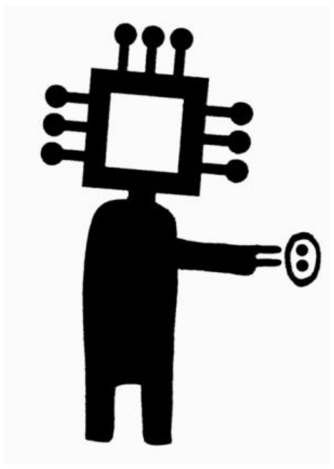
# Realtimesystems
# Character recognition using an artifical neural network in a multiprocessor environment

Marcus Ekerhult

December 6, 2006



**Abstract**

In this paper I will describe the use of Artificial Neural Networks (ANNs) and how they efficiently can be used to emulate human thinking in the domain of artificial intelligence. As an example I have built a character recognition system, using the multi-processing Parallax Propeller microcontroller. As this is a new controller (2006) I will also describe its functionallity. The outcome of the project is a complete embedded system with a PS/2 mouse driver, a VGA driver and an implementation of a simple artificial neural network. The system can recognize up to 4 different characters provided via mouse input, and is scalable with more memory attached.

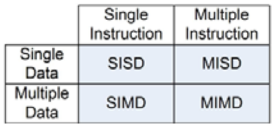
# Contents

# List of Figures

# 1 Distributed parallel processing

## 1.1 Introduction and motivation

In 1966, a man named Michael J. Flynn came up with a four-way classification of computer architectures [Figure 1]. Flynn's taxonomy distinguishes single and multi-processor computer architectures according to the number of concurrent instruction and data streams available in the architecture. The four combinations are **SISD** (single instruction stream, single data stream), **SIMD** (single instruction stream, multiple data streams), **MISD** (multiple instruction streams, single data stream), and **MIMD** (multiple instruction streams, multiple data streams). Flynns classification has become standard and is today widely used.



| | Single Instruction | Multiple Instruction |
|---|---|---|
| Single Data | SISD | MISD |
| Multiple Data | SIMD | MIMD |

Figure 1: Flynn's Taxonomy.

Traditionally, software has been written for serial computation (SISD). This technique is to be run on a single computer having a single Central Processing Unit (CPU). A problem is broken into a discrete series of instructions, the CPU gets these instructions and/or data from memory, decodes the instructions and then sequentially performs them [Figure2]. Only one instruction may be executed at any moment in time. This is how the majority of todays processors work, even though multicore processors are growing rapidly.

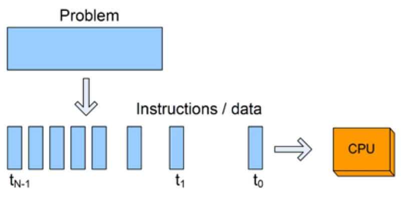


Figure 2: Serial computation.

In the simplest sense, parallel computing is the simultaneous use of multiple processors to solve a computational problem (SIMD, MISD or MIMD). A problem is broken into discrete parts that can be solved concurrently. Each part is further broken down to a series of instructions where each part can be executed simultaneously on different CPUs. This is illustrated in Figure3.

The motivations for parallel processing can be summarized as follows:

- Higher speed, or solving problems faster. This is important when applications have "hard" or "soft" deadlines. For example, we have at most a few hours of computation time to do 24-hour weather forecasting or to produce timely tsunami warnings.

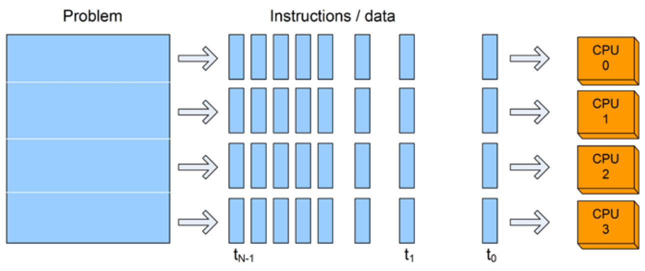


Figure 3: Parallel computation.

