Extended Abstracts of the 2014 International Conference on Solid State Devices and Materials, Tsukuba, 2014, pp102-103

# SoC Realization of LVQ Neural Network with On-chip Learning and Recognition

Fengwei An, Toshinobu Akazawa, Shogo Yamazaki, Lei Chen, and Hans Jürgen Mattausch

Hiroshima University, Higashi-Hiroshima, Japan
anfengwei@hiroshima-u.ac.jp

**Abstract**

**The reported neural network realizes Learning Vector Quantization (LVQ) with high flexibility for different applications. It is based on a hardware/software co-design concept for on-chip learning and recognition and designed as a SoC in 180nm CMOS. The time consuming nearest Euclidean distance search in the LVQ algorithm's competition layer is efficiently implemented as a pipeline with parallel p-word input. Since neuron number in the competition layer, weight values, input and output number are scalable, the requirements of many different applications can be satisfied without hardware changes. Classification of a *d*-dimensional input vector is completed in $\lceil d/p \rceil + R$ clock cycles, where $R$ is the pipeline depth. Adjustment of stored reference feature vectors (FV) during learning is done by the embedded 32-bit RISC CPU, because this operation is not time critical.**

## 1. Introduction

Since the usage of hardware neural networks (HWNN) has remained limited in practical applications, they have become less appealing than they were initially. In this context, adaptability to different applications in addition to high processing speed is a desired feature, which can help to increase the practical popularity of HWNNs. A promising HWNN concept for pattern recognition is the Learning Vector Quantization (LVQ), which was introduced by T. Kohonen [1]. Highly-reliable digital-circuit implementations have been designed previously in [2-3]. On the other hand, analog-circuit implementations lack precision, but can be made much smaller and have less power consumption [4].

In this research, the basic LVQ algorithms, namely, LVQ1 is implemented on a SoC platform. Learning in the LVQ algorithm is realized by modifying the values of feature vectors (FV) in accordance with a distance function, usually the Euclidean distance. The neuron, whose weight vector is most similar to the input, called winner, is adjusted towards the input. Classification after the learning is based on a neighborhood function i.e. the classifier assigns the same class label to all samples that have the same closest FV.

## 2. Nearest-Distance-Search Pipeline with Parallel P-word Input Architecture (PPPI)

A number n of *d*-dimensional FVs ($w_i$) are assumed to be placed into the hidden neurons. Several FVs are normally assigned to the same class representing a Voronoi region. An input vector $x$ is decided to belong to the class to which the nearest $w_i$ belongs based on the Euclidean distance function as shown in (1), where the root operation need not be computed for finding the minimal distance.

$$c = \arg\min\{\sum_{i=0, j\le d}^{n} \sqrt{(x - w_{ij})^2}\} \quad (1)$$

The minimal distance can be found in *O(dn)* time for a *d*-dimensional space and the corresponding search operation is executed using a pipeline with parallel *p*-word input (PPPI) architecture, as shown in Fig.1. The pipeline can in parallel accept *p* dimensions in each clock cycle. The *d*-dimensional input vector and the reference FVs are placed into *p* SRAM blocks in the form of m partial vectors ($m = \lceil d/p \rceil$: the smallest integer not less than *d/p*). As a result, for flexibly satisfying the requirements of different applications, vector dimension *d* and FV number *n* can be set to arbitrary values by the on-chip processor. The signal "Next", which is asserted when the address of the partial input vector is equal to *m*, distinguishes the distance calculation for two different vectors. This means, that number and dimension of input vectors or FVs are freely user controllable in this design. The partial-vector storage and the "Next" signal mainly contribute to the flexibility in the PPPI itself. The winner signal in Fig.1 is a load signal for the address of the winner FV. Afterwards, the class label has to be calculated by the embedded processor for determining the classification result, during both learning and recognition processing.

The minimal distance can be found in *m+R* clock cycles, where *R* is the pipeline depth, except for the first pipeline register bank in Fig.1. Due to the pipeline architecture, the complexity for the recognition in LVQ is not a linear function of *m* (*m>1*). In this work, the PPPI architecture thus enables the time consuming nearest-distance search operation even when applications need high dimensional vectors.

## 3. SoC for Learning and Recognition by LVQ

The learning processing of the basic LVQ1 is defined by equations (2) and (3), which are used to adjust FVs and class

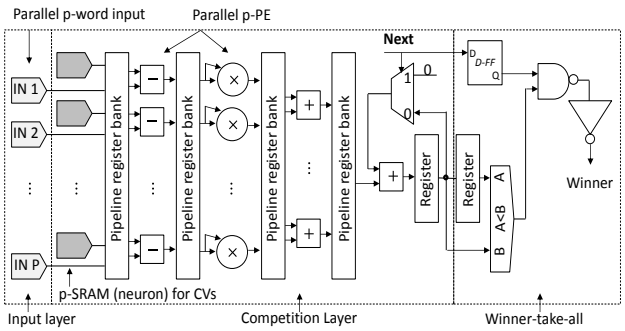


Fig. 1 PPPI architecture for a LVQ neural network. Input layer and neurons are p SRAM arrays for the storage. The "Next" signal is an R clock-delayed signal. The multiplexer is used to separate the input vectors.

boundaries. Suppose $x(t)$ and $w_c(t)$ represent sequences of training vectors and winner FVs in the discrete-time domain, respectively. Then $w_c(t)$ is adjusted according to:

$$w_c(t+1) = w_c(t) - \alpha_t[x(t) - w_c(t)] \quad (2)$$

if $x$ and $w_c$ belong to different classes, or

$$w_c(t+1) = w_c(t) + \alpha_t[x(t) - w_c(t)] \quad (3)$$

if $x$ and $w_c$ belong to the same class.

