Evolution of CAD tools towards third generation custom VLSI design

H. de Man

Classification
Physics Abstracts
85.40

Evolution of CAD tools towards third generation custom VLSI design

H. De Man

IMEC, Kapeldreef 75, 3030 Leuven, Belgium

(*Reçu le 28 juin 1985, accepté le 9 juin 1986*)

Résumé. — On discute ici les tendances dans la CAD des circuits aux applications spécifiques (ASIC). La rareté des concepteurs chevronnés ainsi que d'autres contraintes conduisent à une stratégie de conception qui tend à séparer la conception « système » de la conception « silicium ». Les concepteurs de systèmes utiliseront systèmes du type « intelligence artificielle » qui travaillent à partir d'un langage spécifique. Les architectures seront basées sur des modules réutilisables et paramétrables. Un programme symbolique et interprétatif doit supporter la génération du « layout » ainsi que le test et la simulation temporelle. Les stations de travail seront du type microprocesseur qui allient les styles de programmation du type symbolique et déclaratif. La formation des ingénieurs devra alors se faire dans le sens d'enseigner les langages destinés à produire les circuits électroniques.

Abstract. — In this paper trends in CAD for application specific IC's (ASIC) are discussed. Shortage of skilled silicon designers, too long time to market and too low level of design as in standard cells and gate arrays, lead to a design strategy whereby system design is strictly separated from silicon design. (Meet-in-the-middle design). System designers will use interactive, knowledge based synthesis tools adressing a number of well defined target architectures to be generated from a formal specification language. Architectures are defined as a connection of a well defined set of reusable and parameterizable modules which are predesigned by silicon specialists. This is no longer done on a CALMA type environment but on an interpretative symbolic programming environment. This environment supports automatic parameterization and generation of layout, timing and testing views as well as automatic adaptability to new technology rules. Verification will be shifting away from costly simulation to knowledge based verification, based on a formal definition of design styles and automatic theorem proving. This will require multiprocessor workstations unifying high speed graphics and imperative, declarative and symbolic programming styles. A major problem with this methodology will be the (re)education of design engineers in order to design hardware the « soft » way.

Introduction.

Custom designed IC's or application specific IC's (ASIC's) will cover half of the IC designs by 1990 and 40 % will be designed by the end users. In this paper we will derive the characteristics of quick turnaround design from todays gate-array and standard cell designs. It is shown that the key to success is an intelligent separation between system and silicon design at the level of parameterizable MSI and LSI functions. This is called the *meet-in the middle* design strategy and leads to the concept of silicon compilation.

Today true silicon compilers, including synthesis and test techniques, are only in a research phase. Silicon module design is supported already by so-called module generators. Module generators are software environments and will be only successful if silicon designers get used to the fact that silicon design becomes increasingly a matter of programming of design knowledge. System designers on the other hand will be confronted with formal specification languages as well as the use of interactive graphics for floorplanning.

Finally, verification tools such as simulation at all levels are no longer realistic nor reliable in a VLSI environment.

New techniques based on rule based expert systems will evolve gradually towards *proof of correctness* systems. Re-use of scarce design expertise, especially at the sub-micron silicon level, must be cought in expert systems to be usable by less experienced designers and it is therefore expected that, when prices of AI workstations drop to an acceptable level in ca. 3...4 years, their use will proliferate very rapidly.

Their efficient use however will strongly depend on urgently needed research to formalize design at all levels such that rule formulation can be done in an efficient way. A large scale joint effort of computer science and electrical engineering academia confronted with industrial experience is of prime importance for success. The role of universities in education and

reeducation is of prime importance to succeed. Additional funding is required whereby perhaps also industry should invest universities to get the people taylored in their needs.

Whether all of this will also be of economic impact in Europe depends on the broadness of scope of industrial and financial institutions to invest in adventurous small start-ups based on bright ideas of excellent research teams. The time is ripe for it.

1. CAD tools for gate-arrays and standard cell custom IC's.

In this paper we investigate the evolution of CAD tools for custom design of so-called application specific IC's or ASIC's.

Until today most ASIC's are only substitutes for control logic structures on PCB's. Tools to design so-called VLSI systems on a chip, exploiting massive parallelism to obtain high data throughput are only just now appearing. This evolution will be addressed.

A recent study points to a big market for such ASIC's but, unfortunately, 80 % of it requires production volumes below 5 000 units and is to be used in systems with less than 5 years lifecycle. Therefore it is mandatory that ASIC's are designable by the mass of traditional system designers rather than the few silicon designers available today.

The full custom design style with its very large design spectrum from system to transistor level is clearly unacceptable since it is error prone, requires too much design time and can only be done by silicon engineers. On the other hand, actual design of ASIC's has gone through two mature stages or generations : the gate arrays and the standard cell design styles. We will first learn from these and then try to generalise their good qualities and try to solve some of their drawbacks to go towards the so-called ASIC's of third generation.

The principle of gate arrays and standard cells [1] is well known. It is characterized by the fact that the silicon part is either partly preprocessed (gate arrays) or prestored in a software library (standard cells). The designer specifies his (her) logic circuit as an interconnection of standard TTL (CMOS) catalogue parts on an hierarchical schematics editor using a graphics terminal (Fig. 1). These are available to the system designer from the many successful workstation vendors (Daisy, Valid, Mentor, Silvar-Lisco, Venus...). These workstations (WS) have software which automatically expands hierarchy and maps the expansion into a netlist of primitive silicon cells which are thoroughly proven, debugged and stored in the (often) closed database of the CAD system. This database contains all consistent « views » of these cells and is the link, at the gate level, between the foundry (contracted by the workstation) and the logic designer. As each cell has a Boolean function view, a complete logic simulation model is produced which, in todays system, can be of 10 000 gate complexity. Such simulations do take too much computer time on 32 bit 1...5 MIPS workstations. Therefore most workstations now contain so-called logic simulation *accelerators* which are in fact microcoded processors executing the logic simulation scheduling algorithms. These lead to simulation speeds which are two to three orders of magnitude above software implementations [2]. These simulators form a valid alternative to traditional breadboarding techniques.

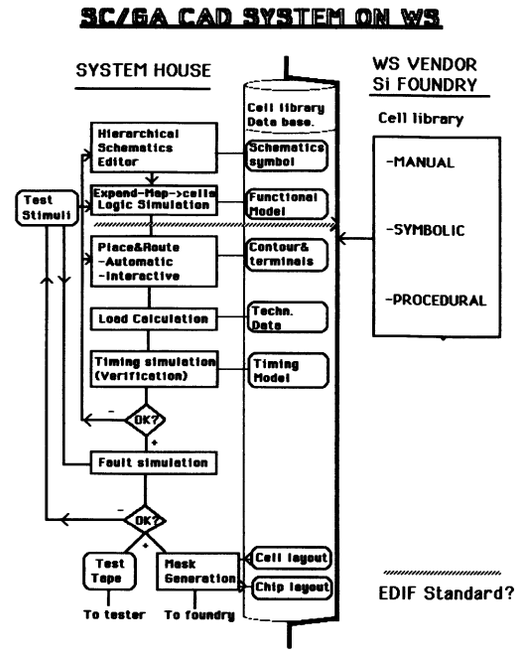


Fig. 1.

Schematics editing, expansion, mapping and logic simulation are now usually called *CAE* (Computer Aided Enginnering) or *front end CAD. Today, in most cases, the involvement of the logic designer stops here.* Netlists, more and more described in the EDIF standard (Electronic Design Exchange Format [3]), are now transferred to the WS vendor (foundry) and from there on the so-called *physical design* starts.

This is the automated placement and routing of cells followed by interconnection parasitic extraction to build a timing model for the circuit. In the past, timing verification has been done using *timing simulation* [4]. It is however difficult to make sure that the timing test pattern finds a valid worst case timing path. Therefore this is more and more substituted by graph based *critical timing path analysis* [5-7]. This insures correctness and is much faster than simulation. *It illustrates the trend away from simulation towards expert analysis systems* (see paragraph 3.4).

