LANGUAGE, COMPILER, AND OPERATING SYSTEM FOR THE CNN SUPERCOMPUTER

by

T. Roska, L. O. Chua, and Á. Zarándy

Memorandum No. UCB/ERL M93/34

30 April 1993

LANGUAGE, COMPILER, AND OPERATING SYSTEM FOR THE CNN SUPERCOMPUTER

by

T. Roska, L. O. Chua, and Á. Zarándy

Memorandum No. UCB/ERL M93/34

30 April 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

LANGUAGE, COMPILER, AND OPERATING SYSTEM FOR THE CNN SUPERCOMPUTER

by

T. Roska, L. O. Chua, and Á. Zarándy

Memorandum No. UCB/ERL M93/34

30 April 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Language, compiler, and operating system for the CNN Supercomputer

T. Roska, L.O. Chua, Á. Zarándy
Version 1.1

April 30, 1993

1 Introduction

The CNN Universal Machine and Supercomputer [10] is the first stored program analog computing array architecture. Its various implementations, in parts, show that for this new kind of analogic computing we need all the esential programming tools which digital computers have, though in different form.

Namely, we hnedd an analogic algorithm (e.g. in a form of a flow diagram), a high level language (e.g. the "Analogic CNN Language (ACL)"), a compiler, an operating system, and a generated machine code. Although they are quite simple in our present phase of making these machines, their existence suggest a similar development to the one we had in the 1970's for microprocessors.

The main difference is accounted for by the presence of analog array dynamics as the key intruction/operation in the analogic CNN algorithms [1-8]. In what follows, we outline the main ideas and a simple implementation. Inclusively, we suggest a framework for the implementation of a development system for CNN universal chips.

2 The analogic CNN algorithm

The analogic CNN algorithm consists of sequential and parallel algorithmic steps. These steps, analog and logic, are implemented by using the following **analogic algorithmic elements** (E).

E1: global input/output operations
E2: global CNN operations with specified cloning templates
E3: local storage
E4: local exchange of information between local memory units
E5: logical computations combining locally stored values
E6: conditional branching

The analogic CNN algorithm can be represented by a flow diagram containing the algorithmic elements E1-E5. This list contains the simplest set of analogic algorithmic elements. The next very simple example shows the basic steps of operations.

Example

Given a black-and-white image. Consider the black pixels as +1 values, and the white ones as -1 values.

Problem

Detect the horizontal zero-crossings. (In other words: extract the vertical edges.)

Solution

Let us solve the problem with the following analogic CNN algorithm:

- Detect black pixels with white direct horizontal right neighbor, and store the result.
- Detect the white pixels with black direct horizontal right neighbor, and store this result as well.
- Apply a pixel by pixel logic "OR"function on the previous two result.

The flow diagram can be seen in Figure 1.

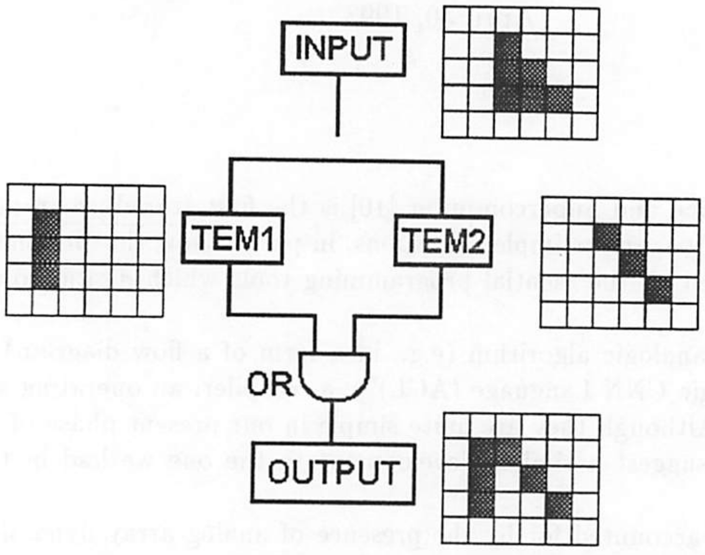


Figure 1. The flow diagram of the analogic CNN algorithm. The operation is demonstrated by a simple example.

The templates used in the CNN algorithm are as follows:

TEM1 (Black_to_White)

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}, \quad I = -1.5,$$

TEM2 (White_to_Black)

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 2 \\ 0 & 0 & 0 \end{bmatrix}, \quad I = -1.5,$$

The operation of the analogic CNN algorithm is also shown in Figure 1. The simple input image is selected for the sake of better illustration.

3 The Global Analogic Program Unit (GAPU) executes the analogic CNN algorithm

The analogic CNN algorithm is performed by the CNN array under the control of the Global Analogic Program Unit (GAPU). We have to load the registers (APR, LPR, SCR) and the GACU (Global Analogic Control Unit). In our example, there are represented as follows:

	A	B	I	
TEM1	(0 0 0 0 2 0 0 0 0)	(0 0 0 0 0 -2 0 0 0)	-1.5	APR(1)
TEM2	(0 0 0 0 2 0 0 0 0)	(0 0 0 0 -2 2 0 0 0)	-1.5	APR(2)

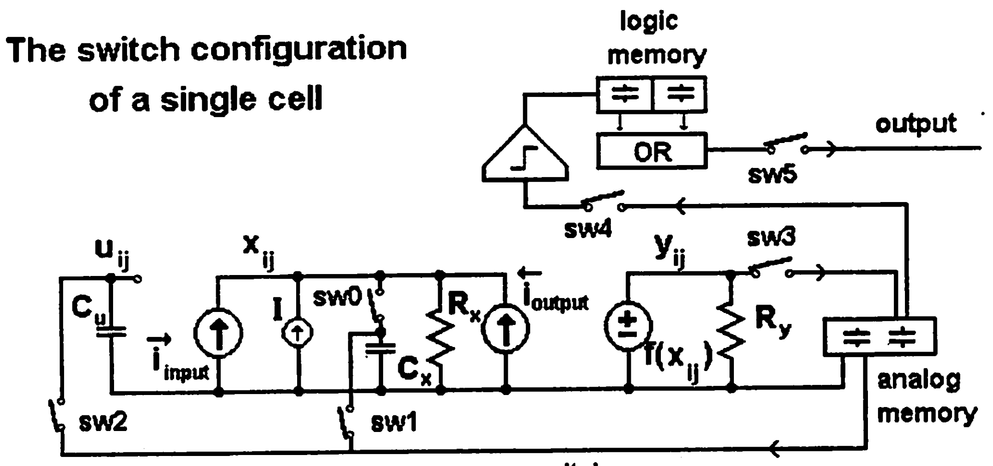
APR

The logic program register contains a single element, whose content is a 2-input OR operation. The first two value of every triple are the two operands, the third is the logic result.

