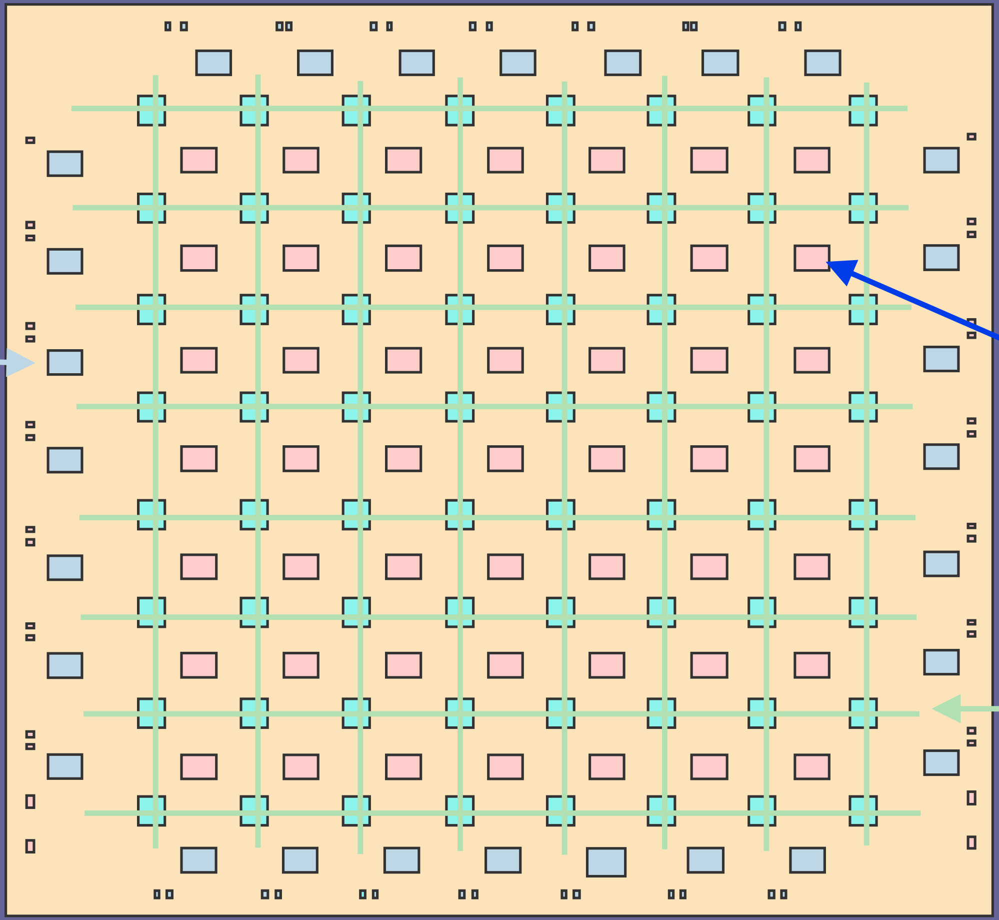
DEC PERLE Board as an EXAMPLE of RECONFIGURABLE HARDWARE

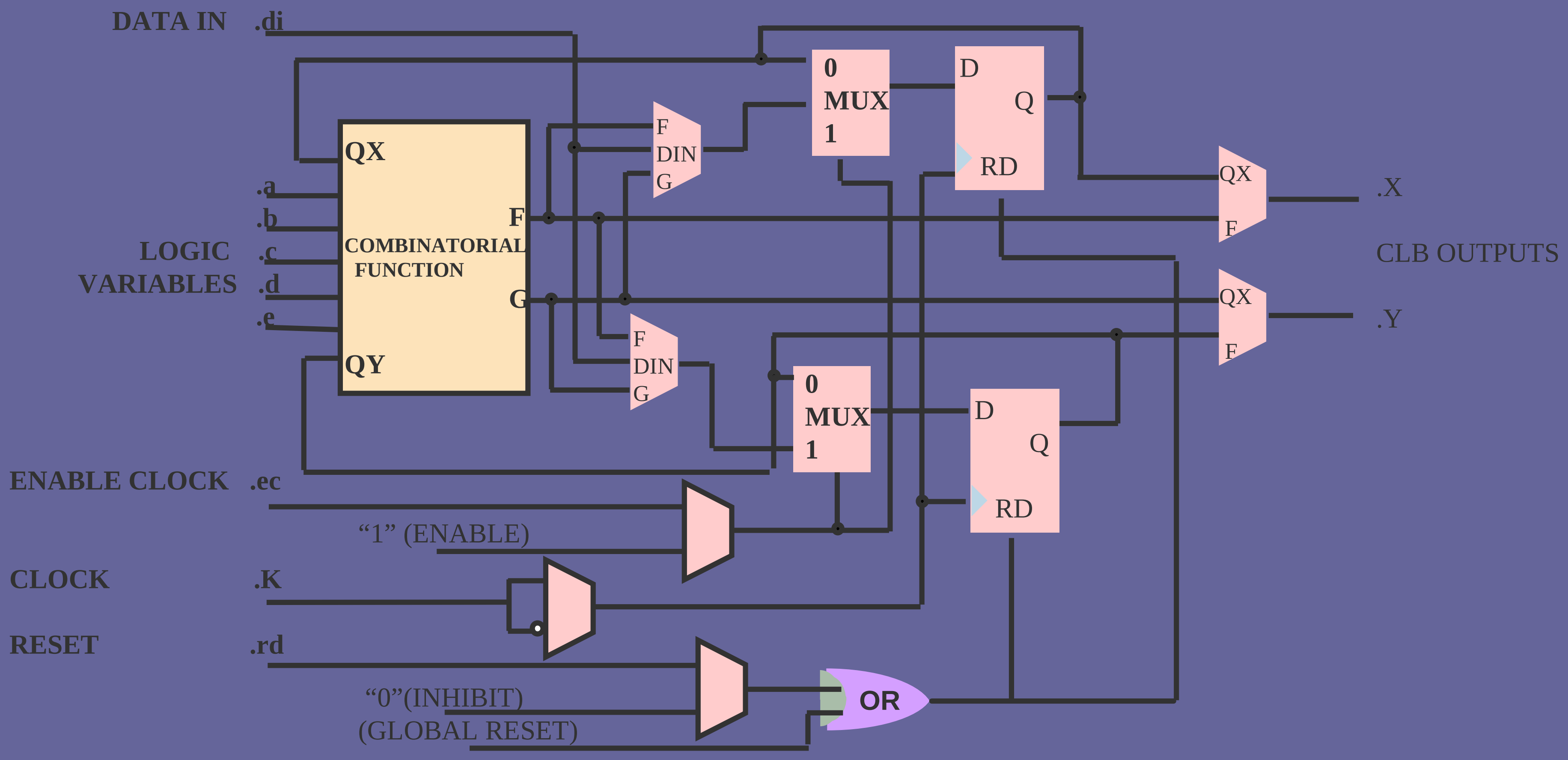
Xilinx Revolutionary Design

- The first commercial FPGA was introduced in 1986 by **Xilinx**
- This revolutionary component has a large internal configuration memory, and *two modes of operation:*
  - in **download mode**, the configuration memory can be written, as a whole, through some external device;
  - once in **configured mode** a FPGA behaves like a regular application-specic integrated circuit (ASIC).

