Chapter 9

The CNN Universal Machine (CNN-UM)

In Chapter 7, we have shown a couple of generic examples which can be solved by a sequence of CNN templates. The hardwired CNN implementations using different CNN components or different templates is , however, totally impractical. In this chapter we show the architecture of the first spatio-temporal analogic array computer, the CNN Universal Machine (CNN-UM).

In the examples mentioned above, and in many other examples including physiologically faithful models of various parts of the nervous system, especially vision, the following 2 completely different types of operations are used to solve a complex task:

- continuous-time, continuos valued spatiotemporal nonlinear array dynamics(2D and 3D arrays).
- local and global logic

Hence, analog (continuous) and logic operations are mixed and *embedded* in the array computer. Therefore we call this type of array computing: *analogic*.

The CNN-UM architecture, shown below

- contains a *minimum number* of component types
- provides *stored programmable* spatiaotemporal array computing, and
- is universal in two senses:

* as *spatial logic*, it is equivalent to a Turing Machine and as a local logic it may implement any *local Boolean function*.

* as a *nonlinear dynamic operator*, it can realize any local operator of fading memory[1], i.e., practically all reasonable operators. Indeed, the CNNUM is a common computational paradigm for as diverse fields of spatiotemporal computing as, for example, retinal models, reaction diffusion equations, mathematical morphology, etc.

Remarks:

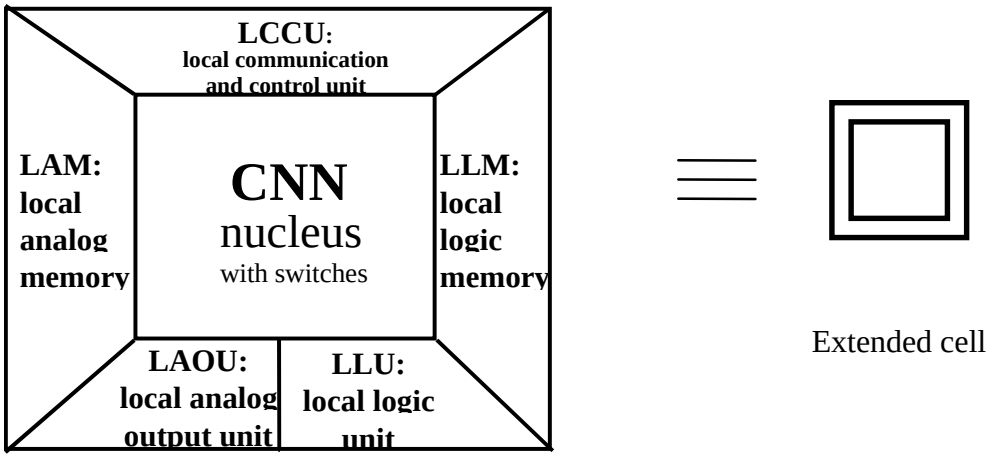
1. The stored program, as a sequence of templates, could be considered as a genetic code for the CNN-UM. The elementary genes are the templates, in case of r=1 it is a 19-real-number code. This, in a way is a minimal representation of a complex spatio-tempral dynamics.
2. In the nervous system, the consecutive templates are placed in space as subsequent layers.

9.1 The architecture

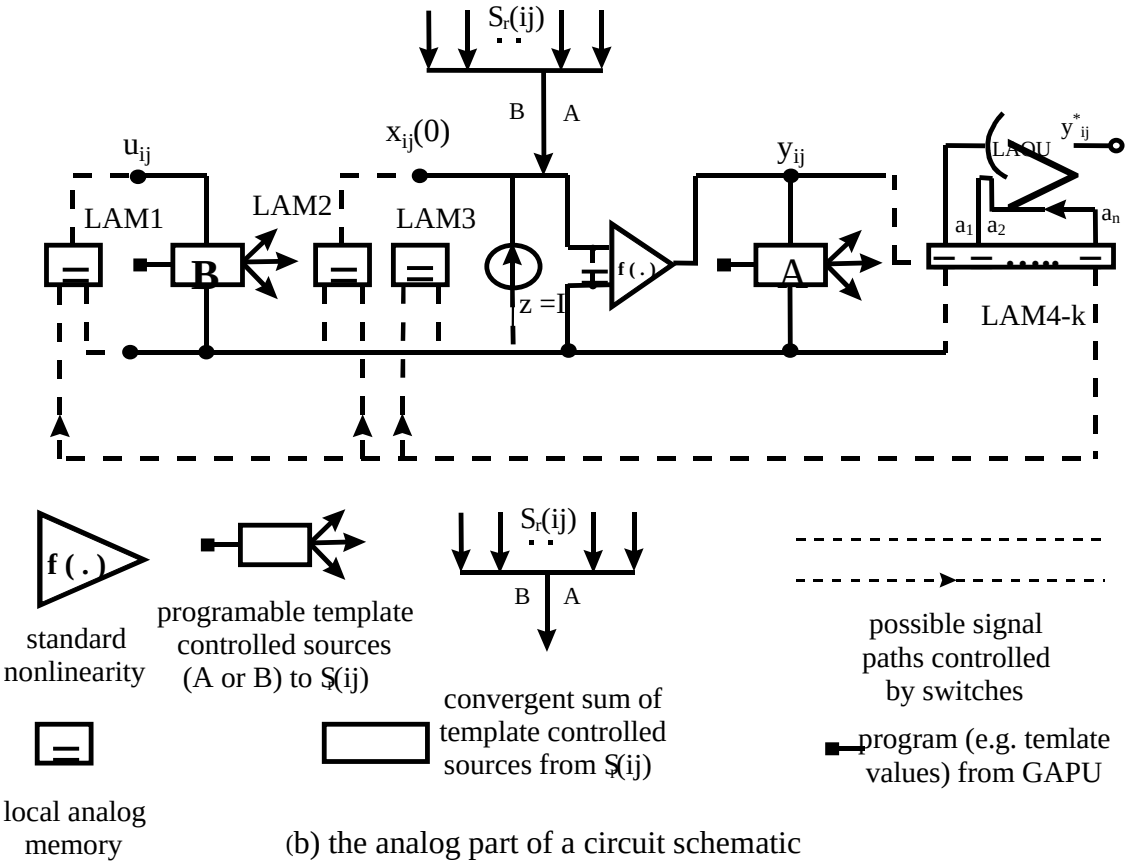
9.1.1 The extended standard CNN universal cell

Actually, in chapter 7, we have shown almost all of the various components we need in the extended standard universal cell, shown schematically in Figure 1.

[1] an operator $y(t)= \hat{y}\,(u_1(t), u_2(t), ..., u_n(t))$ is of fading memory if $\Delta y\,(t)|_{t=t0} \rightarrow 0$ as $\Delta u_i(t-\tau)$ is bounded and $\tau \rightarrow \infty$.