When timing is approved, test patterns have to be generated. Today, this is the weakest part of WS systems. Usually exhaustive fault simulation is done to verify fault coverage of the user defined functional test patterns. This is a very costly operation for which increased use is made of so-called hardware im-

plemented fault simulators. It is this authors opinion that instead more clever self-test [9], scan-path [10] or expert systems to support testability [11] should be used to alleviate this problem. Finally a test-tape as well as a pattern generation tape are generated for the silicon foundry.

The important things to take over from these systems for use in the third generation ASIC's are :

P1 : A complete split between silicon and logic design ;
P2 : A re-use of proven silicon (just as a software re-use of proven procedures) ;
P3 : Accessible by the mass of 250 000 system designers as opposed to the 2 500 silicon specialists [8].

The latter market, only tapped for 2 % by the WS vendors today, is the key to their success [8].

However, although todays standard cell and gate array systems have reached the 10 000 gate level (especially in Japan), there remain serious deficiencies which can to be summarized as follows :

C1 : Silicon density is too low. Experiments [12] indicate differences of a factor 2...3 with respect to manual design mainly because of lack of the exploitation of array structure in logic such as data-paths, ROM, RAM, systolic arrays, etc... [13].
C2 : The gate level is a too low design level. The WS systems support automated layout and re-usable cell design but they neglect the time consuming synthesis of a system in terms of logic gates [14].
C3 : The investment in a cell library is large and binds to a particular technology (foundry). Often when the design is completed the next generation of technology arrives. There is a need to obtain automated *adaptation* of cell libraries to other design rules.
C4 : A lot of effort is put into digital design systems but there is a niche in the market of analog design which is barely addressed yet.

2. Evolution towards true silicon compilation.

Fortunately, there is a lot of effort lately to extend the above methods towards « third generation ASIC's » and to solve the above problems. Third generation ASIC's are characterized by a design methodology which this author characterizes as : *Meet-in the middle design* strategy. This concept is illustrated in figure 2. Instead of designing at the gate level, the system designer decomposes his (her) system specification into so-called Functional Building Blocks (FBB) which are MSI, LSI modules such as data-paths, counters, registers, PLA, RAM, ROM and even, recently, complete microcontroller cores such as in the PLEX [15] system. The layout outline of these modules as well as their terminal position is then used for silicon assembly much in the way a PCB is designed.

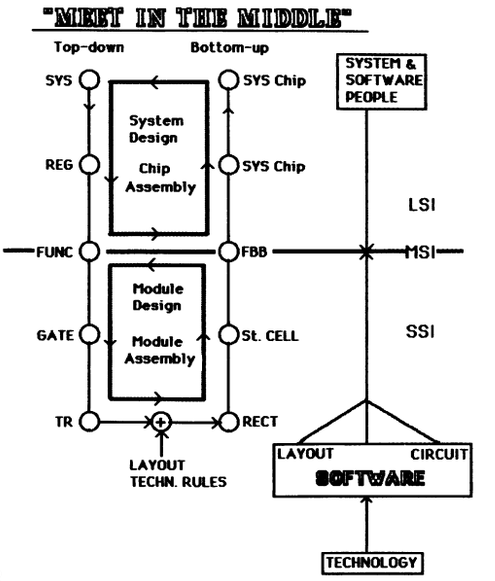


Fig. 2.

The problem is now mainly at the silicon side since FBB's are of much greater variability than logic gates. Therefore the *re-usable* FBB's must be *adaptable* to the system requirements so they have to be *generated* by software *procedures* rather than be fixed geometries such as standard cells. These are called *parameterizable modules* or *paracells* [16]. This leads to the remarkable fact that *silicon design becomes increasingly intermixed with programming which causes a reeducation problem for traditional silicon designers* (see Sect. 3).

When the *flexible structured module generation* is combined with automated synthesis techniques to decompose a formal system description into an interconnection of FBB's and these FBB's are automatically placed and routed for a requested performance we can talk about *silicon compilation.* Indeed, *only in this context we have a true analogy with software compilation of a high level language into machine code in computer-science.*

Figure 3 shows the three essential parts of what this author would categorize as a *true silicon compiler.*

The line in the middle is the *meet-in the middle* line.

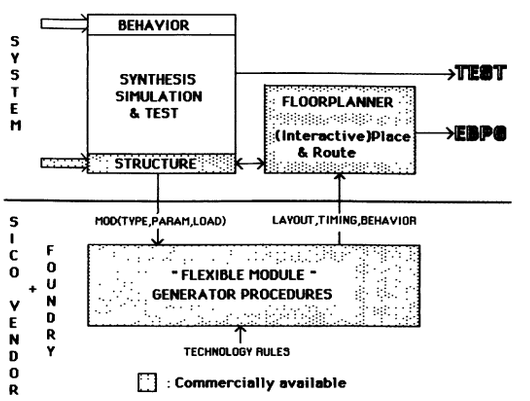


Fig. 3. — Silicon compiler parts.

2.1 THE SYNTHESIS PART OR COMPILER. — The system designer ideally uses a *synthesis* program which is the true compilation part as well as a *floorplanner* tool to assemble the layout from a call, below the middle, to the re-usable *module generator procedures* designed by the silicon designers. We now discuss this in more detail.

The system designer ideally enters his (her) design at the *behavioural* level by describing it into a *system specification language.*

Most of these languages are in full development.

They are mostly PASCAL, ADA or LISP like languages completed with timing expressions and ways to express parallelism.

Recently, there is a trend to opt for *applicative* or *functional* languages, such as the SILAGE [18] language. These languages are non-procedural i.e. they express a set of *simultaneous* single assignment equations and can be *compilated* easily into data-flow graphs and Petri-nets [19]. These compilers can be given hints by means of pragmas (such as in ADA) in order to hint a particular target architecture to the compiler (e.g. parallel- or single processor, bit-parallel, bit-serial, ...). Many approaches are presently used in a research phase to build these silicon compilers. These can roughly be divided into *algorithmic* and *knowledge based* techniques [20, 21] with a mixture of the two probably being the correct approach.

Notice that, ideally spoken, if the rest of the silicon compilation is *correct by construction*, simulation is only necessary at the behavioural level to verify correctness. This simulation is much more effective than at the gate level and allows for an embedding of the chip design in a multichip system description.

For the time being this high level synthesis is only in a research phase and we will come back to it in section 4. It is certainly the area where we will see impressive breakthroughs in the next few years.

Today the system designer usually is doing only a manual (error prone) synthesis, possibly using functional and register transfer simulation to insure functional correctness. The synthesis is done in terms of the *flexible modules* to be described next.

2.2 THE MODULE GENERATOR PROCEDURES. — Re-usable flexible modules are provided as software procedures by the silicon compiler vendor for a given foundry technology. Based on the module specifications by actual parameter assignment from the system designer, the module generator produces four representations of the requested module instance :

1. A bounding box with terminal position intervals for floorplanning.

2. A detailed layout for mask pattern generation.

3. A timing model computed for the instance and parameterized for fan-out and wiring load.

4. A functional model for logic simulation (if no synthesis has been used).

The timing model allows for a detailed timing analysis after the floorplanning phase. This leads potentially to floorplan modification, buffer adaptation or, in the worst case, a change in the architectural description.

Examples of module generation will be given in section 3.

2.3 THE FLOORPLANNING TOOL. — Floorplanning consists of the following steps :

1. Placement of the modules on the chip, subject to a compromise between wire length and area minimization.

2. Routing of power/ground, clock, signal and bus connections between modules (exploiting abutment and gliding terminal positions in flexible modules).

3. Extraction of routing and fan-out loading and predicting timing problems.

It should be noted that automation of the above is much more difficult than standard cell/gate array placement and routing for the following reasons :

— placement of arbitrary shaped blocks, rather than fixed height cells,

— irregular wiring channels and power-ground distribution,

— need for many types of routing : global, channel, switchbox, river, bus... routing.

For a good overview of these problems, see [22].

Although in principle good algorithms exist for pure placement such as slicing [23] and, recently, simulated annealing [24], it is felt by many designers, that none of these techniques has enough *intelligence* to compete with human ingenuity to obtain the best routing pattern. Therefore almost all actually existing floorplanners [25-27] allow, besides automated placement, for user interactive placement of modules. Much more essential is fully automated routing since this is a very error-prone and time consuming activity. Here also a preference goes to interactive use of a *toolbox* of routing strategies [22] rather than a fully automated solution.