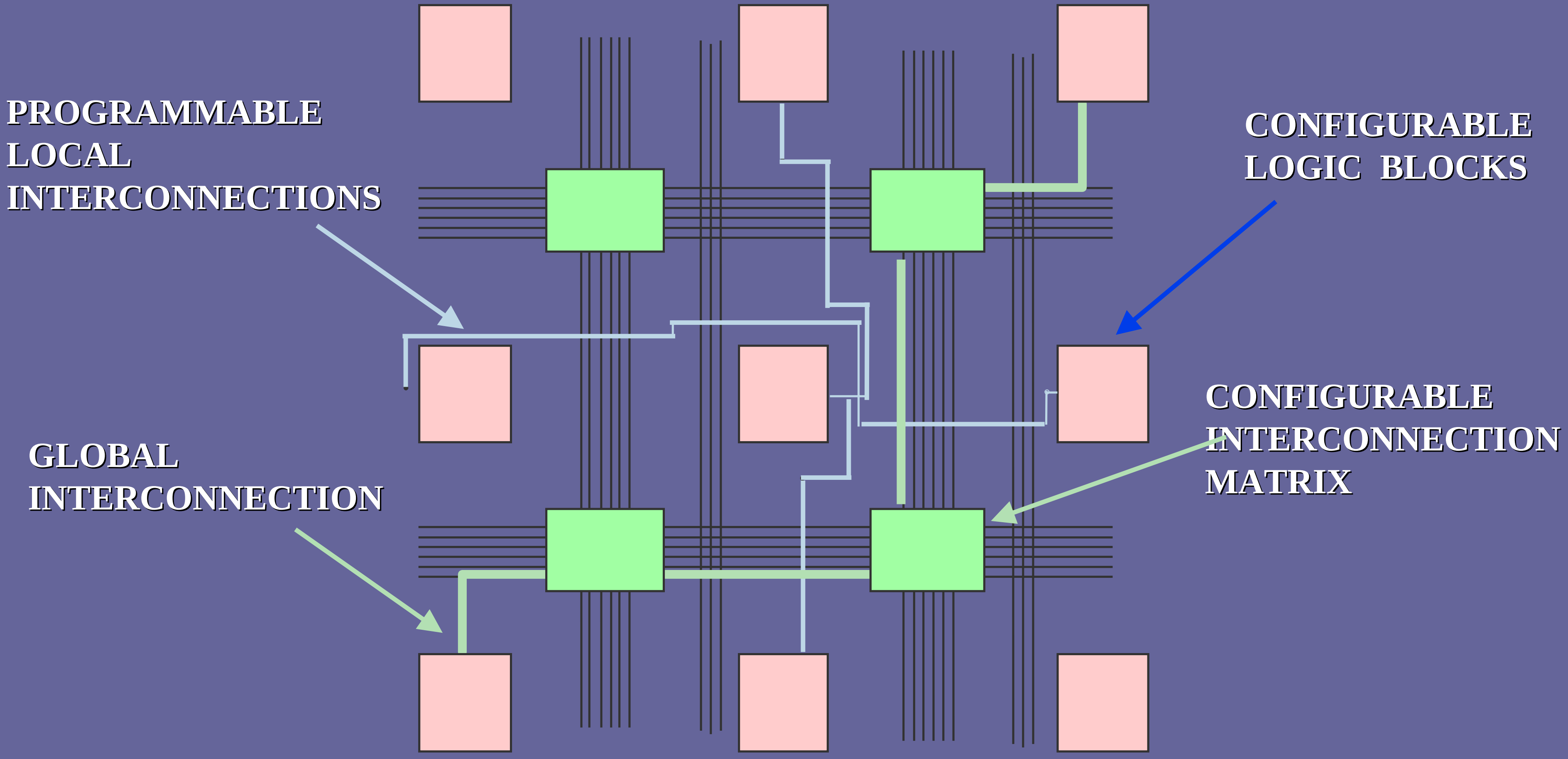
XILINX Field Programmable Gate Array


CONFIGURABLE INPUT/OUTPUT BLOCKS
CONFIGURABLE LOGIC BLOCKS
CONFIGURABLE GLOBAL INTERCONNECTIO

Configurable Logic Block



Interconnections



Xilinx Revolutionary Design

- To realize a FPGA, one simply connects together in a regular mesh,
  - **n * m** identical programmmable active bits (PABs).
- There are many ways to implement a PAB with the required universality. In particular, it can be built from either or both of the following primitives:
  - a configurable logic block implements a Boolean function with k inputs; its truth table is defined by 2^k (or less) configuration bits, stored in local registers;
  - a configurable routing block implements a switchbox whose connectivity table is set by local configuration bits.
- Such a FPGA implements a Von Neumann cellular automaton.
- What is more, the FPGA is a universal example of such a structure:
  - any synchronous digital circuit can be emulated, through a suitable configuration, on a large enough FPGA, for a slow enough clock.

Xilinx Revolutionary Design

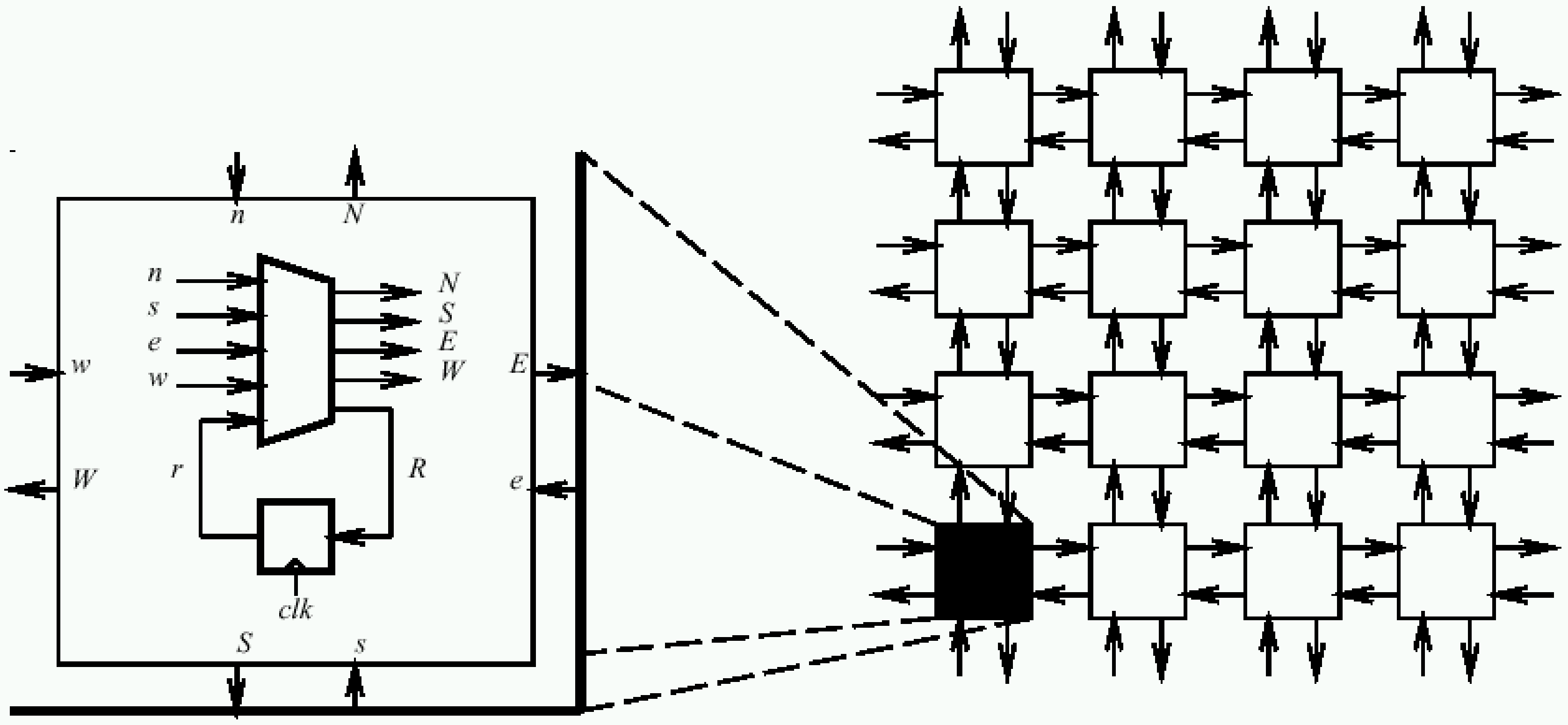
- The FPGA is a virtual circuit which can behave like a
  - number of **different ASICs:** all it takes to emulate a particular one is to feed the proper conguration bits.
- This means that **prototypes** can be made quickly, tested and corrected.
- The **development cycle** of circuits with FPGA technology is typically measured in weeks, as opposed to months for hardwired gate array techniques.
- But FPGAs are used **not just for prototypes**;
  - they also get incorporated in many production units.

Xilinx Revolutionary Design

- In all branches of the electronics industry other than the mass market, the use of FPGAs is expanding, despite the fact that they still cost ten times as much as ASICs in volume production.
- In 1992, FPGAs were the fastest growing part of the semi-conductor industry, increasing output by 40 %, compared with 10 % for chips overall.
  - As a consequence, FPGAs are on the ***leading edge*** of silicon chips.
- They grow bigger and faster at the rate of their enabling technology, namely that of the ***static RAM*** used for storing the internal configuration.

PAM=Programmable Array Memory
PAB=Programmable Active Bit

Only switchbox



This PAB has 4 inputs $\langle n, s, e, w \rangle$, 4 outputs $\langle N, S, E, W \rangle$, one register (flip-flop) with input R and output r, and a combinational gate

$$g(n, s, e, w, r) = (N, S, E, W, R)$$

The truth table of g is specified by $160 = 5 \times 32$ bits.

Host, Memory and FPGA Array

- PAM was a prototype of several chips and boards

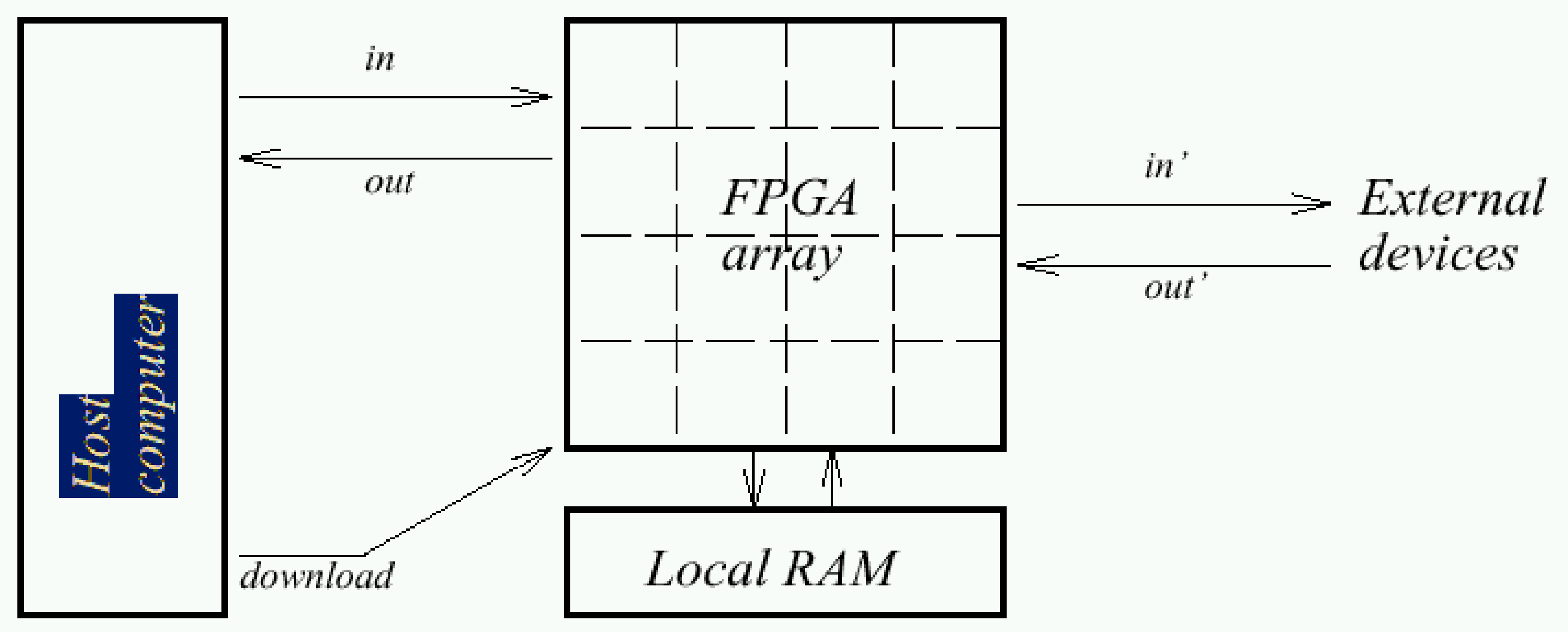


Fig. 2. Programmable Active Memory

• What is DEC-PeRLe Board?

It is a configurable coprocessor board, built with:

• 4×4 Xilinx XC3090 LCA matrix;

• 4×256k memory banks;

• 7 other LCAs, for switching and controlling functions.

