Sequential Threshold Circuits

John C. Platt

Computer Science Department
California Institute of Technology

5197:TR:85

Sequential Threshold Circuits

by

John C. Platt

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

Computer Science Department

Technical Report Number 5197:TR:85

California Institute of Technology

Pasadena, California

1985

This work supported by an NSF Fellowship and by the System Development Foundation.

Acknowledgements

I would like to thank my advisors, Carver Mead and John Hopfield, for the very useful advice and guidance provided during the development of this thesis.

Also, I would like to thank Dick Lyon, Al Barr, and Jim Kajiya for their suggestions and comments about the thesis.

Table of Contents

1 Introduction

In this thesis, a technique for synthesizing sequential circuits from threshold elements is described.

The historical roots and possible future applications of the technique are presented in this chapter. Simple ways of building threshold elements are described in chapter 2. How to connect these elements to form sequential circuits is described in chapter 3. Chapter 4 explains analog Petri nets, a way of describing analog asynchronous behaviour. Chapter 5 show how to convert an analog Petri net to a threshold circuit. Chapter 6 presents conclusions and possible future work.

1.1 Historical Perspective

The synthetic technique described in this paper arose out of three other ideas: J.J. Hopfield's content addressable memory, Petri net modeling of self-timed circuits, and the idea of analog decision elements such as flip-flops.

The active elements that the technique uses are the same as Hopfield's associative memory [Hopfield 84]. Hopfield's neurons are threshold elements, each connected symmetrically to each other, with all of the elements evolving at once. However, in this thesis, the interconnection used is asymmetric, which gives behaviours that depend continuously on time and input. The intuition of state space is still helpful. The idea of using threshold elements to simulate "neural" behaviour has a long history [Minsky 68][McCulloch 43].

The Petri net description is taken from self-timed circuit design, where it has proved to be a compact and powerful way of representing concurrent asynchronous behaviour [Holt 70][Dennis 70][Seitz 70].

The analog asynchronous sequential machine performs analog arbitration, which is exactly the computation that a flip-flop does [Mead 80]. However, in this thesis, a firm framework is provided for translating a behavioural description to an actual circuit.

1.2 Possible Applications

Several possible applications of the analog asynchronous sequential machine exist. First, it can be a control element for other analog circuits. For example, it might be useful in building servomechanisms in robots, where analog position information is changing arbitrarily slowly and crisp decisions need to be based on this information.

Second, it might be useful in asynchronous perceptual tasks, where a circuit must compute continuously. Contrast this to Hopfield's associative memory or travelling salesman [Hopfield 85], which have definite computational starts and stops, because the circuits extremize an energy function. Thus, one might expect an analog self-timed automaton to be useful in speech recognition on an auditory chip, or motion detection on a vision chip.

Finally, the technique might be useful for composing other collective systems and for making fault-tolerant circuits.

2 Threshold elements

This chapter of the thesis describes a simple way of making threshold elements, either using discrete components, or a VLSI chip.

The sequential circuit consists of a group of threshold elements. The circuit is programmed by changing the connections between the threshold elements. Thus, consider an ideal threshold element shown in figure 1.

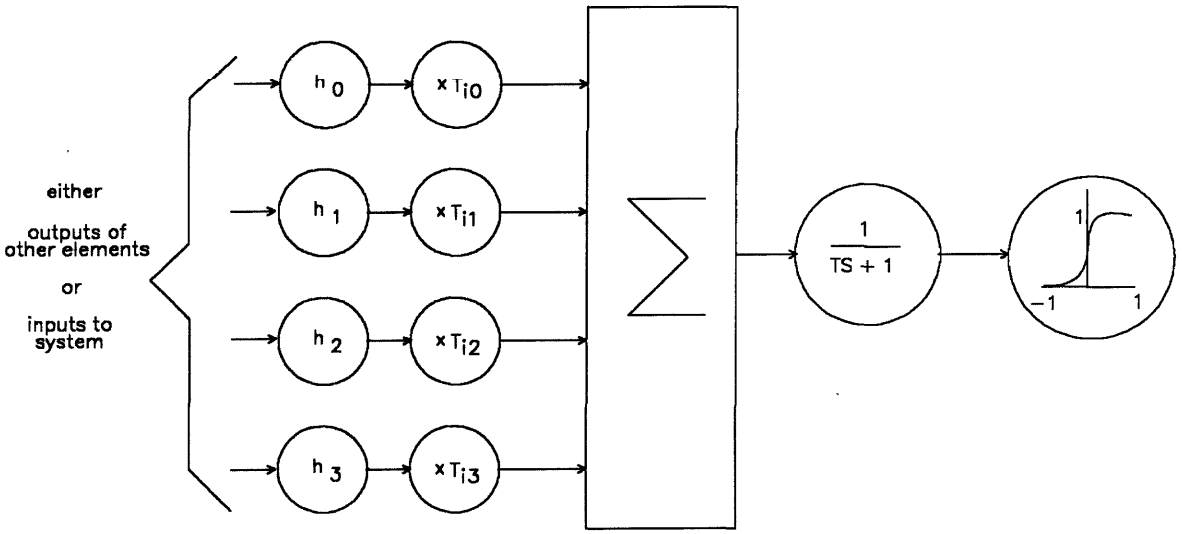

Fig. 1. Ideal Threshold Element (ith element)

Let the output of the ith element be V_i, with $1 \leq i \leq n$, where n is the number of threshold elements. V_i is a voltage between the supply rail voltages 0 and 1; i.e., $V_i \in [0,1]$. One also wants to represent inputs to the system. These inputs are also represented as V_i, with $n+1 \leq i \leq N$. Thus, $\vec{V}$ is an *extended state vector*, whose first n elements are the output voltages and whose remaining elements are inputs controlled by the outside world. The extended state vector is illustrated in figure 2.

The input of each element is a weighted sum. The sum can be over the output voltages V_j or some functions of the output voltages. In general, let the input be

$$I_i = \sum_{j=1}^{M_i} T_{ij} h_j(\vec{V}) - \gamma_i \qquad (1),$$

where the weights T_{ij} may be positive or negative, γ_i is the threshold, and M_i is the number of inputs in element i. Ideally, the behaviour of the circuit should be independent of the scale of the T_{ij} and γ_i, so in this discussion about the ideal element, consider $\sum_j |T_{ij}| + |\gamma_i| = 1$

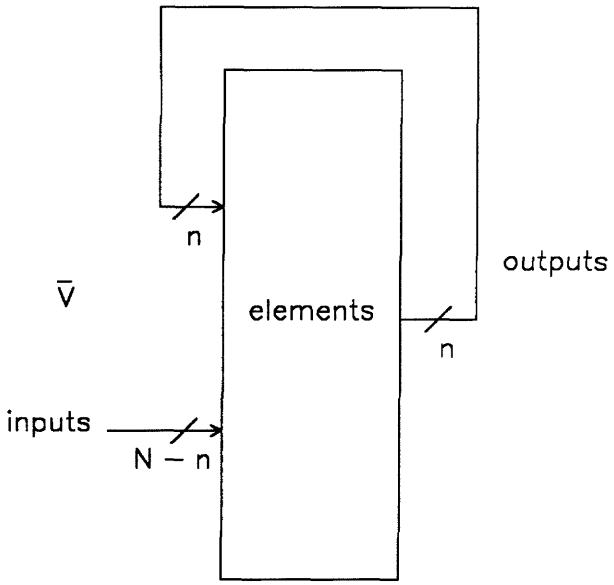

Fig. 2. Extended State Vector

The weighted sum should be integrated with a transfer function of

$$\frac{1}{\tau s + 1}.$$

In other words, perform a "leaky integration" on the input.

After the integration, run the signal through a non-linear transfer function, with a gain of > 1. A non-linear transfer function with a gain greater than 1 is essential to make crisp digital decisions from the analog inputs. An example is a flip-flop, whose stable states go to the voltage rails for a gain greater than 1, but sink to 0.5 for a gain less than 1. In order for any input to possibly make a change in the output from one digital level to the other, the gain has to be more than $1/\min|T_{ij}|$.

In the analysis in the next chapter, there is an assumption that the wires are fast compared to the time constant of the element, so that every element sees the same output voltage for any particular element. Thus, the elements should have a long integration time to make wire delay negligible.

2.1 Discrete Implementation

A discrete circuit to realize the threshold element is shown in figure 3.

Here, the weighted sum is carried out by resistor summing. T_{ij} is the conductance of the resistor connecting the jth input of element i. γ is a conductance also, which is connected to the negative power supply. Various h_j functions may be implemented by no extra circuitry ($h_j(\vec{V}) = V_j$) or by CMOS logic.

The input to the element is $\sum_j T_{ij} h_j(\vec{V})$ when the first op-amp is not saturated, while the gain is $R_f \sum_j |T_{ij}|$. Thus, if one scales the T_{ij}, the circuit behaviour will remain constant, as long as R_f is scaled inversely.

The leaky integration is performed by the resistor and capacitor across the operational amplifiers. This yields a transfer response of $1/\tau s+1$ when the first op-amp is not saturated. When the first op-amp saturates, the input voltage of the op-amp is no longer held at zero. Thus, the input is no longer the proper weighted sum, and the integration eventually stops [Millman 79].

The non-linear transfer response is merely the transfer response of the operational amplifier, with the gain regulated by the ratio of the feedback resistor to the equivalent input resistance. V_i is kept between 0 and 1 by the diode between the output of the first op-amp, and the input of the second op-amp. If the output of the first op-amp rises above ground, the diode becomes reverse-biased, and $-V_i$ is tied to 0. Otherwise, $-V_i$ tracks the output of the first op-amp. The follower gives $-V_i$ a low output impedance.

Thus, V_i varied between ground and the positive power supply, and $-V_i$ varies between the negative power supply and ground.

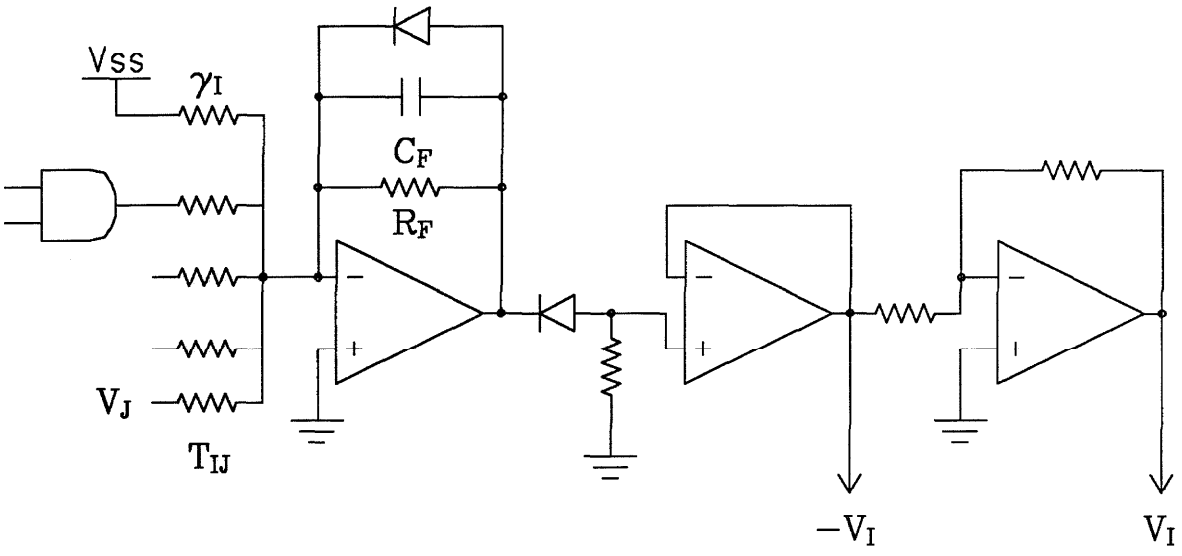

Fig. 3. Discrete Circuit

One immediate awkward question is how to implement negative conductance to achieve a negative T_{ij}. A negative conductance is physically impossible, thus each element produces a positive and a negative output, using a unity gain inverting op-amp. Whenever a negative conductance $-G_{ij}$ is desired, a conductance of G_{ij} is connected to to the negative output $-V_j$.

Figure 3 is only a general circuit diagram. Specific conductances are shown in Appendix B.

2.2 VLSI Implementation

In VLSI, resistors are difficult to build, so the most natural way of implementing a weighted sum is by using currents. The circuit shown in figure 4 is a natural way of implementing the active elements. The element is implemented with reverse logic. Thus, an element is "on" when the voltage is low, and visa versa.

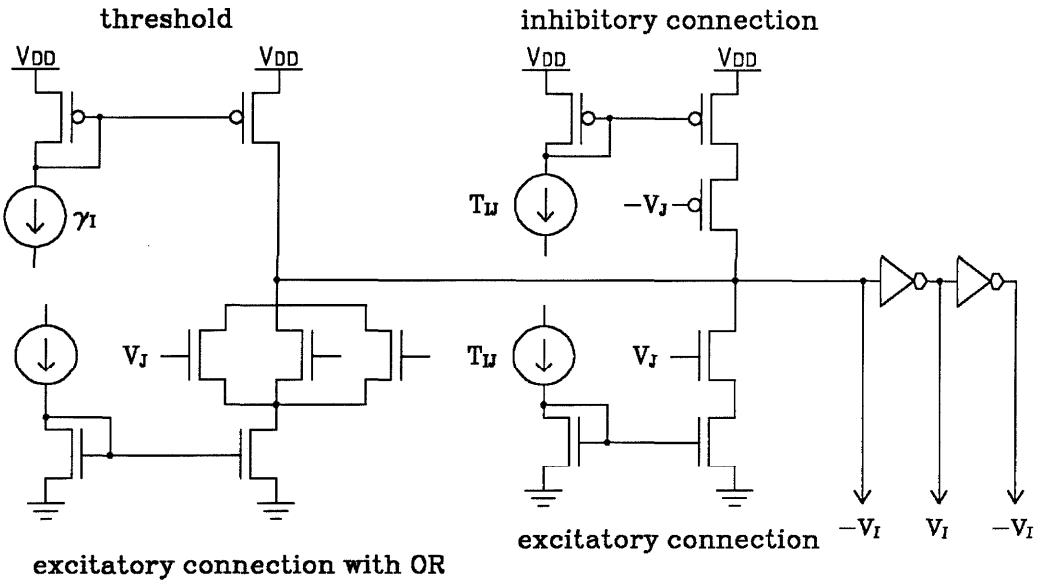

Fig. 4. VLSI Circuit

The circuit in figure 4 is far from ideal. Consider the case where $h_j(v) = v, \forall j$. The input is $\sum_j T_{ij}V_j - \gamma_i$ only when the V_j are either 0 or 1. Otherwise, the sum is really $\sum_j T_{ij}y_j(V_j) - \gamma_i$, where y is the non-linear connection response described in Appendix C. However, $y_j(0) = 0$ and $y_j(1) = 1$.

In figure 4, the negative weights are built by using p-type transistors as pullups. The weights are the current that can go through the current limiting transistors whose gates are part of a current mirror. Thus, only a few weights should be distributed globally.

The integration comes about naturally because of parasitic capacitances of the wire. But, the integration is $1/\tau s$, and the integration is stopped by the voltage rails. Thus, the integration and large gain are mixed in the current summer. The extra inverter stages on the output give additional gain and both senses of output.

In the circuits used in this thesis, there will only be a constant number of weights used. Thus, the T_{ij} currents can be supplied from pads, and the current mirror gate voltage distributed globally.

When one wants to make a γ_i connection, or any other connection that is always on, one merely eliminates in V_j transistor. A positive threshold is implemented with p-type transistors, while a negative threshold is implemented with n-type transistors.

If the inputs to the threshold circuits are in the form of currents, then one uses the circuit in figure 5. In figure 5, the input current is removed from the summing node, except that the current is limited by the T_{ij}.

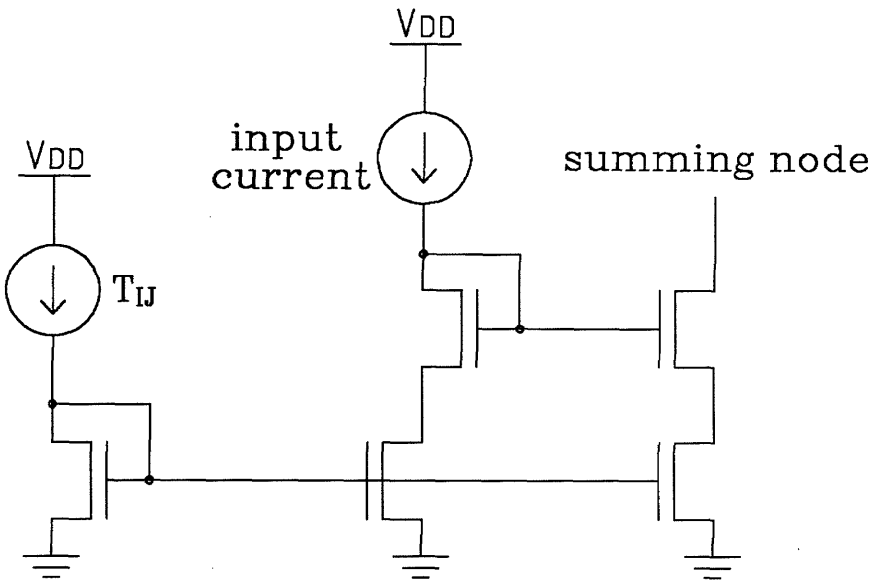

Fig. 5. Analog Input Connection

The current mirrors in figure 4 and figure 5 should be operated below threshold to keep the heat dissipation low, and to minimize the dependence of the current on the drain voltage. However, using sub-threshold current mirrors, errors in the threshold voltage across the chip become exponential errors in the T_{ij}. These errors can be minimized by keeping all the current sources for an element close together and by keeping all the global current-setting transistors close together. Thus, any variation in threshold over a long spatial range is canceled out, since the threshold element only relies on the ratios of the currents. However, if local threshold voltages vary by more than roughly 8 mV, the circuit shown in figure 4 will not work.

