ULONG)ts->buffer)) {
                fprintf(stderr, "HPI write error\n");
                exit(1);
        }
        if (ulLength != ts->size) {
                fprintf(stderr, "HPI only wrote %i bytes\n");
                exit(1);
        }

        /* Use a message to signal the EVM that the transfer is done */
        if (!evm6x_send_message(hBd, (PULONG)&ts->command)) {
                fprintf(stderr, "Send message error!\n");
                exit(1);
        }
        return(0);
}

/* Same format as send_data */
int get_data(struct transfer_s *ts, void *local_buf)
{
        unsigned long int ulLength = ts->size;

        /* Read data from EVM memory */
  if (!evm6x_hpi_read(hHpi, (PULONG)local_buf, (PULONG)&ulLength, (ULONG)ts->buffer)) {
                fprintf(stderr, "HPI write error\n");
                exit(1);
        }
        if (ulLength != ts->size) {
                fprintf(stderr, "HPI only wrote %i bytes\n");
                exit(1);
        }

        /* Signal EVM that transfer is done */
        if (!evm6x_send_message(hBd, (PULONG)&ts->command)) {
                fprintf(stderr, "Send message error!\n");
                exit(1);
        }
        return(0);
}
```

```
/* Write a single 32-bit word to EVM memory */
void hpi_write_word(ULONG addr, ULONG data)
{
        ULONG ulLength = 4;

        if (!evm6x_hpi_write(hHpi, &data, &ulLength, addr)) {
                fprintf(stderr, "Error writing word via HPI\n");
                exit(1);
        }
        if (ulLength != 4 ) {
                fprintf(stderr, "Error writing word via HPI\n");
                exit(1);
        }
}

#ifdef LOAD_FILE
/* Load the .out file and run the program.  Code Composer must not
        be running */
int load_file(HANDLE hBd, LPVOID hHpi)
{
  if ( !evm6x_reset_board(hBd) ) exit(2);
  if ( !evm6x_reset_dsp(hBd,HPI_BOOT) ) exit(3);
  if ( !evm6x_init_emif(hBd, hHpi) ) exit(5);

  /* Load program */
  if (!evm6x_coff_load(hBd,hHpi,EVM_FILE,0,0,0)) exit(8);

  /* Set HPI priority and reset mailboxes */
  hpi_write_word(0x01840070 /*Addr*/, 0x00000010 /*Data*/ );
  hpi_write_word(0x0170003C, 0x0e000000);

  if (!evm6x_unreset_dsp(hBd)) exit(10);
  if (!evm6x_set_timeout(hBd,1000)) exit(11);
        return(0);
}
#endif
```

```
#include "trainBackPropNet.h"
#include "testBackPropNet.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void activateOutput(float *generalizedWeights, float *testSet, float *activityOutput);


/*********************************
*
* Test Neural Network
*
* @return classification of an input image
*
*********************************/

/*
 * prior to using testBackPropNet:
 *
 * initialize array:              float activityOutput[NUM_OUTPUT];  //activation of output units
 *
 * use array generalizedWeights from trainBackPropNet
 * use input testing image
 */
#include "trainBackPropNet.h"

void testBackPropNet(float *generalizedWeights, float *testSet, float *activityOutput, int numUsers)
{
   /*
    * Test
    */

        for (int i = 0; i < numUsers; i++)
                calcActivation(activityOutput, testSet, generalizedWeights, i, 1, 1);
}
```

```
#ifndef __TESTNEURALNETWORK_H__
#define __TESTNEURALNETWORK_H__

void testBackPropNet(float *generalizedWeights, float *testSet,
                                                                                                  float
*activityOutput, int numUsers);

#endif
```

```
#include "trainBackPropNet.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

float computeError(float *activation,int user,int numImgs,int numUsers);
void updateWeights(float *weights, float *activation, float *trainingSet,
                                                                                int user, int numImgs, int
numUsers);


void writePartialResults(float *weights, int numWeights, float error)
{
        FILE *f = fopen("intermediateWeights.txt", "a");
        if (f == NULL) printf("error writing to intermediateWeights.txt \n");
        fprintf(f,"error: %.6f weights:", error);
        for (int i =0; i < numWeights; i++)
                fprintf(f," %.6f", weights[i]);
        fprintf(f, "\n");
        fclose(f);
}

void trainBackPropNet(float *trainingSet, float *weights,
                                                                                              float
*errorPlot, int numImgs, int numUsers)
{
 float *activation;
        for(int i = 0; i < numUsers * WIDTH * HEIGHT; i++) weights[i] = float(0.5) * (rand() % 1000) /
float(1000.0);
        activation = new float[numUsers * numImgs * numUsers];

        for (int user = 0; user < numUsers; user++)
        {
                for (int i = 0; i< NUM_TRAINING_TRIAL; i++)
                {
                        calcActivation(activation, trainingSet, weights, user, numImgs, numUsers);
                        updateWeights(weights, activation, trainingSet, user, numImgs, numUsers);
                        printf("user = %d, error = %f\n", user, computeError(activation, user, numImgs,
numUsers));
                }
        }

        writePartialResults(weights, 9, 0);
        delete activation;
}

float computeError(float *activation,int user,int numImgs,int numUsers)
{
        float error = 0;
        float dAct;
        float target;
        for (int j = 0; j < numImgs * numUsers; j++)
        {
                if (j /  numImgs == user) target = 1;
                else target = 0;
```

