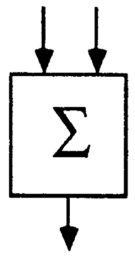
Connectionist architectures

John G. Cleary
Dept. Computer Science, University of Calgary,
2500 University Dr., N.W. Calgary, Alberta, T2N 1N4, Canada.



Abstract

Many schemes for computing have been proposed and variously described as distributed, connectionist, neuron-like and massively parallel. These are difficult to implement in current technology because at the circuit level they imply very complex wiring or switching and at the system level they cannot be efficiently simulated on a traditional von Neumann computer. A VLSI based architecture which avoids some of these problems is proposed. The basic operation implemented is the weighted summing of signals from a large number (~ 1000) inputs using the Σ-chip. Signals are heavily multiplexed, and the weighting coefficients are stored in shift registers on the same chip that does the summing. The result is a VLSI chip which has two ingoing signal pins and one outgoing. As an example a system using the Σ–chip is described which solves the n–Queens logical puzzle. Its application in chess playing is also considered briefly.

I. Introduction

Since the very beginning of computing in the late 1940s there has been interest in a class of computer architectures radically different from the traditional von Neumann style. The major characteristic of these systems is a high degree of overt parallelism with many simple computational units heavily interconnected by low bandwidth communication lines. It is this very broad class of systems whose performance is dominated by their interconnections that I will refer to as "connectionist systems". These systems are of interest because they seem well suited to solving problems of pattern recognition and of adaptation and learning. Also they seem more characteristic of the type of processing in the brain. Unfortunately, to implement such systems using modern technology (here equated with VLSI electronics) poses a considerable challenge. The great deal of connectivity between different units is essential to such systems and implies a lot of wiring to interconnect them. Wiring, however, is the dominant cost in VLSI systems (Sutherland, 1977). The scale of the problem may be appreciated when considering the interconnections in the brain. A human brain has about 10^{11} neurons. Depending on the type of neural cell and the part of the brain an individual cell may have from 1,000 to 100,000 synaptic connections to it .

For someone attempting to recreate a brain in VLSI technology there are (at least) two major problems; the provision of active computing elements corresponding to synaptic connections; and the provision of the interconnections themselves. The fundamental restrictions imposed by VLSI technology are twofold. First that within a chip (or wafer) wiring is essentially two dimensional with only a small number of layers available for wires to cross each other. Second, that only a very small number of wires can be brought off a chip (typically less than 100) and that routing and wiring off chip is much more expensive than that within a chip. The major advantage the brain has is that it can perform three dimensional wiring (although the convoluted folding of the neo-cortex seems to be an attempt to deal with a two dimensional pattern of wiring at a relatively gross level of organization). If this were all that there were to the comparison between the brain and VLSI technology the prospect for a neural style of VLSI computer would be poor. However, VLSI technology has one huge advantage over the brain, it operates much more quickly. Single gate delays are typically of the order of tens of nanoseconds while in the brain times between pulses are typically of the order of milliseconds, a difference of five orders of magnitude. The approach taken here is to trade the difficulty of doing wiring in VLSI against its speed, by multiplexing signals on wires.

Early work on connectionist structures such as the Perceptron focussed on such topics as the ability of such systems to learn to do "anything". This sprang from very early work by (Hebb,1949), (McCulloch,1943) drawing heavily on analogies from the brain. Even von Neumann (1945) spoke of the new designs for stored instruction computers in terms of distributed processing and neural structures. This early context is well summarized in (Hinton,1981) Perceptrons have been largely ignored by the main stream of AI since the work of Minsky and Papert (1968) which showed that in practice they were incapable of learning all tasks. However, recent work (Hinton, 1981) has focussed not on the universal properties of these systems but on their practical importance. They have many useful features that seem at best very awkward in the context of von Neumann computers. For example Hinton (1984) and Kohonen (1984) discuss the ability of connectionist systems to implement content-addressable memory, automatic generalization, learning, and robustness in the face of input errors. One of the important things about these properties is that they appear as primitive properties of connectionist systems, they do not need to be grafted on top of an existing structure. Another important property is that they make intensive and efficient use of the available hardware, Gaines(1969) has also pointed out that this will be necessary if a number of very difficult computing problems are to be solved. This point about use of hardware may be seen in context by noting that most of the circuitry in von Neumann computers is tied up in memory chips each part of which is almost never used and furthermore that as the size of the memory grows less and less use is made of any particular part of the memory.

In the next section the basic architecture of the Σ chip is described. The following sections then consider a simple variant of the chip to solve logical constraint problems (using the n-Queens problem as an example).

II. Σ-chip architecture

Basic architecture

As mentioned above the major cost in VLSI systems is wiring between chips. To alleviate this the Σ-chip multiplexes signals onto a single wire. For the moment I assume that all information is carried as single bits (rather than as parallel words). Without multiplexing each Σ-chip could be thought of as having some large number of lines, say 1000, as inputs and approximately the same number as outputs. On each clock cycle all the outputs are computed as a function of the inputs. As the aim of multiplexing is to reduce the number of signal wires the question arises as to the smallest number of wires that can be used. The simplest scheme which also allows outputs of different chips to be recombined is to have two input wires and one output wire but other values for the number of input wires such as four and nine are also interesting This point is addressed further below). The other basic question that needs to be answered is the degree of multiplexing. This turns out to be a compromise between the needs of individual problems, and the cost of implementation in hardware. This question will be readdressed at the end of this section but for the moment 512 will be used as a likely value. That is each Σ-chip has 1024 virtual input wires and 512 virtual output wires.

