ACCEPTED:
A. Beg, "An Efficient Realization of an OCR System Using HDL," in International Conference on Artificial Intelligence (ICAI'08), Las Vegas, NV, United States, 2008.

An Efficient Realization of an OCR System Using HDL

Azam Beg
College of Information Technology, UAE University, Al-Ain, Abu-Dhabi, UAE
Email: abeg@uaeu.ac.ae

Abstract - *This paper presents a Verilog model of an artificial neural network for Arabic character recognition. A neural network by nature is a non-linear system that presents some unusual challenges when realized in digital domain. A main feature of our proposed model is that it does not require hardware-intensive resources, such as dividers and multipliers. A character recognition accuracy of 80.3% was achieved with the model. The flexible nature of the model allows experimentation with any other different set of neural network weights, without affecting the overall network structure. The network model can be easily adapted to other similarly-written Middle-Eastern/Asian languages, such as Persian (Iran), Urdu (Pakistan), Pushto (Afghanistan), etc. Potential applications of such a system are in portable devices such as pen-shaped text readers/recognizers; Braille-aware or low-vision text-to-speech devices; or in large-scale character recognition systems such as postal mail systems, bank-check processing, etc.*

Keywords: Human-computer interaction, optical character recognition, Verilog modeling, artificial intelligence, and neural networks.

1 Introduction

Artificial neural networks (ANNs) mimic the structure and function (learning and recall) of biological neurons. The ANNs offer parallel processing of data in contrast to the conventional sequential processing computing systems [4].

A *neuron* is the basic building block of an ANN (Fig. 1). A practical ANN has one or more *input neurons*. The real world data is presented to the ANN via input neurons. There are one or more *hidden neurons* that are fed by the input neurons and may not have direct contact with the outside world. The hidden neurons process the complicated input patterns to produce useful outputs. An ANN may comprise of single or multiple layers of hidden neurons. Outputs of hidden neurons drive their respective 'forward' layer of neurons. In single-hidden layer ANNs, the hidden layers connect to the layer of *output neurons*, which produce the final results. When information passes in only one direction and there is no communication among neurons of a particular layer, the network is called a *feed-forward ANN* (Fig. 1) [13].

Usually, an ANN is specifically developed for one particular purpose. The ANNs have been used in a wide variety of applications, including human-computer interaction technologies, such as character recognition, speech recognition, machine translation, Braille systems, etc. Most of these applications are software-based, but there also have been some implementations in hardware [13].

Software implementations of the optical character recognition (OCR) systems have been reported extensively in the research literature. Off-line printed and hand-written text recognition has been done using ANNs and other technologies such hidden Markov models (HMMs) [6], [11], [13]. (We will not delve into any discussion of HMMs as the subject of this paper is ANNs). The main advantage of hardware implementation is the added improvement of ANN speeds as compared to software running on a traditional computer. The ANN hardware has been constructed in the form of VLSI chips and special accelerator boards. The stumbling block may still be the Amdahl's law that limits the speedup gained by a piece of hardware [3].

A hardware-based ANN can take up different architectural forms such as perceptron, feed-forward multi-layer perceptron, radial basis function, etc. An ANN's structural details include number of inputs and outputs, number of layers, and activation functions (AFs). Another design decision is whether the network would be analog,

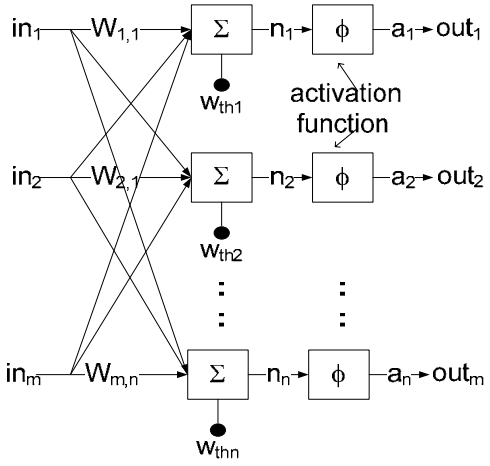

Fig. 1: A single layer of neurons. When used in a feed-forward ANN, the information only flows in one (forward) direction. Also, there is no interaction among different neurons in the same layer.

ACCEPTED:
A. Beg, "An Efficient Realization of an OCR System Using HDL," in International Conference on Artificial Intelligence (ICAI'08), Las Vegas, NV, United States, 2008.

digital, or hybrid. The learning of a network, i.e., determination of neuron count and *neuron weights*, can be done offline or online [3]. Examples of hardware-implemented of ANNs are: an FPGA-based "virtual sensor" [10], and a pulse-density-modulating NN hardware to study its statistical behavior [9].

