A SYSTEM FOR FAST TEXT-TO-BRAILLE TRANSLATION BASED ON FPGAS

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 fast text to Braille translator based on Field Programmable Gate Arrays (FPGAs). Compared with most commercial methods, this translator is able to carry out the translation in hardware instead of using software. To achieve the fast translation, a FPGA with big programmable resource has been utilized, and an algorithm, proposed by Paul Blenkhorn, has been revised to perform the fast translation. The translator has been described using Very high speed integrated circuit Hardware Description Language (VHDL). The test results indicate that the hardware-based translator achieves the same results as software-based commercial translators, and moreover, this system achieves superior throughput compared to Blenkhorn's original algorithm.

1. INTRODUCTION

In 1829, Louis Braille developed a system which allowed the blind to read and write. Braille's system employed a 6-dot cell and was based upon normal spelling. 6 dots can represent only sixty-four combinations including the blank symbol. In the combinations, there are 26 alphabetic letters, decimal numbers, punctuations and sing marks.

Although it is possible to transcribe Braille by simply substituting the equivalent Braille character for its printed equivalent, such character-by-character transcription, known as Grade 1 Braille, is used only by beginners because of its low throughput.

Partly because of the bulk problem of the original Braille [2], and partly to improve the speed of writing and reading, English and many another languages employ contractions [1] [3]. When contractions are used, Braille is usually called "grade 2". In English, almost all Braille is grade 2 with 189 contractions [2].

Since Braille became one of the most important ways for the blind to learn and obtain information, translating normal text into Braille became a necessity. However, considering the number of printed materials to be translated, a fast automatic method to achieve the translation is needed.

Today, most Braille translators rely on the use of a computer and the American Standard Code for Information Interchange (ASCII). In software-based translators, sixty-four ASCII codes, referred to as Braille ASCII codes are employed to represent the sixty-four basic Braille characters. Therefore, the translating process becomes the conversion from ASCII codes into Braille ASCII codes [4].

The most common approach for text-to-Braille translation is the use of programs running in personal computers. Several commercial translating programs are available, such as Duxbury Braille Translator, Megadots and WinBraille. Another solution for text to Braille translation is portable devices. Most of them have multifunction including text-to-Braille translation, such as Mountbatten Brailler. These devices are based on a microcontroller running a translating program. However, when mass text documents need to be translated, a faster method for text to Braille translation is obviously preferred.

Therefore, instead of using software, this paper presents a hardware based solution. Using a FPGA with big programmable resource, we provide a fast parallel method to achieve fast text to Braille translation.

2. COMPUTERIZED BRAILLE TRANSLATION

Several proposals have been made for computerized text-to-Braille translation. One solution, for instance, is the use of production rules derived from a Markov system. This approach has been followed by W. A. Slaby [5]. 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 [1] [6].

Based on Slaby's algorithm, Paul Blenkhorn's proposed a system to convert text into Standard English Braille [1]. This method uses a decision table with input classes and states and a table with more than one thousand rules for translation. The format of each row in the table is:

1-4244-0606-4/07/$20.00 ©2007 IEEE.

Table 1. Fragment of Rule Table

	Input Class	Left Context	Focus	Right Context	Output Text	New State
1	2	~	G	;#	;G	3
2	2	#	G		;G	3
3	1	~	G	;#	;G	4
4	1	#	G		;G	4
5	2	!	GHAI		GHAI	-
6	2	!	GHEAD		GH1D	-
7	2	!	GHEAP		GH1P	-
8	2	!	GHIL		GHIL	-
9	2	!	GHOL	E	GHOL	-
10	2	!	GHOR	N	GHOR	-
11	2	!	GHOUS	E	GH\S	-
12	2		GHUN	T	GHUN	-
13	2		GH		<	-
14	2		GOOD		GD	-
15	2		GOVERN	ESS	GOV]N	-
16	2	~	GO	~	G	-
17	2	!	GG	!	7	-
18	2	.	GREAT		GRT	-
19	1		G	.!	G	-
20	1		G	~	G	-
21	1		G	~	G	-
22	1		G		G	-

Input class <TAB> Rule <TAB> New state

Every rule has the following format:

Left context [focus] Right context = input text

Several wildcards can be used in the left context and the right context. These are as follows:

"!" a letter;

"#" a number;

"!" a space or punctuation (include apostrophe);

"space" only a space character;

"|" zero or more capital signs;

"`" one or more characters that are potentially roman numerals;

";" zero or more letters;

"+" one or more digits;

An example is given here to explain how Blenkhorn's system works. A rule table shown in Table 1 is used for translation.

