MIT Course 6.111: Digital Electronics Lab

Fingerprint Identification

Kevin Amendt
David Friend

May 19, 2006

Abstract

Fingerprint readers are fast becoming commonplace as secure alternatives to text-based authentication. We develop a programmable fingerprint identification system. The system can store users' fingerprints in a memory, and compare future inputs against this database. We use a video camera for image capture.

Our matching algorithm is unique in that it compares the frequency content of two fingerprints. A fast Fourier transform is performed along horizontal and vertical slices of the image. These coefficients are compared against the coefficients of another print. If the error is below a threshold, we determine the prints to be a match.

We implemented our design on a Xilinx FPGA, using the 6.111 labkit. The design is highly modular: the matching algorithm can be easily changed without changing other parts of the system. Likewise, the image capture can be replaced without major changes to the system.

Contents

List of Figures

1 Introduction

Biometric identification has enjoyed a surge of popularity in the last few years. Particularly, fingerprint authentication has become a mainstream product. A search on Amazon.com for "fingerprint scanner" returns two dozen or so results, including products from companies like Microsoft and Logitech. The scanners come in various packages: standalone, built into keyboards, and imbedded in USB keys.

To see the advantages of biometric identification, consider that authentication schemes require the user to prove his identity: he must possess unique knowledge or have a unique possession (or some combination of these two). The two most common types of authentication are keys, such as in the key to your house or car, and passwords. The obvious pitfall of keys and passwords is that people tend to loose their keys and forget their passwords. More critically, these authentication schemes are transferable, and therefore insecure. Biometric authentication, on the other hand, cannot be lost nor transfered, making it an ideal method for user authentication.

Our focus is on fingerprint authentication. Traditionally, fingerprint matching algorithms are implemented in software. As detailed in Section 2, programs extract certain features of the print. The type, number, and location of these features is unique to each person. We explore whether it is possible to accomplish the matching using a hardware implementation. Instead of performing feature extraction, which is tedious even in software, we try to match in the frequency domain.

2 Theory of Matching Algorithm

A fingerprint differs from scan to scan by some combination of two dimensional translation, rotation, and scaling; and a three dimensional 'rolling', which occurs when the user does not place his finger at the same elevation from scan to scan. These variations, especially translation and rolling, make matching a challenge.

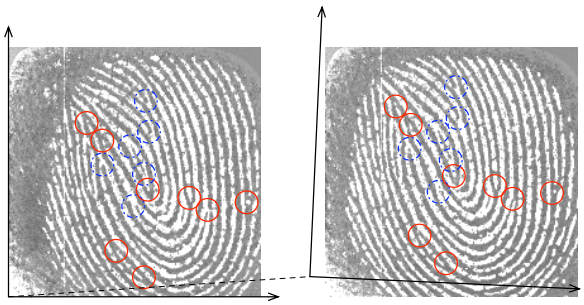

Figure 1: Using feature extraction to match fingerprints

The usual fingerprint matching algorithms use feature extraction. These programs examine the ridges to determine where they begin and end. Figure 1 illustrates this method. Shown are two scans of the same fingerprint.[1] Two types of features are highlighted. Circled in red (solid lines) are locations where ridges end. Circled in blue (dashed line) are locations where ridges split. It is readily apparent that the print on the right is a translated and rotated copy of the left print.

2.1 Alternative Approach

While the traditional algorithms are easily implemented in software, it is another matter design them in hardware. (Although it can be done: S. Patel demonstrated a MATLAB script to match fingerprints, and MATLAB translates easily to hardware[2].) We required a simple method that we could describe in hardware in a few weeks. Following the suggestion of M. Wennergren[3], we focused on the frequency domain.

The frequency domain encodes some measure of the spacing of the ridges on the fingerprint. Let us say that we can always identify two key points on every fingerprint. Then the Fourier transform along the line connecting these two points should be an invariant. This method accounts for translation, rotation, and scaling, and is quite sufficient for our purposes.

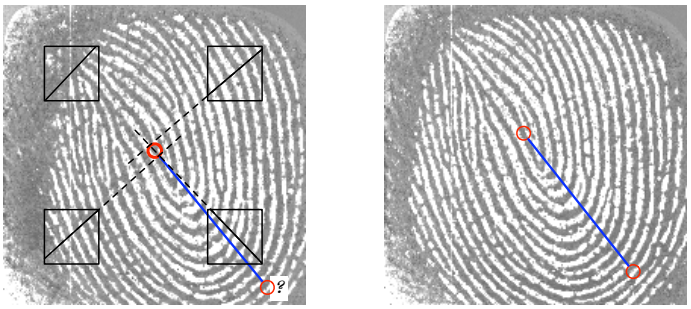

Figure 2: Matching fingerprints in the frequency domain

We used the center of the fingerprint as the first key point. We can easily find the center by taking a pseudo-gradient in four corners and finding where at least two of the lines intersect. This method is illustrated in Figure 2. The second point, however, is intractable. By the time you have described an algorithm to pick out a second point, you might as well perform full blown feature extraction.

We further simplified our problem to use only the center point. We then select the column and row through the center and match the frequency content to the frequency content of the center column and row on another fingerprint. This method preserves translational invariance, but does not match rotated or scaled images.

