Fingerprint Verification System

Cheryl Texin
Bashira Chowdhury

6.111 Final Project
Spring 2006

Abstract

This report details the design and implementation of a fingerprint verification system. The system consists of two parts: an image acquisition module and an image verification module. The image acquisition module involved capturing an image of an inked fingerprint via a web camera and transmitting that image through a video decoder onto a block RAM in the 6.111 lab kit. Using the stored image, the image verification module involves verifying the image via image processing filters to find a match in a pre-formed print database. The image verification module was completed while the image acquisition module was not completed due to wiring and timing errors.

Table of Contents

Table of Figures

I. Introduction

Despite the multitude of personal identification methods currently in practice, fingerprint verification has traditionally been considered one of the most reliable methods available. Typically, fingerprint verification has been manually conducted by experts in print identification, which creates a significant backlog of work due to the tedious and time-consuming task of the manual identification. Consequently, autonomous fingerprint verification systems are in high demand for a wide range of applications that cannot use the manual verification process. These applications include access control, security verification, as well as the more commonly known criminal identification.

An autonomous fingerprint verification system takes advantage of many features of the ridge topology of the fingerprint. In particular, verification systems focus on the the minutiae of the fingerprint. The most commonly examined minutiae are endpoints and bifurications. Fingerprints are obtained by a sensor or a camera and preprocessed to obtain the minutiae of interest. The following matching stage uses the minutiae of interest to establish a correlation between the sample print and a print in a database.

For our final project in 6.111, we focused on implementing a robust fingerprint verification system with a small database. As a part of this design project, we researched, planned, and implemented our fingerprint verification system using the 6.111 lab kit and a web camera obtained through resources available in 6.111. This report details our design and implementation of our system.

II. Design Overview

This fingerprint verification system is composed of two modules: an image acquisition and image verification, both of which are detailed below.

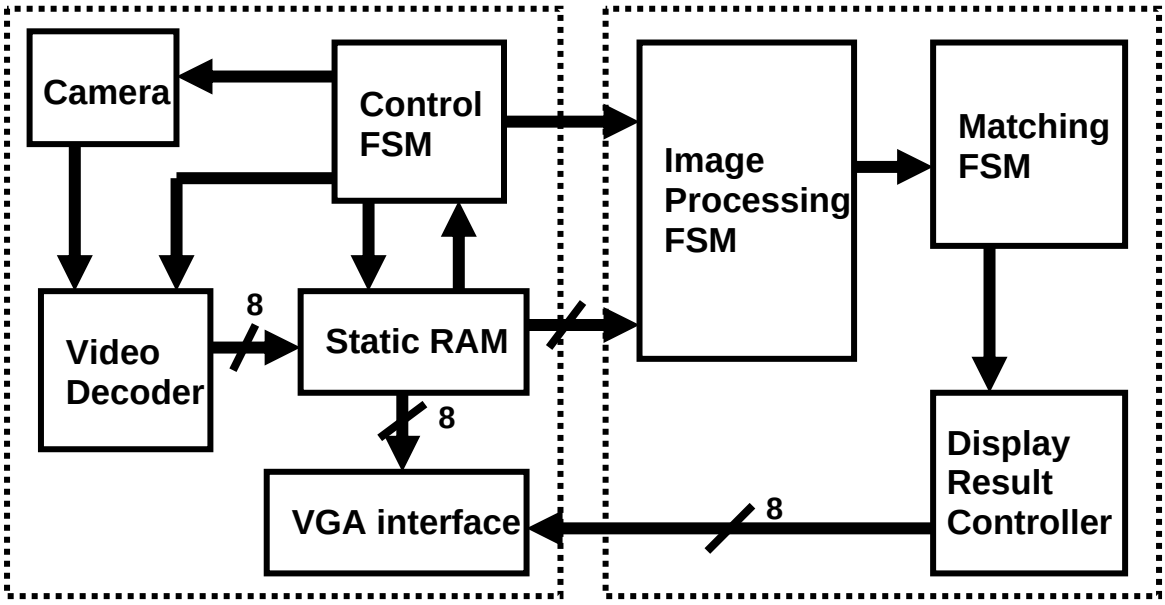


Figure 1. Block Diagram of Fingerprint Verification System

III. Image Acquisition Modules

To acquire an image of the fingerprint, a web camera provided by the 6.111 laboratory was used to capture an image of an inked print. Through the video decoder file provided by the 6.111 staff, the inked image was stored in the ZBT. However, only one frame of the print image was needed. In order to store just one frame of the image in the ZBT, a button on the lab kit was used to trigger a freeze mode on the video decoder. Via button zero on the lab kit, the signal triggered causes the ZBT to hold the last frame

captured from the camera. This mode is shown in the state transition diagram on the following page.

After the last frame of the print image is stored on the ZBT, the image is transferred to an asynchronous FIFO. This step was needed in order to synchronize the clocks of the image acquisition and image verification modules. The camera required the image acquisition module to run at a 65 Mhz clock while the image verification module ran at the 31.5 Mhz clock for its image processing procedures. The asynchronous FIFO allows the image to be stored at 65 Mhz into memory and then later read at the 31.5 Mhz clock.

Finally, the image was to be transferred to a block RAM in order to allow the image verification module to access the print sample. In addition, several block RAMs were to be created in order to create the small database needed for meaningful functioning. However, this step could not be achieved. In the following sections on debugging the image acquisition module, the difficulties in achieving this task will be detailed.

The following block diagram gives a quick visual overview of the image acquisition module.

