Reconfigurable PDA for the Visually Impaired Using FPGAs

Xuan Zhang, Cesar Ortega-Sanchez and Iain Murray
Electrical and Computer Engineering Department
Curtin University of Technology, Perth, Western Australia
c.ortega@curtin.edu.au

Abstract

Curtin University Brailler (CUB) is a Personal Digital Assistant (PDA) for visually impaired people. Its objective is to make information in different formats accessible to people with limited visual ability. This paper presents the design and implementation of two modules: a print-to-Braille translation system and a Braille keyboard controller. The translator implements Blenkhorn's algorithm in hardware, liberating the microprocessor to perform other functions. The Braille keyboard controller along with a low cost keyboard provides users with a note-taking function. These modules are used as intellectual property (IP) cores coupled to a 32-bit MicroBlaze processor in an embedded system-on-a-chip (SoC). In its current implementation, the microprocessor uses a hierarchical interrupt scheme to coordinate IP cores. A prototype of the complete embedded system is under development using Xilinx's FPGAs. The system is a potential platform for the development of embedded systems to assist the visually impaired.

1. Introduction

Curtin University Brailler (CUB) is a Personal Digital Assistant (PDA) under development at Curtin University of Technology to address the special needs of visually impaired people. Functions of the device include print-to-Braille translation, Braille note taking, interface to printer and embosser, and double talk function. A current version of the CUB uses a microcontroller and off-the-shelf components. An FPGA-based version is currently being investigated. More details about CUB can be found in [1].

Figure 1 shows the FPGA-based CUB's block diagram. The system consists of peripheral devices connected to a MicroBlaze via CoreConnect's On-chip Peripheral Bus (OPB) [2]. Some of these devices are "virtual". They are IP cores that are loaded into reconfigurable areas of the FPGA at run-time, improving FPGA usage and overall performance.

In Figure 1 the Universal Asynchronous Receiver Transmitter (UART) is used to communicate the PDA with personal computers or embossers. The interrupt controller handles interrupt requests for the MicroBlaze. A Media Access Controller (MAC) coupled to an on-board physical layer network chip supplies a standard Ethernet connection. Other IP cores implement Print-to-Braille and Braille-to-print translation. Input devices include Braille keyboard, QWERTY keyboard and microphone (Double Talk). Output devices include standard printer, LCD, embosser and speakers or headphones. This paper presents IPs for Print-to-Braille translation and control of Braille keyboard.

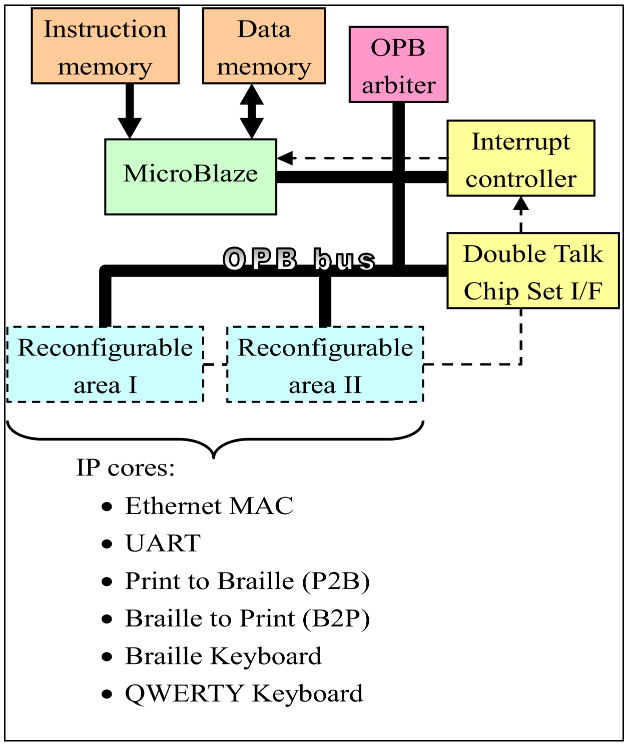


Figure 1. CUB's hardware block diagram

2. Print-to-Braile translation IP

2.1. Print-to-Braille translation

There are two approaches for the translation of text into Braille. One is simply substituting each Computer Braille code for its standard ASCII equivalent, such character-by-character translation, known as Grade 1 Braille, significantly slows down the reading speed [4]. Hence, English and many other languages employ contractions to represent information with a minimum number of Braille characters. When contractions are used, Braille is called Grade 2. In English, Grade 2 Braille uses 189 contractions [5, 6].