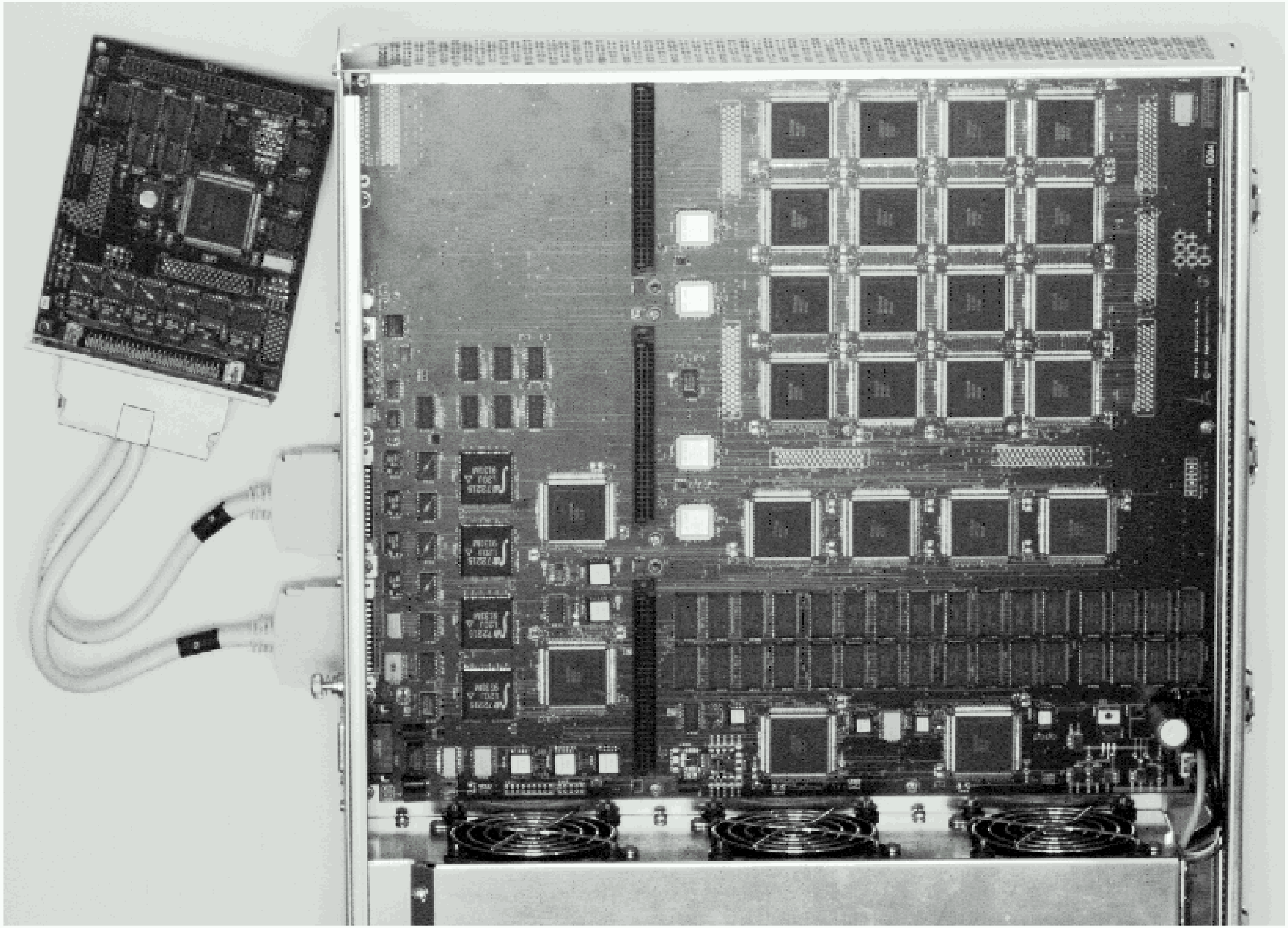


Fig. 4. DECPeRLe-1 and its TURBOchannel interface board

DEC-PERLE-1 architecture

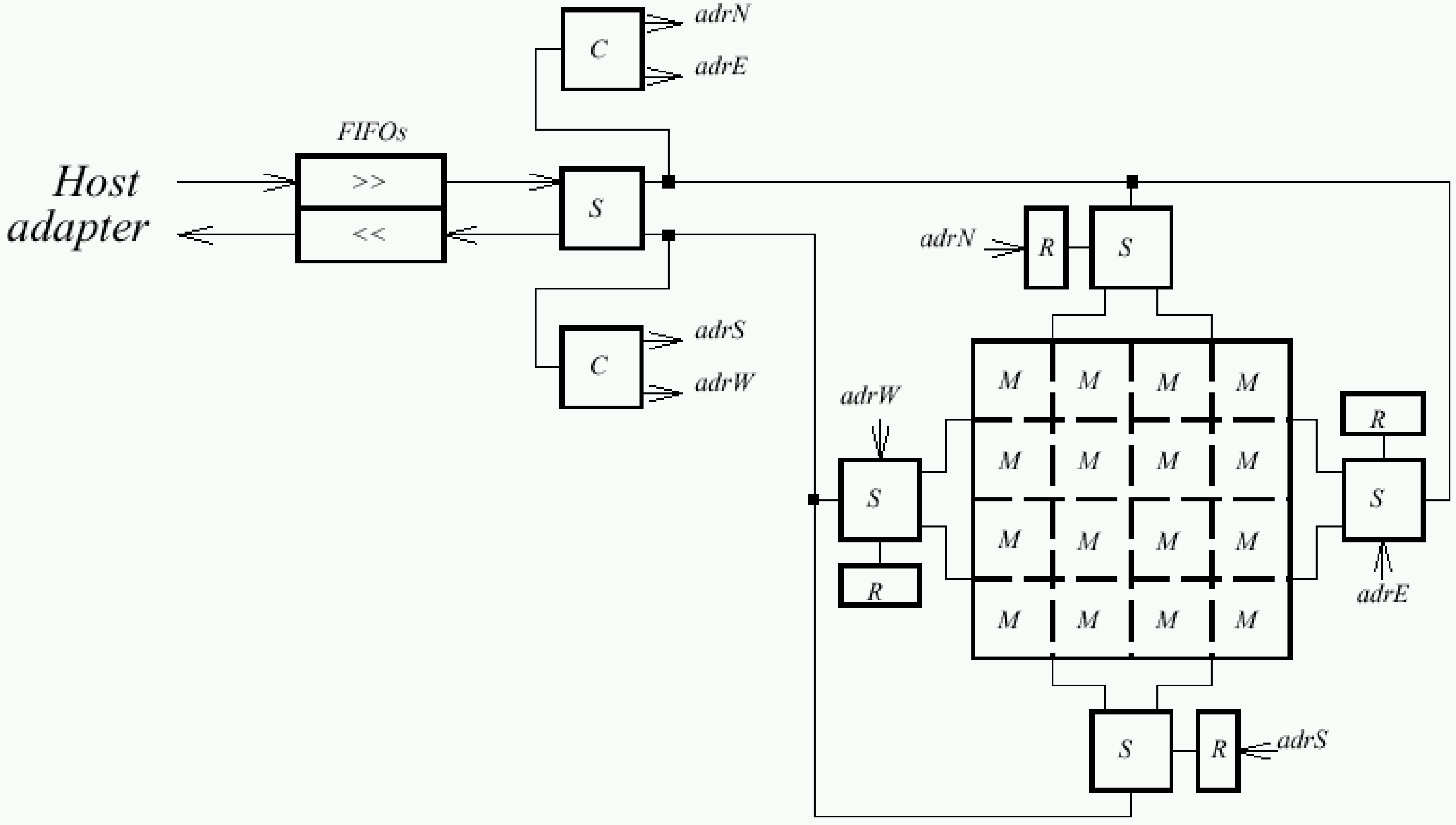


Fig. 3. DECPeRLe-1 architecture

COMPUTATIONAL MATRIX

- Matrix can be used to develop any kind of digital circuitry: data path, control unit and others.
- Typically used to develop the data path of the application
- The interconnection resource between them can be classified into the following three categories: direct connections, buses, and rings.
- **Direct Connections** . These wires connect the adjacent sides of adjacent LCAs.
- The main purpose of direct connections is to extent the internal regularity of the LCA to the matrix level
- The matrix can be seen as a large FPGA with **64 * 80** Configurable Logic Blocks (CLBs) (one XC3090 FPGA has **16 * 20** CLBs). Each LCA has 16 such wires on each side.
- The **direct connections** at the edges of the FPGA matrix four 64-bit-wide connections connected to external connectors, which can be used to connect other devices, for example, another DEC-PERLE-1 board.
- The **buses.**
- The **rings.**

Debugging and reconfiguring

- We take advantage of an extra feature of the XC3090 component:
  - it is possible to **dynamically read back** the contents of the internal state register of each PAB.
- **Clock stepping** facility - stop the main clock and trigger clock cycles one at a time from the host.
- **dynamically read back and clock stepping** provide a powerful debugging tool, where one takes a snapshot of the complete internal state of the system after each clock cycle.
- This feature drastically reduces the need for software simulation of DEC PERLE designs.

One paradigm was systematically applied:

- Cast the inner loop in PAM hardware; let software handle the rest!