As this layout technique differs in many respects from PCB layout, one has to face the acceptability problem of this technique by actual system designers.

In view of the highly heuristic approach of human ingenuity to placement and routing, it is to be expected that floorplanning is an excellent candidate for the future application of *expert system programming* techniques [28]. In this way the expertise of scarce layout specialists can be used to guite the non expert system designer through the composition of a high performance chip based on a silicium compiler output file.

As shown in figure 3, in the actual state of the art, *there are no true silicon compilers available.* What is

commercially available is a *structural specification language*, a selected set of *module generators* for a *given foundry* (true technology independence is still in a research phase in spite of glossy brochure statements) and *interactive floorplanners* with automated routing tools. They are available from new companies such as e.g. Silicon Compilers Inc., VLSI technology Inc. Seattle Silicon Techniques, Silicon Design Labs. etc. A good overview of their status can be found in [29].

Figure 3 also shows that :

1. Actual research should concentrate on the synthesis and testability aspects of silicon compilers of which they are the essential component and the key towards a fast design methodology. This will be discussed in section 4.

2. The fact that module generators are software programs will have a serious impact on the design style of actual silicon circuit- and layout designers. This aspect will be covered in the next section.

3. Module generators and their impact on silicon design.

3.1 ANATOMY OF A MODULE GENERATOR. — Module generators can be defined as *computer programs created* by a *team* of *silicon* circuit and layout designers and *software* engineers. The computer programs *generate* functional and timing instances of the module upon request of the *system designer* or silicon compiler layout. These instances satisfy given specifications based on the input of actual structural, geometrical and electrical parameters. Ideally a module should also be *adaptable* to new technology rules.

Therefore, module generators require a *programming environment* which allows for easy *creation*, *generation* and *adaptation* of modules (Fig. 4). Basically actual module generators generate three types of module structures illustrated in figure 5. They are :

— parameterized structured logic ;
— matrix programmed random logic ;
— standard cell random logic (discussed before).

In structured logic (Fig. 5a) a module is composed by a mathematically definable geometrical composition of primitive cells at transistor level. These cells fit mainly by *abutment* and *stretching* and sometimes by a limited set of personalization routing over cells.

The latter requires at least double metal CMOS technology.

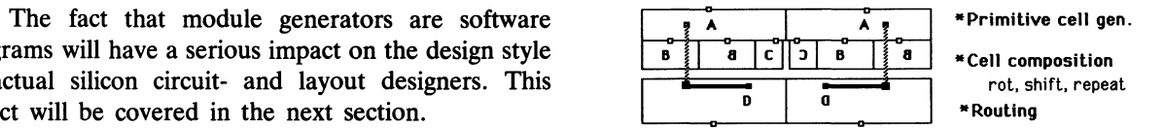


Fig. 4.

Fig. 5a.

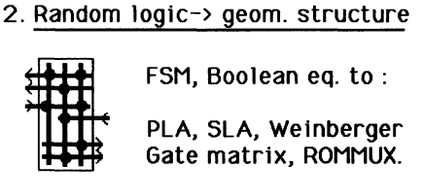


Fig. 5b.

Obviously structure logic is to be used for digital word processing : counters, registers, ALU, memory, multipliers, systolic arrays etc... It provides much better integration density than standard cells [13]. Matrix programmed logic (Fig. 5b) makes use of two-dimensional orthogonal wiring pattern in which randomness is programmed by transistor positions at crosspoints. Such structures can be optimally generated by computer programs starting from Boolean equations or Finite State Machine (FSM) descriptions.

Examples of these are PLA [30, 36], SLA [31], Weinberger-arrays [32, 33] and gate matrices [34, 35].

Figure 4 shows the anatomy of a module generation programming environment to the system designer (or silicon compiler). It behaves as a black box to which the input is a call for instantiation of a parameterized (m, n) module or a set of Boolean equation or FSM description. The output (which should be available in seconds !) is layout, functional and timing information for floorplanning and high level verification.

The role for the silicon designer (create action) is more complicated. The new aspect here is introduced by parameterization and technology independence. As an example, there is indeed a significant difference in the description of an *n by m* bit — as opposed to an 8×4 bit multiplier. The latter can be done only once on a traditional graphics editor, the former requires a

mathematical description in function of n and m but can be re-used for all m and n belonging to the *validity range* of the parameters.

Therefore, contrary to actual practice, the netlist of a given module in terms of cell primitives as well as the description of the relative positioning of the module structure needs to be expressed by a *language* rather than a schematics or graphical entry.

Figure 6 shows a very simple example of such a procedural description for a *repetitive interconnection* using, the HILARICS language [37] developed at IMEC under the EEC microelectronics regulation.

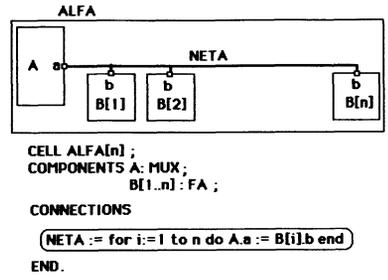


Fig. 6. — Example of a procedural netlist description using HILARICS.

In the sequel we will discuss some trends in cell building procedures (Sect. 3.2), cell composition procedures (Sect. 3.3) and expert verification systems (Sect. 3.4).

3.2 CELL BUILDING PROCEDURES. — In view of the above mentioned need for defining a cell as a flexible interconnection of transistors subject to abutment and stretching constraints, we can not describe cells as fixed coordinate geometries (Fig. 7a) except for very small cells (e.g. memory cells).

An obvious way to obtain flexibility is to describe polygons as functions of symbols (Fig. 7b) which in turn are functions of layout rules and cell constraints.

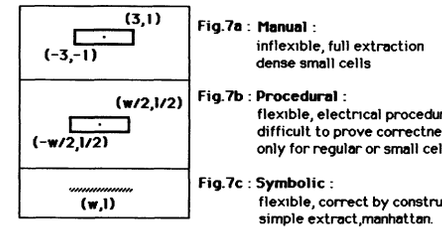


Fig. 7. — Cell building styles.

Netlist and relative positioning procedures are necessary to drive and verify abutment, routing, stretching and compaction during cell building and cell composition procedures necessary to produce the layout outline, terminal coordinate intervals and pattern generator input. Last but not least a module generator requires a verification system which must function also for creation, generation and adaptation phase.

It is clear that during the creation phase, in analogy to the debugging facilities offered to a software programmer, a continuous interactive verification of correctness is necessary at interconnection, functional and electrical level and this for the intended validity range of the module parameter set.

This role is rather traditional but the generation and adaptation phase require intelligent approaches to verification as the silicon compiler, system designer and floorplanner also ask for a very fast generation of instanciated functional and timing models. It is this authors opinion that *this again is an excellent application area for expert system techniques whereby the model building is done based on an intelligent registration by the system of the building sequence of a module.* The use of object oriented programming techniques using frames and demons based on LISP or SMALLTALK AI languages seems appropriate [38-40].

Finally, it should be noticed that adaptation to technology is only possible if some form of parameterization of cell layout parameters is available which calls also for a design *procedural* approach to transistor level design (see Sect. 3.2).

This is called *procedural layout.* Recently quite a lot of effort has been spent in such techniques.

Many of these systems are based on a kind of PASCAL [30, 42] or C language [41, 27].

A disadvantage of this technique is that it requires a new language definition (never complete !) and in order to be efficient in use, it needs to have an *interpretative* character during *creation* and a *compiled* character during *generation.*

A better technique is to make use of e.g. LISP [38], potentially with an object oriented superstructure [39], which can be used with all its available power to provide both interpretative and compiled facilities. Examples of this are DPL [43], ICPL [44] and Cameleon [45] under development at IMEC under the Esprit 97 contract.

It is interesting to note that at Hewlett Packard [46] ICPL is used by silicon designers after 4 weeks of LISP training... an example that, *given a company policy of continued education, designers are susceptable to these new design methods.* It is this authors opinion that the difficulty of the introduction of new techniques in Europe is often more an idea of managers than that of the peoples acceptance of it, provided time is made available as an investment in the future.