Currently, automatic text-to-Braille translation is done by software running in personal computers. Some of these programs can even perform multiple language translation [7]. Unfortunately, commercial software is designed mainly for users with healthy eyesight. Most commercial programs offer low or no accessibility to visually impaired users and are mainly used for the translation of existing printed texts.

In this paper we present an embedded system-on-chip that includes a hardware-based, print-to-Braille translator with greater throughput than commercial software.

2.2. Algorithm

Several approaches have been proposed for automatic text-to-Braille translation. In one instance, Slaby used production rules derived from a Markov system [10,11]. Even though Slaby's approach achieves accurate translations, his solution requires a large number of production rules.

Based on Slaby's work, Paul Blenkhorn proposed a system to convert text into Standard English Braille [12]. His method uses a decision table with input classes and states and a table with over a thousand rules for translation. The format of each row in the table is:

Input class <TAB> RULE <TAB> New state

Every RULE has the following format:

Left context [FOCUS] Right context = result text

Focus is the Braille string to be translated. During translation rules are checked one by one looking for a match against the FOCUS and its left and right contexts. If a match is found then the rule fires and the input string is substituted with the right-hand-side of the rule (result text).

2.3. Architecture

For its hardware implementation, the print-to-Braille translator was divided into 9 blocks. Figure 2 shows a diagram of the translator and its interface.

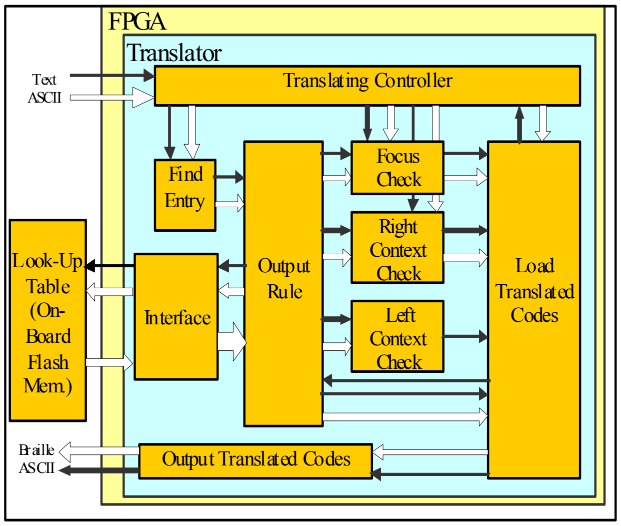


Figure 2. Block Diagram of Print-to-Braille Translator

In Figure 2 the rule table has to be stored in the on-board 16 MB Flash Memory using an independent module (not shown in Figure 2) before translation begins. To read the table, there is an interface between the Flash memory and the Output-Rule block.

The Translating-Controller gets a signal from the Load-Translated-Codes block to receive and store text in its data registers. In this implementation, the translator distinguishes words by detecting spaces.

The Find-Entry block contains addresses for entry rules in alphabetical order. When an entry character is received from the Translating-Controller, the Find-Entry block will output the corresponding address to the Output-Rule block. The entry character is defined as the first un-translated character in the input text string. 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 for Grade 1 translation.

Two operations take place in the Output-Rule block. One is reading rules from the look-up table through the interface, and the other is sending each part of a single rule to Focus-Check, Right-Context-check, Left-Context-Check, and Load-Translated-Codes blocks. The Output-Rule block receives an address from the Find-Entry block, and control signals from the Load-Translated-Codes block indicating if the output rule can be used. If the rule does not find a match, then a signal will be generated by the Load-Translated-Codes block requesting the Output-Rule block to get the next rule. This process continues until a match is found and the focus is successfully translated.

Notice in Figure 2 that the Focus-Check, Right-Context-Check and Left-Context-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 left context were successfully matched. If one of the three fails, then a signal is sent back to the Output-Rule block requesting the next rule. If the focus, right context and left context match one of the rules, then the Load-Translated-Codes block sends the translated codes to the Output-Translated-Codes block, and informs 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 Computer Braille characters one by one. Then the translation of a new set of characters begins. A more detailed description of the translator's architecture can be found in [13].

2.4. Hardware implementation and test

This design has been implemented in VHDL and tested in a Xilinx Spartan-3E FPGA with a 50MHz clock [14]. In this particular chip the translator occupies 1333 out of 4656 programmable slices, which is less than 30% of the FPGA programmable logic. Implementation and test were carried out using Xilinx's ISE suite, version 8.2.