- Higher throughput, or solving more instances of given problems. This is important when many similar tasks must be performed. For example, banks and airlines, among others, use transaction processing systems which handle large volumes of data.
- Keeping the throughput, while lowering the clockspeed and thus also the power consumption. This is important in handheld devices and also on a broader scale in the upcoming trend, sustainable and environmental friendly design.

## 1.2 Structures

There are today a large number of differnet structures for parallel computer systems. In the following sections I will describe four simple approaches. To be able to compare these approaches, two properties have to be defined:

- The diameter ($D$) of a parallel computer system is defined as the longest of the shortest distances between pairs of processors (p).
- The maximum node degree ($d$) is defined as the largest number of links or communication channels associated with a processor (p).

### 1.2.1 Linear array of processors

The basic linear array [Figure 4] has D = p-1, and d=2. The ring variant, shown in [Figure 5], has the same node degree of 2 but a smaller diameter of $D = \lfloor \frac{p}{2} \rfloor$.

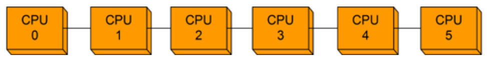


Figure 4: Example of a linear array of processors.

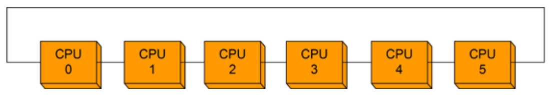


Figure 5: Ring variant of a linear array of processors.

### 1.2.2 Binary tree of processors

The binary tree in [Figure 6] is balanced in that the leaf levels differ by at most 1. If all leaf levels are identical and every nonleaf processor has two children, the binary tree is said to be complete. The diameter of a p-processor complete binary tree is $2log_2(p+1)-2$. More generally, the diameter of a p-processor balanced binary tree architecture is $2\lfloor log_2 p \rfloor$ or $2\lfloor log_2 p \rfloor - 1$, depending on the placement of leaf nodes at the last level. Unlike linear arrays, several different p-processor binary tree architectures may exist. This is usually not a problem as we almost always deal with complete binary trees. The (maximum) node degree in a binary tree is; d = 3.

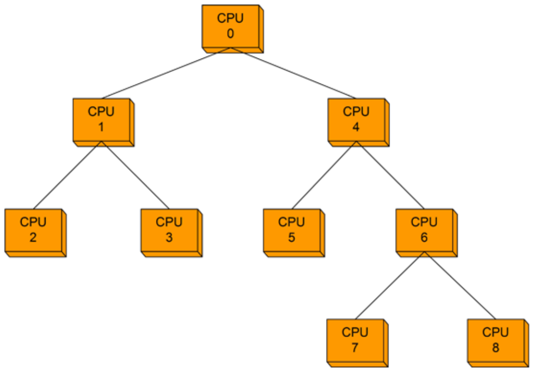


Figure 6: A balanced (but incomplete) binary tree of nine processors.

### 1.2.3 Two-dimensional mesh of processors

The diameter of a p-processor square mesh, like the one in [Figure 7], is $2\sqrt{p}-2$. More generally, the mesh does not have to be square. The diameter of a p-processor $r \times (\frac{p}{r})$ mesh is $D = r + \frac{p}{r} - 2$. Again, multiple 2D meshes may exist for the same number p of processors, e.g., $2 \times 8$ or $4 \times 4$. Square meshes are usually preferred because they minimize the diameter. The node degree for the mesh is d = 4.

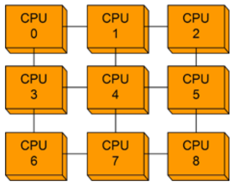


Figure 7: A 2D mesh of nine processors.

### 1.2.4 Multiple processors with shared variables

A shared-memory multiprocessor can be modeled as a complete graph, in which every node is connected to every other node, as shown in [Figure 8] for p=9. In the 2D mesh of [Figure 7], $CPU_0$ can send/receive data directly to/from $CPU_1$

and $CPU_3$. However, it has to go through an intermediary to send/receive data to/from $CPU_4$. For a shared-memory multiprocessor, every piece of data is directly accessible to every processor (we assume that each processor can simultaneously send/receive data over all of its p-1 links). The diameter D=1 of a complete graph is an indicator of this direct access.

