A HARDWARE BASED BRAILLE NOTE TAKER

Xuan Zhang, Cesar Ortega-Sanchez and Iain Murray

Curtin University of Technology
Electrical and Computer Engineering Department
Kent Street, Bentley 6102, Western Australia
+618-9266-4540

i.murray@ece.curtin.edu.au

ABSTRACT

This paper describes a Braille note taker implemented in hardware. The system is able to perform Braille to text translation as well as note taking. A method is presented on how to achieve Braille note taking using a Braille keyboard. To perform Braille to text translation, a translating system has been built based on previous work. Using Very high speed integrated circuit Hardware Description Language (VHDL) and a Field Programmable Gate Arrays (FPGAs) development platform, a system that includes the keyboard controller and translator has been hierarchically described and implemented.

1. INTRODUCTION

Since Braille became one of the most important ways for the blind to learn and obtain information, many kinds of Braille products were invented to help the bind improve their life quality.

Among these products, Braille note takers are popular ones. Nowadays, Braille note takers are widely used by blind people. Especially in the educational context, a note taker is an essential device used by blind students to carry out assignments and take notes.

Most of commercial Braille note takers utilize microcontrollers with software running in them to perform multi-functions including note taking, translation, real time speaking, etc. However, in order to build a universal Braille system on a chip, a Braille note taker is going to be used as a single component implemented in hardware, while a microcontroller is used for interface and control purposes.

Up to six buttons need to be pressed when typing Braille; hence a matrix-like keyboard is not suitable for building Braille keyboards because of overlapping problems. Instead, optical detectors are used to detect when buttons are depressed [1]. For instance, one solution is to use infrared light source/sensor pairs and microcontrollers [2]. When the key is depressed, it breaks the light beam between the source and sensor. Thus, the signal received at the microcontroller from the sensor is changed whenever the key is depressed.

This paper presents a new method for implementing a Braille keyboard using a simple 4 x 6 push button matrix. To solve the overlap problems, the buttons have to be properly organized and located. A keyboard controller was described in VHDL to send and receive scan codes.

In performing Braille to text translation, software-based methods are a good solution. With the use of computer, the internal representation of characters allows the simultaneous display of characters on the screen and on a refreshable Braille display [3]. For this purpose, American Standard Code for Information Interchange (ASCII) is employed.

In software-based methods, sixty-four ASCII codes, referred to as Braille ASCII codes are employed to represent the sixty-four basic Braille characters. Therefore, the translating process can be regarded as the conversion from ASCII codes into Braille ASCII codes.

To perform Braille to text translation this paper presents a novel method to implement hardware-based Braille-to-Text translation. This Braille to text translation system is based on previous work, and the algorithm is based on a translating system proposed by Paul Blenkhorn [4] [5].

2. 4 X 6 MATRIX KEYBOARD

Figure 1 shows the layout of Braille keyboard. Following the prevailing design style, this Braille keyboard is with function and direction keys on it. To build an easy access machine for blind people, the function keys can be defined as speaking, printing, translating, embossing etc. Thus the direction keys can be used as shift, control, capital lock and backspace.

Figure 2 shows the schematic diagram of the 4 x 6 matrix keyboard. From the layout of the schematic, it is clear that the six Braille keys are located in the first column, twelve functions keys in the second and third column, and enter, space and direction keys in the fourth.