To test the system a simple Universal Asynchronous Receiver-Transmitter (UART) was integrated in the FPGA to communicate with a PC using an RS-232 port. Text files were sent to the translator using this channel while translated Braille codes were sent back to the PC.

During testing, outputs of the IP were compared against the outputs of a commercial Braille translation program running in a 16 MHz Mitsubishi microcontroller. Results show that the hardware translator can achieve the same accuracy as the commercial system but the speed is considerable faster. For example, to translate the word "and" into the corresponding Computer Braille character '&' the microcontroller takes 300us, and the FPGA only 12us.

To simplify comparisons, the FPGA used the same 16 MHz clock as the microcontroller; however, during normal operation the FPGA can use a clock as fast as100 MHz. More details on verification can be found in [13].

3. Braille keyboard and its controller

3.1. Braille keyboard

A Braille keyboard is a device for the blind to type Braille characters. Braille keyboards are more complicated than conventional keyboards because up to six keys could be pressed at the same time. Some designs use optical sensors [8, 9].

For the CUB, a low cost 6 x 4 push button matrix keyboard was developed. The six keys used to type Braille dots are allocated to the first column. Keys in the second and third columns are used to indicate special functions. The fourth column contains keys for ENTER, SPACE and arrows (Left, Right, Up, Down).

Figure 3 shows the electrical diagram of Curtin's Braille keyboard. Inputs (columns) receive a 4-bit scan code containing only one zero. When a key is pressed the zero propagates to the corresponding output (row). Individual keys can be identified by combining the input and output codes.

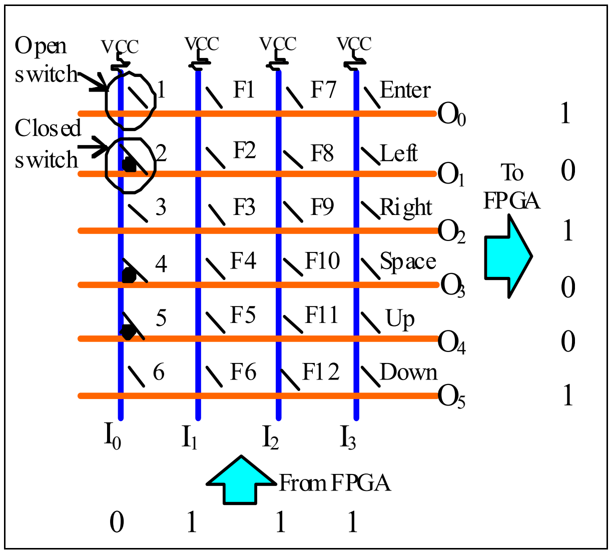


Figure 3. Matrix Keyboard Scheme

3.2. Keyboard controller

The keyboard controller consists of two parts: A Code Scanner and a Decoder. The Code Scanner sends 4-bit scan codes to inputs I_0 to I_3 of the keyboard and receives 6-bit codes from outputs O_0 to O_5. The central element of this mechanism is a circular shift register initialised with "0111". The Code Scanner module is driven by the state machine shown in Figure 4.

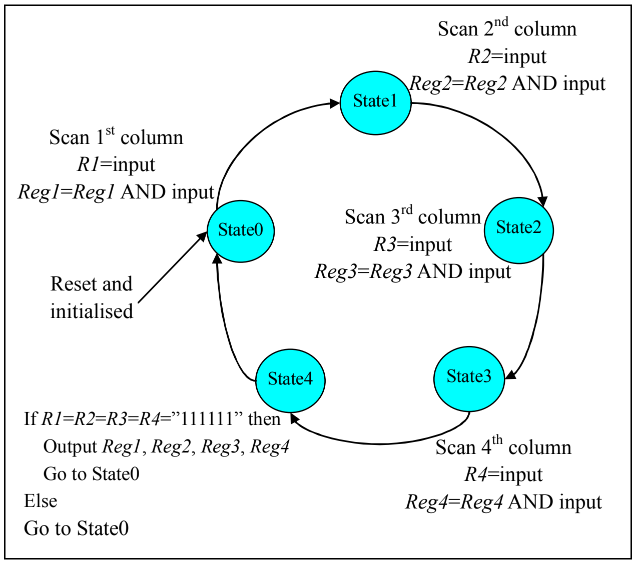


Figure 4. State Machine in Code Scanner

The state machine driving the keyboard controller works as follows:

- In State 0, code "0111" is presented in I_0-I_3 providing a zero to the first column. Two 6-bit registers called **R1** and **Reg1** are used to store input signals and both are initialised to all ones. After a debouncing period of about 13 milliseconds, the six bits input codes are saved into register **R1**. Meanwhile, **Reg1** stores the logic AND operation between the input and its current value to detect if multiple keys have been pressed in the same column at different times.
- In the next three states the same process is repeated to scan the next three columns using codes "1011", "1101", and "1110" respectively. Likewise, in each state, two registers are used to store input codes and the result of the AND operation. There are in total two groups of 4 registers [**R1**, **R2**, **R3**, **R4**] and [**Reg1**, **Reg2, Reg3**, **Reg4**].
- In the State 4, the registers R1, R2, R3 and R4 are checked to see if the value for each register is all ones, indicating that all keys have been released. If they have not, the state machine will go back to the State 0 to repeat the operation. If all keys have been released, the 24 bits stored in registers Reg1 to Reg4 are sent to the decoder, and the registers are set to all ones.

When keys have been pressed and released the decoder receives 24-bit codes from the code scanner and translates them into the corresponding Computer Braille or control code. The decoder sends its output to the serial transmitter or Braille-to-Text translator depending on the function that was selected.

3.3. Implementation and test

The keyboard and its controller were also tested in the Spartan 3 development board. The test system including the controller and a serial asynchronous transmitter occupy 137 out of 3584 programmable slices, which amounts to 3% of the programmable resources. Computer Braille codes generated by the controller are sent through the transmitter to a personal computer and displayed in a window using HyperTerminal. Results showed that the Braille keyboard and its controller are reliable and because they are based on a simple printed circuit board (PCB), they are more affordable and easy to manufacture than complicated mechanical or optical alternatives. More details on the keyboard can be found in [15].

4. The embedded system

4.1. Hardware development

The embedded system implementing the Braille PDA consists of a MicroBlaze microcontroller and the hardware modules described in previous sections. Xilinx's Platform Studio (XPS) suite was used to develop the embedded system's hardware and software components.

XPS uses IBM's CoreConnect bus to implement embedded systems [2]. CoreConnect allows the interconnection of devices with different data widths using an On-Chip Peripheral Bus (OPB).

After all hardware modules have been integrated, XPS creates a software library for the whole system and drivers for each one of the hardware components. These drivers supply basic functions to read from and write to every module. If the system is specified as interrupt-driven, XPS also generates primitive interrupt service routines (ISR).

Figure 5 shows the block diagram of an embedded system that integrates a MicroBlaze with the hardware translator and the Braille keyboard controller described in previous sections.

The translator receives ASCII codes from MicroBlaze using an 8-bit register connected to the OPB bus. Results of the translation, which have a maximum of 12 characters (96 bits), are saved in three 32-bit registers which can be accessed by MicroBlaze in 3 clock cycles.

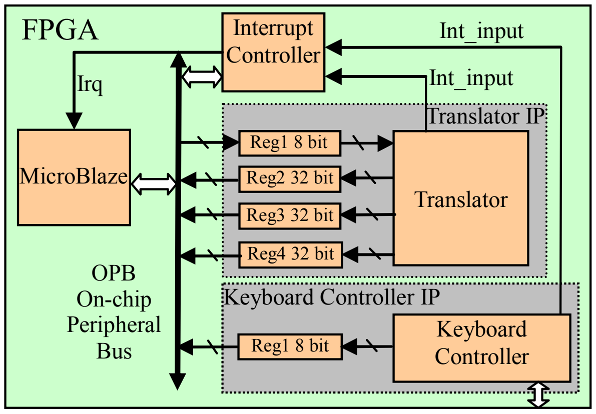


Figure 5. Interconnections between IPs and MicroBlaze

The keyboard controller connects to the keyboard's inputs and outputs via FPGA's pins. When keys are pressed and released, the controller sends Computer Braille codes to the MicroBlaze using an 8-bit register connected to the OPB.

Another feature of this system is the use of an interrupt controller (IC). MicroBlaze can handle only one interrupt source and in order to use multiple interrupts, an interrupt controller connected to the OPB is necessary [16].

In Figure 5, each block of the system is driven by an on-board 50MHz clock source. The IC receives two interrupts associated to the data ready signals from the translator and keyboard controller. If both interrupts arrive at the same time, the keyboard controller has higher priority than the translator. Interrupt signals are active high for one clock cycle when data from either module are ready to be read. After solving priorities, the IC will interrupt the MicroBlaze which will then execute interrupt service routines according to interrupt vectors provided by the IC.

