Proceedings of the 7th WSEAS International Conference on Neural Networks, Cavtat, Croatia, June 12-14, 2006 (pp7-10)

FPGA Implementation of a Neural Network Control System for a Helicopter

M. Zheng, M. Tarbouchi, D. Bouchard, J. Dunfield,
Department of Electrical and Computer Engineering,
Royal Military College of Canada,
Kingston, Ontario, Canada, K7K 7B4

Abstract – ***In this paper the design and development of an intelligent controller based on artificial neural networks (ANN) on a Field Programmable Gate Array (FPGA), for a four-rotor helicopter to be capable of achieving vertical take off and to be able to sustain a specified attitude, is presented. To overcome challenges due to the complexities of creating a Neural Networks Controller to work in real-time, a hardware-friendly training algorithm is chosen. The ANN is implemented on a Virtex-II Pro XC2VP30 FPGA from Xilinx. Simulation results are analyzed to highlight the performance of the hardware.***

1. Introduction

This work is a continuation of a research work on a Neural Network Based Control of a Four Rotor Helicopter [1]. The device used was a helicopter of a four-rotor configuration (Figure 1). A neural network controller aimed to autonomously control the roll, pitch and yaw of the four rotor helicopter system. With the addition of height control, the helicopter would be able to hover. By the use of offsets in the control feedback loop, the helicopter could then be made to travel horizontally. If navigation and a behaviour capability were then added, the helicopter would become a fully autonomous hoverable robot. This paper focuses on the neural network control aspects and challenges of performing this work on a Field Programmable Gate Array (FPGA).

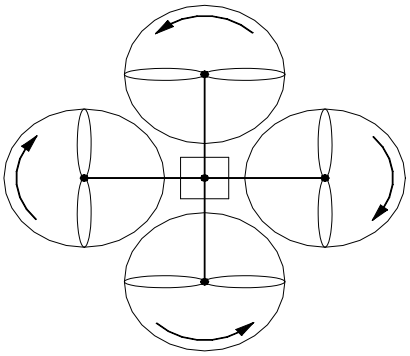

Figure 1: Four-rotor Configuration

Originally, the helicopter came with a manual controller, and requires an experienced operator to manipulate the controller. In [1], Dunfield *et al*. developed a neural network control system on a microcontroller to autonomously control the roll, pitch and yaw axis of the four rotor helicopter system (see Figure. 2).

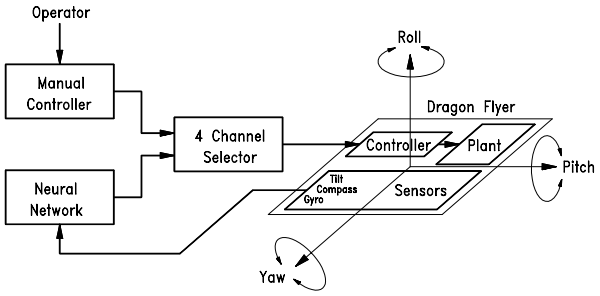

Figure 2: Helicopter Control System

The neural network developed in [1] was a 2-layer feed-forward network, with 6 hidden layer neurons and 3 output layer neurons. The electrical sensors and compass installed on the helicopter detected the position of the helicopter, and collected the signal data for off-line training of a neural network. The neural network simulates the activity of the operator, and, in accordance with the current position and movement, it sends out a control command. The training was done off-line due to hardware limitations of the microcontroller.

The neural network trained off-line [2,3,4] had the following limitations: (1) due to the sampling method and process, the training data contained errors; and, (2) the amount of off-line training data was limited.

With on-line training, data can be sampled and collected in any real-time scenarios, allowing the neural network to learn tasks for which it has not been previously trained. An FPGA is well suited for this task. In addition, the typical computational characteristics of ANNs such as parallelism, modularity and dynamic adaptation can be realized in hardware.

2. Theory of the Algorithm

In order to perform on-line training, the NN architecture and training algorithm must be implemented on an FPGA chip [5,6]. In this paper a fast, robust, and hardware-friendly training algorithm is used, based on a technique presented by Gadea *et al*. [7].

2.1 Forward Computation

The computation performed by each neuron (in the hidden layer) is as follows:

$$o_k^{(s)} = f(H_k^{(s)}) = f(\sum_{j=1}^{N_{s-1}} w_{kj}^{(s)} o_j^{(s-1)} + \theta_k^{(s)}) \qquad (1)$$

where k=1, … , N and s=1, … , M, and,

M : Number of the layers in a Neural Network;

N : Number of the neurons in a layer;

$H_k^{(s)}$: Weighted sum;

$w_{kj}^{(s)}$: Synaptic weight;

$\theta_k^{(s)}$: Bias.

$o_k^{(s)}$: Neuron output;

$f(H_k^{(s)})$ = Activation function.