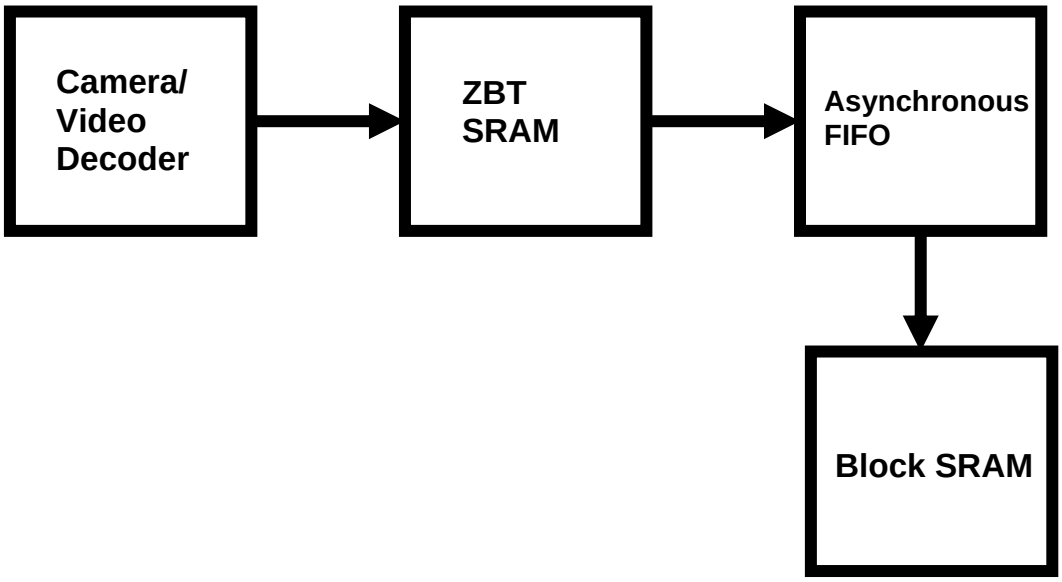


Figure 2. Block Diagram of Image Acquisition Module

A. Camera and Video Decoder

The camera captured images via the web camera and transmitted them to the ADV7185 on the 6.111 lab kit. Using the video decoder code provided by the 6.111 staff, the print image was stored into the ZBT RAM. However, the code needed to be modified in order to store only one frame of the print image. The code provided by the 6.111 staff essentially allowed for streaming video via the web camera, but the image acquisition

needed only one frame of this video. In order to accomplish this, button zero was used as a signal trigger to switch the write enable signal for the ZBT RAM. When button zero is pressed, the write enable signal for the ZBT RAM goes from low to high to stop the video decoder from writing to the ZBT and to allow the asynchronous FIFO to begin to read from the ZBT.

This freeze mode was introduced into the initial data path of the image acquisition process because it would allow the user to correct the print for smudges and correctly orient the print for proper verification.

The Verilog code for this part of the image acquisition process is found in the appendix to this report.

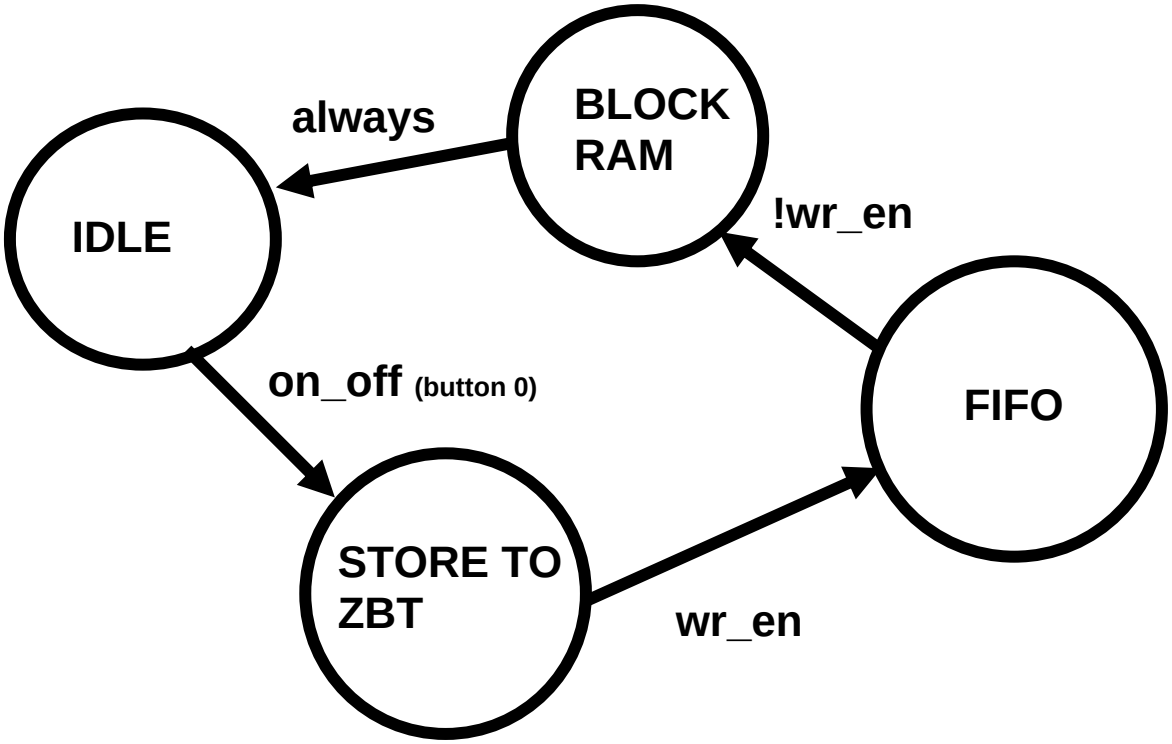


Figure 3. State transition diagram for control FSM

B. Asynchronous FIFO image transfer

A separate memory interface was made for the asynchronous FIFO created via the Coregen RAM generator in the Xlinx navigator. The following state transition diagram shows how the memory interface for the asynchronous FIFO operates. It essentially follows the memory interface FSM shown in Lecture 7. However, this is not the proper FSM for the asynchronous FSM because it did not correctly address the data from the ZBT. The correct FSM for the FIFO should take into account the need to wait two cycles for the data to be available from the ZBT, and it should store every fourth pixel on every

third line. The FSM and its accompanying Verilog code do not take these considerations into account. The code only stores every fourth pixel and does not allow two cycles before the data is available. Technically, the user would not know what pixel is actually being stored on the FIFO since it does not take into account these conditions.