The basic calculation to be performed by the Σ-chip is a linear weighted sum of its inputs. That is, each of its 512 outputs is a different weighted sum (a much more general class of possible computations will be considered below). Thus the chip must contain two things, somewhere to store the weights for the sums and somewhere to accumulate the sums. On the Σ-chip the weights are stored as values circulating in shift registers, this provides a compact and simple storage scheme and as will be seen allows convenient access to the values while accumulating the sums. Figure 1 shows the layout of the Σ–chip, the weights in the shift registers occupy the bulk of the chip. Down one side of the chip runs space for accumulating sums and distributing the inputs. So as to avoid long signal wires driving many devices the inputs are distributed by another two shift registers. Similarly the signals for the output line are placed on a shift register and moved off the chip in sequence.

Figure 2 shows the relationship between the input and output values for a simple system where the values are multiplexed in groups of four. The figure steps through one complete cycle of input values, from time 1 to time 4. (The numbers given in the figure are markers showing the progress of values through the shift registers not actual values). At time 1 the first inputs are moved into the shift registers and the output value, which eventually will appear as the output at time 1 on the next cycle, begins its journey down the output shift register. Simultaneously the two weights needed for multiplying with the first input values are available to be read from the memory shift registers. At the next time 2 the previous inputs and the output have moved down the shift registers and the appropriate weights are again positioned to be read from the memory for the inputs received at both times 1 and 2. At this time no output need be calculated. By time 4 the first inputs have reached the end of the registers and the output computed at time 1 is ready to be output on the next clock cycle. During the next full cycle the remaining outputs are accumulated and appear on the chips output. Of course this activity is overlapped, there is always a value being placed on the output line, always an input being received, and always a weighted product being accumulated at each position through the chip.

The whole chip can thus be divided into 512 slices. Each slice contains all the weights for one virtual output line and enough logic to accumulate the weighted sum of the inputs and to output the result at the appropriate time. It is useful to partition the logic in each slice into two parts as shown in Figure 3. First, there is some straightforward logic that passes the input and output values through as simple shift registers and also another shift register which passes along a "select" flag to signal when to generate an output value. Second there is a black box which does the accumulation of weighted inputs and the generation of the outputs. It is convenient to leave this as a black box as it can be constructed in many different ways depending on the representations used for input values and weights. Two possible forms for this black box will be investigated in the following sections. In fact, it can be any finite state machine with the following inputs and outputs:

inputs:

- two or more input values from chip input lines;
- two or more weights;
- a "select" flag which if on says that an output should be generated;

output:

- one output value generated when "select" flag is on.

These finite state machines will be referred to as accumulators without prejudicing the fact that they can perform quite different functions.

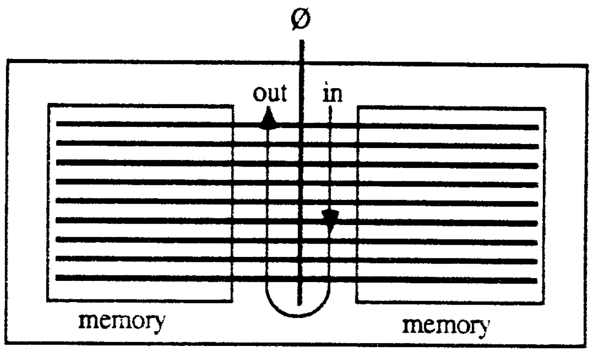
Important parameters

Within this basic architecture a great deal of variation is possible. The following sections consider the details of possible applications but even without these details it is possible to evaluate in a general way the execution speed and size of the chip. The major parameters which determine the performance are :

- •N - the number of values multiplexed onto a single line and;
- • i - the number of input wires.

Speed

The time for all outputs to be computed from one group of multiplexed inputs is proportional to N. So the less multiplexing the less time is taken. However, less multiplexing implies more interchip wiring down to the ridiculous limit where N=1 and nothing has been gained. The speed of the chip is also determined by the time it takes to accept one bit from all the inputs and generate one output bit. This will will be fast compared with say a full microprocessor because the speed of a chip is determined by the longest "wires" inside the chip and by the number of gate delays required on each clock cycle. All the data wires inside the chip are short. The majority just interconnect adjacent cells in shift registers and the longest of these span the distance between successive slices of the chip. Also there is only a one or two gate delay between successive cells in a shift register. The slowest part of the chip is thus likely to be the finite state machines on each layer. The subsequent sections propose that these be very simple so there seems to be no difficulty in maintaining high clock rates. One remaining problem might be the distribution of the clock across the chip. However, by cunningly arranging the parts of the chip it is possible to substantially alleviate even this problem. As shown in the diagram below if the shift registers containing the memory for the weights are bent round to occupy the two sides of the chip the input lines and output lines are brought together. This means that the clock can be distributed via the tree pattern shown. Because all the data interconnections are only short local connections it does not matter if there is skew between the clock where it enters the chip and the clock signal at the ends of the distribution tree.



The result of all this is it should be easy to obtain clock rates on chip comparable to modern microprocessors (10 to 20 MHz). It is conceivable that clock speeds of 100MHz are possible although whether these could be carried off the chip is questionable. For later estimates a clock speed of 20MHz will be assumed. If N=512 this implies that the time to process one group of inputs will be 26μsecs. Slower than nonmultiplexed electronics but still three orders of magnitude faster than neurons in the brain.