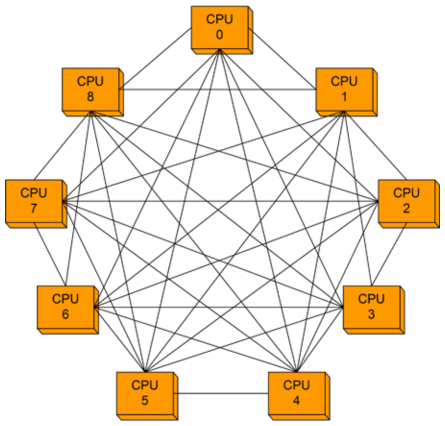


Figure 8: A shared-variable architecture modeled as a complete graph.

# 2 Artificial Neural Networks

## 2.1 Introduction and motivation

Artifcial neural networks (ANNs) can be most adequately characterised as 'computational models' with particular properties such as the ability to adapt or learn, to generalise, or to cluster or organise data, and which operation is based on parallel processing. Artificial neural networks can also be seen as simplfied electrical models of the biological nervous systems. They are built up by a very large number of processing elements (neurones), all working together to solve specific problems.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits [3]. But the technology available at that time did not allow them to do much. Today on the other hand we are able to implement very large networks, mostly due to a higher device density of VLSI chips. Still due to the fact that we know very little about the nervous system our models are necessarily gross idealisations of real networks of neurones.

## 2.2 Why use neural networks?

Neural networks have a remarkable ability to derive a meaning from complicated or imprecise data. They can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

The motivations for artificial neural networks can be summarized as follows:

- Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
- Real Time Operation: Artificial neural network computations may be carried out in parallel and carry out computations very fast.
- Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

### 2.3 Two approaches

During the years various of different network models have been developed. In this chapter I briefely describe two commonly used models.

#### 2.3.1 Feed-forward networks

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any computational unit (node) does not affect that same node. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition.

#### 2.3.2 Feedback networks

Feedback networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can be extremely complicated. Feedback networks are dynamic; their state is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.

# 3 The Propeller controller

## 3.1 Introduction

Parallax is a company that in the end of 2005 changed the market for embedded applications when they introduced the Propeller ”super-microcontroller” with eight high speed 32-bit microcontrollers inside. Each internal microcontroller (cog) has access to the Propeller chips 32 I/O pins and 32 KB of global RAM. Each cog also has its own 2 KB of RAM that can either run a so called Spin code interpreter or an assembly language program.

The Spin language is the high-level programming language created by Parallax explicitly for the Propeller chip. Cogs executing Spin code do so by loading a Spin interpreter from the Propeller chips ROM. This interpreter fetches and executes Spin command codes that get stored in the Propeller chips global RAM. Propeller cogs can also be programmed in low-level assembly language. Whereas high-level Spin tells a cog what to do, low-level assembly language tells a cog how to do it. Assembly language generates machine codes that reside in a cogs RAM and get executed directly by the cog. Assembly language programs make it possible to write code that optimizes a cogs performance (a speed factor of around 250); however, it requires a more in-depth understanding of the Propeller chips architecture.

### 3.1.1 Cogs

All cogs (numbered 0 to 7) contain the same components [Figure 9]: a Processor block, local 2 KB RAM configured as 512 longs (512 x 32 bits), two I/O Assistants with PLLs, a Video Generator, I/O Output Register, I/O Direction Register, and other registers not shown in the diagram. Each cog is designed exactly the same and can run tasks independently from the others. They also all have access to the same shared resources, like I/O pins, Main RAM, and the System Counter. One can most likely view the internal structure as a shared variable architecture.

Cogs can be started and stopped at run time and can be programmed to perform tasks simultaneously, either independently or with coordination from other cogs through Main RAM. Regardless of the nature of their use, the Propeller application designer has full control over how and when each cog is employed; there is no compiler-driven or operating system-driven splitting of tasks between multiple cogs. This method empowers the developer to deliver absolutely deterministic timing, power consumption, and response to the embedded application.

### 3.1.2 Clock

All eight cogs are driven from the same clock source, the System Clock, so they each maintain the same time reference and all active cogs execute instructions simultaneously. The chip includes an internal PLL which allows the cogs to run at a maximum speed of 80MHz. The architecture is unfortunately not pipelined so the clockspeed and throughput is not 1:1, but 4:1. This gives a maximum throughput of 160MIPS when all cogs are up and running.