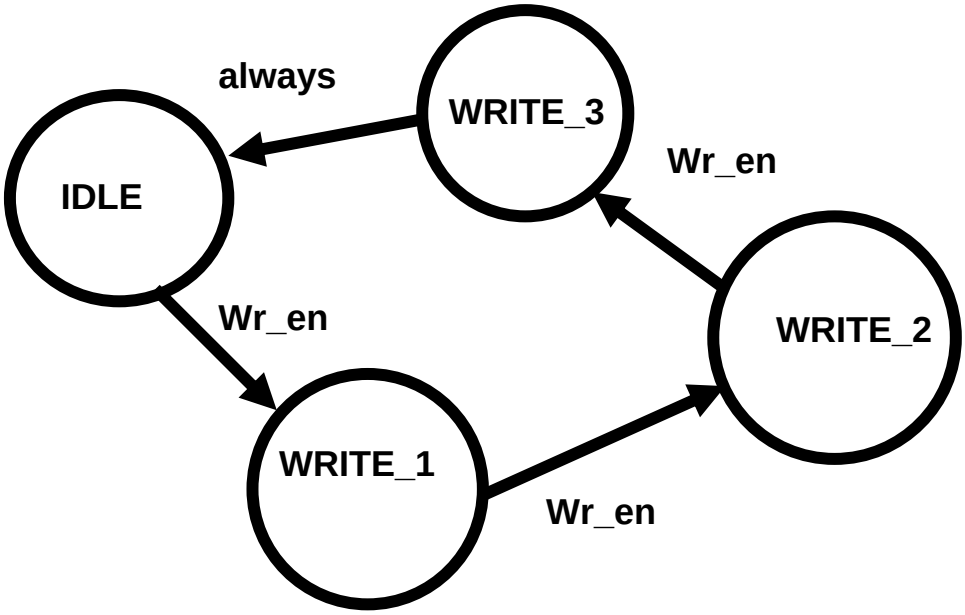


Figure 4. State transition diagram for FIFO interface

C. Block RAM

Via the Coregen RAM generator in the Xlinx navigator, the block RAM which would have stored the print image was created. In addition, the memory interface mimicked the interface created for the asynchronous FIFO was created. However, the block RAM never contained any data because the timing was not correct on the asynchronous FIFO to allow for proper image transfer.

D. Debugging the Image Acquisition Module

There were several problems with debugging the image acquisition module. Because the asynchronous FIFO FSM did not work properly, the image was never stored on the block RAM. Thus this module could not properly interface with the image

verification module. In addition, the initial storage onto the ZBT did not work properly for several days due to difficulties with the lab kit. After transfer to a different lab kit, the initial storage stage was debugged. Also the differences between the clocks did not enable facilitation of system integration.

III. Image Verification

The image processing portion of this project has three major stages. In the first stage an edge detection filtering operation is performed on the original binary image of a fingerprint, which was 256x256 pixels. The second stage maps direction vectors onto the edges. Finally, the matching stage sums the direction vectors to generate a metric for comparing fingerprints. Each of the three stages is designed as a minor FSM. All of these are controlled by one major FSM. The figure below shows the block diagram of the full system.

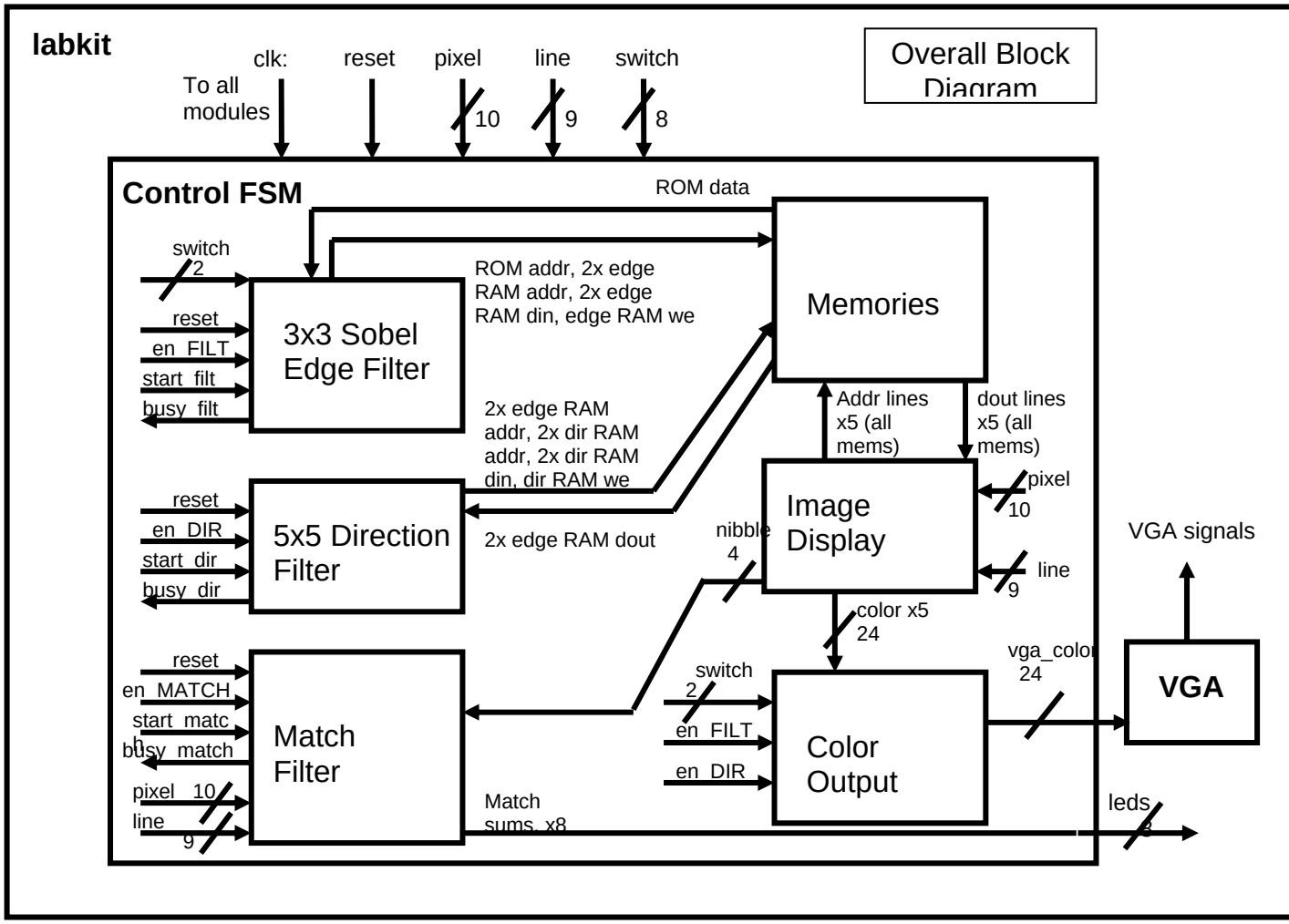


Figure 5. Block diagram for image verification module

User interaction with the labkit for the image processing is limited to the switch panel, the enter button, and the button numbered zero. The switch panel allows the user

to change views, adjust the threshold for the edge filtering (which will be explained in more detail later), and control the LED output. Pressing enter starts the processing and pressing zero resets the system. It is important to note that the reset function re-initializes the registers and the state machines, but does not clear the RAMs. Each component of the system will now be described, beginning with the filters.