Size

There are three components to the size of the chip. First there are the input, output and select registers connecting the slices. The size of these will be determined by the product of the number of input wires and the degree of multiplexing: $i \cdot N$. Second there are the finite state machines. The size of these is unknown but will be an increasing function of i, and the number of them will be proportional to N. The third component is the shift registers for the weights. The size of them will be determined by $i \cdot N^2 \cdot w$ where w is the number of bits required for each weight. For example, with $i=2$, $N=512$ and $w=1$ the total number of cells is 5×10^5. Each cell needs about 5 devices (this figure is taken from a design project using CMOS using conservative design rules (Schack,1985)). This gives a chip with about 2.5×10^6 devices. At the moment (1986) the upper limit for one chip is about 10^6 devices.so this is too large although time will eventually take care of that.

One way to get around the problem of too many devices for the chip follows from the observation that only about four wires connect successive slices of the chip: two (or more) input wires, the select wire and the output wire. Thus it is easy to split the chip horizontally into a number of successive chips with four outputs from one chip going directly to four inputs on the next. Figure 4 shows the arrangement envisaged. Given that the the number of devices will force the splitting of a single Σ–chip into a number of subchips there is no point in having a chip with more than one output wire. This is equivalent to two chips with the same inputs and a single output.

As we have seen there are a number of tradeoffs between size, speed and the amount of external wiring on a Σ-chip. There is another important parameter which needs to be kept in mind when considering these tradeoffs, that is the number of different input values available when computing an output. This determined by the product $i \cdot N$. The example below shows that to do interesting computations with the Σ-chip this value needs to be large. For example, with $i=2$ and $N=512$ each output bit can be a function of 1024 different input bits. Although, this is not such a large number when it is compared with the brain where each neuron can have from 1,000 to 100,000 afferent synapses and perhaps even more which could potentially make a connection.

III. Logical constraints

I will now consider the n-queens problem as the first example of a problem soluble using the Σ-chip. In this example each input or output bit will code for a single true/false logical value. Each weight will consist of a single bit used to select relevant bits from the input. The accumulators will count the number of input bits which are true and which have a weight of one. Put another way the inputs are anded with the weights and the resulting true values are counted. The output of each accumulator will be determined by a threshold and some control bits (it is assumed that the programmer can set both the control bits and the threshold on each of the accumulators independently). When an output is to be generated the threshold is compared with the sum and the sum is reset to zero. A true will be output depending on whether the control bits have selected a comparison of greater, greater or equal, less, less or equal, equal,or not equal.

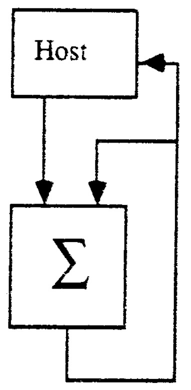
Within this scheme all the standard logical functions can be obtained. For example, the and of some set of inputs is obtained by setting the threshold to the number of selected inputs and using a comparison of greater or equal (an input is selected if its corresponding weight is 1). In similar fashion or and not can be constructed. Exclusive or of two inputs is obtained by setting the threshold to 1 and the function to equals. As well, other more complex functions not easily expressed in boolean logic can be obtained, for example, "any two of some large set of inputs are true".

The n-queens problem has been chosen as an example because it is a well known representative of a large class of combinatorial problems where some finite structure is to be constructed subject to a set of constraints. The finite nature of the problem is essential to its representation within the Σ-chip framework. It will be shown that the Σ-chip can provide a very fast check on whether a proposed solution satisfies the constraints and further that it can propagate constraints and detect values which are forced to unique values. This is also proposed as an example of the sort of processing that is needed when processing sensory input. Often there are a number of distinct features (say edges and vertices in a picture) which must be solved in terms of a coherent three dimensional scene.

Conflict detection

n queens are to be placed on an n×n board in such a way that no queen attacks another along the same horizontal row, vertical column or diagonal. The first problem given to the Σ-chip will be to detect when a given placement of queens is in conflict. Each of the possible n^2 positions for a queen is allocated a single bit. Detection of a conflict is done separately for each row, column and diagonal by counting the number of queens on each (that is setting the weights for the positions along it to 1). If there are two or more queens then there is a conflict, that is, the threshold is set to ≥2 (this is an example of a function not easily computed using boolean logic). Finally all the outputs from the rows, columns and diagonals can be ored to get a single conflict/no conflict bit. For a given n (for example, n=8) there are n^2 (64) bits for the queen positions, $4 \cdot n-2$ (30) for the conflicts on the rows, columns and diagonals and 1 for the final output (see Figure 6).

The diagram below shows the type of arrangement necessary to integrate the Σ-chip with a host computer for this class of problem. There needs to be an interface from the host which can send the board position as a string of bits, similarly the final output of the chip needs to be able to be intercepted by the host (there are many ways of doing this, they wont be considered further). In this instance one Σ-chip is sufficient for quite large problems. Part of the output of the chip is the bits which say whether any row, column or diagonal is in conflict. So that this can be used to compute the final conflict bit, the output is fed back to the chip as one of its inputs. The chip thus has two inputs, one from the host specifying the current board position and the other its own output fedback.



