4th International Conference on Electrical and Computer Engineering
ICECE 2006, 19-21 December 2006, Dhaka, Bangladesh

TEXT-TO-BRAILLE TRANSLATOR IN A CHIP

Xuan Zhang, Cesar Ortega-Sanchez, and Iain Murray

Electrical and Computer Engineering Department
Curtin University of Technology
Kent Street, Bentley 6102, Western Australia
E-mail: i.murray@ece.curtin.edu.au

ABSTRACT

This paper describes the hardware implementation of a text to Braille Translator using Field-Programmable Gate Arrays (FPGAs). Different from most commercial software-based translators, the circuit presented in this paper is able to carry out text-to-Braille translation in hardware. The translator is based on the translating algorithm, proposed by Paul Blenkhorn [1]. The Very high speed Hardware Description Language (VHDL) was used to describe the chip in a hierarchical way. The test results indicate that the hardware-based translator achieves the same results as software-based commercial translators, with superior throughput.

1. INTRODUCTION

Before the invention of the original dot-based written language for the blind early in the nineteenth century, visually impaired people was, for all practical purposes, unable to read. The dot-based language made a big contribution to improve the way visually impaired people learnt and communicated with the rest of the world. However, the original dot-based language was not easy to use because it employed twelve dots to represent characters, with up to 4096 possible combinations.

In 1829 Louis Braille developed a system based on a 6-dot cell, which allowed the blind to read and write more easily. In the 64 possible combinations, there are 26 alphabetic letters, decimal numbers, punctuations and sign marks.

Useful as it was, the original Braille system suffered from low speed because text had to be spelled letter by letter. To solve this problem, English and other languages introduced the use of contractions [1, 2, 3]. When contractions are used, Braille is usually called "grade 2" in contrast to "grade 1" transcriptions where all words are spelled out letter-by-letter. In English, almost all Braille is grade 2 with 189 contractions [3].

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, manual translation is time-consuming and prone to have errors; hence different devices to perform automatic translation have been conceived.

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].

Another solution for text to Braille translation is portable devices. These devices are based on a microcontroller running a translating program that in most cases relies on a dictionary or a look-up table to carry out the translation [5].

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 rule table with all rules for translation [6, 7]. The format of each row in the table is:

Input class <TAB> Rule <TAB> New state

The system presented in this paper only considers grade 2 Braille translations; hence the table can be simplified by ignoring input classes and states. Only rules are considered.

Every rule has the following format:

Left context [focus] Right context = input text

To make rules generic, several wildcards are used in the left context and the right context [2].

2 ARCHITECTURE OF THE SYSTEM

Figure 1 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 text data in registers. In this particular implementation, the maximum number of characters for one translation is forty, which is enough for five words. 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 both the focus-check block and the right-context-check block.

The find-entry block receives the entry character from the translating controller and outputs a particular address to the output-rule block. In this block, there is a look-up table which stores all the entry addresses. If an address corresponds to a particular entry character, this address and an address ready signal is sent to the output-rule block. However, if no entry address can be found for a particular character, then the character and a fail signal is sent to the output-translated-codes block.

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, left-context-check, and load-translated-codes blocks. Input signals for the output-rule block are all the outputs from the find-entry block, the look-up-table block and the feedback signals from load-translated-codes block that indicate if the output rule is used correctly.

The find-entry block sends an address to the look-up-table to read one rule and send it separately to focus-check, right-context-check, left-context-check and out-rule block. If the rule does not find a match, then a feedback signal is generated and the output-rule block gets the next rule and sends it until a match is found and the focus is successfully translated.

The focus can have one or more characters. Also, the same focus can have different left and right contexts. If a string is identical to the focus in a particular rule, the right and left contexts need to be checked as well. If all three parts match, the rule fires, and the string can be translated.

The focus-check and right-context-check blocks receive not only the rule form output-rule block, but