DEC-PERLE-1 architecture

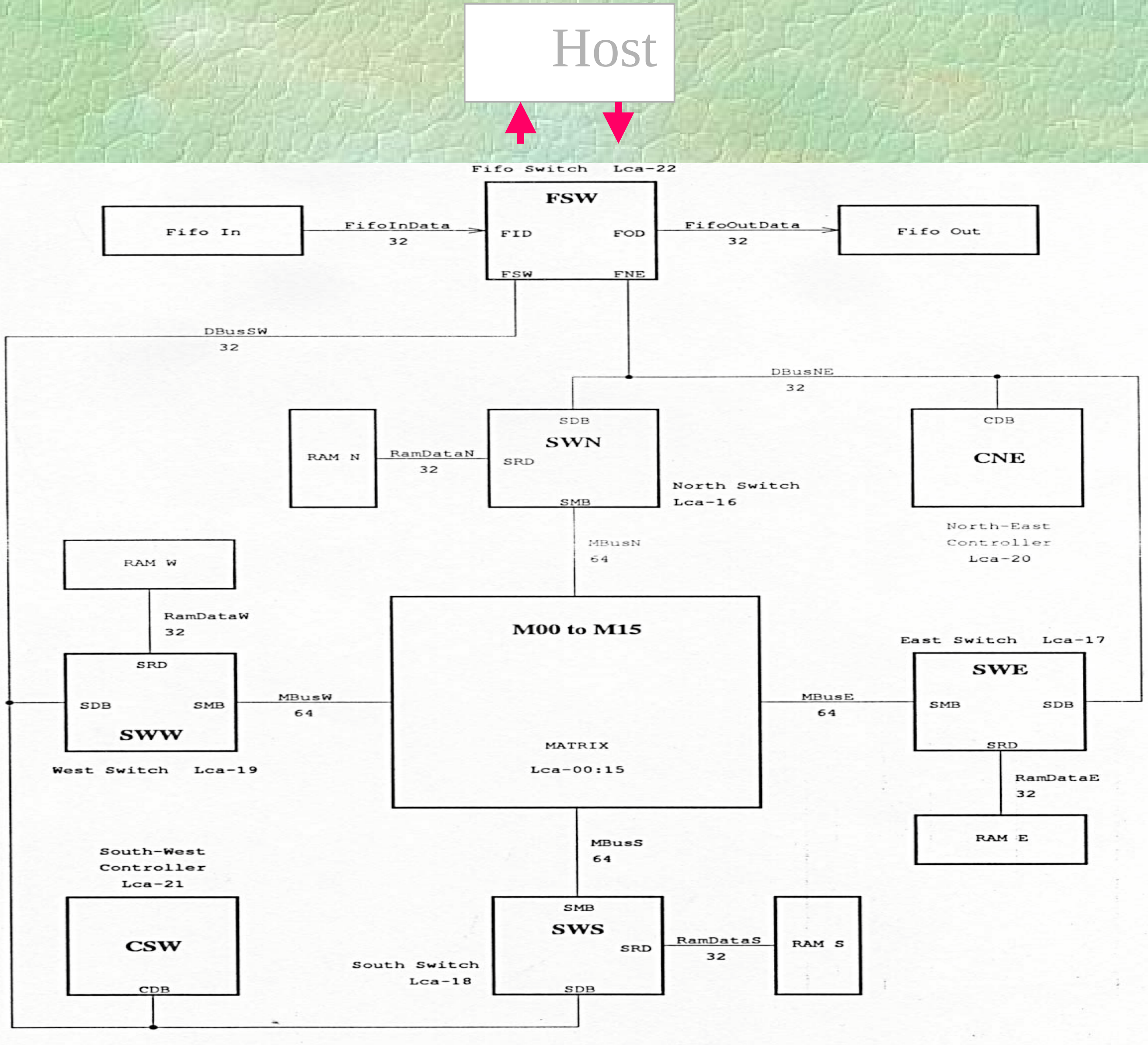


Figure 3: Perle-1 Data Buses

• Computational matrix and interconnections

The 4×4 Xilinx XC3090 LCAs

Interconnections:

Direct connections,

to expand the internal structure of the LCAs to the board level

(to a certain extent the entire matrix may be viewed as a unique and huge 2-D regular

array of bit-level programmable logic cells);

Buses, global data distribution;

Rings,

connect all the LCAs in the matrix, global control distribution.

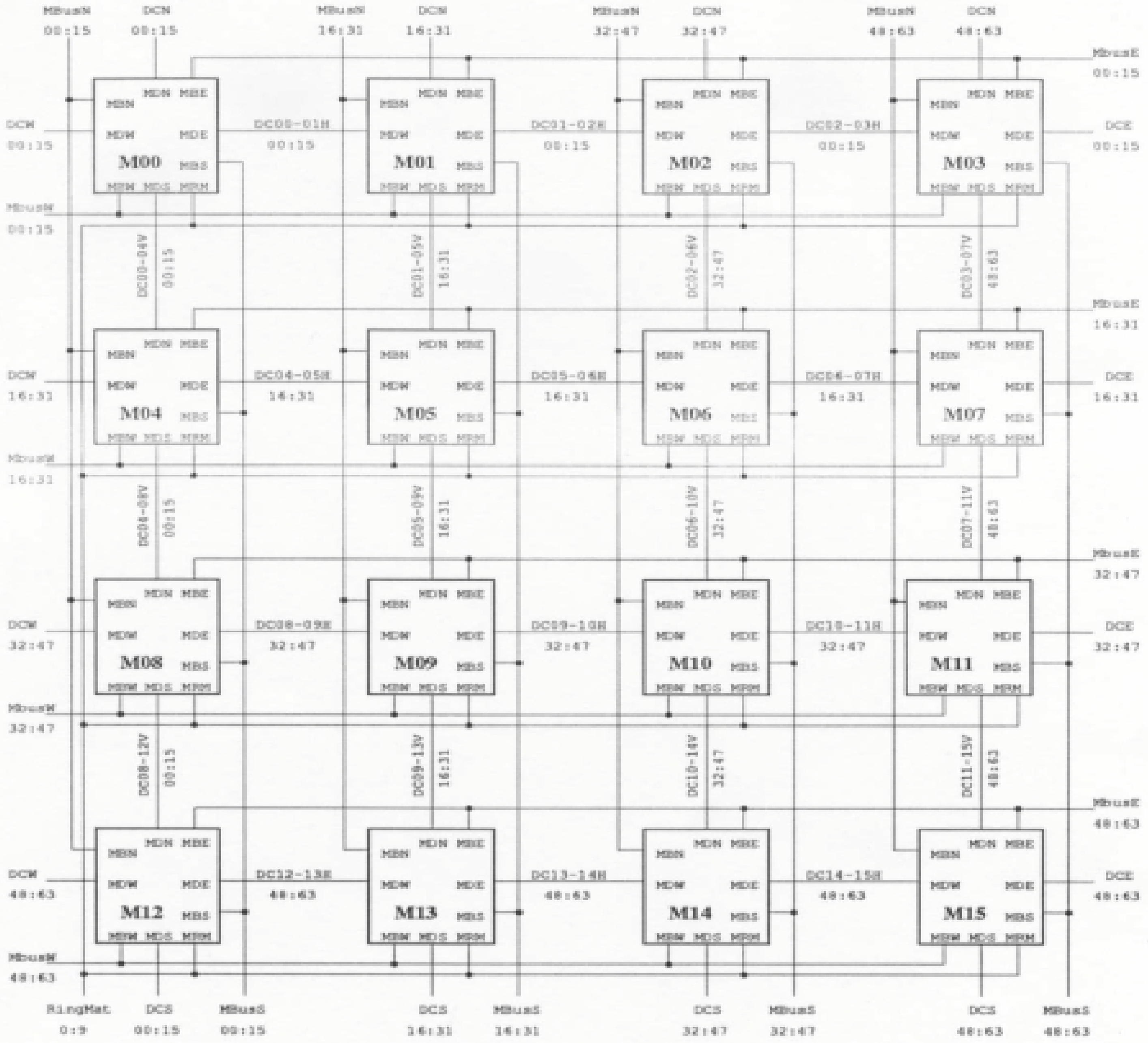


Figure 1: Perle-1 matrix

Matrix, buses, and connections

- DCN, DCE, DCS and DCW: North/East/South/West matrix side to connectors
- MDN, MDE, MDS and MDW: Matrix North/East/South/West direct connections
- MBN, MBE, MBS and MBW: North/East/South/West matrix buses

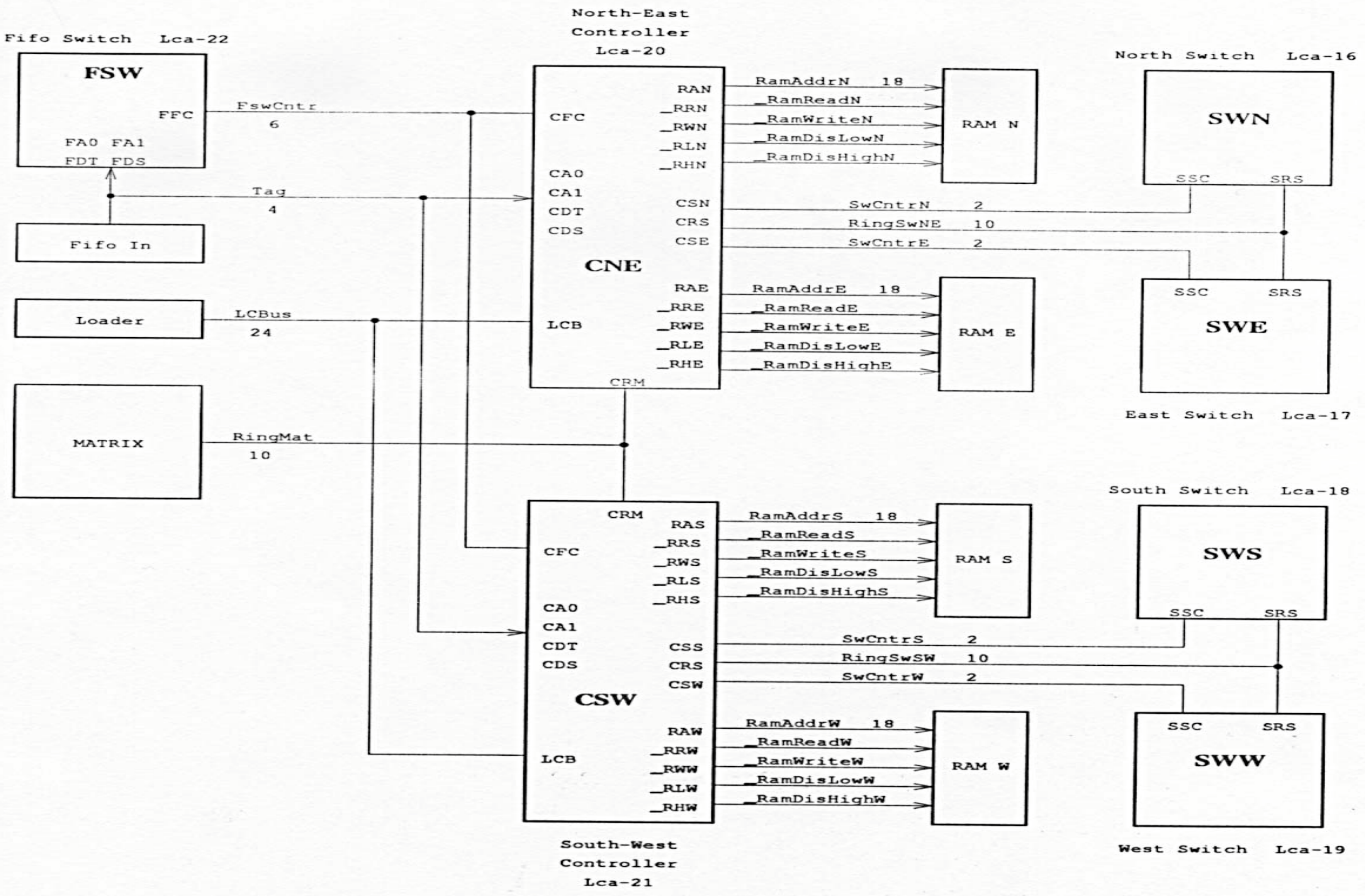


Figure 5: Perle-1 Control Wires

• How to Use DEC-PeRLe Board?