Given this arrangement it takes two full computing cycles before the final conflict value has been computed correctly. To bolster the assertion that only one Σ-chip is needed note that one of the input wires needs n^2 bits for the board specification and the other $6 \cdot n-4+1$ for the row, column and diagonal conflicts and the final conflict value. Given a chip with i=2 and N=512 problems as large as n=22 can be handled (n^2=484 and $6 \cdot n-4+1=129$).

Propagating constraints

The second n-queens task given to the Σ-chip is the detection of situations where it can be deduced that a queen must be placed in certain positions and also the implementation of a more sophisticated failure detection. As before the queens are represented by n×n bits, however, this time an array of "available positions" is computed as well. This is an array of n×n bits which will be true if that position is not under attack and does not currently contain a queen. Each such bit is computed by checking that no queens are positioned horizontally, vertically or diagonally from its position. The diagram below shows the pattern of weights set to 1 for one such detector on an 8×8 board.

		1			1		
			1		1		1
				1	1	1	
1	1	1	1	1	1	1	1
				1	1	1	
			1		1		1
		1			1		
	1				1		

Every row and column in an n-queens problem must contain exactly one queen. So, given the available bits it is possible to detect rows and columns where no queen can be placed (none of the n-positions has a queen or is available). For a given row, this means summing n inputs from the available bits and n from the queen bits. If none of these is set (sum < 1) then no solution is possible. Overall failure can be detected by 'or'ing together the bits from the row, column and diagonal conflict detectors (from the earlier simple solution) as well as these row and column unavailable detectors.

If any row or column has just one position available then a queen must be positioned there. This situation can be detected by constructing for each board position two detectors, one which is true if a queen is forced because of the situation in the row and the other which is true if the column forces a queen. I will refer to these as horizontal and vertical forcing bits, or hf and vf for short. Consider for a moment one hf bit. The easiest way to compute it is to check that all the other available bits in the row are false and that no queen is attacking the position. This is done by using a set of weights for all the remaining available bits in the current row as well as the same pattern of weights as for detecting available bits. (This last may seem redundant but it saves forming the negation of all the available bits and using them). The diagram below shows the pattern of weights for one hf bit on an 8×8 board:

queens

		1			1		
			1		1		1
				1	1	1	
1	1	1	1	1	1	1	1
				1	1	1	
			1		1		1
		1			1		
	1				1		

available

1	1	1	1	1		1	1

The final step is to use these values to update the placement of queens on the board. In the simple algorithm used earlier the values of the queen placements are supplied by the host computer. To allow both the host and the internal processing to set queen values two sets of queen values are used, one supplied by the host and the other computed internally. Each internal queen bit is computed as the or of the corresponding hf and vf bits as well its own previous value and of the queen value supplied by the host. That is the formula for computing the internal queen bits is:

$$q'_i \leftarrow q'_i + hf_i + vf_i + q_i \geq 1$$

where q' is the internal queen value, hf and vf are the horizontal and vertical forcing values and q is the queen value supplied by the host. This has the required effect that once an internal queen bit has been set it remains fixed on. This is fine until the host needs to reset the values to test a new configuration. To enable such a resetting, the following slightly more complex equations are used:

$$q''_i \leftarrow q''_i + q_i + run \geq 2$$
and $$q'_i \leftarrow q''_i + hf_i + vf_i \geq 1$$

where q" is a new set of internal queen values and run is a single bit supplied by the host. 'run' is used to to reset the values of q". During normal running run is set to true, so that the first equation reduces to $q''_i \leftarrow q''_i + q_i \geq 1$, that is, q''_i is the or of itself and q_i. When the host wishes to reset q" run is made false and the equation becomes $q''_i \leftarrow q''_i + q_i \geq 2$. This forces all values of q" to zero unless the q" bit is already on and the q bit is also on. The effect then is that when run is again made true, q" will be set equal to q.

The only matter still remaining is the detection of termination. The hf and vf values might force a number of queen positions on in sequence (see Figure 6 for an example). Termination is easily determined by checking that all the hf and vf values are false, then no more queens will be forced on and no more changes in output will occur. Also of interest to the host is when a solution has been completed successfully. This can be detected by counting the number of queens and setting the threshold to $\geq n$.

The outputs required by the host are the values of q" (the new set of queen values), the detection of failure flag, and the termination and successful solution flags mentioned above. Figure 5 summarizes the equations for this problem and gives a layout of four Σ-chips which can handle problems up to n=16 (when i=2 and N=512).

Figure 6 shows a sequence of steps for the algorithm. An initial situation with four queens placed on the board is found to have no possible solutions. This is done by a combination of forcing queens into places where one must go and detecting the subsequent conflicts. The detailed steps start with a set of queen positions transmitted from the host.

After 4 full computing cycles one hf (horizontal forcing) bit is set. This forces one queen onto the board. After another 3 cycles, there are two hf bits and one vf bit set. These force three queens onto the board. After another 2 cycles, this situation is found to have a conflict as there are two queens along one of the diagonals (marked in Figure 6). This conflict has turned on the f (failure) bit. As well the s bit is set because there are 8 queens on the board and the t bit is set because none of the hf or vf bits are on.

The Σ-chip program explained here is intended to be used in conjunction with a sequential program in the host which generates potential solutions. The main advantage it provides is a very quick and relatively sophisticated way of pruning such searches. The type of analysis used here can be carried further by for example checking that a solution is not some rotation or reflection of a solution which would have been generated earlier.

