# Self-organisation and its application to binding

by

Ahmed Hemani

Dept. of Electronic System Design, Royal Institute of Technology

100 44, Stockholm, Sweden.

email: ahmed@inmic.se

ABSTRACT: *This paper presents Kohonen's self-organisation algorithm as an optimisation tool. Its application is illustrated by applying it to a high-level synthesis(HLS) problem - binding: the task of assigning operations to specific instances of functional units. It is a crucial problem, as it influences the interconnect, wiring and register cost. This Self-Organising Binder (SOB) has a built-in hill-climbing mechanism, and being based on neural-network it can be easily parallelised. We apply SOB to benchmark examples and show that the results are comparable to the best reported in the field.*

KEYWORDS: binding, high-level synthesis, optimisation, self-organisation, neural-networks.

## 1.1 Introduction

The recent surge in the interest in neural networks has seen development of some promising optimisation techniques; prominent among these optimisation techniques are the Hopfield's algorithm [Hop86] and the Kohonen's self-organisation algorithm [Koh88]. The VLSI CAD community has also taken note of these optimisation techniques, and legitimised it by a special session in the 26th Design Automation conference(DAC) to evaluate the capabilities of neural networks to solve the layout optimisation problems. Three papers, [Ran89], [Yih89], and [Yu89] were presented, and all of them used Hopfield's algorithm. [Kim89] and [Hem1-90] reported use of Kohonen's algorithm to solve the cell placement problem. The author has also reported the use of Kohonen's for scheduling in [Hem89], [Hem2-90]. Further, self-organisation based scheduling and binding algorithms are the focus of author's thesis [Hem92].

Hopfield's algorithm, though explored more than Kohonen's algorithm, suffers from two problems: 1) in its original form, it works as a steepest descent algorithm and 2) improvement in the quality of the result comes at the expense of increase in percentage of illegal solutions, and vice-versa.

This paper presents Kohonen's algorithm as a general optimisation technique and uses *binding*, one of the optimisation problems in HLS, as an example. The following factors motivated us to use Kohonen's algorithm:

- It has a built-in hill-climbing mechanism. This overcomes the limitation of Hopfield's algorithm.
- Compared to simulated annealing, which is known for its hill-climbing mechanism, Kohonen's algorithm can be implemented for much faster execution. There are two reasons for this: 1) in simulated annealing, at every iteration a cost function has to be evaluated. Usually, the better the desired performance, the more expensive the cost function is to evaluate. In contrast to this, the cost function in Kohonen's algorithm is embedded as one of the characteristics of its outcome, i.e., uniform distribution and clustering. 2) Kohonen's algorithm is inherently parallel in nature, so it would be relatively easier to parallelise compared to simulated annealing. However, what makes self-organisation faster, also makes it less versatile than simulated annealing. Simulated annealing allows an arbitrary objective function, enabling it to be applied to a wide range of optimisation problems, whereas self-organisation is restricted to those problems whose objective function can be related to the outcome of the self-organisation process.
- It can be easily adapted to guarantee that it always produces a legal solution.
- It has a complexity of $O(k_{max}.n^2)$, where $k_{max}$ is the number of control steps and $n$ is the number of operations in the problem. The complexity is derived in [Hem92].

Compared to the above advantages that SOB has, by virtue of being based on Kohonen's self-organisation algorithm, the other approaches to the binding problem have some shortcomings in the optimisation characteristics of the algorithms they use. Following is a review of some of the well known binding approaches.

- [HAL88] has a greedy approach to binding, but it takes the register usage into account, which helps improve the quality of the result. But being greedy, the binding algorithm is prone to locally minimum solutions.