From a designer's viewpoint, customized hardware can take relatively much longer time than the use of a programmable device, such as an field programmable gate array (FPGA), or complex programmable logic device (CPLD); both of these devices offer tens of thousands of building blocks, and their distinctions tends to blur. FPGAs have allowed implementation of complex prototypes and actual systems. RTL languages such as Verilog and VHDL are among the common hardware description languages (HDLs) used for hardware development. A higher level of abstraction is possible with languages such as Handel-C. However, this paper covers only on the Verilog implementation of the ANN [8].

In general, the implementation of a digital design in FPGA requires the following steps: (1) design entry using an HDL or schematic; (2) logic synthesis to convert HDL or schematic into a netlist, i.e., specific gates and wires; (3) logic implementation to map the gates and wires into the FPGA and creation of corresponding *bitstream* (defining opening and closing of switches for signal routing); (4) downloading bitstream into a physical FPGA chip; and (5) validating the design by providing stimuli to inputs and by measuring the outputs.

2 ANN Implementation of an Arabic OCR System

Arabic characters can have variations in their shapes; a letter can take up four or more shapes. This means that the original character set of 89 letters (inclusive of punctuation and special characters) expands to 157 different forms. This variation adds to the complexity of an OCR system. Another challenge in such a system is handling the letters which vary only by the number of dots, for example, the letters *baa*, *taa*, and *thaa*; and letters *jeem*, *haa*, and *khaa* (Fig. 2) [5]-[7].

A fully-fledged OCR system generally reads a complete page worth of information. The page may need to be rotated to remove any skew before it is split into individual lines. Each line is then split into words or letter groups. A software-based Arabic OCR system can use a large dataset for development/testing, for example, DARPA Arabic OCR Corpus [12] that comprises 345 scanned pages (at 600 DPI) of Arabic text (nearly 670k characters) from different sources such as magazines, newspapers, and a few four computer printouts (Fig. 3).

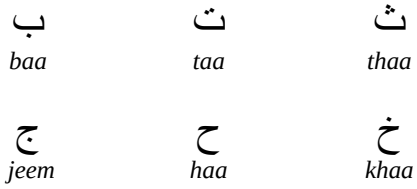

Fig. 2: The main body of letter *baa* is similar to letters *taa* and *thaa*. The only differences are the placement and count of dots. Similar situation happens with three letters *jeem*, *haa*, and *khaa*.

Two basic methods used in OCR schemes are: *matrix matching,* and *feature extraction*. Matrix matching tends to be algorithmically simpler and hence better suits a hardware implementation. Matrix matching involves comparing a character with a library of character matrices or templates.

Unlike a software-based, full-page OCR system, our proposed hardware is limited to only individual letters of Arabic language. Additionally, we opted for a matrix-matching approach due to its simplicity in terms of hardware. The development of the Arabic OCR-ANN involved the following steps:

(a) Creating training samples of Arabic letters (*alif* to *yaa*).

(b) Use the training samples to create a software-only ANN; this determined the neuron-count and neuron-weights.

(c) Implement the individual Verilog modules: adder, multiplier, shifter (divider), and AF (ramp, and piece-wise-linear approximation of a sigmoid).

(d) Implement a single neuron using the individual blocks developed in step (c).

(e) Build a hidden layer consisting of multiple neurons using weights acquired in step (b).

(f) Build an output layer which consists of multiple neurons using weights acquired in step (b).

(g) Connect inputs, hidden and output layers together.

(h) Test the complete design and analyze the results.

We will discuss these steps in detail in the following sections.

2.1 ANN Data

As mentioned earlier, the character recognition task for the proposed Verilog model was simplified by (1) including only the main Arabic characters in the dataset (and by excluding any punctuation and special symbols), and (2) using only a single form of each letter (instead of up to 4 for some of them). The 28-character ANN training set was manually created (and was not derived from scanned samples). Shapes of the characters were varied

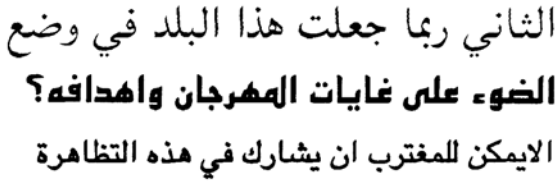

Fig. 3: Samples of computer generated text in DARPA Arabic OCR Corpus.