Chess

As a further example of what can be done with this style of processing the problem of checking for patterns on a chess board will be briefly examined. A suitable representation of the board has first to be chosen. The one most suitable for pattern recognition seems to be to assign one bit to each possible piece in each possible position on the 8×8 board. Each player has six types of pieces (pawn, rook, bishop, knight, king and queen). So allowing for two players this gives a total of $2 \cdot 6 \cdot 64 = 768$ bits to represent a board. Some of the interesting features which can be easily computed using the Σ–chip in one or two summing steps are:

•all positions under attack by one colour; ;

•all positions under attack by the opposite colour: use the result from above and check also that there is a piece in the threatened position;

•all pieces which are covered by a piece of their own colour (that is if they are taken the attacker can be taken in turn): as above;

•all positions which would place pieces under a fork attack: as for detecting an attack but set the threshold to 2 rather than 1.

Because so much pattern selection can be done in parallel much more attention can be paid to a single position. In sequential programs possibilities such as forks take a lot of processing to check in every position, however, they only occur rarely. Thus it may nor be worthwhile checking for them even though they are very important when they do occur. In a parallel system such as this this does not matter as the check takes only space not time.

Summary

The main point of these examples is that a simple device processing a large number of inputs in parallel can in some problems provide a very quick check that constraints are satisfied. This type of processing can be characterized as a pattern driven approach. A situation is generated and then a lot of parallel checking is done to see if it satisfies a large set of interdependent constraints. It is not clear though that the class of problems helped in this way is sufficiently large to warrant creating a new breed of hardware. Time will tell, however, there are some pointers that the approach is promising. It seems to be characteristic of the very successful type of strategy employed by the human brain when playing games such as chess. Humans typically examine only a few potential game positions but unleash very sophisticated pattern matching on each position. Typical sequential programs examine many millions of potential positions but analyse them only very superficially. Also it may be characteristic of perceptual and sensory processing tasks where a complex but fixed situation is carefully examined in many different ways to extract features.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

References

Anderson, J.A. and Hinton, G.E. (1981) "Models of information processing in the brain," in *Parallel Models of Associative Memory*, eds. Hinton, G.E. and Anderson, J.A. Hillsdale, NJ:Erlbaum.

Gaines, B.R. (1969) "Stochastic Computing Systems," in *Advances in Information Science* Vol. II, ed. Tou, J. Plenum.

Hebb, D.O. (1949) *Organization of behaviour*. New York: Wiley.

Hinton, G.E. and Anderson, J.A. eds (1981) *Parallel Models of Associative Memory*. Hillsdale, NJ:Erlbaum.

Kohonen, T. (1984) *Self-organization and associative memory*. Springer-Verlag, Berlin.

McCulloch, W.S. and Pitts, W.H. (1943) "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, **5**, 115-133.

Minsky, M. and Papert, S. (1969) *Perceptrons*. Cambridge, Mass.: MIT Press.

Sutherland, I.E. and Mead, C.A. "Micro-electronics and computer science," *Scientific American*, **237**(3), 210-228.

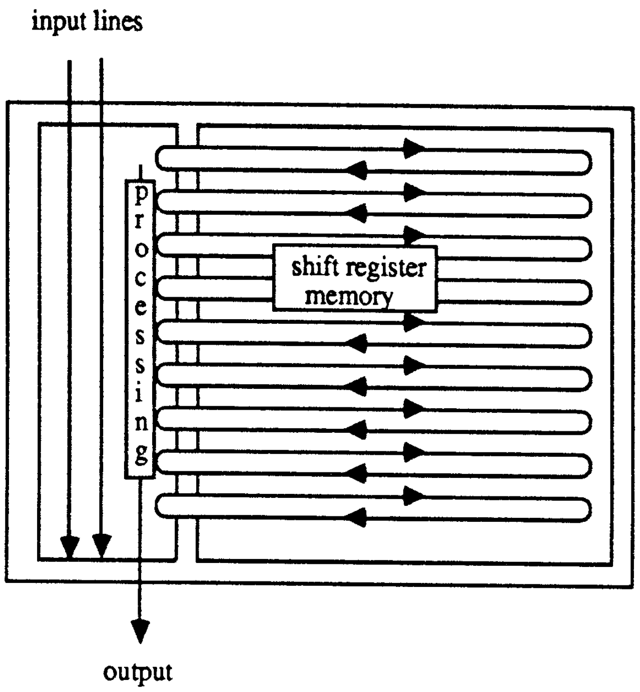


Figure 1. Layout of Σ-chip.