Procedural layout also has the advantage that, besides geometrical procedures, one can write an electrical procedure as well, based on the layout rules and electrical parameters of the technology used.

In this way, based on the interconnection procedure, immediately an electrical network can be created such that costly extraction CPU time is avoided. Even intelligent reduction techniques in RC networks can be programmed into the generated network such that the number of nodes in the circuit is drastically reduced for timing simulation.

An example can be found in [30].

However, procedural layout at transistors level has a serious drawback : it is very hard to formulate all mathematical restrictions on more than 10 geometrical items in a cell while still guaranteeing correctness under the required range of layout and cell parameters.

Therefore the technique is only applicable to very regular structures such as present in the *matrix programmed random logic* structures mentioned in section 3.1.

A possible solution to this problem has been presented by Kraak *et al.* [47] at ESSCIRC '85, whereby the *parameterization equations* are generated from an existing Manhattan type layout. By assigning new layout rules, transistor dimensions or terminal constraints, a new cell can be generated. This can be an important technique to support future procedural layout systems.

Cell flexibility can also be obtained by using *symbolic layout* techniques (Fig. 7c). Hereby layout is described by a loose relative placement of *layout symbols* for devices and *interconnection layers.* This is done on a symbolic editor. A spacing program then generates a true layout satisfying layout rules (defined in a technology file), wiring (terminal positions) and positional constraints. Hierarchical systems also support abutment procedures at the next level of the hierarchy.

Graph based [48, 45, 49] as well as virtual — grid based [50, 52] spacing have been reported. Their relative merits are discussed in [53]. Symbolic layout (restricted to Manhattan layout) has the obvious advantage of being flexible both for adaptation and composition. It is correct by construction and easily usable for electrical model generation since the electrical role of each symbol is predefined.

A disadvantage of the system is that often compaction algorithms are experienced as too difficult to control by the designer and layouts are not as dense as hand — or procedural layout usually provide.

A compromise, whereby the symbolic layout is procedurally generated (easy to do) followed by compaction to provide flexibility is an interesting technique for matrix programmed logic structures [54].

Last but not least, for single cells, one can also use a symbol based true layout whereby correctness at design time is checked by incremental design rule checking (DRC) and incremented changes are taken care of by plowing, move and compaction techniques such as in the MAGIC [55] and MOVE [56] systems. In a good module generator all these techniques should be made available in a user friendly *toolbox* programming environment.

Very small or very high performance cells can be done using the *MAGIC-MOVE* approach, more complex cells using automated abutment using symbolic layout or the Magic approach followed by the parameterization approach of [47].

Procedural or procedural-symbolic techniques are most appropriate for small cells of *matrix programmed logic.* As a typical example of a fully procedural module generator, figure 8 shows some aspects of the FSM and Boolean equation *silicon compiler PLASCO* [30] developed at IMEC. Figure 8a shows that it is a silicon compiler since it contains logic synthesis tools (optimal state assignment SOAP) based on [57] and product term and literal minimization (PRESTOL II [60] based on a heuristically best combination of ESPRESSO II [58] and PRESTO [59] techniques). The logic structure produced is a set of PLA matrices with a high degree of adaptability to wiring constraints (by folding) and to performance constraints by partitioning *Adaptability* to layout rules and wiring constraints is also insured by a full procedural layout definition of the PLA cell layout as well as electrical representation for timing parameter extraction. Notice also that, in case the PLA is embedded in a scan path, also ATPG is provided using techniques described in [61].

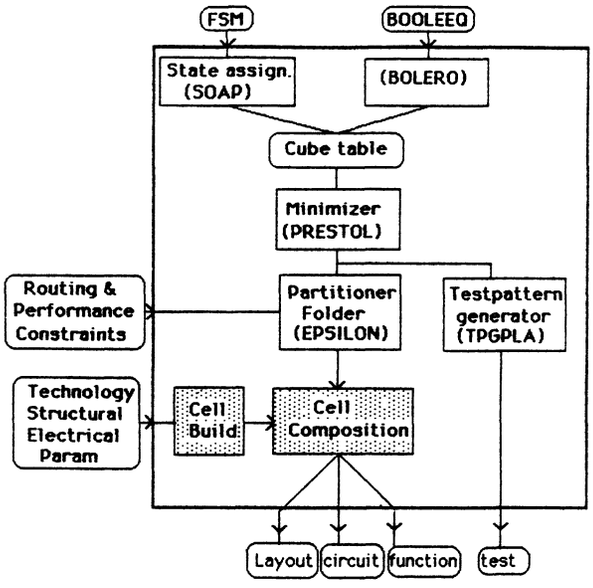
Figure 8b shows a straight forward standard PLA but figure 8c shows a PLA with the same functionality adapted for floorplanner routing constraints. Notice stretching for abutment in the routing environment.

Figure 9 shows a typical example of symbolic layout whereby figure 9a is the stored symbolic mother cell and figure 9b, c are two spaced instances of the cell respectively for 5 μm and 3 μm technology rules showing the adaptability feature of symbolic layout. Figure 9c shows a hierarchical use of symbolic cells which are composed by abutment and wiring compaction to an 8 bit data-path for a signal processor. This is a form of cell composition to be discussed next.

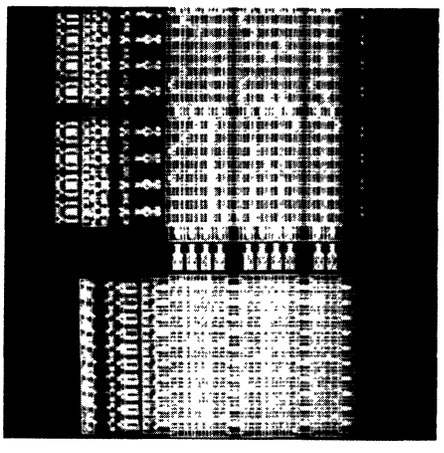
3.3 CELL COMPOSITION TECHNIQUES. — Cell composition consists in the assembly of a parameterized module from its primitive transistor cells in order to generate layout, functional and timing models for floorplanning and verification. The preferred technique, in order to save area, is to obtain connection by abutment or restricted routing over the cells. Generally a composition procedure consists of :

a) relative two dimensional-positioning rectangular cells ;

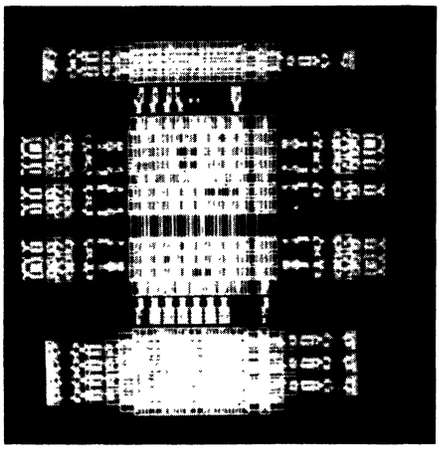
b) cell orientation towards abutment by rotation and mirroring operations ;



a)



b)



c)

Fig. 8. — PLA generator PLASCO as an example of a flexible procedural module generator. a. Block diagram of software packages in PLASCO. b. Standard PLA. c. Folded PLA subjected to wiring constraints and abutment causing cell stretching.

