Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 15640, Pages 1–9
DOI 10.1155/ASP/2006/15640

FPGA Prototyping of RNN Decoder for Convolutional Codes

Zoran Salcic, Stevan Berber, and Paul Secker

Department of Electrical and Electronic Engineering, the University of Auckland, 38 Princess Street, Auckland 1020, New Zealand

Received 30 May 2005; Revised 29 November 2005; Accepted 21 January 2006

Recommended for Publication by Roger Woods

This paper presents prototyping of a recurrent type neural network (RNN) convolutional decoder using system-level design specification and design flow that enables easy mapping to the target FPGA architecture. Implementation and the performance measurement results have shown that an RNN decoder for hard-decision decoding coupled with a simple hard-limiting neuron activation function results in a very low complexity, which easily fits into standard Altera FPGA. Moreover, the design methodology allowed modeling of complete testbed for prototyping RNN decoders in simulation and real-time environment (same FPGA), thus enabling evaluation of BER performance characteristics of the decoder for various conditions of communication channel in real time.

1. INTRODUCTION

Recurrent type neural networks (RNN) have been successfully used in various fields of digital communications primarily due to their nonlinear processing, possible parallel processing that could accommodate recent requirements for high-speed signal transmission and, also, expected efficient hardware implementations [1]. In the past several years substantial efforts have been made to apply RNNs in error control coding theory. Initially, these networks were applied for block codes decoding [2, 3] and then for convolutional [4–7] and turbo codes decoding [8]. In [5–7], it was shown that the decoding problem could be formulated as a function minimization problem and the gradient descent algorithm was applied to decode convolutional codes of a small code rate, and the developed recurrent artificial neural network (ANN) algorithm did not need any supervision. That algorithm was later implemented in hardware using floating-gate MOSFET circuits [9].

Theoretical base for the decoding of generalized convolutional codes of rate $1/n$ was developed and reported in [1, 10]. Simulation results have shown that the RNN decoder can in fact match the performance of the Viterbi decoder when certain operating parameters are adopted. Simulations have also revealed that the RNN decoder performs very well for some convolutional codes without using the complicated simulated annealing (SA) technique required by other codes. However, for the RNN decoder to be of any real practical use, it must have a hardware realization that offers some benefits in terms of decoding speed, ease of implementation, or hardware complexity. The hardware implementation of artificial neural networks has been an active area of research. As techniques for implementing neural networks evolve, the RNN decoder, which has already shown to be competitive at an algorithmic level, may become a viable option in practical implementations. This motivated us to investigate possibilities of the practical HW implementation of the decoding algorithm based on RNN application using FPGA technology.

In this paper we investigate hardware implementation of the RNN decoder using readily available hardware design methods and target technologies. An obvious choice of target technology is FPGAs, due to being capable of exploiting the parallelism inherent to the RNN decoder, but also for rapid prototyping and analysis of implementation options.

1.1. *FPGA implementation of ANNs*

The biologically inspired neural models generally rely on massive parallel computation. Thus the high-speed operation in real-time applications can be achieved only if the networks are implemented using parallel hardware architectures [11].

FPGAs have been used for ANN implementation due to accessibility, ease of fast reprogramming, and low cost, permitting the fast and nonexpensive implementation of the whole system [12]. In addition, FPGA-based ANNs can be tailored to specific ANN configurations; there is no need for worst-case fully interconnected designs as in full-custom

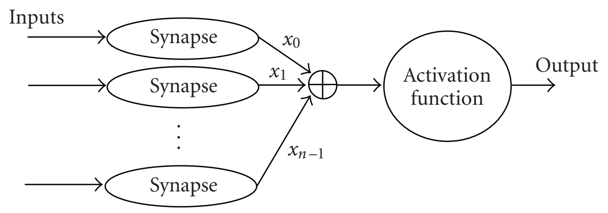

Figure 1: General neuron structure.

VLSI [13]. For hardware implementation it is considered important to separate the learning and retrieval phase of an ANN. However, this technique is not directly applicable to the RNN decoder, as it in fact does not require training as such. However, its implementation is essentially an implementation of a learning algorithm (gradient descent).

In general, all ANN architectures consist of a set of inputs and interconnected neurons with the neuron's structure as in Figure 1. The neuron can be considered the basic processing element, and its design determines the complexity of the network. The neuron consists of three main elements: the synaptic connections, the adder, and the activation function. The fundamental problem limiting the size of FPGA-based ANNs is the cost of implementing the multiplications associated with the synaptic connections because fully parallel ANNs require a large number of multipliers. Although prototyping itself can be accomplished using FPGAs which offer high number of multipliers, the overall goal of the RNN decoder design is to use as few resources as possible as the decoder is usually only a part of a bigger system. Practical ANN implementations are accomplished either by reducing the number of multipliers or by reducing the complexity of the multiplier. One way of reducing the number of multipliers is to share a single multiplier across all neuron inputs [14]. In [13, 15] another method of reducing the circuitry necessary for multiplication is proposed which is based on bit-serial *stochastic* computing techniques. A successful prototyping of a neuro-adaptive smart antenna beam-forming algorithm using combined hardware-software implemented radial basis function (RBF) neural network has been reported in [16].