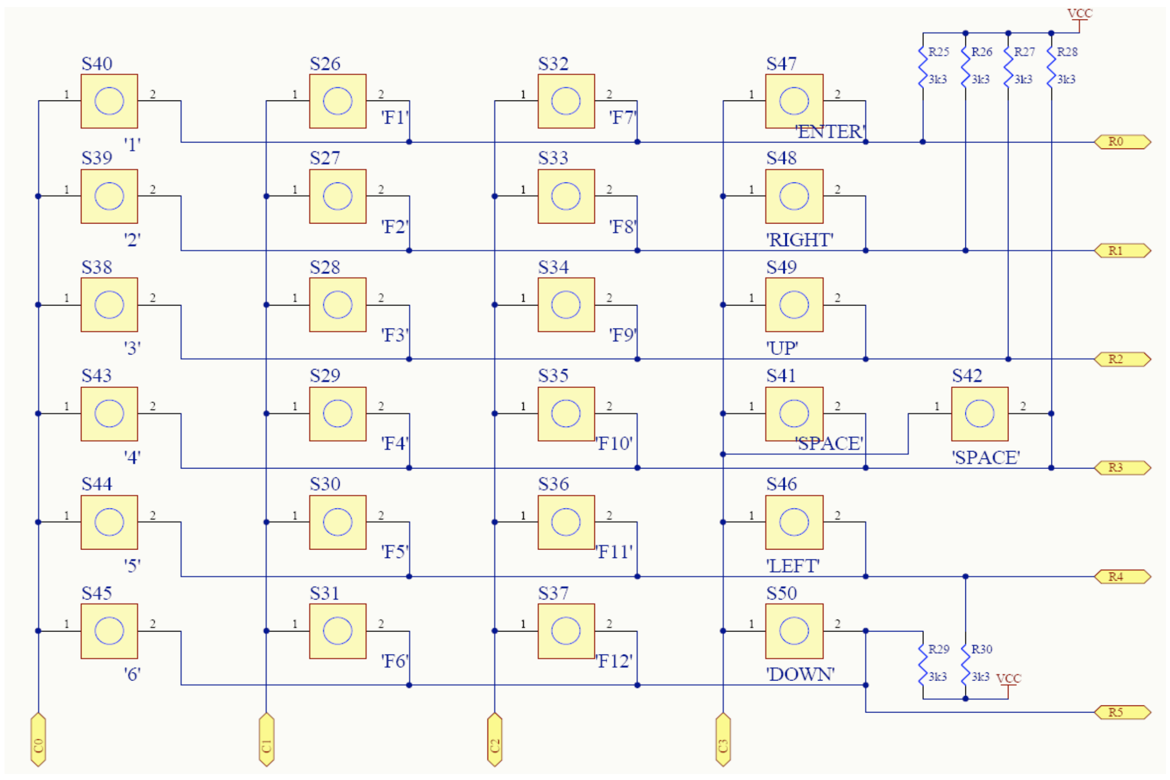
Fig. 2. Schematic Diagram of Braille keyboard

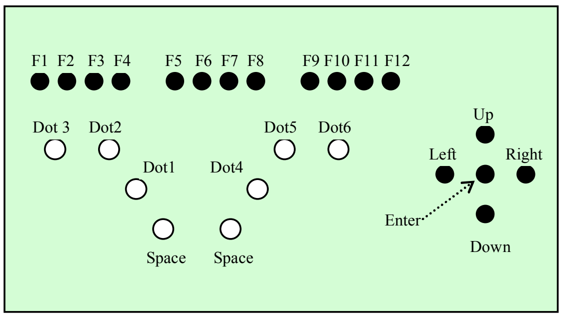

Fig. 1. Layout of Braille Keyboard

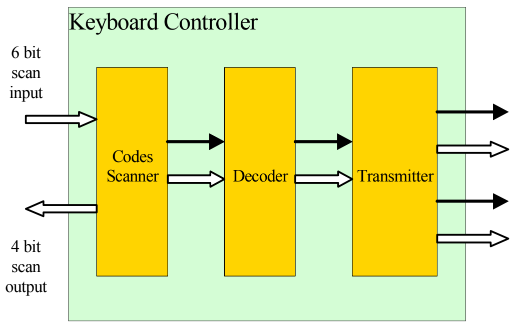

Fig. 3. Block Diagram of Keyboard Controller

The keyboard controller in FPGA keeps sending four bit scan codes to the keyboard inputs from C0 to C3 and scans one column at a time. The controller sends one bit low signal to the corresponding column to be scanned, and for others three columns sends high signals.

To overcome the bouncing problem, the controller reads the 6-bit output of the keyboard 13 milliseconds later after sending scan codes. If no button is pressed, the outputs from keyboard remain high. Once one button is pressed, the output at the row where the button is located becomes low.