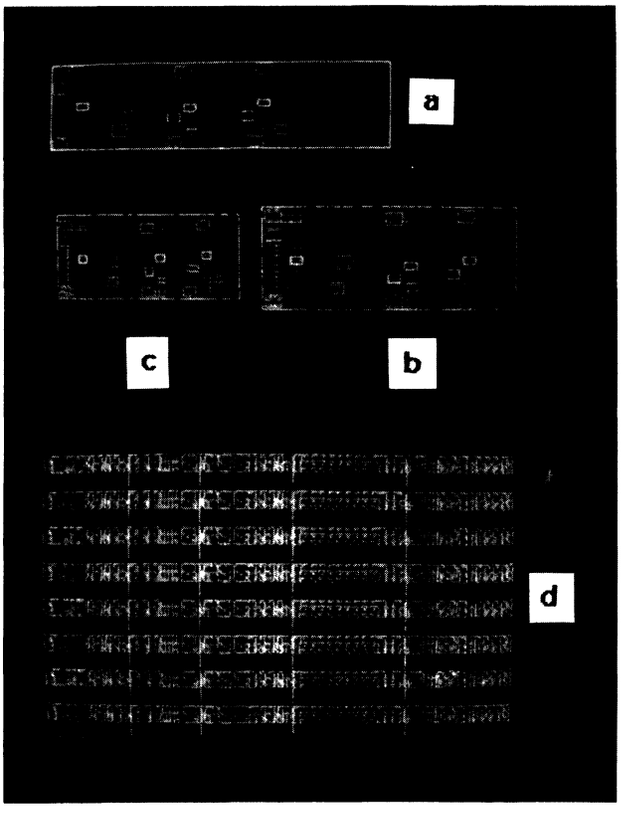


Fig. 9. — a. Symbolically stored cell. b. Spacing for 5 μm design rules. c. Spacing for 3 μm design rules. d. Using the cells for composition of 8 bit data path.

c) symbolic routing of personality wiring over (between cells) using routing functions (river, bus, switchbox, channel routing procedures) ;

d) compaction of relative positioning to obtain abutment by terminal move, abutment stretch and compaction operations.

Notice that all of the above *operations* are definable as functions operating on a list of objects. This forms an almost natural application of *functional* and *object oriented programming*. Therefore all composition, in this authors opinion, is best done in a *procedural, interpretative way using e.g. a LISP environment.*

Hereby one should be realistic enough to realize that, to todays 32 bit 1..5 MIPS workstations, LISP implementation very rapidly get out of hand due to software type identification and poor virtual memory management. Therefore, as long as LISP-like machines remain too expensive [63] one should use the LISP environment on a traditional workstation to do the interpretative composition and call PASCAL or C modules for the procedural operations. So until ca. 1989, when prices of AI machines will be competitive with UNIX workstations and AI software will be abundantly available [63], this remains a viable composition method on todays UNIX oriented workstations.

As an example we show in figure 10 a LISP *interactive cell composition procedure* on a VAX 11/780 using the

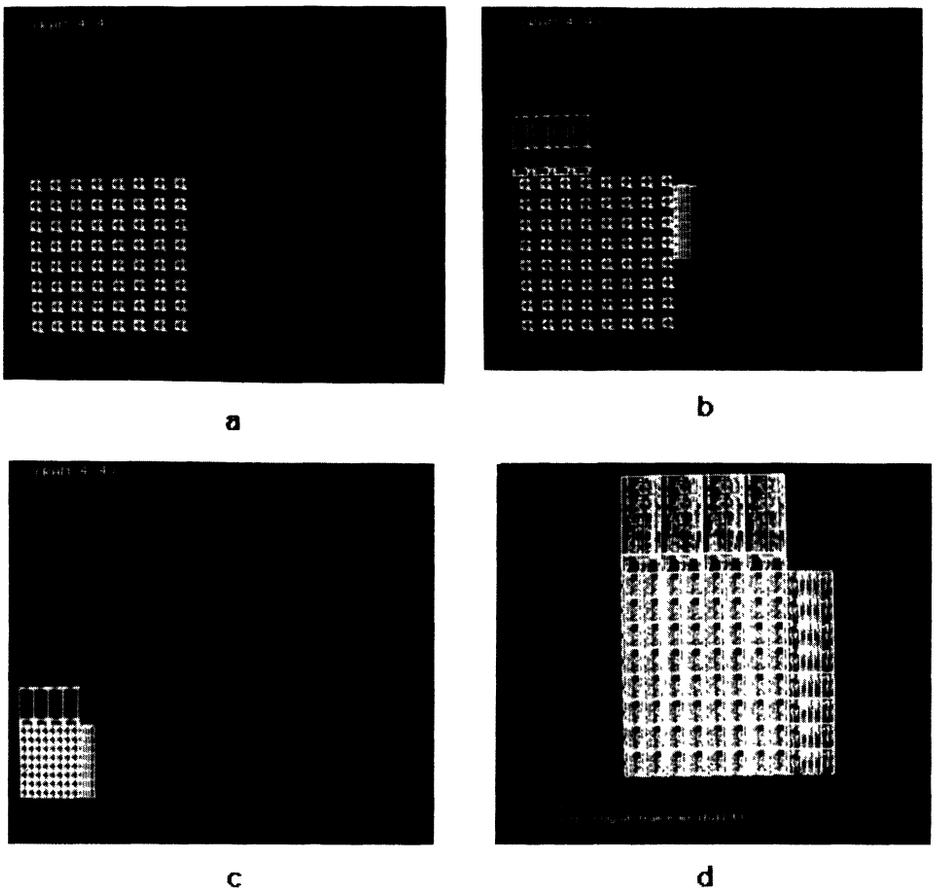


Fig. 10. — LISP based cell composition of a RAM. a. Procedural interpretative relative placement of ramcells. b. After relative positioning of symbolically designed decoder, sensing amplifiers and 2-bit decoders. c. Compaction causes abutment and stretching of all cells. d. Layout function produces final layout.

DEC LISP and PASCAL environment under VMS, and a 4 115 Tektronix colour graphics terminal. This system is part of Cameleon [45] being developed at IMEC under EEC Esprit 97 contract.

In such an interpretative environment the designer first specifies the relative placement of symbolic (procedural) cells using the *parameterized* relative position LISP function *add-comp* of a given type of cell at a list of relative position functions of the parameters.

The LISP function, when instantiated for its parameters, calls interpretatively a graphics PASCAL routine to show directly the relative cell positioning defined in the add-comp function.

As a result, the designer has an immediate graphical feedback to check the correctness of his (her) design.

Figure 10a shows the relative placement of $8 = 2 \times m\,(m = 4)$ columns of RAM cells and $2^{n-1} = 8\,(n = 4)$ rows of the RAM cell. In this case m is the wordlength and n is the address width of the RAM.

Figure 10b shows the result after adding the relative positioning of an $n - 1$ bit X-decoder, $m\,(= 4)$ sensing amplifiers and 2 bit Y decoders.

Figure 10c shows the result after the *compact-x, compact-y* functions are executed while figure 10d shows the result of the *layout* command which produces the instantiated $(m = 4, n = 4)$ RAM.

Notice that during the « compact » operations all cells are fitted by abut and stretch operations. After this interactive session a LISP program is available, which, after compilation and test over the validity range of parameters provides a new module generator for the silicon compiler or system designer.

It is to be noted that after successful abutment also an electrical, functional and timing model is available as will be discussed next. *The above clearly shows how important advanced programming techniques are becoming, even for the traditional silicon designers.*

3.4 EXPERT VERIFICATION SYSTEM. — In a module generator the verification system must check correctness during creation and adaptation and generate functional and timing models during generation phase.

Correctness means that, over the validity range of parameters, connectivity, synchronism, functionality, timing and electrical behaviour (noise margins) are maintained.

The traditional way of doing this is by simulation.

However simulation is a subjective test method which depends entirely on the test patterns for the problems already expected by the designer.

It is costly in CPU time and does not necessarily detect *unexpected* problems *which are the most important to catch anyway.*

It was therefore suggested by this author as early as in 1981 [64] to use what is now called, *rule based systems* to check circuits for correctness or to subject them automatically to local *guided simulation*, including the generation of testpatterns and the interpretation of simulation results. The latter is indeed part of the simulation cycle which is often *more costly in designers time than the actual CPU time spent in simulation.* (Dominance of intelligence over blind number crunching !). In the mean time, in an EEC microelectronics project MRO3KUL a prototype of such an expert system DIALOG [65] has been created. Another such effort is e.g. the RUBBIC [65] system developed in Berkeley. Many more are under development now. First it is to be noted that an expert system [28] differs strongly from a *procedural* program in the sense that the introduction of new rules, expression *good design,* in an expert system is done very easily without a program rewrite or a redefinition of a datastructure. This is necessary since *modules* are designed according to a *design style* (e.g. circuit types, clocking schemes, register types, testability, circuitry etc...) which is to be expressed by a knowledge engineer.

Rules can be expressed easily in so called object oriented languages [39, 40], which are usually built on top of LISP. These languages allow to define very complex data types and their properties, relations and attributes. Objects can inherit properties through hierarchy and communication between them can happen through messages, which do not have to know the contents of the object to operate on it.

