Writing Recognition

Stephanie Hsu

6.111 Fall 2006

Final Project

Abstract:

Writing recognition programs are an increasing component of electronic tools such as personal data assistants and document processing suites. However, the challenges of recognizing a hand-drawn image stems from the myriad of ways each character can be depicted. The project aims to implement a form of writing recognition by constraining the user to a specific set of predetermined instructions and searching for the correct character based on the directional movement of the sketch. In this project, the user draws a character by moving a mouse

Contents

I. Overview

The project aims to recognize a user-drawn letter or number. The user sketches the character by clicking and moving the mouse within the writing pad displayed on the monitor. The strategy to recognize the written character is based on techniques used by writing recognition software installed on personal data assistants and relies on a predetermined set of stroke rules for each character.

1.1 User Interface

The user interfaces with the project through the mouse and the display monitor. A mouse cursor moved by the user is displayed on the monitor. The outlines of a writing pad and the recognized character are also shown on the display. Outlines of the areas on the display are shown in Figure 1.

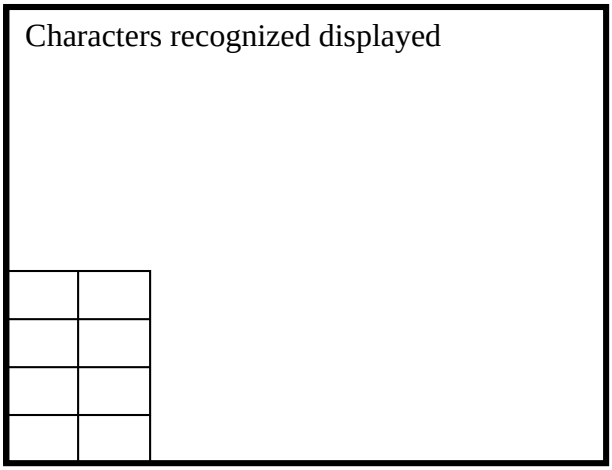

Figure 1: Outline of Display

The characters the project can recognize are shown in Figure 2. The user must begin sketching the character on the dot and follow the stroke order displayed in Figure 2. Lower and upper-case letters are not distinguishable by the order of the mouse stroke. An upper case letter will be displayed if the right mouse-button is clicked while a lower-case letter will be displayed if the left mouse-button is clicked while the user is writing a character. In addition, the user must write the character within the writing pad outline on the monitor.

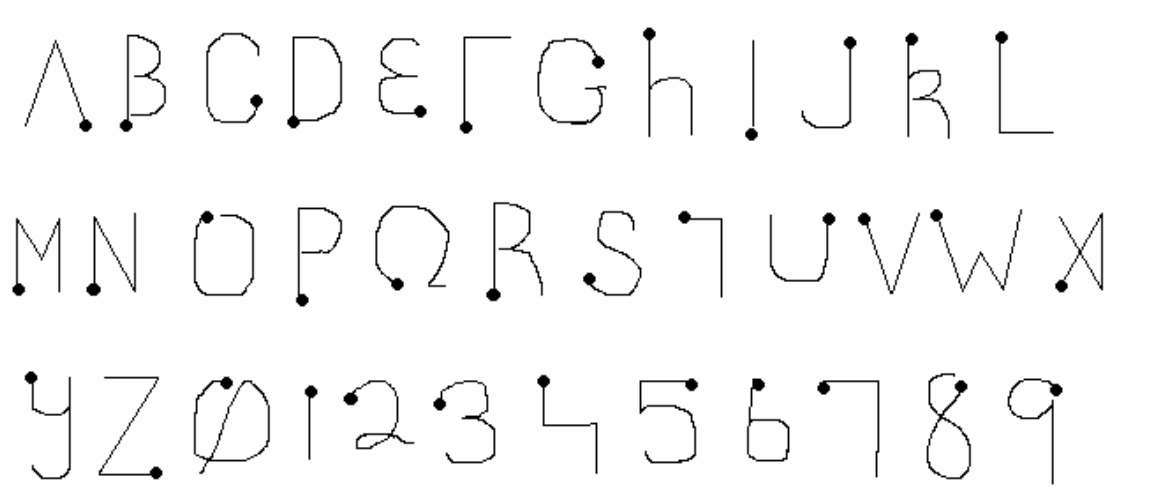

Figure 2: Recognizable Letters and Numbers: The dot indicates the starting point of the mouse