Assume that we want to translate the word "GO". If the word is between two spaces, then we can use the spaces as the left and right contexts. For the first step, the system will find the table entry according to the first letter of this word. Obviously, the entry is letter 'G'. Then, the system will go through the rules of letter 'G', and check the rules including focus, input class, present state, left and right context, one by one, until finding the rule "2~[GO]~=G-". Because all the information of this rule matches the input, the translated result is "G". The hyphen mark for new state means that new state keeps unchanged.

Blenkhorn's algorithm has been implemented in a C program, proving that the algorithm works well. However, modifications are necessary for its implementation in hardware.

In the system presented in this paper, input class and sates are not used, because when the system performs the Grade 1 and Grade 2 Braille translation all the rules, except those for letter signs with the index input class 1 or 2, have present state as either 1 or 2. Therefore, those rules always have a value of 1 according to the decision table.

On the other hand, rules which have present states 3 and 4 in the rule table are always valid and once the next space character is found, the system will change the state to 1 or 2. In summary, if computer Braille is not going to be considered, the decision table is not necessary.

3. HARDWARE-BASED FAST TRANSLATION

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 [7].

Take the rule table with the letter 'A' started as an example to explain the translation process. There are 50 rules in this group, and the terminating rule is "1 [A] = [A] -". If a contraction "AR" needs to be translated, the system has to check the 5 rules before the rule "2 [AR] = & -". In this case, the string "AR" can be translated fast. But, if a word "ALTOGETHER" needs to be translated, the system has to check 36 rules before the rule "2 ~ [ALTOGETHER] = ALT -". Especially, when only the terminating rule has to be used, the translation speed will be significantly slow down. However, the translation process can be accelerated, if the 'A' group is separated into small subgroups which can be used in parallel.

Therefore, in this paper, a parallel translating method is discussed. To achieve fast translation, independent translating cells have been built. In each cell, there is an alphabetically ordered sub-table. During translation process, those translating cells which are activated perform translation concurrently.