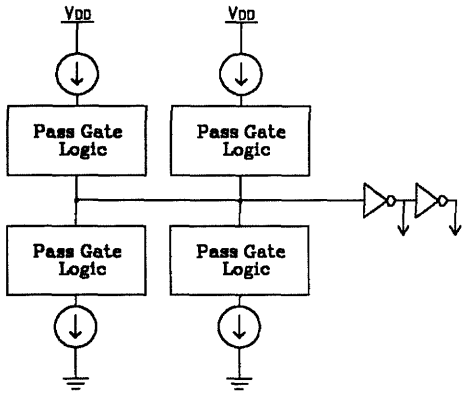

Fig. 6. Pass Gate Threshold Logic

VLSI presents an opportunity for making more interesting h_j than with simple resistors. Namely, beneath each current source that sets T_{ij}, one can put pass gate logic that implements compositions of min and max of any of the V_j. Pass gate logic is extremely

useful, since pure threshold logic does poorly at doing min and max (the equivalent of Boolean AND and OR). Thus, one has a new logic family, doing small Boolean computations with the pass gate logic, and large sums with the current summer. This logic family, shown in figure 7, will be useful in implementing analog sequential circuits.

3 Sequencing Theory

In the last chapter, simple ways of building threshold elements were presented. In this chapter, techniques are described for connecting the threshold elements to make sequential circuits.

To make a sequential circuit, one wants certain elements to turn on or off based on the present state of the system. To determine whether an element will turn on or off, one can look at the input to element i,

$$F_i = \sum_j T_{ij} h_j(\vec{V}^0) - \gamma_i, \tag{2}$$

where F_i is called the *"force"* on element i when the system is in state $\vec{V}^0$. Here, T_{ij} is the weight of the jth connection to element i, h_j is a function the computes the input to the jth connection of element i and γ_i is the threshold of element i.

The force is the current going into the summing node of an element when the element is not saturated. The sign of F_i will control whether the element will turn on or off. So, if $F_i > 0$ for a particular element, then V_i will increase. If $F_i < 0$ for a particular element, then V_i will decrease. Since the non-linear transfer function has high gain, $F_i > 0$ will make $V_i \rightarrow 1$ and $F_i < 0$ will make $V_i \rightarrow 0$. Notice that the force is linear in T_{ij} and $h_j(\vec{V})$. Thus, it is mathematically easier to handle than the entire non-linear system.

Hence, to predict what the system will do at a particular time, find $\vec{V}^0$, and compute the force vector $\vec{F}$ from the known matrix T and $\vec{\gamma}$. Conversely, if one wants to control what the system will do given a particular $\vec{V}^0$, then one designs T and $\vec{\gamma}$ to produce the desired force vector $\vec{F}$. In other words, one manipulates the weights between the elements to give the desired forces. Or, in circuit terms, one manipulates the transconductances between elements to give currents of the proper sign.

3.1 Constraints

How does one determine the values of T and $\vec{\gamma}$ to yield the desired forces?

First, one has to pick a representation for what particular elements mean. Then, one can develop a set of behavioural constraints. The constraints are in either of two forms.

1) If an element A is on, then turn on element B.
2) If an element C is on, then turn off element D.

If one has a constraint like that of case 1, then one puts excitatory connections ($T_{ij} > 0$) from A to B. If the constraint is like that of case 2, then one puts inhibitory connections ($T_{ij} < 0$) from C to D. The weights of the connections still need to be found.

The idea of turning constraints into connections is analogous to digital asynchronous design, where the next-state function is mapped into connections of the Boolean logic. Notice, also, that the connections are local and Hebbian [Hebb 49], that is, any connection depends only on the desired states of the two elements it connects.

In other circumstances, one wants to turn an element on or off based on a set of Boolean functions of the output of some elements. In this analysis, the output of the elements are treated as digital values. The constraints come in two forms, again.

3) If a Boolean function of a set of elements E is 1, then turn on element F.
4) If a Boolean function of a set of elements G is 1, then turn off element H.

In case 3, make an excitatory connection from the output of the Boolean function to element F. In case 4, make an inhibitory connection from the output of the Boolean function to element G. Again, the weights of these connections need to found.

One can make a picture showing the results of the above decisions. Such a picture is called a "wiring diagram". An example of a wiring diagram is shown in figure 7.

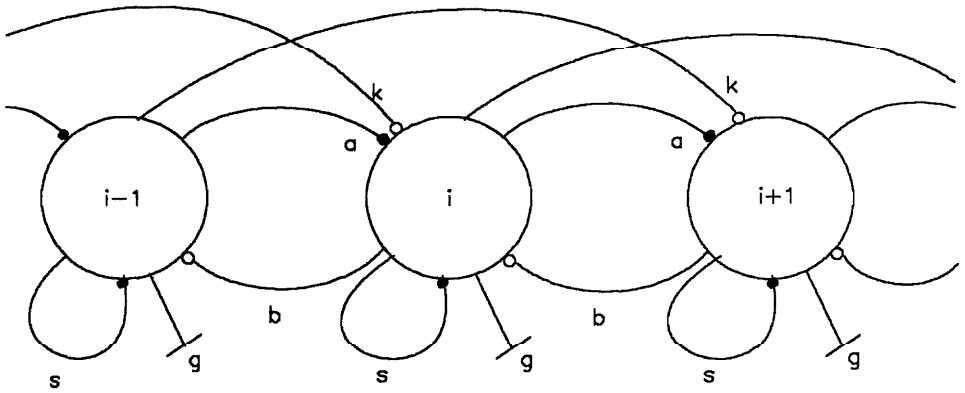

Fig. 7. Connections for a Section of an Oscillator

The wiring diagram shows the connections between elements, without showing the internal circuitry of the elements themselves. The notation in figure 7 will be an intermediate notation between the behavioural description and the final circuit. The arcs ending in filled-in circles mean excitatory connections ($T_{ij} > 0$). The arcs ending in open circles mean inhibitory connections ($T_{ij} < 0$). The t-shaped arcs that only connect to one element are thresholds. The symbols by the arcs are the T_{ij} or γ_i of the connections, which are *parameters* that need to be found.

To find the strengths of the connections, apply the behavioural constraints. A constraint specifies that an element should turn on or off when the system is in a certain state. Consider the case where element i should turn on when the system is in state $\vec{V}^0$. Therefore, $F_i > 0$. using the definition of F_i yields

$$\sum_j T_{ij} h_j(\vec{V}^0) - \gamma_i > 0. \tag{3}$$

Similarly, if the constraint states that element i should be off when the system is in state $\vec{V}^0$, then $F_i < 0$, which implies

$$\sum_j T_{ij} h_j(\vec{V}^0) - \gamma_i < 0. \tag{4}$$

Thus, the strengths of the connections T_{ij} and the thresholds γ_i must satisfy linear inequalities so that the behavioural constraints are satisified. Each linear inequality describes a half-space in a high dimensional space. If one picks the T_{ij} and the γ_i inside the intersection of all the half-spaces, then the behavioural constraints should be satisfied. One should pick T_{ij} and γ_i far away from the boundaries of the intersection of the half-spaces, so that if the connection strengths are slightly in error, the constraints are still satisfied.

3.2 Ring Oscillator

A simple example should illustrate the point of the last section. Consider the task of making an oscillator. In this section, a ring oscillator is defined in an unusual way. Namely, the elements of the oscillator should turn on in order, usually one on at a time, with two on transiently.

The behavioural description for the oscillator can be translated into excitatory and inhibitory connections.

1) If element i is on, it should tend to remain on. Thus, there should be self-excitation.
2) If element i is on, and element $i-1$ is off, then element $i+1$ should turn on. Thus, element $i-1$ should inhibit element $i+1$, while element i should excite element $i+1$.
3) If element i is on, then element $i-1$ should turn off. Thus, element i should inhibit element $i-1$.

When put together, the connections should look like figure 7, which shows a slice of three elements out of an oscillator. The connections should be replicated for all elements in the oscillator.

Thus, element i will tend to excite $i+1$, inhibit $i-1$ and $i+2$, and have hysteresis. These connections make an asymmetric connection matrix, which was first suggested for sequencing in Hopfield's associative memory paper [Hopfield 82]. The asymmetry is in contrast to the use of the symmetric matrices in the memory task and the travelling salesman task.

Now, we start to translate behavioural constraints into mathematical inequalities. Using the weights s, a, b, k, and g, as noted in figure 7, the following conditions arise:

1) If element i is on, it should tend to remain on, if nothing else is on. Thus, $s > g$.
2) If element i and element $i-1$ are on, element $i-1$ should shut off. Since $i-1$ has hysteresis, one must take the force owing to s into account. Hence, $s - b < g$.
3) If element i is on, and element $i-1$ is off, then element $i+1$ should turn on. Thus, $a > g$.
4) If both elements i and $i-1$ are on, then element $i+1$ should not be turned on. Thus, $a - k < g$.
5) In addition, all the variables should be positive, since one wants to preserve the sense of excitation and inhibition.

3.3 Race Prevention

The inequalities in the last section did not consider *races* between

Consider the state when element i is on and element $i+1$ is just turning on. Constraint (3) above makes element i turn element $i+1$ on. Simultaneously, constraint (2) makes element $i+1$ turn element i off. If element i is much faster than element $i+1$, then i may shut off before $i+1$ is fully turned on. Therefore, the is a *race* between constraints (2) and (3) in the previous section. Thus, the circuit may fail to oscillate, with all the elements turning off.

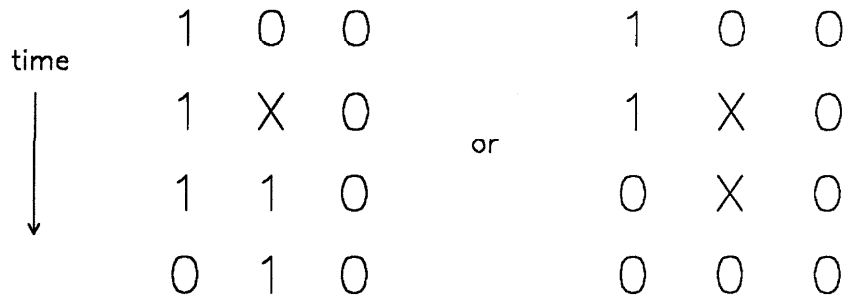

Fig. 8. Race between Self-Reinforcement and Inhibition

As can be seen in figure 8, a race happens when two conditions hold:

1) More than one constraint is applicable to a particular state of the circuit,
2) Applying the constraints in different orders

yield different next states. A race in a sequential threshold circuit is analogous to a critical race in the synthesis of asynchronous digital machines. To resolve a race, one must make certain that certain constraints are applied before other constraints.

Thus, to resolve the race in figure 8, one wants to make sure element $i+1$ is self-reinforcing *before* it shuts off element i. Consider the discrete circuit where the current into a element from one connection T_{ij} is linear in V_j. As element $i+1$ turns on, there is a voltage x_1 where element $i+1$ starts to shut off element i.

$$s - x_1 b - g = 0 \Rightarrow x_1 = \frac{s-g}{b}. \tag{5}$$

Also, there is a voltage x_2 where element $i+1$ starts to self-reinforce.

$$x_2 s - g = 0 \Rightarrow x_2 = \frac{g}{s} \tag{6}$$

To make sure that element $i+1$ is self-reinforcing before i shuts off requires

$$x_1 > x_2 \Rightarrow \frac{s-g}{b} > \frac{g}{s} \Rightarrow s(s-g) > gb \tag{7}$$

Therefore, to prevent races, in a circuit with resistors, one sets up *voltage inequalities*, which may result in non-linear inequalities in the parameters. If all the races in the constraints are removed, then the circuit becomes speed-independent.

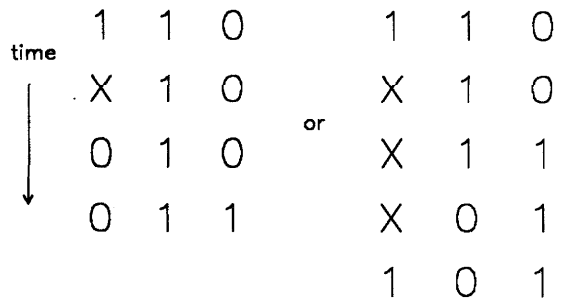

Fig. 9. Race between Self-Reinforcement and Disinhibition

Another race in the oscillator happens when i and $i+1$ are on, and $i+2$ is not yet on. There is a race between i ceasing to be self-reinforcing and i preventing $i+2$ turning on.

Let x_3 be the voltage at which element i prevents element $i+2$ from turning on. Thus,

$$a - kx_3 - g = 0 \Rightarrow x_3 = \frac{a-g}{k}.$$

To prevent the race requires

$$x_2 > x_3 \Rightarrow \frac{g}{s} > \frac{a-g}{k} \Rightarrow kg > (a-g)s. \tag{8}$$

The complete set of inequalities for the oscillator is presented below.

$$\begin{array}{llll} a > g, & s > g, & s - b < g, & a - k < g, \\ s - b < g, & s(s-g) > gb, & kg > (a-g)s. & \end{array} \tag{9}$$

Thus, a good solution to these inequalities is

$$a = 2, \qquad b = 1.5, \qquad k = 3, \qquad s = 2, \qquad g = 1. \tag{10}$$

The experimental verification of this solution is presented in §3.5.

3.4 VLSI Race Prevention

The VLSI circuit does not have resistors, hence does not have an input current linear in the output voltages. Thus, the analysis of the previous section is not directly applicable.

However, one can use the gain of the inverters to order events. Consider the circuit in figure 10.

Let the time constant of the input be much longer than any other time constants in the inverter chain. Also, assume that $V_{dd} > V_{th}$. Start V_1 at 5 volts. Then, transistor A is off. As V_1 decreases to 4 volts, transistor A turns on. Next, the inverter output goes high, turning transistor B on when V_2 reaches 1 volt. V_2 keeps rising, which pulls down V_3 to 4 volts, which then turns on transistor C. Thus, the gain of the inverter orders events by steepening the rise of the output voltage.

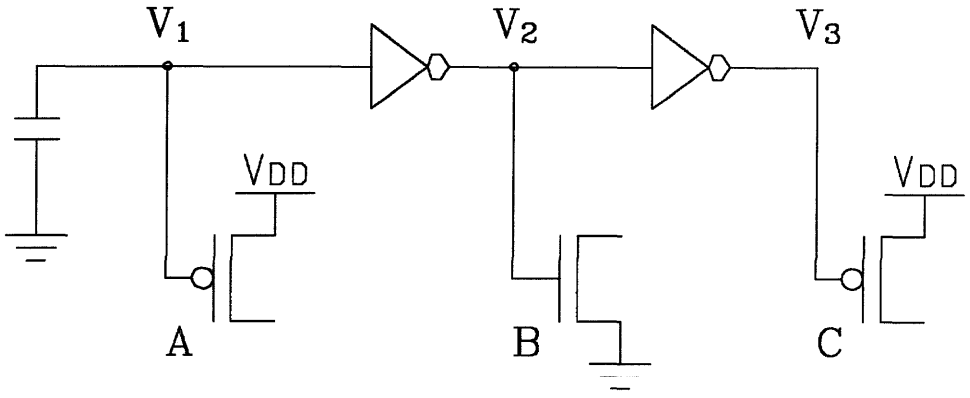

Fig. 10. Inverter Chain Used to Order Events

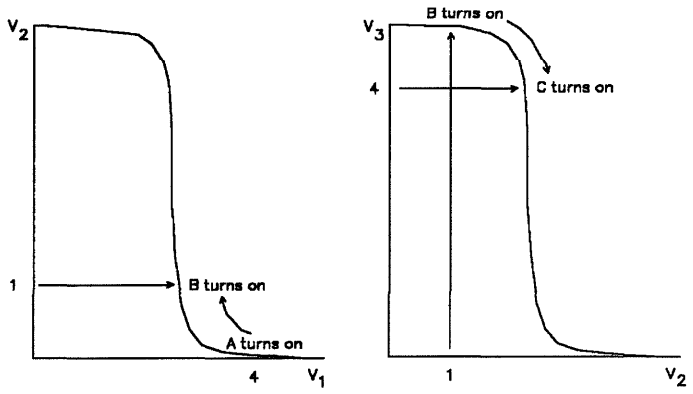

Fig. 11. Gain Used to Order Events

When V_1 starts at 0 volts, all of the transistors are on. As V_1 climbs to 2.5 volts, both V_2 and V_3 head towards 2.5 volts, also. When V_1 is just above 2.5 volts, V_3 goes to 5 volts, shutting off transistor C. V_3 goes to 5 volts first because it has the highest gain driving it. Subsequently, V_2 goes to 0 volts, shutting off transistor B. Finally, V_1 goes to 5V, shutting off transistor A.