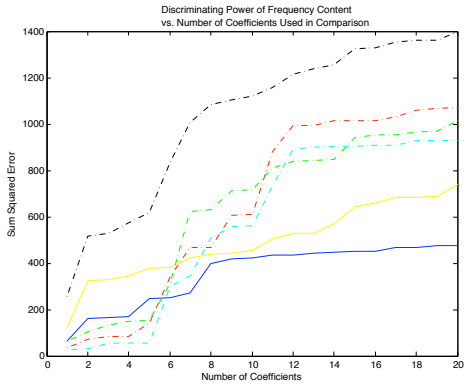

Figure 3: Ability of algorithm to discriminate different fingerprints

The comparison in frequency space is performed as a straight forward, summed-squared-error calculation. A low error implies the prints match better. If the method works well, then we can always find a threshold error, below which we say that the prints match. Figure 3 shows how well this scheme works. The dashed lines represent the error between pairs of non-matching prints. The blue and yellow (solid) lines show the error between pairs of matching prints.

The plots also show that the difference between the error associated with a match and the error associated with a non-match flattens out around $n = 16$. There is no point in calculating the remaining coefficients. When summing the error, we discard the constant component and look at the first 16 coefficients. Conceptually, this cutoff removes the high frequency components, which are likely to fluctuate wildly from scan to scan due to noise.

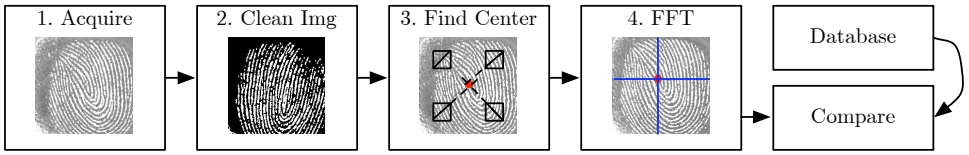

Figure 4: Algorithm Flow Chart

Figure 4 describes our final algorithm for matching fingerprints. Notice that we have "cleaned" the image, which includes converting grayscale to black and white, and interpolating to reduce noise. Details of these procedures are described in Section 4.

2.2 Setbacks and Further Simplification

We originally wanted to use a fingerprint sensor from Veridicom International. Unfortunately we damaged it in the process of trying to attach wires to the surface mount, 0.5 mm pitch pins. This setback forced us to look for other means of acquiring the fingerprint. It also meant that a good week of work was wasted.

In order to complete the project in some form, we were forced to eliminate the find center step. Unfortunately this simplification means that our project is no longer translational invariant. We use constant coordinates for the center of the fingerprint. The user must position the fingerprint the same way each scan, or our double-simplified algorithm will not work.

3 Operation

There are two modes of operation: a user can be added to the database and an unknown user can be identified based on the entries in the database. The mode is controlled by switch 7. On corresponds to add user, off corresponds to identify. Figure 5 shows the user interface.

3.1 Adding Users to Database

To add users to the database, turn switch 7 to the on position. The LED display will show `Add User 0`. The number at the end of the display indicates the user ID associated with the person. The user ID can be changed by toggling switches 0 through 2. The camera should be positioned over the user's fingerprint (use a 2D printout, not the finger itself). Then press the

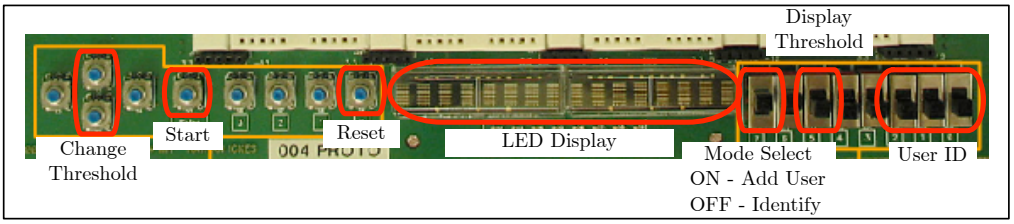

Figure 5: FPGA User Interface

start button. The LED display briefly switches to `Processing`, and then displays the result of the operation for a second.

As an example, let us add a user to entry number 3. Turn switch 7 on. Select user ID as 3 by setting switch 2 off, switch 1 on, and switch 0 on (`011`). Position the camera and press the start button. `Processing` flashes briefly, and the result is displayed for one second: `Added User 3`. The system returns to idle mode and another operation can be performed.

The database can be cleared by pressing the reset button.

3.2 Identifying Users

To identify users, turn switch 7 to the off position. The LED display will show `Identify`. Position the camera over the fingerprint (a 2D picture, not the finger itself), and press the start button. The display will flash `Processing` and then display the result, either which entry it matched (`Matched 1`) or no match (`No Match`). Obviously if there are no entries in the database, this operation will always show no match.

3.3 Setting the Threshold

As described in Section 2.1, the matching algorithm requires a threshold. While this value is initialized to what we believe to be a reasonable value, it can be changed. The current threshold can be displayed by switching switch 5 to on. The hexadecimal representation of the threshold displays in the LED display. The value can be adjusted by pressing the up and down buttons. Not shown in Figure 5, a coarse adjustment is also available with the left and right buttons. The threshold will return to its default value of `4E00` on reset.

4 Implementation

We implemented our design in Verilog and programmed it onto a Xilinx FPGA. The entire system is controlled by the controller FSM. The overall system is shown in Figure 6. Arrows represent the data flow. We have omitted control signals from this plot for clarity. Shaded blocks represent buffers and are passive modules.

We capture the fingerprint with an NTSC compatible video camera. Frames are continuously written into the scanned image buffer and also to a display buffer, which is displayed on the monitor. When the user presses the enter button, the controller sends a start signal to the processing controller and simultaneously stops the 'NTSC to BRAM' module from writing to the scanned image buffer and display.