In the expert debugging system DIALOG [65] the language LEXTOC is implemented on top of LISP which allows for a very powerful expression of rules about MOS circuits.

An outline of the DIALOG program is shown in figure 11. Based on a MOS transistor network, generated from the module generator, the user can start an interactive or automatic verification procedure which goes through the following stages :

3.4.1 *Decompilation.* — Hereby the MOS network is first checked for *valid circuit configurations.* These are the MOS circuits belonging to the given design style. For example static CMOS using passive multiplexing trees and level sensitive scan registers of prescribed circuitry. All circuits violating these rules are flagged as illegal. Then follows a *high level timing verification.* Starting from the primary cloks, the derived clocks are found and the register/combinational logic partitioning

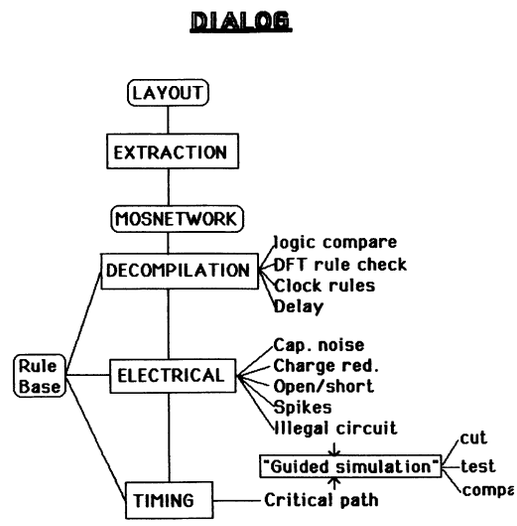


Fig. 11.

is found. The *validity of the clocking rules* to insure level sensitivity is verified. Violations are flagged *and explained.* If applicable, design for testability rules are also verified.

If all this has been checked and corrected, it is possible to use user defined hierarchy in the procedural interconnection description to extract the Boolean description of the cells and to *compare* them to the logic definition. This technique does render logic/switch level simulation unnecessary and leads to a much higher design reliability. Of course this supposes that extraction is possible which imposes restrictions to the design. These however are often coinciding with design for testability rules.

In the future we believe that tricks at the circuit level should be avoided in order to make design reliable and fast. Complexity should be handled at the high level of design not at the bottom.

Figure 12 shows an example of response of a the DIALOG program to a clocking rule verification request. The system has identified in red colour that a

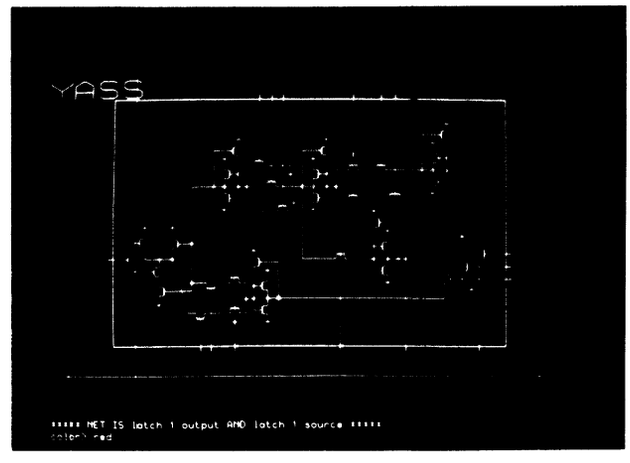


Fig. 12. — Example of the use of DIALOG. The red path is a possible race path because the clocks along the path have the same phase.

loop exists containing only a single clockphase which can lead to a race problem. Notice that the *user interface* is quite essential to the success of a debugging system. Back annotation to layout or schematic is necessary just as a language debugger in software indicates bugs at the appropriate statements in the language.

3.4.2 *Electrical rule checking.* — This substitutes for intensive SPICE like simulations. Hereby we search in each primitive circuit configuration found under 1, for illegal topological circuit configurations (open, short, odd *n*- p-MOS configurations etc.). When all circuits are acceptable circuit configurations, a checking is made whether they produce the correct logic levels. Therefore a large number of rules deals with capacitive noise feedthrough, impedance ratios, charge redistribution effects both in dynamic and static circuits (capacitive spikes). Most of these rules are simple *if* (*property, relation*) *then* (*action*) not involving heavy mathematical analysis. Therefore three types of decision can result :

— Clear violation of a rule even by simple analysis : direct action required by designer ;

— Problem below reliable threshold. Warning but acceptable ;

— The problem needs quantitive evaluation. In such case the DIALOG system will *cut out* the necessary and sufficient subcircuit *together with an appropriate testpattern* to check the problem by *guided simulation.* This technique is very valuable since it requires only a small modular circuit simulator and the designer does not have to spend his (her) time in specifying a testpattern. It is often overlooked how much costly time is spent in CAD/I/0. It does not help to speed up a CAD program by a factor of ten if I/0 is as slow as before. Notice that this technique is also very well applicable to the *adapt phase.* By storing all testpatterns as well as the whole testprocedure (as *metaknowledge*) the next generation of cells can be verified by the prestored verification rules.

3.4.3 *Timing verification.* — When high level clocking rules are satisfied, detailed timing can be verified. Hereto first the timing of the clock edges of latches is computed starting from the primary clock edge timing. Then the combinational time spans are computed. In the next step the critical path through combinational logic blocks are searched. To this end we use a gradual refinement technique. Using simple RC models first the bulk of the shorter paths are eliminated. For the remaining paths worst case test patterns for the gates in the path are selected on a rule base. Using a circuit simulation module delays are calculated automatically and the worst case timing paths are identified. Finally the logically impossible paths are eliminated and, if requested, the most critical paths are *cut out*, *test patterns* are computed and a detailed simulation is done using a circuit simulator to compute accurate timing to be compared to the time span allowable from the clock analysis.

Again by using metarules it is possible to write automatic timing procedures which will be called when timing models are requested during the *generate* or *adapt* phase. The technique can also be used to perform effectively a number of exhaustive tests of a parameterized module during *creation* time in order to build empirical parameterized timing models a within a given validity range of the parameter set. The DIALOG system has been programmed in a LISP-Pascal environment on VAX under VMS.

As discussed in [65], this leads to rather slow programs on a VAX but it has been shown that under certain restrictions it is easy to translate the system automatically into a Pascal program which is capable of analysing circuits at a rate of 50 k transistor/h. This situation will change in the future when the cost of LISP-machines will come down drastically.

The price of these machines will drop at a rate of 35 %/year [63] such that *it is to be expected that by the end of this decade object oriented programming machines may substitute many of the actual UNIX based workstations.*

Research on the application of expert systems in all parts of CAD is rapidly progressing. In fact expert systems are useful in all applications where *design style dependent heuristics* have to be used. Below are given a few already published areas of great promise.

1. Cell and module layout. Placement of transistors in a cell which insures signal and power constraints in order to obtain an optimally packed layout is the work of layout experts using numerous common sense rules. Recently rule based expert systems have been reported both at the detailed layout level (TALIB [67]) and the symbolic level (TOPOLOGIZER [68]).

2. Floorplanning, especially from the viewpoint of placement and optimal interconnection from a performance viewpoint is also a rule based problem. Furthermore many routing algorithms do not give 100 % routing. In such cases the last difficult connections could be made by a rule based system fed by the experience of experts [69].

3. Design for testability. There is not just one method to improve the testability of a design. A set of techniques exists, which is context dependent. The technique used depends strongly on the architecture and the economic constraints. Therefore it is again possible to formulate a set of rules which, based on all these factors, generates modifications to a design in order to improve its testability. A good example of such a system is described by Abadir and Breuer [11].

Finally, a very important area where expert systems go hand in hand with algorithmic systems can be found in the area of synthesis which will be discussed in the next section.

4. True silicon compilers. What does exist ?

As stated in section 2, the true silicon compiler generates valid layout from a behavioural specification. To our knowledge no such compiler is commercially available yet, except perhaps for small finite state automata.

On the other hand many of these compilers are under development all over the world. The most successful attempts that come close to manual layout are addressing themselves to a rather specific *target architecture*. As stated in [70] no single synthesis system will evolve but as many as there are target architectures.