Thus, the time ordering of events down the chain will be

$$\text{A turns on} < \text{B turns on} < \text{C turns on}$$

and

$$\text{C turns off} < \text{B turns off} < \text{A turns off.}$$

The time ordering can be generalized to more inverters. The lowest gain signals will always switch on first, and switch off last.

The RC delay of the inverter chain is assumed to be negligible in the above analysis. This assumption implies a lower bound on the time constant of the summing node. If τ_{sum} = the time constant of the summing node, τ_{inv} = the time constant of the nodes in the inverter chain, α is the gain of the inverter, and n is the number of inverters, then

$$\tau_{\text{sum}} \gg \tau_{\text{inv}} \alpha^{n-1}.$$

As an example of using an inverter chain to order events, consider the oscillator. One wants the self-excitation of i to turn on before the inhibition of $i-1$. Excitation uses an N-type transistor, while inhibition uses a P-type transistor. Thus, the self-excitation synapse needs to be a transistor similar to B in figure 11, while inhibition synapse, which needs to be slower, should be like C in figure 11. Thus, the oscillator should use the chain shown in figure 12.

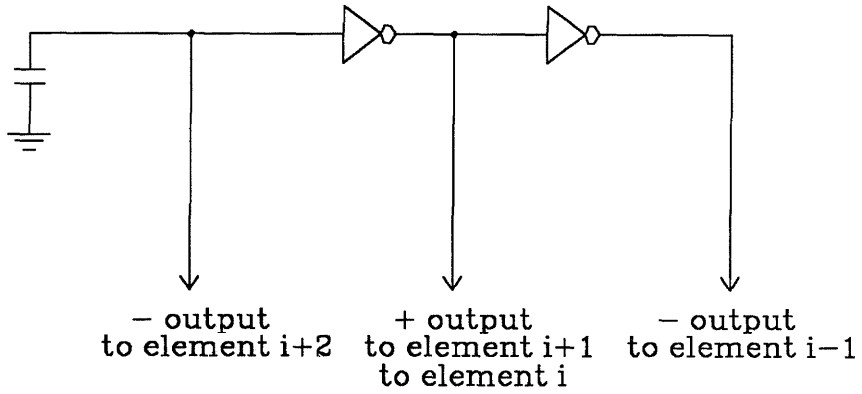

Fig. 12. Inverter Chain Used in Oscillator

In summary, the only inequalities used by the VLSI circuits are the linear ones. Namely,

$$a > g, \qquad s > g, \qquad s - b < g, \qquad a - k < g. \tag{11}$$

A good solution to these inequalities is

$$a = 2, \qquad b = 2, \qquad s = 2, \qquad k = 2, \qquad g = 1. \tag{12}$$

3.5 Experimental Results

If the inequalities are relevant to the actual operation of the circuit, then violating the inequalities should make the circuit fail. An experiment was performed on a oscillator with 4 elements. All the parameters were held fixed, except for one, which was progressively increased or decreased until the oscillator failed. The exact circuit used is presented in Appendix B. The theoretical and observed failure points are listed in the table of experiment 1, along with the theoretical and observed failure mode. A theoretical prediction of a "complex" failure means that two or more inequalities are violated close together in parameter space, thus the failure mode depends on the exact capacitances of the circuit.

Experiment 1. Discrete oscillator with all capacitors same

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	1.0	0.95	1 on	1 on
b	1.0	0.95	1 & 2 on	1 & 2 on
k	2.0	0.53	oscillates with 3	oscillates with 3
s	1.8	1.3	all off	All off
g	0.8	0.33	complex	1 & 2 on

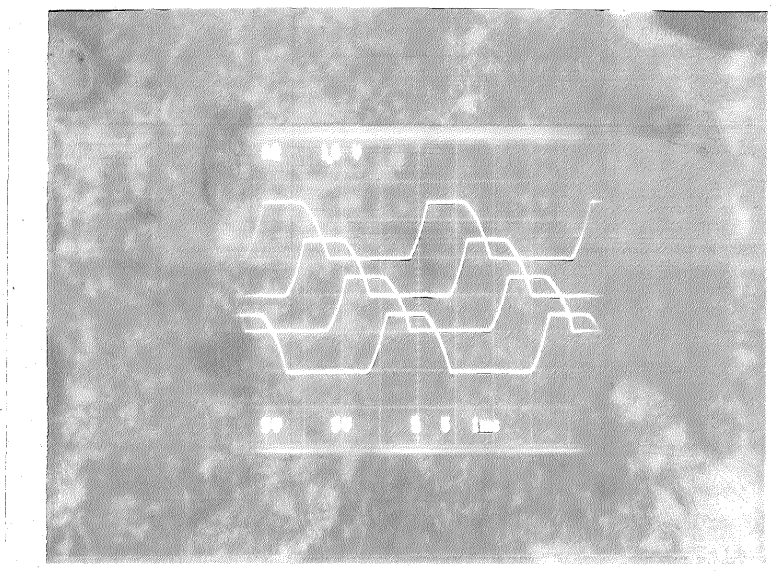

Fig. 13. Output of Oscillating Discrete Threshold Elements

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure Mode	Observed Failure Mode
a	2.5	4.3	oscillates with 3	oscillates with 3
b	2.0	4.8	all off	all off
k	∞	> 19	none	none
s	2.5	2.6	1 & 2 on	1 & 2 on
g	1.1	1.5	all off	all off

Experiment 2. Discrete oscillator with capacitors 1 and 2 100 times bigger than the rest

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	1.0	0.95	1 on	1 on
b	1.0	0.95	1 & 2 on	1 & 2 on
k	2.0	2.0	oscillates with 3	oscillates with 3
s	1.8	2.0	all off	all off
g	0.8	0.57	complex	oscillates with 3

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure Mode	Observed Failure Mode
a	2.5	2.4	oscillates with 3	oscillates with 3
b	2.0	1.9	all off	all off
k	∞	> 19	none	none
s	2.5	2.0	1 & 2 on	1 & 2 on
g	1.1	1.0	all off	all off

The observed minima and maxima do not exactly match the theoretical because the resistors corresponding to each parameter were changed by 10% increments. In experiment 1, the k, s, and g parameters' observed minima are much lower than those predicted by theory. Similarly, in experiment 1, the a, b and g parameters' maxima were much higher than that predicted by theory.

The theory was conservative because the parameters are limited by a race preventing inequalities: $s(s-g) > gb$ and $kg > (a-g)s$. The race inequalities assume a worst case: that certain elements are infinitely fast. Thus, the inequalities are conservative, and the system may work outside the theoretical bounds. Notice that when the capacitors are not as well matched, the system needs to be more speed independent, and the observed bounds on the parameters are closer to the theoretical limits.

In addition to changing the connection weights, the capacitances and the gains of the elements were changed. These experiments were done with the parameter values in (10).

The capacitances of individual elements could be varied over a factor of 1000, and the circuit will still oscillate. However, as the circuit slowed down, more and more gain was needed to keep the circuit oscillating, as shown in experiment 3. The capacitance shown is the ratio of the largest capacitor to the smallest capacitor.

Experiment 3. Gain versus capacitance

Capacitance	Gain Needed
1	11
10	30
100	75
1000	200

In the VLSI case, the experiments were done with a numerical simulation of a four element oscillator, which fully explained in Appendix C. Figure 14 shows the output of an element and it's sucessor. Notice that the traces slightly overlap: one element does not turn off until the next one turns on.

Figure 15 shows the output of elements 1 and 3. These should not overlap, and they do not.

Again, each parameter was varied independently until the circuit failed.

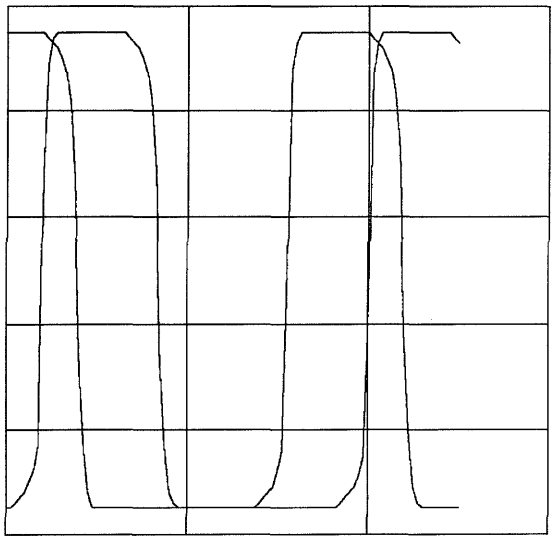

Fig. 14. Simulated VLSI Oscillator, Elements 1 and 2

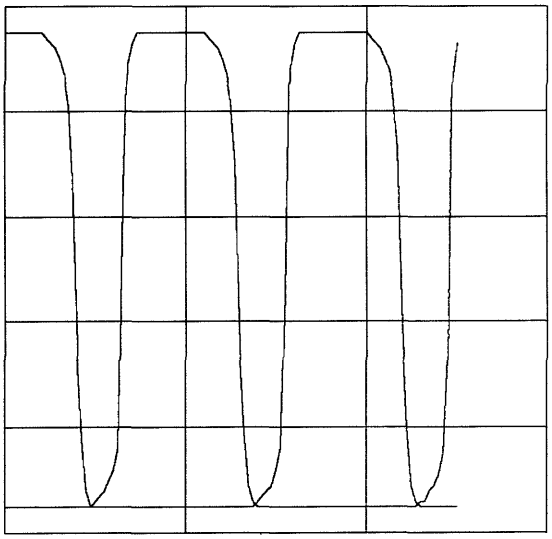

Fig. 15. Simulated VLSI Oscillator, Elements 1 and 3

Experiment 4. VLSI oscillator with all capacitors same

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	1.0	1.0	1 on	1 on
b	1.0	1.0	1 & 2 on	1 & 2 on
k	1.0	0.2	oscillates with 3	oscillates with 3
s	1.0	1.0	all off	all off
g	0.0	-0.6	complex	oscillates with 3

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure Mode	Observed Failure Mode
a	3	4.8	oscillates with 3	oscillates with 3
b	∞	> 1000	none	none
k	∞	> 1000	none	none
s	3.0	3.0	1 & 2 on	1 & 2 on
g	2.0	1.9	all off	all off

Experiment 5 VLSI oscillator with one capacitor 100 times smaller than the rest

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	1.0	1.0	1 on	1 on
b	1.0	1.0	1 & 2 on	1 & 2 on
k	1.0	1.1	oscillates with 3	oscillates with 3
s	1.0	1.1	all off	all off
g	0.0	0.1	complex	oscillates with 3

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure Mode	Observed Failure Mode
a	3.0	2.9	oscillates with 3	oscillates with 3
b	∞	> 1000	none	none
k	∞	> 1000	none	none
s	3.0	3.0	1 & 2 on	1 & 2 on
g	2.0	1.8	all off	all off

Here, there are no race inequalities, so the boundaries are matched pretty well. Also notice that there is much more room for error in the VLSI parameters, because there are fewer constraints on the parameters.

In experiment 5, The g parameter breaks down above zero because the system is not perfect; there is always some leakage current from various connections.

4 Analog Petri Nets

Petri nets arose to model systems with concurrent behaviours [Peterson 81]. In this paper, they are used in an opposite sense: to describe compactly a concurrent behaviour that needs to be implemented. Thus, a designer using the technique in this paper would give a Petri net to a program, which would then produce a circuit that implements that Petri net.

A Petri net can best be described graphically.

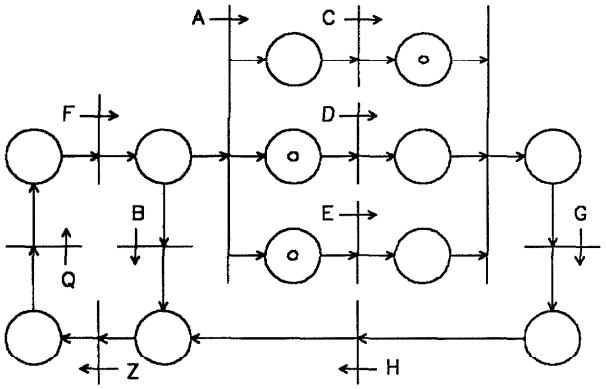

Fig. 16. A typical Petri net

In figure 16, the circles are called *"places"*, the bars are called *"transitions"*, and the small circles are called *"tokens"*. The arrows connecting the places and transitions are called *"arcs"*. A place holds a non-negative number of tokens, and the transitions serve to move tokens from place to place. A particular assignment of tokens to places is called a "marking" [Peterson 81].

A Petri net moves tokens from place to place when transitions *fire*. When a transition fires, it takes all the tokens from all the places that have arcs going into the firing transition (the *input places*), and it adds a token to all the places that have arcs coming from the transition (the *output places*). A transition can fire when it is *enabled*. If the transition has an input associated with it (denoted by a letter and arrow going through the transition), then it is enabled when all the input places have at least one token in them, and the input associated with the transition is on. When a transition has no input associated with it, it is called an *"automatic transition."* and only needs a token in each of its input places to fire.

The above definition is not the most general definition of a Petri net [Peterson 81], but it will be enough for this paper.

4.1 Examples of Petri Nets

Some simple examples should illustrate the above rules.

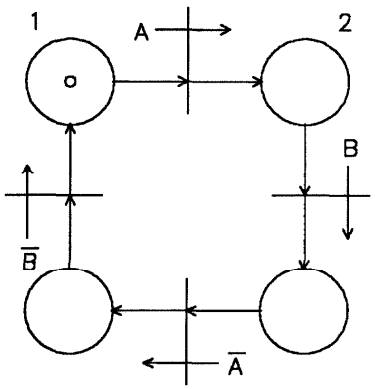

Fig. 17. A simple example

In the example in figure 17, the token will only move from place 1 to place 2 when input A is active. If the transition were automatic, then as soon as a token would appear in place 1, it would get moved to place 2.

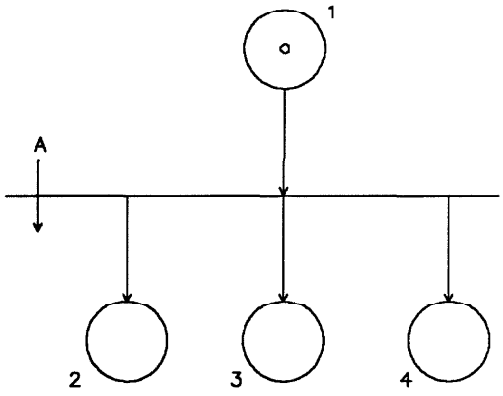

Fig. 18. A fork

Figure 18 illustrates behaviour that is known as a *"fork"*. Namely, there is more than one place following the transition. Thus, if a token appears in place 1, and input A is active, then a token must be put in all the places 2, 3, and 4, while eliminating the token in place 1.

The example in figure 19 shows the reverse of a fork, known as a *"join"*. Notice that a transition can serve as both a fork and a join. The example must have a token in places 1, 2, and 3 before the transition fires, and puts a token into place 4, destroying all the tokens in 1, 2, and 3. Notice that figure 19 shows an automatic transition.

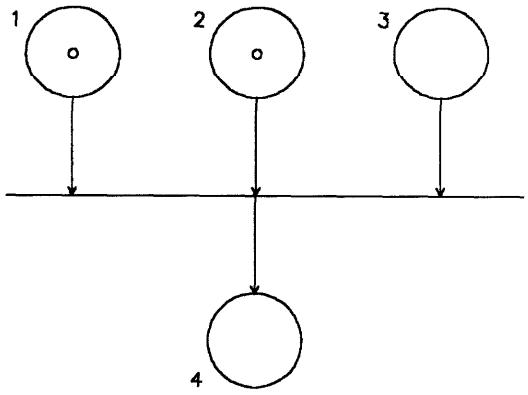

Fig. 19. A join

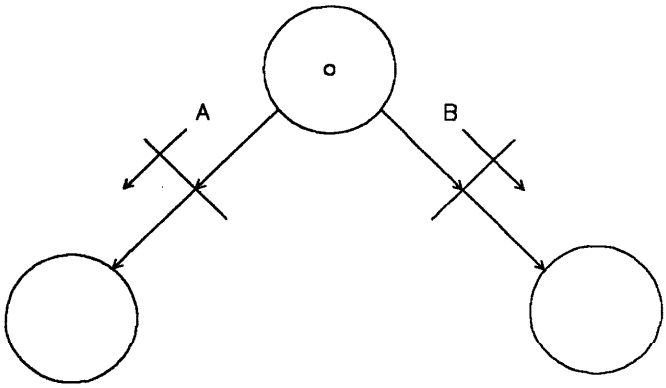

Fig. 20. Conflict

Figure 20 shows an important property of Petri nets. Consider the case where inputs A and B are both on, then a token comes into place 1. Both transitions are enabled, so both can fire. Yet, if one fires, then the other becomes disabled, so it cannot fire. Thus, there has to be mutual exclusion between all the transitions emanating from a place, since only one may fire. Having more than one transition competing for a token is known as a *"conflict"*. Conflict is important, since any decision specified by a Petri net automatically implies conflict, hence mutual exclusion.