(a) the main components in the extended cell



(b) the analog part of a circuit schematic

Figure 1 (a) and (b)

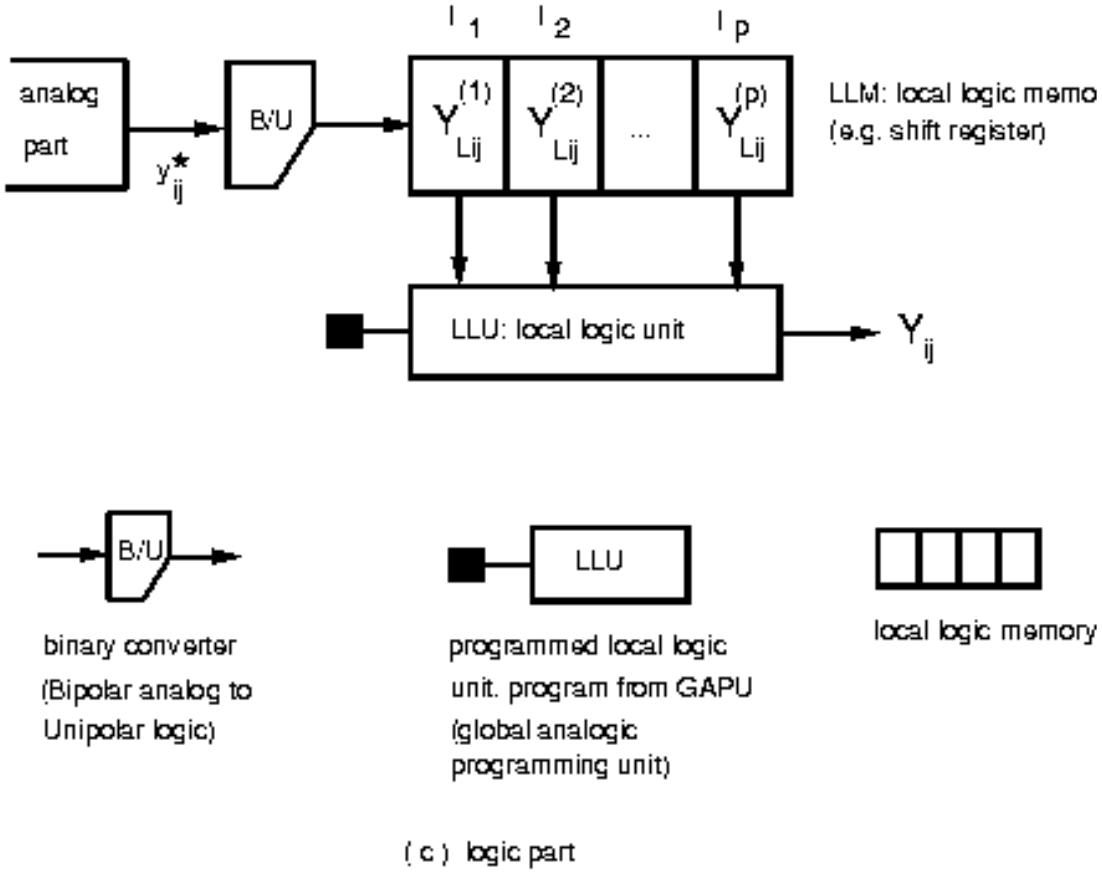


Fig.1 The extended standard CNN universal cell. (a) the main components (b) the analog part of a circuit schematic. (c) the logic part.

We have two elements not yet introduced in chapter 7.

The *local analog output unit* (LAOU) is a multiple-input single output analog device. It has the same function for continuous signal values as the local logic unit (LLU) for logic values. Namely, it combines local (stored) analog values into a single output. We may have used it for analog addition in section 7.6, instead of using the CNN cell for addition.

The *local communication and control unit* (LCCU) receives the programming instructions, in each cell, from the *global analogic programming unit* (GAPU), namely

- the analog template values (A, B, and z).
- the logic function codes for the local logic unit, and
- the switch configuration of the cell specifying the signal paths and some settings in the functional units (e.g., f(.), LAOU, GW(.)).

This means, at the same time, that we need registers (storage elements) in the GAPU for these 3 types of information, namely:

- an analog program register (APR) for the CNN templates.
- a logic program register (LPR) for the LLU functions, and
- a switch configuration register (SCR).

In Figure 1(b) the analog part of a circuit schematic of the cell is shown. We are keeping in mind an electronic or a physiological model, though, except a capacitor, no implementation-dependent elements are shown. An electronic integrated circuit (VLSI) implementation of these elements will be discussed in Chapter 15.

We assigned separate local analog memory places for the input (u), initial state (x(0)), threshold (z), and a sequence of outputs ($y^{(n)}$), however, a single local analog memory with a few places can also be used for all of these signals/data.

In Figure1(c) we show the logic part. We have introduced the elements already in chapter 7. The "global wire" (GW(.)) operator receives inputs from all cells, their cell logic outputs are $Y_{ij} := Y^{(k)}_{ij}$, k: specified.

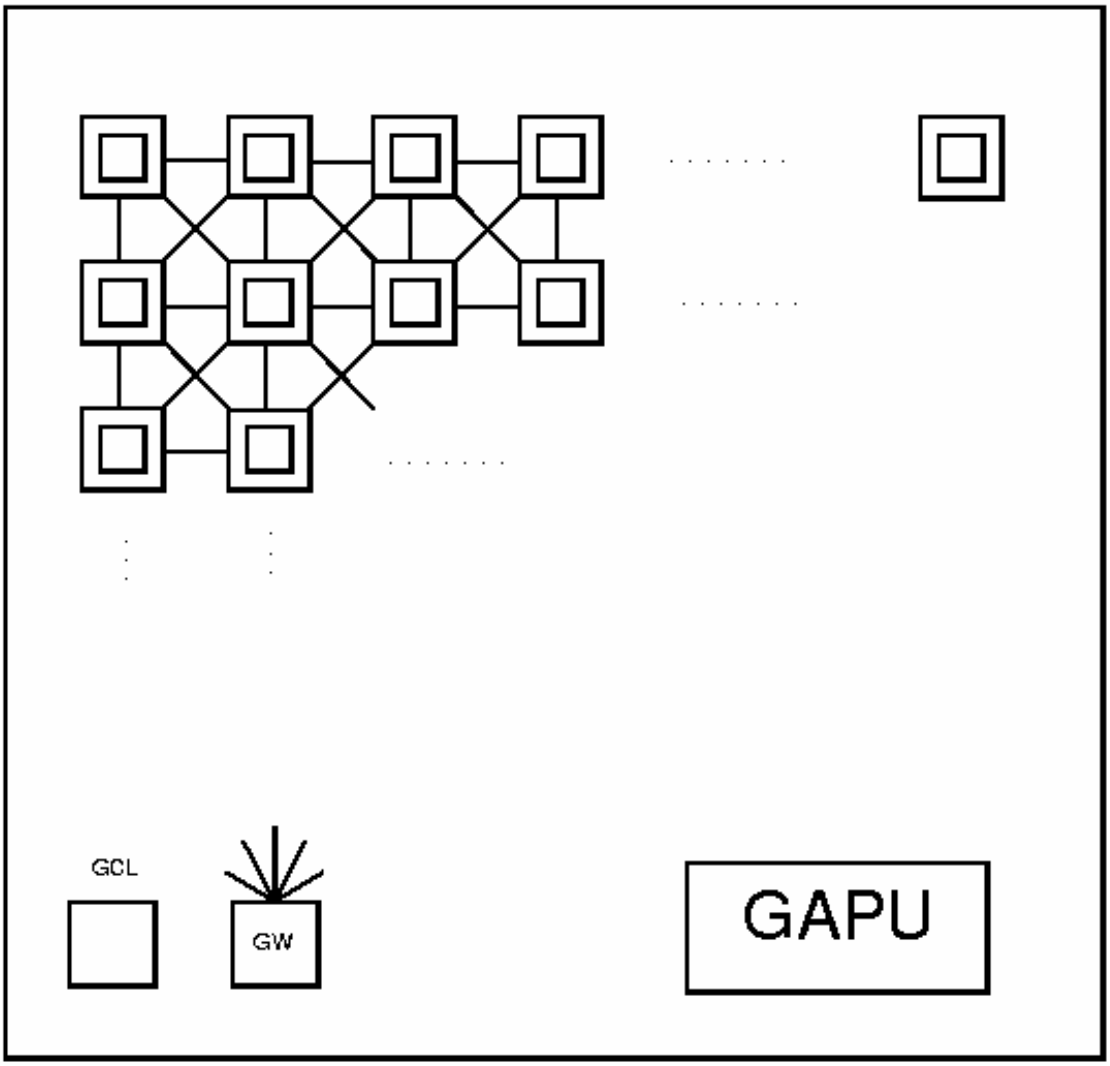
Remarks:

There are other, very useful possibilities related to a "global wire". For example, weighted analog outputs of each row may be calculated and added for the whole array.

9.1.2. The global analogic programming unit (GAPU)

This unit is the "conductor" of the whole analogic CNN universal machine, it directs all the extended standard CNN universal cells.

Figure 2 shows that, in addition to the 3 registers we already discussed in section 9.1.1 (i.e., the APR, LPR, SCR), the global analogic programming unit (GAPU) hosts the main control of the array which is placed in the global analogic control unit (GACU). Indeed, this is the (digital) machine code of the sequence of instructions of the given analogic CNN program.



GCL : global clock

GW : global wire

: extended standard CNN
Universal cell

GAPU	
APR:	analog programming instruction register
LPR:	logic program instruction register
SCR:	switch configuration register
GACU:	global analogic control unit

Fig. 2 The structure of the CNN universal machine

Why stored programmability is possible?

In digital computers, we tacitly assume and taken for granted that, for any sequence of instructions,

(i) all the transients decay within a specified clock cycle, and

(ii) all the signals remain within a prescribed range of dynamics (including dissipation, slope, etc.).

These conditions are not trivial in digital implementations either. Think about what would happen if a 75 MHz Pentium processor would have a clock of 100MHz. Clearly it would not work because of violating the first condition above. It may even destroy it due to violating the second condition.

A unique feature of the CNN dynamics and the CNN-UM architecture is that we can assure conditions (i) and (ii) as well. It is much less trivial here than in the digital case. Our main elementary instructions are the CNN templates and the local logic operations. But the CNN templates may induce the most exotic dynamics. The global clock (GCL) has a faster clock cycle for the logic part than for the analog part.

The global analogic control unit stores, in digital form, the sequence of instructions. Each instruction contains the operation code (template or logic), the selection code for the parameters of the operation (the code for the 19 values: A, B, z; or the code of the local logic function), and the switch configuration. The parameters are stored in the registers (APR, LPR, SCR).

Figure 3 shows the arrangement of the GAPU from this point of view.

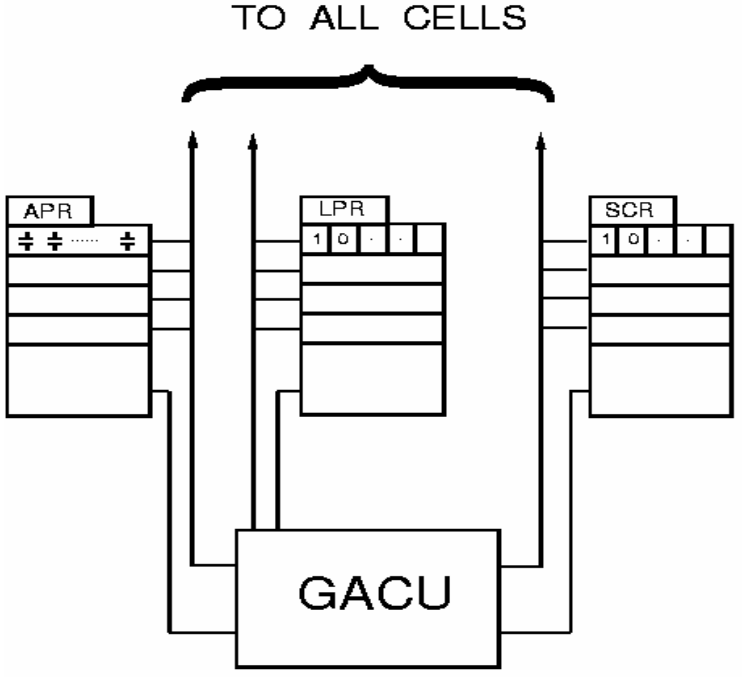


Fig. 3. The organization of the GAPU

9.2 A simple example in more details

In this example we show a complete sequence of various forms of an analogic CNN program as it is executed on a CNN Universal Machine. The outline and description of such a program contains the following information.

- *Global task*
- The *flow diagram* of the algorithm.
- The description of the algorithm in high level α language (*analogic CNN language*) or in an assembler (the analogic machine code, AMC).
- The result of an α compiler in the form of *an analogic machine code (AMC)* as a sequence of *macro instructions* and its *binary form (optional).*

The physical code generated by the CNN operating system and the controlling CNN chip "platform" is not shown here.

This example, called BARS-UP, is interesting in itself. The *global task* is shown in Fig. 4, we have to detect all objects, which have bars pointing upwards, and a continuous (to this bar) middle segment (many animals are responding to these objects by firing some neurons in their infero-temporal cortex).

The flow diagram of the analogic CNN algorithm is shown in Figure 5 with the intermediate results. The α language description (Version 2.1) is shown in Figure 6. We will show later the other codes generated by the α compiler.

The global task is: detect those objects which have bars pointing upwards. A typical input→output image pair is shown below. The original image is called BarsUpTest, the output is RESULT.

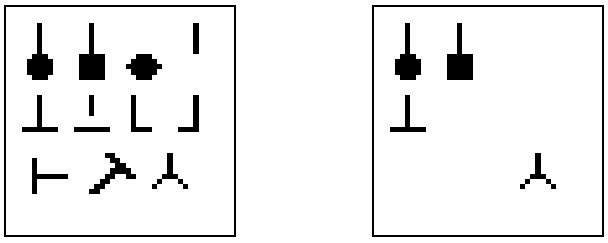
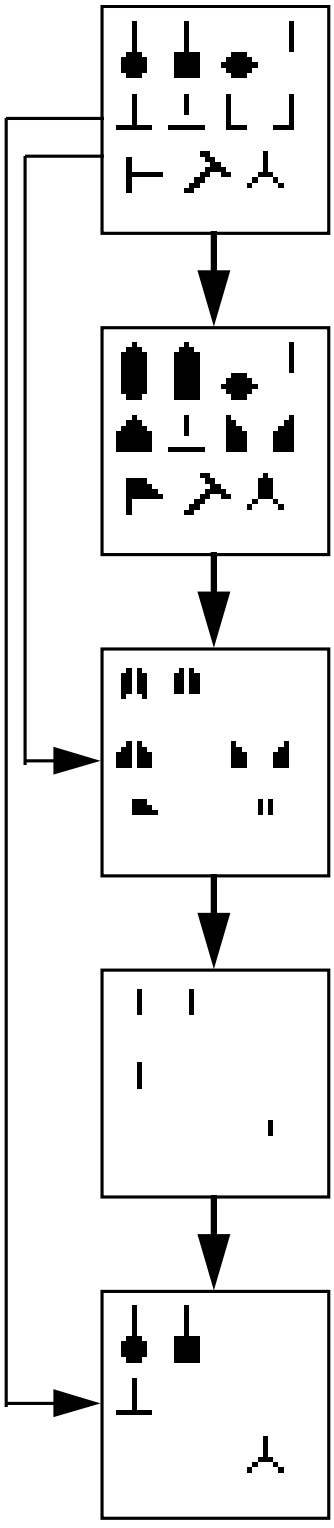


Figure 4: The global task

Remarks:

Here, we have a 5x5 template. Its actual physical implementation is not considered here. There are several ways to realize this "large neighborhood" CNN template. For example, to decompose it into several 3x3 templates.