Requirements for proper installation and/or use of the PeRLe Board and software:

Hardware: a TURBO channel-based DEC station with a PeRLe-1 board;

Operating system: Ultrix version 4.2 or later;

Software: DEC C++ compiler; Xilinx development software for XC3000;

Disk space: >20MB.

Making Your Own Design

1. Design Partition

Map your design onto the FPGA chips according your design and the constraint of the PeRLe-1board. Some of the FPGA chips may not be used.

2. Design Entry

Describe the hardware part of the application, i.e., the PeRLe-1 configuration(s) involved.

with (1)Xilinx-supported schematic editor, or

(2) VHDL, or

(3) C+ + and PeRLe-1 library

Then synthesis your design with synthesis software to generate a XNF file.

3. Runtime Program

Design the program that will run on the host CPU and drive the hardware design described above.

4. Design Compilation

(1) Each of the resulting XNF files (one per LCA chip actually used in the PeRLe-1 board) must be passed through the standard Xilinx tools for technology mapping, placement, and routing, design rule checking and bitstream generation.

(2) the individual bitstream files(*.rbt*) must be converted into a PeRLe-1 downloadable configuration file(*.pl*).

5. Design verification

Run the design on a "representative" set of inputs, under control of it's driving program, in a variety of modes.

DEC-PERLE-1 BOARD FOR FAST PROTOTYPING

- Fast prototyping environment based on arrays of FGPAs.
- Digital's Paris Research Laboratory developed its third generation board, **DEC-PERLE-1** in 1992.
- The board is organized around a central computational matrix made up of 16 Xilinx XC3090 LCAs, surrounded by a four 1MB RAM banks, and 7 other LCAs to implement switching and controlling functions.
- We understand now difficulties that exist in creating Learning Hardware.
- The user has to understand well all programmable resources of the board, otherwise the logic design becomes non-mappable to FGPGA wiring resources.
- The designer needs to take into account this architecture from the very beginning of designing hardware rather than to design first and next try to map.

CONTROL

- Switches and I/O buses.
- **Control resource**.
  - MATRIX RINGS
  - RAM ADDRESS
  - RAM CONTROLS
  - SWITCH CONTROLS
  - FIFO CONTROLS
  - TAGS
  - CLOCK CONTROL
  - LCBus

CLOCK MODES

- Under software (the program running on the host) control, the clock generator may be put in the following operation modes:
- **STOP MODE**: No clock is generated in this mode.
- **FREE-RUN MODE**: This is the normal operating mode, where the clock continuously runs at the prescribed frequency.
- **BURST MODE**: This is a mode where, under software control, the clock generator will generate a burst of 1 to 31 clock ticks at the prescribed frequency, then stop. This is useful to implement step and double-step debugging modes.
- **AUTOSTOP MODE**: There are two autostop modes:
- **FifoIn-Autostop** and **FifoOut-Autostop**.
  - In the **FifoIn-Autostop** mode, **clock0** will automatically stop whenever the design attempts to read an empty input FIFO.
  - Similarly, in the **FifoOut-Autostop** mode, **clock0** will automatically stop whenever the design attempts to write a full output FIFO.
  - These two modes can be enabled at the same time.
- For instance, the CCM design runs in this mode.

CLOCK MODES

- **CLOCK1-DIV2** : This mode is useful for very high performance designs. **Clock1** runs at half the speed of **clcok0**. This allows the RAM and FIFOs to be operated on half the speed of the matrix.
- **clock0 stop**.The clock0 may stop under control of the application on the board. This is usually used to implement **flow-control**, where the entire datapath is stopped waiting for input data (when the input FIFO is empty) or output space (when the output FIFO is full). It is much more efficiently and easily implemented this way than through the global distribution of a clock enable signal. In effect, when application runs entirely on **clock0** and both autostop modes are enabled, the application can be seen as a perfect synchronous system without flow-control concern. The **clock0** signal will stop under one or more of the following conditions:
  - (1)The active-low $\overline{ClkStop signal is asserted from one of the controllers.
  - (2) In the **FifoIn-autostop** mode, the input FIFO is empty and the active-low $\overline{FifoInRead}$ signal is asserted from one of the controllers.
  - (3) In the **FifoOut-autostop** mode, the output FIFO is full and the active-low $\overline{**FifoOutWrite**}$ signal is asserted from one of the controllers. The memory subsystem and the FIFOs are clocked by **clock1** . This means that it is still possible to perform memory and/or FIFO operations even when **clock0** is stopped.

DEC-PERLE Mode of Operation

- **Slow mode**. Under control of an application on the board, it is possible to slow down the clock (divide its frequency by 4) by asserting the active-low NOT{**ClkSlow}** signal from one of the controllers.
- This is useful when an application can run at a very high speed, but must infrequently perform an operation that is impossible to be performed at the high speed (like stopping the clock, or accessing the FIFOs).
- The NOT{**ClkSlow}** can be asserted at any speed, but its operation is asynchronous, that is, it will take an unpredictable number of cycles for it to be effective.
- If the operation frequency is less than 80 MHz, this number of cycles is however guaranteed to be less than or equal to 6.

- **Host interface**.

The DEC-PERLE-1 application is running under the control of the software program executed on the host computer.

The communication between DEC-PERLE-1 application and its driving software program can be done through FIFOs or LCBus.

FIFOs. There is a 32-bit-wide, 512-word-deep FIFO in each direction

- These FIFOs are called **input FIFO** for the Host-to-PAM direction and **output FIFO** for the PAM-to-Host direction, respectively.
  - On the application side, their data wires are connected to the **Fifo Switch** LCA and their control wires to the two Controller LCAs.
- Both FIFOs are purely synchronous devices when operated from the application side.
  - They appear to be always available for reading or writing in **autostop** mode.
- The input FIFO and output FIFO are synchronous devices that offer two active-low status signals
  - NOT{**FifoInEmpty**} and NOT{**FifoOutFull**} and
  - two active-low command signals NOT{**FifoInRead}** and NOT{**FifoOutWrite**}.
- These four signals are connected to the two Controller LCAs CNE and CSW.
- The input FIFO can be written and the output FIFO can be read by the driving software through the runtime library.

LCBUS

- The LCBus is a 24-bit-wide general purpose register that can be read and written by both the software and the application design.
- The LCBus can be used for asynchronous communication between the Controller LCAs and the software program.
- Under the software control, the direction of each bit can be set independently of the others.
- Initially (after download), all bits are set for PAM-to-Host communication.

Tags.

•Every word that the software (the program running on the host) pushes into the input FIFO is ``**tagged**'' with 4-bit value.

These tag bits are read from the input FIFO at the same time as the data word, and are available on both Controller LCAs and on the **Fifo Switch**.

Timing of DEC PERLE

- The user of the board has to know the delays of different kinds of connections, so that he can make reasonable trade-off decisions for his designs.
- For instance, the delay of matrix rings is 43ns, and the delay of matrix direct connection is 24ns. For a given signal, if the designer can use either the matrix rings or the matrix direct connection, then the matrix direct connection should be a better choice.
- It would be very difficult to have a GA make good timing decisions.
- The above described hardware resources have been created for a class of applications, so they are not necessarily optimal for any particular application. The very useful features in designs are: large memories, vertical and horizontal buses and direct connections, global connections, clock control modes and debugging modes. However, the designer is often confronted with too few connections in FPGA resources to map his virtual architecture.
- This requires frequent modifications, or may require a total redesign.
- The most difficult are architectures as CCM, which have many buses and many global signals between control units and data paths.