To achieve the reconfigurability shown in figure 1 a microcontroller, FPGA or CPLD will hook onto the instruction bus of the system and when a particular hardware function is required, the corresponding IP will be loaded. This is possible thanks to a property called dynamic partial reconfiguration available in some modern FPGAs [17]. At the present stage of the project partial reconfiguration has not been attempted yet.

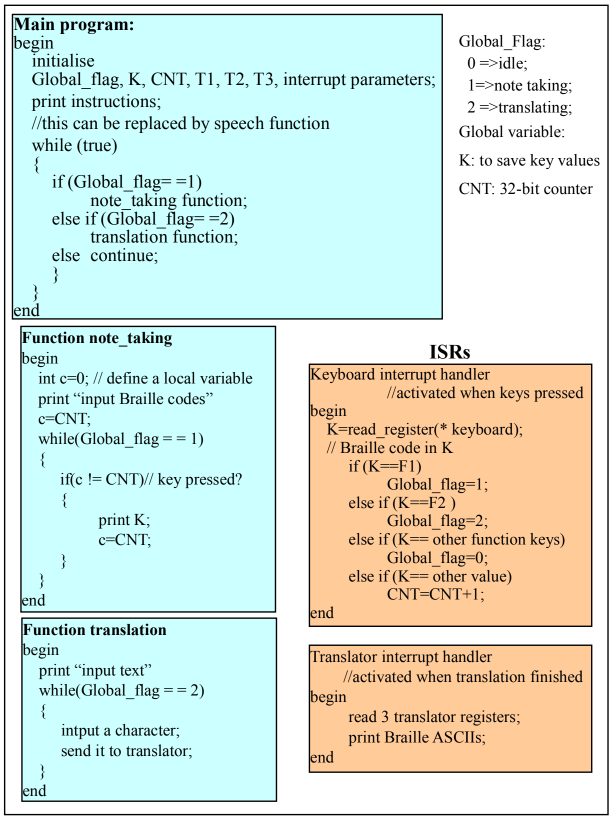
4.2. Software development

Xilinx's software development tools support C and assembly languages. Users only have to write code to implement the high-level function of the system because all basic functions are available in libraries generated automatically during synthesis of the embedded system. Since print-to-Braille translation and keyboard control were implemented in hardware, the corresponding ISRs were very simple.

Figure 6 shows pseudo-code for a simple application implementing some of the CUB functionality. The main program runs an infinite loop executing the function selected by a global flag. The value of the flag changes according to the function selected using the Braille keyboard. For the tests reported in this paper only two functions were available: note-taking and print-to-Braille translation.

In note-taking mode the CUB reads the Braille keyboard and sends the corresponding ASCII code to a monitor using an RS-232 connection. The keyboard ISR is invoked when one or more keys on the Braille keyboard have been pressed and released. If the key was a command, then the keyboard controller will issue a code indicating what task to carry out next.

In print-to-Braille translation mode the CUB receives ASCII characters form a PC and returns the corresponding Grade 2, Computer Braille codes. The translator ISR is invoked when the translation of a word has finished.



```
Main program:
begin
   initialise
   Global_flag, K, CNT, T1, T2, T3, interrupt parameters;
   print instructions;
   //this can be replaced by speech function
   while (true)
   {
      if (Global_flag= =1)
            note_taking function;
      else if (Global_flag= =2)
            translation function;
      else  continue;
      }
   }
end
```

```
Global_Flag:
  0 =>idle;
  1=>note taking;
  2 =>translating;
Global variable:
K: to save key values
CNT: 32-bit counter
```

```
Function note_taking
begin
   int c=0; // define a local variable
   print "input Braille codes"
   c=CNT;
   while(Global_flag = = 1)
   {
      if(c != CNT)// key pressed?
      {
            print K;
            c=CNT;
      }
   }
end
```

```
Function translation
begin
   print "input text"
   while(Global_flag = = 2)
   {
      intput a character;
      send it to translator;
   }
end
```

ISRs

```
Keyboard interrupt handler
            //activated when keys pressed
begin
   K=read_register(* keyboard);
   // Braille code in K
      if (K==F1)
            Global_flag=1;
      else if (K==F2 )
            Global_flag=2;
      else if (K== other function keys)
            Global_flag=0;
      else if (K== other value)
            CNT=CNT+1;
end
```

```
Translator interrupt handler
      //activated when translation finished
begin
      read 3 translator registers;
      print Braille ASCIIs;
end
```