The disadvantage for matrix keyboard is overlap. When one or two keys are depressed, the keyboard can work well. However, if keys to be pressed are more than two, overlaps may happen.

For example, in figure 2, if keys s26, s39, and s40 are depressed, the wire which is connected to C0, R0 and R1 is conductive. Therefore, when the controller sends a low signal to scan the second column, this low signal passes through three buttons and goes to the output R1. The result is that instead of getting three keys pressed, the controller gets four which are S26, s27, s39, and s40. To avoid this, the six Braille keys are kept in the same column.

Figure 3 shows the block diagram of the Braille keyboard controller.

The block codes scanner generates 4-bit outputs with only one of the bits set to zero; and gets 6-bit inputs coming back from the keyboard. There is a state machine working in the FPGA to generate scan codes for the keys that were pressed. The state machine works as follows:

1. In the first state, the scanner sends four bits codes "0111" scanning the first column. After jumping over the bouncing process which is about 13 milliseconds, the six bits input codes will be checked. If the value is all ones, i.e. "111111", it means that no button was pressed in the first column. This value will be stored in a specific register, called Release Register. A logic AND manipulation happens to the input codes and another register which was initialized to all ones, and the results will be saved in the same register, if the input value is not all ones.
2. The same process is repeated in the next three states to scan the next three columns. But instead of "0111", codes "1011", "1101", and "1110" are sent respectively in the three states.
3. In the fifth state, the four Release Registers are checked to see if all buttons have been released. If they have not, the state machine will go back to the first state to repeat the operation. If all buttons are released, the input codes stored in registers are sent to the next block, and the registers will be set to all ones.

The decoder block receives the data from the codes scanner and decodes the 6-bit input to Braille ASCII codes or particular control codes corresponding to the particular buttons that were pressed. The transmitter block outputs the data to the serial transmitter or Braille-to-Text translator depending on the particular function selected.

3. BRAILLE-TO-TEXT TRANSLATOR

3.1 PAUL BLENKHORN'S ALGORITHM

The early work on computerized translation of Braille was basically concerned with the translation from text in Braille [6] [7]. One solution, for instance, is the use of production rules derived from a Markov system. This approach has been followed by W. A. Slaby [8]. However, this solution results in a rapid increase of the number of productions rules.

In 1980's, Slaby developed another system for German contracted Braille translation which uses rules with left and right contexts. This system allows non-experts to modify rules to perform translation of different languages into Braille [5] [9]. However, this algorithm used by text to Braille translation was also applicable to the Braille to text translation.

Based on Slaby's system, in 1995, Paul Blenkhorn's proposed a system to convert Standard English Braille into text [5]. This method uses a decision table with input classes and states, and a rule table containing all rules for translation. This system can be readily updated and modified by people who are not experts in computer algorithms.

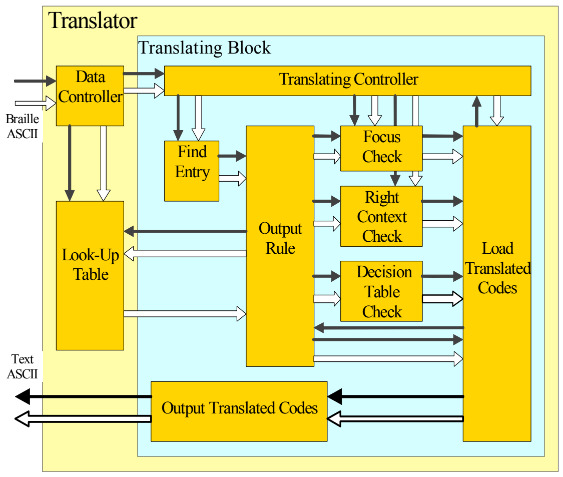

Fig. 4. Block diagram of Braille to text translator

The format of each row in the table is:
Input class <TAB> Rule <TAB> New state
Every rule has the following format:
[Focus] Right context = input text

According to Blenkhorn's algorithm, all rules are listed in ASCII alphabetical order. For rules whose focuses start with the same character(s), the order in which they appear in the table is related to their priority. So, the rules have to be checked in order from top to bottom. The first rule which is found has to be used. Therefore, actually the Blenkhorn's algorithm can be regarded as a Markov system [10].