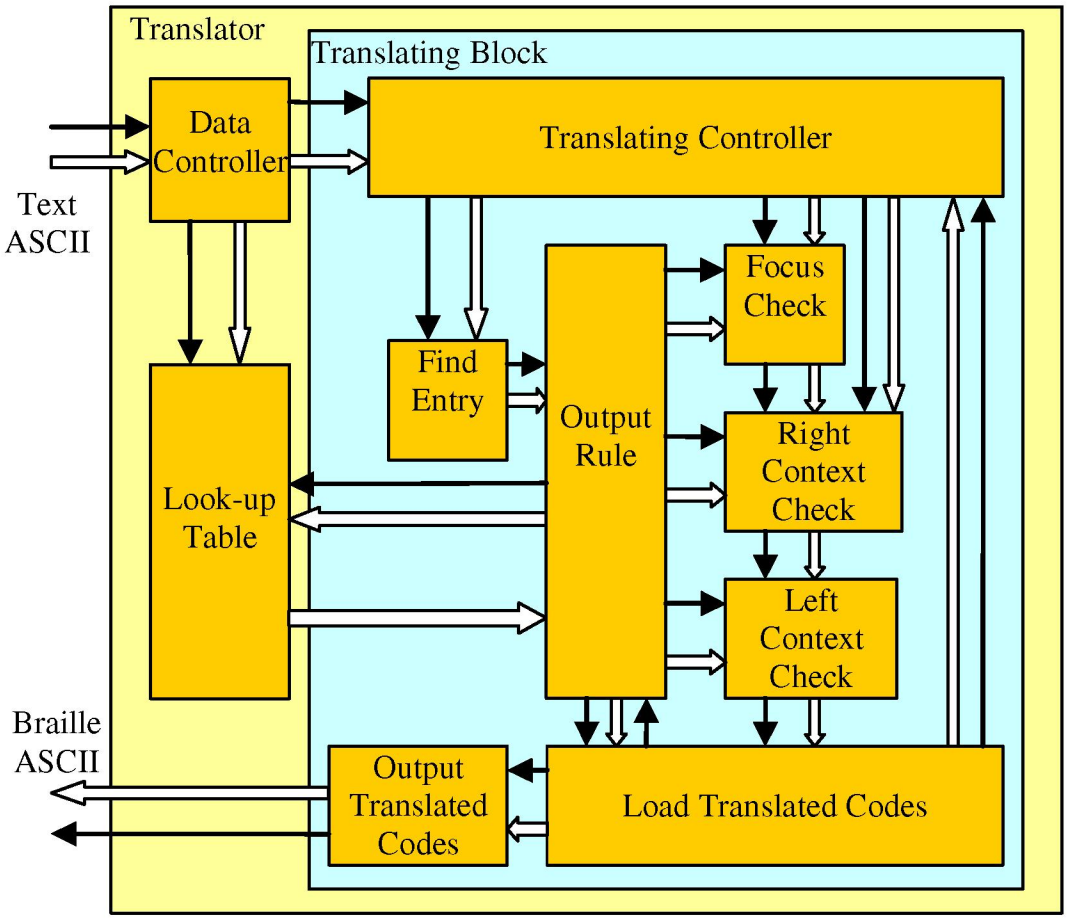

Figure 1. Block diagram of text to Braille translator

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 three blocks perform similar functions; hence only focus-check block are described next.

As shown in Figure 1, the focus is checked first, right context second and left context third. If the focus for a particular rule matches the input, then the letter number of the focus and a check complete signal are sent to the right-context-check block. However, if the focus does not match any of the rules, then a signal is sent to the right-context-check block. This signal is also passed to the left-context-check block, the load-translated-codes block, the output-rule block and finally to the translating controller.

If the focus, right context and left context match one of the rules, then the load-translated-codes block sends the translated codes, which were already stored in its registers, to the output-translated-codes block, and sends feedback signals to the translating controller to let it know how many characters were translated. After one group of characters has been translated, the output-translated-codes block transmits the characters one by one. Then a new cycle starts.

3 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 [8]. To implement the system, a Xilinx's Virtex-4 FPGA evaluation board was used. 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 2 shows the setting used to test the translator.

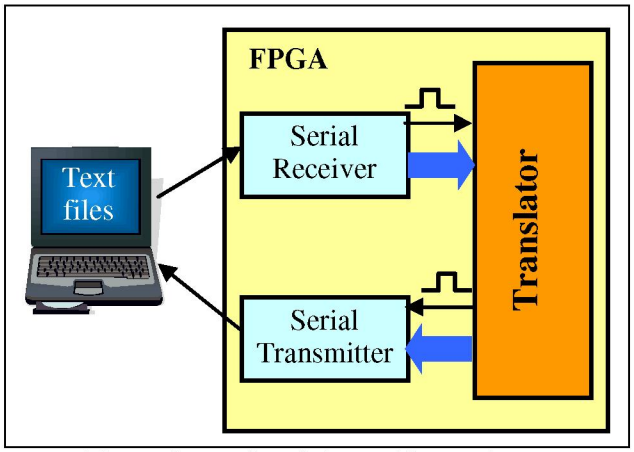

Figure 2. Test bench for Braille translator.

The system depicted in Figure 2 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. A signal is generated to indicate to the translator that a new character has just been received.
3. The translator takes the new character and stores it in a buffer. Characters are received and 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 in 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. The reason for this difference is that the current implementation works on complete words. This means that a whole word has to be translated and transmitted to the computer before the next word can be received. In high-performance applications, parallel communications could be used to increase the throughput of the system.

The FPGA evaluation board includes 64MB of DDR SDRAM, 4MB of Flash, USB-RS232 bridge, a 10/100/1000 Ethernet PHY, 100 MHz clock source, RS-232 port, and additional user support circuitry to develop a complete system [9]. For testing purposes, only the RS-232 port and FPGA have been involved. There is a PowerPC 405 processor integrated in the Xilinx XC4VFX12 FPGA, this processor will be involved in future work for embedded applications.

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 [10].

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.

During testing, outputs of the hardware translator were compared against the outputs of a commercial Braille translation program. The results show that the hardware translator is able to perform translations with the same accuracy as the commercial system.

4 CONCLUSIONS AND FUTURE WORK

The design and implementation of an FPGA-based, 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.

REFERENCES

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

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

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

[4] Durre I. and Tuttle D. "A Universal Computer Braille Code for Literary and Scientific Texts", International Technology Conference, December 1991.

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

[6] Slaby W. "The MARKOV system of production rules: a universal Braille translator", ACM SIGCAPH Computers and the Physically Handicapped, issue 15, pp. 53-59, 1975.

[7] Slaby W. "Computerized Braille Translation", Journal of Microcomputer Application, vol. 13, issue n2, pp. 107-113, 1990.

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

[9] Memec Company, "Virtex-4 FX12 LC Development Board User's Guide", electronic document, version 1.0, 2005.

[10] Xilinx Company, "Virtex-4 User Guide", electronic document, version 1.4, 2005.