```
				dAct = target - activation[j + user * numImgs * numUsers];
				error += dAct * dAct / 2;
	}

	return error;
}

void updateWeights(float *weights, float *activation, float *trainingSet,
											int user, int numImgs, int
numUsers)
{
	float target;
	float dw;
	for (int w = 0;w < WIDTH * HEIGHT; w++)
	{
		dw = 0;
		for (int img = 0; img < numImgs * numUsers; img++)
		{
			if (img / numImgs == user) target = 1;
			else target = 0;
			dw += (target - activation[img + user * numImgs * numUsers])
					* activation[img + user * numImgs * numUsers]
					* (1 - activation[img + user * numImgs * numUsers])
					* trainingSet[w + img * WIDTH * HEIGHT] *
float(LEARNING_RATE);
		}
		weights[w + user * WIDTH * HEIGHT] += dw;
	}
}

void calcActivation(float *activation, float *trainingSet, float *weights,
												int user, int
numImgs, int numUser)
{
	float temp = 0;
	for (int j = 0; j < numImgs * numUser; j++)
	{
		temp = 0;
		for(int i = 0; i < WIDTH * HEIGHT ; i++)
		{
			temp += weights[ i + user * WIDTH * HEIGHT] *
			trainingSet[i + WIDTH * HEIGHT * j];
		}
	//	printf("%f \n", temp / WIDTH / HEIGHT);
		activation[j + user * numImgs * numUser] = float(1.0) / (1 + exp(-1 * temp / WIDTH /
HEIGHT));
	}
}
```

```
#ifndef __NEURALNETWORK_H__
#define __NEURALNETWORK_H__

#define WIDTH                      (72 * 72)
#define HEIGHT                     1
#define NUM_OUTPUT                      1
#define NUM_TRAINING_TRIAL   1000
#define LEARNING_RATE       0.5
#define ALPHA              0.1      //momentum parameter

void trainBackPropNet(float *trainingSet, float *weights,
                                                                              float
*errorPlot, int numImgs, int numUsers);

void calcActivation(float *activation, float *trainingSet, float *weights,
                                                                    int user, int
numImgs, int numUser);
#endif
```

```
/***************************************************************
 * Course:  ECE 18-551
 * File:    testingevm.c
 * Purpose: Performs the testing
 * Name:    Group 8
 ***************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include <common.h>
#include <board.h>          /* EVM library */
#include <pci.h>            /* PCI communication library */
#include <dma.h>

/* Defines for the size of the image */

float weights[72 * 72];
float testResult[17];
char testArray[72 * 72];

/* Function prototypes */
int request_transfer(void *buf, int size, int command);
int wait_transfer();
int dma_copy_block(void *src, void *dest, int numBytes, int chan);

/* Version 1.  Quadruple *for* loop, out of external memory */

float doTesting(void)
{
        int i;
        float temp = 0;

        for(i = 0; i < 72 * 72 ; i++)
        {
                temp += weights[ i] *
                testArray[i];
        }
        //      printf("%f \n", temp / WIDTH / HEIGHT);
        return 1.0 / (1 + exp(-1 * temp / 72 / 72));
}


int main(void)
{
  int count;
  evm_init();              /* Initialize the board */
  pci_driver_init();       /* Call before using any PCI code */

  DMA_AUXCR = 0x00000010;     /* Set priority of HPI over CPU to avoid crashing */

  /* Get Km off the PC */
  request_transfer(testArray, 72*72, 0x00);
  wait_transfer();
  count = 0;
  while( wait_transfer() == 0x03) // keep processing until the PC wants to quit
  {
              request_transfer(weights, 72*72*4, 0x01);
              wait_transfer();   /* Wait until it's done */

              testResult[count ++] = doTesting(); //   Compute first version of correlation

              /* A command of 0x04 means evm is ready for next pixture*/
              request_transfer(NULL, 0, 0x04);
  }

  request_transfer(testResult, 17 * 4, 0x05);
}

/* Use mailbox 1 for address, 2 for size, and 3 for command */
int request_transfer(void *buf, int size, int command)
{
   amcc_mailbox_write(2, size);
   amcc_mailbox_write(3, command);
            pci_message_sync_send((unsigned int)buf, FALSE);
            return(0);
}

/* The PC will send a message when the transfer is complete.  Wait
            for that to happen */
int wait_transfer()
{
            unsigned int value;
            pci_message_sync_retrieve(&value);
            return(value);
}

/* Use the request_transfer framework to send data to be output to
            the screen.  The PC uses command 0x03 to indicate data to be
            output */
int pc_printf(char *message) {
  int len;

  len = strlen(message);
  while(len%4) len++;       /* Must be multiple of 4 */

  request_transfer(message, len, 0x03);
  wait_transfer();
  return(0);
}
```