- [FACET86] uses a clique partitioning algorithm to solve the binding problem, register allocation and interconnect allocation. But as clique partitioning is an NP-complete problem, it uses heuristics to cut down the search space. As is the case with heuristics, they work well for most problem instances but not for all of them, and do produce sub-optimal solutions.
- [SPLICER88] uses a branch and bound search technique for doing binding, register allocation and interconnect allocation tasks. Rather than do each task individually, it binds operations in one control step and then immediately does register and interconnect allocation. It is possible to incorporate look ahead to improve the quality of result, but the author of the algorithm concluded that deeper look-ahead did not always pay off. The lack of hill-climbing mechanism makes this approach also susceptible to locally minimum solutions.
- [Devdas89] incorporates a hill-climbing mechanism by using the simulated annealing algorithm and models the entire problem of datapath synthesis as a two dimensional placement problem. The results achieved by this algorithm are good, but the execution time tends to be long. The execution time for simulated annealing (as reported in [Devdas89]) is an order of magnitude slower than the self-organised based scheduling and binding algorithms, when executed on sequential computers of comparable speed, see end of section 1.5.1.

## 1.2 Self-organisation as an optimisation tool

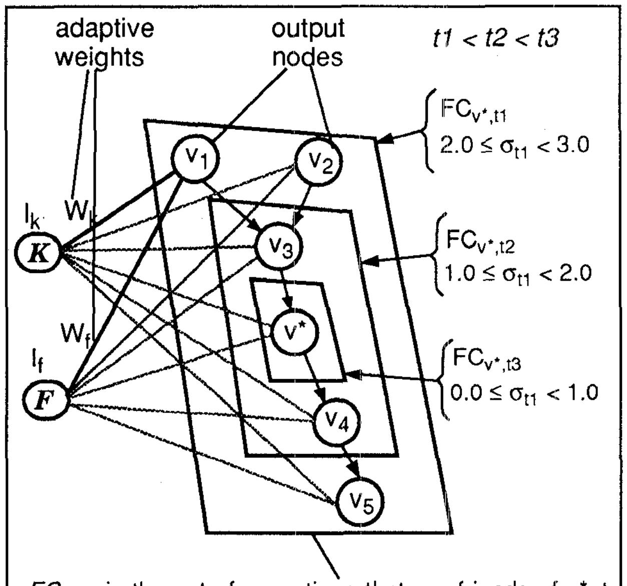
Lack of space does not allow the presentation of generic Kohonen's self-organisation algorithm and justification of its three optimisation characteristics: hill-climbing, uniform distribution and clustering. However, the utility of the three characteristics is briefly discussed below:

- The benefit of hill-climbing as a characteristic in an optimisation algorithm is easy to appreciate: it prevents the algorithm from getting stuck in locally minimum solutions.
- Uniform distribution can be used in optimisation problems where a cost function can be minimised by balancing the value of one variable of the objective function across another. For instance in scheduling, resource usage is minimised by uniformly distributing operations across control steps. [HAL88] exploits this fact and uses a force directed scheduling algorithm to achieve uniform distribution. Employing the same fact, the author has formulated scheduling as a self-organisation problem [Hem89], [Hem1-90], [Hem92]; the scheduling algorithm, as one might expect, is called the Self-Organising Scheduler (SOS).
- Clustering can be used in optimisation problems where the cost function can be related to the clustering of objects in proportion to some proximity measure between objects. A direct application of this characteristic is to the placement problem, where clustering in proportion to connectivity of cells results in minimisation of the total wire length. [Kim89] and [Hem2-90] have reported the use of self-organisation for cell placement. Binding also exploits the clustering characteristic of the self-organisation process. This is discussed in detail in the next section.

## 1.3 Binding as a self-organisation problem

In this section we will formulate binding as a self-organisation problem.

### 1.3.1 Network



$FC_{v^*,t1}$ is the set of operations that are friends of v*at iteration t1, this set is controlled by the process parameter - friendship circle $\sigma_t$ at iteration t.

Fig 1. Organisation of the neural network used by SOB.

The neural network used by Kohonen's self-organisation algorithm (see Fig 1.) has three components.

1. Input Nodes

Input nodes correspond to the dimensions of the output space, or the design space in terms of the binding problem. There are two dimensions of the design space: the time dimension and the resource dimension. Correspondingly, there are two input nodes: the time node, denoted *by K* for control steps, and the resource node, denoted by *F* for functional units(FUs). A point in the design space is represented by an input vector $(I_k, I_f)$ at the input nodes.