The activation function is *logsig*:

$$f(x)_{\log sig} = \frac{1}{1 + \exp(-x)} \qquad (2)$$

2.2 Backward Computation

The local gradients $\delta_j^{(s)}$ was calculated as follows:

$$\delta_k^{(s)} = \varepsilon_k^{(s)} f'(H_k^{(s)}) \qquad s = 1,...,M \qquad (3)$$

The error term $\varepsilon_k^{(s)}$ was calculated as follows:

$$\varepsilon_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = m \\ \sum_{j=1}^{N_{s+1}} w_{jk}^{s+1} \delta_j^{s+1} & s = 1,...,m-1 \end{cases}$$

where $f'(H_k^{(s)})$: the derivative of the activation function.

2.3 Weight Update

The weight changes for all the weights were calculated as follows:

$$\Delta w_{kj}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \qquad (4)$$

All the weights were updated as follows:

$$w_{kj}^{s}(n+1) = \Delta w_{kj}^{(s)}(n) + w_{kj}^{(s)}(n)$$

where $k = 1, \ldots, N_s$ and $j = 1, \ldots, N_{s-1}$, and,

$w_{kj}^{s}(n+1)$ = Updated synaptic weight (or bias);

$\Delta w_{kj}^{(s)}(n)$ = Change in synaptic weight (or bias);

$w_{kj}^{(s)}(n)$ = Old synaptic weight (or bias);

3. Hardware Design

3.1 Xilinx System Generator Based Design Flow

Figure 3 shows System Generator Based design flow [2]. First of all, in this approach, the design process uses a unique platform for both simulation and implementation. In addition, the implementation phase does not build the algorithm repetitively, because the simulation model represents in every detail the design to be implemented in the FPGA. After the hardware-in- the loop verification, the design can be converted directly into hardware without further debugging and testing. Thus the design process can be accelerated.

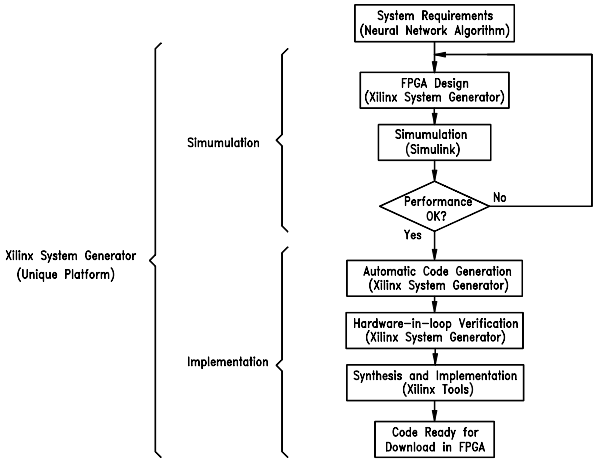

Figure 3: System Generator Based Design Flow

3.2 Hardware Design

Figure 4 shows the whole neural network. It has 9 inputs (6 input data, 3 training data), 6 hidden layer neurons, 3 output layer neurons, and 3 outputs. There are 2 directions of data flow in the network: forward computation and backward propagation.

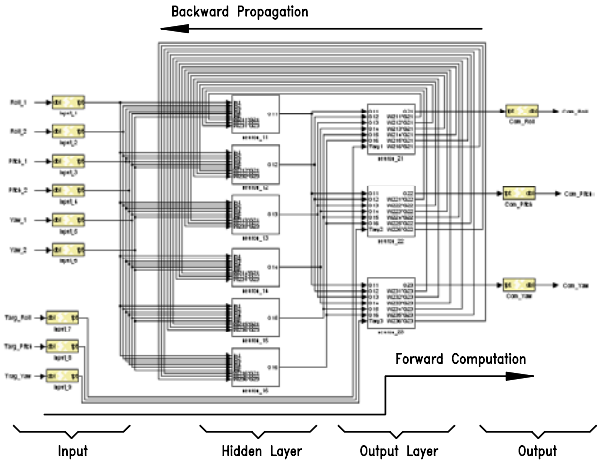

Figure 4: Neural Network

4. FPGA Implementation

Figure 5 shows the completed design diagram of the neural network with peripheral components. It has 3 inputs, 1 output, and 4 modules, where: *nn_4_1_clk_wrapper* is the Neural Network Module, *uarts* is the UART Module, *CLKDLL* is the Clock Divider Module, *applic* is the Cooperation Module.

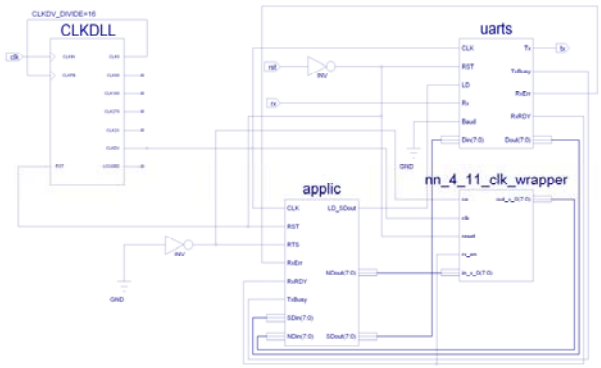

Figure 5: System with Peripheral Component

The completed design was guided though the operations of synthesis and implementation, in order to generate the bit-stream for FPGA configuration. At the final step, the design was downloaded to the FPGA using.