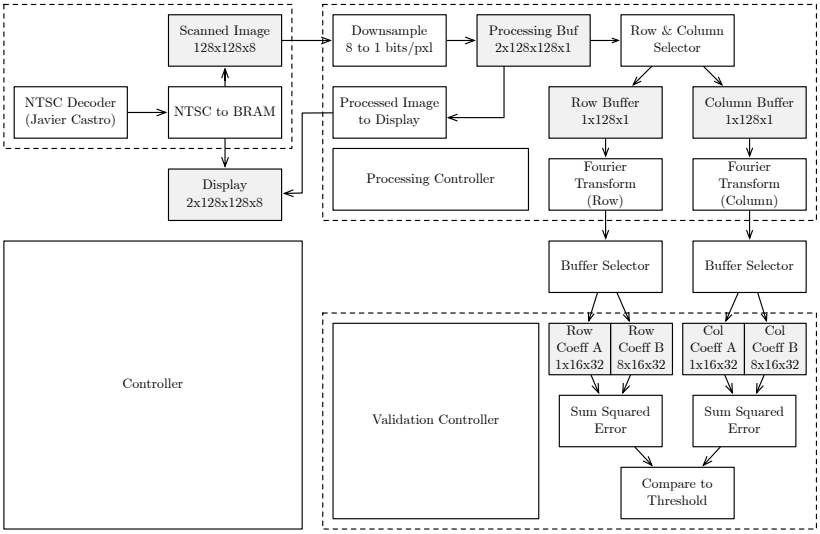

Figure 6: Dataflow through the identification system

The processing controller sends a start signal to the Downsample module, which reads the in the scanned image at 8 bits per pixel of grayscale, and converts it to a black and white image (1 bit per pixel). The threshold, above which pixels are classified as white and below which they are set to black, is the average brightness of the image.

When the downsample module has completed, the processing controller sends a start signal to the row and column selector. This selects the appropriate row and column from the image and stores them into row and column buffers. An FFT module calculates the first 17 coefficients of the Fourier transform of the row and column.

There are two modes in the system. A fingerprint can be read in for identification ('identify'), or it can be read in to be added to the database of known fingerprints ('add user'). The mode is selected with switch 7 on the FPGA. When the switch is off, the system is in identify mode. Conversely, when the switch is on, the system is in add user mode. An ID number in the range 0 to 7 can then be selected with switches 0-2.

If the system is in add user mode, the Fourier coefficients are stored in the B buffers. The B buffers have space for eight sets of coefficients, corresponding to the eight different user ID's. In add user mode, after storing the coefficients to the appropriate entry in the database, the system returns to idle mode.

If the system is in identify mode, the Fourier coefficients are stored in the A buffers. The controller then sends a start signal to the validation unit. The validation reads in the coefficients from the A buffer and compares them to each set of coefficients in the B buffer. The validation unit determines if the A buffer entry matches any B entries, and if so, which entry in the B buffer was matched.

This process takes about 2 ms. The result (i.e. 'Added User 6', 'Matched 1', or 'No Match') is displayed on the labkit's led matrix for a second before the system returns to idle mode.

4.1 Conventions

We choose to use 128 by 128 pixels for our fingerprints.

4.2 Buffers, Image Buffers, and Double Buffers: David

Because we are storing minimal data, we are able to use block RAM for our entire project. We synthesized a block RAM in Corgen which had 16384 8-bit addresses. We call this a `buffer_128x128x8`. This storage unit is used in the VGA display's double buffer, the original image storage, and in the processing unit's double buffer.

We also have three other sizes of buffers: a `buffer_1x128x1` to store a single row or column, a `buffer_1x16x32` to store a single set of Fourier coefficients in the A buffers, and a `buffer_8x16x32` to store 8 sets of Fourier coefficients in the B buffers.

Because we constant access the image sequentially (requesting each pixel in order, from 0,0 to 128,128), it was convenient to wrap the buffer with a sequential access interface. The `img_buffer` module contains a pin to enable sequential access. When this input is high, the data are not accessed through the address. The module accepts a `reset_pxl_pos` pulse, which returns the internal counter to zero. Then a single pixel is read through the `re` (read enable) pin. Each time `re` is high on the rising clock edge, the value of the pixel pointed to by the internal counter is put on the data out line, and the internal counter is incremented.

Because the `img_buffer` registers its inputs, the read and write delays increase. The read delay, between `addr` being valid at the rising clock edge, or, if sequential access is enabled, `re` being high at the rising clock edge, to the data becoming valid is two clock cycles. The write delay is one clock cycle.

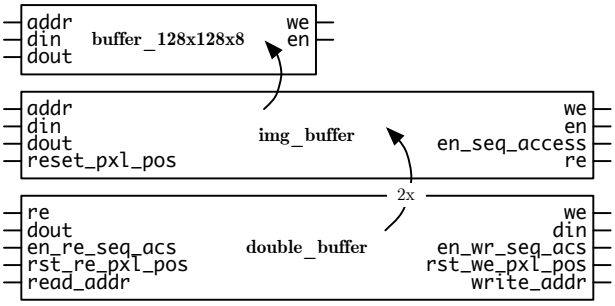

Figure 7: Buffer Modules

Finally, we have a third memory module, a double buffer. A double buffer is used both in the processing unit and the display unit. A double buffer uses two 128x128x8 `img_buffer`s. One buffer is designated as a 'read' buffer, and the other is a 'write' buffer. In the processing unit, this is useful when we want to clean the image. We can read in the current copy of the image from the 'read' buffer and write a modified copy to the 'write' buffer. When we are done, we simply switch the buffer designations. The next processing step reads in the improved image, and the older version is overwritten.