A. Sobel Edge Detection Filter:

The Sobel Edge Detection Filter performs two convolutions on the original fingerprint, one in each the horizontal and vertical direction. The filters are of size 3x3, and are scaled as shown below. Note that the images below are drawn as they would be passed over the image, although the convolution algorithm is defined as using the filter mirrored over the origin. Assume the "flip" step of the convolution's "flip and slide" process has already occurred.

Vertical Filter:

1	0	-1
2	0	-2
1	0	-1

Horizontal Filter:

1	2	1
0	0	0
-1	-2	-1

The value of the pixels lying in the designated locations, with the pixel being processed lying in the center of the filter, is multiplied by the given scale factor and the results are summed. The original image was binary, so the range of values resulting from the filter is -4 to 4. The edge map is also binary, so there had to be some method of generating an "edge" or "no edge" signal. A simple threshold was chosen to do this. Any value above the threshold indicated that there was an edge, while a value equal to or below the threshold indicated that there was no edge. The threshold is input through the lowest two bits of the switch. The switch setting at the start of the filtering operation designates the threshold.

Usually, the results from the horizontal and vertical convolutions would be combined to form a single edge map. However, the image resulting from combining the convolved images was misaligned. This occurred because of the strictly positive threshold. Comparing the magnitude of the filtered result to the threshold would have given a clearer combined image. However, the high frequency of the edges in a fingerprint meant that there would be a very large number of edges in the final image if both the positive and negative values were kept. For simplicity, only the positive values were kept, and the filtered images were kept separate as horizontal and vertical edge maps.

The Sobel Edge Detection Filter was implemented as a FSM with 11 states. Each state performs a single action, although it would be possible to merge some of the states

and maintain the same functionality. Because this filter is designed as a minor FSM, it remains in the initial state until it receives an enable signal. The FSM then enters the filtering routine and outputs a busy signal until the filtering has been completed. The state transition diagram is drawn below, followed by an explanation of the states.

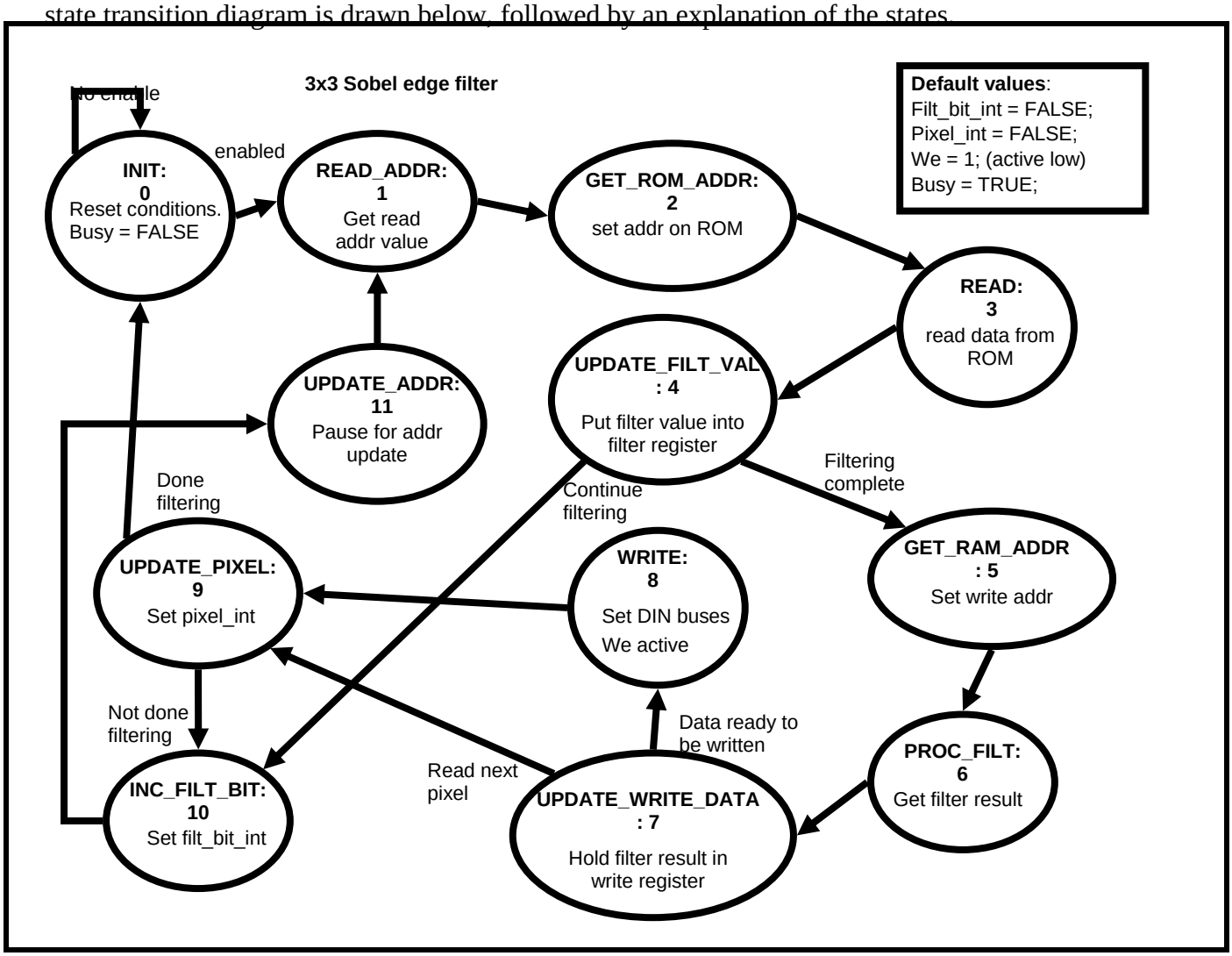


Figure 6. Sobel filter state transition diagram

In state 1, the read address is stored in a register. The next state sets this read address onto the ROM address bus. The third state reads the data presented by the ROM, and the FSM moves to state 4 where it updates the filter register. The filter register holds all of the values within the 3x3 area covered by the filter. At this point, the FSM checks if it has all of the filter values. If so, it goes to the write process. Otherwise, the FSM goes to state 10 where it sets the interrupt to indicate that the filter count should be incremented. In the next state, the FSM pauses for the read address to be updated before it returns to state 1.