2. Output Nodes

Output nodes correspond to the objects we want to self-organise in the output space. In terms of the binding problem, these correspond to the operations we want to bind, i.e, one output node for every operation to be bound. These are shown in Fig 1. by small circles with operations $v_x$ inside them. The arrows show the dependencies between them.

3. Adaptive weights

Every output node is connected to the two input nodes by a pair of adaptive weights $(W_k, W_f)$. These adaptive weights serve two purposes: a) they allow all output nodes to evaluate the input vector simultaneously, b) specify the position of output nodes or operations in the schedule space. As we simulate the self-organising process on sequential computers, the latter role of specifying position is more significant from our viewpoint. Moreover, in this role, these weight pairs specify the state of the self-organising process at any instance and indeed the final result, i.e, the binding. For each operation, the $W_k$ weight specifies the control step in which it will be executed, determined by scheduling, and the $W_f$ weight specifies the FU type and instance to which it is bound.

Besides these three components, there are two important concepts related to the self-organisation that must be defined in terms of the binding problem.

1. Friends:

In binding we define friendship as follows:

- two operations are friends, if they are dependent. This *qualifies* friendship.
- the friendship between two friends is inversely proportional to the dependency distance between them. This *quantifies* friendship. Dependency distance between two operations is defined as the number of data edges between them in the data flow graph representing the behaviour to be synthesised. Because dependency distance quantifies friendship, it is also called friendship distance.

2. The friendship circle:

The friendship circle, a process parameter, controls how many friends an operation has at any instance (iteration). Friendship circle is also defined in the same terms as the degree of friendship: for an operation $v^*$, a friend $v$ is still a friend, if the friendship distance between $v$ and $v^*$ is less than the current value of the friendship circle. This is shown in Fig 1. where the polygon enclosing operations represents the friends that the operation $v^*$ has at a particular iteration. The shrinking polygon size with increasing iteration represents the fact that $v^*$ gradually loses friends, the more distant ones being the first to go and then the closer ones. The friendship circle is large in the beginning and it shrinks at the rate determined by the complexity and the size of the problem. Informally, the more complex the problem, the slower the rate at which it shrinks.

### 1.3.2 Design space and constraints on movement

The output space is linear in Kohonen's algorithm, whereas the design space for binding is discrete. To resolve the incompatibility, we treat the design space as linear. In the control step dimension, each control step $(k_1, k_2, ... k_{max})$ is given a resolution, which naturally corresponds to the length of the clock period (see Fig 2.). In the FU dimension, each FU type $(t_1, t_2, ... t_{max})$ is similarly given a resolution. This resolution is called type frame: a contiguous segment in resource dimension corresponding to each FU type(see Fig 2.). The width of the type frame is calculated from the estimated number of units required for the FU types, a step done during the allocation phase in SYNT(a HLS system, of which SOS and SOB are components).

Objects to be self-organised are free to move in the entire

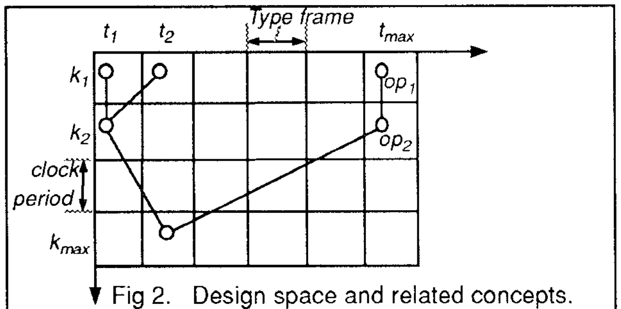


Fig 2. Design space and related concepts.

output space in the Kohonen's algorithm. In SOB however, an operation's movement in the resource dimension is restricted to its type frame, as an operation's position in resource dimension specifies the type and instance of FU type that would service it. As binding is done after scheduling in SYNT, operations are not allowed to move in the control step dimension.