In the neuron from Figure 1, the complexity of the adder depends on the precision of the inputs from the synapses and on the number of inputs to each neuron. The adders may be shared across inputs with intermediate results being stored in an accumulator. Of particular importance is the hardware implementation of the neuron activation function. The sigmoid function, traditionally used in ANNs, is not suitable for direct digital implementation as it consists of an infinite exponential series [8, 17]. Thus most implementations resort to various methods of approximating the sigmoid function in hardware typically by using lookup tables (LUTs) to store samples of the sigmoid function for approximation, with some examples of this technique reported in [11, 13]. However, the amount of hardware required for these tables can be quite large, especially if one requires a reasonable approximation. Other implementations use adders, shift registers, and multipliers to realise a digital approximation of the sigmoid function. In [17] a second-order nonlinear function was used to approximate the sigmoid and was implemented directly using digital components. Also in [18] a piecewise linear approximation of the *tanh* function was implemented for the neuron activation function. A coarse approximation of the sigmoid function is the threshold (hard-limiting) function, as used in [19, 20].

1.2. RNN decoder design objectives and methodology

Our approach to RNN decoder implementation, based on its model presented in [10], was to evaluate the design on an example case decoder in order to identify the issues involved with a hardware implementation of the decoders of any complexity with the following main goals:

(i) to investigate how decoder functionality can be simplified and practically implemented on an FPGA device;
(ii) to evaluate decoder performance degradation imposed by the limited bit resolution and fixed-point arithmetic, compared to original model implemented in MatLab [10];
(iii) to evaluate the complexity and decoding speed of the simple case hardware realised RNN decoder and subsequently estimate the complexity of more powerful RNN decoders suitable for industrial use.

An additional goal was to evaluate the suitability of a high-level FPGA design methodology for use with ANN-type prototyping. The FPGA design flow used Altera's *DSP Builder* software integrated into Mathwork's "*Simulink-to-Algorithm*" development environment [21, 22] as the starting specification. This also enabled design of a testbed in the form of the whole communication system model to evaluate decoder performance in realistic conditions at Simulink level first and then, after synthesis, at a very high speed where the decoder was implemented in FPGA.

2. SYSTEM OVERVIEW

The design of an RNN decoder [10] first requires the specification of a number of system characteristics. Of most importance was the choice of convolutional code, as this affects the physical bitrate of the system and determines the structure and required operation of the RNN decoder. In accordance with the simple case design philosophy, the code of choice is specified by the generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \qquad (1)$$

Matrix G specifies the feedbacks in the encoder structure and is related to theoretical derivative of the noise energy function as explained later. The choice of this code is based on the resulting structure of the RNN decoder. Firstly, simulations showed that this simple code does not require SA to perform very close to the equivalent Viterbi decoder [23]. This eliminates the need for the implementation of an independent noise source to each neuron with decreasing noise

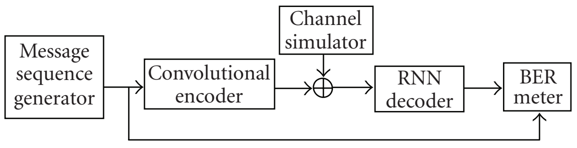

FIGURE 2: Testbed system model.

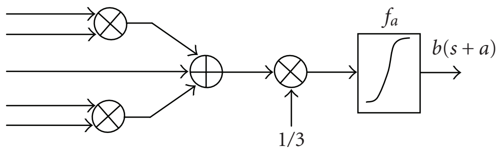

FIGURE 3: Newtron model.

variance (for codes that require SA, some VLSI-efficient techniques for generating multiple uncorrelated noise sources are provided in [24, 25]). Secondly, the resulting neuron structure for this code is very simple with the expression for the gradient update term consisting of a minimal number of arithmetic operations. This particular code provides a coding gain of approximately 3 dB at the BPSK theoretical probability of error of 10^{-4}. Hard-decision decoding is also chosen to minimise the precision required by the RNN decoder. Thus the decoder inputs are of two-level (binary) precision. This also permits the use of pseudorandom binary sequence (PRBS) generators for the hardware implementation of a channel simulator.

In order to verify the correct operation of the designed RNN decoder throughout the design process, a number of additional components were implemented which collectively make up a testbed effectively representing the basic communication system. The system-level model is shown in Figure 2.

3. DECODER AND TESTBED DESIGN

In this section we first describe selected RNN decoder implementation, including some design tradeoffs, and then the remaining components of the testbed as described in Figure 2.

3.1. *RNN decoder*

A communication system that contains a rate $1/n$ convolutional encoder, which generates a set of encoded n bits for each message bit at the input to encoder at the discrete time instant s, is shown, analyzed, and theoretically described in [26]. The encoder is defined by its constraint length L and a logic circuit that defines the mapping of the input bits into the code word at the output.