When more than one transition is in conflict and are enabled, there should be some rule for determining which transition should fire. For example, one could pick at random amongst the enabled transition, firing the picked one, and disabling the rest. Another rule is known as *arbitration.* If the firing transition was the first enabled transition, then the net is said to provide arbitration. Arbitration can lead to the system to be metastable for an unbounded length of time, or making decisions non-deterministically.

4.2 Safe Marking

An initial marking is called *"safe"* when for all possible markings of the net that are reachable from the starting marking, there is never more than one token per place.

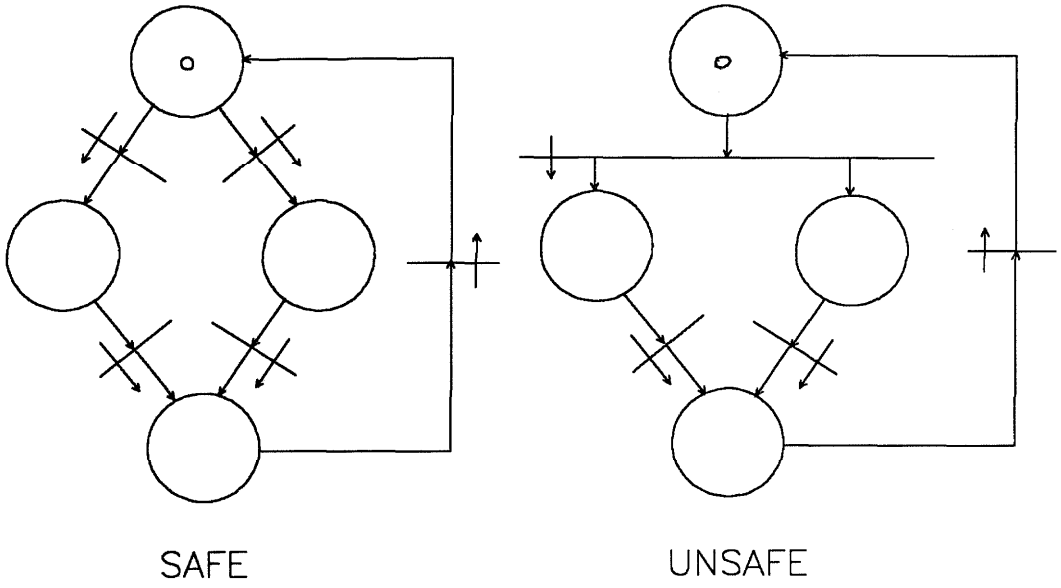

Fig. 21. Unsafe and Safe Markings

Safeness is a subset of a property called *"boundedness"*, which means that every place must have less than a certain finite number of tokens.

The synthetic technique presented in this paper will implement only safe markings of Petri nets.

4.3 Inhibitor Arcs

Normal Petri nets are not powerful enough to be Turing Universal, even when unbounded, because they cannot test for the absence a token. If the transition firing rules are modified to include *inhibitor arcs*, an unbounded Petri net can become Turing Universal. Thus, inhibitor arcs are powerful.

With inhibitor arcs, a transition is enabled when all its normal input places have tokens and none of the inhibitor places have any tokens in them. It then takes all the tokens from the normal input places when it fires. It does not take any tokens from the inhibitor places.

An inhibitor arc is a hole sensor. One has to make sure that the hole does not disappear while the inhibitor arc is preventing the transition from firing. Thus, there must be arbitration between the destination of the inhibitor arc, and any input transitions that could feed tokens to the origin of the inhibitor arc.

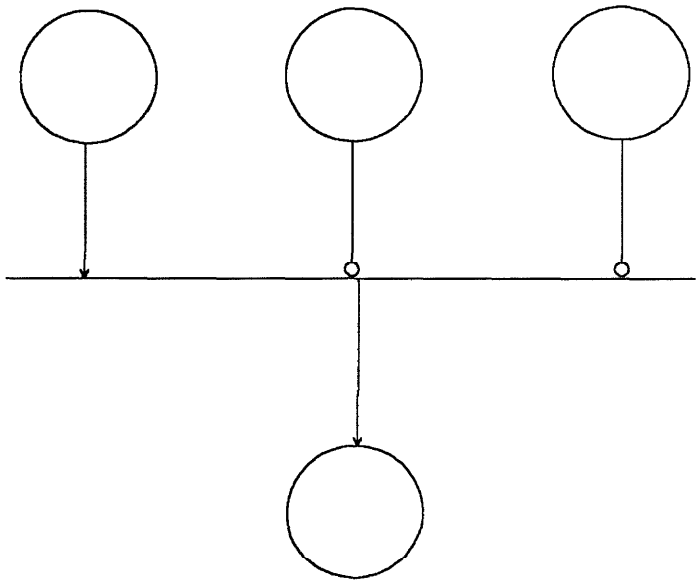

Fig. 22. Inhibitor Arcs

4.4 Analog Petri Nets

An analog Petri net is one where the inputs to the transitions can take on any analog values between 0 and 1. A transition is considered enabled when its input is above a given threshold and the normal conditions on previous places hold. As with digital Petri nets, tokens are digital objects, but the inputs are now analog.

The main difference between a normal digital Petri net and an analog Petri net is *analog arbitration.* Namely, if there is a conflict in the Petri net, and all the inputs to the transitions are on before a token arrives at the input place, the transition with the largest input should fire. Analog arbitration can not be done if the inputs are digitized.

In the case where the inputs turn on after the token is in the input place, the transition with the largest input integrated over time will fire. Thus, analog arbitration and time arbitration are performed by the same circuit.

In summary, if a designer specifies a safe analog Petri net with inhibitor arcs, then the algorithm presented in this paper should be able to turn the net into a circuit.

5 An Analog Asynchronous Sequential Machine

Now let us attack the problem of making a circuit that will implement a Petri net specification. Such a machine is called an analog asynchronous sequential machine (AASM). First, let us choose a representation for the problem, and solve the inequalities corresponding to behavioural constraints.

An analog Petri net can be made of fragments, containing places and transitions, which when composed together will form any possible analog Petri net. This section will present implementations of the Petri net fragments that can be glued together to implement an entire Petri net. The fragments are composed by using the same threshold elements for the input place of a second fragment and output place of a first fragment.

First, consider the simplest possible Petri net: one that has no conflicts, no forks or joins, and no automatic transitions. A circuit that implements such a Petri net is simply an oscillator that stops at each state until a certain input is given to the system. Such a Petri Net can be composed of fragments shown in figure 23.

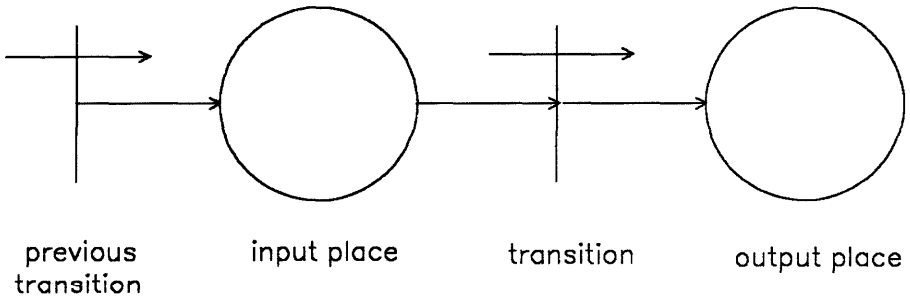

Fig. 23. A Simple Petri Net Fragment

The implementation of the fragment will b similar to the oscillator in §3. Namely, let there be one element for each place, and one element for each transition, with the input corresponding to the transition being a wire going into the transition element.

First, let us convert the behavioural constraints needed to implement a Petri net fragment to excitatory and inhibitory connections.

1) If a place or transition is on, it will tend to remain on. Thus, there should be self-excitation.
2) A transition turns on when the previous place is on and the input is on. Thus, there should be an excitatory connection from a place to a transition, and an excitatory input.
3) A place will turn on when a previous transition is on and when the place that previously held the token is off. Thus, there should be excitation from a transitions to an output place, and inhibition from the input place to the output place.
4) A place should turn off after it has fired a transition. Thus, put inhibition from a transition to the input place.

5) However, a transition should not turn on if its input place has not turned off any previous transitions. Thus, there should be inhibition from a transition to any transitions following its output place.
6) A transition should turn off after it has turned on its output place. Thus, put inhibition from the output place to the transition.

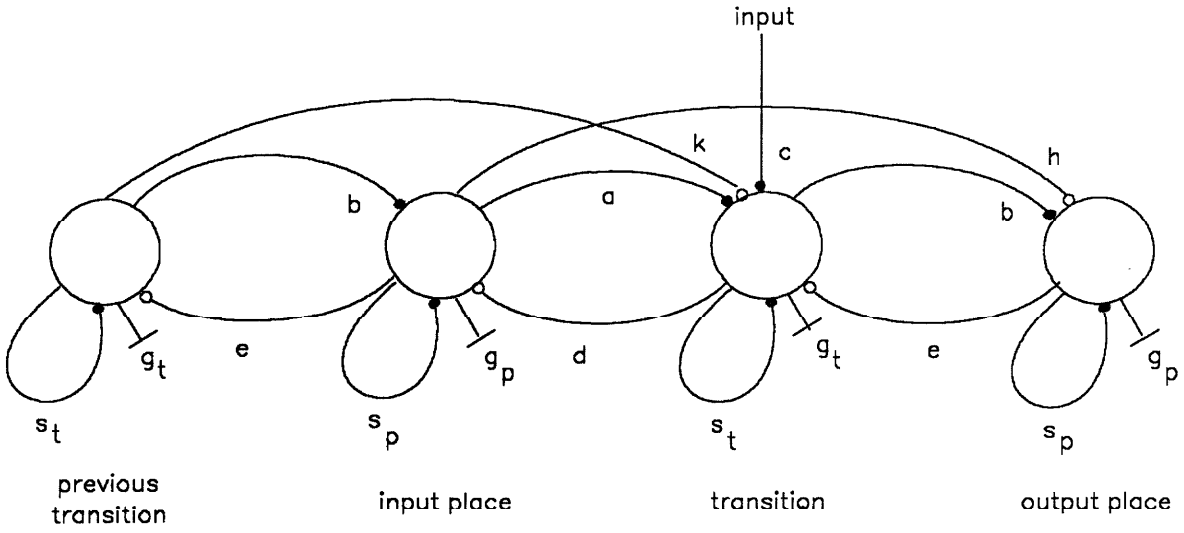

Fig. 24. Wiring Diagram for Implementing Simple Petri Net

The above wiring list is summarized in figure 24. The wiring is similar to the oscillator, except that the input moves the token along. The place-to-place and transition-to-transition inhibitory connections can be used, because the net should have a safe marking. A safe marking prevents tokens from running into each other as they travel around the net.

Now, we have the following constraints on the weights based on the behavioural constraints.

1) If a place or transition is on, it will tend to remain on, if everything else is off. Thus, $s_p > g_p$ and $s_t > g_t$.
2) A transition will turn on only if its preceding place and its input is on. Thus, $a + c > g_t, a < g_t, c < g_t$
3) A place will turn on if its preceding transition is on and the preceding place has been shut off. Thus, $b > g_p$.
4) A place should turn off if its previous transition is off, and its next transition is on. Thus, $s_p - d < g_p$.
5) Similarly, a transition will turn off when its input place is off and its output place is on, even when the input is on, which requires $s_t + c - e < g_t$.
6) A transition should not turn on if a previous transition is on. Thus, $a + c - k < g_p$.
7) Similarly, an output place should not turn on if its input place is still on. Thus, $b - h < g_p$.

For the VLSI circuit, these are all the constraints that are needed.
A summary of these constraints is

$$\begin{array}{llll} s_p > g_p, & s_t > g_t, & a + c > g_t, & \\ a < g_t, & c < g_t, & b > g_p, & \\ s_p - d < g_p, & s_t + c - e < g_t, & b - h < g_p, & a + c - k < g_t, \end{array} \tag{13}$$

A good solution to this system is

$$\begin{array}{lllll} s_t = 2, & s_p = 2, & a = 0.7, & b = 2, & c = 0.7, \\ d = 2, & e = 2, & h = 2, & k = 2 & g_p = 1, \quad g_t = 1. \end{array} \tag{14}$$

The races in these constraints are similar to those of the oscillator. Namely,

1) A place should self-reinforce before it inhibits its previous transition.
2) A place should cease self-reinforcement before it allows the next place to turn on.
3) A transition should self-reinforce before it inhibits its input place.
4) A transition should cease self-reinforcement before it allows the next transition to turn on.

For the VLSI circuit, figure 25 shows that the races are eliminated by properly assigning the outputs in the inverter chain of the elements.

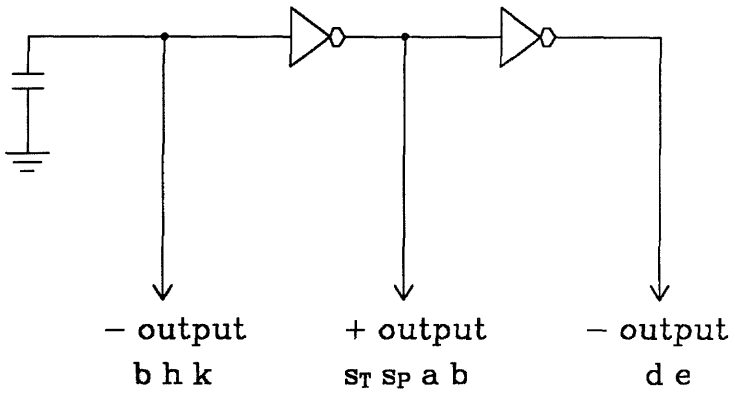

Fig. 25. Inverter Chain for Simple Fragment

In the discrete circuit with resistors, non-linear inequalities are needed to prevent races. Following the analysis of the oscillator, let

x_1 = the voltage necessary for an input place to prevent an output place from turning on,
x_2 = the voltage necessary for place self-reinforcement,
x_3 = the voltage necessary for a place to turn off its previous transition.
x_4 = the voltage necessary for a transition to prevent a succeeding transition from turning on,
x_5 = the voltage necessary for transition self-reinforcement with input,
x_6 = the voltage necessary for transition self-reinforcement without input,
x_7 = the voltage necessary for a transition to turn off its input place.

Thus,

$$b - x_1 h - g_p = 0 \Rightarrow x_1 = \frac{b - g_p}{h}, \tag{15}$$

$$x_2 s_p - g_p = 0 \Rightarrow x_2 = \frac{g_p}{s_p}, \tag{16}$$

$$s_t - x_3 e - g_t = 0 \Rightarrow x_3 = \frac{s_t - g_t}{e}, \tag{17}$$

$$a + c - x_4 k - g_t = 0 \Rightarrow x_4 = \frac{a + c - g_t}{k}, \tag{18}$$

$$x_5 s_t + c - g_t = 0 \Rightarrow x_5 = \frac{g_t}{s_t + c}, \tag{19}$$

$$x_6 s_t - g_t = 0 \Rightarrow x_6 = \frac{g_t}{s_t}, \tag{20}$$

$$s_p - x_7 d - g_p = 0 \Rightarrow x_7 = \frac{s_p - g_p}{d}. \tag{21}$$

Prevention of races requires

$$x_3 > x_2 > x_1, \qquad x_5 > x_4, \qquad x_7 > x_6. \tag{22}$$

which implies

$$\begin{aligned} g_p h > s_p(b - g_p), \qquad & s_p(s_t - g_t) > g_p c \\ g_t k > (a + c - g_t)(s_t + c), \qquad & s_t(s_p - g_p) > g_t d. \end{aligned} \tag{23}$$

A good solution to entire system of both (13) and (23) is

$$\begin{aligned} s_t = 1.5, \qquad s_p = 2.5, \qquad a = 0.8, & \\ b = 1.4, \qquad c = 0.4, \qquad d = 2.0, \qquad & e = 1.0, \\ h = 1.6, \qquad k = 0.6, \qquad g_p = 1.0, \qquad & g_t = 1.0. \end{aligned} \tag{24}$$

Experimental results using this solution are presented in §5.3

5.1 Conflict in the Petri Net

A Petri net without conflicts is not useful, because it cannot make decisions. The network developed above cannot handle net conflicts, because it does not have mutual exclusion between the possible decisions. Mutual exclusion has to be put in explicitly. Hence, consider the basic conflict fragment shown in figure 26.

The representation will be as before, with one element per place and transition, except now there should be mutual inhibition between the possible decisions. The wiring diagram is shown in figure 27.

The mutual inhibition serves as an flip-flop to make the decisions mutually exclusive. Mutual exclusion is still needed by more than two conflicting transitions, thus the circuit will contain an *N-flop*, an N-way mutual inhibition between all the conflicting transitions.

There are two ways of implementing an N-way mutual inhibition. One could make inhibitory connections from all of the other competing transitions to a transition. In this case, the inhibition is proportional to the sum of the outputs of the other competing transitions, which makes the analysis for the discrete circuit easy.