### 3.1.3 Hub

To maintain system integrity, mutually-exclusive resources must not be accessed by more than one cog at a time. The Hub maintains this integrity by controlling access to mutually-exclusive resources, giving each cog an opportunity to access the resources in a round robin fashion from Cog 0 through Cog 7 and back to Cog 0 again. The Hub, and the bus it controls, runs at half the System Clock rate. This means that the Hub gives a cog access to mutually-exclusive resources once every 16 System Clock cycle. This a rather big bottleneck which slows down the execution time.

### 3.1.4 Semaphores

There are eight lock bits (also known as semaphores) available to facilitate exclusive access to user-defined resources among multiple cogs. If a block of memory is to be used by two or more cogs at the same time and that block consists of more than one long (four bytes), the cogs will each have to perform multiple reads and writes to retrieve or update that memory block. This leads to the likely possibility of read/write contention on that memory block where one cog may be writing while another is reading, resulting in misreads and/or miswrites. Because locks are accessed only through the Hub, only one cog at a time can affect them, making this an effective control mechanism. The Hub maintains an inventory of which locks are in use and their current states, and cogs can check out, return, set, and clear locks as needed during run time.

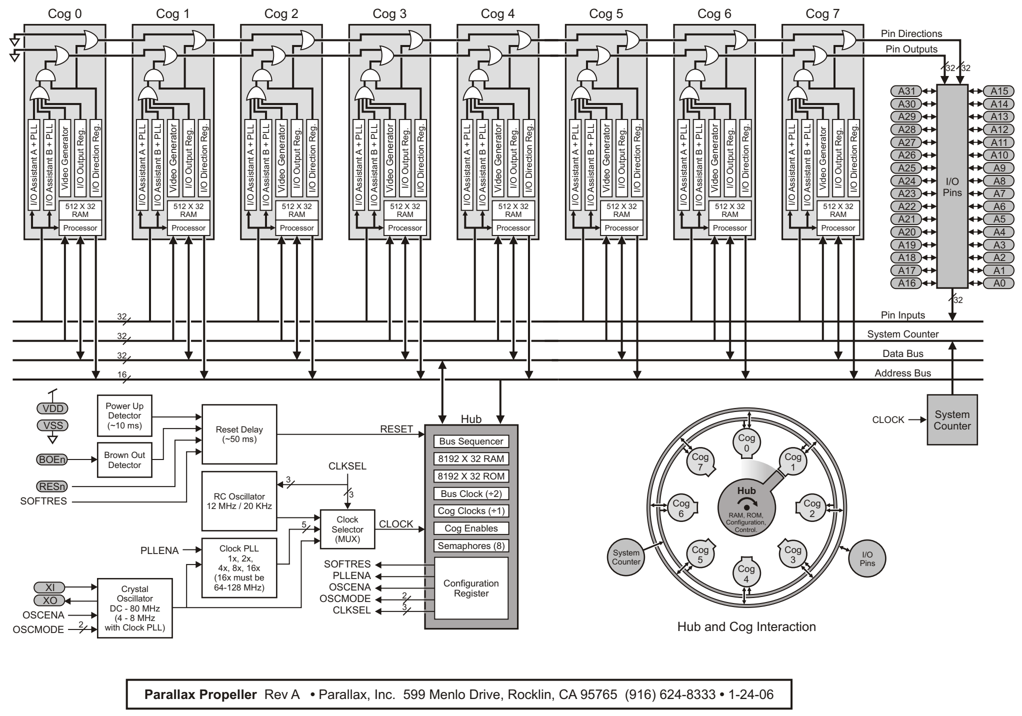


Figure 9: The Propeller architecture.

# 4 Project - Character Recognition

## 4.1 Introduction

IBM has through an anlysis shown that for a character recognition system (CRS) to be considered acceptable by most users, an accuracy of 97% or more is a must [4]. To develop a CRS that is both reliable and natural enough to be comfortable, the system must be highly adaptable. Creating software that is as adaptable as its users are unique is a very challenging problem for conventional computer algorithms. This is why many people in the field of handwriting recognition are turning to neural networks to perform the recognition processing.