The write process begins with state 5, in which the FSM sets the write address onto the edge RAM's address bus. The filter register is processed to determine both the horizontal and vertical filter results in state 6. In state 7, the FSM puts those results into

the write registers, which will later be written into the edge RAMs. If all 8 bits have been filled in, the FSM enters state 8 in which it presents the write registesr onto the data in buses and activates the write enable by pulling it low. Following the write, or if the write did not occur, is the UPDATE_PIXEL state. Here the interrupt to increment the current pixel is set. If the pixel count has reached the end, the FSM has finished filtering and returns to the initial state. If the filtering is incomplete, the FSM moves to state 10 and continues with the filtering process.

This FSM instantiates two submodules to simplify the filtering operation. The "byte_bit" submodule converts a pixel number into an address for the memory, and the bit location of that pixel number within that address location. This function is useful for reading and writing individual bits, since the memories accessed with this FSM are 8 bits wide. The 'byte_bit" submodule was used both for the data being read off of the ROM with the original fingerprint, as well as for the data being written to the RAMs containing the edge maps.

The second submodule, "read_loc", is responsible for updating the next pixel to be read. The filtering occurs over an image, which is laid out in 2D. The memories are addressed one-dimensionally, however. For an image of dimensions MxN, the relative pixel location can be converted into an address for the memory, keeping in mind that the image is indexed from the top left corner. To move up or down one row, the value of N is subtracted or added, respectively. Left or right transitions are simply -1 or +1, respectively. The "read_loc" module outputs the appropriate pixel location relative to the current pixel being written based on the location of the filter. The filter is indexed as shown below. The current pixel is always scaled by zero, so its value is irrelevant and is not read.

0	1	2
3	N/A	4
5	6	7

B. Direction Filter

The Direction Filter is another minor FSM, and it is very similar in operation to the Sobel Edge Detection Filter. There are 4 filters for this FSM, and the filtering is performed on both the horizontal and vertical edge maps. These are 5 point filters, and each filter isolates a direction vector, either horizontal, vertical, or +/- 45 degrees. Instead of summing the scaled values of each pixel, these filters are checking that all of the values under the pixel window are high. The submodule "dir_filt" performs this check and assigns the appropriate output. The filters are shown below, with the center pixel of each filter being the current pixel.

Horizontal: | * | * | * | * | * |

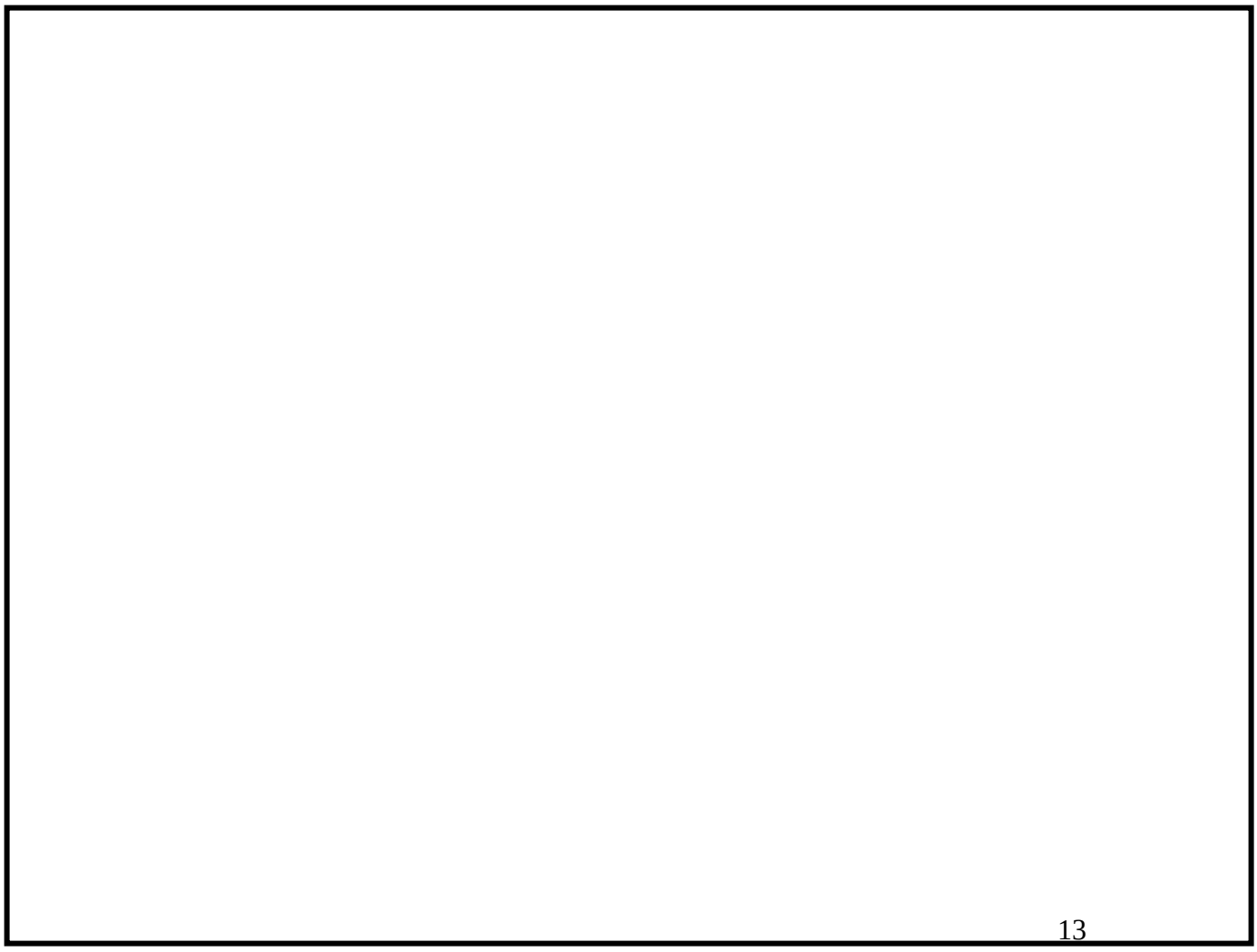
Vertical:
*
*
*

```
                    | * |
                    | * |

+ 45 degrees:                 | * |
                            | * |
                          | * |
                        | * |
                    | * |

-45 degrees:        | * |
                        | * |
                          | * |
                            | * |
                              | * |
```

The FSM for the Direction Filter is set up almost exactly the same as the FSM for the Sobel Edge Detection Filter. However, in the Direction Filter, the data is being read off of the edge RAMs instead of the ROM. Each edge map results in a separate direction map. The state transition diagram can be seen below, but the description of the states is not repeated (see the description of the Sobel Edge Detection Filter states).