### 1.3.3 Binding by self-organisation

The objective of the binder is to minimise the costs of interconnect and wiring. These costs are directly influenced by how the operations are bound to the instances of

FU types. For example, operations sharing data, if bound to the same instance, would require multiplexers with fewer inputs and less wiring as compared to if they were bound to different instances. Another way to look at the binding problem is to treat it as a multi-partitioning problem, i.e., partition the control and data flow graph into as many partitions as the number of units decided by the scheduler, such that the sum of connections between the partitions is minimised. Though the scheduler puts a lower bound on the number of registers required, binding also influences register usage: HLS systems often allocate more registers than the lower bound to bring down the cost of mux-inputs. By minimising the number of interconnections between partitions, the binder can reduce the need to allocate the extra registers.

As operations move only in the resource dimension during binding, when the scheduled graph is self-organised by SOB, operations sharing data become clustered in the resource dimension, i.e., their $W_f$ weights have similar values. This has the effect of aligning such operations in the resource dimension. (EQ 1.1) can then be used to extract the instance, $n_{ti}$, of the FU type, $t_i$, that would service these operations. Obviously, if the values of $W_f$ are *similar enough* for a group of operations, (EQ 1.1) would yield the same instance of FU type, due to the rounding effect. Consequently, for each FU type - $t_i$, operations to be serviced by it are partitioned into $N_{ti}$ groups, where $N_{ti}$ is the *number of units required of the FU type*, $t_i$, and is decided by scheduling.

$$n_{ti} = \frac{W_f \; Modulo \; type \; frame \; width}{N_{ti}} + 1 \qquad (EQ\ 1.1)$$

## 1.4 The self-organising binder - SOB

In this section we describe each step of the SOB algorithm.

Step 1: Set process parameters

There are three process parameters that must be assigned an initial value at the beginning of the process. These are: 1) the initial friendship circle - *i_FC*, 2) the current friendship circle - *c_FC*, which is updated at every iteration and 2) the amount by which *c_FC* must be shrunk at each iteration - $\Delta FC$. These are set as follows:

- *i_FC:* This parameter together with D*FC* decides the number of iterations the process goes through before converging and the number of hill-climbing moves the process has at the beginning of the process. This is set to half the *critical path:* the largest dependency distance between any two operations.
- *c_FC:* Initially this has the same value as *i_FC*.
- $\Delta FC$*:* This is made inversely proportional to the complexity of the graph. The complexity of a graph is dependent on two parameters: 1) the potential parallelism and 2) the freedom operations have.

Step 2: Generate input vector.

The following equations are used to generate the input vector - $(I_k, I_f)$ - with uniform distribution over the design space.

```
$I_k$ := random () MODULO length of time dimension;
$I_f$ := random () MODULO width of resource dimension;
```

Step 3: Set up a competition.

A competition is set up among operations who are eligible, i.e., operations whose time frame(range of legal control steps for an operation) includes $I_k$ and this type frame includes $I_f$. The winner, $v*$, is the operation whose distance to the input vector is minimum. Algorithmically:

```
FOR every eligible operation - v DO
    distance := $|W_{kv} - I_k| + (W_{fv} - I_f|$;
    IF distance > winning_distance THEN
        winning_distance := distance;
        $v* := v$;
$v*$ is the winner;
```

Step 4: Move the winner and its friends.