Original input image
BarsUpTest

Result of HOLLOW template:

A =

0.5	-.5	0.5
0.5	2	0.5
0.5	0.5	0.5

B =

0	0	0
0	2	0
0	0	0

z = 3

Logic XOR applied to the previous two images

Result of HORDIST template:

A = 2

B =

0	0	0	0	0
0	0	0	0	0
0.25	-0.25	0	0	0
0	0	0	0	0
0	0	0	0	0

z = -1.5

Result of RECALL template:

A =

0.5	0.5	0.5
0.5	4	0.5
0.5	0.5	0.5

B =

0	0	0
0	4	0
0	0	0

z = 2.1

input: original image;
initial state: previous image

Fig. 5

Visual Feature Detection
(α-language, version 2.1)

```
        FUNCTION BARS-UP;
xLoad (LLM1, BarsUpTest);
        LLM3:= LLM1;
        HOLLOW(LLM1,LLM1,LLM2,10,-1);
        LOGXOR(LLM2,LLM3,LLM1,10,-1);
        HORDIST(LLM1,LLM1,LLM2,10,-1);
        RECALL(LLM2,LLM3,LLM1,10,-1);
xSAVE(RESULT,LLM1);
        ENDFUNCT;
```

Figure 6

Here, in this function description we have used two new α instructions:

xLOAD(local memory, file name) and

xSAVE(file name, local memory)

These are the input and output instructions from and to the digital environment.

9.3 A very simple example on the circuit level

In the following example, we will explain the functional details of the CNN-UM operation on the functional circuit level. Even though the example is very simple, it contains the micro steps. At the same time, it is not a transistor level description. Some transistor level implementation details will be described in Chapter 15.

The task:
Detect the horizontal intensity changes on a black-and-white image (Figure 7 shows an example)

The steps of the solution:

- detect those white pixels which have a black pixel on their direct right hand side (detection means to put the detected pixel to the black value, i.e. +1)
- detect those black pixels which have a white pixel on their direct right hand side
- apply a pixel by pixel logic OR function

The flow diagram of the algorithm and the templates:

The first step is performed by a template TEM1 and the second step by TEM2. The two results are combined with a local logic OR operation.

The flow diagram with image fragments representing input, output, and intermediate results is shown in Figure 7.

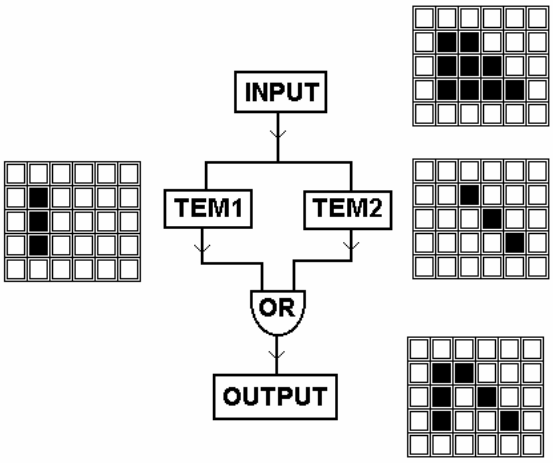


Figure 7. The flow diagram of the analogic CNN algorithm. Operation is illustrated on a simple test image fragment

The templates used in the CNN algorithm are as follows:

TEM1 (white to black):
$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ -2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = -1.5$$

TEM2 (black to white):
$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & -2 \\ 0 & 0 & 0 \end{bmatrix}, I = -1.5$$

The macro code of the algorithm:

As an example of the analogic macro code (AMC) description, we show the description of our very simple algorithm:

```
LOADTEM     >FF80, APR1          ; loading template (TEM1)
LOADTEM     >FF60, APR2          ; loading template (TEM2)
```

```
COPY        A_M2C, >FF40, LAM1          ; copy Analog image from
                                        ; Memory to Chip

RUNTEM      APR1, LAM1, LAM1, LLM1      ; run TEM1 template operation
RUNTEM      APR2, LAM1, LAM1, LLM2      ; run TEM2 template operation
RUNLOG      OR, LLM1, LLM2, LLM3        ; run local logic operation OR

COPY        L_C2M, LLM3, >FF00          ; copy binary (Logic) image
                                        ; from Chip to Memory
END
```

The syntax of the AMC instructions are simple:

```
LOADTEM     [source], [target];
COPY        [type], [source], [target];
RUNTEM      [template], [input], [init. state], [output];
RUNLOG      [type], [op1], [op2], [result];
```

The memory address is hexadecimal, ant the type of the image has a mnemonic name.

The core of the algorithm, in addition to the image and template downloading and the output image uploading, is represented by the 3 consecutive AMC instructions denoted by italic comments. That is:

- run TEM1 (stored in APR1) with input and initial state defined by the original input image stored in LAM1) and place the result (after converting from bipolar analog representation to unipolar binary one) in local logic memory LLM1
- run TEM2 (stored in APR2) with input and initial state defined by the original input image stored in LAM1) and place the result (after converting from bipolar analog representation to unipolar binary one) in local logic memory LLM2
- apply the local logic unit (LLU) with a logic OR operation on the two intermediate results stored in local logic memories LLM1 and LLM2 and place the result in LLM3

These three macro instructions will be converted into a series of elementary machine micro instructions, as shown later.

Next, we will not go into the details how the CNN operating system (COS) generate the machine micro code to be put into the GACU of the CNN Universal Chip (and how to fill the registers of the GAPU), however, we want to show the functional circuit level operation of an extended CNN cell. We will show soon the operations generated by the machine level micro instructions in the very details. First, we show an extended cell.

The functional circuit level schematics of an extended cell:

An extended cell is shown in Figure 8.

The local analog memory (LAM) has two places, LAM1 and LAM2. The analog cell contains two auxiliary storage capacitors at the input and at the state, respectively. The

iinput and **i**output values represent the weighted sums (as currents) from the inputs (B template) and from the outputs (A template) of the neighbor cells.

The local logic memory has three places, LLM1, LLM2, LLM3. LLM1 and LLM2 is implemented as a shift register, the input is stored on LLM1 and every new input shifts the content by one place to right (from LLM1 to LLM2, etc.). If we want to store a LAM value in (LLM1, LLM2), an automatic bipolar analog to unipolar binary converter is applied, shown after sw4. The local logic unit (LLU) in this cell is an OR function. It has a direct LLM3 output buffer.