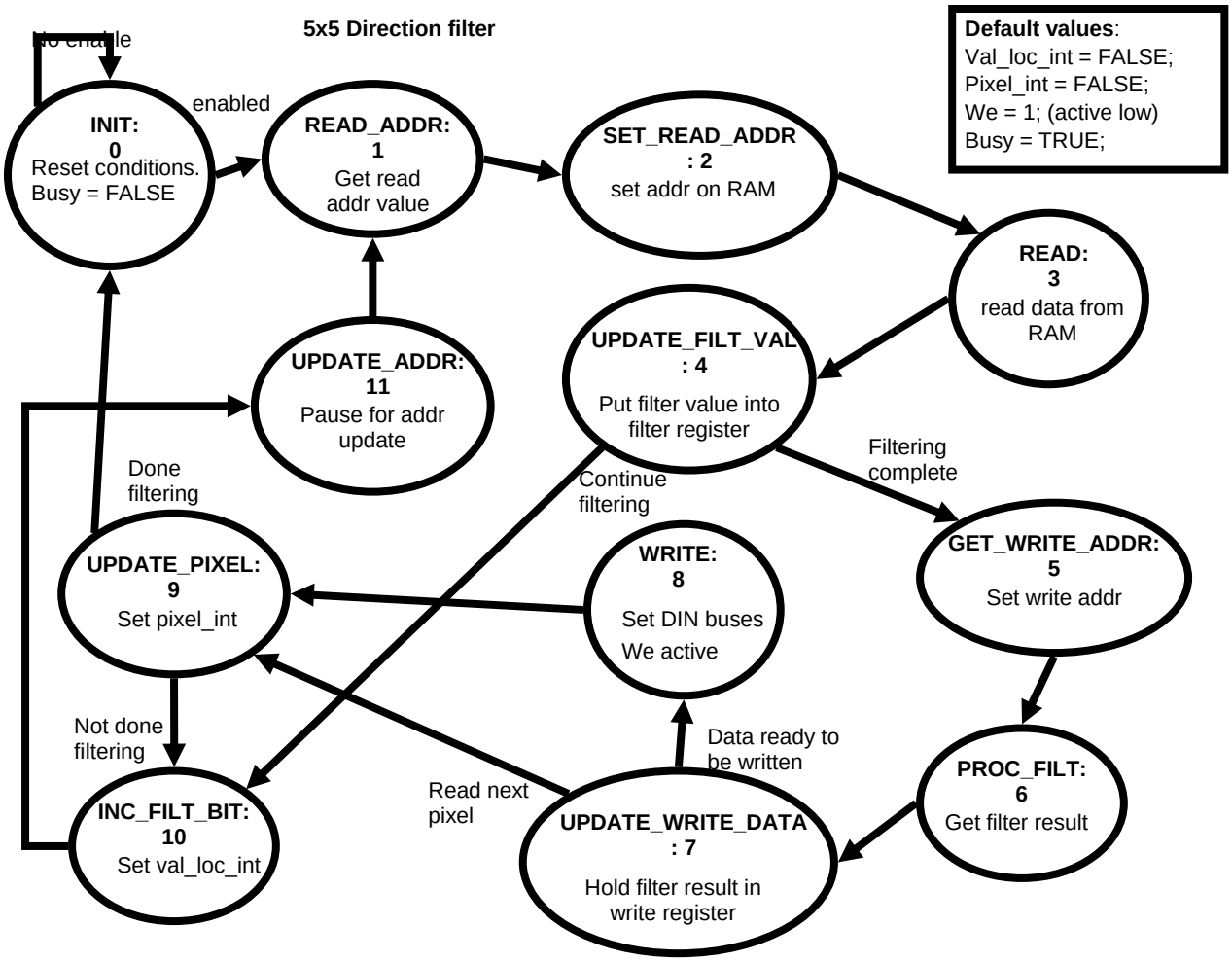


Figure 7. Direction Filter FSM

There are a few differences between the Direction Filter and the Sobel Edge Detection Filter. First of all, the pixels being read now occupy a 5x5 grid rather than a 3x3 grid, although not all of those pixels are used. The submodule “read_loc5x5” is an updated version of “read_loc” that accounts for this. The new index for this submodule is given below, with pixel number 12 as the one currently being filtered.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Because there are multiple directions that can result from this filtering, a nibble (4 bits) was used to code the result on the direction RAMs. This required the use of the “byte_nibble” submodule to address the direction RAMs. This new submodule is similar to “byte_bit” but accounts for the fact that only two pixels can be encoded at each

memory address rather than eight. The “byte_bit” submodule is still used to address the edge RAMs for the read operations.

C. Match FSM

The Match FSM is used to provide a measure for a future comparison of fingerprints. This is also a minor FSM, with both the busy and enable signals. The image is divided into four quadrants, as show below.

0	1
2	3

In each quadrant, the number of +/- 45 degree direction vectors are summed, resulting in 8 values on which to base the comparison. This FSM needs to read each pixel on the direction RAM, and it takes advantage of the fact that the display code already reads out each pixel. The Match FSM uses the current line and pixel counts to determine which quadrant it is in and then increments the appropriate counter based on the direction vector read at that location. The state transitions are shown below.