Below is given a short outline of todays research in this area. Good review papers can be found in [71, 14, 72, 73].

The first true silicon compiler is probably the MAC-PITIS compiler of MIT [74]. It produces a microcoded processor based on a LISP like behavioural description language. The resulting layout however has very poor density and it is difficult to evaluate the performance of the generated processor.

Based on the MAC-PITTS work, the SILC compiler [75] is under development at GTE. Hereby attention is paid to performance and the compiler is running from a new hardware description language which, besides behavioural constructs also contains structural and relative placement constructs which are used as hints to the compiler. It is implemented in the FLAVORS [39] object oriented programming environment.

The tendency is now quite general to include such *pragmas* to direct the compilation process in a target direction. This is also the case in the Sili [76] compiler recently announced by AT & T. In this compiler also user defined heuristics are programmed as rules in an expert system. The compiler maps directly into a target architecture parts netlist and in microcode fields for the generated dapapaths.

A lot of activity takes place in the area of compilation of Digital Signal Processing (DSP) chips. In this respect we mention the FIRST [77] silicon compiler for bit-serial implementation of DSP circuits based on a block-diagram like description of the algorithm. In the Esprit 97 [78, 79] project a silicon compiler CATHEDRAL 1 for the automated design of wave digital filters from specs to layout of bit-serial architectures has been developed. It differs from the FIRST compiler by the fact that also the filter design and optimization part is included. At the University of Berkeley a similar program LAGER [80] is operative for the direct implementation of DSP algorithms at the block diagram level into add-shift multiprocessor bit-parallel structures. The design level is now being extended to the algorithmic level based on an applicative language SILAGE [18].

The use of artificial intelligence techniques is likely to have a strong influence on silicon compilation. This respect we refer to the pioneering work going on at Carnegie-mellon University [81] and AT & T [20]. A general approach to silicon compilation which addresses a wide range of architectures is the Yorktown Silicon Compiler developed at IBM and reported in detail in [82].

Last but not least, a lot of excellent work in the area of high level architectural synthesis is going on at the University of Karlsruhe [83] (CADDY) and Kiel [84] (MIMOLA).

The wide range of activities in this field indicate that there is good hope that true silicon compilation will lead to very efficient commercial packages that will allow to make the *one month system chip* a true fact by the end of this decade. An important aspect again will be the acceptability of the system designers of high level formal *language* specification for what used to be *hardware* design. The role of universities in a continued education role is of great importance in this respect.

5. Conclusions.

The design of complete application specific systems on a chip will become possible thanks to a so-called meet-in-the middle design strategy whereby system design is separated from silicon design.

This will only be possible if both silicon as well as system design are more formally based on software design principles. Programming environments for silicon modules are now becoming commercially available. In order to achieve another major breakthrough a lot of research effort is necessary in the area of high level architectural synthesis and automated design for testability.

In all these areas a lot of heuristic expert knowledge leads to the best solutions. Therefore the use of expert system techniques will make its inroads in the CAD area. This will only be economically justified when, as expected, the price of LISP-like machines will decrease by a factor of five. This is expected to happen by the end of this decade. At that point AI workstations will take over from the actual UNIX based machines. On these machines the best of algorithmic programming in PASCAL or C will be combined with object oriented programming styles for the rule based heuristic parts. These expert systems are extremely important in view of the shortage of VLSI design talent.

Universities will have a very important role to play in (re)-educating designers towards a much more software and formalism oriented design style whereby correct-by-construction methods should prevail over the more traditional design tricks used so abundantly in the past. In Europe more in particular, one must stimulate cooperation between Electrical Engineering and Computer Science departments. Relevant research is only possible by the formation of centres of excellence whereby academia, researchers and students are cooperating on large visionary projects devoted to relevant problems of the future and covering several disciplinary levels.

In this respect a cooperation of such groups with

industry is very necessary to keep projects realistic but also to bootstrap industry towards more progressive attacks of the numerous problems.

Last but not least, we should find a mechanism in Europe which stimulates new ventures for small companies based on promising academic or industrial research. This author believes that such mechanism needs to be based more on private capital (perhaps of the bigger companies) than on governmentally sponsored research. The latter is indeed performing its role as a starter of new ideas very well but the effort to transform an idea into a product is to be undertaken by small start-ups which can hardly get funded in the existing financial system. A solution is urgently required in order to get economical results out of public funding. A closer look at the US or Japanese system is very useful in that respect.

Acknowledgments.

All views expressed in this paper are the result of numerous discussions with collegues all over the world. In particular however they are the results of a joint view within the IMEC research group and of the partners in the Esprit 97 and the MR03KUL EEC microelectronics project. I thank all who have contributed to the ideas exspressed in this paper.

References

[1] KESSLER, A. J. *et al.*, Standard Cell VLSI Design : A Tutorial, *IEEE Circuits Devices Mag.* **1,** No. 1, (1985) 17-34.

[2] MARINO, J. T., *Low-Cost Hardware Logic Simulator/Fault Grade Machine*, Proceedings of the IEEE Custom Integrated Circuits Conference, p. 242-244, Rochester, New York, May 21-23, 1984.

[3] CRAWFORD, J., *An Electronic Design Interchange Format*, Proceedings of the ACM/IEEE 21st Design Automation Conference, p. 683-685, Albuquerque, New Mexico, June 1984.

[4] NEWTON, A. R., *Timing, Logic and Mixed-Mode simulation for large MOS IC's*, Nato Advanced Study Institute Series, E-48, Sijthoff & Noordhoff, p. 175-239, 1981.

[5] PAULINE *et al.*, *A Timing Verification System Based on Extracted MOSVLSI Circuit Parameters*, Proceedings of the 18th Design Automation Conference, p. 288-292, 1981.

[6] OOSTERHOUT, J. K., *Crystal : A Timing Analyser for nMOS VLSI Circuits*, Proceedings Third Caltech Conference on VLSI, R. Bryant ed., Computer Science Press, p. 57-70, 1983.

[7] OOSTERHOUT, J. K., *Switch-Level Delay Models for Digital MOS VLSI*, Proceedings ACM/IEEE 21st Design Automation Conference, p. 542-548, Rochester, New York, May 21-23, 1984.

[8] TUCKER, B. W., *Electronic CAD/CAM. Is it Revolution or Evolution*, Proceedings 22nd ACM/IEEE Design Automation Conference, p. 830-834, Las Vegas, Nevada, June 23-26, 1985.

[9] KOENEMAN, B. *et al.*, *Built-in Logic Block Observation Techniques*, Digest 1979 Test Conference, 79CH1509-9C, p. 37-41, October 1979.

[10] WILLIAMS, T. W. *et al.*, Design for Testability. A Survey, *IEEE Trans. Comput.*, **C-31,** No. 1 (1982) 2-15.

[11] ABADIR, M. S. *et al.*, A Knowledge Based System for Designing Testable VLSI Chips, *IEEE Design Test Comput.*, **2,** No. 4 (1985) 56-68.

[12] SAIGO, T. *et al.*, *A Triple-Level Wired* 24K *Gate CMOS Gate Array*, Digest of Technical papers of the International Solid-State Circuits Conference, ISSCC-85, p. 122-123, February 1985.

[13] KASAI, R., FUKAMI, K. *et al.*, An Integrated Modular and Standard Cell VLSI Design Approach, *IEEE Solid-State Circuits*, Vol. **SC-20,** No. 1 (1985).

[14] SANGIOVANNI-VINCENTELLI, A., *An Overview of Synthesis Systems*, Proceedings Custom Integrated Circuits Conference, CICC-85, p. 221-225, May 1985.

[15] BURIC, M. *et al.*, *The Plex Project : VLSI Layouts of Microcomputers Generated by a Computer Program*, Proceedings IEEE International Conference on Computer-Aided Design, ICCAD-83, p. 49-50, 1983.

[16] FOX, J., SURACE, G. *et al.*, *A Self Testing* 2 *Micron CMOS Chip Set for FFT Applications*, Digest of Technical Papers of the 11th European Solid-State Circuits Conference, p. 13-24, Toulouse, France, September 16-18, 1985.

[17] KOOMERS, MOTO-AKA, *Proceedings of the* 7th *International Conference on Computer Hardware Description Languages* (North-Holland) 1985.