The noise energy function $f(B)$ is expressed as a function of the received and message bits and the decoding problem is defined as a noise energy function $f(B)$ minimization problem [10]. The update rule is expressed as

$$b(s+a)^{(p+1)} = b(s+a)^{(p)} - \alpha \frac{\partial f(B)}{\partial b(s+a)}\bigg|_{p}, \quad \text{for a} = 0, 1, \ldots, T, \tag{2}$$

where α is a gradient update factor or a learning rate factor and can be chosen to eliminate self-feedback in the RNN [10], or in the form

$$b(s+a)^{p+1} = f_a\left(\frac{1}{\sum g}\sum_{k=0}^{L}\sum_{j=1}^{n} g_{j,k} \times \left[r_j(s+a+k-1)\cdot\prod_{i=1,\, i\neq k}^{L} b(s+a+k-i)^{g_{j,i}}\right]\right). \tag{3}$$

Applying this rule, the neuron update expression for the convolutional code specified by (1) is given by

$$b(s+a)^{(p+1)} = f_a\left(\frac{1}{3}\left[r_1(s+a)b(s+a-2)^p + r_2(s+a+1) + r_1(s+a+2)b(s+a+2)^p\right]\right), \tag{4}$$

where term 1/3 comes out from the fact that the gradient update factor is chosen to eliminate the self-feedback of the neural network.

The learning rate, α, is chosen to eliminate self-feedback in the network, and the f_α term represents the neuron activation function. The structure of the corresponding neuron is shown in Figure 3. In this case the multiplication by the 1/3 term simplifies the neuron model because the previous value has no influence estimating the current value of the information bit. The RNN decoder, which is constructed as an array of $T+1$ neurons (16 for this code), is shown in Figure 4.

To simplify the hardware implementation, a hard-limiting (HL) activation function is used and the final implemented neuron model is shown in Figure 5. The multiplication by the 1/3 term is removed since this gain factor has no effect when the HL activation function is used.

Implementation of the neuron requires two multipliers, an adder, and a realization of the HL activation function. Previous derivations involving the RNN decoder structure have been based on the assumption that signal levels in the network are bipolar ($b(s) \in \{+1, -1\}$). However for the hardware implementation we simply map the bipolar logic back to binary signal levels using the mapping of (3):

$$b(s) = \begin{cases} +1 \longmapsto 0, \\ -1 \longmapsto 1. \end{cases} \tag{5}$$

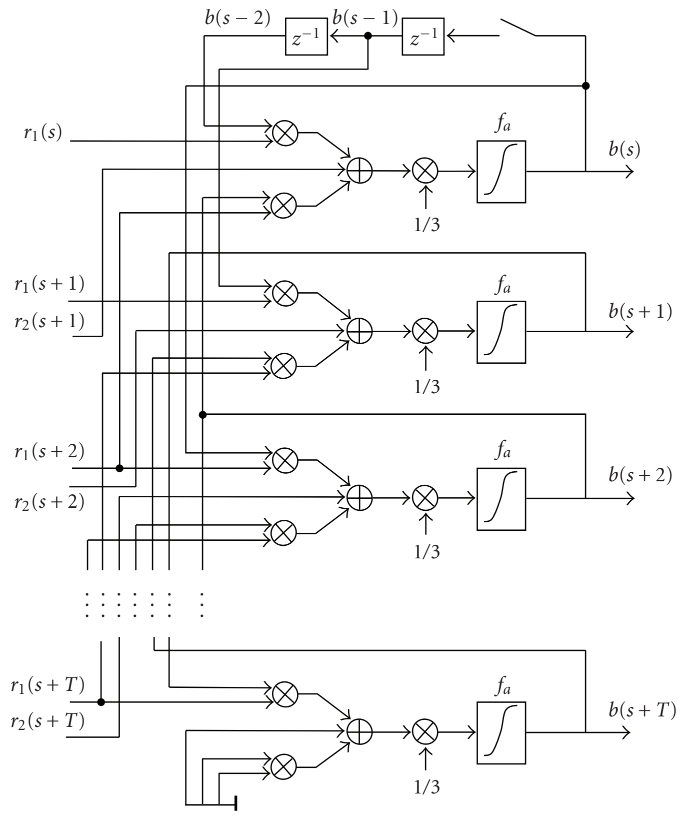

Figure 4: RNN decoder model.

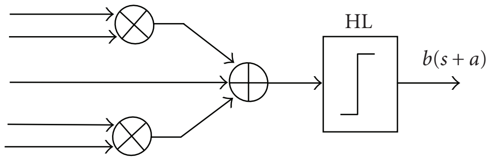

Figure 5: Neuron with HL activation function.

Table 1: Mapping from bioplar to binary for neuron multiplication.

a	b	$a \bullet b$		a	b	$a \oplus b$
+1	+1	+1		0	0	0
−1	+1	−1	$\Longrightarrow$	1	0	1
+1	−1	−1		0	1	1
−1	−1	+1		1	1	0

All the inputs to the neuron are binary signals, due to the hard-decision channel outputs and the HL activation function. Thus, according to Table 1, the multiplication can be performed using simple XOR logic gates.