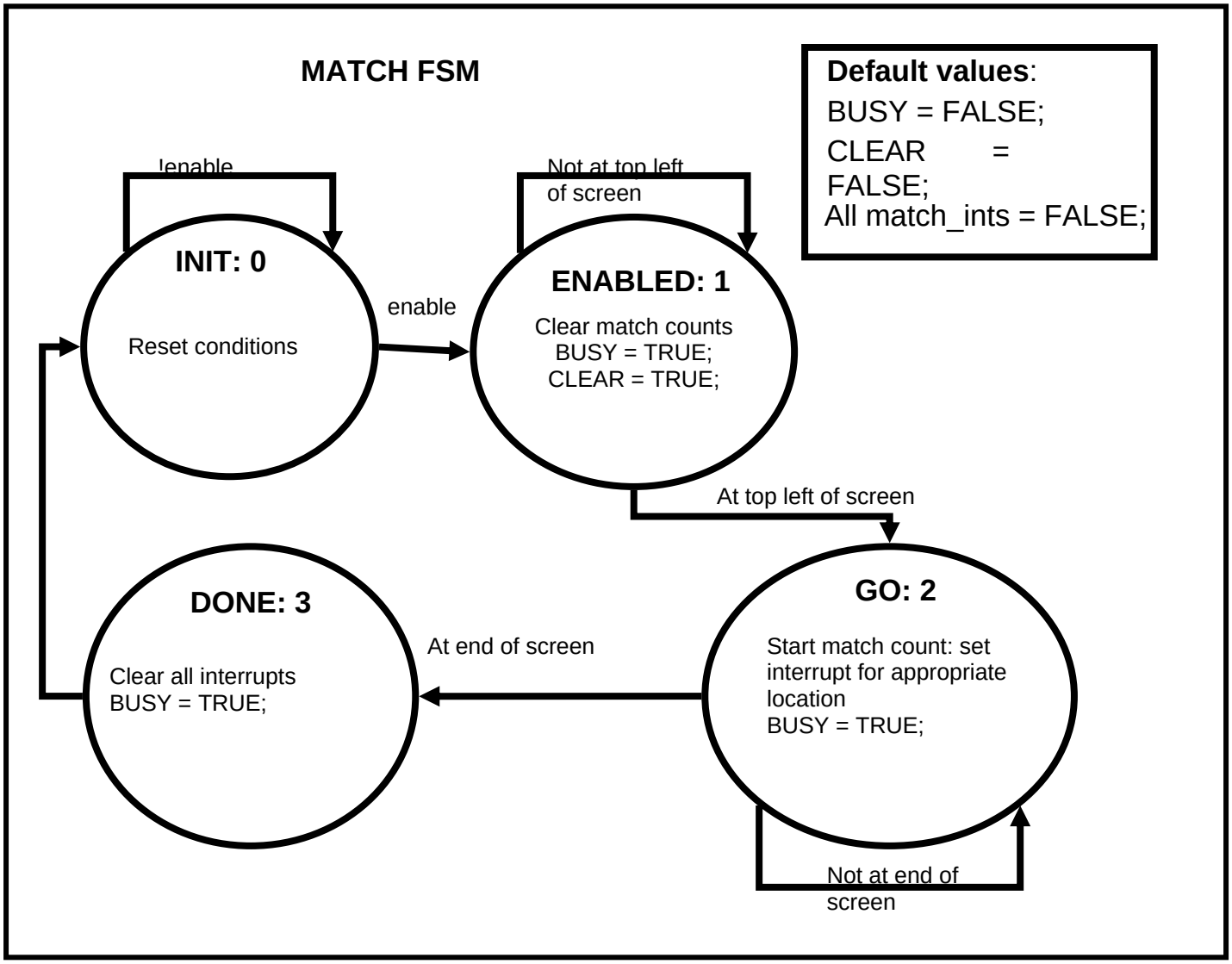


Figure 8. Major FSM state transition diagram

The majority of the function of the Match FSM resides in the "GO" state. The "ENABLED" state is necessary to generate an accurate count. Upon receiving the enable signal, the FSM enters this state where it clears the current match-sum registers, and waits for the pixel and line count to return to the top left corner of the screen before proceeding.

The "GO" state is held for one entire pass through the image. A case statement operating on the quadrant location sets the appropriate interrupt for the current quadrant. In sequential logic, each match sum register holds its value unless that interrupt is high. If the corresponding interrupt is set, the nibble that was read off of the direction RAM is checked. If it equals the correct direction vector, the register is incremented. Otherwise, the register continues to hold its value.

Each quadrant is 128x128 pixels, which would require 15 bits to store. However, it is impossible for an entire quadrant to be made up of one direction vector, due to the edge detection pre-processing that occurred. The LEDs are used to display the low 8 bits of the final match values. As these can store at maximum a value of 255, they cannot be used as an exact count, but can be used to observe that the match sums are approximately correct. The switch is used to view the separate match-sum outputs. The table below lists the corresponding match-sum output for each switch position.

Table 1. Match-sum switches

Switch[7:4]	Match-sum output (LED off for a zero input)
0	Quadrant 0, +45 degrees
1	Quadrant 0, -45 degrees
2	Quadrant 1, +45 degrees
3	Quadrant 1, -45 degrees
4	Quadrant 2, +45 degrees
5	Quadrant 2, -45 degrees
6	Quadrant 3, +45 degrees
7	Quadrant 3, -45 degrees
8-15;	Defaults to zero, all LED's are off

D. Displaying Images

There are two image display modules, imgdisp and imgdisp_dir, because of the different addressing that is needed for the binary images and for the images using nibbles. Both essentially do the same thing, however. They cycle through the ROM or RAM addresses sequentially, reading out each pixel value and converting it into a 24-bit color. For the binary memories, a value of "1" displays as white, while a value of "0" displays as black. For the direction map RAMs, horizontal or vertical directions are displayed as blue. An angle of 45 degrees in the positive direction is displayed as green, while 45 degrees in the negative direction is displayed as red, and no direction is displayed as white. For both image display modules, any pixel not within the display area in the center of the screen is displayed as MIT red.

The final step in setting the actual VGA color is choosing which ROM or RAM color output to select. The "colorOUT" module selects a color based on the switch input. The table below shows the switch-memory mapping.