In this extended cell we have six switches: sw0, sw1, sw2, sw3, sw4, sw5. Depending on their positions, ON or OFF, they code different switch configurations. The sequence of switch configurations is stored in the switch configuration register (SCR). Below, we show five switch configurations (sconf0, sconf1, sconf2, sconf3, sconf4) which define five actions in each and all cells (fully parallel).

Switch configuration; and corresponding action	sw0	sw1	sw2	sw3	sw4	sw5
sconf0 ; load input and initial state from LAM1	off	on	on	off	off	off
sconf1 ; start transient	on	off	off	off	off	off
sconf2 ; store the result in LAM2	on	off	off	on	off	off
sconf3 ; store LAM2 in LLM	off	off	off	off	on	off
sconf4 ; activate the logic operation and put the result in LLM3	off	off	off	off	off	on

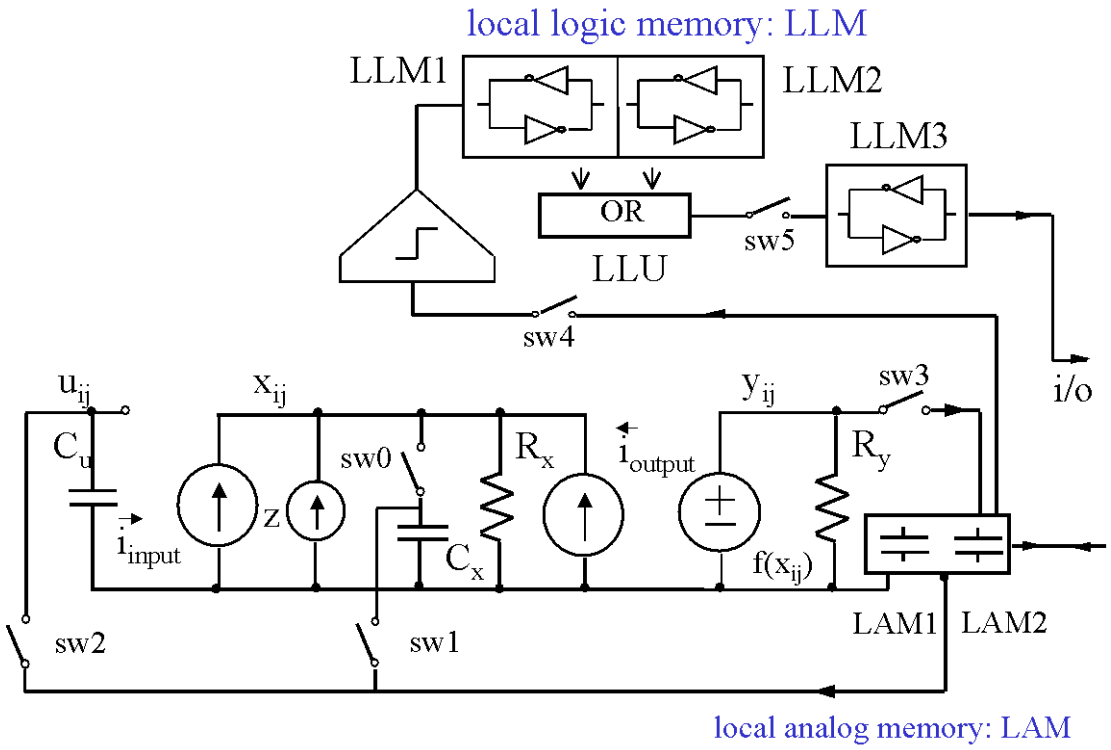


Figure 8. A very simple extended cell with the six switches, sw0, sw1, sw2, sw3, sw4, sw5 and the logic output (at the output of LLM3). It is supposed that the input image has been downloaded to LAM1.

The content of the Global Analogic Programming Unit (GAPU) :

First we specify the registers. Part of the content of the switch configuration register (SCR) has already been defined. This will be enough for running the three consecutive core macro instructions defined above.
The analog program instruction register (APR) contains two templates, that is, the two sets of the 19 numbers defined by TEM1 and TEM2, coded some appropriate way in APR1 and APR2.
The logic program instruction register (LPR) contains the codes for the logic operations of the local logic unit (LLU), here we need only the OR operation, it is stored, and coded in an appropriate way in LPR1.

The sequence of the actions in the CNN Universal Machine with our simple extended cell, and the registers defined right now, is coded in the Global Analogic Control Unit (GACU). In our example, for the three macro instructions defined above, for implementing the core of our algorithms (running the two consecutive templates and the logic OR operation with the appropriate storage of the intermediate results), the sequence of macro instructions of the GACU are as follows.

Here, we suppose that the templates, the local logic operator and the input image are loaded (TEM1 and TEM2 in APR1 and APR2, respectively, the OR operation in LPR1, and the input image, pixel by pixel, in the LAM1 place of each extended cell). Then the next sequence is applied:

Action code	*Comment*
sconf0;	load input and initial state from LAM1
select APR1;	tune the template element values defined by TEM1
sfonf1;	start the analog spatiotemporal transient
sconf2;	store the result in LAM2
sconf3;	store LAM2 in LLM1
sconf0;	load input and initial state from LAM1
select APR2;	tune the template element values defined by TEM2
sfonf1;	start the analog spatiotemporal transient
sconf2;	store the result in LAM2
sconf3;	store LAM2 in LLM1 (the former LLM1 value will be automatically shifted to LLM2)
select LPR1;	tune to the local logic operation OR
sconf4;	calculate the OR operation and store the result in LLM3

In the first two action groups, the first two actions are made parallel.
The five extended cell configurations corresponding to sconf0, sconf1, sconf2, sconf3 and sconf4 are shown on Figures 9, 10, 11, 12, and 13, respectively. The comments are referring to the last two action groups (activating TEM2 and OR). The closed switches are shown bold. Hence, it is easy to detect the active parts of the circuit.