The 3-way adder can be combined with the HL activation into a single logic truth-table, which is shown in Table 2 for the bipolar signal case and subsequently converted to the hardware binary case. It is implemented using simple logic gates and the resulting digital neuron design is shown in Figure 6.

The Simulink model of the RNN decoder incorporates an array of 16 neurons each connected to each other according to (4) for a = $\{0, 1, 2, \dots, 15\}$. The received 2-bit symbols (channel output) are shifted into the array of 2-bit shift registers. The operation of the RNN decoder begins with the initialization of the receive bit registers and neuron registers to zero. A clock signal is provided by the control unit to each neuron register for the update of the neuron states on each iteration. The neurons are actually updated on only nine of the ten available clock cycles. On the tenth clock the neuron states are cleared for the next set of iterations and the chains of receive bit registers are clocked in order to shift the next 2-bit noisy code word into the network. At this same time a bit is shifted out of the RNN decoder, which represents the decoded bit estimate. Two single bit registers are used to store the two most recent past bit decisions, which are required as inputs to the upper neurons in the network.

3.2. Other system components

3.2.1. Source generator

The requirement at the transmitter is to have information bits generated with equal probability that are suitably

Table 2: Mapping from bioplar to binary for neuron adder+HL.

a	b	c	$a \dashv b \dashv c$	HL
+1	+1	+1	3	+1
+1	+1	−1	+1	+1
+1	−1	+1	+1	+1
+1	−1	−1	−1	−1
−1	+1	+1	+1	+1
−1	−1	−1	−1	−1
−1	+1	+1	−1	−1
−1	−1	−1	−3	−1

	a	b	c	HL
	0	0	0	0
	0	0	1	0
	0	1	0	0
$\Longrightarrow$	0	1	1	1
	1	0	0	0
	1	0	1	1
	1	1	0	1
	1	1	1	1

uncorrelated and have a random statistical nature. A PRBS is a semirandom sequence in that it appears random within the sequence length, providing an adequate level of "randomness," but the entire sequence repeats indefinitely. A PRBS can be generated using a linear feedback shift register (LFSR). As the shift-register length m is increased, the statistical properties of the maximal length sequence becomes increasingly similar to a truly random binary sequence [27]. An LSFR with 18 stages was chosen which generates a suitably long m-sequence of length $2^{18} - 1 = 262143$ and only requires a single XOR gate for implementation. The LFSR can be represented by the generator polynomial:

$$G(X) = 1 + X^{11} + X^{18}. \tag{6}$$

3.2.2. Convolutional encoder

The Simulink implementation of convolutional encoder, which subsequently was synthesized into FPGA, specified by (1) is shown in Figure 7. The encoder comprises two single bit shift registers and an XOR gate performing modulo-2 addition. The two encoder outputs are multiplexed onto a single bit bus which is clocked at twice the rate of the shift registers. The multiplexer select signal is provided by the control unit.

3.2.3. Binary symmetric channel

The binary symmetric channel (BSC) model requires generation of a PRBS with a specified probability of error, P_e, corresponding to a channel Eb/No. Such a PRBS generator will be hereon referred to as a PRBS-PE generator, which is illustrated in Figure 8. The LFSR used in the source generator model described in section generates a PRBS with probability of error of 0.5. It is possible to generate sequences of varying error probability by comparing the state value of the LFSR at each shift with a register holding a fixed value (corresponding to the desired P_e), and generating a 1 if the LFSR is less than or equal to the compare register value, or a 0 if not.

The LFSR and compare register lengths must be chosen large enough to allow sufficient P_e resolution

$$P_e^{\text{res}} = \frac{1}{2^m - 1}. \tag{7}$$

The LFSR of length $m = 18$ gives a P_e resolution of $3.814 \bullet 10^{-6}$, which is considered sufficient for the channel environments considered here. For this resolution the highest channel Eb/N_o we can simulate for (excluding the zero-noise case) is calculated to have a maximum $Eb/N_0 \sim 10\,\text{dB}$ which is sufficient for our testing purposes. In [28] it was shown that the output of the PRBS-PE generator described is correlated since each output sample is based on m-1 bits of the LFSR from the previous shift. It was found by simulation that 11 shifts of the LFSR per clock were sufficient to reduce the autocorrelation statistics of the PRBS-PE sequence in order to make the output more random [26].

3.2.4. BER counter

To complete the system model, a module for calculating the transmission bit-error-rate is included. The BER counter model uses two 25-bit counters. The first counts the total number of bits and runs at the information bitrate (1/20 times the system clock rate). The second counter counts the number of bit errors and is clocked each time an assertion signal is given from the output of a comparator, which compares the output of the source generator with the output of the RNN decoder. The source generator signal is delayed by 18 clock cycles for synchronisation with the decoded bit stream, due to the delays introduced by the transmission.