Table 2. Memory mapping switches

Switch[3:0]	Memory Displayed
0	ROM, original fingerprint image
1	RAM, vertical edge filtered
2	RAM, horizontal edge filtered
3	RAM, vertical direction filtered
4	RAM, horizontal direction filtered
5-15	Green screen, invalid switch input

The “colorOUT” module also checks the enables on the edge and direction FSMs. If either of those enables is on, the screen is displayed as blue to indicate that the filtering operation is occurring. This is also done because the memories are being addressed by the filter FSMs, so the display would be incorrect.

E. Control FSM (Major FSM)

The control FSM manages all of the signals used by the submodules, as well as interfacing with the labkit. This major FSM is responsible for handling the operation of the three minor FSMs which implement the three stages of the image processing used for this project. The control FSM also instantiates the submodules that handle the memories, image display, and color output to the VGA. Finally, the control FSM uses the four most significant bits of the switch to set the LED output to the appropriate match sum.

The most important function of the control FSM is managing the data and address buses for the memories. Each memory is being accessed by multiple modules, those for displaying the images and those for processing the images. Since each module is adjusting the address for its own purpose, the control FSM is responsible for selecting the address to actually be put on the memory bus to avoid bus contention. The enable signals sent to the filtering modules are used to do this.

The control FSM has 8 states. There is a state for reset, in which initial conditions are set up. The FSM immediately moves from reset to the display state. The display state is the default state for this FSM. All of the enables are off, and the image display modules have control of the address buses. Once the user has pressed the enter button to start the filtering operations, the FSM enables the appropriate filter and moves to a wait state until the busy signal response from the minor FSM has gone low. The FSM performs the Sobel Edge Detection Filtering first, then the Direction Filtering, and finally performs the Match–sum calculations. All three minor FSMs occur in that order for every filtering operation, and the control FSM returns to the display state once the Match FSM has signaled that it finished. The state transition diagram is below.

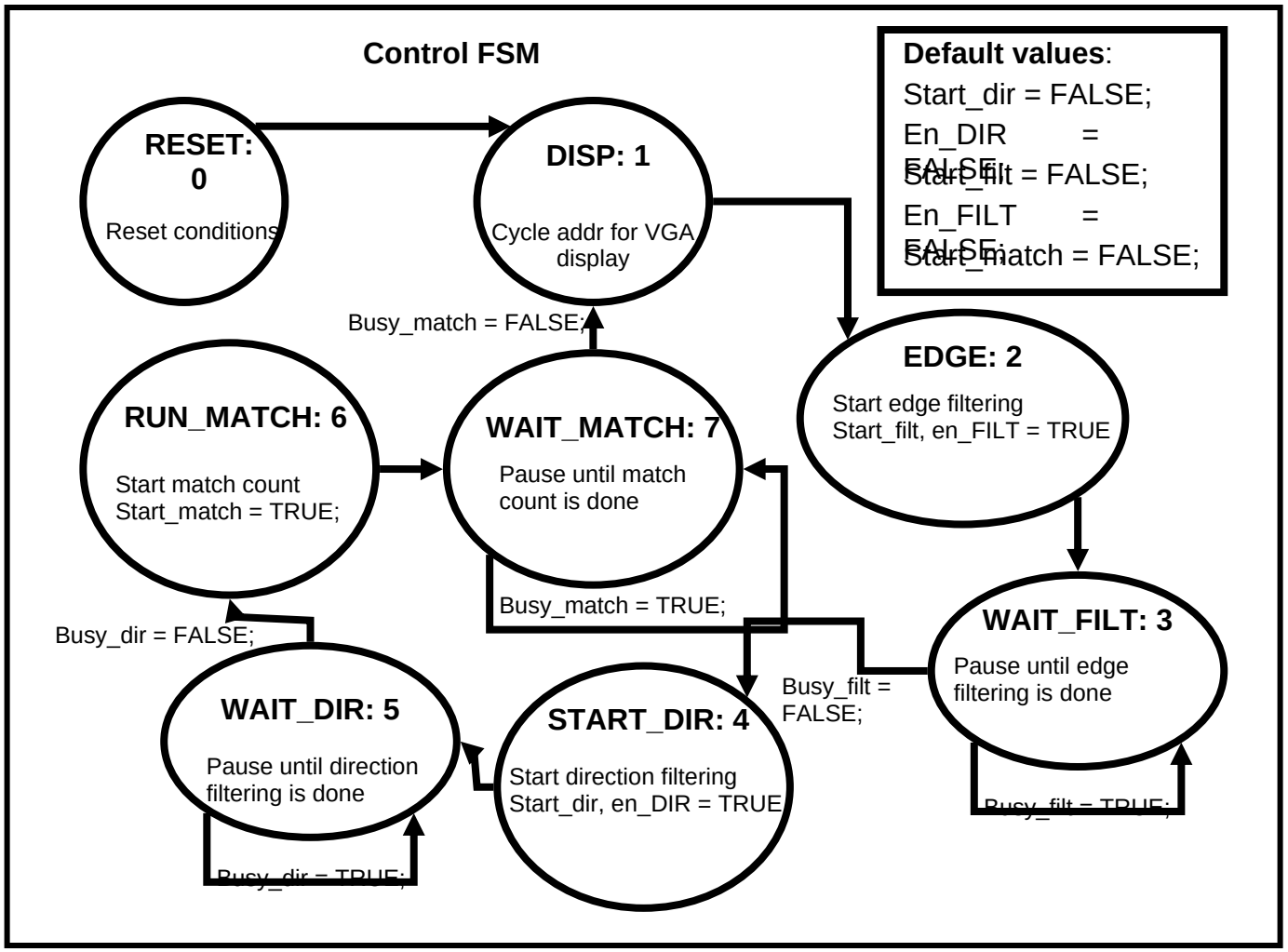


Figure 9. Match FSM state transition diagram

F. Matching

The final step to this project would be to perform the match using the match-sums generated by the Match FSM. A database of images would be created, by scanning them in and storing them in RAM. A test image would then be scanned in and filtered to generate the match-sum values. Each of the database images would also be filtered using the same threshold and they would each have a corresponding set of match-sum values. The match-sum values of the test image would then be compared to the match-sum values of the database images. If the test values were within a given threshold of one set of the database values, a match would be declared. If there are multiple matches in the database, the database image with the set of match-sum values closest to the test image would be chosen.

IV. Conclusion

This report detailed the image acquisition and image verification processes for a fingerprint verification system. While the system was not completely implemented, the lessons learned from this project showed the importance of good design planning and the need for early implementation in order to fully achieve a working digital design.