Figure 6. Structure of Software Implementation

5. Conclusions and future work

This paper presented the implementation and test of an embedded system that includes a print-to-Braille translator and a Braille keyboard. The system was developed and tested using a Xilinx Spartan 3 development board. The print-to-Braille translator is able to achieve the same accuracy as commercial software with much higher throughput. The translation IP liberates the processor from the heavy burden of running this computationally expensive algorithm, and at the same time simplifies software development. Furthermore, Blenkhorn's algorithm was designed so that Braille translation in languages other than English can be achieved by simply changing the rule table. In the CUB this table is stored in an external memory module, which adds another level of reconfigurability to the system.

The proposed Braille keyboard and its controller are a low cost solution that could be part of a commercial Braille personal digital assistant (PDA).

An important feature of the system is that it is interrupt-driven, which simplifies the development of software if there is need to include additional PDA functions. It is expected that some of these new functions will be implemented in hardware liberating microcontroller-power for other high-level applications such as an embedded real-time operating system.

The microcontroller-based version of CUB is now in the industrial prototype stage. The FPGA implementation is being explored to reduce cost and augment functionality in the future. The development of the FPGA-based CUB will span several stages. At the current stage the FPGA is connected as a peripheral to the microcontroller. Individual IP modules are being developed to implement in hardware functions that currently run in software. Work is being carried out to include features like voice-recognition, text-to-voice synthesis and Braille-to-Text translation. The next stage will be to incorporate dynamic reconfigurability into the system. This will require major modifications to the hardware and software architectures. It is expected that new challenges will have to be faced; for example, as more functions are incorporated into the system, the capacity of the chip may become an issue.

At this point in time minimizing cost is not a priority because new devices with more capabilities, at lower costs keep appearing every semester. However, it is expected that the actual cost of the final version of the FPGA-based CUB will be dominated by the cost of the FPGA itself.

References

[1] Curtin University, CUB web page. http://bauhaus.ece.curtin.edu.au/~iain/CUB/

[2] Xilinx, "Processor IP Reference Guide", Feb. 2005. http://www.xilinx.com/support/documentation/sw_manuals/XPS71i_proc_ip_ref_guide.pdf

[3] K. R. Ingham, "Braille, the Language, Its Machine Translation and Display", IEEE Transactions on Publication, Vol. 10, no. 4, Dec. 1969, pp. 96-100.

[4] Sullivan J. E., "DOTSYS III: A Portable Braille Translator", in proceedings of the ACM annual conference, issue 15, New York, Mar. 1975, pp. 14-19.

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

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

[7] Duxbury Braille Translator web page. http://www.duxburysystems.com

[8] Spragg J., "Interfacing a Perkins Brailler to a BBC micro," Microp. Microsystems, Vol. 8, pp. 524-527, 1984.

[9] Evans D.G., Pettitt S. and Blenkhorn, P., "A Modified Perkins Brailler for Text Entry into Windows Applications", IEEE Transactions on Rehabilitation Engineering, Vol. 10, no. 3, Sep. 2002, pp. 204-206.

[10] Slaby W. A., "The Markov System of Production Rules: A Universal Braille Translator", ACM SIGCAPH computers and the physically handicapped, no. 15, 1975, pp. 53-59.

[11] Slaby W. A., "Computerized Braille Translation", journal of microcomputer applications, Vol. 13, no. 2, 1990, pp. 107-113.

[12] Blenkhorn P., "A System For Converting Print into Braille", IEEE transactions on rehabilitation engineering, Vol. 5, no. 2, Jun. 1997, pp. 121-129.

[13] Zhang X., Ortega-Sanchez C. and Murray I. "A System For Fast Text-To-Braille Translation Based On FPGAs", 3rd Southern Conference on Programmable Logic, Mar del Plata, Argentina, 26-28 February, 2007.

[14] Xilinx, "Spartan-3E Starter Kit Board User Guide", Mar. 2006. http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[15] Zhang X., Ortega-Sanchez C. and Murray I. "A Hardware-Based Braille Note Taker", 3rd Southern Conference on Programmable Logic, Mar del Plata, Argentina, 26-28 February, 2007

[16] Xilinx, "Using and Creating Interrupt Based Systems", Jan. 2005. http://www.xilinx.com/support/documentation/application_notes/xapp778.pdf

[17] Xilinx, "Managing Partial Dynamic Reconfiguration in Virtex-II Pro FPGAs", Xcell Journal, Fall 2004.