3.2.5. Control unit

The control unit, which is not shown in Figure 2, provides the necessary clock signals to the registered devices within the design. Three different clock rates are required within the design, all derived from the same system clock. The fastest rate clock signal is required by the neurons within the RNN decoder. On each of these clocks the neuron states are updated, which correspond to one iteration of the network. These can be clocked at the system clock rate, however a more conservative approach was taken which clocks the neurons at half the system clock rate. Ten iterations of the network are used for decoding each message bit, making the information bitrate of the system 1/20 times the system clock rate. The source generator, the convolutional encoder and the BER calculator modules are all clocked at this rate. Since the convolutional encoder is of rate 1/2, the symbol rate is twice that of the information bitrate. Thus a clock signal of one tenth the system clock rate is used to clock the convolutional encoder output, the BSC channel module, and parts of the RNN decoder circuitry.

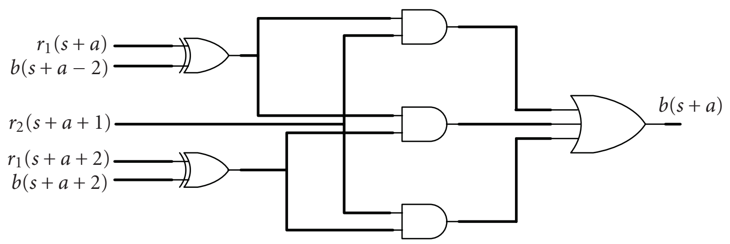

FIGURE 6: Final digital implementation of neutron.

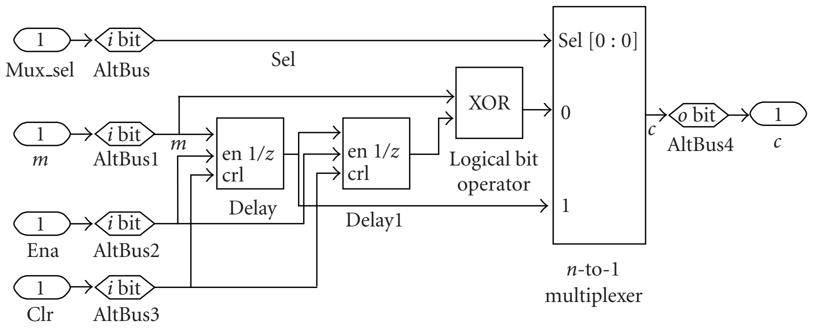

FIGURE 7: Simulink convolutional encoder model.

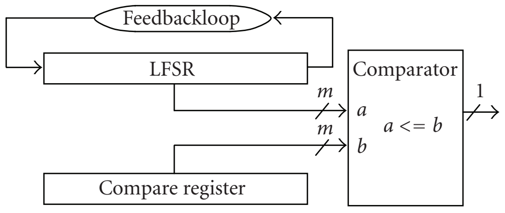

FIGURE 8: PRBS-PE block diagram.

TABLE 3: Simulated BERs for the Simulink RNN decoder implementation.

E_b/N_0	P_e	CMA-REG	Simulink BER
0	0.0786	20617	0.0631
1	0.0563	14574	0.0333
2	0.0375	9832	0.0175
3	0.0229	5997	0.0083
4	0.0125	3277	0.0037

4. PERFORMANCE ANALYSIS

In this section we show that the performance of the hardware RNN decoder matches that predicted by the software simulator for the specific network configuration. Functional simulation results of the RNN decoder are also analysed, along with postsynthesis simulation results.

The described Simulink model was tested for a number of channel Eb/N_o levels, as shown in Table 3. The table shows the theoretical no-coding probability of error, and the value of the channel compare register used in each case.

Figure 9 shows the BER performance of the MATLAB software model versus the Simulink-implemented hardware model for the given code and RNN configuration. The hardware RNN uses parallel update with nine iterations per bit and hard-limited activation function. The performance of the hardware model is equivalent to that predicted by the MATLAB software simulator, which verifies that the hardware implementation works correctly. The performance difference between the Viterbi decoder and the RNN decoder is due to the type of activation function employed. In this case the HL activation function was used to reduce the implementation complexity considerably.

Following the functional system verification, the postsynthesis timing verification was performed to prove the RNN decoder still operates correctly when register-to-register propagation delays are imposed. The Altera EP20K30-EQC208-1 device was targeted for synthesis, which is the smallest device in the APEX 20 K family. The RNN decoder uses only 63 logic elements and can run at the maximum operating clock frequency of 107.1 MHz. With 20 system clock cycles required for decoding each information bit, the maximum data throughput of the RNN decoder is 5.36 Mbps. If a less conservative clocking approach was taken, where the neurons are updated on every clock cycle (rather than every second), then the data throughput is doubled.