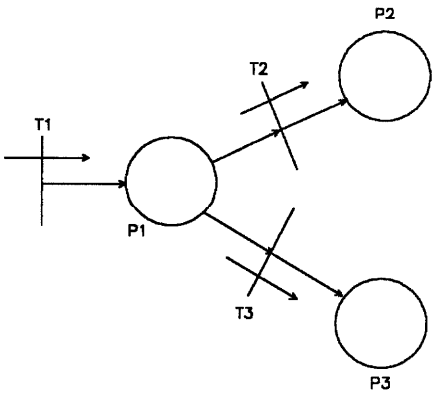

Fig. 26. Basic Conflict Element

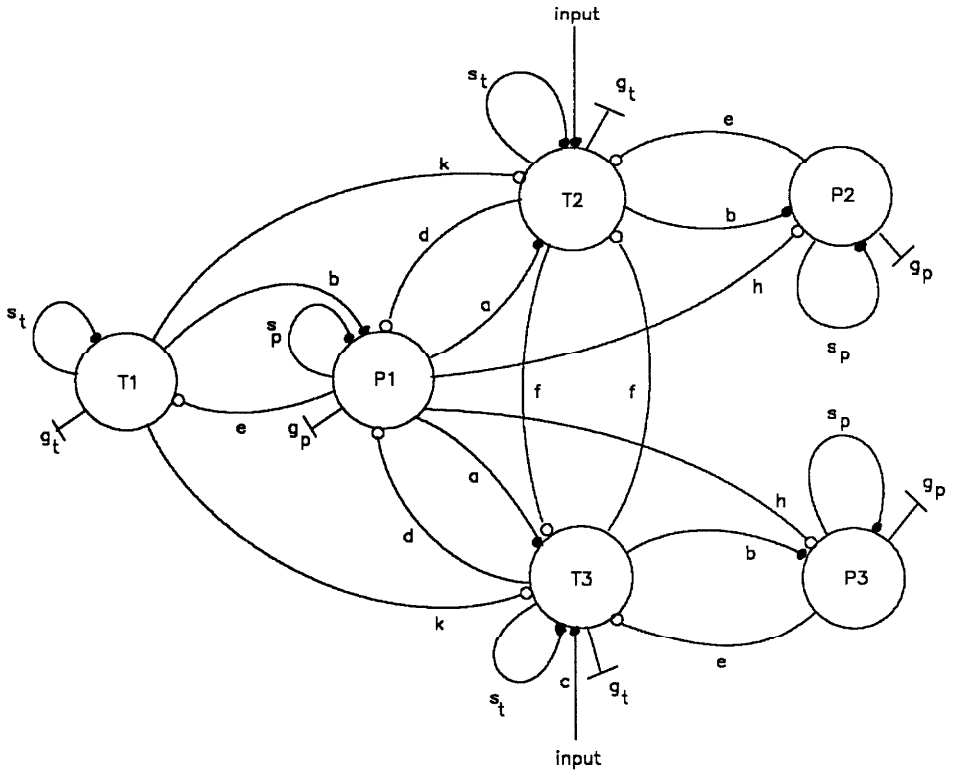

Fig. 27. Wiring Diagram for Conflict

One could also take the OR of all of the other competing transitions, which gives an inhibition proportional to the maximum of the outputs of the other competing transitions. The OR was used in the simulated VLSI N-flop, where it was implemented using pass gate logic.

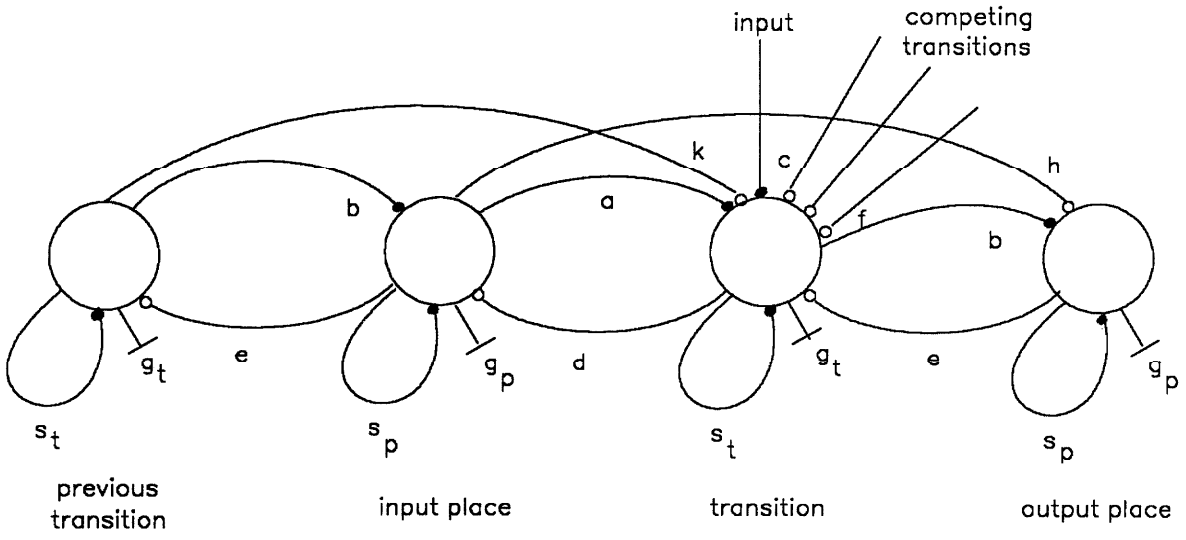

Fig. 28. Wiring Diagram for Summed Mutual Inhibition

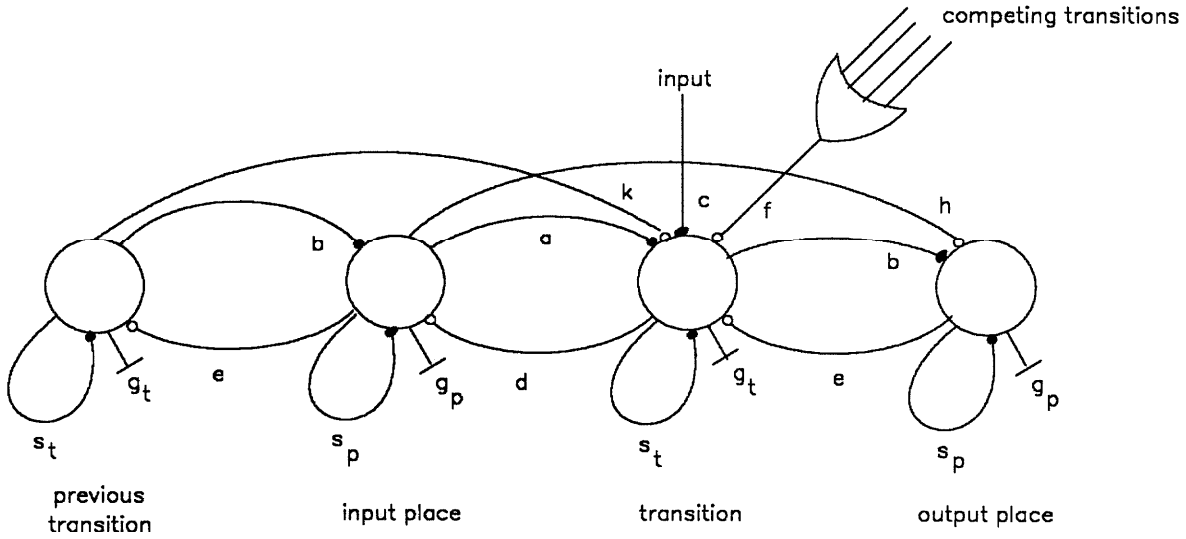

Fig. 29. Wiring Diagram for ORed Mutual Inhibition

The constraints on the behaviour of the simple fragment still apply, but now the mutual exclusion must be added explicitly. Thus, the mutual inhibition f is added.

Extra constraints are needed to solve for f. Namely, if a transition is on, no other transition should turn on, which implies

$$f > a + c - g_t \tag{25}$$

Unfortunately, the mutual inhibition introduces more races. There is a race between a transition trying to be self-reinforcing and a transition being shut off by a competitor. One wants the mutual inhibition to win over the self-reinforcement so that only one transition can turn on. Also, there is a race between the mutual inhibition trying to shut a transition off and the transition trying to turn on its output place. Finally, there is a race between the mutual inhibition and the transition trying to shut off its input place. In both these cases, the mutual inhibition should win, because only one transition should be permitted to do anything.

In the discrete circuit, consider the worst case: two competing transitions that start to turn on at exactly the same time, with exactly the same input. Ideally, the N-flop should stay at a metastable voltage V_∞. V_∞ is reached when the inhibition is equal to the excitation:

$$a + c + V_\infty s_t - V_\infty f = g_t \Rightarrow V_\infty = \frac{a + c - g_t}{f - s_t}. \tag{26}$$

Since $a + c - g_t > 0$, to get metastability at $V_\infty > 0$ requires

$$f > s_t. \tag{27}$$

Thus, the race between self-reinforcement and mutual inhibition is resolved.

If the transitions are in a metastable state, they should not turn on any output places, or turn off the input place. Thus,

$$V_\infty b < g_t, \qquad s_p - V_\infty d > g_p \tag{28}$$

which imply

$$g_t(f - s_t) > b(a + c - g_t), \qquad (s_p - g_p)(f - s_t) > d(a + c - g_t). \tag{29}$$

If one treats f as a free variable, and substitute the parameters from (24) into (29), one gets

$$f > 1.8. \tag{30}$$

Thus, a good choice for f is

$$f = 2.5. \tag{31}$$

In the VLSI circuit, the inputs to f connections should turn on before s_t, b, or d. Thus, the inverter chain that should be used in the threshold element is shown in figure 30. For the VLSI circuit, it is adequate to take

$$f = 1.0 \tag{32}$$

5.2 Special Cases

To implement the behaviour of an arbitrary Petri net, some special case fragments must be considered.

The first case to worry about is an automatic transition, which is accomplished by tying an input permanently on. Fixing a voltage, as opposed to adding a new parameter helps to limit the number of parameters for the system. This fragment will work with all the other cases, including conflict.

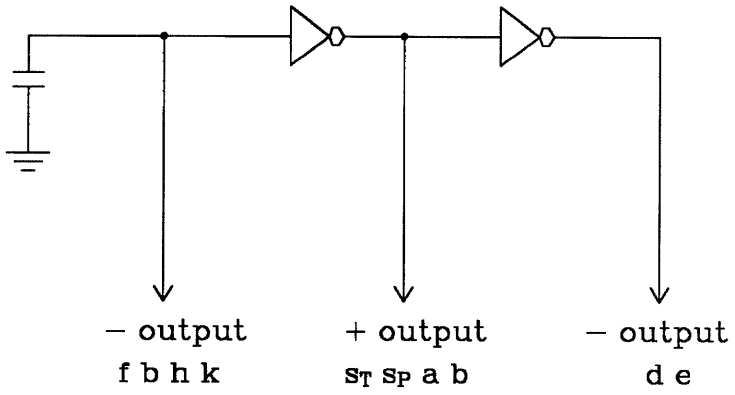

Fig. 30. Inverter Chain for Mutual Inhibition

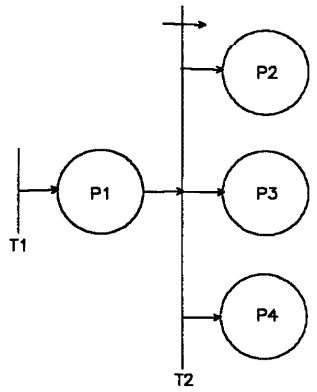

Fig. 31. Fork Fragment

Another type of Petri net fragment is a multiple fork or join. A multiple fork requires a transition to wait until *all* of its output places are on before it can shut off. To do this for an arbitrary number of output places requires arbitrarily accurate threshold element. Thus, one should use an AND gate for the inhibition of the multiple fork transition.

In the VLSI circuit, the AND is implemented with pass gate logic. In the discrete circuit, the AND is implemented with a CMOS gate, as explained in appendix B. CMOS gates have non-linear transfer responses, however, and hence affects the race inequalities. Namely, if the threshold of the CMOS logic is 0.5, then the output places will turn off the transitions when all of them reach an output voltage of 0.5. Thus, the x_3 in equation (17) is always 0.5. Thus, the inequalities in (22) imply

$$0.5 > x_2 \Rightarrow 0.5 > \frac{g_p}{s_p} \Rightarrow s_p > 2g_p. \tag{33}$$

The inequality in (33) is satisfied by the parameter values given in (24).

A multiple join is similar to a multiple fork, because it requires *all* of its input places to be on before it can turn on. Again, this needs an AND, because there could be an arbitrary number of input places. This AND is implemented by an AND pass gate in the VLSI circuit, and by CMOS logic in the discrete circuit.

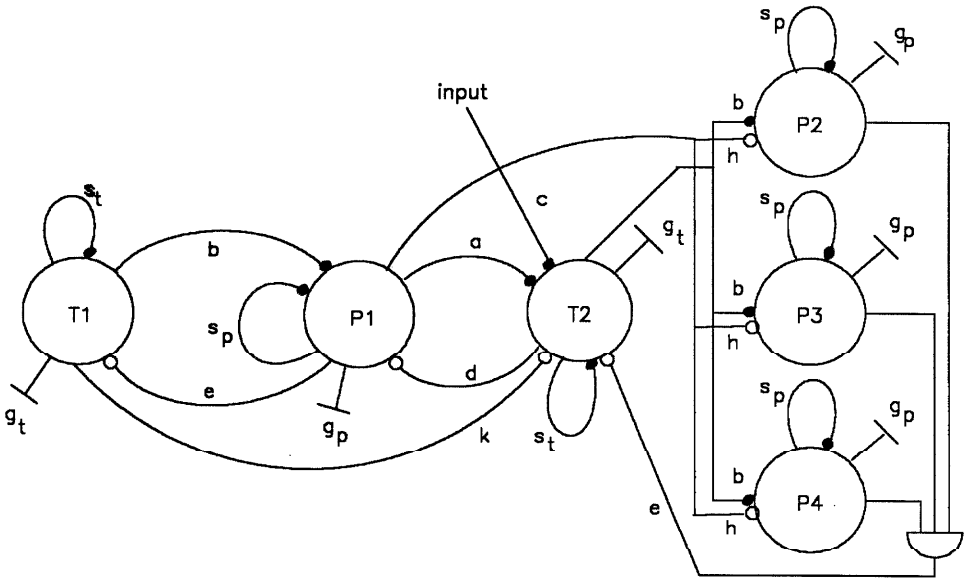

Fig. 32. Wiring Diagram for a Fork Fragment

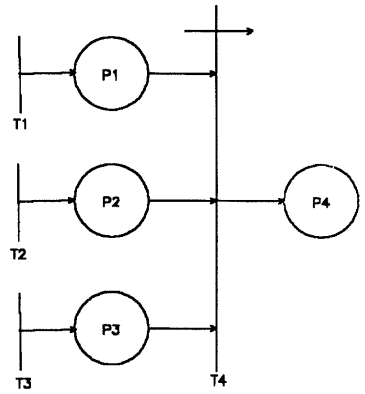

Fig. 33. Join Fragment

Since the a connection is not involved in any of the race inequalities, there are no extra nonlinear inequalities needed. However, in the discrete circuit, there is a difficulty with the h and k connections. There are two different ways of treating the h connections. One could simply OR all of the input places to the multiple join, then feed the result to the output place with an inhibitory strength of h. Similarly, one could OR the previous transitions to the multiple join, and feed the result to the multiple join, with an inhibitory strength of k. The VLSI simulations handled the h and k connections in exactly this way, using pass gate logic for the ORs.

The problem with using ORs in the discrete circuit is that the CMOS logic has nonlinear transfer functions. If one assumes a threshold of 0.5, then the x_1 in equation (15) is 0.5. Thus, (22) imply

$$x_2 > 0.5. \tag{34}$$

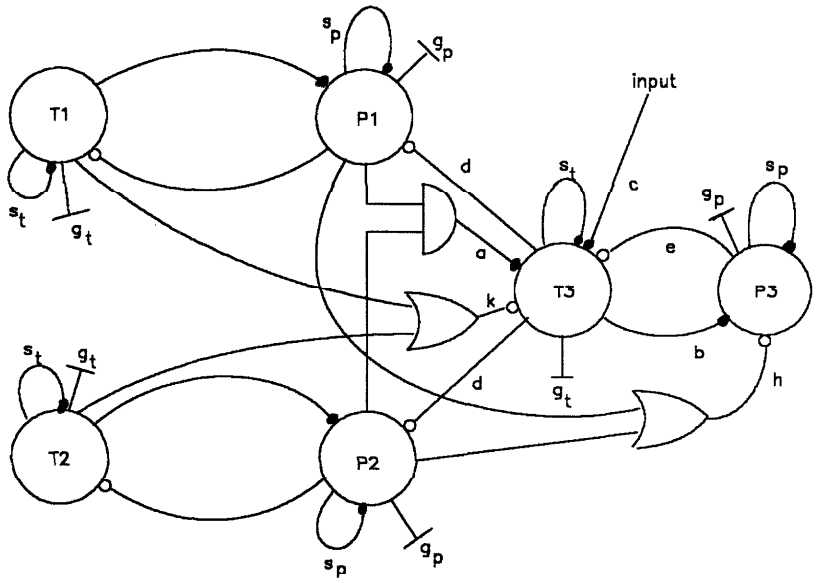

Fig. 34. VLSI Wiring Diagram for a Join Fragment

which directly contradicts (33).