1 1 1 — 1 0 1 — 0 1 1 — 0 0 0	LPR(1)

LLU1 (OR)

The cell circiut diagram for this simple example is shown below. The only analog memory (denoted here by LAM) is the original LAM4 [10].



	code \ switch	0	1	2	3	4	5
load input & initial state from LAM(1)	S0	off	on	on	off	off	off
start transient	S1	on	off	off	off	off	off
store the result in LAM(2)	S2	on	off	off	on	off	off
store LAM(2) in LLM	S3	off	off	off	off	on	off
calculate the logic operation	S4	off	off	off	off	off	on

Figure 2. The switch configuration of a CNN cell in our example.

The switch configurations coded and stored in a SCR specify the following actions:

S0: load input and initial state from LAM(1). i.e. the first element in the memory LAM
S1: start transient
S2: store analog output in LAM(2)
S3: send the analog output stored in LAM(1) to LLM(1) after shifting the LLM register by one step right (LAOU is now just a single wire)
S4: activate the local logic unit

The specific switch configuration used in our simple algorithm can be seen in Figure 2. The SCR is introduced to minimize the number of switch control wires. If we use three wires to control the local communication and control unit (LCCU), the SCR stores the codes of the different states, for example, as follows:

	$S0$	$S1$	$S2$	$S3$	$S4$	$S5$
*wire*1	0	1	0	1	0	1
*wire*2	0	0	1	1	0	0
*wire*3	0	0	0	0	1	1

The LCCU decodes the 3bit codes from among the 6 states of switch configurations providing the cell functions described above.

```
begin;
reset;
load(TEM1);
load(TEM2);
load(PIC);
sel(S0);       load input & initial state from LAM(1)
sel(TEM1);     tune TEM1
sel(S1);       start the analog transient
sel(S2);       store the result in LAM(2)
sel(S3);       store LAM(2) in LLM
sel(S0);       load input & initial state from LAM(1)
sel(TEM2);     tune TEM2
sel(S1);       start analog transient
sel(S2);       store the result in LAM(2)
sel(S3);       store LAM(2) in LLM
sel(OR);       tune   OR
sel(S4);       calculate the logic operation
save(PIC);
end:
```

Figure 3. The intermediate machine code for the GACU. The first five and the last two instructions (the uncommented) are external I/O operations, the others are internal executable instructions.

Now, let us define the macro machine code of the program, namely, the set of instructions for the global analogic control unit (GACU) and the input-output macros. We have already defined the contents of the registers APR, LPR, SCR. By selecting only a single item within a register, all CNN cells will be controlled in the same way. By selecting for example a template SEL(TEMi), the template elements (transconductans in the case of silicon implementation) of every cell will be set to these given values. By selecting only a local logic unit function, SEL(LLU), the given truth table (or the PLA content) will be loaded into all cells. By selecting only a switch configuration, SEL(Si), the appropriate switch position will be set in all cells.

Keeping all of this in mind, an intermediate control code of our CNN analogic program for the GACU can be seen in Figure 3. The Appendix contains the detailed explanation of these steps.

This program contains some complex operations, therefore it is an intermediate control code. For example, the output is a sequence of several elementery steps. Therefore we have to represent these by the elementary steps in the GACU. The final control code of the GACU contains the sequence of these elementery steps. Of course, the GACU contains control logic hardware which controls all the registers as well as the timing clocks. At least two clocks are needed: one for the logic operations and one for the analog operations (for the local and propagating transients as well as for the input/output row-wise control).

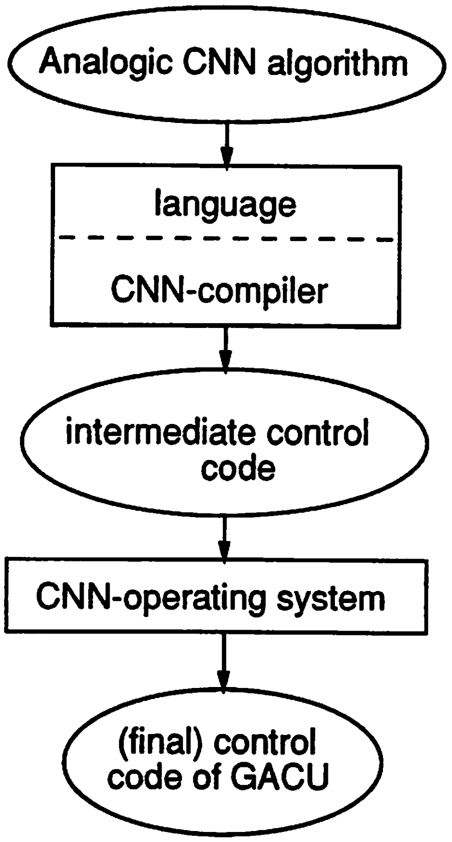


Figure 4. The steps from a conceptual analogic CNN algorithm to an executable control code.

4 From the analogic CNN algorithm to the operating GAPU

Figure 4 shows the way how we can generate the codes of the GACU from the conceptual description of an analogic CNN algorithm (or flow diagram).

4.1 The CNN language

Any algorithm, to be executed on a given hardware, should be described by an appropriate language for a compiler. Such a language ("ACL") has been developed for the analogic CNN algorithm. The ACL is a quite simple language constructed by the following language elements:
- template and image declarations,
- analogic CNN operations,
- other commands (e.g. save output image in a background storage, display output, end of program, etc).

The templates and the images are stored in standard file formats. Intermediate (temporal) images are stored in the onchip local analog/logic memory (LAM/LLM). The syntax of the analog CNN operation is the following:
$OUTPUT = sub(TEMPLATE, INPUT, INITIAL\_STATE[, SPACE\_VAR\_THRESHOLD])$.
If the initial state and the imput image are the same a "*" symbol can be used instead of repeating the name. If the analog transient is not effected by one of the picture variables, the "don't care" signal ("-") can be used. The syntax of the local logic operation is as follows:
$RESULT = sub(LOGICAL\_OPERATION, OPERAND1, OPERAND2)$.
Let us describe the previous analogic example in ACL program language.

```
& input =" example.img";   /* input image declaration */
& TEM1 =" bl2wh.tem";   /* template declaration */
& TEM2 =" wh2bl.tem";

left_edges = sub(TEM1, input, *);   /* analog CNN operation */
right_edges = sub(TEM2, input, *);
edges = sub(OR, left_edges, right_edges);   /* logic operation */

 * so;   /* save output */
 * end;   /* end of program */
```

A compiler can generate intermediate codes for different hardwares. For example the DUALCOMP compiler [11] generates intermediate code for a CNN hardware accelerator board[11]. Another code generator is under development for the CNN Univeral Chip.