In the VGA display module, this double buffer structure is useful to seamlessly update the display. The 'read' buffer is the one that is displayed to the user, while the camera writes to the 'write' buffer. When the camera has read an entire frame, the buffer switches the designations. Now the new image is displayed and the older image is overwritten.

The double buffer module also implements sequential access. The read buffer and the write buffer have separate sequential access enable pins. In this manner the read can be sequential, and the write random access, if desired. The double buffer adds a one cycle read and write delay to the

image buffer due to its registering of the inputs. The read delay is three clock cycles and the write delay is two.

4.3 Image Capture: David

We originally intended to use a commercial scanner from Veridicom International to capture the fingerprint (model number FPS200). When we received the scanner, the pins were of a non-standard pitch. Under the advice of Gim Hom, we tried to solder the connections directly. Unfortunately, the scanner was damaged in the process and we were forced to abandon a week's worth of effort and quickly throw together and image capture from a NTSC compatible video camera.

We used Javier Castro's NTSC decoder and his NTSC to ZBT ram modules to jump-start our image capture. We modified his NTSC to ZBT module to write into our display buffer and into our original image buffer. Our NTSC to BRAM module constantly presents data to the display buffer and the original image buffer. The controller FSM lets the `we` signal pass when the system is in ACQUIRE mode, and sets the `we` inputs to these two memories to low when the user presses the start button.

While the system is in the AQUIRE mode, the display's buffers are swapped at the end of every frame, about 30Hz. This swapping gives the effect of video. Note that the NTSC to BRAM module samples the incoming picture by taking only the top left 256x256 pixels. It then only writes the odd lines and even columns, resulting in a 128x128 image.

4.4 VGA Display: David

The VGA display module determines the colors of the pixels to display on the screen. It consists of two parts: a black background, and a 128x128 image displayed in the center. It reads the image data from the 'read' buffer of the double buffer contained within the module. Other modules can modify the displayed image by writing to the 'write' buffer, and then pulsing the `cycle_disp`, input. The `cycle_disp` swaps the read and write buffer designations, presenting the new image to the user and allowing other modules to write a new frame.

The read delay is three clock cycles, which has the effect of shifting the image over by three columns. The columns wrap around, and the first three columns of the displayed image are actually the last three columns of the image in the 'read' buffer. We fix this shift by delaying all the horizontal and vertical sync by three clock cycles, while keeping the line and pixel count the same. The effect is that the pixel and line count signals into the VGA display module are advanced by three clock cycles, thus fixing the shift.

4.5 LED Display: David

The LED display shows the current operating mode of the system. The four main modes are Identify, Add User, Processing, and displaying the result of the last operation. Additionally, the LED display will switch to showing the current threshold level when switch 5 is turned on.

The implementation is straight forward use of the `alpha_display` module. We wrap 16 instances of the `alpha_display` module into the `display_state` module, one for each character. The controller FSM provides a three bit selector input, the display state, and the module displays the current mode of the system.

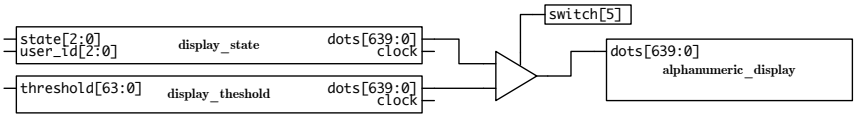

Figure 8: Block Diagram of LED Display Module

The threshold display is handled by the `display_threshold` module, which also has 16 instances of the `alpha_display` module, one for every 4 bits of the threshold level. When switch 5 is off, the dots signal into the `alphanumeric_display` module comes from `display_state`. When switch 5 is on, the dots signal comes from the `display_threshold`. The controller FSM has no knowledge of the state of switch 5.

4.6 Controller FSM: David

The controller module is the big picture controller for the system. From the point of view of the controller module, the system can be in one of four states: ACQUIRE, PROCESS, VALIDATE, and DISPLAY_RESULT. Also, the controller updates the LED display by changing its state input.

The ACQUIRE mode is the idle state. The system is constantly acquiring video frames and updating the VGA display. In this mode the user can select either identify or add user, via switch 7. The system waits in this state until the the user presses the start button.

When the user presses the start button, the controller moves to the PROCESS state. Here it pulses the start signal for the processing controller module and then moves to a WAIT state where it remains for as long as the processing controller module asserts its busy signal.

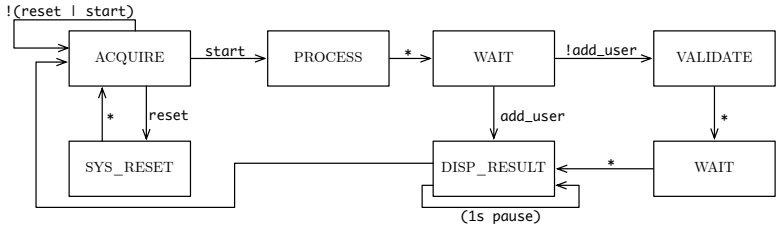

Figure 9: Controller Module State Transition Diagram

After the processing controller is done, the controller module makes a choice. If the the mode was Add User, then it is done. It moves to the DISPLAY_RESULT state and remains there for a second so that the user can see the result of his action displayed on the LEDs. If the mode was Identify, an additional validation step is required.