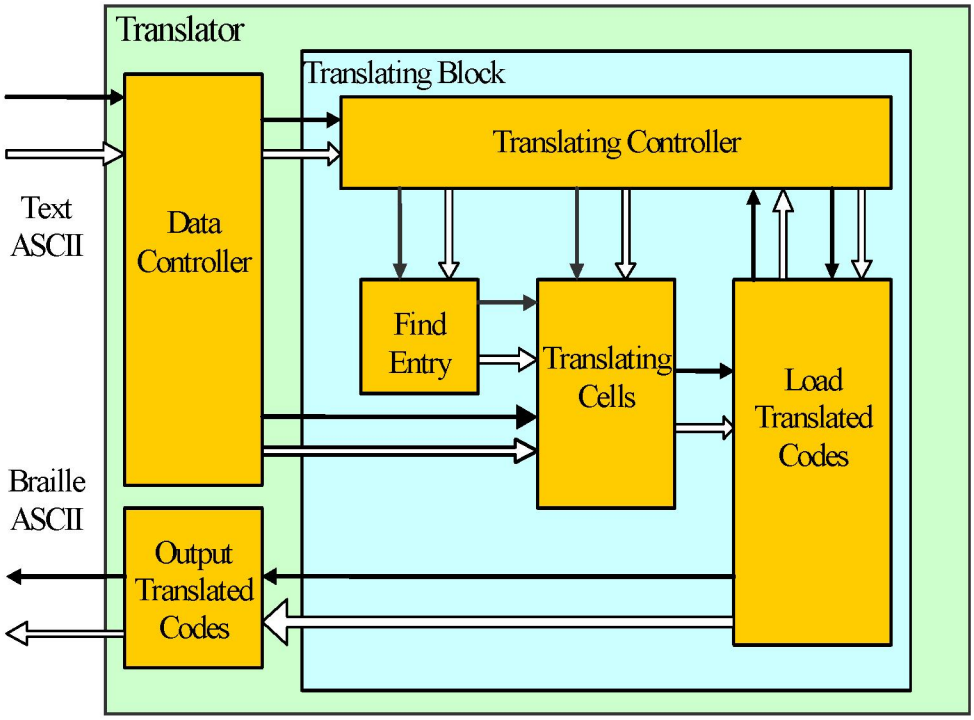

Fig. 1. Block Diagram of Text to Braille Translator

The principles of generating subgroups can be described as follows:

- Keep the original order of the rule table unchanged.
- For letter rules, use original terminating rules as one single subgroup, called terminating subgroup. The cell which stores the terminating rules is called terminating cell. Therefore, when translation is performed, the terminating subgroup never fails to be used.
- Rules have to be separated into groups properly, so that only one translating cell except the terminating cell is able to apply a particular rule successfully during translating process. Therefore, if one rule's focus is part of another rule's, and there is no left and right context to distinguish these two rules, they can not be separated.

Take the 'A' rules as an example to explain the principle of generating subgroups. In the 'A' table, those rules with focus "AND " and "AND" are used as a subgroup. In this case, it will not happen that the contraction "AND" will be translated by two cells. The rules started with the string "AFTER" need to be used in one subgroup. Using this method, the 'A' rules can be separated into 7 subgroups, while the biggest table, 'B' table with 122 rules, can be separated into 9 subgroups. For those tables with small number of rules, such as 'J' table which only has 10 rules, it is not necessary to separate into subgroups.

4. ARCHITECTURE OF THE SYSTEM

Figure 1 shows a block diagram of text to Braille translator implemented in an FPGA. Before the translation starts, the data-controller receives the rule tables and distributes them to particular block RAMs located in translating cells. Then the data-controller is ready to receive text.

The translating controller block gets feedback from the load-translated-codes block and also receives and stores the text data in registers. The load-translated-codes block feeds back the number of translated characters so that the translating controller can skip over those characters and find a new entry. The entry character is sent to the find-entry block. The original text is sent to translating cells. In this particular implementation, the translator carries out the conversion word by word and five words at a time.

The find-entry block receives one entry character from the translating-controller and outputs addresses for the corresponding translating-cells. The entry character is the first un-translated character in the input text string. In find-entry block, there is an address decoder that translates the entry characters into addresses. If no entry address can be found for a particular character, then the untranslated character and a fail signal are sent to the output-translated-codes block.

The translating cells receive un-translated codes from the translating-controller as well as addresses from the find-entry block. The parallel translating processes is shown in Figure 2. Those cells which received addresses will carry out the translation.

