

Synthesising Hardware for Neural Networks with CADDY/CALLAS

H. Speckmann, P. Thole, W. Rosenstiel
Wilhelm-Schickard-Institut f. Informatik
University of Tübingen
7400 Tübingen, Sand 13, Germany
email: speckman@peanuts.informatik.uni-tuebingen.de

Abstract

In this paper we present a system for automatic synthesis of special purpose hardware for neural networks. Only an algorithmic description of the behaviour of the hardware and a simulation environment have to be written by the designer using VHDL. This description can be automatically mapped onto hardware using the synthesis tool CALLAS and the design system MENTOR. Using the concept of a simulation block connected to the actual design level of the hardware implementation the consistency of the different design levels can be proved. By using reprogrammable gate arrays (FPGAs) with this system we are able to do rapid prototyping of neural network hardware in a few days.

1 Introduction

Large CPU-times for training large data sets to neural networks and the impossibility of real-time evaluation constitute a serious obstacle for practical applications of neural networks, especially software implementations on a SISD-computer. Running software simulations on MIMD-computers, e.g. transputer networks, or on SIMD-computers (we are using MasPar), bring up the performance. For some general concepts which have many applications specialized hardware may further improve the performance. But the development of special purpose hardware is time consuming and expensive. So there is a need for automatic synthesis for special purpose hardware supporting rapid prototyping. We propose to use field programmable gate arrays (FPGAs), which are not very expensive and reprogrammable directly by the user. As design example we have chosen Kohonen's selforganizing map (SOM) which has been introduced by T. Kohonen [4]. On the one hand this unsupervised learning network is used with many applications, e.g. speech and character recognition or data analysis of chemical gas-sensors what we are currently doing [2]. On the other hand there are only a few hardware implementations [7].

2 The design flow of the automatic synthesis

As shown in figure 1 the hardware design process consists of three main parts:

- high-level synthesis
- logic synthesis with technology mapping

- consistency checking with an uniform simulation environment

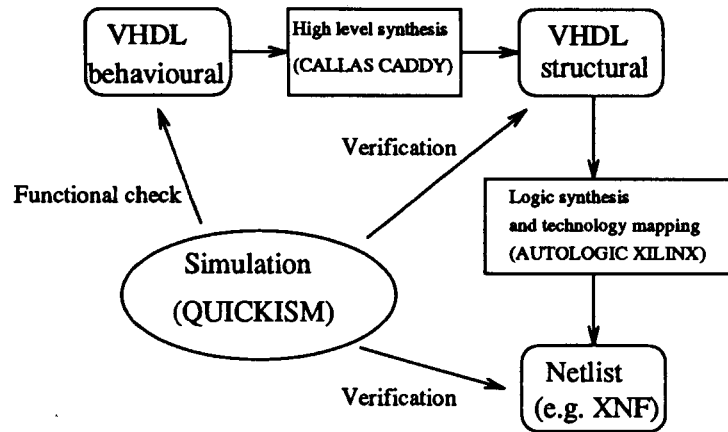


Figure 1: The designflow of the synthesis

Input for the high-level synthesis are flow graphs generated by a flowgraph compiler from a VHDL behavioural description of the neural network algorithm. The algorithm is specified by sequential statements in a single process. Because VHDL [3] was primarily designed for simulation, there are some difficulties to use VHDL as input for high-level synthesis concerning the capture of timing constraints. Both synthesis tools offer different solutions. Using CADDY [5] the user has influence to the timing by a preference function, which is minimized during allocation and scheduling. So a design space exploration is done by choosing a data path modul with different area and delay properties. In contrast to CADDY, CALLAS [1] does no design space exploration. The user has to map I/O-operations to clock cycles in the VHDL behavioural description. This timing concept guarantees the paradigm "What you simulate is what you synthesize".

To verify the functional correctness of the implementation and the consistency of the different design layers the designer has to implement a simulation environment also using VHDL. The different design levels can be simulated with MENTOR'S QUICKSIM. For this reason we have to find an unconventional way to build a simulation environment wich is described later.

After the synthesis with the high-level synthesis tool CALLAS/CADDY the AUTOLOGIC VHDL tool from MENTOR GRAPHICS is used to synthesize a netlist of simple generic components from the structural VHDL description, next AUTOLOGIC performs logic synthesis and technology mapping. At this state a timing simulation of the design is possible. We use XILINX FPGAs as target technology. So the result, a XNF netlist in this case, can be mapped onto the FPGA with the XILINX tools [9].

3 Synthesis study: A coprocessor for Kohonen's SOM

3.1 Short description of SOM's algorithm

For our design study we use a modified version of Kohonen's algorithm [8, 4]. Generally the SOM consists of a two-dimensional array of identical processor units (PU_{ij}) with each processor unit storing a single vector w_{ij} with the components w_{ijk} , i and j denote the position of the processor unit in the array. When the map is trained, each processor unit computes the euclidian distance D_{ij} of the input vector $S = (s_1, \dots, s_n)$ and the stored vector w_{ij} according to (1).

$$D_{ij} = \sum_{k=1}^{k=n} (w_{ijk} - s_k)^2 \quad (1)$$

After computing the distance for all PE_{ij} , the map searches the processor unit which stores the most similar vector with the minimum distance D_{ijmin} to the input vector of the array, and then the processor unit and its neighbourhood are adapted to the input vector according to (2).

$$w_{ijk}(t+1) = (1 - e_{ij}(t)) * w_{ijk}(t) + e_{ij}(t) * s_k \quad (2)$$

where $e_{ij}(t)$ is the adaptation function, which indicates the degree of adaptation of the processor units towards the input vector ($0 \leq e_{ij}(t) \leq 1$, $e_{ij}(t) \rightarrow 0$ towards the training time t). For the adaptation function e_{ij} we use the Gaussian function, which decreases linearly during training time in width and height.