1.2 Character Recognition Algorithm

The algorithm for recognizing characters relies on a specific set of instructions the user must follow to write a character. Each character begins and ends in a pre-determined area of the writing pad, which is divided into eight cells as shown in Figure 3. Based on the beginning and ending cell of a writing sequence, the algorithm assigns one of seventeen possible FSMs to attempt to identify the character. The assigned FSM analyzes the sequence of cells that the mouse travels through to match the sketched character with the correct ASCII code. If no character match can be found, the display will output a question mark.

0000	0001
0010	0011
0100	0101
0110	0111

Figure 3: Writing Pad

This method serves to quickly divide possible matches into sections based on the broad criteria of the first and last cell entered by the mouse. Each FSM then sequentially analyzes the sequence of cells the mouse traveled through to identify the character written. A sample FSM that analyzes sequences beginning in cells 0 or 2 and ending in cells 1 or 3 is shown in Figure 5.

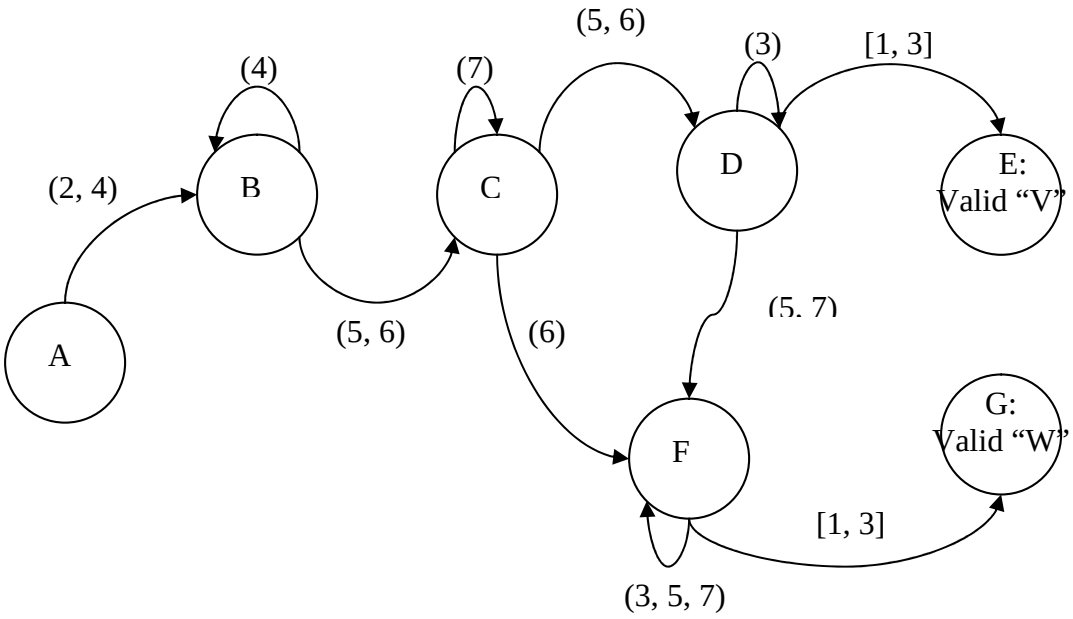

Figure 5: FSM 13, which decides between "V" and "W." Numbers in ( ) are the input cells from memory that cause the FSM to change state. Numbers in [ ] indicate that the input cell is the last of the sequence.