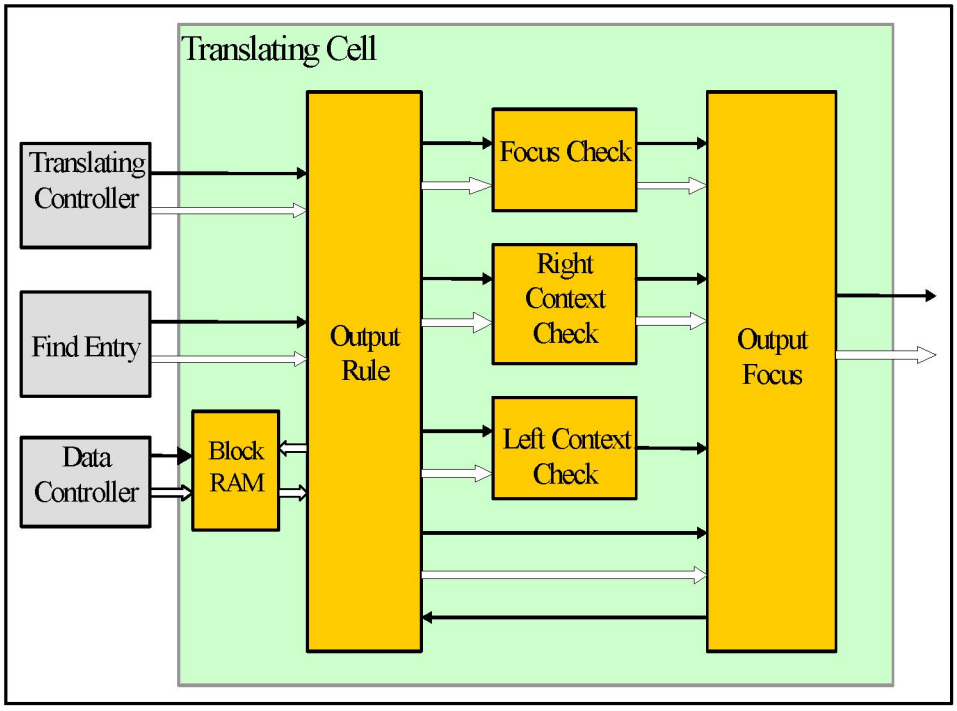

Fig. 3. Block diagram of a translating cell

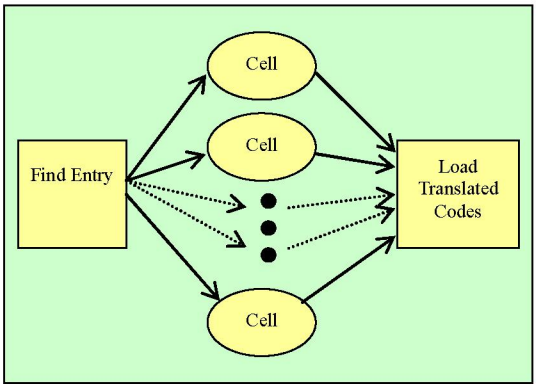

Fig. 2. Translating in Parallel

Figure 3 shows the block diagram of a translating cell. Every cell has a block RAM where the table rules are stored in alphabetical ordered. Before the translation process starts, the un-translated codes from the translating-controller are sent to the focus and right-context check blocks by the output-rule block. Then the output-rule receives an address and gets a particular rule from the rule table. The rule will be separately sent to the three following blocks. The focus, right-context and left-context check blocks are built using finite state machines which are able to check if the rule can be applied.

As shown in figure 3, the three blocks work concurrently, providing better performance than sequential implementations [8].

Each block generates signals for the output-focus block indicating if the focus, the right context or the left context were successfully matched. The translation output will be sent to the load-translated-codes block. If one of the three fails, then a signal is sent back to the output-rule block requesting the next rule. If no rule can be used, a signal will be generated and sent to the load-translated-codes block indicating that the translating cell cannot find a match for translation.

The load-translated-codes block will receive translation results from the terminating cell or one of other cells. The terminating never will fail to be applied. However, compared with other cells, the terminating cell has lower priority. Therefore, if the load-translated-codes block receives translated codes from two cells respectively, the codes from terminating cell will be discarded.

Therefore, the load-translated-codes block will output the translation according to set priorities. Meanwhile, it will send signals to the translating-controller block to indicate 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 can begin.

5. IMPLEMENTATION AND TEST

The translator has been implemented using a Top-Down design methodology where high level functions are defined first, and the lower level implementation details are filled in later [9].

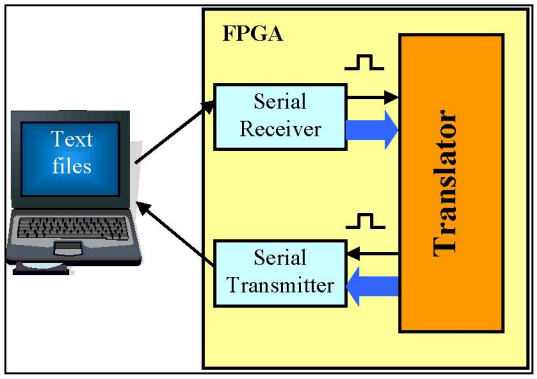

Fig. 4. Test bench for Braille translator

Table 2. Timing comparison between sequential and parallel method

Un-translated focuses	Translation results	Time by sequential method (clock cycles)	Time by parallel method (clock cycles)
AND	&	137	136
AS	Z	309	34
ABOUT	AB	326	52
AGAIN	AG	360	86
AFTER	AF	465	190
ALSO	AL	543	34
ALREADY	ALR	582	67
ALTOGETHER	ALT	622	101
ACCORDING	AC	663	134
AINES	A9NES	689	34
A	A	770	207