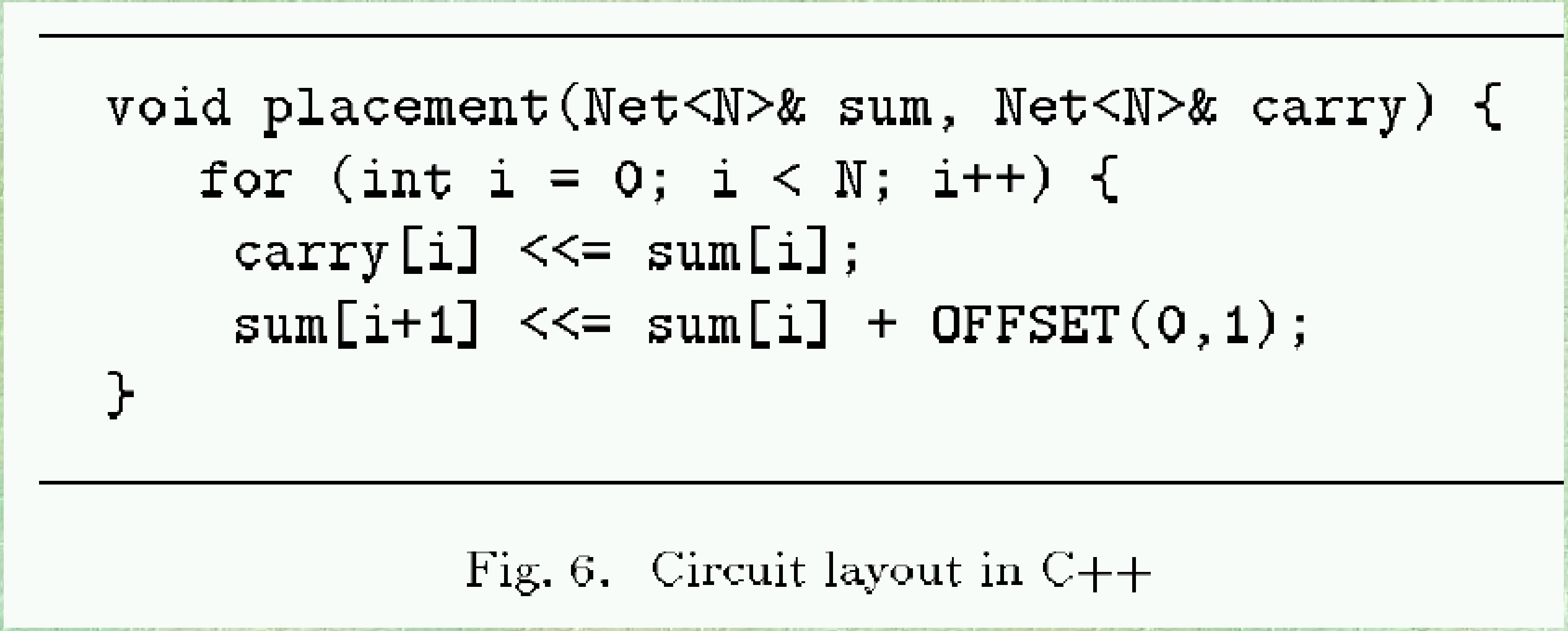
Concluding on DEC PERLE properties

- Concluding, DEC-PERLE-1 board, similarly to other FPGA boards, advocates very regular design styles without long and many control signals.
- It is then good for small SIMD processors, pipelining, systolic processors, cellular automata or complex Boolean functions.
- The basic design principle is: ``**map two-dimensional tables to two-dimensional logic resource** arrays''
- The design can be developed incrementally thanks to its easy memory access, host interface with FIFOs, and the clock debugging modes and tags.

PROGRAMMING THE DEC-PERLE-1 BOARD

```
template<int N>
struct RippleAdder: Block {
  RippleAdder(): Block("RippleAdder"){}
  void logic(Net<N>& a, Net<N>& b, Net<N>& c,
             Net<N>& sum, Net& carry) {
    input(a); input(b); input(c);
    output(sum); output(carry);
    for (int i = 0; i < N; i++){
      sum[i] = a[i] ^ b[i] ^ c[i];
      carry[i] = (a[i] & b[i])
               | (b[i] & c[i])
               | (c[i] & a[i]);
    }
  }
};
```

Fig. 5. Circuit description in C++



```
void placement(Net<N>& sum, Net<N>& carry) {
   for (int i = 0; i < N; i++) {
    carry[i] <<= sum[i];
    sum[i+1] <<= sum[i] + OFFSET(0,1);
}
```

Fig. 6. Circuit layout in C++

PROGRAMMING THE DEC-PERLE-1 BOARD

- For using DEC-PERLE-1 board, we must run an application-specific program on the host computer which connects to the DEC-PERLE-1 board.
- On the other hand, the 23 FPGA chips of the DEC-PERLE-1 must be programmed to realize an application-specific hardware.
- Therefore, A DEC-PERLE-1 program consists of two parts:
  - the **driving program** which runs on the host and controls the DEC-PERLE-1 hardware.
  - A 1.5 MB **bitstream** which programs the 23 XC3090 FPGAs of the DEC-PERLE-1 to realize an **application-specific hardware**

The Driving Program

The driving program is written in C or C++ and is linked to the runtime library encapsulating a device driver.

The requirement for developing the driving program is the C or C++ programming environment and the DEC-PERLE-1 runtime library.

The runtime library.

- The runtime library of DEC-PERLE-1 is essential to the developer who develops the **driving program** which runs on the host computer and controls the DEC-PERLE-1 hardware for the application.
- The runtime library is the only way to access DEC-PERLE-1 hardware for the driving program.
- The runtime library developed by DEC's Paris Research Laboratory provided a few essential controls to the application driving program:
  - (1) A UNIX I/O interface, with open, close, read and write.
  - (2) Download the configuration bitstreams from host to DEC-PERLE-1, and/or read back the values of all the flip-flops of all the LCAs.
  - (3) Read/write static RAM on DEC-PERLE-1 by the software program.
  - (4) Control the mode and speed of DEC-PERLE-1 clock by the software program

SOFTWARE DESIGN STEPS

- For generating 1.5MB bitstream that programs the XC3090 FPGAs to realize the application-specific hardware, the following steps are involved:
- **Design Partition**
  - In this step the design is mapped onto 23 FPGA chips according to the logic design and the constraints of the DEC-PERLE-1 board.
  - Some of the FPGA chips may be not used.
  - For example, the CCM design uses only 17 FGPA chips of all 23 chips, because we were not able to find better mapping despite many efforts.
  - The steps 2 and 3 should be carried out separately for each FPGA chip that is used in the design.

Design Entry

- In this step, the design is created for each FPGA used in the design separately.
- This step produces **Xilinx netlist file** (XNF file) for the next step.

There are three kinds of design entry methods:

- (1) **Schematic editor** to create the XNF file
- (2) **Hardware description language**: the designer can use VHDL (or other hardware description language) to create the design, then the synthesis software is used to synthesize and optimize the design and produce the XNF file
- (3) **PerleDC library**. Another possible way is to use a C++ program and the PerleDC library to describe the design. Individual configuration of each FPGAs involved in your design are described by this C++ program. Compiling and running this C++ program generates the XNF file of the design. There are many tools that can be used. For instance, there are four sets of tools available at EE of PSU as of this writing: Xilinx Foundation Series, OrCAD Express 7.0, Mentor's Leonardo, Summit. Both Xilinx Foundation Series and OrCAD Express 7.0 support schematic editor and hardware description language.
- **Design Implementation**.
- **Map, place**} and **route** your design, and finally **generate the bitstream file** by using Xilinx development tools. Since all FPGAs used on DEC DEC-PERLE-1 board are XC3090 FPGAs, the user needs Xilinx development tools that support XC3090 FPGA.
- **Design Verification**. At this step, the bitstream generated at the previous steps is downloaded into the DEC-PERLE-1 board and the design is tested. If something goes wrong, you may need to modify your design at design entry step, then regenerate the bitstream file, download it to DEC-PERLE-1 board and test your design again

Examples of Applications

In what follows, $a \div b$ denotes the quotient and $a \cdot|\cdot b$ the remainder in the euclidean integer division of a by b.

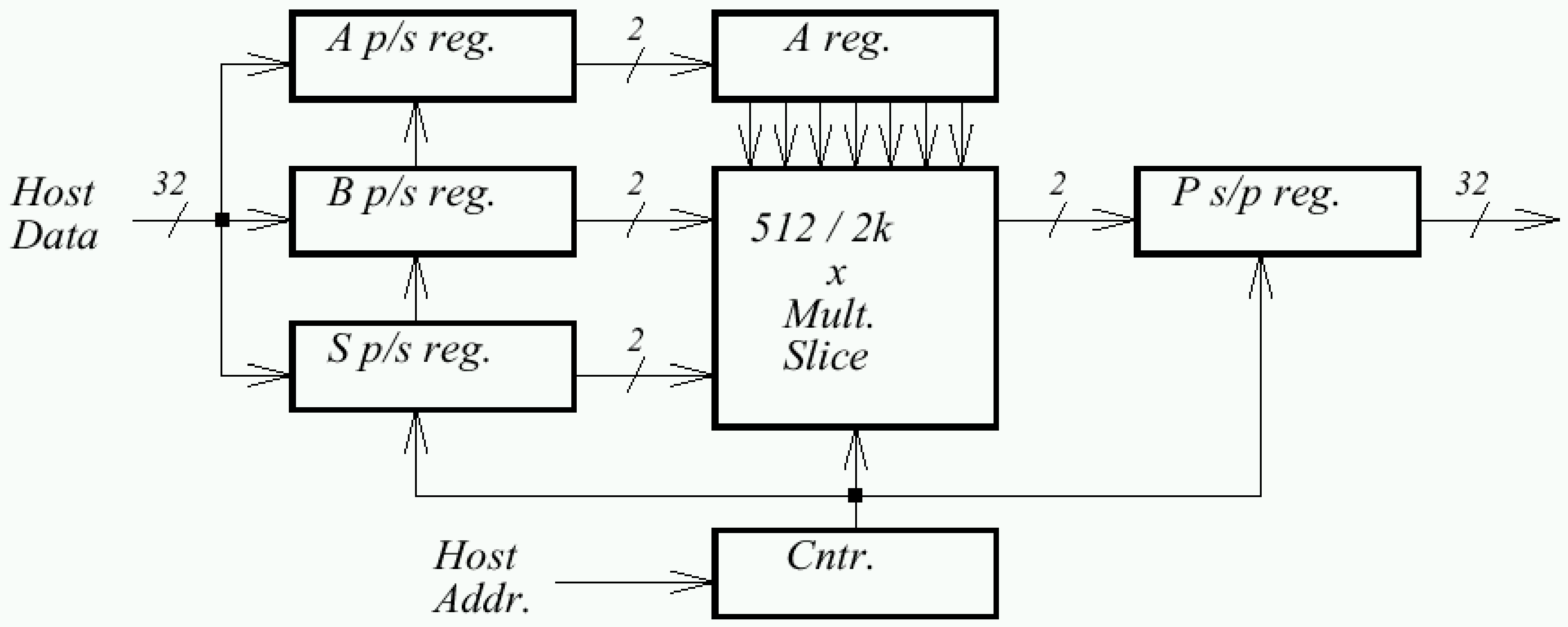


Fig. 7. Long multiplication

B. RSA cryptography

To investigate further the tradeoffs in our hybrid hardware and software system, we have focused on the RSA cryptosystem [23]. Both encryption and decryption involve computing modular exponentials, which can be decomposed as sequences of long modular multiplications, with operand sizes ranging from 256 bits to 1k bits.

Starting from the general-purpose multiplier above, we have implemented a series of systems spanning two orders of magnitude in performance, over three years.

Our first system [18] uses three differently programmed Perle-0 boards, all operating in parallel with the host. At 200 kb/s decoding speed, this was faster than *all* existing 512-bit RSA implementations, regardless of technology, in 1990. A survey by E. Brickell [24] grants the previous speed record for 512-bit key RSA decryption to an ASIC from AT&T, at 19 kb/s.