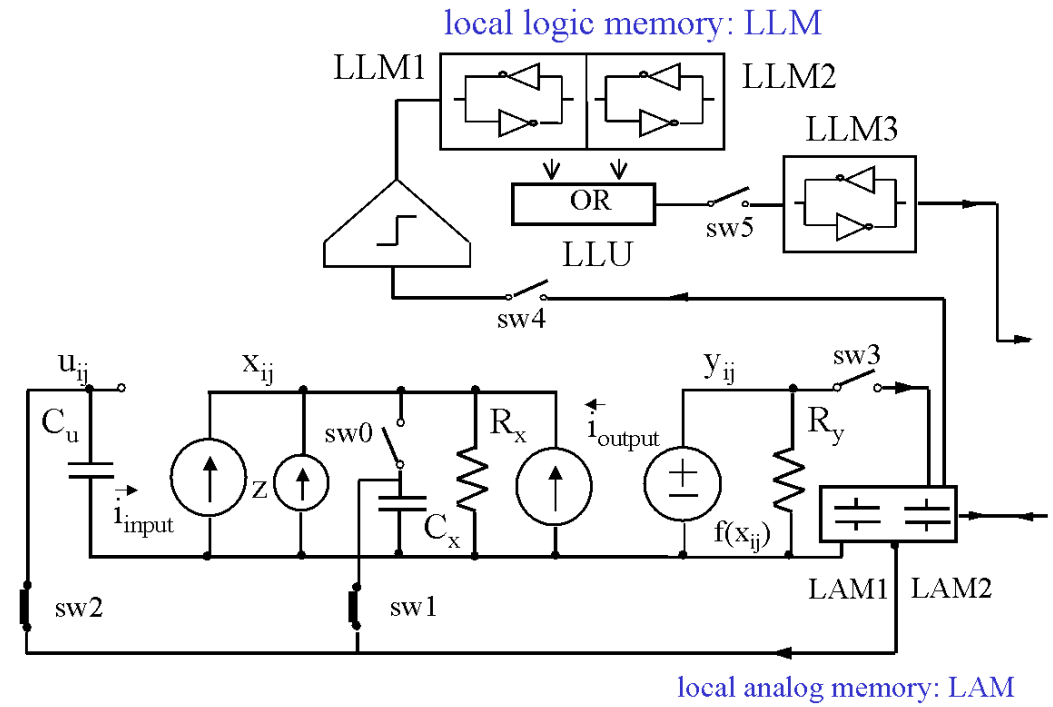


Figure 9. Sconf0; load input and initial state from LAM1

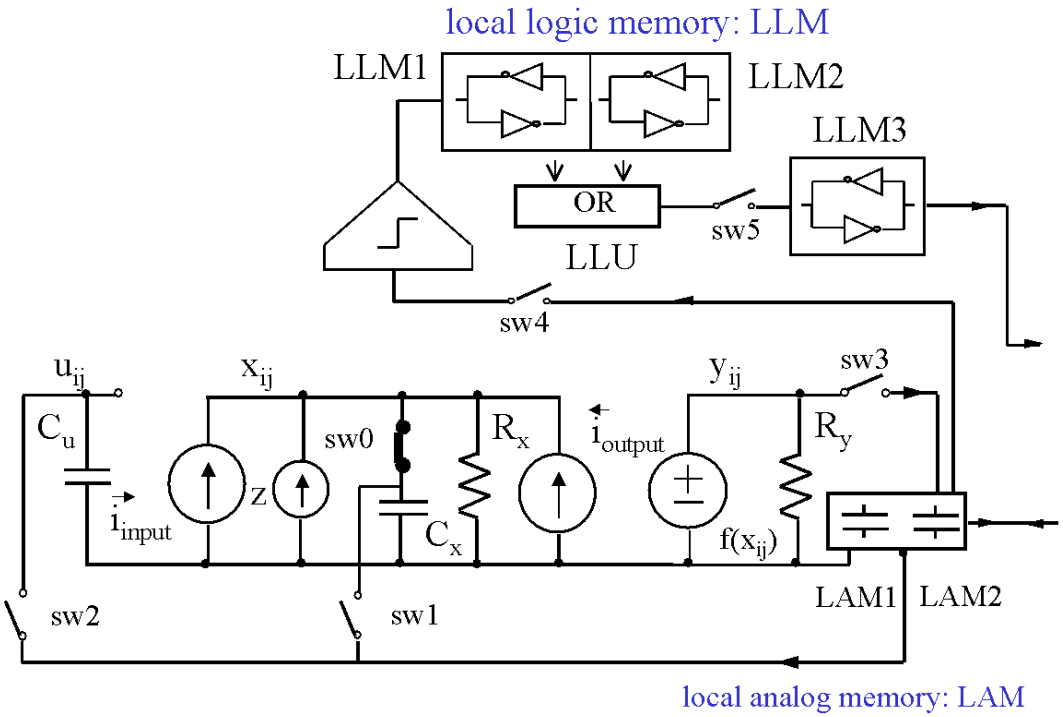


Figure 10. Sconf1; start transient

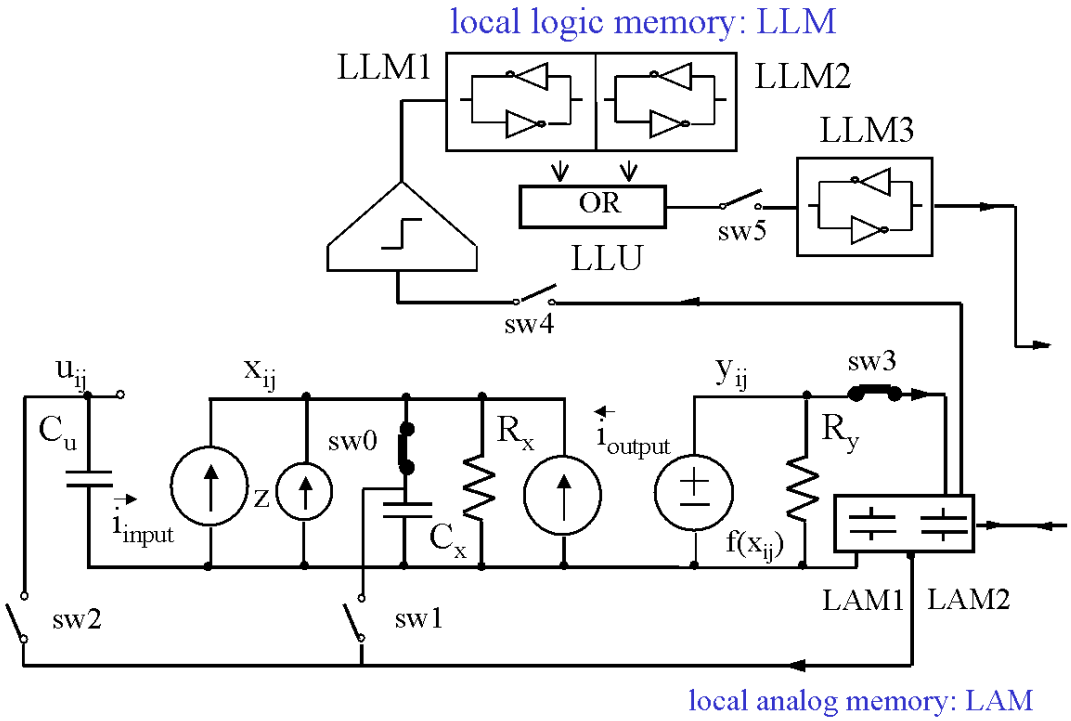


Figure 11. Sconf2; store the result in LAM2

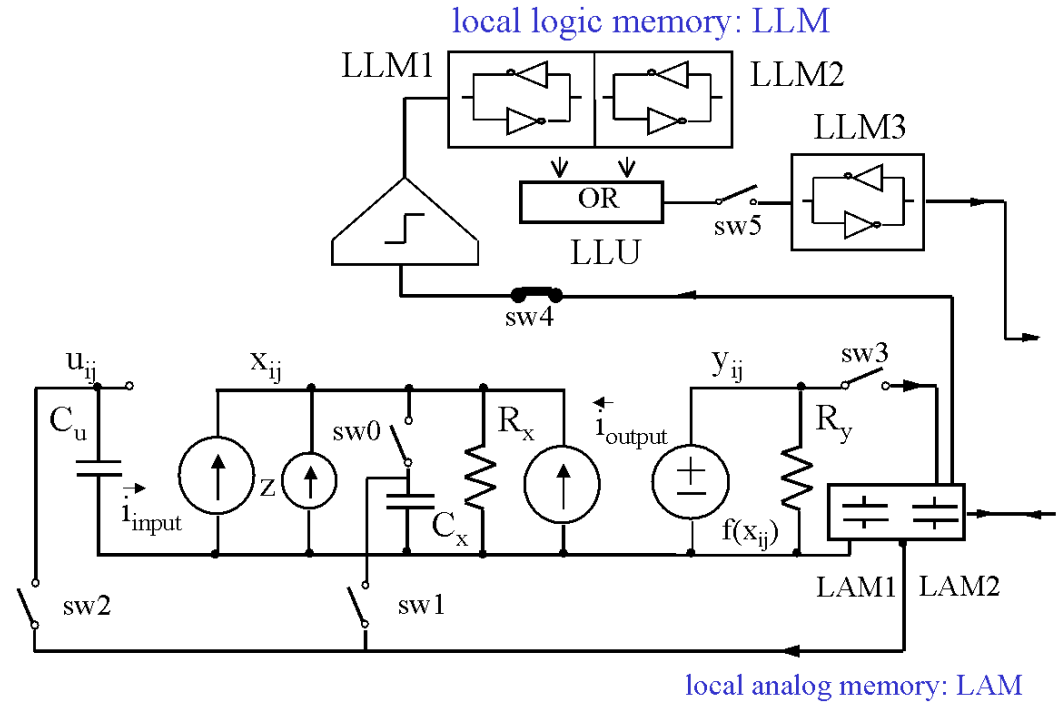


Figure 12. Sconf3; store LAM2 in LLM

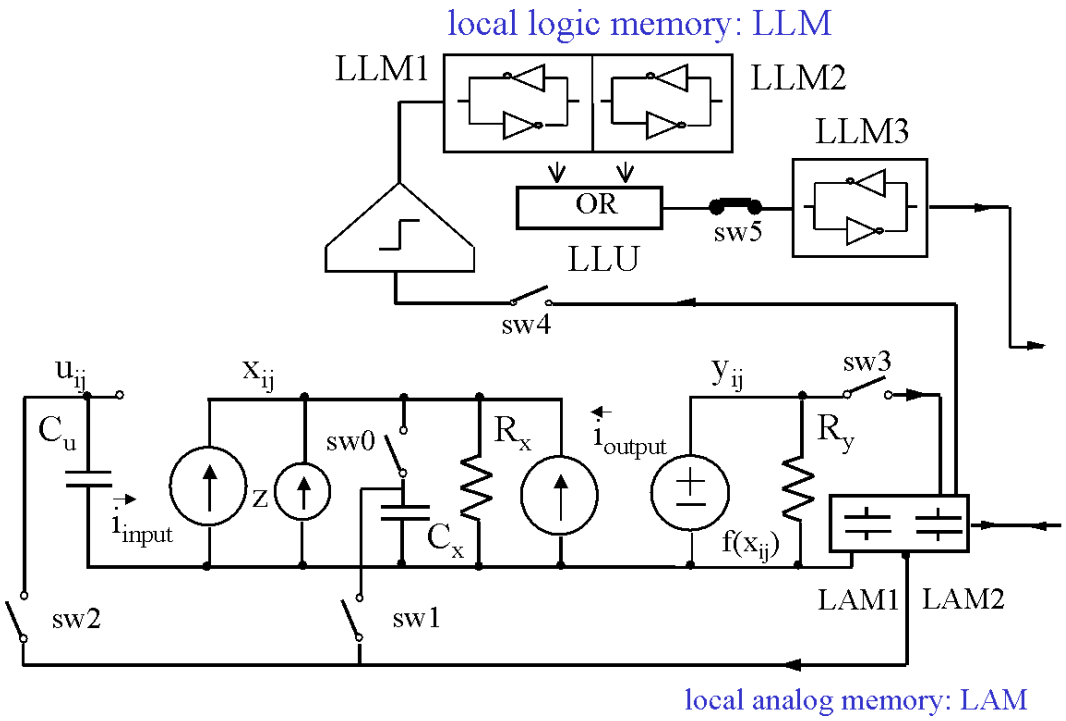


Figure 13 Sconf4; activate the logic operation and put the result in LLM3

9.4 Language, compiler, operating system

In the preceding Chapters we have learned a few languages of different levels to describe the analogic CNN algorithms. In Figure 14 we summarize the various steps how our high level α instructions code will be translated into a running program on a physical chip. It shows the main software levels of this process.

On the lowest level, the chips are embedded in their physical environment. The AMC code will be translated into firmware and electrical signals.