In comparison to previous hardware implementations, the designed LVQ SoC shows high flexibility, because the updating function of FVs is implemented on an embedded RISC CPU by software rather than by a reusable circuit. In addition to FV updating, FV initialization, data transfer, and winner-class calculation are also carried out via software.

## 4. Implementation Results and Conclusion

An LVQ neural network according to the developed SoC architecture has been fabricated in 180nm CMOS technology (Fig.2) including 10KB shared memory which can store 2KB input vectors and 8KB FVs, respectively. In this work, $p$ = 8 and 16-bit precision are chosen for the PPPI architecture which thus has a throughput of 128 bits per clock cycle and a pipeline depth of 7 stages.

Since no previous LVQ VLSI-research had the goal of high flexibility, the achieved flexibility result are difficult to compare and are described here by three aspects as follows: Differing from the previous ASIC implementation, firstly, dimensions of input vectors and FVs are changeable between 1 to 1024; additionally, the number of FVs is scalable from 1 to 512; finally, the learning rate $\alpha_t$ may be constant or decreased monotonically for adapting to different applications. Overall flexibility is not limited by the hardware design flexibility described above, but can be extended by the on-chip embedded processor using an external memory.

The area of the fabricated chip is 4.88 mm$^2$ and the average power consumption is 214 mW. Fig.2 shows the chip photograph and the specifications of the fabricated chip. Shared memory (SRAM array for input vectors and FVs) and RISC CPU with $32 \times 32$ multiplier consume about 43% and 6%, respectively. The efficiency of the fabricated chip is evaluated by the recognition speed $S_r$ and the learning speed $S_l$. In the case where dimensionality of input vector and FVs is less or equal than 8, the minimal $S_r$ is 0.32 µs and the minimal $S_l$ of each iteration is 20.9 µs, when only one FV is stored in each hidden neuron. In other words, we achieve a recognition rate of 3125000 vectors per second and a learning rate of 48500 vectors per second. Generally, number and dimension of the vectors affect actual recognition and learning speed.

| | |
|---|---|
| Technology | 180 nm |
| Chip area | 4.88 mm$^2$ |
| Power supply | 1.8 V |
| Power dissipation | 214 mW |
| Frequency | 25 MHz |
| Bit precision | 16-bit |
| Dimension of input vectors | 1-1024 |
| Number of neurons | 1-512 |
| Minimal recognition speed (µs) | 0.32 |
| Minimal learning speed (µs) | 20.9 |

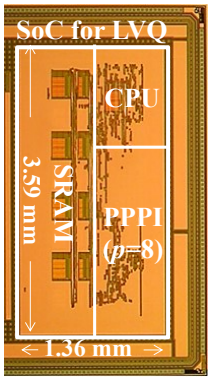


Fig. 2 Micrograph of the designed SoC in 180nm CMOS technology and the achieved performance results.

Because all previous state-of-the-art work needs an external control unit or a processor to perform learning and recognition, comparison with our results is difficult. After technology normalization based on the constant field scaling concept [5], the throughput of this work is 1.96x higher than the previous LVQ neural network design in [2], using Manhattan distance (simpler approximation to Euclidean distance), as listed in Table 1. For the aspects of area consumption and power dissipation, the comparison should consider the trade-off between flexibility and performance since our chip has an on-chip processor and larger shared memory for prioritizing flexibility over size. The design in [2] requires communication to a host PC and an external control unit.

Table I Performance List

| | [2] | This work |
|---|---|---|
| Technology | 0.8µm | 0.18µm |
| Bit precision | 8-bit | 16-bit |
| Max dimension flexibility | 128 | 1024 |
| Parallelism | 16 | 8 |
| Max number of references | 16 | 512 |
| Storage capability (bit) | 8K | 96K |
| Distance metrics | Manhattan | Euclidean |
| Throughput ($\times 10^9$ bits/s) | 1.14 | 2.23 |
| Area (mm$^2$) | 28.58 | 4.88 |
| Power dissipation (mW) | 425 | 214 |

In conclusion, the proposed SoC for a LVQ neural network with on-chip learning and recognition capability has high performance, low power consumption and in addition large flexibility for realizing various practical applications.


## Acknowledgments

The VLSI-chip was fabricated through the chip fabrication program of VDEC, the University of Tokyo in collaboration with, Rohm, Synopsys, and Cadence. The used standard cell library was developed by Tamaru/Onodera Lab. of Kyoto Univ. and released by Prof. Kobayashi of Kyoto Inst. of Tech.


## References


[1] T. Kohone, *Proceedings of the IEEE,* Vol. 78 (9), pp. 1464-1480, 1990.

[2] M. Porrmann, U. Witkowski, and U. Rückert, *IEEE Trans. Neural Networks,* Vol. 14 (5), pp. 1110-1121, 2003.

[3] P. Ienne, P. Thiran, and N. Vassilas, *IEEE Trans. Neural Networks*, Vol. 8 (2), pp. 315-330, 1997.

[4] L. M. Reyneri, *IEEE Trans. Neural networks*, 14(1), pp. 176-194, 2003.

[5] D.J. Frank et al., *Proc. of the IEEE, Vol. 89*, pp. 259-288, 2001.