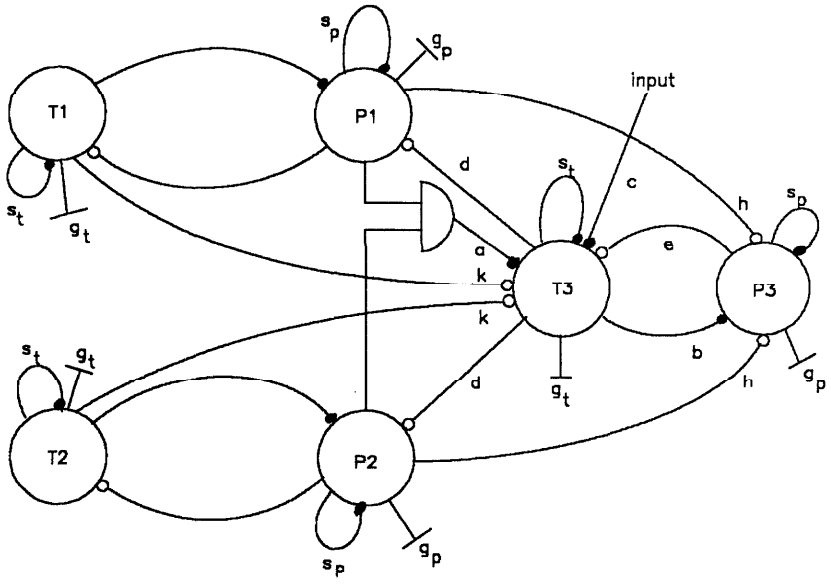

Fig. 35. Discrete Wiring Diagram for a Join Fragment

The solution to the above problem is to use a wired OR for the k connections. The wired OR is shown in figure 35. Namely, make a k inhibitory connection from each previous transition to the multiple join transition. This is permissible, since k does not have an upper bound in (13) or (23): inhibiting with a strength of nk will work just as well as k, where $n \geq 1$. Using a wired OR also makes the circuit simpler, with fewer CMOS gates to wire.

Similarly, the h connections of a multiple join can be implemented using a wired OR, since h has no upper bound in (13) or (23).

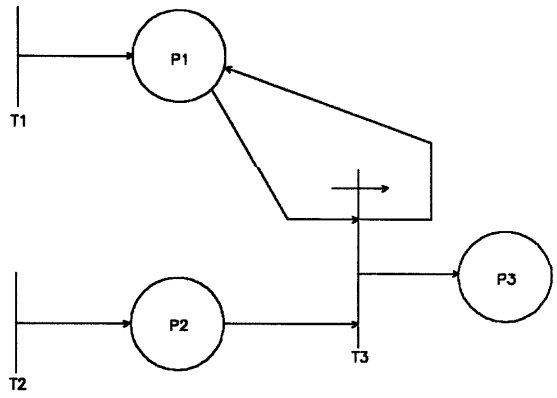

Fig. 36. A Petri Net with a Self-Loop

Another case to consider is a *self-loop.* A self-loop is a place that serves as both an input place and an output place for the same transition. This intuitively means that a token in the place helps to enable the transition, but does not get consumed in the firing of the transition.

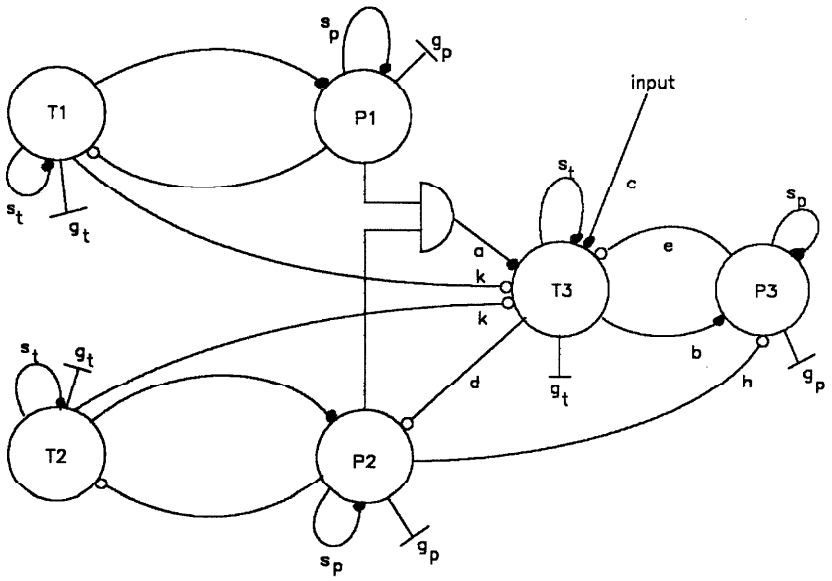

Fig. 37. Wiring for a Self-Loop (Discrete Circuit)

Thus, if a transition has a place that is both an input and an output, then

$$d = 0, \tag{35}$$

to make sure the token is not consumed. Notice that there still can be conflict between a self-loop transition and a normal transition, and there still should be mutual inhibition between the transitions. Also notice that the self-loop a connection should get ANDed in

with any other *a* connections, if the transition has a multiple join. An *h* connection should not be used between the self-loop place and the output place, otherwise the output place will never turn on. There should be a *k* connection between the transitions before the self-loop place and the self-loop transition.

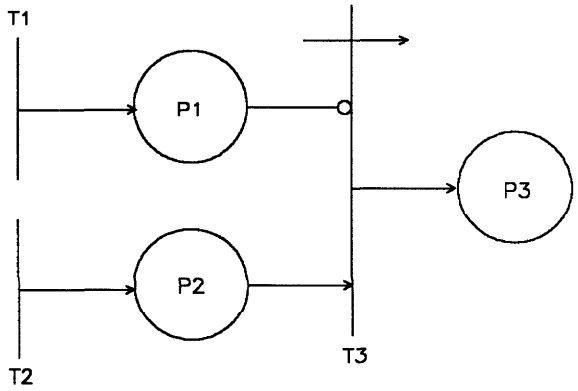

Fig. 38. Inhibitor Arc Petri Net

The last case to consider is making inhibitor arcs, which are similar to multiple joins. The difference is that the logic is reversed: the transition should not fire if the inhibitor place is on. Thus, the AND gate should receive an inverted output from any inhibitory places. There should be no *h* or *k* connections from the inhibitor places to the inhibitor transition or the output places. There should, however, be mutual inhibition between the transitions before the inhibitor place and the inhibitor transition, as discussed in §4.3.

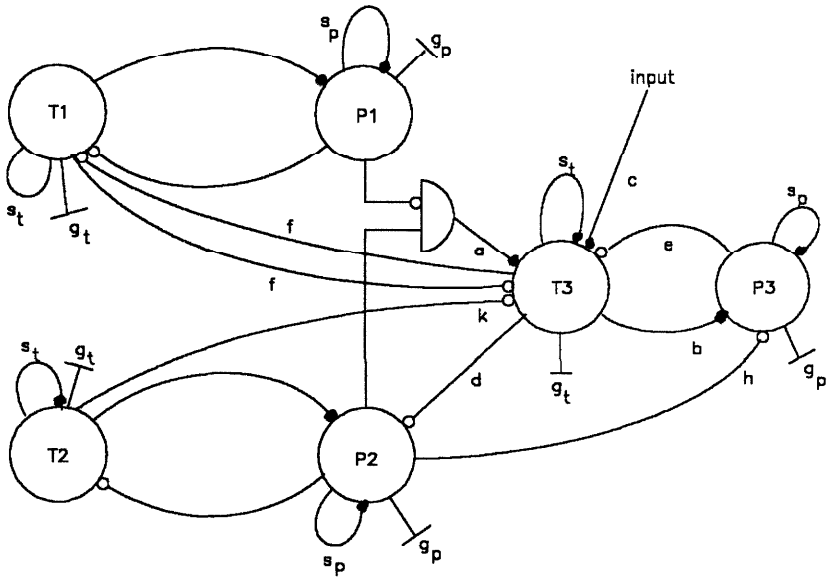

Fig. 39. Inhibitor Arc Wiring Diagram

5.3 Discrete Circuit Experimental Results

The following Petri net was implemented using the circuit in figure 3, using the parameter values from equations (24) and (31). Notice the Petri net has some of the special cases, such as a conflict, a fork, and a join.

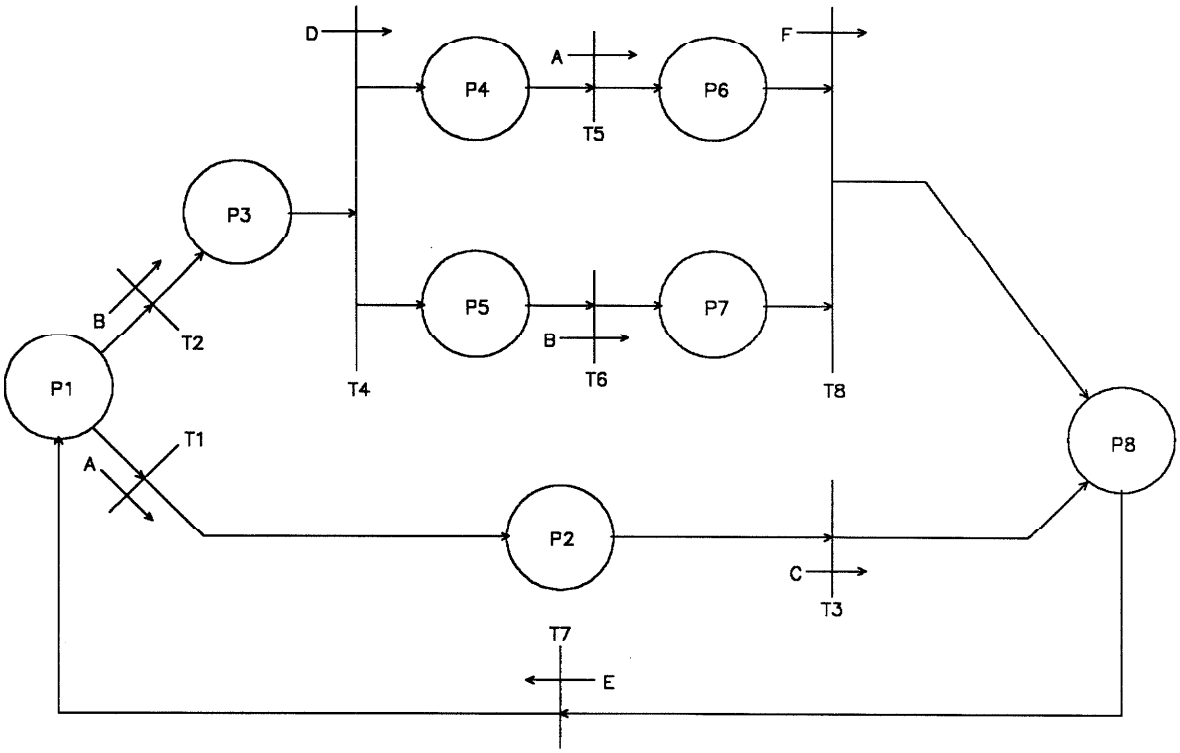

Fig. 40. Experimental Petri Net

The circuit's behaviour was precisely the same as the specified Petri net, thus showing that the synthetic technique in chapter 3 does indeed work.

If one sets all the inputs greater than 0.5, and starts the circuit with one place on, then the circuit turns into an oscillator, as can be seen in figure 41, which is an oscilloscope trace of the output voltages of 4 sequential places in the net. One can also make an unsafe marking by putting two tokens in the net. The two tokens still propagate through the net, as seen in 42.

If the circuit is made to oscillate and if input A is made larger than input B, the circuit will travel exclusively down the upper branch of the Petri net in figure 40. As A is decreased, or B is increased, the system becomes metastable, then finally only proceeds down the lower branch of the net. The mutual inhibition is evident in figure 43, where the output of two mutually inhibiting transitions is shown. First, both grow, then the N-flop makes a decision, and one transition turns on, and the other one turns off. Thus, the circuit does analog arbitration in practice, as well as in theory.

To check if the constraints really control the behaviour of the circuit, the connection strengths were changed systematically, until the circuit failed to oscillate. In experiments below, the behaviour of "ringing" means that a place or transition would not turn itself

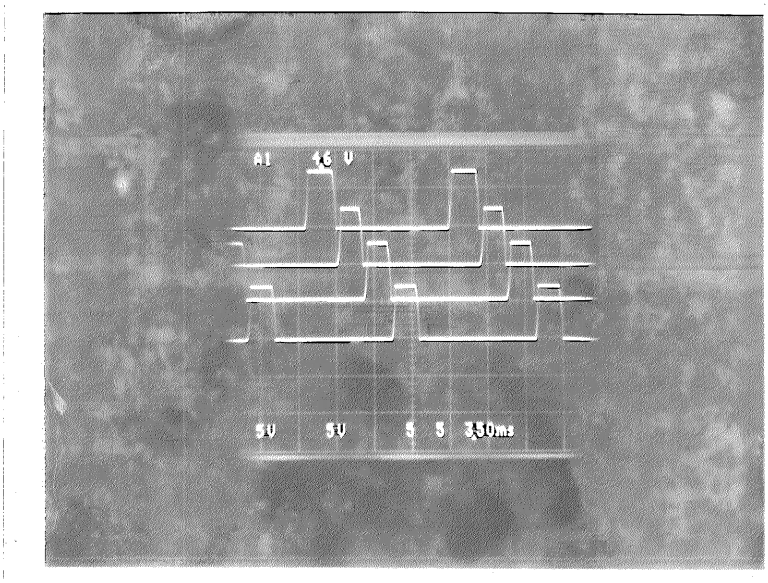

Fig. 41. One Token Travelling through the Net

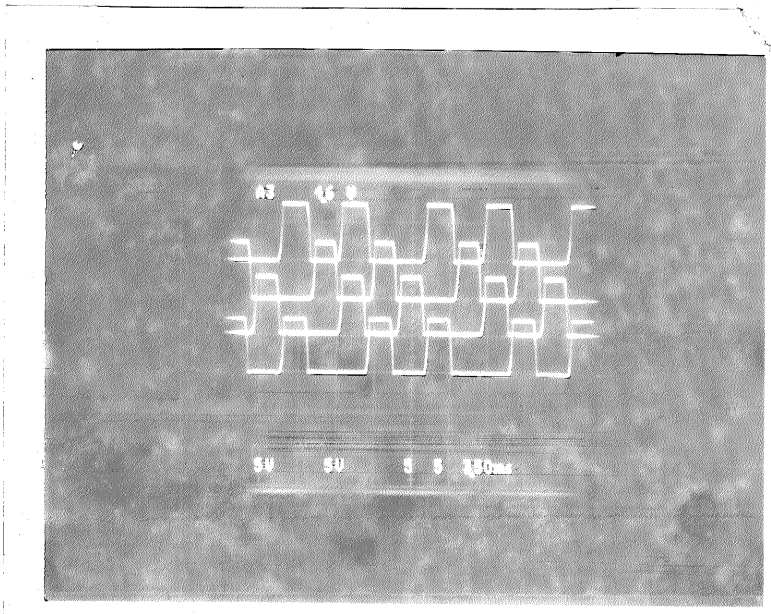

Fig. 42. Two Tokens Travelling through the Net

on, then turn on the next element, then turn itself off. Rather, it would oscillate a few times before finally shutting off.