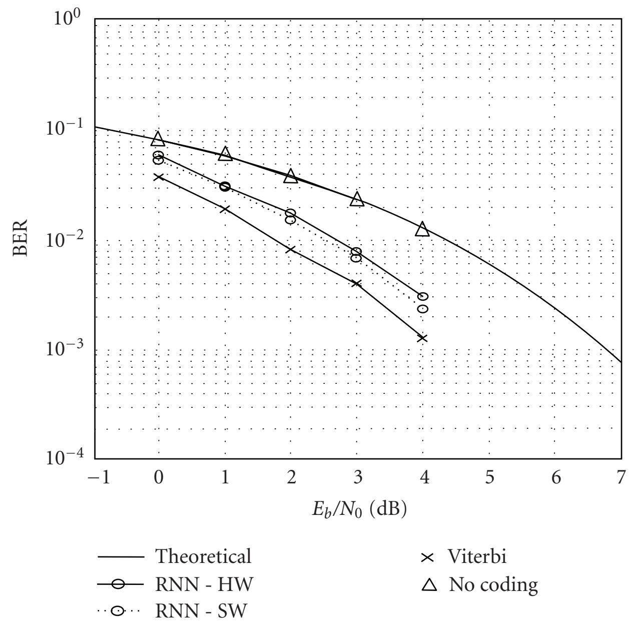

FIGURE 9: BER performance for both software and hardware models.

5. EVALUATION OF THE RNN DECODER HARDWARE COMPLEXITY

The regular structure of the RNN decoder is very beneficial when considering the time requirements for the development, and the operation of the decoder is also relatively straightforward. Furthermore, the neurons are only locally connected in the sense that each neuron is physically connected to several neurons within its vicinity, and not to all neurons in the network. This is likely to reduce routing problems often encountered in ANN hardware implementations. The specific RNN decoder implemented here is unlikely to be used in an industrial channel coding application except in the case when a simple algorithm and small power consumption of the circuits are strongly required. For this reason we investigated the complexity requirements of a more powerful RNN decoder implementation.

Industrial applications of convolutional coding currently employ codes of constraint length up to 7 or 9. These codes offer superior error correcting capacity over shorter codes, however generally result in complex decoder structures. The complexity of the RNN decoder for these longer codes can be estimated based on the computational requirements of the neuron update equation. The number of multiplications required per neuron update is calculated as

$$N_{\text{mul/neuron}} = 1 + \sum_{k=1}^{L}\sum_{j=1}^{n}\left[\left(g_{j,k}\right)\sum_{i=1,i\neq k}^{l}\left(g_{j,i}\right)\right] \tag{8}$$

and the number of additions as

$$N_{\text{add/neuron}} = \left[\sum_{k=1}^{L}\sum_{j=1}^{n}\left(g_{j,k}\right)\right] - 1. \tag{9}$$

An industry standard convolutional code of constraint length $L = 7$, as defined by the IEEE 802.11a wireless LAN standard [29], is shown in (10),

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{10}$$

In this case a fully parallel implementation of the neuron would require 41 multipliers and a 10-way adder. Note that if hard-decision decoding is used with an HL neuron activation function, these multipliers can be implemented with XOR gate structures. The simple RNN decoder implementation previously showed that the HL activation function decreased the performance of the decoder. To overcome this, an 8-level (3-bit) approximation of the sigmoid activation function can be implemented using an LUT in the FPGA device. For signal width consistency this strategy could be coupled with a 3-bit soft-decision decoding strategy, thus all the multipliers in the decoder would need to multiply 3-bit operands.

The $L = 7$ code described above requires 35 neurons for adequate decoding performance. In this case the total number of multipliers required equates to $41 * 35 = 1435$, which is likely to consume a very large amount of logic resources, especially if the multipliers have operand widths of greater than 1 bit. The amount of logic could be reduced considerably if a fewer number of neurons are implemented and a sequential neuron update strategy is adopted where a smaller number of neurons are time-division-multiplexed across the network. In fact, simulation results show [16] that the sequential update variation offers improved decoding performance over the fully parallel decoding strategy.

It was found that for a fixed number of encoder outputs, n, both the number of addition and multiplication operations required per iteration can at worst increase polynomially with the encoder constraint length, L, [10]. Also, with a fixed value of L, the number of addition and multiplication operations required per iteration can at worst increase linearly with n. However the Viterbi decoder complexity (for both memory requirements and computations) increases exponentially with the constraint length of the code L. Thus, the improved complexity trend of the RNN decoder over the Viterbi decoder might make it more practical for decoding large constraint length convolutional codes.

RNN decoders of this complexity are also likely to require the SA technique due to the highly nonlinear cost functions involved. This requires a noise input to each neuron with decreasing noise variance during decoding. In [25] a VLSI-efficient technique for generating multiple uncorrelated noise sources is described, which uses only a single LFSR. If this technique is adopted, then the logic resources required to do this are likely to be less significant compared to the demands of the multiplier circuitry.

An alternative strategy may be to employ a simpler convolutional code such as that shown below

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

and couple this with a high-resolution soft-decision decoding strategy. This code belongs to a special class of codes that performs well for the gradient descent algorithm without SA techniques [10]. If a mixed signal hardware implementation was adopted, then the neurons could be implemented using analog components to give high signal resolution. The performance of this code is shown in [26]. The RNN decoder requires few network iterations per bit as well as no neuron noise inputs. The number of multiplications and additions required per neuron update and calculated for this case is $N_{\text{mul/neuron}} = 5$ and $N_{\text{add/neuron}} = 4$.