4.2 The CNN operating system

The intermediate control code generated by a CNN compiler is generally not yet executable because there are some frequently repeated operations which should be divided into elementary steps (exacutable by the GACU). In our example, the executable machine code described in Figure 3 contains an input instruction LOAD(PIC). To load an input image, except we have on-chip photoreceptors, we need a sequence of row-wise loading of analog values from an outside source (e.g. a camera or an analog RAM). Hence, LOAD(PIC) will be represented by a sequence of elementary machine code instructions of the GACU controlling the row-wise input of an image matrix. In addition to generating the executable code, the operating system maintains utilities, as in case of digital microprocessor.

4.3 Adaptation and flexibility in programming

The CNN Universal Chip can perform two types of adaptations:

- cell by cell local adaptation, (e.g. the previously calculated local intensity average controls the DC threshold (I) level)
- global programmed adaptation.

In the latter case, the GACU may have a global memory and arithmetic logic unit (ALU). This global memory may contains the values of predefined cells (e.g. board cells to avoid side effects).
In physical implementation the GACU may be realized as a simple microprocessor (e.g. 8080).

5 The CNN environment

The CNN Universal Chip, like the digital microprocessor, is not a stand alone unit. It demands a digital computer as a user interface, analog and digital RAMs for data and program storage, and sensor arrays as input sources. Using special sensor arrays, the CNN becomes capable of solving various 2D or 3D problems. The CNN environment can be seen in Figure 5.

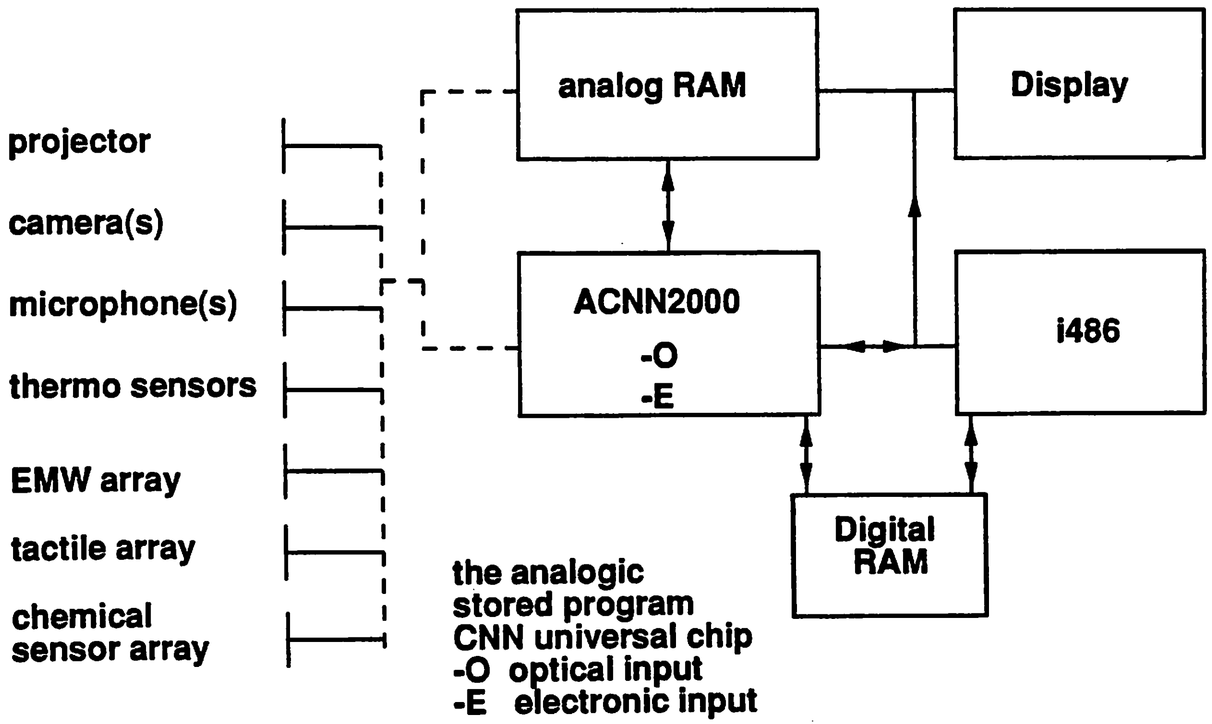


Figure 5. The CNN environment.

5.1 The CNN development system

Like all digital microprocessors, the analogic microprocessor, the CNN Universal Chip, needs a development system. A development system has many functions. Just to list a few:
- design of analogic algorithms,
- debugging these algorithms,
- compiling the algorithms and test machine code (for some well defined hardware),
- developing new compilers,
- testing working hardware prototypes.

The presently available "CNN Workstation, Version 5.1" (Figure 6.) [11] already provides the first four functions. Having machine code for a new CNN hardware, especially for a new CNN Universal Chip, additional hardware/software elements must be developed according to the specification of the existing system.