In Blenkhorn's system, there are totally 498 rules in the translation table. Because the format of each rule is simple and each part of one rule can be checked separately, therefore, it makes the system easily be implemented in hardware.

3.2 ARCHITECTURE OF THE TRANSLATOR

Figure 4 shows a block diagram of the Braille-to-Text translator implemented in an FPGA. It shows that the translating block consists of 8 sub-blocks.

The translating-controller block gets feedback from the load-translated-codes block and also receives and stores the Braille codes in registers. Before translation, the rule table needs to be saved to the look-up-table block which consists of FPGA's RAM-blocks.

In this particular implementation, the translator carries out the translation one word at a time. Words are assumed to be separated by spaces.

In Figure 4 the find-entry block receives one entry character from the translating-controller block and outputs a particular address to the output-rule block. The entry character is the first un-translated character in the input text string.

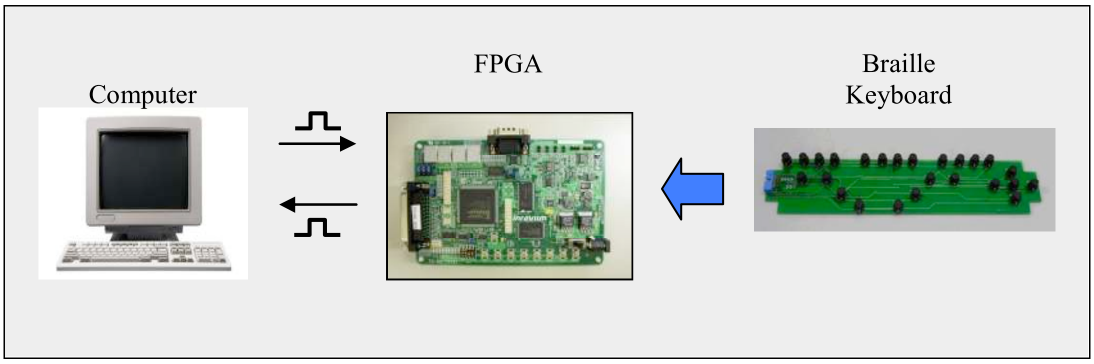

Fig.5. Test Bench for Braille Note Taker

In find-entry block, there is an addresses decoder located. If an address corresponds to a particular entry character, it is sent to the output-rule block. However, if no entry address can be found for a particular character, it means the entry character is not in the list. Therefore, a fail signal is issued and the character will be output without translation.

Two operations keep running in the output-rule block. One is reading rules from the look-up-table block, and the other is sending every single rule to focus-check, right-context-check, decision-table-check, and load-translated-codes blocks. The output-rule block receives signals from the find-entry block obtaining addresses, and signals from the load-translated-codes block that indicate if the output rule can be used.

The output-rule block sends an address to the look-up-table to read one rule at a time and sends it separately to focus-check, right-context-check and decision-table-check blocks. If the rule does not find a match, then a signal is generated and the output-rule block gets the next rule and sends it. This process continues until a match is found and the focus is successfully translated. The decision-table-check block checks current state and input class according to the look up table. The focus-check and right-context-check blocks receive not only the rule from output-rule block, but also the whole group of words to be translated from the translating controller because more than one letter of focus and right context might need to be checked. These two blocks perform similar functions.

As shown in figure 4, the focus-check, right-context-check and decision-table-check blocks work concurrently, providing better performance than sequential implementations. Each block generates signals for the load-translated-codes block indicating if the focus, the right context or the decision table were successfully matched. If one of the three fails, then a signal is sent back to the output-rule block requesting the next rule. Otherwise, the load-translated-codes block sends the translated codes to the output-translated-codes block, and informs to the translating-controller block how many characters were translated. After one group of characters has been translated, the output-translated-codes block transmits the corresponding Braille ASCII characters one by one. Then the translation of a new set of characters begins.