1.3 Implementation

The project consists of 3 large modules, each of which is broken down into several smaller submodules. The connections between the modules are shown in Figure 6. The mouse module decodes data input from a PS/2 mouse interfaced with the labkit and translates the information into x and y position coordinates on the display screen. In addition, the module also outputs information about the buttons clicked on the mouse. The character recognition module stores information about the movement of the mouse when the user is writing by storing the sequence of cells that the mouse passes through into BRAM. The BRAM is accessed by the character recognition to implement the algorithm described in section 1.2. Once the character has been identified, the ASCII code is sent to the display module and output onto the monitor.

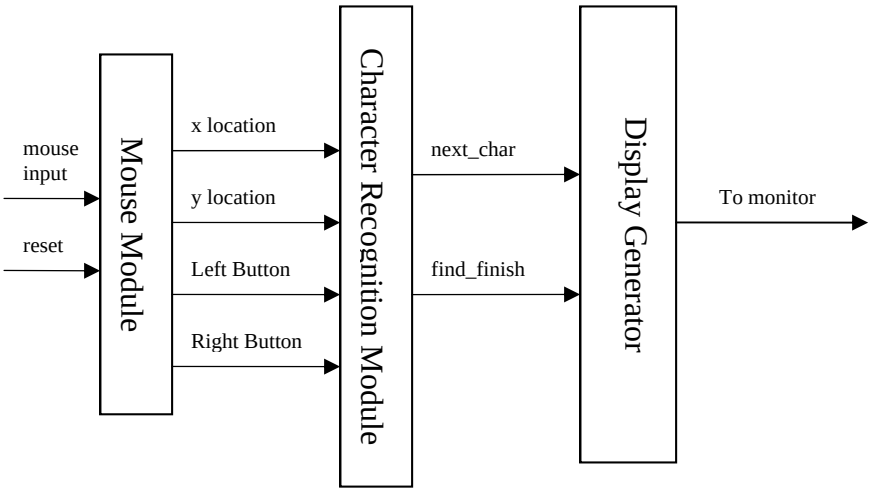

Figure 6: Block Diagram Major Modules

II. Modules

2.1 Mouse Module

The mouse module receives data from a PS/2 mouse interfaced with the labkit, which transmits changes in the x and y position of the mouse as well as information about the mouse. The mouse module (modified example from the 6.111 code bank) uses the top right hand corner of the screen as a reset point and determines the actual location of the mouse on the screen. The x and y coordinates are output to the character recognition module. Figure 7 outlines the input and output signals of the mouse module.

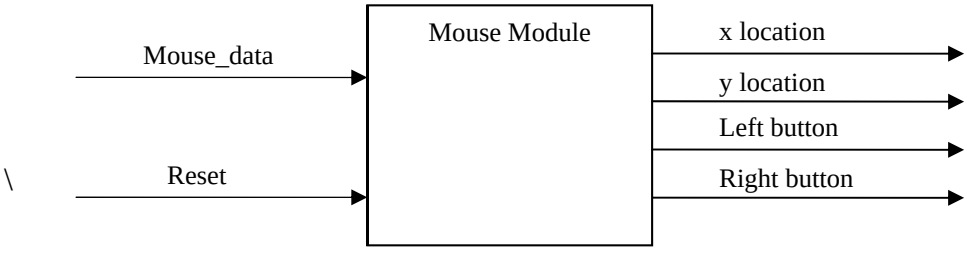

Figure 7: Mouse Module

The mouse module runs continuously and goes to the upper left-hand corner of the screen once the reset button is pushed. The module was taken from the Fall 2005 6.111 code bank [1].

2.2 Character Recognition Module

The bulk of the project, the character recognition module analyzes the position of the mouse on the screen, identifies the character the user is sketching, and outputs the ASCII code for the recognized character to the display module. Figure 8 illustrates the components of the module.