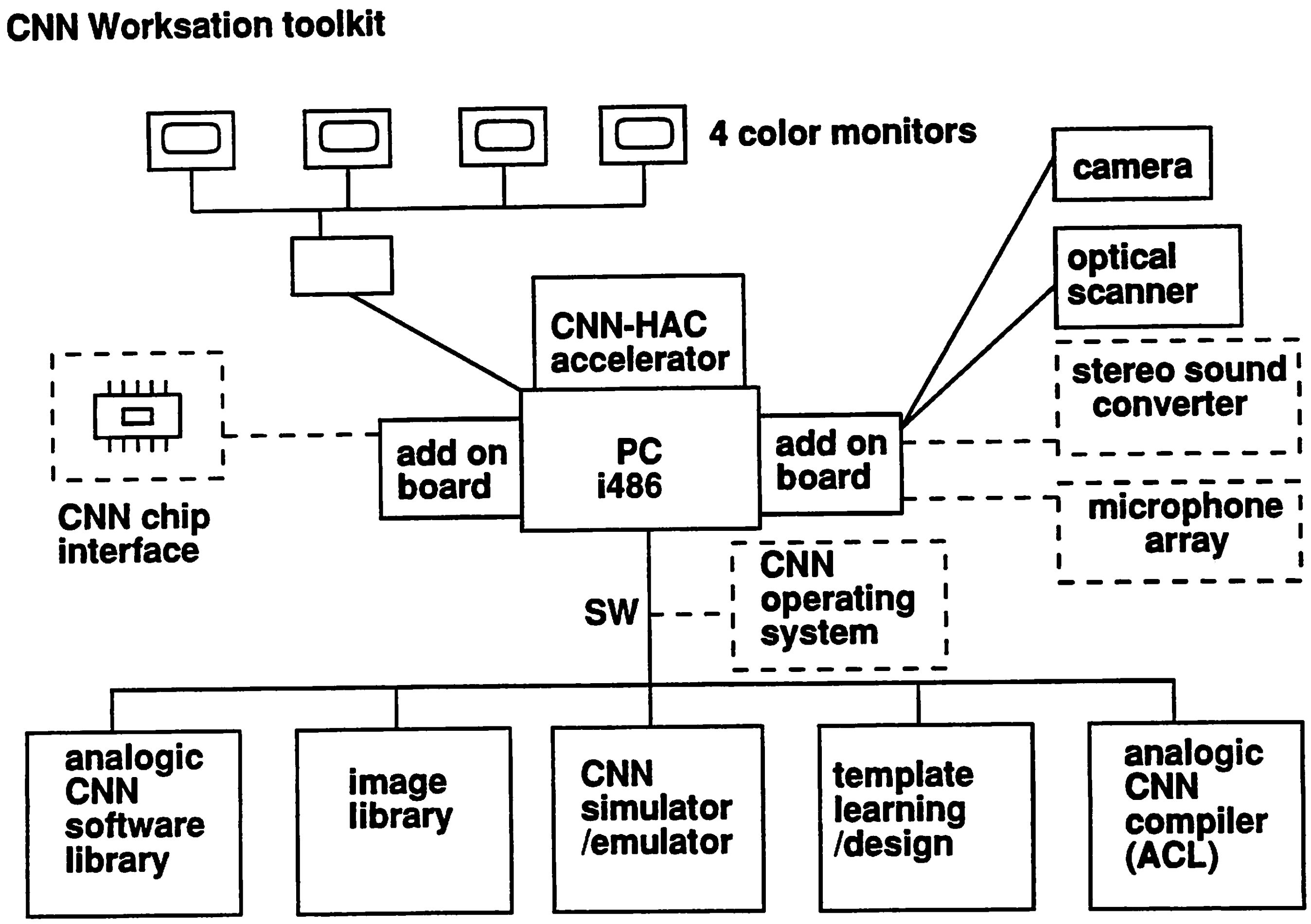


Figure 6. The CNN Workstation as a CNN microprocessor developing system.

Using our CNN Workstation, as a prototype we have summarized the framework of analogic CNN algorithm development in Figure 7.

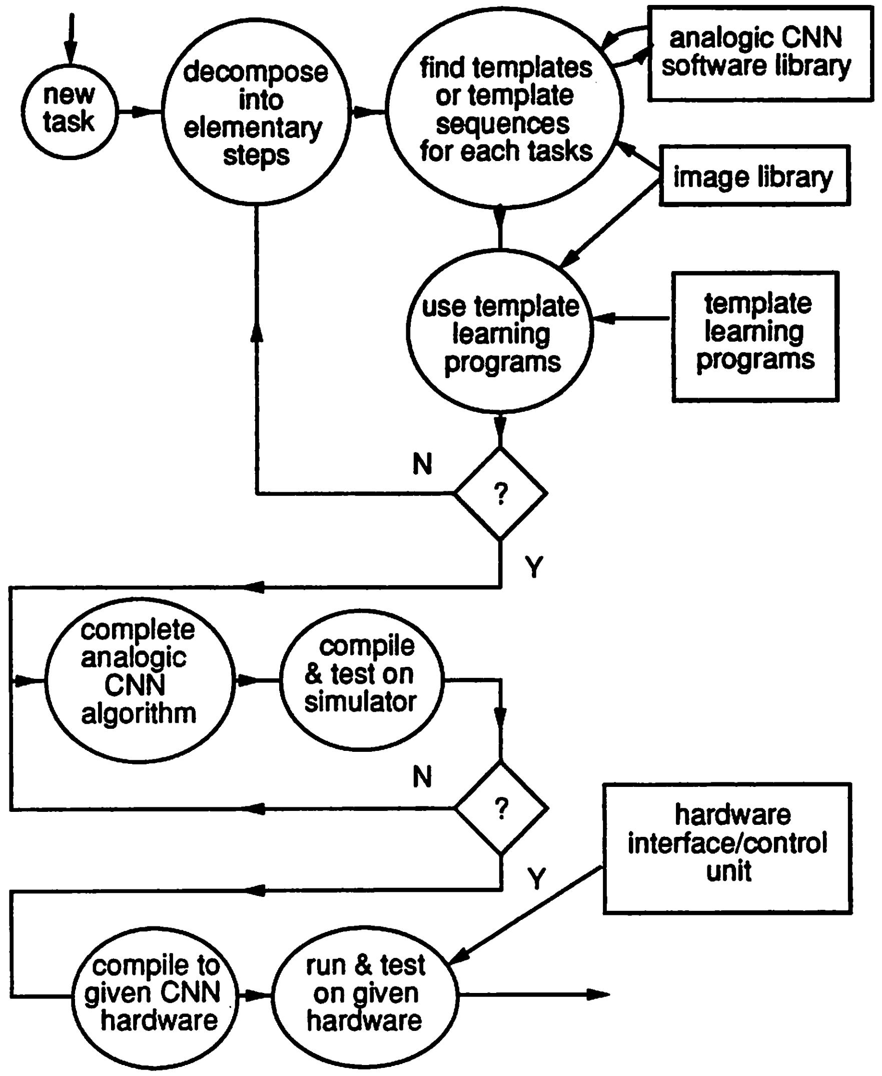

new task
decompose into elementary steps
find templates or template sequences for each tasks
analogic CNN software library
image library
use template learning programs
template learning programs
N
?
Y
complete analogic CNN algorithm
compile & test on simulator
N
?
Y
hardware interface/control unit
compile to given CNN hardware
run & test on given hardware

Figure 7. The analogic CNN algorithm developing system.

6 Conclusion

In the present phase of development of CNN Universal Chips (essentially all parts of a CNN Universal Chip is operating somewhere), it is clear that we will soon need all the tools for stored-program analogic computing that were needed in the time when the first microprocessors were developed. Some of these tools are available now, as we have outlined in this report.

7 Acknoledgements

The support of the grant INT 90-01336 of the National Science Foundation (incooperation with the Hungarian Academy of Sciences) is acknoledged. Thanks are due to S. Zöld and A. Radványi

for usefull advices and discussions.

8 References

[1] L.O.Chua and L.Yang, "Cellular neural networks:Theory", IEEE Transactions on Circuits and Systems, Vol.35, pp.1257-1272, 1988.

[2] L.O.Chua and L.Yang, "Cellular neural networks: Applications", IEEE Transactions on Circuits and Systems, Vol.35., pp.1273-1290.

[3] T.Roska and L.O.Chua, "Cellular Neural Networks with Nonlinear and Delay-type Template Elements" Proc. of IEEE Int. Workshop on Cellular Neural Networks and their Applications CNNA-90, pp.1225, 1990. extended version in Int. J. Circiut Theory and Applications, Vol.20.pp469-481, 1992.

[4] L.O.Chua, T.Roska, P.L.Venetianer and .Zarándy, "Some Novel Capabilities of CNN: Game of Life and Examples of Multipath Algorithms", Report DNS-3-1992, Dual and Neural Computing Systems Res.Lab., Computer and Automation Institute of the Hungarian Academy of Sciences, Budapest, 1992

[5] J.A.Nosek, G.Seiler, T.Roska, L.O.Chua, "Cellular neural networks: Theory and circuit design", Report TUM-LNSTR-90-7, Technische Universitat Munchen Dec. 1990. also Int. J. Circuit Theory and Applications, Vol.20.pp533-553, 1992.