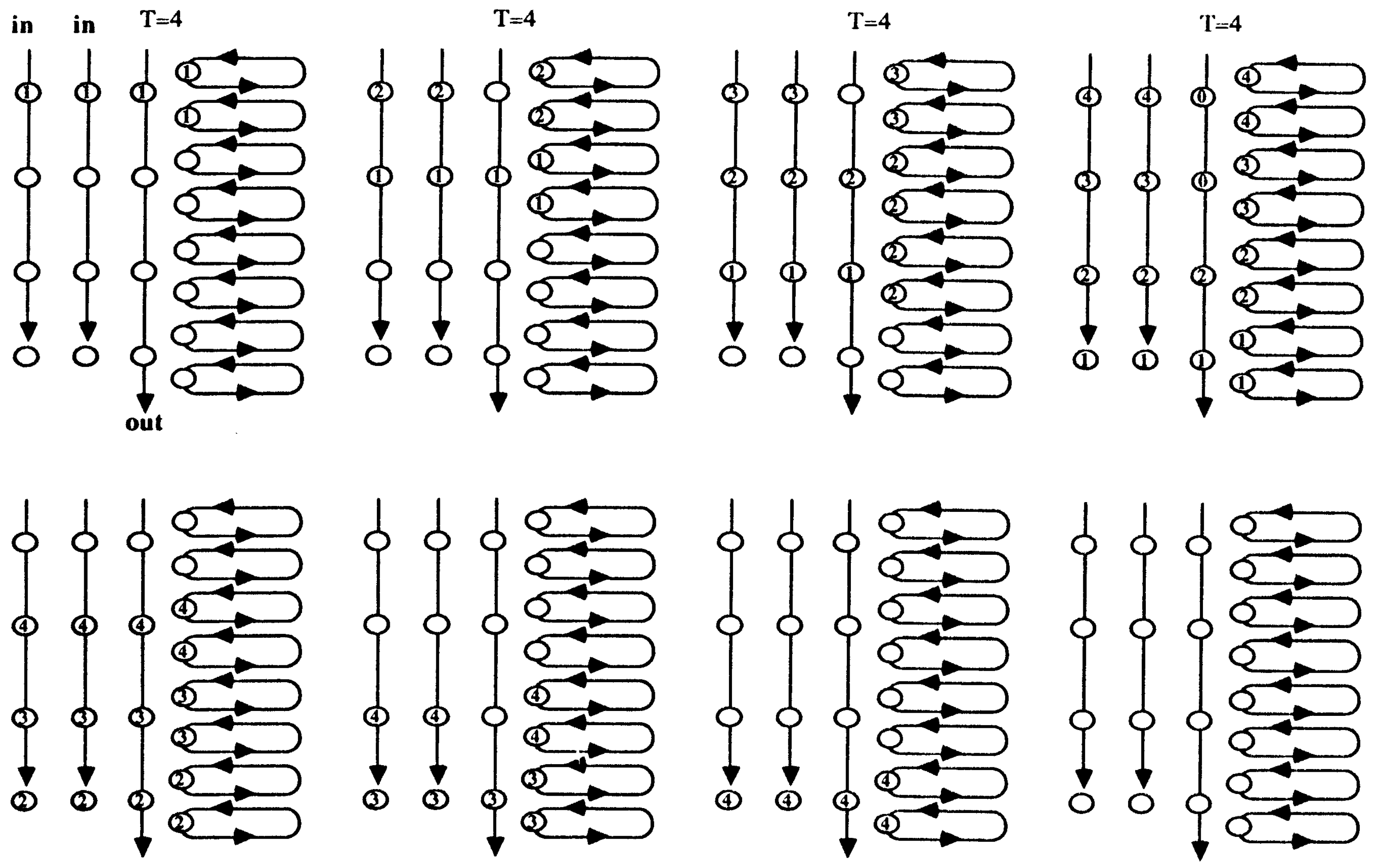


Figure 2. Relationship of inputs, weights and outputs.

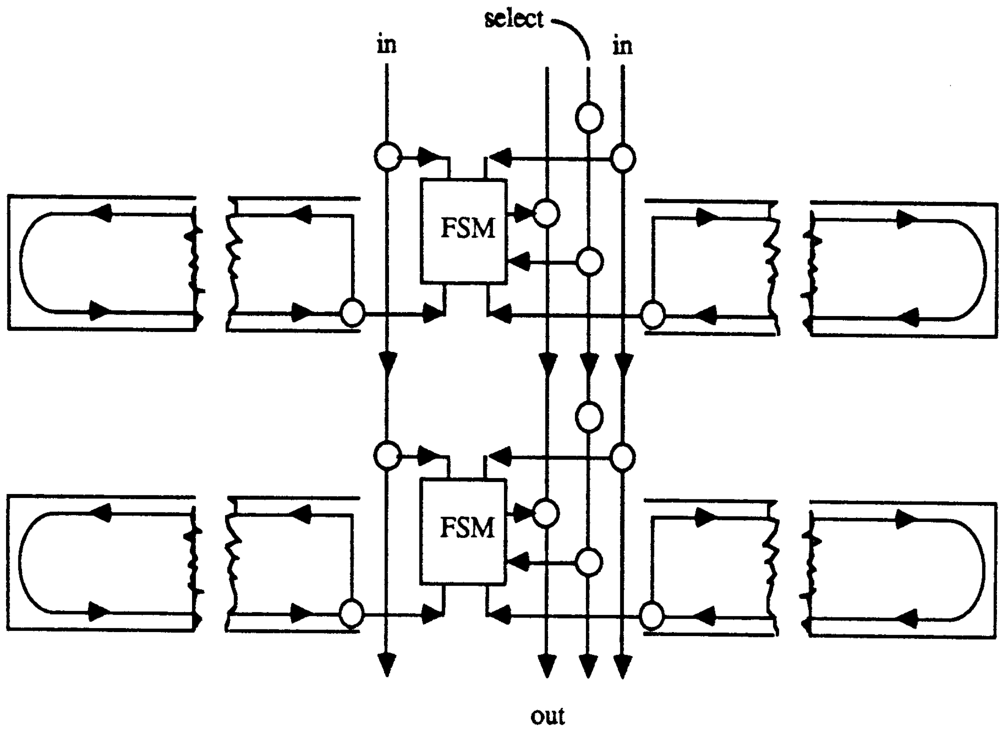


Figure 3. Logic for individual slices of chip.