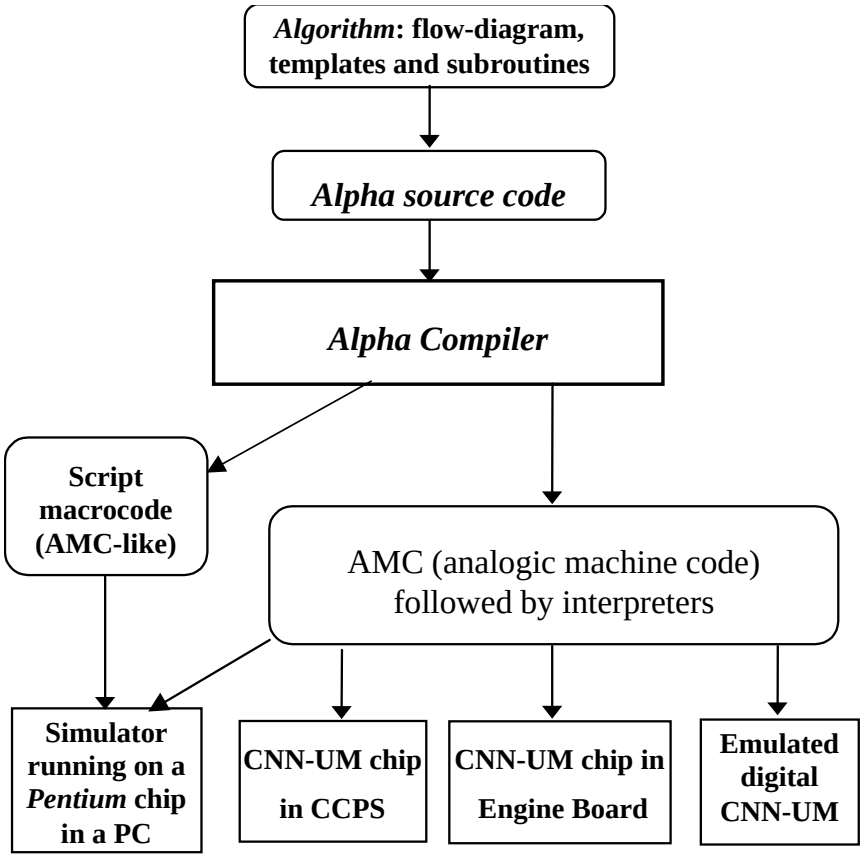


Figure 14. The levels of the software and the core engines

On the highest level, the **α compiler** generates a macro (assembly) level code called analogic macro code, AMC. The input of the **α compiler** is the description of the flow diagram of the algorithm using the **α language**.

The AMC like CNN Script Description (CSD) code is used for the software simulations to control the different parameters of the simulation as well as to specify the graphical demonstration of the results, as we have shown in Chapter 4. Here, the physical processor is the Pentium microprocessor, controlled by the physical code running under an operating system (like WINDOWS or UNIX). The simulator can also be used directly from the α source code via the compiler and the AMC (with default operating and graphical parameters).