3.2 Design of the coprocessor

No.	Mode
000	Initialization of the weight memories
001	Reading out of the weight memories for evaluating the learning results
010	Calculation of the minimum distance
011	Adaptation of one processing unit
100	Reading in of an input vector

Table 1: Working modes of the coprocessor

First the coprocessor is specified with behavioural VHDL. The model of the neural network used as example for implementation and synthesis is based on the vector processing concept for the SOM described in [6]. It has 5 working modes implemented with a case statement in VHDL (table 1). We have implemented a coprocessor which can calculate a SOM consisting of 4 processing units each holding a vector with 4 components .

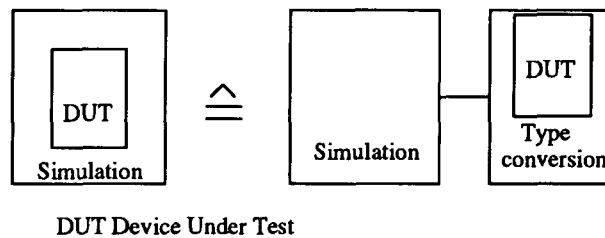


Figure 2: Conventional and unconventional way to simulate a VHDL description

A simulation environment is written used in all design levels from behavioural description down to the FPGA. In contrast to the conventional way of testing a VHDL description where a circuit is embedded into a simulation environment written with VHDL, we use a new concept, enabling to simulate all design levels with the same simulation environment (figure 2).

The circuit's VHDL description is embedded into a VHDL environment converting the original signals to signals being accepted by QUICKSIM. Additionally a simulation environment has to be written, which contains the classical signal assignments. Both building blocks have to be connected to a whole system (see figure 3) and the simulation with QUICKSIM can be started.

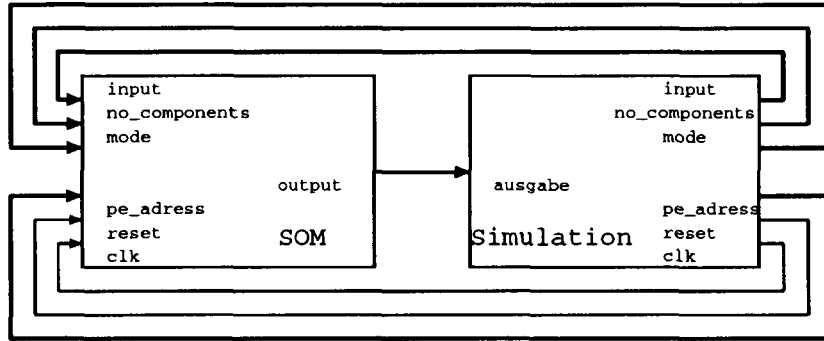


Figure 3: Schematic of the whole implementation consisting of the simulation block and the SOM's body

After specification the design was synthesized to structural VHDL using the synthesis tool CALLAS or CADDY. The consistence of the resulting design to the behavioural description is verified with the simulation environment. After that the structural VHDL description of the SOM can be mapped to a XILINX FPGA using MENTOR's autologic and XILINX mapping tool.

4 Results and Conclusions

We have described a complete system enabling an unexperienced designer to synthesize a complex hardware from a short behavioural VHDL description. In our example the behavioural description consists of about 150 lines. The structural VHDL description of the synthesized hardware consists of about 3000 lines. We map this description to XILINX FPGAs. This results in 720 configurable logic blocks (CLBs) which fit into three XC3090 or two XC4008 FPGAs. Of course other target architectures are possible, e.g. ASICs.

Hardware implementations of neural nets are still in the area of research. A lot of also "real-time" experiments with different hardware architectures are required and necessary to find good hardware implementations for different neural net applications. Our prototype synthesis and evaluation system is an important step towards this goal.

References

- [1] J. Biesenack und A. Langmaier. *CALLAS Manual*. Siemens AG, 1992.
- [2] J. Göppert, H. Speckmann, W. Rosenstiel, W. Kessler, G. Kraus, und G. Gauglitz. Evaluation of spectra in chemistry and physics with Kohonen's selforganizing feature map. In *Neuronimes 92*, 1992.
- [3] IEEE. *IEEE Standard VHDL Language Reference Manual*, 1987.
- [4] T. Kohonen. *Selforganization and associative memory*. Springer Verlag Heidelberg New York Tokyo, 1984.
- [5] B. Kretschmar, P. Gutberlet, P. Thole, und W. Rosenstiel. The CADDY synthesis system - a brief introduction -. In *Eurochip course on high level system design*, 1992.

- [6] H. Speckmann, P. Thole, und W. Rosenstiel. Hardware implementations of Kohonen's selforganizing feature map. In *IJCNN Beijing, China*, Seiten III 183–187, 1992.
- [7] P. Thole, H. Speckmann, und W. Rosenstiel. A hardware supported system for Kohonen's selforganizing map. In *Proc. of the Third International Conference on Microelectronics for Neural Networks*, Seiten 29 –34. UnivEd Technologies Ltd, 1993.
- [8] V. Tryba. *Selbstorganisierende Karten: Theorie, Anwendung und VLSI-Implementierung*. VDI Verlag, 1992.
- [9] XILINX. *The programmable gate array data book*, 1991.