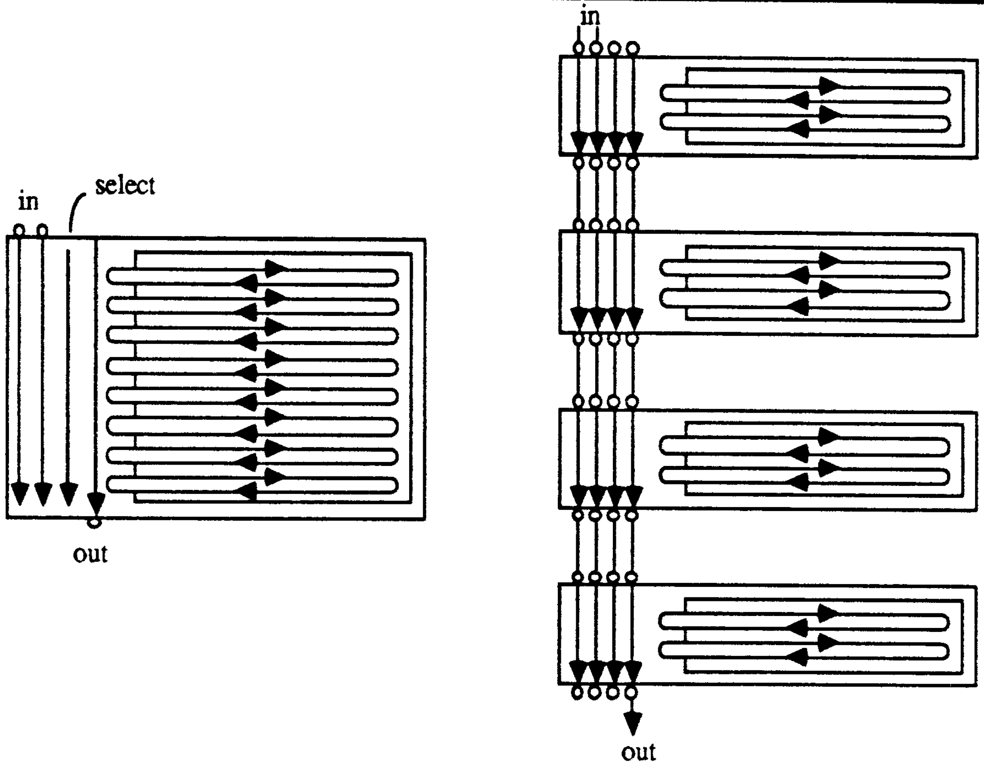


Figure 4. Splitting of one Σ-chip into four chips.

Values used (for n=16)

$q_{1..256}$	positions of queens (input from host)
$q'_{1..256}$	positions of queens (as computed internally)
$q''_{1..256}$	positions of queens (temporary value)
run	used to reset q'' values (input from host)
$r_{1..16}, c_{1..16}, d_{1..60}$	detectors of conflicts along rows, columns and diagonals
$a_{1..256}$	positions where queen can be placed without conflict
$hna_{1..16}, vna_{1..16}$	detect that rows and columns have no positions available for queens
f	no solution possible (a conflict or unavailable row or column)
$hf_{1..256}, vf_{1..256}$	must contain a queen (only one position available in a row or column)
t	process of forcing new queens has terminated
s	n(16) queens have been placed on board

Equations (Σq indicates a sum over some subset of q)

$q''_i \leftarrow q''_i + q_i + run \geq 2$

$q'_i \leftarrow q''_i + hf_i + vf_i \geq 1$

$r_i \leftarrow \Sigma q' \geq 2$ $c_i \leftarrow \Sigma q' \geq 2$ $d_i \leftarrow \Sigma q' \geq 2$

$a_i \leftarrow \Sigma q' = 0$

$hna_i \leftarrow \Sigma a = 0$ $vna_i \leftarrow \Sigma a = 0$

$f \leftarrow \Sigma r + \Sigma c + \Sigma d + \Sigma hna + \Sigma vna \geq 1$

$hf_i \leftarrow \Sigma a + \Sigma q' = 0$ $vf_i \leftarrow \Sigma a + \Sigma q' = 0$