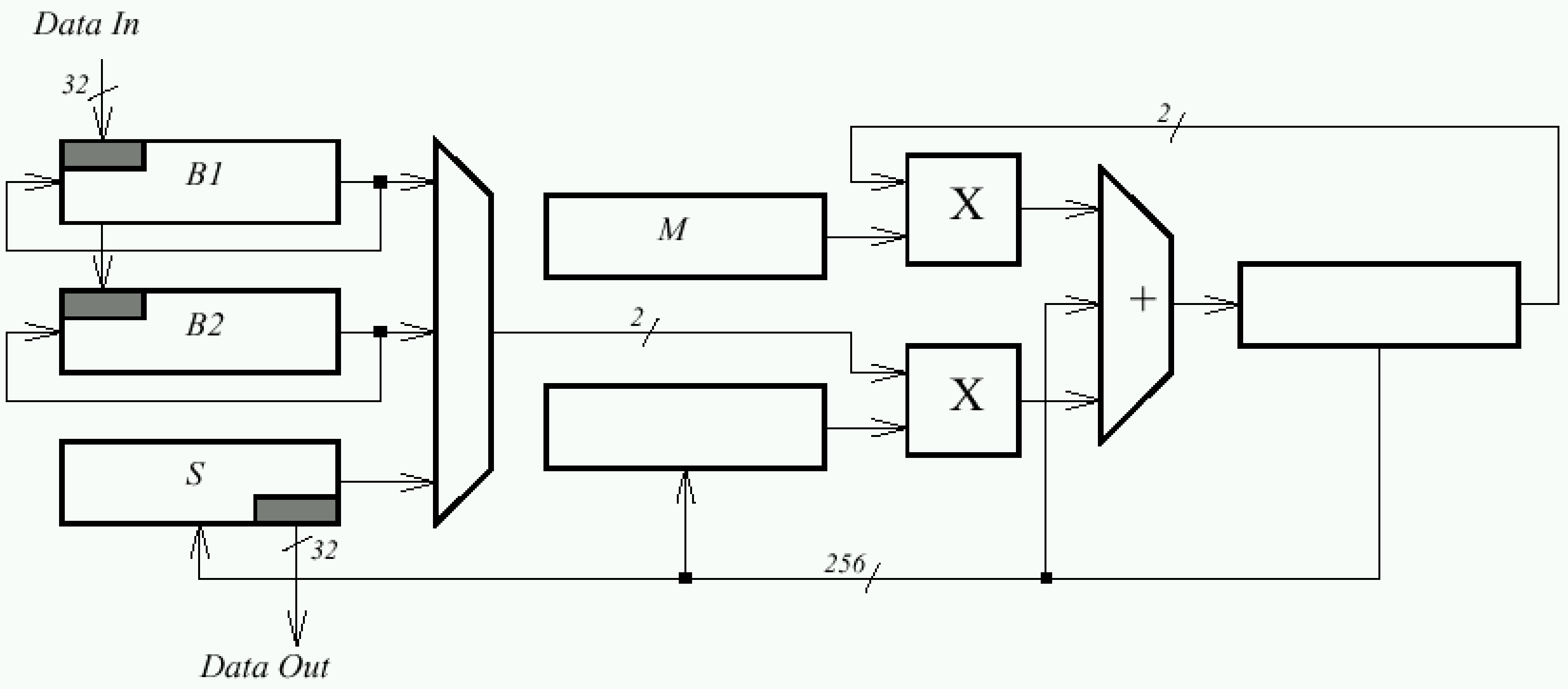


Fig. 8. RSA cryptography

C. *Molecular biology*

Given an alphabet $\mathcal{A} = (a_1, \ldots, a_n)$, a probability $(S_{ij})_{i,j=1\ldots n}$ of substitution of a_i by a_j, and a probability $(I_i)_{i=1\ldots n}$ (resp. $(D_i)_{i=1\ldots n}$) of insertion (resp. deletion) of a_i, one can use a classical dynamic programming algorithm to compute the probability of transforming a word w_1 over $\mathcal{A}$ into another one w_2; this defines a *distance* between words in $\mathcal{A}$.

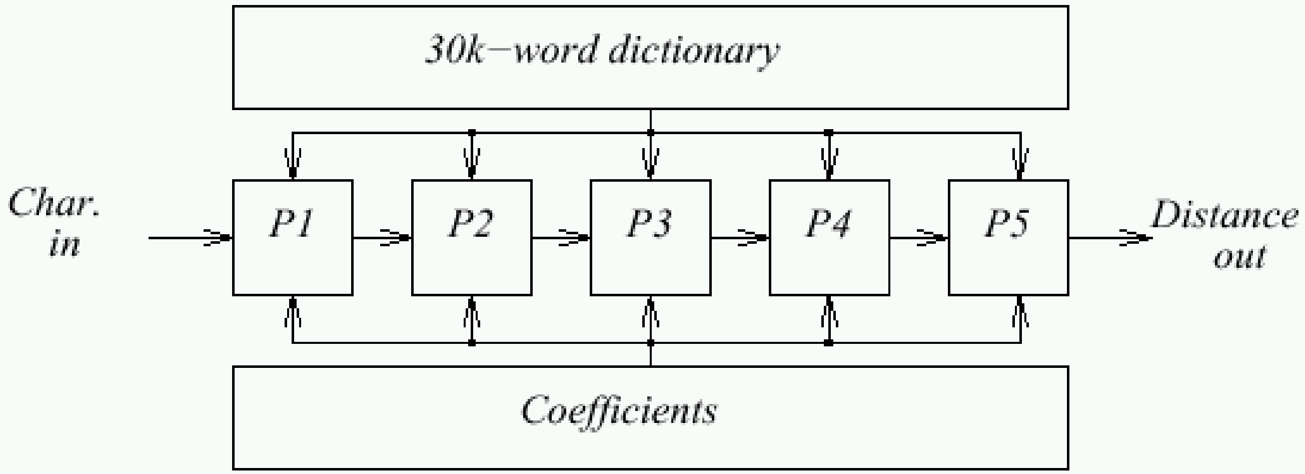


Fig. 9. String matching

Applications include automated mail sorting through OCR scanners, on-the-fly keyboard spelling corrections, and DNA sequencing in biology.

D. Heat and Laplace equations

Solving the heat and Laplace equations has numerous applications in mechanics, integrated circuit technology, fluid dynamics, electrostatics, optics and finance [27].

The classical *finite difference method* [28] provides computational solutions to the heat and Laplace equations. Vuillemin [29] shows how to speed-up this computation with help from special-purpose hardware. A first implementation of the method on P_1, by Vuillemin and Rocheteau [29], operates with a pipeline depth of 128 operators. Each operator computes $\frac{T+T'}{2}$, where T and T' are 24-bit temperatures.

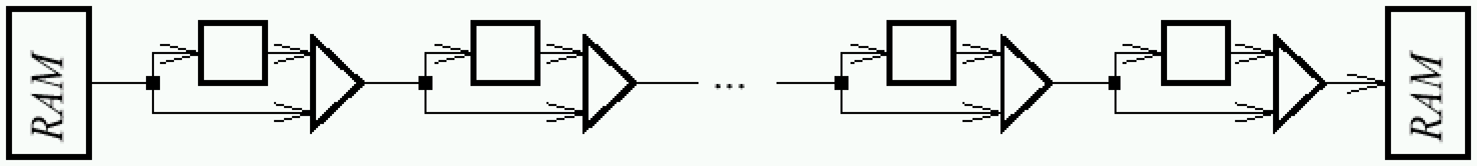


Fig. 10. Heat and Laplace equations

- At 20 MHz, this first design processes 5G operations| add and shift|per second.
- For such a smooth problem, one can easily show that fixed-point yields the same results as floating-point operations.
- The performance achieved by this first 24-bit P1 design thus exceeds those reported by McBryan et al. [30] [31], for solving the same problems with the help of supercomputers.
- A sequential computer must execute 20 billion instructions per second in order to reproduce the same computation.

$$E(\vec{N}) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} w_{i,j} N_i N_j$$

where $\vec{N} = (N_0, \ldots, N_{n-1})$ is a vector of binary variables and $(w_{i,j})_{0 \leq i,j < n}$ is a fixed matrix of *weights*. It is typically used to find approximate solutions to some $\mathcal{NP}$-hard problems, such as graph partitioning and circuit placement.

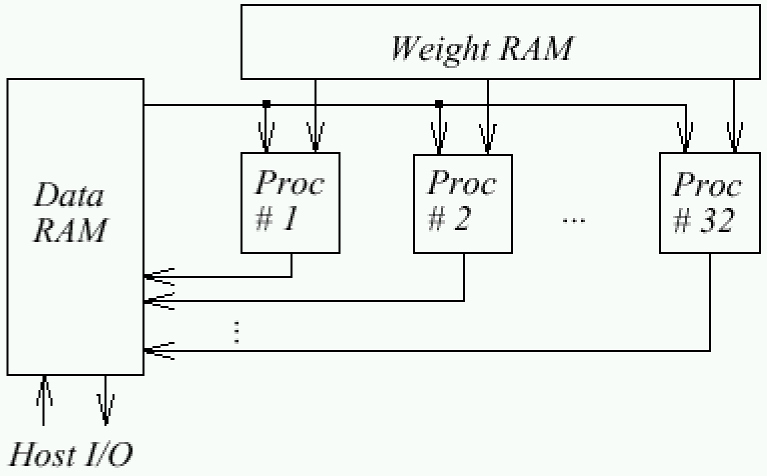


Fig. 11. Boltzmann machine

The latest P_1 realization solves problems with 1400 binary variables, using 16-bit weights, for a total computing power of 500 *megasynapses per second*. (The megasynapse is the traditional unit used in this field: it amounts to one million additions and multiplications by small coefficients.)