The controller module pulses the validation module's start signal and then moves to a WAIT state where it remains for as long as the validation controller module asserts its busy signal. After the validation controller is done, the controller moves to the DISPLAY_RESULT state to pause and inform the user about the result of the identification.

4.7 Processing FSM: David

The processing FSM controls the processing steps of the system. These states are abstracted from the main controller for modularity.

The processing controller begins on a start pulse from the controller. It sends a start signal to the downsample module, which reads in the image from the scanned image buffer and writes it to the processing unit's double buffer.

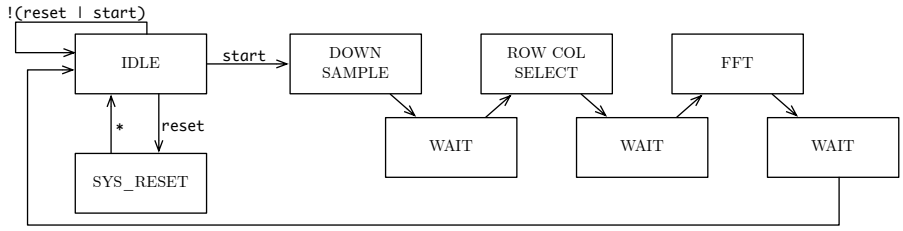

Figure 10: Processing Controller State Transition Diagram

Next the processing controller starts the row and column selector. When this process has finished, the processing controller starts the Fourier transform. The buffer selectors start from a reset signal from the FFT modules.

After all of these processes have finished, the processing controller sets its busy signal to low and returns to the idle state.

4.8 Downsample Module: David

The downsample module converts an 8 bit per pixel image into a binary, 1 bit per pixel image. It consists of two main steps. The first is to calculate the threshold value, equal to the average value of all the pixels in the original sample. Thresholding dynamically, and not against a constant value, ensures that we achieve the maximum contrast of the image. After computing the threshold value, the module then reads the original image again, this time outputting the new binary image by comparing each original pixel to the threshold.

Other than to binarize the image, the other main contribution of the downsample module is to read the image from the scanner buffer into the processing buffer where the processing modules can easily access and modify it.

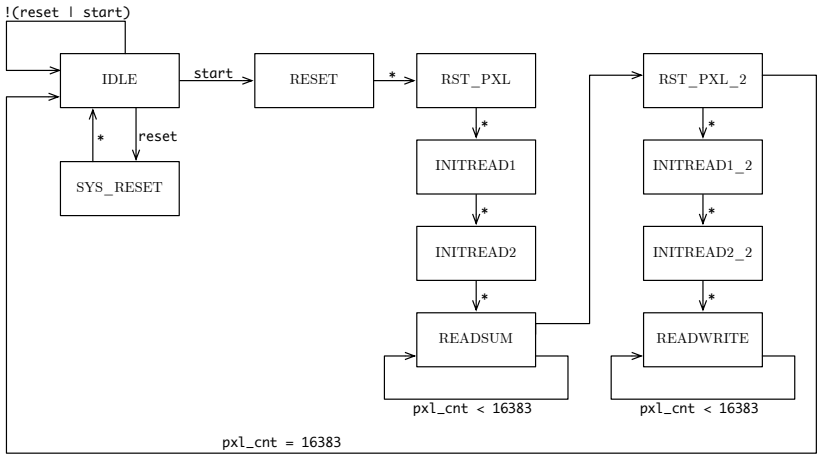

Figure 11: Downsample State Transition Diagram

4.9 Display Processed Image: David

For debugging purposes it was convenient to implement a module to display the processed image. This module reads the modified image from the processing buffer into the display buffer.

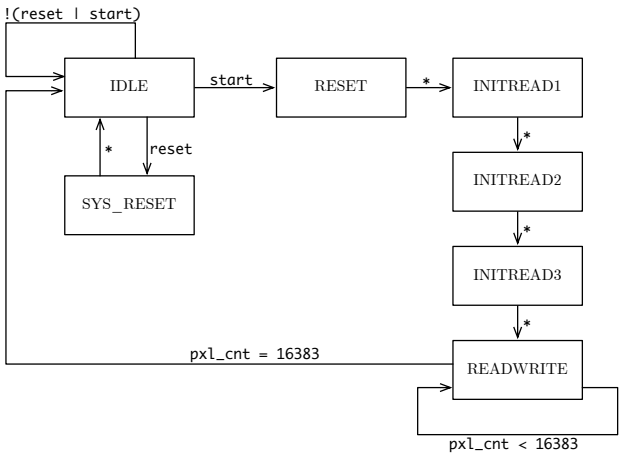

Figure 12: Display Processed Image State Transition Diagram

4.10 Gradient Module

Due to time constraints realized when our original scanning method failed to work, we did not implement this module in hardware. We did, however, code it in MATLAB using a coding style that we hoped would easily translate into hardware. In brief, we found a pseudo-gradient by finding a line that maximized the number of crossings.

4.11 Find Center Module

Due to time constraints realized when our original scanning method failed to work, we did not implement this module in hardware. We did, however, code it in MATLAB using a coding style that we hoped would easily translate into hardware. In brief, we described the center as an intersection between at least two of the gradients that fell inside the bounds of the image.