In this chapter a complete CRS is implemented in the earlier described Propeller microcontroller. The system is based on a simple feed-forward neural network [Figure 10], and incapsulates a hidden layer with four nodes, thus capable of recognize up to for characters. There are two software switches ($S_1 \& S_2$), where $S_1$ determines the network state; either learning or working, and $S_2$ determines what character to gather information about. The "SELECTOR" is basically just a max function, giving the max of $Q_1 ... Q_4$.

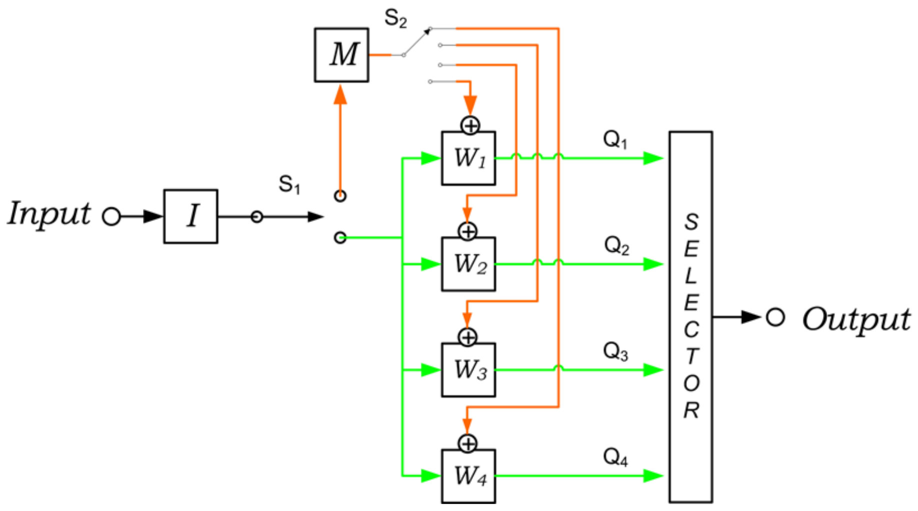


Figure 10: The developed Feed-forward network.

## 4.2 Network architecture

Here I will present a functional description over the developed network. Let us start with an empty system, which means that we first have to learn the system characters before trying to recognize them. The setup will start with $S_1$ connected to the M-box (learning mode) and $S_2$ connected to $W_1$. The I-box gets as input a 128*128 matrix of the sampled character [Figure 11]. It discretizes it into a 16*16 matrix due to memory requierments, and sends it into the M-box, which only transforms the zeroes to negative ones. $S_2$ makes the matrix flow into $W_1$ where it is added to the $W_1$ internal matrix. This procedure has to be repeated a couple of times to get a good estimation of the character.

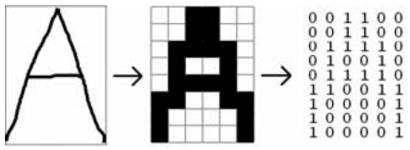


Figure 11: Discretization of the sampled character (I-box)

When we want the system to recognize characters we connect $S_1$ with the four nodes (working mode). The I-box will work just as when we used it in learning-mode, hence the four nodes will in parallel recieve the I-matrix. The

W$_k$ nodes include both the matrix defining a character and also a computational algorithm, defining its output:

$$Q_k = \frac{\sum_{i=1}^{x} \sum_{j=1}^{y} W_k(i,j) * I(i,j)}{\sum_{i=1}^{x} \sum_{j=1}^{y} \{W_k(i,j) > 0\}}$$

This gives a measure of how well the recognition system identifies an input pattern as a matching candidate for one of its learnt patterns. The greater the value of $Q_k$, the more confident does the system bestow on the input pattern as being similar to a pattern already known to it.

## 4.3 Implementation

The network described in the previous chapter were implemented on a development board from Parallax [Figure 12]. The application is written in SPIN with some additional assembly code for hardware drivers (VGA & PS2). The main code runs in a big neverending loop that includes a statemachine for defining what the program should do; learning or working mode.