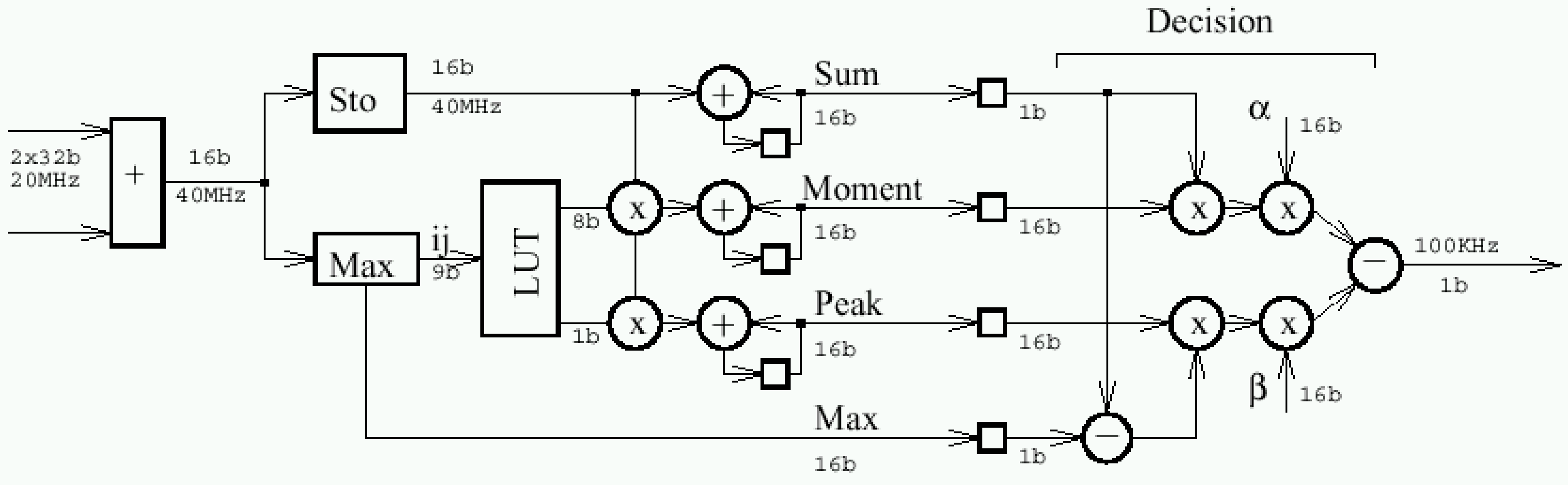


Fig. 13. Calorimeter datapath

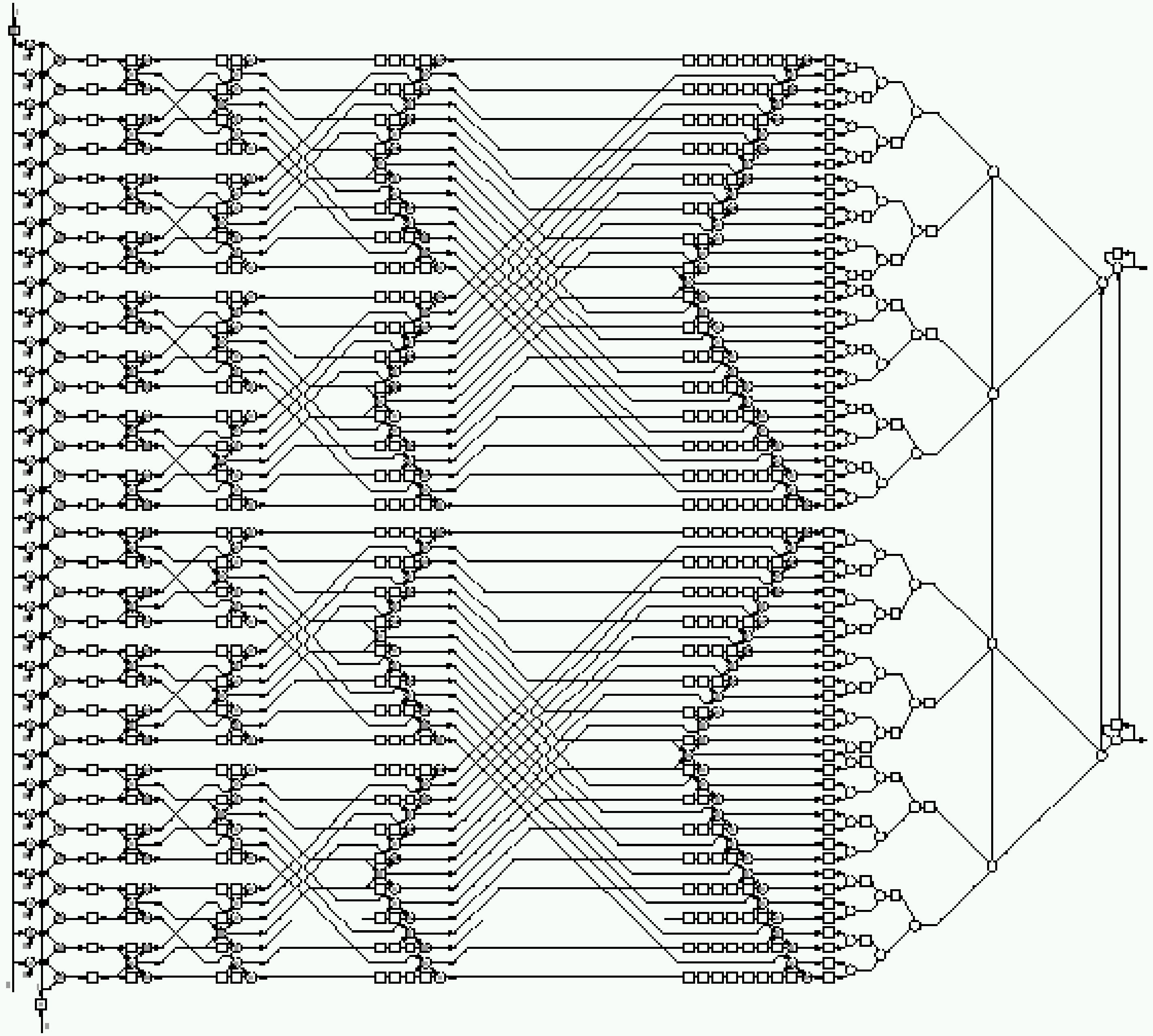


Fig. 14. Fast Hough transform

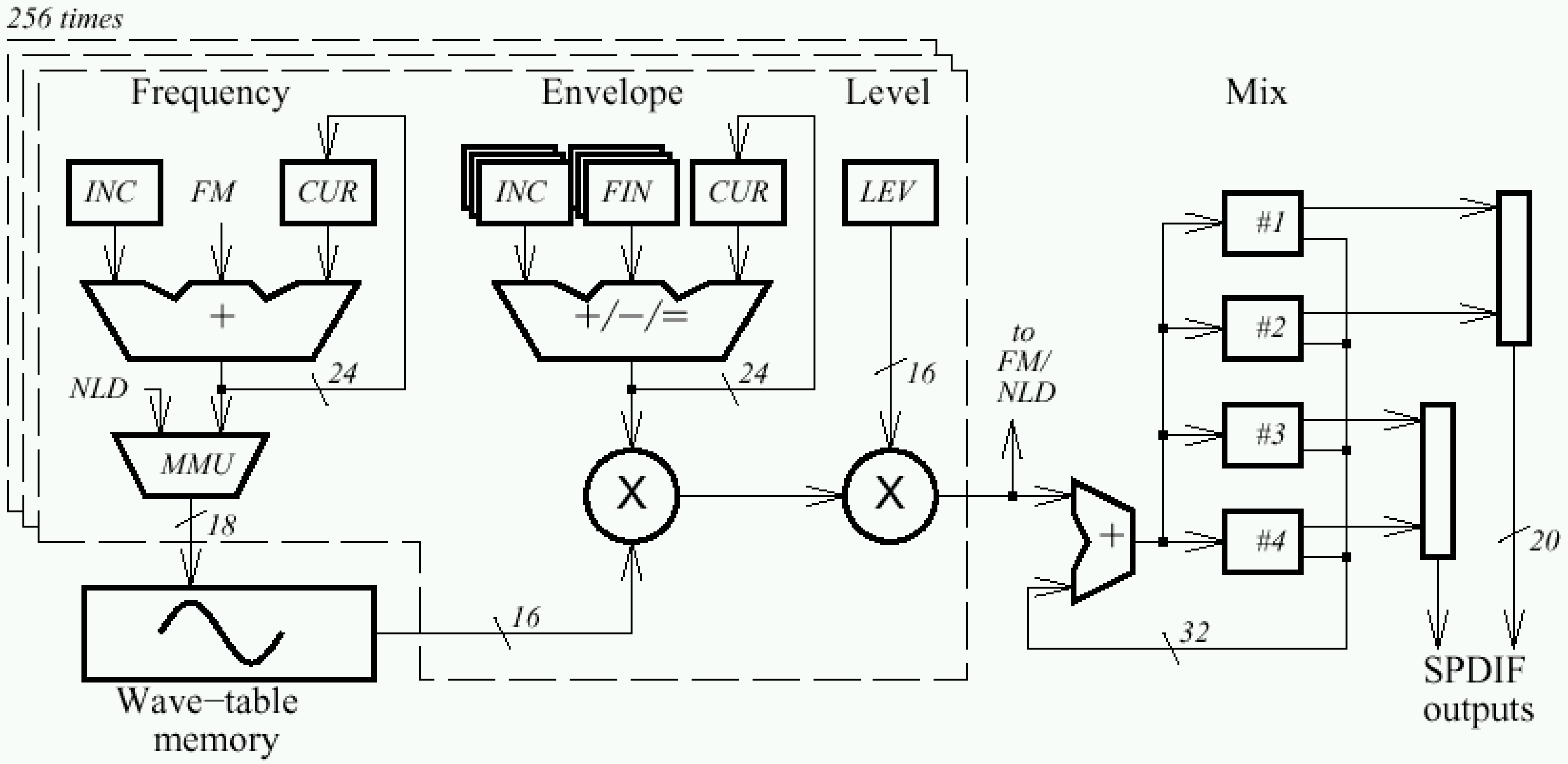


Fig. 15. Sound synthesizer

CONCLUSIONS

- Principles of Learning Hardware as a competing approach to Evolvable Hardware, and also as its generalization.
- Arithmetic Operations, Image Processing, Data Mining machines.
- DEC-PERLE-1 is a good medium to prototype such machines, its XC3090A chip is now obsolete.
- This can be much improved by using XC4085XL FPGA and redesigning the board.
- Massively parallel architectures such as CBM (DeGaris, Korkin, Buller) based on new Xilinx series 6000 chips will allow even higher speedups.

Questions

- Show how to use Xilinx to implement both PAM models described above: switchbox and logic.
- Show how to map any of the examples to actual structure of DEC-Perle board. Describe precisely mapping to CLBs and pin assignments to pins of Xilinx chips in the structure.

Sources

Literature: Vuillemin

Students: Jinshan Huo

David Foote

Qihong Chen

Instructor: Dr. Marek Perkowski