4.12 Row and Column Selector Module: Kevin

The Row and Column selector module is responsible for one action. On a start command it reads the captured fingerprint image out of the 2x128x128 binary processing buffer and writes a previously specified row and column of the image into two 1x128x1 line buffers. There is a buffer for the row, and one for the column. Since the image is 128x128 pixels, the first six bits of the memory address correspond to the column of the image, and the last six bits correspond to the row of the image. Data from the processing buffer are connected directly to both the row and column buffers. Data are read out of the processing buffer sequentially, and also written into the line buffers sequentially. Data are only 1 bit wide. Write enable signals to the column and row buffers are asserted only when the needed pixel is read out of memory. Using this scheme it is easy to determine which memory addresses correspond to which pixels, and also when to write the data.

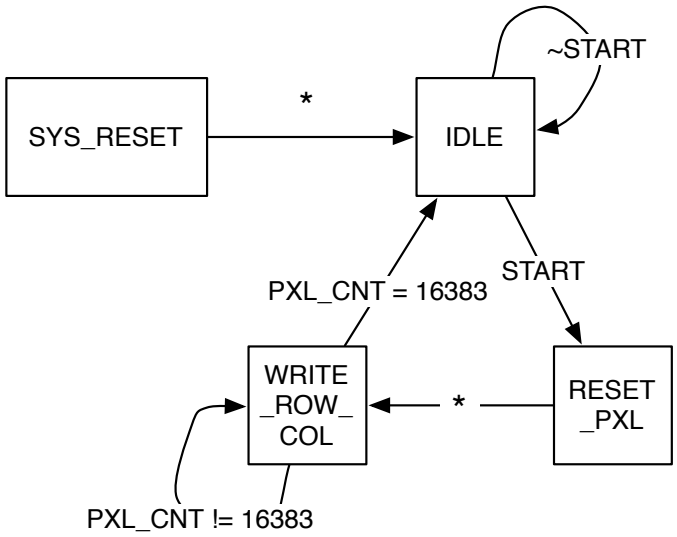

Figure 13: Finite State Diagram for the Row Column Selector Module

This functionality was implemented with a finite state machine. The finite state diagram is shown in figure 13.

4.13 Fourier Transform Module: Kevin

The Fourier Transform module is responsible for interfacing with the Fast Fourier Transform (FFT) Coregen unit and ensuring that the timing parameters are met. The Fourier Transform module completely wraps the FFT unit, so any interface involving the FFT goes through the Fourier Transform module. It also calculates the square of the magnitude of the coefficients.

There are two identical Fourier Transform modules. On a start signal from the processing controller, it sends another start signal to the FFT (FFT_start) and reads sequentially out of a line buffer, either the row or column buffer, and supplies those values to the FFT module. The data is delayed four clock cycles from the FFT_start signal as required by the FFT unit. Each pixel is streamed in every clock cycle, from pixel 0 to pixel 127. Data are read into the real input, and the imaginary input is tied to ground. The module will then wait while the FFT unit calculates the first 16 Fourier transform coefficients. These are sent, again in a streaming fashion from coefficient 0 to coefficient 15, through logic that calculates the square of the magnitude of the coefficients, to the Buffer selection module with the proper write enable signals. The FFT outputs the real and imaginary parts of the Fourier Transform and these are squared, then summed together to compute the square of the magnitude. It is this 32 bit wide value that is streamed into the Buffer Select module to be written to memory. Data emerging from the FFT is in twos complement, and must be converted to unsigned notation before being summed and squared.

This functionality is implemented using a finite state machine. The finite state diagram is shown in Figure 14.

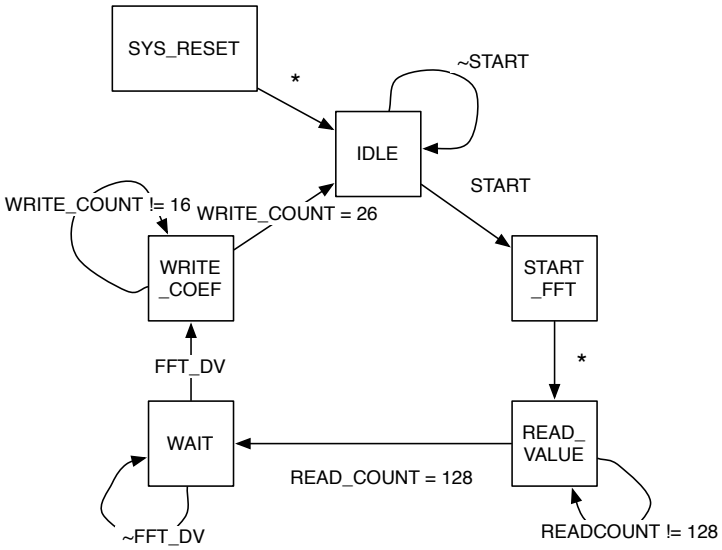

Figure 14: Finite State Diagram for the Fourier Transform Module

4.14 Buffer Selection: Kevin

The Buffer Selection module works in tandem with the Fourier Transform module to write Fourier coefficients into memory. There are two modes the system can be in, add user, or identify. If the system is in add user mode, data from the Fourier Transform module is written to the specified user location in memory buffer B. If the system is in identify mode, data is written to buffer A. There are two identical modules in the system, one for each Fourier Transform module. One corresponds to the row, and one to the column. The data from the Fourier Transform module is connected to both memories. When the Fourier Transform module begins a write (precipitated by the fft_dv signal), depending on its mode (identify or add user), the Buffer Select module will assert the write signal for the proper memory buffer and calculate the correct starting address. Our algorithm actually discards the constant coefficient. In coefficient0s place, zeros are written. As data is streamed in every clock cycle, it will increment the memory address accordingly. Memory Buffer B is 8x16x32, designed to hold up to 8 different images. The data coming in is 32 bits wide. The address lines to both memory A and B are used by both the Sum Squared Error module as well as the Buffer Selector module, so the module does not drive the address pins when not in use.