BRAM:

The 16x8 bit memory block stores the sequence of cells the mouse passes through while writing a character.

mouse_cell controller:

The mouse_cell controller uses the x and y coordinates extracted from the mouse input data to identify the area of the display the mouse cursor occupies. If the mouse cursor is not within the writing pad, mouse_cell is set to 1000. Otherwise, mouse_cell equals the number of the cell (as shown in Figure 3) it occupies. Mouse_cell is output to the mouse_finish controller and the BRAM. Parameters within the module determine the size of the writing pad.

mouse_finish controller:

The mouse_finish signal is only low if the user is clicking within the writing pad. The controller uses the mouse_cell signal to determine whether or not the user has finished writing by only remaining low if mouse_cell equals a valid cell number within the writing pad. In addition, mouse_finish will be high if the number of cells traced by the mouse within a sequence exceeds the size of the BRAM. The signal is output to the write_address controller, the read_address controller, and the BRAM.

A low mouse_finish allows the write_address controller to store information into memory because the inverse of mouse_finish is input into the BRAM as the write enable signal. Since the user has not finished writing when mouse_finish is low, the cells in the writing pad that the mouse travels through are stored sequentially into BRAM. Once the user has stopped writing, either by releasing the mouse button, leaving the writing pad, or traveling through more than 16 cells, mouse_finish becomes low and the module begins the process of identifying the character. A high mouse_finish allows the read_address controller to begin accepting memory addresses from the FSM submodule. In addition, the signal controls the address input into the BRAM.

write_address controller:

If mouse_finish is high, the write_address controller increments the BRAM write address each time mouse_cell changes. In addition, the controller keeps track of the highest address used for any sequence and outputs the signal to the BRAM read_address

controller. Highest_address serves to mark the end of a stored sequence as the BRAM is not cleared between successive sequences to be stored.

read_address controller:

The read_address controller manages the data read from the BRAM. Once the user has finished writing and mouse_finish becomes high, the read_address controller identifies the first and last cell of the stored sequence from the BRAM by using the highest_address signal. The start_id signal then becomes high for one clock cycle to trigger the identify FSM submodule. During operation of the FSM, the read_address controller accepts addresses from the FSM and returns the data stored in the memory to the FSM.

Identify FSM:

The Identify FSM module uses the begin_cell and end_cell data from the read_address controller to decide which of seventeen FSMs to use. The module outputs FSM_num and a signal, idFSM_end, indicating that the correct FSM has been found to the FSM module. The FSM module will only run if idFSM_end remains high.

FSMs:

The FSMs module contains the main algorithm to determine which character was written. Using FSM_num, the module identifies the FSM to be used and outputs fsm_address to the read_address controller to indicate which entry of the memory block is needed. Once the chosen FSM has reached a conclusion, the ASCII code for the character is output on next_char to the display module. In addition, the runFSM_end signal becomes high. The second output of the character recognition module, find_finish, is determined by the AND of runFSM_end and idFSM_end. Once both signals are high, the module has finished analyzing the sequence of cells currently stored within the BRAM, and the user can begin writing another character.