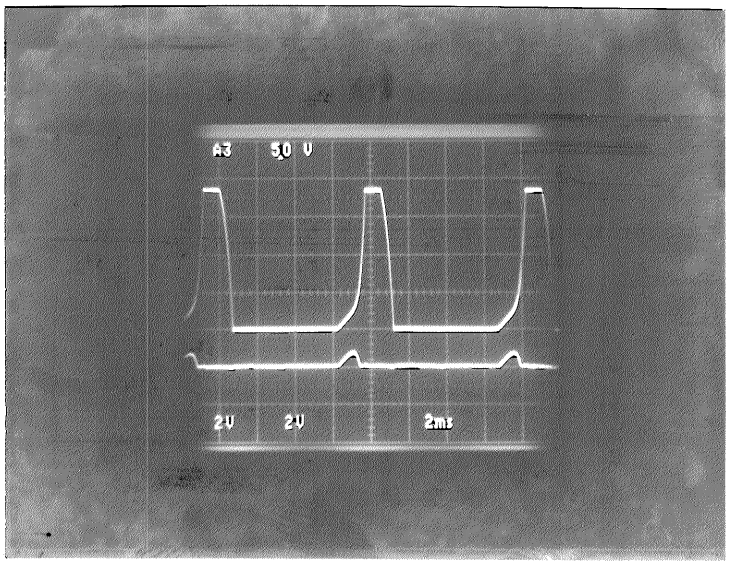

Fig. 43. Output of Mutually Inhibiting Transitions

Experiment 6. Discrete AASM with equal capacitors

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	0.6	0.61	P on	P on
b	1.0	1.1	T on	T on
c	0.2	0.20	P on	P on
d	1.5	1.5	P & T on	P & T on
e	0.9	0.98	P & T on	P & T on
f	1.8	1.9	no mutual inhibition	no mutual inhibition
k	0.4	0	oscillates with 2P	none
h	1.0	0	oscillates with 2T	none
s_t	1.3	0.5	token loss	token loss
s_p	2.0	1.3	token loss	token loss
g_t	0.9	0.80	complex	P & T on
g_p	0.5	0.46	complex	P & T on

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure mode	Observed Failure mode
a	1.0	0.89	off T fires	off T fires
b	1.6	3.3	oscillates with 2P	oscillates with 2P
c	1.0	0.88	T on with no P	T on with no P
d	2.8	5.0	token loss	token loss
e	2.2	> 10	token loss	none
f	∞	> 10	none	none
k	∞	> 10	none	none
h	∞	> 10	none	none
s_t	1.6	1.7	P & T on	P & T on
s_p	3.0	2.8	P & T on	P & T on
g_t	1.1	1.1	complex	P on
g_p	1.2	1.3	complex	T on

Experiment 7. Discrete AASM with P5, P6, T7, and T8 capacitors 100 times larger

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	0.6	0.61	P on	P on
b	1.0	1.1	T on	T on
c	0.2	0.20	P on	P on
d	1.5	1.7	P & T on	P & T on
e	0.9	0.98	P & T on	P & T on
f	1.8	1.9	no mutual inhibition	no mutual inhibition
k	0.4	0.42	oscillates with 2P	oscillates with 2P
h	1.0	0.81	oscillates with 2T	oscillates with 2T
s_t	1.3	0.98	token loss	token loss
s_p	2.0	2.1	token loss	token loss
g_t	0.9	0.74	complex	"rings"
g_p	0.5	0.55	complex	"rings"

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure mode	Observed Failure mode
a	1.0	0.89	off T fires	off T fires
b	1.6	1.9	oscillates with 2P	oscillates with 2P
c	1.0	0.88	T on with no P	T on with no P
d	2.8	3.3	token loss	token loss
e	2.2	2.5	token loss	none
f	∞	> 10	none	none
k	∞	> 10	none	none
h	∞	> 10	none	none
s_t	1.6	1.7	P & T on	P & T on
s_p	3.0	2.8	P & T on	P & T on
g_t	1.1	0.97	complex	local oscillation
g_p	1.2	1.1	complex	lost token

In experiments 6 and 7, the inputs to the transitions were applied continously. The inequalities in (13) and (23) are therefore too strict, since they assume that the input to a transition may go away after the transition is turned on. A weaker set of constraints that were used to calculate the theoretical limits are

$$\begin{aligned} s_p > g_p, \quad s_t + c > g_t, \quad & a + c > g_t, \\ a < g_t, \quad c < g_t, \quad & b > g_p, \\ s_p - d < g_p, \quad s_t + c - e < g_t, \quad & b - h < g_p, \quad a + c - k < g_t, \\ g_p h > s_p(b - g_p), \quad & s_p(s_t + c - g_t) > g_p e \\ g_t k > (a + c - g_t)(s_t + c), \quad & (s_t + c)(s_p - g_p) > g_t d. \end{aligned} \tag{36}$$

Also, to test the maximum a, T7 was turned off, so that the circuit failed by oscillating. To test maximum c, an initial condition of all off was given to the circuit. The circuit then failed by turning on some elements.

As in experiment 1, when the capacitors are all the same, the theory is very conservative. Again, the theory is conservative because the nonlinear inequalities assume that certain elements are changing infinitely quickly. When the capacitors become very different, the circuit must become more speed independent, hence the observed bounds tighten up. The remaining 10% error can be attributed to the fact that the measurements were done with resistors that were made in 10% intervals. In summary, it seems that the theory matched the experiment well.

5.4 Numerical Simulations of a VLSI AASM

A VLSI AASM was simulated using the method outlined in Appendix C. To check that mutual inhibition works for a reasonable number of elements, the Petri net in figure 44 was simulated.

In figure 45, the voltage of the summing node of the mutually inhibiting elements is shown. Notice that the element that is pulled down fastest wins, and pulls back up all the other transitions.

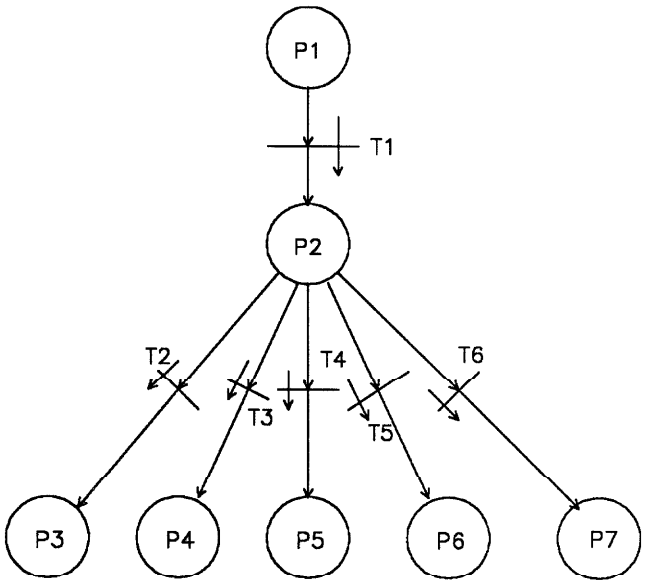

Fig. 44. VLSI Experimental N-flop

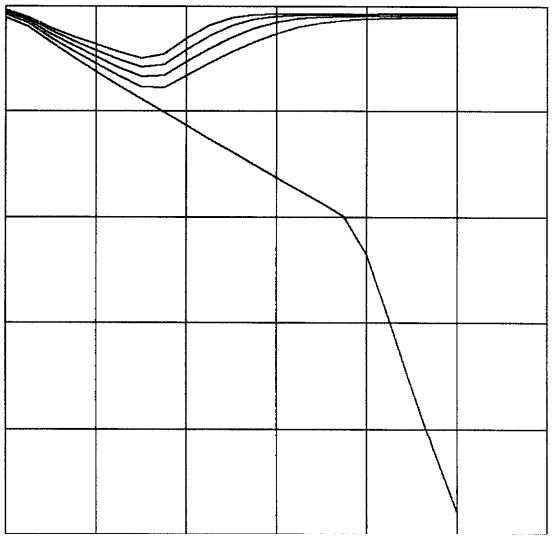

Fig. 45. Output of Competing Transitions

To test the special cases, two other Petri nets were tested. Figure 46shows a test of a fork and join. The circuit still waited for all of the input places, even when one of them was 100 times slower than the rest. Figure 47shows a test of an inhibitor arc and self-loop. Again, the circuit behaved as specified, even when the capacitances were varied by a factor of 100.

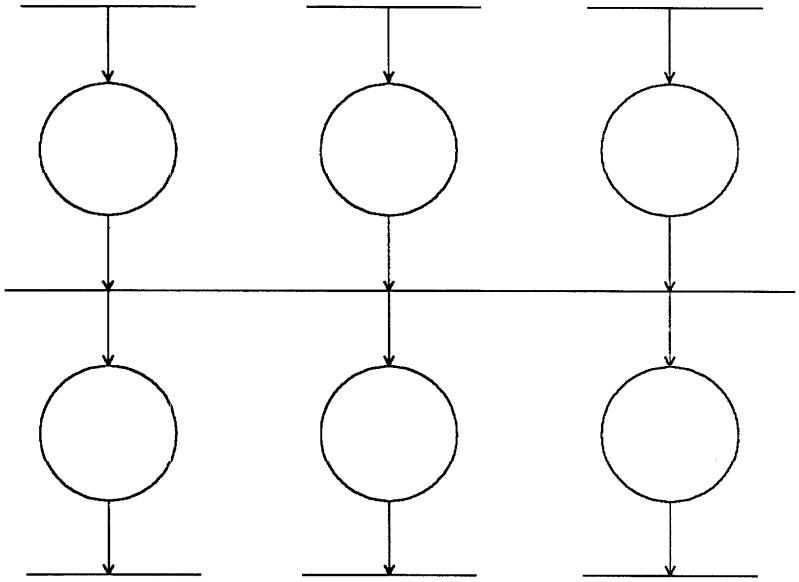

Fig. 46. Fork and Join Test

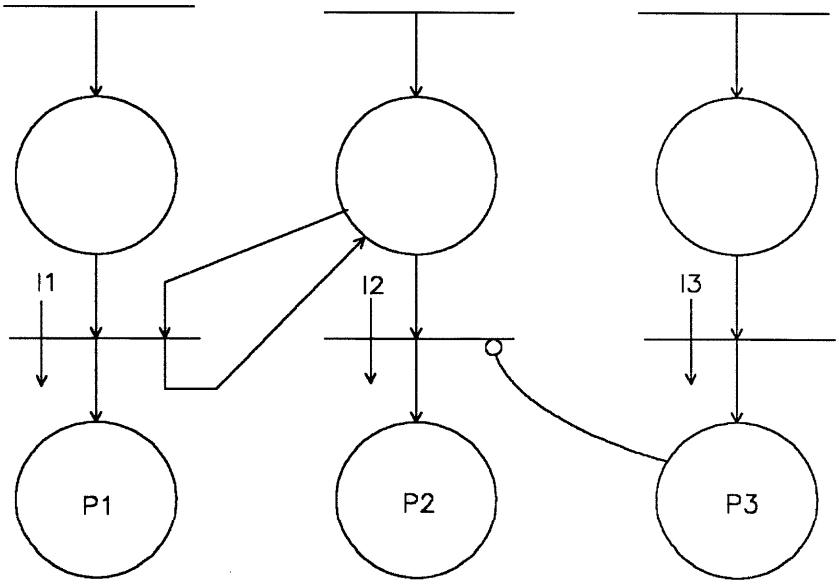

Fig. 47. Self-loop and Inhibitor Arc Test

Experiment 8. Simulated VLSI N-flop with equal capacitors

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	0.3	0.4	P on	P on
b	1.0	1.0	T on	T on
c	0.3	0.4	P on	P on
d	1.0	1.0	P & T on	P & T on
e	1.7	1.7	P & T on	P & T on
f	0.4	0.4	no mutual inhibition	no mutual inhibition
k	0.4	0	extra tokens	none
h	1.0	0.1	extra tokens	extra tokens
s_t	0.3	0	token loss	none
s_p	1.0	1.1	token loss	token loss
g_t	0.7	0.8	T always fires	T always fires
g_p	0.7	0.1	complex	all P on

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure mode	Observed Failure mode
a	1.0	1.0	off T fires	off T fires
b	3.0	3.7	extra tokens	extra tokens
c	1.0	1.0	T fires without P	T fires without P
d	∞	> 1000	none	none
e	∞	> 1000	none	none
f	∞	> 1000	none	none
k	∞	> 1000	none	none
h	∞	> 1000	none	none
s_t	3.3	3.3	P & T on	P & T on
s_p	3.0	3.0	P & T on	P & T on
g_t	1.4	1.3	P on	P on
g_p	2.0	2.0	token loss	token loss

Experiment 9. Simluated VLSI N-flop with P2 and T3 capacitors 100 times smaller

Parameter	Theoretical Minimum	Observed Minimum	Theoretical Failure mode	Observed Failure mode
a	0.3	0.4	P on	P on
b	1.0	1.0	T on	T on
c	0.3	0.4	P on	P on
d	1.0	1.0	P & T on	P & T on
e	1.7	1.7	P & T on	P & T on
f	0.4	0	no mutual inhibition	none
k	0.4	0.4	extra tokens	extra tokens
h	1.0	1.1	extra tokens	extra tokens
s_t	0.3	0.3	token loss	token loss
s_p	1.0	1.1	token loss	token loss
g_t	0.7	0.8	T always fires	T always fires
g_p	0.7	0.1	complex	local oscillation

Parameter	Theoretical Maximum	Observed Maximum	Theoretical Failure mode	Observed Failure mode
a	1.0	1.0	off T fires	off T fires
b	3.0	2.9	extra tokens	extra tokens
c	1.0	1.0	T fires without P	T fires without P
d	∞	> 100	none	none
e	∞	> 100	none	none
f	∞	> 100	none	none
k	∞	> 100	none	none
h	∞	> 100	none	none
s_t	3.3	3.3	P & T on	P & T on
s_p	3.0	3.0	P & T on	P & T on
g_t	1.4	1.2	P on	P on
g_p	2.0	1.9	token loss	token loss

As in experiments 6 and 7, the experiments were carried out with the inputs on continuously. Thus, the theoretical limits were based on

$$\begin{aligned} s_p > g_p, \quad & s_t + c > g_t, \\ a + c > g_t, \quad & a < g_t, \\ c < g_t, \quad & b > g_p, \\ s_p - d < g_p, \quad & s_t + c - e < g_t, \\ b - h < g_p, \quad & a + c - k < g_t, \end{aligned} \tag{37}$$

Also, the maximum of a and the minimum of g_t was tested with all inputs off. The maximum of c was tested with an initial condition of all off.

The h and k observed minima are much lower than the theoretical minima in experiment 8, because the capacitors are all equal, which means that there are no fast places or transitions that can turn on faster than their predecessors turn off.

The mutual inhibition never failed in experiment 9, because T3 was so much faster than the rest of the competing transitions.

In general, the theoretical parameters matched the observed parameters.

6 Conclusions

In this thesis, a method of designing asynchronous sequential circuits using threshold elements was presented. Using threshold elements is useful for making a sequential machine that is sensitive to the analog value of inputs. A digital circuit could be desgined to do analog arbitration, but it would need analog-to-digital converters and ALUs, and thus be inefficient.

However, threshold elements are inaccurate. To construct a many-input AND or OR gate using one threshold element would require exact weights. This thesis showed that one could easily mix Boolean logic with analog threshold elements.

The VLSI implementation of threshold logic presented in this thesis is cheap: only two transistors per input. The summing is accomplished by a wired OR. Thus, threshold elements are easy to build, and one could potentially put many threshold elements on a chip to build interesting systems. If one uses current summing in sub-threshold, however, threshold variations may cause the circuit to fail.

A simple generalization of a weighted sum of outputs was presented in this thesis, namely, a weighted sum of Boolean functions of the outputs, seen in figure 48. Perhaps one could use this new logic family to build interesting collective circuits.

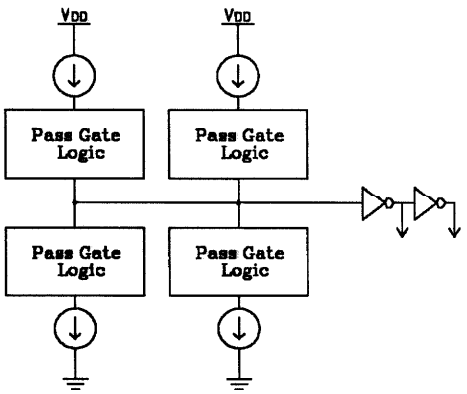

Fig. 48. New Logic Family

One can also compare the threshold circuits presented in this thesis with other collective circuits done with threshold elements. The circuits in this thesis used asymmetric connections between the elements. In previous work [Hopfield 82], only symmetric connections were truly controllable, with an energy function.

An analysis by forces was presented in this thesis. Forces can be thought of as a generalization of energy to asymmetric matrices. By constructing matrices with specified forces, asymmetry was shown to generate simple, controllable behaviours. The force analysis isn't restricted to sequential circuits. Any threshold circuit should be amenable to force analysis.

6.1 Future Work

There are several interesting ideas that grow out of this thesis. First, there is the direct application of these circuits to real-world problems. For example, an analog sequential circuit might be useful in speech recognition.

Another route to take is to design special purpose collective circuits using force analysis, analogous to the energy function analysis done in the travelling salesman problem [Hopfield 85]. Namely, if the desired behaviour of a circuit is an asynchronous transition between states, then force analysis should yield a design.

In this thesis, states that had only one element on were made temporarily stable. As can be seen in Appendix A, the temporarily stable states can have more than one element on, thus introducing redundancy into the circuit. Redundancy makes the circuit fault-tolerant: if any element fails, the circuit can still function. Thus, threshold elements could potentially be made fault-tolerant and the force analysis still applies.

Finally, one can possibly compose collective circuits using the techniques presented in this thesis. Consider the sequential machine of §5. The tokens travel through the Petri net, until a conflict is reached. The mutual inhibition of the conflict circuit fragment does the real computation of the circuit: analog arbitration. Instead of mutual inhibition, perhaps a more complicated collective circuit could be substituted. For example, the simple Petri net fragments could asynchronously activate associative memories, or travelling salesman circuits, which would then make decisions, and activate other collective circuits in the network. Perhaps these circuits are amenable to force analysis, also.

Bibliography

[Dennis 70] Dennis, Jack B., *Modular, Asynchronous Control Structures for a High Performance Processor,* Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, pp 55-80, 1970.

[Hebb 49] Hebb, D. O., *The Organization of Behavior,* Wiley, 1949.

[Hollaar 83] Hollaar, L., *Direct Implementation of Asynchronous Control Units*, IEEE Trans. on Computers, Vol 31, No. 12, pp 1133-1141.

[Holt 70] Holt, Anatol W. and F. Commoner, *Events and Conditions,* Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, pp 3-52, 1970.

[Hopfield 82] Hopfield, J.J., *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, Proc. Natl. Acad. Sci. USA, Vol 79, pp 2554-2558, 1982.