This paper presents the case when a convolutional encoder has one input. However, a more complex encoder having k inputs and n outputs is analysed in [30]. It is shown that the neuron structure of these complex decoders includes a number of multipliers and adders as well as the activation functions.

6. CONCLUSIONS

An RNN decoder hardware design has been described, which has been developed under Simulink using the Altera DSP Builder FPGA design tools. An efficient testbed has also been developed which resembles a basic communication system, and allows testing of the RNN decoder hardware model for various AWGN channel conditions.

The RNN decoder consumes very little device resources. It has also been shown that if neurons are iterated at the system clock rate, the result is a very fast parallel convolutional decoder. The hardware RNN decoder for industry standard convolutional codes suffers from a high number of multipliers and may limit the practicality of the RNN decoder as a channel decoding solution. For the RNN decoder to be of practical use, it seems that decoding speed must be sacrificed by using time-division multiplexing in order to reduce the amount of logic resources used.

The sequential update strategy is another option for neurons updating in order to estimate message bits. According to this strategy the cost function is updated through each neuron update, and not necessarily in the steepest direction, but in the steepest direction relative to the current variable being updated [26]. Thus, the number of neurons is reduced and the multiplexers have to be added to apply the incoming bits to the inputs of the neurons. The sequential update strategy is possible to the identical structure of all neurons in the decoder network. Generally, the parallel update strategy accommodates a higher speed decoding while the sequential update strategy is slower but requires lower number of neurons. In practice, it would be advisable to investigate a pipelined version of the sequential update strategy which may result in a significant speedup.

The RNN does offer some advantages over Viterbi decoder implementations. The RNN decoder, being an iterative decoding technique, demands very little memory resources, which otherwise can be a bottleneck in Viterbi decoder implementations [31]. The regular structure of the RNN decoder is also an advantage which is likely to reduce design time for applications that use specific and nonstandard convolutional codes. The DSP Builder design flow has proven to be a useful and fast method of design prototyping. An advantage of this method is the ability to easily integrate nonsynthesizable Simulink modules into the hardware model for system testing and verification purposes throughout the development cycle.

REFERENCES

[1] M. Ibnkahla, "Applications of neural networks to digital communications—a survey," *Signal Processing*, vol. 80, no. 7, pp. 1185–1215, 2000.

[2] J. Bruck and M. Blaum, "Neural networks, error-correcting codes, and polynomials over the binary n-cube," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 976–987, 1989.

[3] I. B. Ciocoiu, "Analog decoding using a gradient-type neural network," *IEEE Transactions on Neural Networks*, vol. 7, no. 4, pp. 1034–1038, 1996.

[4] A. Hamalainen and J. Henriksson, "A recurrent neural decoder for convolutional codes," in *Proceedings of IEEE International Conference on Communications (ICC '99)*, pp. 1305–1309, Vancouver, Canada, June 1999.

[5] A. Hamalainen and J. Henriksson, "Convolutional decoding using recurrent neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '99)*, vol. 5, pp. 3323–3327, Washington, DC, USA, July 1999.

[6] A. Hamalainen and J. Henriksson, "Novel use of channel information in a neural convolutional decoder," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN '00)*, vol. 5, pp. 337–342, Como, Italy, July 2000.

[7] X.-A. Wang and S. B Wicker, "Artificial neural net Viterbi decoder," *IEEE Transactions on Communications*, vol. 44, no. 2, pp. 165–171, 1996.

[8] M. E. Buckley and S. B. Wicker, "Neural network for predicting decoder error in turbo decoders," *IEEE Communications Letters*, vol. 3, no. 5, pp. 145–147, 1999.

[9] A. Rantala, S. Vatunen, T. Harinen, and M. Aberg, "A silicon efficient high speed L = 3 rate 1/2 convolutional decoder using recurrent neural networks," in *Proceedings of 27th European Solid-State Circuits Conference (ESSCIRC '01)*, pp. 452–455, Villach, Austria, September 2001.

[10] S. M. Berber, P. J. Secker, and Z. Salcic, "Theory and application of neural networks for $1/n$ rate convolutional decoders," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 8, pp. 931–949, 2005.

[11] X. Yu and D. Dent, "Implementing neural networks in FPGAs," *IEE Colloquium on Hardware Implementation of Neural Networks and Fuzzy Logic*, vol. 61, pp. 1/1–1/5, 1994.

[12] S. Coric, I. Latinovic, and A. Pavasovic, "A neural network FPGA implementation," in *Proceedings of the 5th Seminar on Neural Network Applications in Electrical Engineering (NEUREL '00)*, pp. 117–120, Belgrade, Yugoslavia, September 2000.

[13] S. L. Bade and B. L. Hutchings, "FPGA-based stochastic neural networks-implementation," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 189–198, Napa Valley, Calif, USA, April 1994.

[14] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proceedings of International Joint Conference on Neural Networks (IJCNN '90)*, vol. 2, pp. 537–544, San Diego, Calif, USA, June 1990.