In this step the winner and its friends, decided by *c_FC*, are moved towards the Input vector. The amount by which they move is decided by gain, a function of iteration and friendship distance, and is defined by (EQ 1.2). As can be seen gain is large in the beginning, when *i_FC* is equal to *c_FC* and then gradually decreases with *c_FC*. Besides, it also decreases with the friendship distance - $fd_{v*,v}$ - between the winner, $v*$, and its friend $v$. The smaller the friendship distance, the larger the gain.

$$gain_{v^*,v} = \frac{1}{((i\_FC - c\_FC) + 2) \cdot fd_{v^*,v}} \qquad (EQ\ 1.2)$$

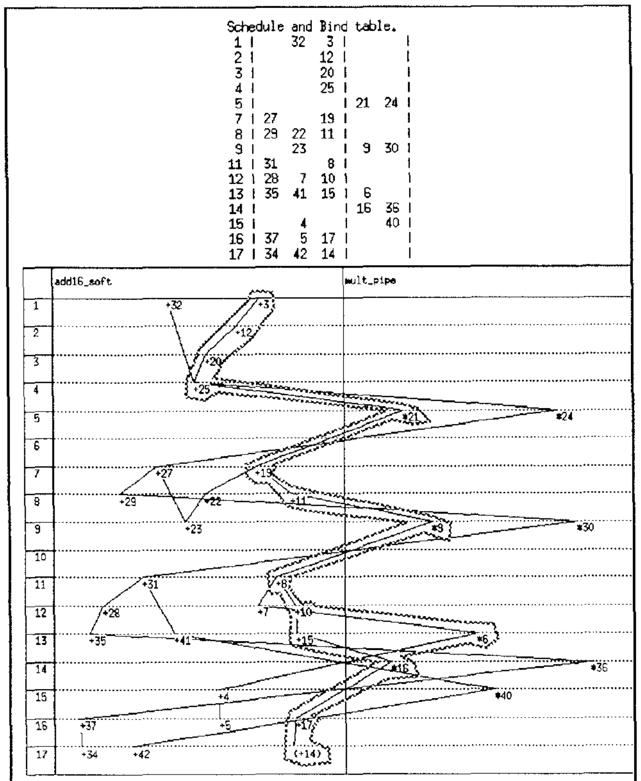
As operations move only in the resource dimension, *only the $W_f$ weight needs to be adapted.* Algorithmically:

```
FOR every friend v of the winner v* DO
    IF friendship_distance$_{v*,v}$ < c_FC THEN
        $Wf_v := Wf_v + gain_{v*,v} (I_f - Wf_v)$;
```

Step 5: Update process parameters.

This simply amounts to reducing *c_FC* by $\Delta FC$, and checking that if *c_FC* is less than two, the process is terminated. The reason for terminating the process when *c_FC* is less than two is that if the friendship circle shrinks to less than two, then only primary moves are possible, which results in operations getting misaligned.

## 1.5 Examples



```
Schedule and Bind table.
 1 |      32   3 |        |
 2 |          12 |        |
 3 |          20 |        |
 4 |          25 |        |
 5 |             | 21  24 |
 7 | 27       19 |        |
 8 | 29   22  11 |        |
 9 |      23     |  9  30 |
11 | 31        8 |        |
12 | 28    7  10 |        |
13 | 35   41  15 |  6     |
14 |             | 16  36 |
15 |       4     |     40 |
16 | 37    5  17 |        |
17 | 34   42  14 |        |
```

Fig 3. Binding results of the 5th order filter, using 17 control steps.

### 1.5.1 Fifth order elliptic filter

This example was used as a benchmark in the 1988 work shop on high level synthesis. It has 43 operations and 60 dependency constraints. Because the code is flat, the potential parallelism is high and thus constitutes a difficult scheduling and binding problem. The commonly applied allocation constraint is the use of either a pipelined multiplier that has a propagation delay of two clock periods and a latency of one clock period or an ordinary multiplier with a propagation delay of two clock periods. The other type of FU used is an adder that has a propagation delay of one clock period. Three designs are usually quoted and are created by using 17, 18 and 19 control steps. The binding result for the 17 control step case is shown in Fig 3.. As binding influences register usage and number of mux-inputs, these parameters are compared, though strictly speaking these parameters are also influenced by scheduler, register and interconnect allocator tasks. As the scheduler in the SYNT system is also implemented using the self-organisation algorithm, its influence in the results being compared would be helpful in judging the efficacy of self-organisation as an optimisation tool. TABLE 1.1 compares the results of SOB to those of [HAL88] and [ELF90]. The following observations can be about the binding results:

- The bind table (See Fig 3.), shows that the filter has been partitioned into five groups: three groups of addition operations and two groups of multiplication operations. These are shown as columns in the table. To show the correspondence between the self-organised graph and the bind table, one of the data streams that has been clustered is shown encircled by a shaded polygon.This clustered data stream corresponds to the third column of add operations and the first column of multiply operations. An interesting trade-off that SOB has made can be seen in control step 16, where operation +*5* would like to be in the same column (group) as +*17*. The reason is that, +*5* is in the second column where it has fewer friends than it has in the third column. However, +*17* has more friends than +*5* in the third column, and thus merits being there.
- The number of registers and the mux input used by SYNT(See TABLE 1.1) are comparable with [HAL88] and [ELF90]. SYNT uses one register less than [HAL88] at the expense of slightly more mux inputs, except for the 18 control step case where it uses one mux-input less and one register less. This results are achieved inspite of the fact that SYNT uses a simple mux model for interconnect units, whereas [HAL88] uses a bus model, that requires less mux-input than the mux model. Similarly, [ELF90], which uses sophisticated hierarchical interconnect model also has comparable results to SYNT. For the 19 control step case it uses the same number of registers but more mux-inputs, whereas for the 17 control step case it performed better than SYNT, primarily because of the sophisticated interconnect model it uses.

TABLE 1.1 Results of the fifth order elliptic filter.

| | HAL88 | | | SYNT | | | ELF90 | |
|---|---|---|---|---|---|---|---|---|
| control steps | 19 | 18 | 17 | 19 | 18 | 17 | 19 | 17 |
| adder | ++ | +++ | +++ | ++ | +++ | +++ | ++ | +++ |
| multiplier-pipelined | * | * | ** | * | * | ** | * | ** |
| registers | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| nr. of mux-inputs | 26 | 34 | 31 | 28 | 33 | 35 | 30 | 28 |

For the 19 control step case, only one multiplier is used, thus SOB need not partition multiplication operations, as they are all executed on the same multiplier; however, there are two adders and thus SOB is required to partition the addition operations into two groups with a minimal of interconnection between them. The success with which SOB partitions the add operations into two groups can be judged from the fact that only two registers are needed for exchange of data between the two adders(data path not shown due to lack of space). This localisation of registers to FUs, not only minimises the mux cost, but also translates into shorter, local interconnections, thereby minimising routing cost.

Though [Devdas89] did not report binding results (number of regs. and mux-inps) for this example, it is interesting to compare the execution times of simulated annealing and self-organisation based scheduling and binding algorithms. For the 17 control step case, the simulated annealing algorithm took 240 secs on VAX 8650, whereas the sum of execution times for SOS and SOB is 10 seconds on a SPARC IPC.

### 1.5.2 The differential equation example

This example was first presented in [HAL86], and later on used by [HAL88], [SPLICER87] and [CATREE87]. The commonly applied constraint is to use a multiplier and other FU types that take a clock period and the other constraint is to use a pipelined multiplier that takes two clock periods and has a latency of one clock period. Using a pipelined multiplier allows the designer to use one multiplier and a clock period that is shorter than the one when regular multiplier is used. Effectively, global time constraints remains the same, whereas the cost of multiplier goes down by almost half. TABLE 1.2 compares the results of the SYNT system to that of [HAL88] and [SPLICER87]/[SPLICER88]. For the pipelined multiplier case both [HAL88] and SYNT has same results, this is to be expected as the [HAL88] datapath for the pipelined multiplier is dominated by mux interconnect units. However, in case of a regular multiplier where [HAL88] uses buses, the SYNT results show more mux-inputs. The same arguments could be applied to comparison with [SPLICER87]/[SPLICER88].