[Hopfield 84] Hopfield, J.J., *Neurons with Graded Response Have Collective Properties Like Those of Two-State Neurons*, Proc. Natl. Acad. Sci. USA, Vol 81, No. 10, pp 3088-3092.

[Hopfield 85] Hopfield, J.J., *"Neural" Computation of Decisions in Optimization Problems*, Caltech, preprint.

[McCulloch 43] McCulloch, W. S. and W. Pitts, *A logical calculus of the ideas imminent in nervous activity*, Bull. Math. Biophys., Vol 5, p 115.

[Mead 80] Mead, Carver and Lynn Conway, *Introduction to VLSI Systems,* Addison-Wesley, 1980.

[Millman 79] Millman, Jacob, *Microelectronics: Digital and Analog Circuits and Systems,* McGraw-Hill, 1979.

[Minsky 68] Minsky, M. and S. Papert, *Perceptrons,* MIT Press, 1968.

[Peterson 81] Peterson, James L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[Seitz 70] Seitz, Charles L., *Asychronous Machines Exhibiting Concurrency,* Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, pp 93-106, 1970.

Appendix A: The Relationship to Hopfield's Model

The idea of constraints on forces is more general than shown in §4. Namely, one does not need to just manipulate the forces on one element, one can manipulate the forces on an entire vector of elements. In the Petri net, the one-element representation is analogous to a one-hot code in asynchronous digital design[Hollaar 83], where each state is represented by one feedback path turning on. Using full state vectors is like using a more complicated coding scheme for states. This can be done to make the network redundant and fault-tolerant.

For example, if one wants a group of five elements, represented by

$$\vec{M}^0 = 1111100000 \tag{38}$$

to turn on the next five elements,

$$\vec{M}^1 = 0000011111 \tag{39}$$

one simply makes excitatory connections from the first group of five to the second group of five ($h_j(\vec{V}) = V_j$). This generalization of link-by-link wiring can be represented as an *outer product.* Namely, if one wants to give a force α to the elements on in $\vec{M}^1$ when the system is in state $\vec{M}^0$, one simply uses

$$T_{ij} = \beta M_i^1 h_j(\vec{M}^0) \tag{40}$$

To find the force owing to this matrix, one plugs equation (40) into the force equation (2), without the threshold.

$$\begin{aligned} F_i(\vec{M}^0) &= \sum_j \beta M_i^1 h_j(\vec{M}^0) h_j(\vec{M}^0) \\ &= \beta M_i^1 \sum_j h_j^2(\vec{M}^0) \end{aligned} \tag{41}$$

Thus, all the elements on in $\vec{M}^1$ get a force with strength $\beta \sum_j h_j^2(\vec{M}^0)$. Thus,

$$\beta = \frac{\alpha}{\sum_j h_j^2(\vec{M}^0)} \tag{42}$$

Because the force is linear in T, one can superimpose force components made by outer products by superimposing outer products in the matrix T. Thus, the idea of wiring up the forces separately still holds, even when the forces act on entire vectors.

With the introduction of multi-element representations, the idea of *state space* becomes useful. State space is the space of all possible output voltages, thus is continuous. Adding outer products to the matrix attributes meaning to particular points in state space. But, because of the continuity of the analog sum done by the elements, regions around these points also mean the same thing. Thus, using multi-bit representations is analogous to using error correcting codes to tolerate faults.

Thus, the connection to Hopfield's associative memory becomes clear [Hopfield 82]. To make an associative memory, one uses symmetric outer products to make attractors. Thus, to make state A stable, one uses

$$T_{ij} = A_i A_j,$$

which generates a force

$$F_i = A_i A_j V_j = A_i(\vec{A} \cdot \vec{V}).$$

Thus, the associative memory gets forced to a memory with a strength proportional to the distance the system is from the memory. Thus, the system usually proceeds to the nearest memory, since that has the strongest force.

Hopfield's associative memory is fault-tolerant because of multiple bits per memory. One can use multi-element state vectors with the AASM, also. Consider a circuit with m element per state. If any element is stuck on or off, or if any connection is broken, there will be a relative error of $1/m$ in some parameter. The VLSI parameters can withstand a $1/3$ error, so that using three elements per state will be fault tolerant. Namely, one element per place or transition can be destroyed, and the circuit will still work.

There are some problems with using the simple redundancy described above. First, the circuit requires nine times the number of synapses than the irredundant circuit. This may be too costly. Second, the circuit becomes much less fault-tolerant if there are large multiple forks or joins, since the AND gates break if any one of their inputs is stuck on or off. Third, the connections are still local, so that faults that are spatially correlated might destroy the circuit.

Appendix B: The Experimental Discrete Circuits

To test the theory in §3, a 4-stage ring oscillator was built. The circuit diagram of one element is presented in figure 49. Let this be element i. All the connections are translation invariant, so a connection to $i-1$ means a connection to the previous element in the oscillator.

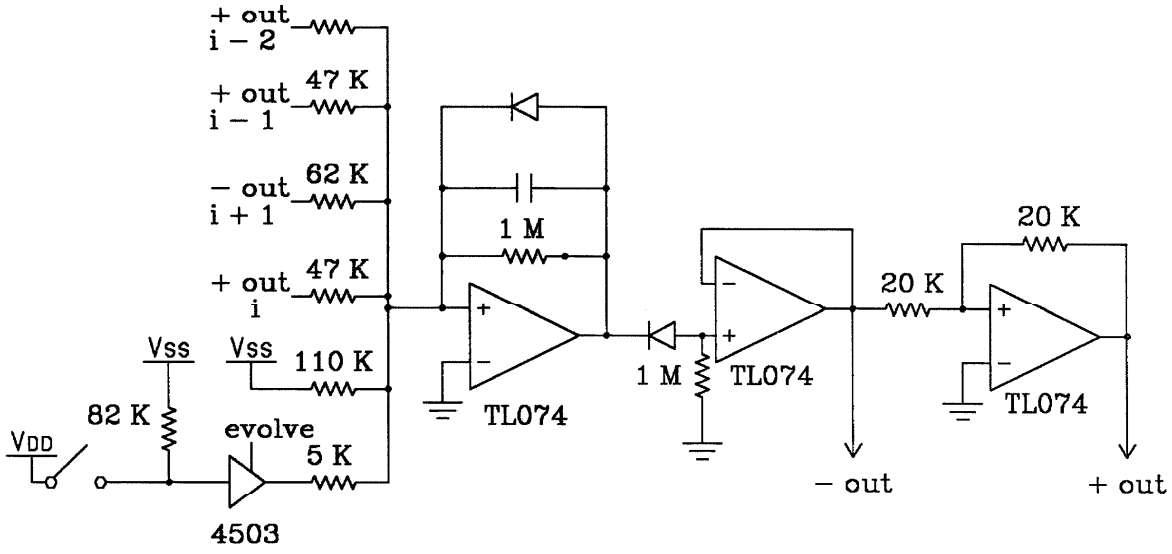

Fig. 49. One-fourth of the 4-Ring Oscillator

The circuit runs on a dual-rail power supply of -9V/+9V.

The resistances of the summing network shown in figure 49 correspond to the weights in (10). All the connections that have the same parameter are put in the same resistor pack, so that changing a resistor pack will globally change the parameter.

The starting state of the system is applied through the 5K resistors hooked up to the tri-state buffer. Then, the drivers are disabled, and the system evolves.

The experimental circuit for the AASM is similar to that of the four-ring oscillator. The connections were designed to implement the Petri net in figure 40.

Here, the conductances shown in figure 50 and figure 51 correspond to the weights in (24) and (31).

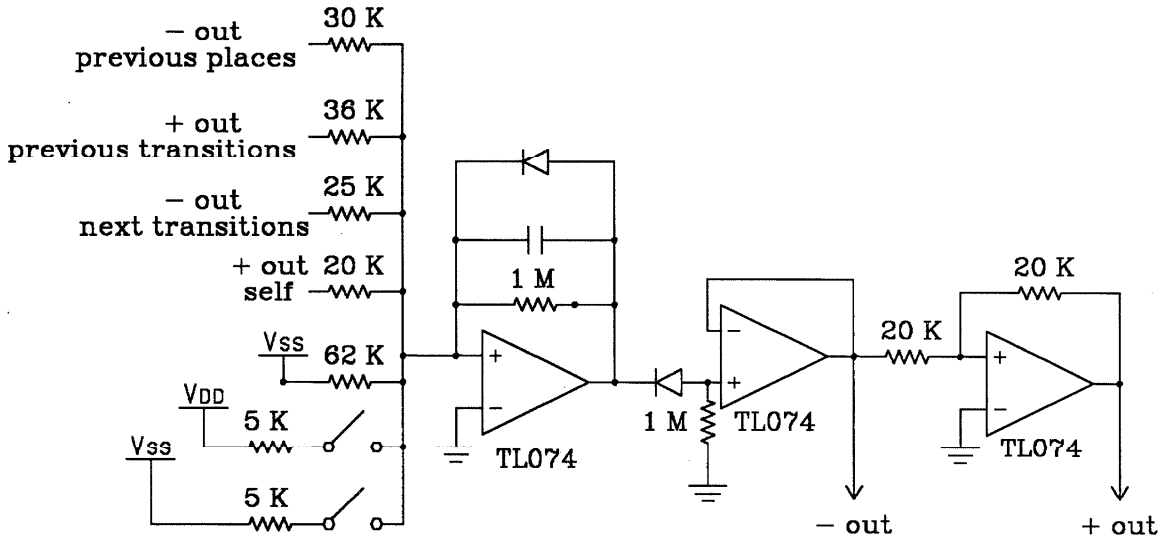

Fig. 50. Experimental Circuit for a Place in the AASM

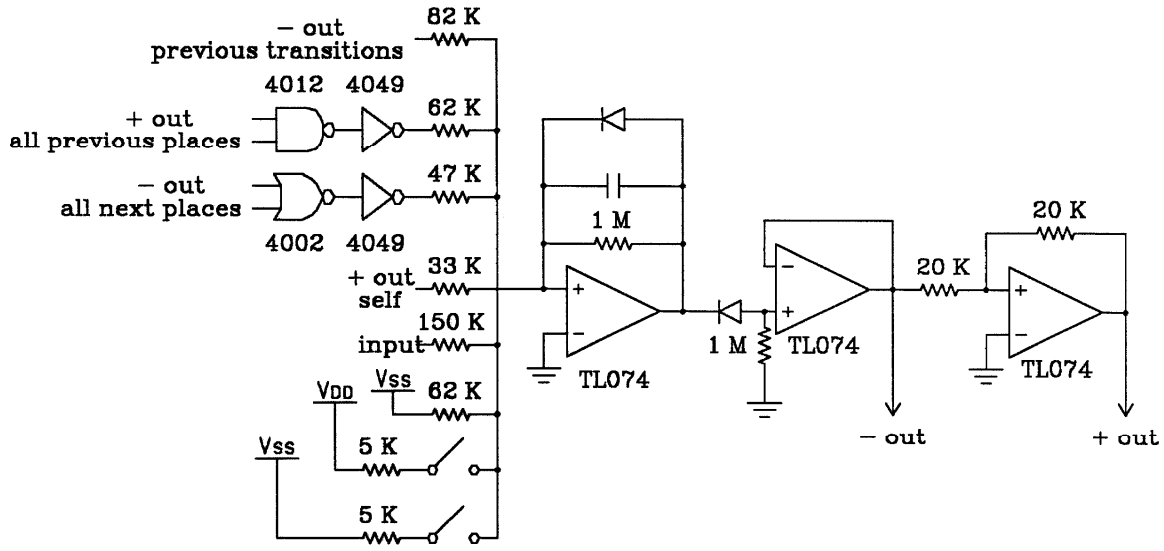

Fig. 51. Experimental Circuit for a Transition in the AASM

Again, a power supply of -9V/0/9V is used. A full 15V is not used to protect the CMOS logic. The output swing of the op-amp is -7.5V to +7.5V. Thus, the full output of an op-amp through a fixed resistance does not generate the same current as the output of the CMOS logic, or a power supply connected through the same resistance.

A simple remedy is applied to the mis-matched voltage problem. First, the CMOS logic (except for the tri-state drivers) is run on a power supply of 0V/+7.5V, to make the outputs consistent with the op-amps and to make the switching threshold for the CMOS

logic to be at 0.5. The extra power supply is generated by a voltage follower op-amp whose positive input is held at the proper voltage. Notice that all inputs to the CMOS must be between 0V and 7.5V. Thus, to get a inverted input to a CMOS NOR gate, one must invert the non-inverted output of the op-amp, instead of using the inverted output.

The CMOS logic whose inputs are hooked to positive outputs have a power supply of 0V/+7.5V. The CMOS whose inputs are hooked to negative outputs have a power supply of -7.5V/0V. As in the oscillator case, all inputs to this CMOS logic must be in the voltage rails, which requires extra inverters. The extra voltages are generated from two voltage followers.

The circuits shown for the AASM are really particular examples of an element. In the place circuit, one resistor per previous and next transitions must be used. In the transition circuit, if the transition is a multiple fork, then the inhibition from all output places must be fed through the NAND gate. Otherwise, a simple resistor will suffice. Similarly, if the transition is a multiple join, then excitation from all the input places must be fed into the NOR gate. For all of the other connections, one resistor per connection should be used.

Appendix C: VLSI Simulation

The VLSI circuit was simulated using Gear's method for stiff non-linear ordinary differential equations. The differential equation that approximates the behaviour of the VLSI threshold elements is

$$\dot{v}_i = \frac{1}{C_i} \sum_j f_j(v_i) T_{ij} g_j(\vec{v}), \tag{43}$$

where v_i = the voltage on the summing node of threshold element i, assumed to go between 0 and 1; T_{ij} = the connection strength from element j to element i, and C_i is the capacitance of the summing node of element i.

Here, $f_j(v_i)$ represents the saturation of the individual connections. Thus, f_j depends on the type of transistor used in the connection. Namely,

$$f_j(v_i) = \begin{cases} v_i/(v_i + 1/s), & \text{if } T_{ij} > 0; \\ (1 - v_i)/(1 - v_i + 1/s), & \text{if } T_{ij} < 0. \end{cases} \tag{44}$$

Here, s regulates the width of the ohmic region. The larger s is, the narrower the ohmic region. In the simulations,

$$s = 100 \tag{45},$$

which corresponds to an ohmic region of 50mV for a VDD of 5V.

The g_j function is a composition of two functions y_j and z_j:

$$g_j(v_j) = y_j(z_j(\vec{v})). \tag{46}$$

Here, y_j is the connection non-linearity and z_j is the inverter chain non-linearity combined with any AND or OR used. y_j depends on the type of transistor, again,

$$y_j(v) = \begin{cases} 0.98 + 0.1v, & \text{if } T_{ij} > 0 \text{ and } v > 0.2 \text{ (above threshold)}; \\ \exp(100v - 20), & \text{if } T_{ij} > 0 \text{ and } v < 0.2 \text{ (below threshold)}; \\ 1.08 - 0.1v, & \text{if } T_{ij} < 0 \text{ and } v < 0.8 \text{ (above threshold)}; \\ \exp(80 - 100v), & \text{if } T_{ij} < 0 \text{ and } v > 0.8 \text{ (below threshold)}. \end{cases} \tag{47}$$

The exponential behaviour simulates the exponential dependence of the current on v below threshold. But, there is a current limit imposed by the T_{ij}, which is reflected in the slow linear growth in the current when the connection is fully turned on.

In the simplest case, with no AND or OR, z_j is rather simple. Notice, in figure 30, there are three possible outputs of a threshold element: a fast inverted output, used for f and b type connections; a non-inverted output; and a slow inverted output, used for d and e type connections. z_j depends on which of these outputs are needed. Thus, define a new function z^*,

$$z_j(\vec{v}) = z^*(v_j) = \begin{cases} v_j, & \text{if fast inverted output;} \\ w(v_j), & \text{if non-inverted output;} \\ w(w(v_j)), & \text{if slow inverted output.} \end{cases} \tag{48}$$

Here, $w(v)$ is the inverter transfer response, approximated by

$$w(v) = (0.5 - v)/(|2v - 1| + 1/\alpha) + 0.5, \tag{49}$$

where α is the gain of the inverter, assumed to be 10 in the simulations.

A connection with an OR is simulated with taking either the minimum or the maximum of the set of z_j^* being ORed. Namely,

$$z_j(\vec{v}) = \begin{cases} \min(z^*(v_j)), & \text{if inhibitory OR;} \\ \max(z^*(v_j)), & \text{if excitatory OR.} \end{cases} \tag{50}$$

Simlarly, a connection with an AND is simulated by taking a minimum or a maximum of the set of z_j^* being ANDed. Namely,

$$z_j(\vec{V}) = \begin{cases} \max(z^*(v_j)), & \text{if inhibitory AND;} \\ \min(z^*(v_j)), & \text{if excitatory AND.} \end{cases} \tag{51}$$

There are still some special cases to be taken care of. For a excitatory connection that is always on, use $z_j(\vec{v}) = z^*(1.0)$. For an inhibitory connection that is always on, e.g., a threshold, use $z_j(\vec{v}) = z^*(0.0)$. For an input, just use a current

$$i_{\text{input}} = T_{ij}X,$$

where X is the input, which is restricted to [0,1].