This functionality is implemented using a simple finite state machine. The finite state diagram is shown in Figure 15.

4.15 Summed Squared Error: Kevin

The Sum Squared Error Module is responsible for computing the square of the error between the magnitudes of different images Fourier coefficients. The module interfaces with Buffer A and B, and sends its data to the comparator module. It takes in an image number from the comparator to determine which image in memory B to begin addressing. On a start signal from the comparator module, the Sum Square Error module reads the squared magnitude of the first coefficient, one

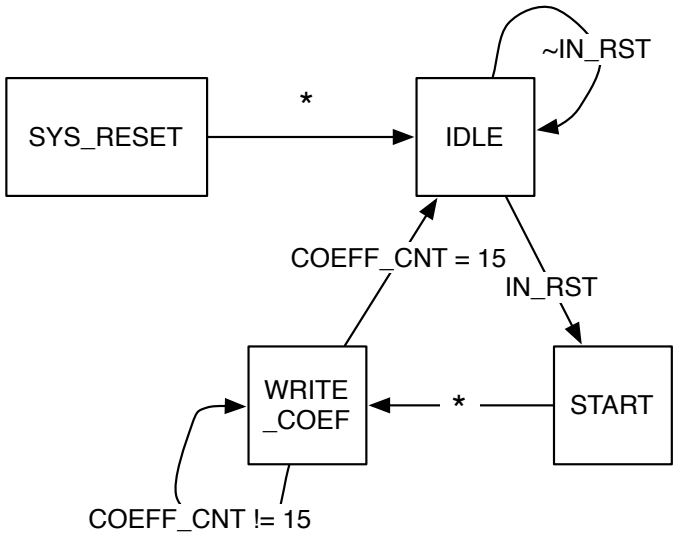

Figure 15: Finite State Diagram for the Buffer Selector Module

from memory A and one from memory B. It takes the absolute value of the difference of these two numbers. This is the error. The error is squared. The module reads in the next squared magnitude, calculates the squared error and adds to the first squared error. It continues this pattern until it has summed up all squared errors. This value is sent to the Comparator module. Data entering the module is 32 bits wide. Data leaving the module is 64 bits wide. A healthy bit width was used to ensure no overflow occurred, which would be detrimental to the system. The address lines to both memory A and B are used by both the Sum Squared Error module as well as the Buffer Selector module, so the module does not drive the address pins when not in use.

This functionality is implemented using a finite state machine. The finite state diagram is shown in Figure 16.

4.16 Validation Controller and Comparator: Kevin

The Comparator Module is responsible for controlling the two Sum Square Error modules as well as making the comparison to determine a match. On a start signal from the Controller module, the Comparator sends start signals to the Sum Square Error modules and waits for them to complete their computation. Once done signals have returned the squared error for both column and row are compared with a threshold value. If both are lower than the threshold value, it is a match, if one is above the threshold, no match. The module specifies an image number for the square error modules to calculate. This refers to the location of memory B to read from. The comparator will run through every image that has been stored in memory. It skips any memory blocks that have not been initialized with a fingerprint. The threshold value can be controlled by the user interface. Once a match is made, it will output that a match occurred as well as the image number matched until another request for comparison is made by the Controller.

This functionality is implemented using a finite state machine. The finite state diagram is shown in Figure 17.