TABLE 1.2 Results of the differential equation example.

| | HAL88 | | SYNT | | SPLICER88 | |
|---|---|---|---|---|---|---|
| clock period (ns) | 100 | 50 | 100 | 50 | 100 | 50 |
| control steps | 4 | 8 | 4 | 8 | 4 | 8 |
| multiplier-regular | ** | | ** | | ** | |
| multiplier-pipelined | | ** | | ** | | ** |
| registers | 5 | 5 | 5 | 5 | 5 | 5 |
| nr. of mux-inputs | 14 | 13 | 10 | 13 | 11 | 16 |

### 1.6 Summary

We have presented: 1) The advantages of Kohonen's self-organisation algorithm over Hopfield's algorithm and simulated annealing. 2) Reviewed some of the state-of-art binding approaches as compared to the binding algorithm proposed in this paper. 3) Brief discussion of the optimisation characteristics of the self-organisation algorithm. 4) The formulation binding as a self-organisation problem and then discussed how the objective of binding is realised by self-organisation. 5) A detailed description of the steps involved in the self-organising binding algorithm, and lastly 6) Benchmark examples to compare the results of the binding algorithm to other well known systems.

### 1.7 References

[CATREE87] C.H. Gebotys, M.I. Elmasry, "A VLSI Methodology with testability constraints.", Proceedings of the 1987 Canadian conference on VLSI, Winnipeg, Oct. 1987.

[Devadas89] Srinivas Devadas, R. Newton, Algorithms for Hardware allocation in Data path synthesis. IEEE transactions on CAD, July 1989.

[ELF90] Tai A Ly, W Lloyd Elwood and Emil Girczyc, A generalised interconnect model for datapath syntehsis, Proc. of the 27th DAC, 1990.

[FACET86] C.J. Tseng and D.P. Siewiorek, Automated synthesis of Datapath in digital systems, IEEE transactions on CAD, July 1986.

[HAL88] P. G. Paulin,High level synthesis of digital circuits using Global Scheduling and binding algorithms. Phd thesis, Carleton University, Canada. 1988.

[Hem89] Ahmed Hemani, Adam Postula, NISCHE: Neural net inspired scheduling algorithm. Fourth international workshop on high level synthesis, Oct, 1989, Kennebunkeport, Maine, USA.

[Hem1-90] Ahmed Hemani, Adam Postula, Cell placement by Self-organisation, Neural networks, Vol. 3, 1990.

[Hem2-90] Ahmed Hemani, Adam Postula, A neural net based Self-organising scheduling Algorithm, EDAC Glasgow, March, 1990.

[Hem92] Ahmed Hemani, High-level synthesis of synchronous digital systems using self-organisation algorithms for scheduling and binding, Phd thesis, submitted to Dept. Of Electronic system design, Royal Inst. of Technology, 1992.

[Hop86] D. Tank, and J. J. Hopfield, Simple neural optimisation networks: An A/D converter, Signal decision circuit, and a linear programming circuit.. IEEE transactions on circuits and systems, Vol. CAS-33, No. 5, May 1986.

[Kim89] Sung-Soo Kim, Ching-Min Kyung,Circuit Placement in Arbitrarily-Shaped Region using Self-organisation. ISCAS '89.

[Koh88] T. Kohonen, Self-organisation and associative memory. Second edition. Springer Verlag. 1988.

[Ran89] Ran Libeskind-Hadas, C.L. Liu, Solutions to module orientation and rotation problem by Neural computation networks, Proc. of the 26th DAC, 1989.

[SLICER87] B.M. Pangrle, D.D. Gajski, Slicer: A state synthesiser for intelligent Silicon Compilation, Proc. ICCD, 1987.

[SPLICER88] B. M. Pangrle, Splicer: A Heuristic Approach to Connectivity Binding, Proc. of the 25th DAC, 1988.

[Yih89] Jih-Shyr Yih and Pinaki Mazumdar, A neural network design for circuit partitioning. Proc. of 26th DAC, LasVegas, U.S.A. 1989.

[Yu89] Meng-Lin Yu, A study of the Applicability of Hopfield Decision Neural Networks to VLSI CAD. Proc. of 26th DAC Las-Vegas, U.S.A. 1989.