The system has been successfully implemented in a Xilinx's Virtex-4 FPGA evaluation board [10]. The texts to be translated, as well as the results of the translation were stored in a PC as text files and transmitted using an RS-232 serial connection.

Figure 4 shows the setting used to test the translator. The system works as follows:

1. The text to be translated is sent to the FPGA through a serial link using Hyper Terminal.
2. Part of the FPGA implements a receiver that converts serial data into bytes that are loaded into the translator.
3. The translator takes the new character and stores it in a buffer. Characters are stored until a space is detected. At this point the translation process described in section 2 takes place.
4. 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 the text file to be translated at 4,800 bauds and sends the translated text back to the PC at 57,600 bauds. In this setting, because this system is only able to translate groups of characters, after translation is done, the serial transmitter has to send all the translated codes to computer before the next group of text is received. That is why the baud rate for transmitting is 12 times as fast as receiving.

To simplify the implementation, all rules were modified to be of the same length. ASCII code 0 was used as end-sign for every part of the rule. Although this method increases the amount of memory required, Virtex4 FPGAs have dedicated memory blocks that can contain the complete table.

All the blocks of the serial communication and the translator were implemented in VHDL. Xilinx's ISE FPGA-development suite was used for system implementation, synthesis, simulation, and FPGA configuration.

For testing, outputs of the hardware translator were compared with the outputs of the previous work which uses sequential translating method [11]. The results show that using parallel method is able to perform translations with superior speed. Using the simulation tool, ModelSim, the numbers of clock cycles using sequential and parallel method can be accurately calculated. The sample test results are shown in Table 2.

6. CONCLUSIONS AND FUTURE WORK

The design and implementation of an FPGA-based, fast text-to-Braille translator has been presented. In its current version, the system can be used in embedded and high-performance applications. However, there are several improvements which will be incorporated in future versions of the hardware translator.

For example, the current system is a stand-alone component. Its structure has to be changed for every individual application. An improved version will incorporate the hardware translator in a system on a chip for multifunctional text-Braille translation. The system will consist of a microcontroller for interface and control, and the text-Braille translator, all integrated in one single chip.

For further improvement, a multi-language-Braille translator will be considered. Look-up tables for different languages could be stored in flash memory so that when translation of text in a particular language is required, the microcontroller loads the corresponding look-up table into the FPGA.

Standards for Braille translation are much higher than for print. This level of accuracy is necessary because Braille uses the same ASCII code for different purposes according to the context. Hence, even slight errors can cause extreme difficulties in interpretation. The results obtained with the hardware-based translator show that the system is able to implement text-to-Braille translation with high accuracy.

7. REFERENCES

[1] Blenkhorn, P., "A System for Converting Print into Braille", IEEE Transactions on Rehabilitation Engineering, vol. 5, No. 2, pp. 121-129, 1997.

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

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

[4] Jørgen V., "Computerized Braille production", ACM SIGCAPH Computers and the Physically Handicapped, issue 15, pp. 35-40, 1975.

[5] 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

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

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

[8] 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

[9] Zeidman, B., Designing with FPGAs and CPLDs, CMP books, 2002, ISBN: 1-57820-112-8.

[10] Memec Inc. Virtex-4(TM) FX12 LC Development Board User's Guide, electronic documentation, version 1.0, 2005.

[11] Zhang X., Ortega- Sanchez C., and Murray I., "Text-to-Braille Translator in a Chip", International Conference on Electrical and Computer Engineering, 2006