[6] H.Harrer and J.A.Nosek, "Discrete time Cellular neural networks: Architecture Applications and realization", Report TUM-LNSTR-90-12, Technische Universitat Munchen Nov. 1990. also Int. J. Circuit Theory and Applications, Vol.20.pp453-467, 1992.

[7] J. Henseller and P.J Braspenning, "Membrain a Cellular neural network model based on vibrating membrane," Int. J. Circuit Theory and Applications, Vol.20.pp483-496., 1992.

[8] A. Radványi, K. Halonen, T. Roska,"The CNNL simmulator and some time varying template", Report DNS-9-1991, Dual and Neural Computing Systems Res.Lab., Computer and Automation Institute of the Hungarian Academy of Sciences, Budapest, 1991

[9] L.O.Chua, T.Roska," The CNN Paradigm" IEEE Transactions on Circuits and Systems Series I, March 1993.

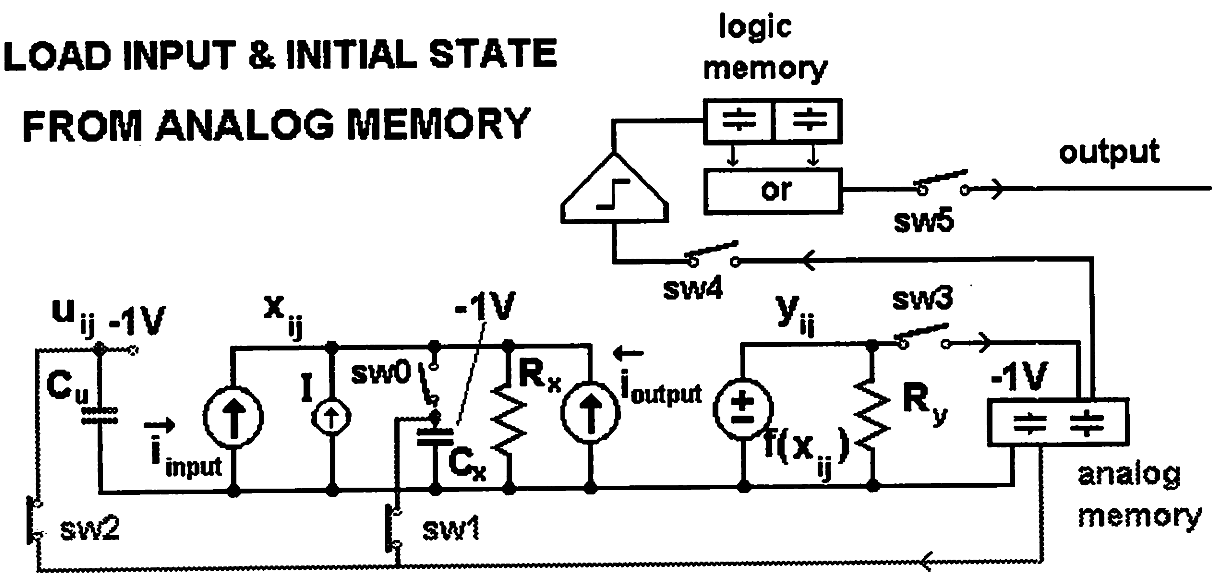
[10] T. Roska, L.O. Chua,"The CNN Universal Machine: An Analogic Array Computer" IEEE Transactions on Circuits and Systems, Series II March 1993

[11] "The CNN Workstation, Version 5.1, Users guide", MTA-SzTAKI Bp.1992.

9 Appendix

In the following figure series, the calculation process of our example for one single cell can be seen. For the sake of simplicity, there is only one clock shown in the figures. The analog transients settle down in 20 ns and the execution of the other instructions take 10 ns. In every figure, the currently active parts of the circuit are denoted by grey lines. The figure series does not contain individual figures for the template and logical operation tuning steps, but the clock indicates the time they elapsed.

LOAD INPUT & INITIAL STATE FROM ANALOG MEMORY

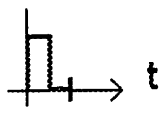


code	sw0	1	2	3	4	5
S0	off	on	on	off	off	off

SEL(S0)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)

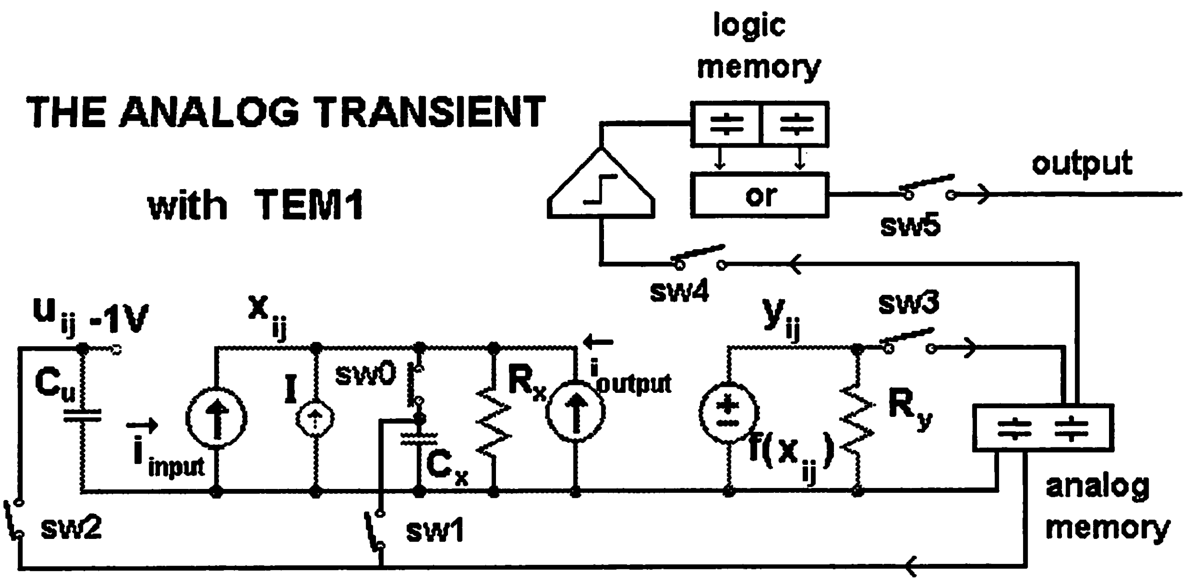
clock



10[ns]