ACCEPTED:
A. Beg, "An Efficient Realization of an OCR System Using HDL," in International Conference on Artificial Intelligence (ICAI'08), Las Vegas, NV, United States, 2008.

slightly to get 5 samples each for a given character (Fig. 4). This resulted in a total of 140 training samples.

In order to keep the ANN input count and hence the overall hardware to a reasonable size, we chose a grid of 8x8 to define all characters; this resulted in 64 Boolean-input ANNs. The network's output layer was made of 28 neurons; each neuron represented a single character.

2.2 ANN Configuration and Training

A software package called BrainMaker [2] was used to train the ANNs. The package uses fully-connected, feed-forward, back-propagation networks. The training with the package provided us with the neuron counts and weights required for hardware development. Our experiments were limited to ANNs made up of single hidden layer of neurons, because one such layer is considered sufficient for modeling most non-linear systems [4]. In general, a neuron can implemented using the equation [13]:

$$out = f(net) = f\left(\sum_{j=0}^{n-1} a_j . w_j + w_n\right). \quad (1)$$

As we can see above, in order to model a neuron, we need these building blocks: multipliers, adders, an AF, and a threshold unit. The multipliers are used to calculate the product of an input with its corresponding weight. In our application, the input values are binary ('0' or '1'). So the multiplication operation can be very simply and *cheaply* implemented by a buffer with an enable line. The enable input is connected to the binary inputs (Fig. 5).

The ANNs produced with BrainMaker yield weights in the continuous (and not integer) values: -7.99 to +7.99. In order to simplify the hardware design, we multiplied all the original weights by 100, and used only the integer parts of the multiples. So the new value range is -799 to +799.

$$w_i' = \text{int}(100 * w_i) \quad (2)$$

Our experiments involved multiple configurations of the hidden layer. We varied the neuron count in the hidden layer from 2 to 30; we found an 8-neuron layer to fulfill our character recognition accuracy requirement of 95% (an arbitrary value). Using this size as a guideline, we were able to select the bit-widths of the computing elements, especially the adder count and widths. The adder requirements for a hidden-layer neuron are listed in Table 1. Each hidden layer neuron has 64 inputs (i_0 … i_{63}) and 65 weights; 64 weights correspond to 64 inputs, while one is a *threshold* weight (for producing non-zero output, in case all inputs are zeros).

As there are 8 neurons in the hidden layers, each of the output layer neurons has 8 inputs. There are 28 neurons in the output layer; each neuron belongs to one of the 28 letters. Note that output-layer neurons don't have threshold weights [2].

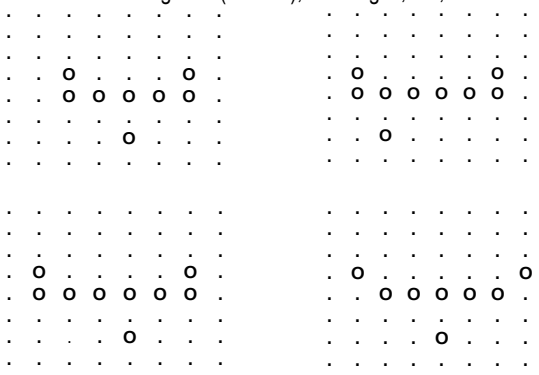

Fig. 4: Four training samples for letter *baa* (ب). The circles represent binary '1' inputs, and the dots represent the '0' inputs to the ANN. The samples assume 'thinned' characters, meaning all lines are one pixel wide.

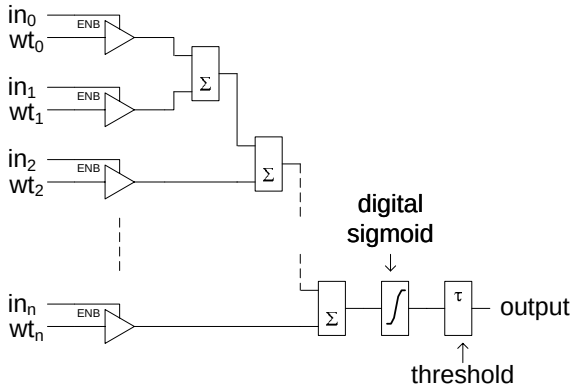

Fig. 5: Block diagram of a digital neuron in the hidden/output layers. Basic elements include buffers, adders, sigmoid function, and threshold units.

Table 1: Sixty four adders of different sizes (bit widths) are needed to implement one hidden neuron (as shown in Fig. 5) that has 64 inputs and 65 weights. Each input is between -800 and 800.