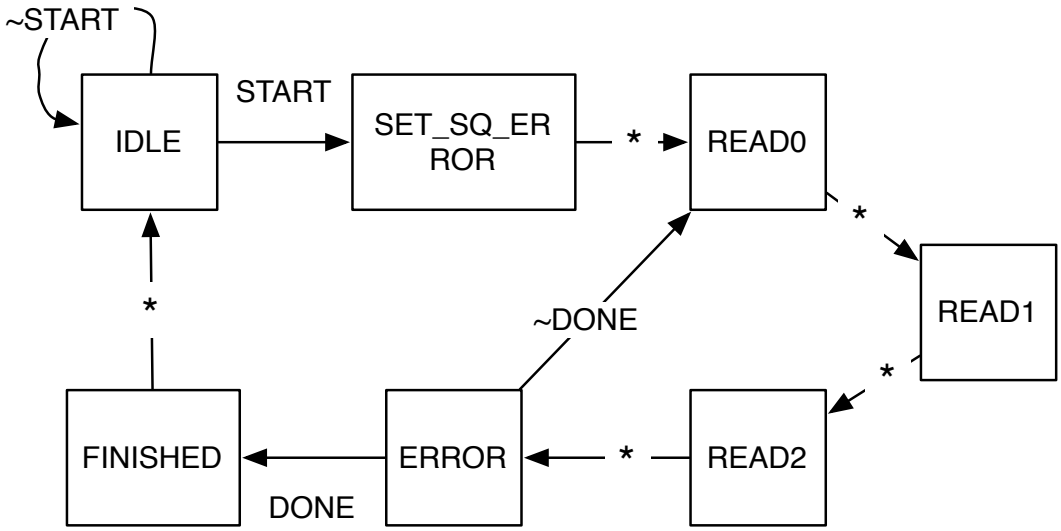

Figure 16: Finite State Diagram for the Sum Squared Error Module

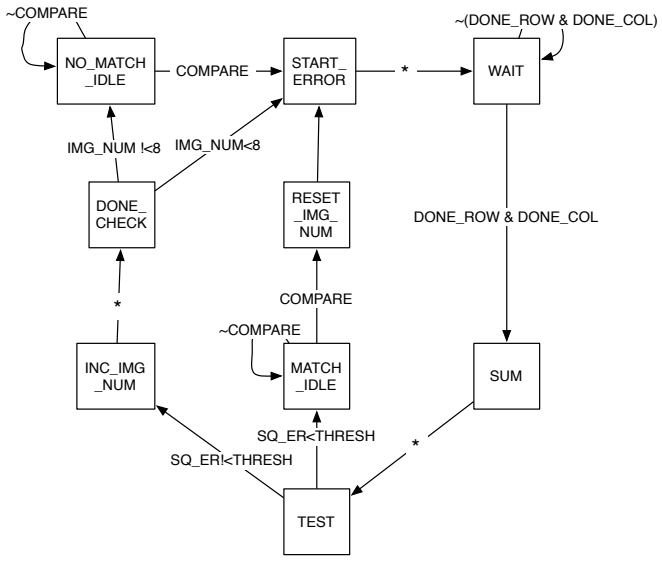

Figure 17: Finite State Diagram for the Comparator Module

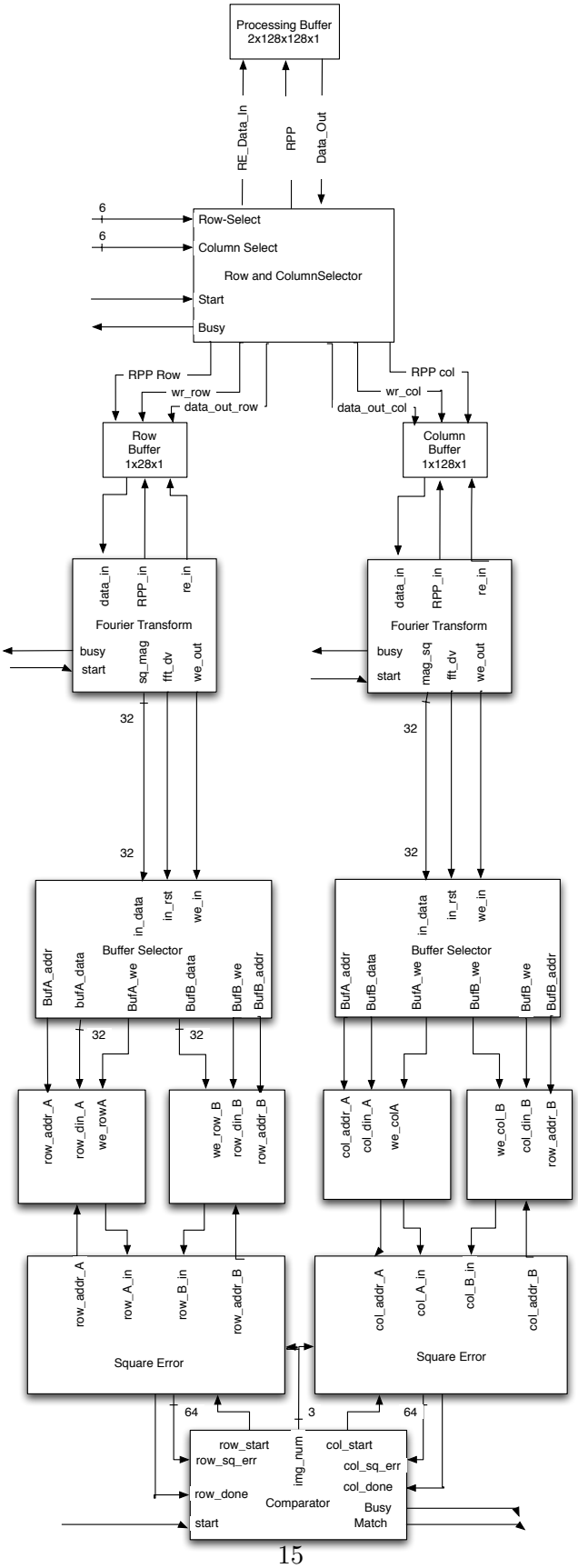

Figure 18: Detailed Block Diagram of the System

5 Conclusion

We have demonstrated a working fingerprint matching system. The current implementation is a good first pass of the algorithm. With more time, we could refine it to be more tolerant of variant scans.

The algorithm is unique in its comparison of frequency instead of feature extraction. This is attractive both for the speed gain, as well as ease of implementation in digital hardware.

We are particularly proud that we were able to quickly implement a video capture of the image in two days after our scanner did not work out.

References

[1] R. Guerrieri, "Unibo fingerprint capacitive sensor," University of Bologna, Tech. Rep., 1997, fingerprint database is available from `http://www-micro.deis.unibo.it/~tartagni//Finger/FingerSensor.html`.

[2] S. Patel, "Fingerprint verification system," 2002-2004, available online from Sourceforge: `http://fvs.sourceforge.net/`.

[3] M. Wennergren, "Spectral fingerprint matching," Master's thesis, Lund University, Centre for Mathematical Sciences, November 2004.