[15] B. Maunder, Z. Salcic, and G. Coghill, "High-level tool for the development of FPLD-based stochastic neural networks," in *Trends in Information Systems Engineering and Wireless Multimedia Communications Proceedings of the International Conference on Information, Communications and Signal Processing (ICICS '97)*, vol. 2, pp. 684–688, September 1997.

[16] W. To, Z. Salcic, and S. K. Nguang, "Prototyping neuro-adaptive smart antenna for 3G wireless communications," *EURASIP Journal on Applied Signal Processing*, vol. 7, pp. 1093–1109, 2005.

[17] J. J. Blake, L. P. Maguire, T. M. McGinnity, and L. J. McDaid, "Using Xilinx FPGAs to implement neural networks and fuzzy systems," *IEE Colloquium on Neural and Fuzzy Systems: Design, Hardware and Applications*, vol. 133, pp. 1/1–1/4, 1997.

[18] W. G. Teich, A. Engelhart, W. Schlecker, R. Gessler, and H.-J. Pfleiderer, "Towards an efficient hardware implementation of recurrent neural network based multiuser detection," in *Proceedings of IEEE 6th International Symposium on Spread Spectrum Techniques and Applications*, vol. 2, pp. 662–665, Parsippany, NJ, USA, September 2000.

[19] D. Abramson, K. Smith, P. Logothetis, and D. Duke, "FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problems," in *Proceedings of Euromicro Conference (EUROMICRO '98)*, vol. 2, pp. 688–693, Vesteras, Sweden, August 1998.

[20] P. Larsson, "Error correcting decoder implemented as a digital neural network with a new clocking scheme," in *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, vol. 2, pp. 1193–1195, Detroit, Mich, USA, August 1993.

[21] The Mathworks Incorporated, http://www.mathworks.com.

[22] Altera Corporation, http://www.altera.com.

[23] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[24] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu, "A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 1, pp. 109–123, 1991.

[25] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu, "Generating multiple analog noise sources from a single linear feedback shift register with neural network applications," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 1058–1061, New Orleans, La, USA, May 1990.

[26] P. Secker, "The decoding of convolutional codes using artificial neural networks," M.S. thesis, The University of Auckland, Auckland, New Zealand, 2003.

[27] S. S. Haykin, *Communication Systems*, John Wiley & Sons, New York, NY, USA, 2001.

[28] P. P. Chu, "Design techniques of FPGA based random number generator," in *Proceedings of Military and Aerospace Applications of Programmable Devices and Technologies Conference*, Laurel, Md, USA, September 1999.

[29] IEEE Std 802.11a-1999 (Supplement to IEEE Std 802.11-1999), Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHZ Band.

[30] S. M. Berber and Y.-C. Liu, "Theoretical interpretation and investigation a 2/n rate convolutional decoder based on recurrent neural networks," in *Proceedings of The 4th International Conference on Information, Communications & Signal Processing (ICICS '03)*, Singapore, December 2003, 5 pages, 2C3.5.

[31] B. Pandita and S. K. Roy, "Design and implementation of a Viterbi decoder using FPGAs," in *Proceedings of the IEEE International Conference on VLSI Design*, pp. 611–614, Austin, Tex, USA, October 1999.

Zoran Salcic is the Professor of Computer Systems Engineering at the University of Auckland, New Zealand. He holds the B.E., M.E., and Ph.D. degrees in electrical and computer engineering from the University of Sarajevo received in 1972, 1974, and 1976, respectively. He did most of the Ph.D. research at the City University, New York in 1974 and 1975. He has been with the academia since 1972, with the exception of years 1985–1990 when he took the posts in the industrial establishment, leading a major industrial enterprise institute in the area of computer engineering. His expertise spans the whole range of disciplines within computer systems engineering: complex digital systems design, custom-computing machines, reconfigurable systems, field-programmable gate arrays, processor and computer systems architecture, embedded systems and their implementation, design automation tools for embedded systems, hardware/software codesign, new computing architectures, and models of computation for heterogeneous embedded systems and related areas. He has published more than 180 refereed journal and conference papers and numerous technical reports.

Stevan Berber was born in Stanisic, Serbia in 1950. He completed his undergraduate studies in electrical engineering in Zagreb, master studies in Belgrade, and Ph.D. studies in Auckland, New Zealand. Before coming to the academic world, he had been working nearly 20 years in research institutions and in telecommunication industry. His research interests were in the following fields: mobile communication systems, digital transmission systems in integrated services digital networks (ISDN), and systems for supervision and control of ISDN networks. At present Stevan is with the University of Auckland in New Zealand. His research interests are in the field of digital communication systems (modulation and coding theory and applications), in particular CDMA systems and wireless computer and sensor networks. His teaching interests are in communication systems, information and coding theory, digital signal processing, and computer networks. He is an Author of more than 50 referred journal and international conference papers and 7 books. Stevan has been leading or working on a number of research and industry projects.

Paul Secker completed his B.E. and M.E. degrees in computer systems engineering at the Auckland University in 2001 and 2003, respectively. His research interests are in complex digital systems and their applications in digital and wireless communications.