Adder size (bits)	Adder count
12	1
13	3
14	5
15	10
16	20
17	25
TOTAL	64

2.3 Activation Function Implementation

Usually all neurons of an ANN layer are based on the same type of AF. Conventionally, the neurons in the entire ANN employ the same AF, but the output layer may use a different AF type. An AF can be implemented in many different ways. Early ANNs made use of simple threshold functions which means the output is one or zero. This was a non-differentiable function which could be improved upon by using a continuous function. Generally, a sigmoid (S-shaped) AF is considered to be one of the most

ACCEPTED:
A. Beg, "An Efficient Realization of an OCR System Using HDL," in International Conference on Artificial Intelligence (ICAI'08), Las Vegas, NV, United States, 2008.

effective functions. An AF may always produce just positive values (such as logistic function), or both negative and positive values (such as hyperbolic tangent function). As known well, a sigmoid function is defined as

$$f_1(x) = \frac{1}{1+e^{-x}}, \quad (3)$$

and a hyperbolic tangent is defined as [13]

$$f_2(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}. \quad (4)$$

An analog neuron can be implemented using non-linearity of a CMOS device, but it suffers from the effects such as thermal drift; and the device may not be re-programmed. A neuron's digital implementation, on the other hand is programmable, and is able of producing consistent results. However, the non-linear nature of a neuron's AF poses a design challenge when realized in digital hardware.

There are several ways of approximating the sigmoid function (equation (3), e.g., first or second order piecewise linear (PWL) approximation, combinational approximation, etc. We opted for the first-order three-line approximation. After extensive experimentation, we arrived at the following set of equations:

$$AF_i = \frac{i}{64} - 50, -800 \geq i \geq -201, \quad (5)$$

$$AF_i = \frac{i}{4}, i = -200 \geq i \geq 199, \quad (6)$$

$$AF_i = \frac{i}{64} + 50, 200 \geq i \geq 800, \quad (7)$$

where i is the input to the AF ranging from -800 to 800; the AF output ranges from -64 to 64. Fig. 6 shows the plots of actual sigmoid and its PWL-based approximations. In the figure, 'PWL1' corresponds to equation (5), 'PWL2' to equation (6), and 'PWL3' to equation (7). One can notice that the equations (5)–(7) require these computing elements: divider, and adder/subtractor. As a divider can be quite costly in terms of hardware (i.e., device/area cost), we chose the denominators of the fractions such that they can be represented in 2^n form. This allowed us to replace the division operation with simple right shifts: 2-bit shifting for division-by-4, and 6-bit shifting for division-by-64. Table 2 mathematically compares the actual sigmoid function represented by equation (3), and its PWL approximations with equations (5)–(7). The standard deviation varies between -3.49 and 2.11, while the average error ranges from -2.71% to 2.69%. The average errors are quite comparable to different methods proposed in [8]. Finally, the RMS error is 6.7%.

2.3 Threshold Function Implementation

The threshold function in our neurons outputs '0' if the input is less than or equal to zero; else the output is '1'. The hardware implementation of this function simply comprises of a most-significant bit comparator (Fig. 5).

2.4 Model Simulation and Analysis

The proposed Verilog models were simulated using ModelSim XE/Starter 5.5e_p1 revision 2001.11. Although the prediction accuracy of a software-only model is up to 95%, the hardware model has lower accuracy of 80.3%. The higher error rate could be mainly attributed to the following factors:

1) Approximation of the sigmoid AF with a three-line PWL functions: 5 or 7-line approximation may provide better results. Actual dividers, although being hardware-intensive blocks, could be tried out instead of simple shifters. Use of a lookup table instead of linear approximations could be investigated.

2) Lower input count: The input samples are made up of 8x8 grids. Larger grids such as 8x10, 10x10 could be tried. Although this will lead to higher hardware implementation cost.

3) Similarity of letter shapes: Additional training samples could improve the ANN learning ability, hence improving the prediction accuracy. This may not affect the ANN hardware configuration.

3 Conclusions

A Verilog implementation of an ANN for the purpose of recognizing Arabic language letters has been proposed in this paper. The Verilog/hardware implementation of the ANN had lower accuracy than its software counterpart. One reason for reduced accuracy is the PWL approximation of non-linear sigmoid AF. The hardware's character recognition accuracy is still significantly high at

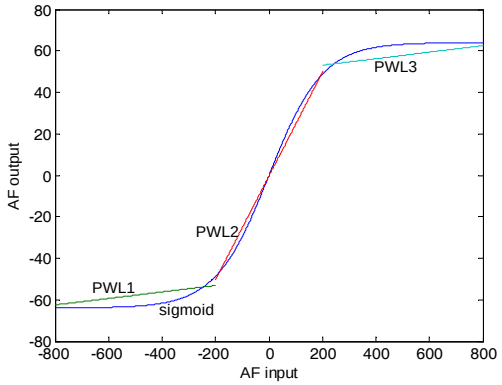

Fig. 6: Graphical comparison of a sigmoid function with its three-line PWL approximations (PWL1-3). The input range is [-799 … +799], and the output range is [-64 … +63].