One of my goals was to run all four nodes in there own processors, making the network completely parallel. I didn't realize that the hardware drivers would in the end consume four cogs, which made that goal not achievable. Instead I let the mouse sampling and drawing functions reside in its own processor, which gives a much smoother graphical line when inputing a character. The total cog usage can be seen in [Figure 13].

The VGA and graphic driver was originally devloped by the Parallax development team, and can be fetched through the forum [5]. It has been optimized and slimmed down to a minimum to be able to fit in my project. When drawing a character, one is drawing into a (global) memory area (8 kB), from which the I-box gets its input. In [Figure 14] an "A" has been drawn and also recognized by the system.

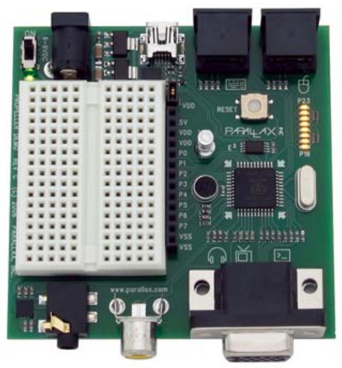


Figure 12: The development board that was used.

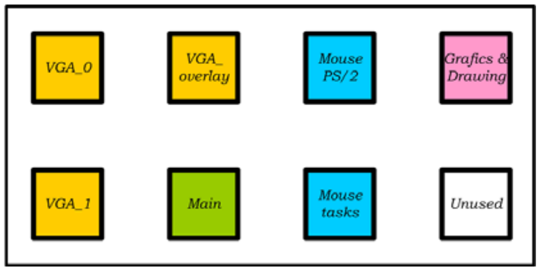


Figure 13: Cog usage.

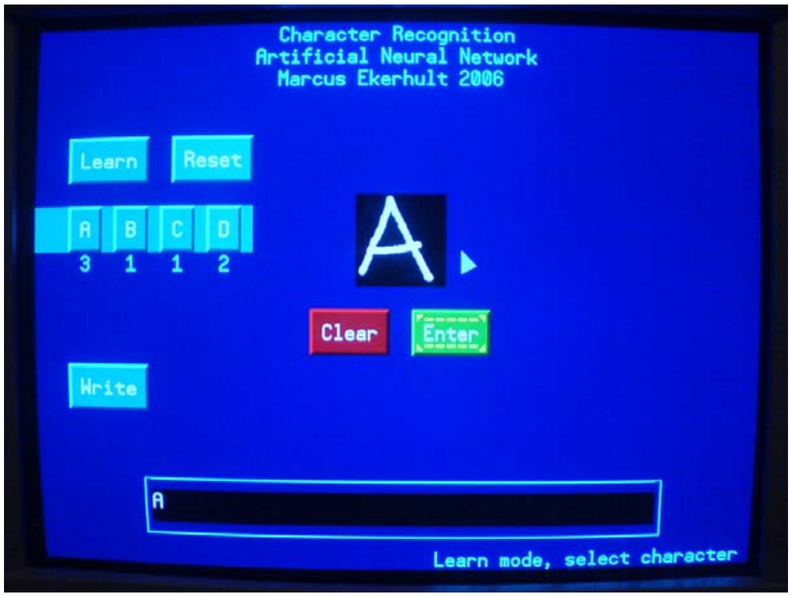


Figure 14: The GUI displayed on a VGA screen.

## 5 Conclusions

Artificial neural networks are very well suited for real time systems due to their fast response and computational speed which are due to their parallel architecture. Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks that are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency. The implementation described in this report was succesfull and will hopefully help others in their understanding of ANNs.

## References

[1] Dave Prochnow, Experiment with Artificial Neural Networks, Tab Books, 1988

[2] Behrooz Parhami, Introduction to Parallel Processing, Kluwer Academic Publishers, 2002

[3] Ben Krose Patrick & van der Smagt, An introduction to Neural Networks, The University of Amsterdam, Eighth edition November 1996

[4] Alexander J. Faaborg, Using Neural Networks to Create an Adaptive Character Recognition System, Cornell University May 2002
December 2006 avaliable at: `http://alumni.media.mit.edu/~faaborg/research/cornell/hci_neuralnetwork_finalPaper.pdf`

[5] Parallax Propeller Forum
`http://forums.parallax.com/forums/default.aspx?f=25`