As an example for an AMC code in assembly format and in hexadecimal format, these codes for the program example BARS-UP, described in Section 9.2, are shown in Figures 15 and 16, respectively.

Analogic Macro Code (AMC) description of BARS-UP

```
COPY    B2C_L2L, >FFC0, 1 * board to chip copy ( to LAM1 )
LOADT   >FFA0, 1          * load template1
LOADT   >FF80, 2          * load template2
LOADT   >FF60, 3          * load template3
RUNA    1, 1, 1, 2        * run template1
RUNTL   CXOR, 2, 2, 2     * logic XOR
RUNA    2, 2, 2, 2        * run template2
RUNA    3, 2, 1, 2        * run template3
COPY    C2C_L2L, 2, >FFC0 * chip to board copy ( from LAM2 )
```

syntax:

COPY	[type], [source], [destination]
LOADT	[source], [destination]
RUNA	[template], [input], [init. state], [output]
RUNL	[type], [op1], [op2], [output]

All the parameters are chip or board memory addresses, except the [type] parameters

Figure 15.

Compiled Analogic Macro Code

in hexadecimal format

hexa	binary	code
12h	0000 0000 0001 0010	COPY
8h	0000 0000 0000 1000	B2C_L2L
FFC0h	1111 1111 1100 0000	>FFC0
1h	0000 0000 0000 0001	1
62h	0000 0000 0001 0010	LOADT
FFA0h	1111 1111 1010 0000	>FFA0
1h	0000 0000 0000 0001	1
62h	0000 0000 0001 0010	LOADT
FF80h	1111 1111 1010 0000	>FF80
2h	0000 0000 0000 0010	2
62h	0000 0000 0001 0010	LOADT
FF60h	1111 1111 1010 0000	>FF60
3h	0000 0000 0000 0011	3
61h	0000 0000 0001 0001	RUNA
1h	0000 0000 0000 0001	1
1h	0000 0000 0000 0001	1
1h	0000 0000 0000 0001	1
2h	0000 0000 0000 0010	2
61h	0000 0000 0001 0001	RUNL
5h	0000 0000 0000 0101	5
5h	0000 0000 0000 0101	5
5h	0000 0000 0000 0101	5
2h	0000 0000 0000 0010	2
	.	
	.	
	.	

Figure 16

Consider now the CNN Universal Machine Chip, called CNN-UM chip. We need the appropriate software levels and a hardware-software environment. This is the CNN Chip Prototyping System (CCPS). In the CCPS we may also use the AMC code as the input.

In Figure 17 we show the flow diagram of the whole process down to the physical chip.

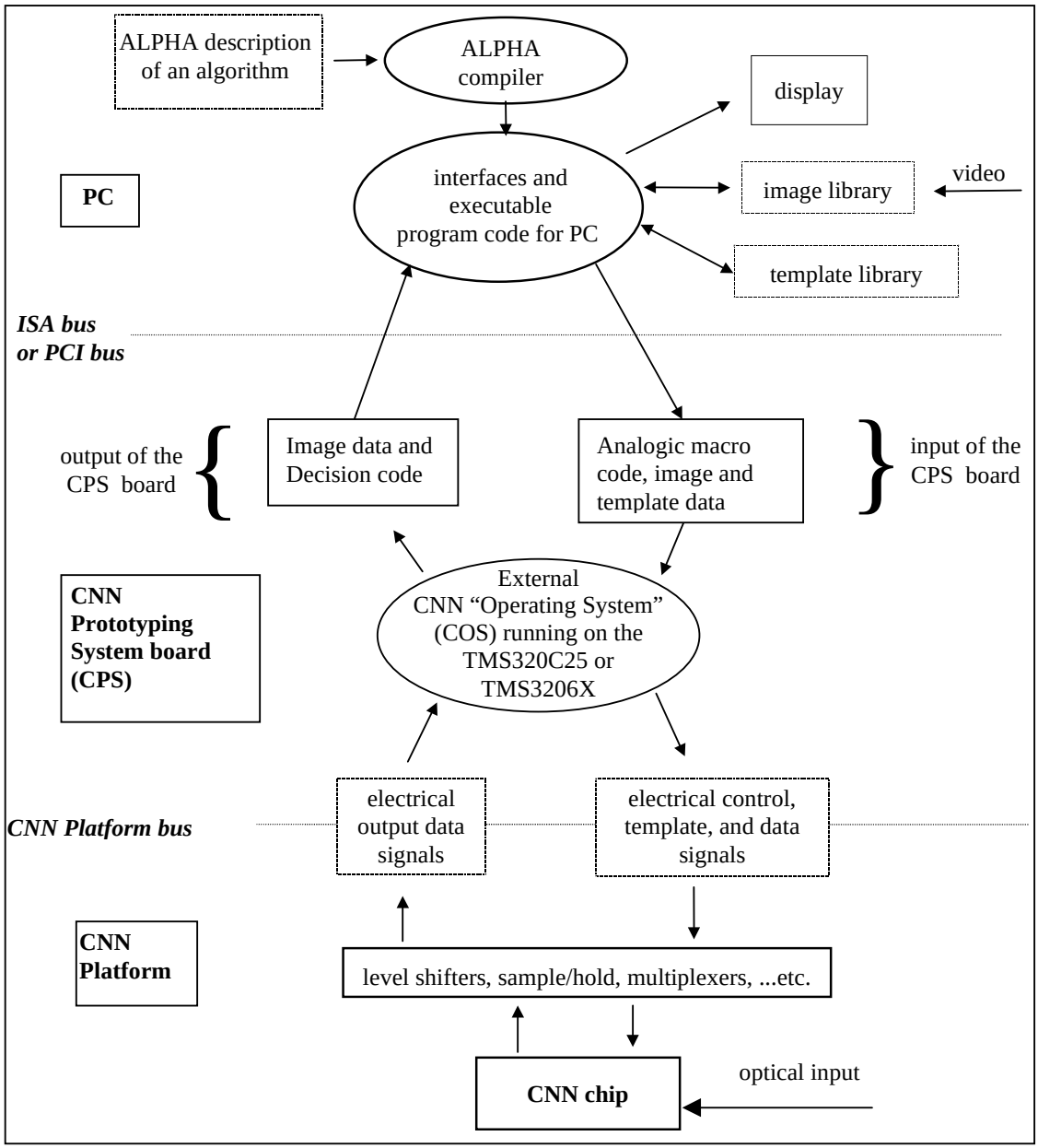


Figure 17 The architecture of the CNN Chip prototyping System (CCPS)

In this chip prototyping system the CNN-UM chip is hosted in a separate Platform, connected to a PC. A special purpose add-in-board, the Chip Prototyping System Board (CPS board) is serving as the hardware environment for the CNN Operating System (COS).

To make the whole CNN computer self contained we need a CNN Universal Chip set [2] and implement it on an Engine Board.

In single board or single chip solutions the CPS board and its software is integrated into the CNN-UM chip or board.

We stop here, not to explain more details. Our aim was to show that writing analogic CNN programs in high level languages (like the α language) the rest of the familiar computing infrastructure is ready to execute these programs in different formats and physical implementations. As to the latter, Chapter 15 will describe the main types and parameters of the physical implementations.

[2] T. Roska, "The CNN Chip set, engine board and the visual mouse", Proc. IEEE CNNA-96, pp. 487-492, Seville, 1996