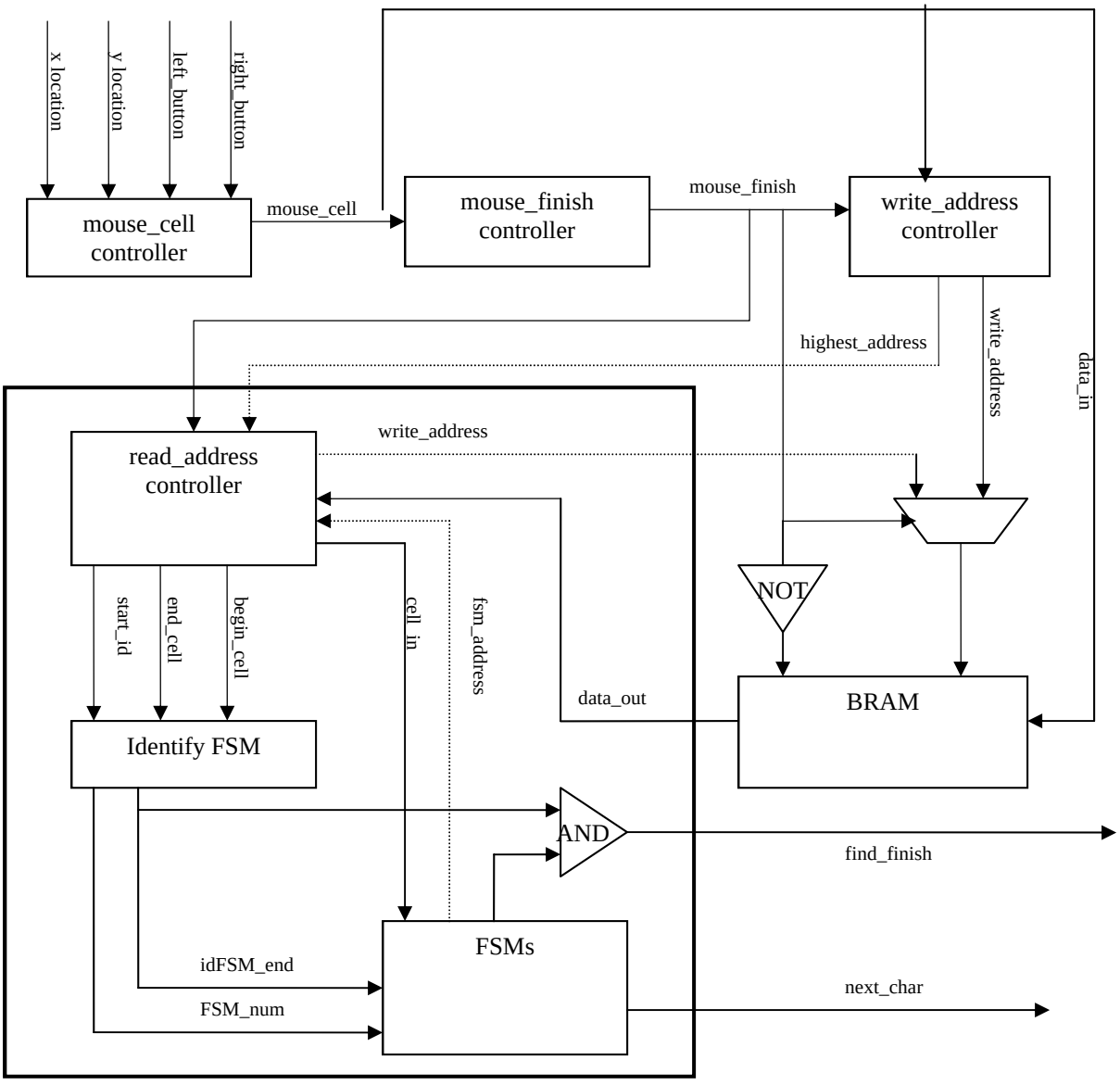

Figure 8: Character Recognition Module

2.3 Display Module

The display module uses the ASCII codes sent from the character recognition module to display the character on the computer monitor. Beginning in the upper-left corner of the screen, the characters recognized are displayed in a string. Figure 9 outlines the components within the display module.

char_string controller:

The char_string controller module keeps track of the section of char_string that should be changed by the next character recognized. In addition, the user can also delete characters displayed by sketching a backspace.

char_string_disp:

The char_string_disp module decodes the string of ASCII codes from the char_string controller and sends the correct address to the font_rom. The returned font_byte contains the pictorial encoding for the desired character. Cx and cy, the coordinates of the upper left-hand corner of the displayed string, are set to zero. Char_string_disp generates the image through combinational logic and returns a blank pixel if hcount and vcount are not within the parameters of the displayed string. The char_string_disp module was taken from the Fall 2005 6.111 sample code bank [1].