$t \leftarrow \Sigma hf + \Sigma vf = 0$

$s \leftarrow \Sigma q' = 16$

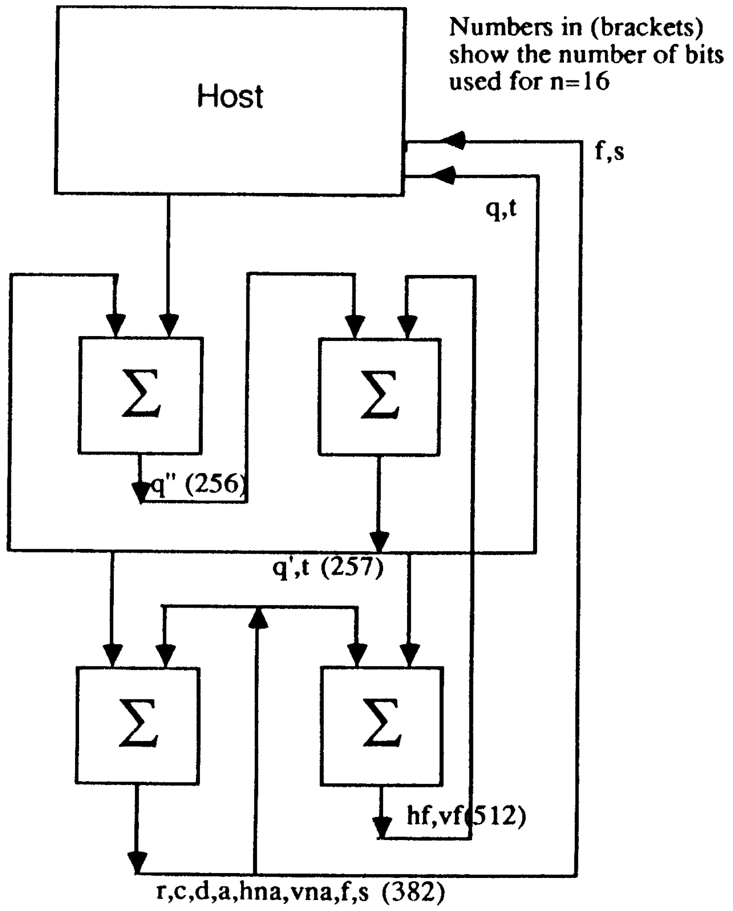

Numbers in (brackets) show the number of bits used for n=16
Host
f,s
q,t
Σ
Σ
q'' (256)
q',t (257)
Σ
Σ
hf,vf(512)
r,c,d,a,hna,vna,f,s (382)

Figure 5. Solution to n-queens including forcing queens.

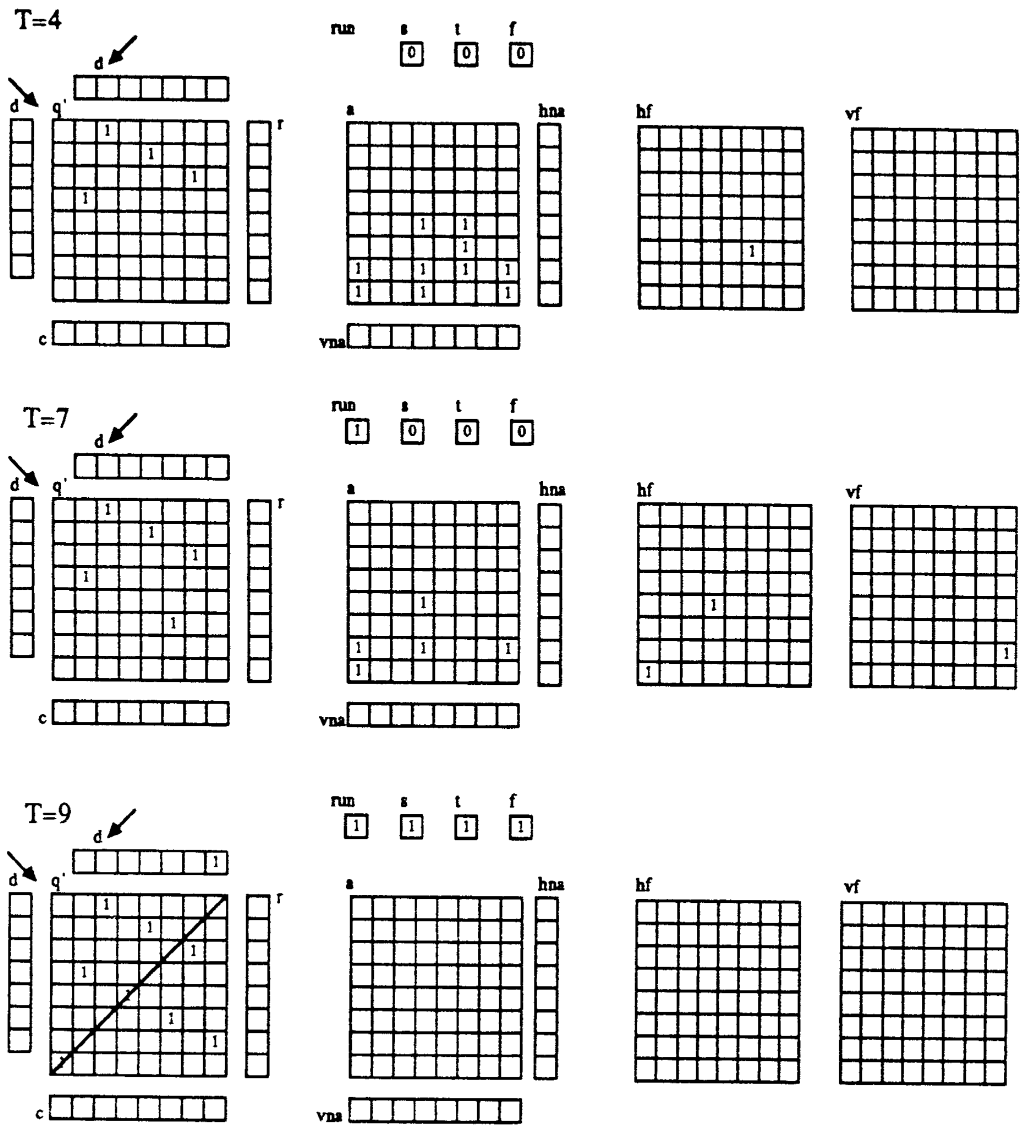


Figure 6. Execution of n-queens problem.