5. Results

5.1 Hardware Simulation Results

Figure 6 shows the hardware simulation results. There are 3 curves in this graph: Initial Output Curve, Target Curve, and On-line Training Curve.
From the Figure 6-6, several conclusions can be drawn: (1) The On-line Training Curve is different from the Initial Output Curve, meaning that the on-line training took place during the process. (2) The On-line Training Curve is different from the Target Curve, meaning that the NN had not been over-trained. (3) After the transfer of the design into hardware, the system can still achieve a stable state and convergence result.

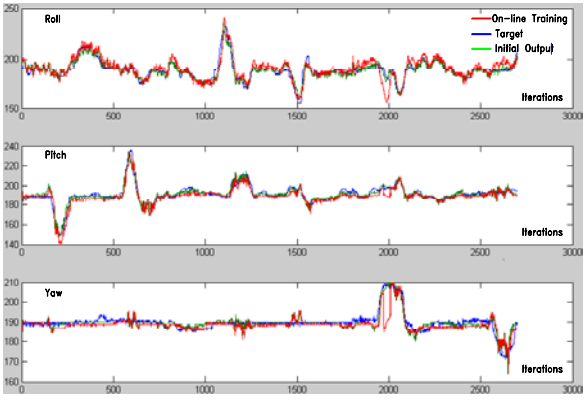

Figure 6: Hardware Simulation Results

4.2 FPGA Implementation Results

Table 1 shows the area report of FPGA implementation results.

Implementation Result		
Number of FPGA Devices Utilization	Total Gate	2,945,873
	MULT 18x18s	135/136, 99%
	Slice Flip Flops	1381/27392, 5%
	4 input LUTs	3123/27392, 11%
	External IOBs	4/556, 1%
	Logic Slices	2026/13696, 14%

Table 1: Summary of the Area Report

Table 2 shows the timing report. The minimum period of one forward and back-propagation calculation is less than 100 ns. Compared to the software neural network that took approximately 20 ms to finish a forward calculation (i.e., 200,000 times longer), this is really a breakthrough.

Implementation Result		
Timing Summary	Minimum period	48.744 ns
	Maximum Frequency	20.515 MHz

Table 2: Summary of the Timing Report

5.2 Design Verification

Using the hardware-in-the-loop technology, the FPGA can be accessed from the Simulink interface. The hardware-in-the-loop block can be looked at as a Simulink block. But different from the software simulation, during the test, the program is actually running on the hardware and the data is transferred between the PC and FPGA board.
To test the design, the input vectors and the target vectors were sent to the NN implemented on the FPGA for one epoch training process. The real-world training data was tested on the FPGA board. The output data was then received by the PC and presented on a Simulink Scope block. The result is consistent with the On-line Training Curve in Figure 6. The design was therefore tested and verified.

6. Conclusion

In this paper, an on-line training neural network was designed and implemented on an FPGA chip. The major conclusions are as follows:
(1) A hardware-friendly on-line training algorithm was chosen for the neural network. Based on a mathematical analysis, on-line training can improve system performance.
(2) The hardware data representation needs to be of the fixed-point type. The integer type was unsuitable for the FPGA, since the data values in the back propagation network are much smaller than in the feed-forward network.
(3) Compared to the software NN in the microcontroller, the FPGA provided an increase in processing speed of 5 to 6 orders of magnitude.

(4) Xilinx System Generator design tool bridges the gap between control system design and hardware design. It makes the whole research possible, without writing challenging VHDL code.

References

[1] Dunfied J., Tarbouchi M., Labonte G., *Neural network based control of a four rotor helicopter*, Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on Volume 3, 8-10 Dec. 2004 Page(s):1543 - 1548 vol. 3.

[2] Ricci F., Hoang Le-Huy, An FPGA-based rapid prototyping platform for variable-speed drives, IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the] Volume 2, 5-8 Nov. 2002 Page(s):1156 - 1161 vol.2.

[3] Kishan M., Chilukuri M., Ranka S., *Elements of Artificial Neural Networks*, MIT Press 2000, Page 5.

[4] Heemskerk J.N.H., *Overview of Neural Hardware*, Unit of Experimental and Theoretical Psychology, Leiden University, The Netherlands, 1995.

[5] Foo S.K., Saratchandran P., Sundararajan N., *Parallel implementation of backpropagation on transputers*, Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on Volume 3, 25-29 Oct. 1993 Page(s):3058 - 3061 vol.3.

[6] Rose J., Gamal A. El, Sangiovanni-Vincentelli A., *Architecture of field-programmable gate arrays*, Proceedings of the IEEE, vol. 81, no. 7, pp. 1013-1029, July 1993.

[7] Gadea R., Cerda J., Ballester F., Macholi A., Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation, System Synthesis, 2000. Proceedings. The 13th International Symposium on 20-22 Sept. 2000 Page(s):225 - 230.