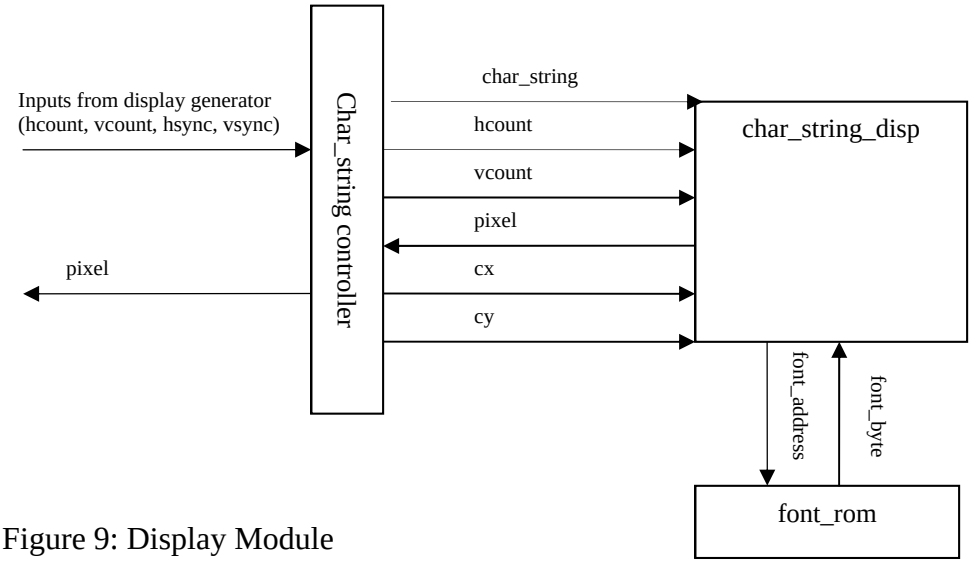

Figure 9: Display Module

III. Testing and Debugging

The most serious debugging issues predictably occurred while testing the components of the character recognition module and while synchronizing the control signals between the modules. Each component of the module relies on a control signal that exhibits a specific behavior to trigger the submodule to begin but that could not always be accurately simulated in ModelSim.

A particular example of this issue occurred with the mouse_finish and find_finish signals. These signals have a circular relationship in that one triggers the other to go high or low after some function has been completed. In addition, mouse_finish and find_finish also act as control signals for other components, such as the BRAM write enable signal and the read_address controller. Test modules within ModelSim confirmed that each individual block was fully functional when mouse_finish and find_finish were input manually into the test cases. However, it was difficult to simulate more than one writing sequence as the mouse- and find- finished signals were determined by different components.

The fsm_address also needed to be controlled by more than the idFSM_end signal, which is only low for one clock cycle. The initial design allowed the fsm_address to increment if idFSM_end was high, which caused the FSM to cycle rapidly when the read_address controller was not ready and settle into the null state. Although fsm_address resets to zero when the Identify FSM module sets idFSM_end to high, the FSM state remained in the null state because no character had been identified. This problem was solved by also ensuring that the Identify FSM module had reached the state in it's module that accepted fsm_addresses from the FSM before allowing fsm_address to increase and the FSM to change state.

A second issue arose when reading data from the BRAM. There is a three clock cycle delay between the output of an address from the FSM module to the input of the correct data from the BRAM because the address and the return data first pass through the read_address controller. Therefore, the FSMs were delayed to accommodate the time lag. In ModelSim, this issue was not caught as the simulator was unable to read from memory and the data inputs were manually created but was a major source of error when all modules were connected.

Debugging the FSMs proved to be tedious but relatively simple. The path of the mouse could be traced through the FSM if an error occurred and the state transition logic fixed easily. The displays on the labkit were useful as they were set to display FSM_num and next_char.

VI. Conclusion

The main challenge of the project was the development and implementation of the writing recognition algorithm. While the current method of dividing the recognizable characters into seventeen subsets neatly segments a larger pool of possibilities into smaller, more easily manageable sections, the initial requirement of predetermined starting and stopping areas decreases the usability of the algorithm. The user must remember the specific instructions for each character to use the program as the instructions distort the usual image of the character.

A source of the difficulties with writing recognition lies in the individual nuances of each character. Using only 8 cell divisions, the writer must exaggerate his movements with the mouse to ensure that correct character is identified. Dividing the writing pad into greater numbers of cells would help alleviate this issue but would introduce an added level of complexity to the algorithm and result in increased memory requirements and analysis times.

Sources

[1] “6.111 Handouts” <http://web.mit.edu/6.111/www/f2005/index.html>.