5. IMPLEMENTATION AND TEST

The system has been successfully implemented in a Xilinx's Virtex-4 FPGA evaluation board [11]. The synthesis results show that this application is a resource-saved design, because only 2274 over 7168 slices are used and 37 percent block RAMs are occupied.

The current version note taker only has two functions which are note taking and Braille to text translation. Therefore, only two function keys have been defined, F1 for note taking, and F2 for translation.

Figure 5 shows the system that was implemented for testing purposes. The RS-232 serial connection is used to send the rule table to the look-up table and save the Braille codes or translation results in a text file. The system works as follows:

1. The rule table is sent to the FPGA through a serial link using Hyper Terminal before operation starts.
2. If the note taking function is selected, Braille code which is typed in will be sent to a serial transmitter directly.
3. If the translation function is selected, the translator takes the Braille character and stores it in a buffer. Characters are stored until a space or carriage return is detected. At this point the translation process described in section 3 takes place. The results of the translation are sent to a serial transmitter so that they can be received and stored in a text file by the computer.

For the implementation reported in this paper, the FPGA receives a text file containing the rule table and sends the translated text or typed Braille characters back to the PC at 57,600 bauds.

The test results show that the Braille keyboard is able to run the note taking well, and the translator can achieve Braille-to-Text translation as good as commercial software.

6. CONCLUSIONS AND FUTURE WORK

The implementation of a hardware-based Braille note-taker has been described in this paper. The system has been implemented and tested using VHDL and FPGAs. The resulting integrated circuit can be incorporated into a portable text processor for the blind, removing the need to rely on a computer to perform Braille-to-text translation or printer control.

At the current stage, this Braille note taker can be used as a stand-alone component. However, it also can be included in a universal Braille system on a chip for the future work.

In this case, more functions will be considered. For instance, a real-time speaking function will supply auditory feedback to allow the detection and correction of typing errors. Furthermore, interfaces to control a Braille embosser and a printing machine will also be incorporated.

7. REFERENCE

[1] J. Spragg, “Interfacing a Perkins Brailler to a BBC micro,” Microprocessors Microsyst., vol. 8, pp. 524–527, 1984.

[2] Evans, D.G. Pettitt, S. Blenkhorn, P., “A Modified Perkins Brailler for Text Entry into Windows Applications”, Neural Systems and Rehabilitation Engineering, IEEE Transactions on Rehabilitation Engineering, Volume: 10, Issue: 3, pp. 204 – 206, 2002

[3] Durre, K. P 1990, “BrailleButler: A new approach to non-visual computer applications”, in Proceedings of the 3rd Annual IEEE Symposium on Computer-Based Medical Systems, North Carolina at Chapel Hill, pp.97-104

[4] Zhang X., Ortega-Sanchez C., and Murray I., “Hardware-Based Text-to-Braille Translator”, the 8th International ACM SIGACCESS Conference on Computers & Accessibility, Portland, Oregon, USA, pp. 229-230, 2006

[5] Blenkhorn, P. “A System for Converting Braille into Print”, IEEE transactions on Rehabilitation Engineering, vol. 3, no. 2, pp.215-221, 1995.

[6] Jonathen, A 1972, “Recent Improvement in Braille Transcription”, in Proceedings of the ACM annual Conference, vol. 1, Boston, pp. 208-218

[7] Jørgen Vinding, 1975, “Computerized Braille production”, ACM SIGCAPH Computers and the Physically Handicapped, issue 15, pp. 35-40

[8] Wolfgang, A. Slaby, 1975, “The MARKOV system of production rules: a universal Braille translator”, ACM SIGCAPH Computers and the Physically Handicapped, issue 15, pp. 53-59

[9] Wolfgang, A. Slaby, 1990, “Computerized Braille Translation”, Journal of Microcomputer Appl., vol. 13, issue n2, pp. 107-113

[10] Peter, L., Introduction to Formal Languages and Automata, Boston: Jones and Bartlett, 2001, ISBN: 0763714224

[11] Memec Inc. Spartan-3 LC Development Board User’s Guide, electronic documentation, version 2.0, 2004.