THE ANALOG TRANSIENT

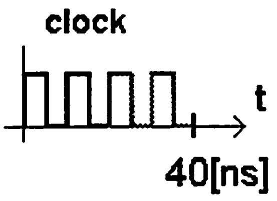
with TEM1

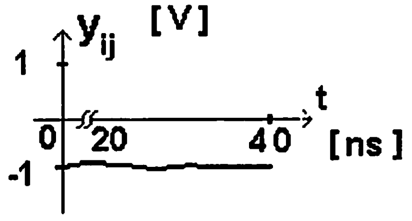


code	sw0	1	2	3	4	5
S1	on	off	off	off	off	off

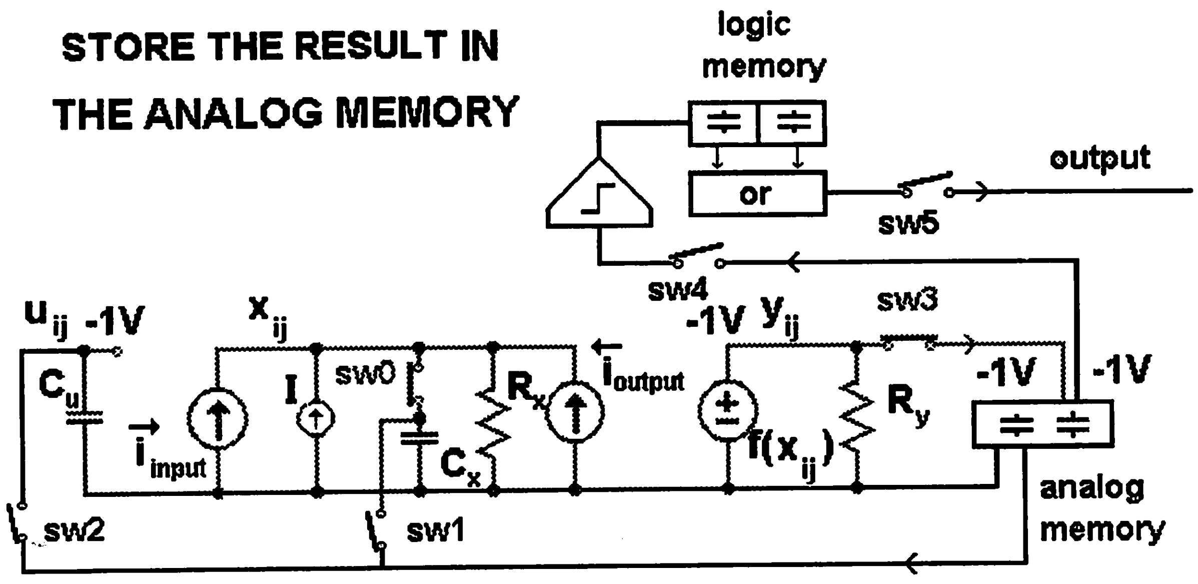
SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)





STORE THE RESULT IN THE ANALOG MEMORY

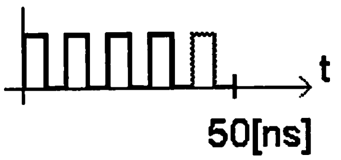


code	sw0	1	2	3	4	5
S2	on	off	off	on	off	off

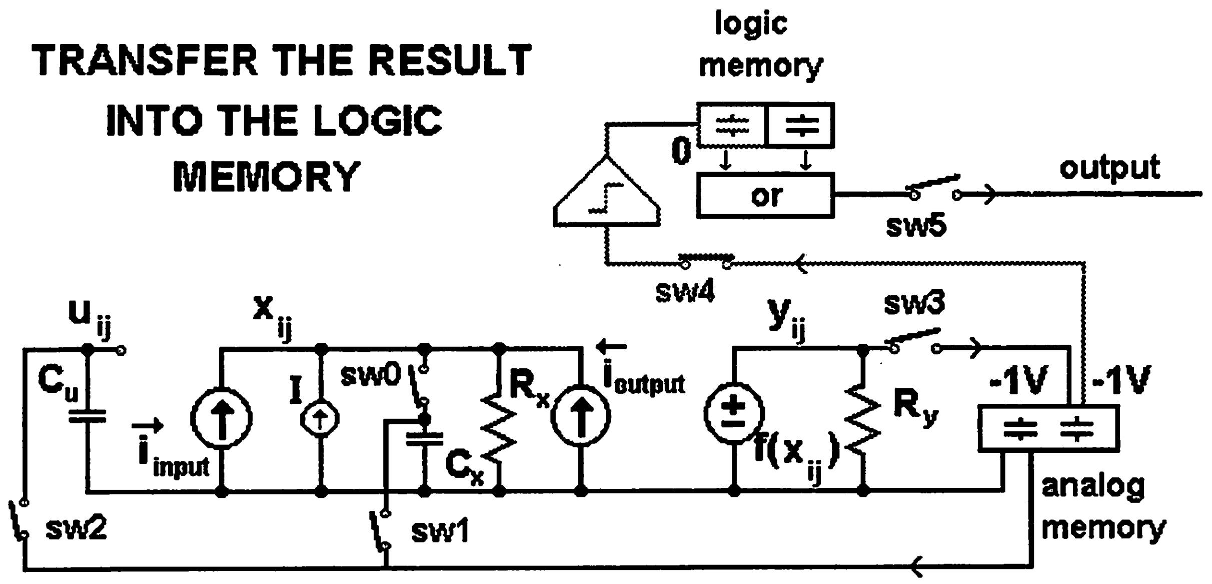
SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)

clock



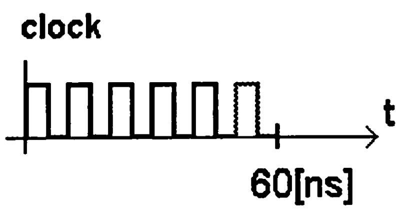
TRANSFER THE RESULT INTO THE LOGIC MEMORY