Table 2: Numerical comparison of actual sigmoid AF with its PWL approximations.

	Standard deviation	Average error
Line 1 (eq. 5)	2.08	-2.71%
Line 2 (eq. 6)	-3.49	2.49%
Line 3 (eq. 7)	2.11	2.69%

ACCEPTED:
A. Beg, "An Efficient Realization of an OCR System Using HDL," in International Conference on Artificial Intelligence (ICAI'08), Las Vegas, NV, United States, 2008.

80.3%. Further improvements are possible without significantly altering the hardware, for example, by using a larger dataset for training. Different neuron weights can be easily incorporated in the ANN as the weights are inputs to individual neurons and do not require change in network topology.

The hardware implementation is somewhat independent of the language being recognized. Languages that use character sets similar to Arabic, such as Persian and Urdu can be also recognized by the proposed hardware. The only changes required would be to the neuron counts and their respective weights.

We are currently investigating an expanded training character-set for the ANNs. We are also looking into metrics such as the actual cost/size of the hardware by implementing the proposed design using synthesis tools (such as Synopsys Design Compiler). A pipelined approach, especially to save on adder hardware (inside a neuron) is also under consideration. The findings from these experiments will be reported in a future publication.

Acknowledgements

The author acknowledges the contribution made by UAE University students (Sameira Sultan, Amal Ahmad, Shama Ali, Alyaa Rashed, and Aisha Sultan) in conducting the ANN experiments and Verilog development, as a project.

References

[1] B. Al-Badr and S. Mahmoud, "Survey and Bibliography of Arabic Optical Text Recognition," Signal Processing, vol. 41, no. 1, pp. 49-77, 1995.

[2] BrainMaker User's Guide and Reference Manual, 7th ed. California Scientific Software Press, California, 1998.

[3] C. S. Lindsey and T. Lindblad, "Review of hardware neural networks: a user's perspective" [Online]. Available: http://www.particle.kth.se/~lindsey/elba2html/elba2html.html

[4] I. H. Witten, and E. Frank, "Data Mining," Morgan Kaufman Publishers, San Francisco, CA, 2nd ed., 2005.

[5] J. Bellegarda and D. Nahamoo, "Tied Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition," IEEE Int'l Conf. Acoustics, Speech, Signal Processing, vol. 1, pp. 13-16, Glasgow, Scotland, May 1989.

[6] K. Aas and L. Eikvil, "Text page recognition using grey-Level Features and Hidden Markov Models," Pattern Recognition, vol. 29, pp. 977-985, 1996.

[7] M. Allam, "Segmentation Versus Segmentation-Free for Recognizing Arabic Text," Proc. SPIE, vol. 2,422, pp. 228-235, 1995.

[8] M. T. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," IEE Proceedings on Computers and Digital Techniques, Vol. 150, Issue 6, pp. 403-411, Nov. 2003.

[9] M. Yasunaga and Y. Hirai, "Ising model calculation using PDM neural network hardware: Boltzmann statistical mechanics embedded in the hardware," International Conference on Neural Networks, Vol. 2, pp. 948 – 952, Jun. 1997.

[10] M. A. A. Leon, A. R. Castro, and R. R. L. Ascencio, "An artificial neural network on a field programmable gate array as a virtual sensor," Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications, pp. 114 – 117, 1999.

[11] N. B. Amara and A. Belaid, "Printed PAW Recognition Based on Planar Hidden Markov Models," 13th Int'l Conf. Pattern Recognition, vol. 2, pp. 220-224, Vienna, 1996.

[12] R. B. Davidson and R. L. Hopley, "Arabic and Persian OCR Training and Test Data Sets," Proc. Symp. Document Image Understanding Technology (SDIUT97), pp. 303-307, Annapolis, Md., 1997.

[13] T. Masters. Signal and Image Processing with Neural Networks. John Wiley & Sons, New York, 1994.

[14] W. Cho, S. W. Lee, and J. H. Kim, "Modeling and Recognition of Cursive Words with Hidden Markov Models," Pattern Recognition, vol. 28, pp. 1,941-1,953, 1995.