code	sw0	1	2	3	4	5
S3	off	off	off	off	on	off

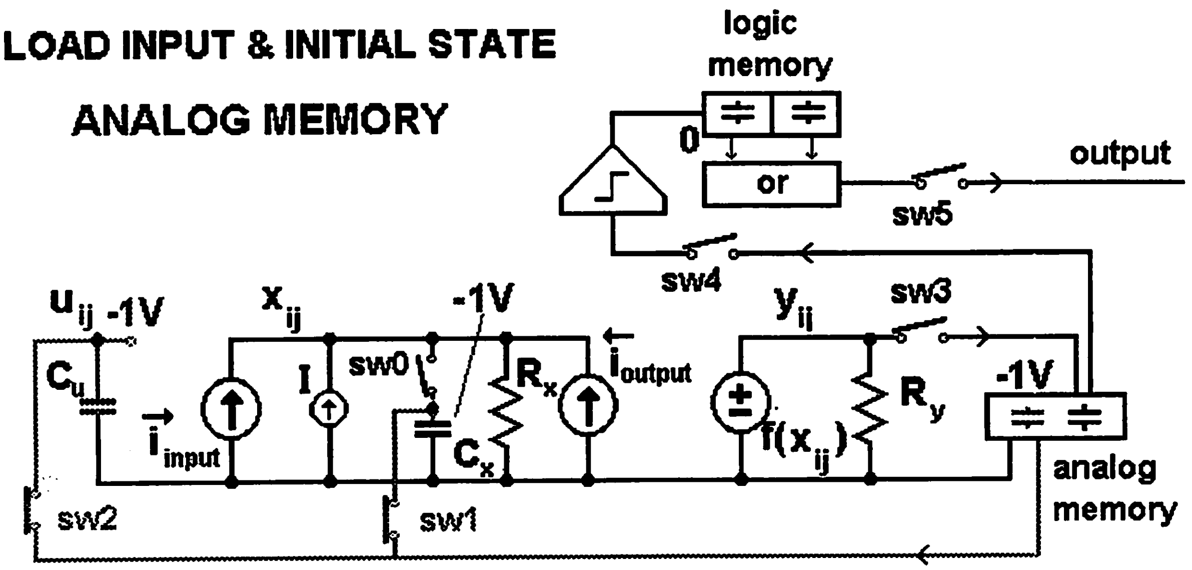
SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)



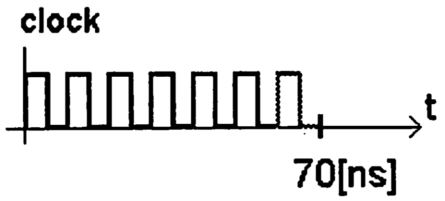
LOAD INPUT & INITIAL STATE

ANALOG MEMORY



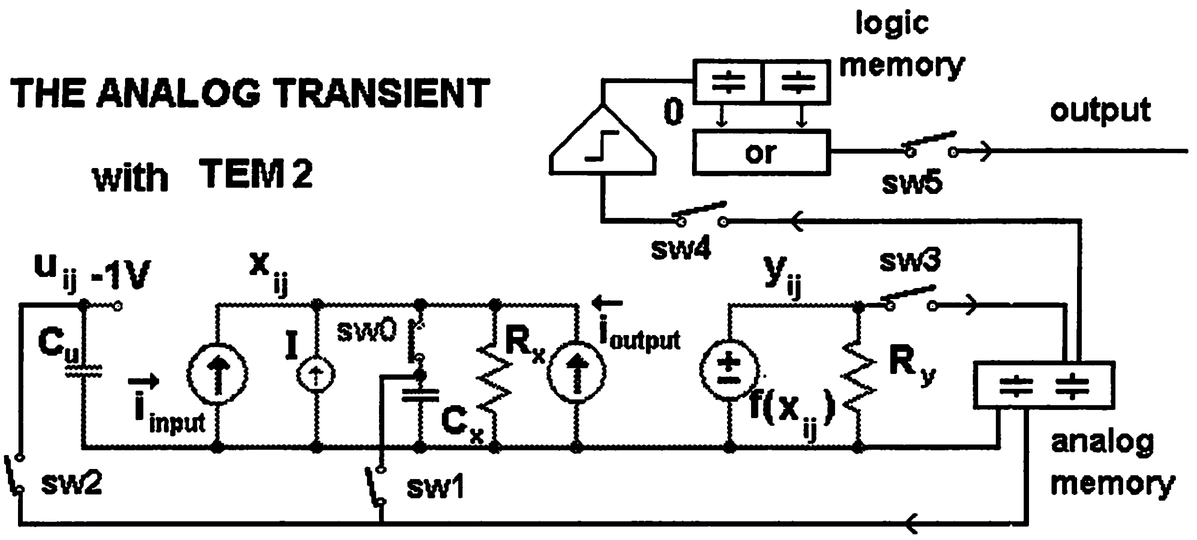
code	sw0	1	2	3	4	5
S0	off	on	on	off	off	off

SEL(S0) SEL(S0)
SEL(TEM1) SEL(TEM2)
SEL(S1) SEL(S1)
SEL(S2) SEL(S2)
SEL(S3) SEL(S3)
SEL(OR)
SEL(S4)



THE ANALOG TRANSIENT

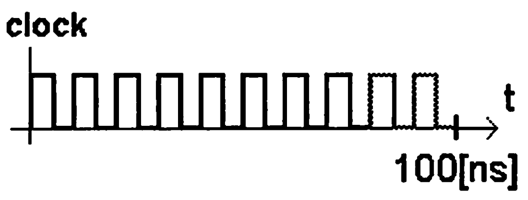
with TEM 2

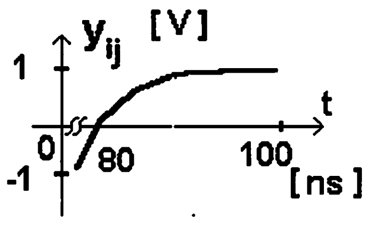


code	sw0	1	2	3	4	5
S1	on	off	off	off	off	off

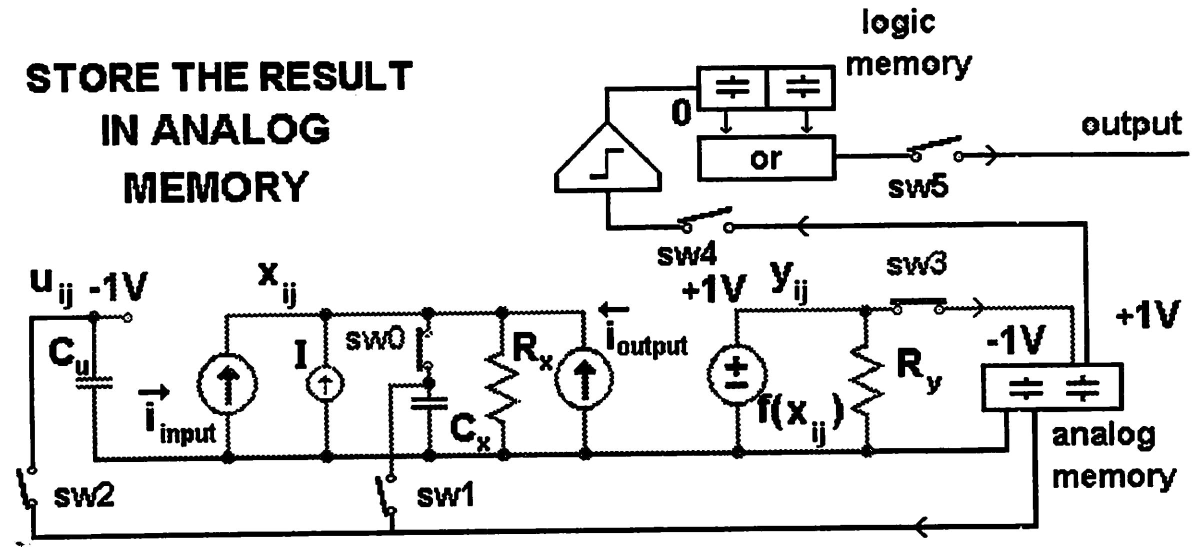
SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)





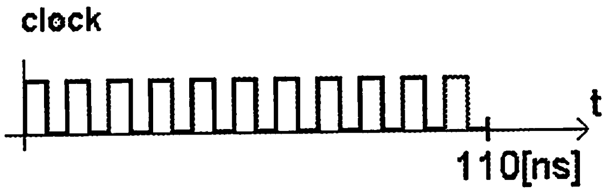
STORE THE RESULT IN ANALOG MEMORY



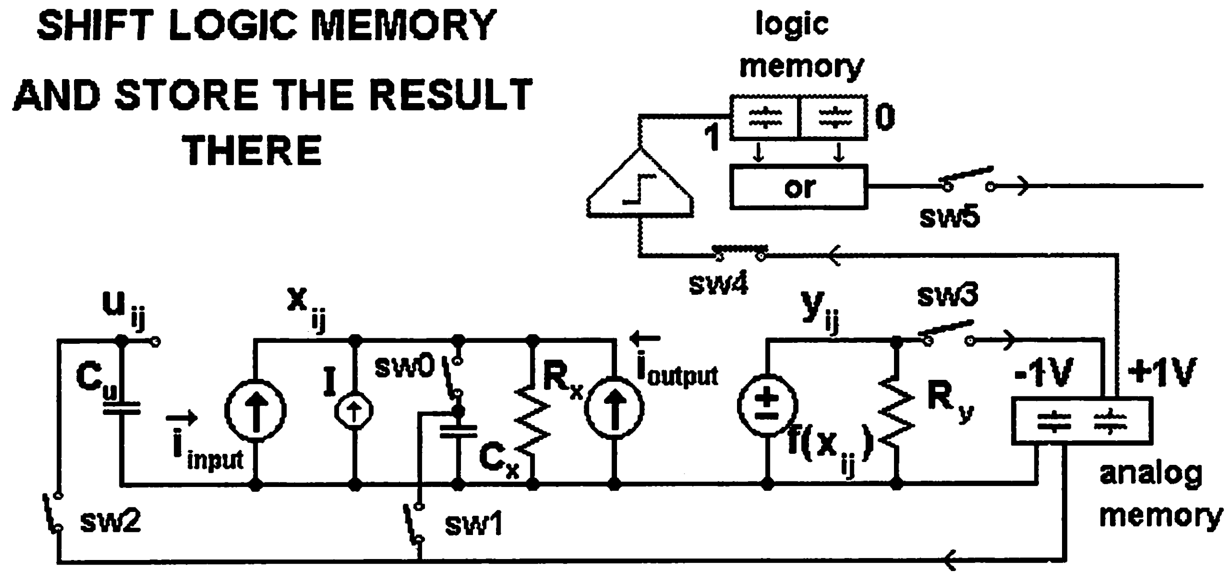
code	sw0	1	2	3	4	5
S2	on	off	off	on	off	off

SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)



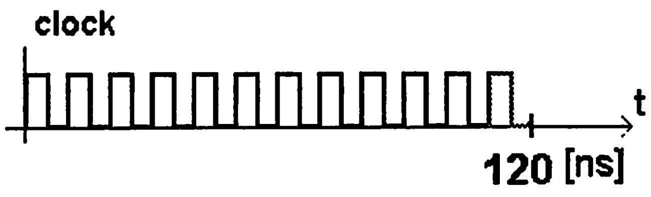
SHIFT LOGIC MEMORY AND STORE THE RESULT THERE



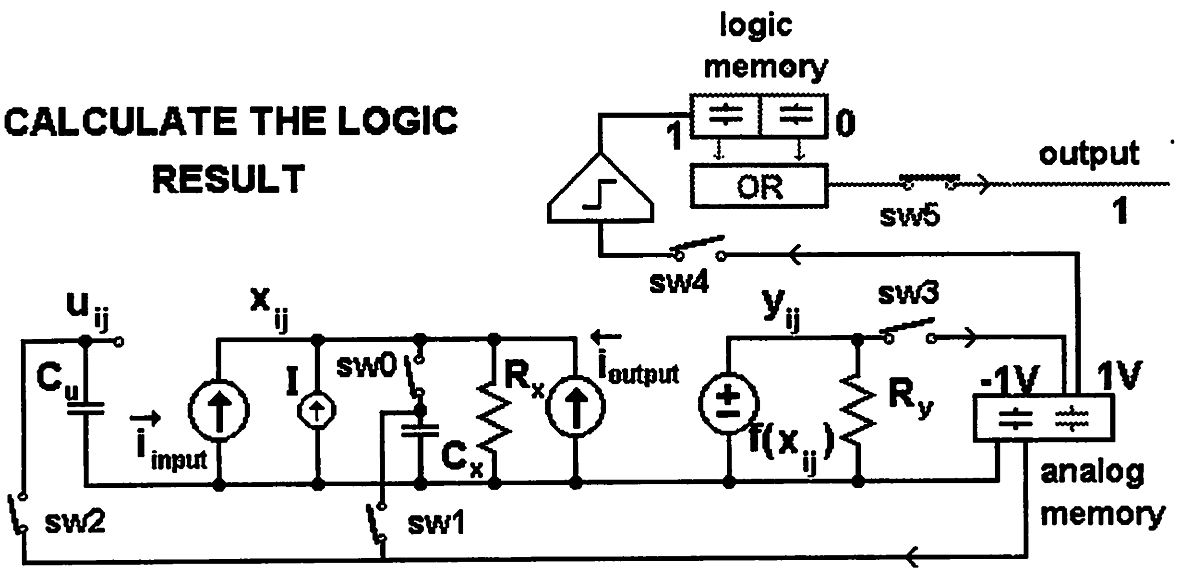
code	sw0	1	2	3	4	5
S3	off	off	off	off	on	off

SEL(S0)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)



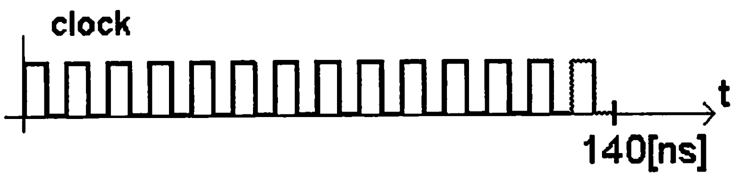
CALCULATE THE LOGIC RESULT



code	sw0	1	2	3	4	5
S4	off	off	off	off	off	on

SEL(SO)	SEL(S0)
SEL(TEM1)	SEL(TEM2)
SEL(S1)	SEL(S1)
SEL(S2)	SEL(S2)
SEL(S3)	SEL(S3)

SEL(OR)
SEL(S4)



clock
t
140[ns]