[18] HILFINGER, P., *A High-level Language and Silicon Compiler for Digital Signal Processing*, Proceedings 1985 IEEE Custom Integrated Circuits Conference, CICC-85, p. 213-216, Portland, Oregon, May 20-23, 1985.

[19] FELDBRUGGE, F. H. J., *VLSI and Petri-nets*, Nato Advanced Study Institutes Series E-47 (Sijthoff & Noordhoff) 1982, p. 285-300.

[20] KOWALSKI, T. J., THOMAS, D. E., *The VLSI Design Automation Assistant : What's in a Knowledge Base*, Proceedings of the 22nd ACM/IEEE Design Automation Conference, p. 252-258, Las Vegas, Nevada, June 23-26, 1985.

[21] KOWALSKI, T. J., GEIGER, D. J., *The VLSI Design Automation Assistant : A Birth in Industry*, Proceedings of the 1985 International Symposium on Circuits and Systems, p. 889-892, Kyoto, Japan, June 5-7, 1985.

[22] SOUKUP, J., Circuit Layout, *Proc. IEEE*, **69,** No. 20 (1981) 1281-1304.

[23] LAUTHER, U., *A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation*, Proceedings of the 14th Design Automation Conference, p. 1-10, June 1979.

[24] KIRCKPATRICK, S. *et al.*, Optimization by Simulated Annealing, *Science* **220** (1983) 671-680.

[25] ANTOGNETTI, P., ARATO, G. *et al.*, *ARIANNA : A Floor-Planning Tool*, Technical Digest of papers of the 11th European Solid-State Circuits Conference, Toulouse, France, September 16-18, 1985.

[26] TRIMBERGER, S. *et al.*, *Automatic Layout in an Open Design System*, VLSI Design, p. 88-98, May 1985.

[27] BURIC, M. *et al.*, *Silicon Compilation Environments*, Proceedings IEEE Custom Integrated Circuits Conference, p. 208-212, May 1985.

[28] HAYES-ROTH *et al.*, Building Expert Systems (Addison-Wesley) 1983.

[29] YOUNG, J., IC-Design Automation strides into Silicon Compilation Era, *Electronics* (1985) 58-63.

[30] BARTHOLOMEUS, M. *et al.*, *PLASCO : A Procedural Silicon Compiler for PLA Based Systems*, Proceedings of the IEEE 1985 Custom Integrated Circuits Conference, p. 226-229, 1985.

[31] PATIL, S. A. *et al.*, A Programmable Logic Approach for VLSI, *IEEE Trans. Comput.* **C-28** (1979) 594-601.

[32] WEINBERGER, A., Large Scale Integration of MOS Complex Logic, *IEEE Solid-State Circuits* **SC-2** (1967) 182-190.

[33] WING, O., HWANG, S. *et al.*, Gate Matrix Layout, *IEEE Trans. Comput.-Aided-Design Integrated Circuits Syst.* **CAD-4,** No. 3 (1985) 220-231.

[34] KANG, S. *et al.*, Gate Matrix Layout of Random Control Logic in a 32-bit CMOS CPU Chip Adaptable to evolving Logic Design, *IEEE Trans. Comput.-Aided-Design* **CAD-2** (1) 1983.

[35] PIQUET, C., Design Methodology for Full Custom CMOS Microcomputers, *Integration VLSI J.* **1,** No. 4 (1983) 335-350.

[36] CHUQUILLANQUI, S., *PAOLA : A Tool for Topological Optimization of Large PLAs*, Proceedings of the Design Automation Conference, p. 300-306, Las Vegas, June 1982.

[37] VANDEN MEERSCH, E., *HILARICS Manual*, Report 5-B3-1 of the Report No. 5 on MR-03-KUL EEC Report. Available from IMEC, April 1985.

[38] WINSTON, P., HORN, B., LISP (Addison Wesley) 1981.

[39] Symbolics Inc., *Reference Guide to Symbolics-LISP*, March 1985, p. 417-441.

[40] GOLDBERG, RUBSON, *SMALLTALK-80 : The Language and its Implementation* (Addison-Wesley) 1983.

[41] MATHESON, T. G. *et al.*, *Embedding Electrical and Geometrical Constraints in Hierarchical Circuit-layout Generators*, Proceedings IEEE International Conference on Computer-Aided Design, p. 3-5, Santa Clara, California, September 12-15, 1983.

[42] MULLER, B. *et al.*, *The Chipgenerator Concept. A New Approach to Full Custom CMOS IC Design*, Proceedings of the 11th European Solid-State Circuits Conference, p. 186-192, Toulouse, France, September 16-18, 1985.

[43] BATALI, J., HARTHEIMER, A., *The Design Procedure Language*, A.I. Memo No. 598, MIT, September 1980.

[44] KUCHINSKI, A., *ICPL Integrated Circuit Procedural Language*, Proceedings of the IEEE ICCD-84 Conference, October 1984.

[45] RIJNDERS, L. *et al.*, *Cameleon Version 1.1, Users Guide*, Report nr. 5-C1-3 of EEC project MR-03-KUL, Available from IMEC.

[46] SORENS, M. J. *et al.*, *Using ICPL : A Programmatic IC Design language*, Proceedings of the ACM/IEEE Compcon Conference, September 1984.

[47] KRAAK, M., KOOPMANS, J. M. *et al.*, *Cell Layout Library Parameterization*, Digest of Technical Papers of the 11th European Solid-State Circuits Conference, p. 257-262, Toulouse, France, September 16-18, 1985.

[48] HSUEH, M. Y. *et al.*, *Computer-Aided Layout of LSI Circuit Building Blocks*, Proceedings of the International Symposium on Circuits and Systems Conference, p. 474-477, July 1979.

[49] DEVECCHI, D. *et al.*, *Symbad : A Tool for a New Layout Methodology*, Proceedings IEEE 1985 Custom Integrated Circuits Conference, p. 64-67, Portland, Oregon, May 20-23, 1985.

[50] WESTE, N., *Virtual Grid Symbolic Layout*, Proceedings of the 18th Design Automation Conference, Nashville, p. 225-233, June 1981.

[51] WESTE, N., MULGA — An Interactive Symbolic Layout System for the Design of IC's, *Bell System Techn. J.* **60,** No. 6 (1981) 823-858.

[52] ROSENBERG, J. *et al.*, *A Vertically Integrated VLSI Design Environment*, Proceedings of the 20th Design Automation Conference, p. 31-38, June 1983.

[53] WESTE, N. *et al.*, *Principles of CMOS VLSI Design : A Systems Perspective* (Addison-Wesley) 1985, Chapt. 7.

[54] VAN VLIERBERGHE, S. *et al.*, *Symbolic Hierarchical Artwork Generation System*, 22nd ACM/IEEE Automation Conference, p. 789-793, Las Vegas, Nevada, June 23-26, 1985.

[55] OOSTERHOUT, J., The MAGIC VLSI Layout System, *IEEE Design Test Comput.* **2,** No. 1, (1985) 19-30.

[56] BERGMANN, N., *Move — A Useful Primitive for a Variety of IC CAD Tools*, Digest of Technical Papers of the 11th European Solid-State Circuits Conference, p. 178-185, Toulouse, France, September 16-18, 1985.

[57] DE MICHELI, G., *Kiss : A Program for the Optimal State Assignment of Finite State Machines*, Proceedings of the 1984 International Conference on Computer-Aided Design, p. 209-212, Santa Clara, USA, November 1984.

[58] BRAYTON, R. *et al.*, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer Academic Publishers) 1984.

[59] BROWN, D. W., *A State-Machine Synthesizer SMS*, 18th Design Automation Conference, p. 301-305, June 1981.

[60] BARTHOLOMEUS, M. *et al.*, *Prestol-II : Yet Another Logic Minimizer for Programmed Logic Arrays*, Proceedings of the 1985 International Symposium on Circuits and Systems, p. 447-450, Kyoto, Japan, June 5-7, 1985.

[61] OSTAPKO, D. L. *et al.*, Fault Analysis and Test Generation for PLA's, *IEEE Trans. Comput.*, **C-28,** No. 9, 1979.

[62] ZINSNER, R. *et al.*, *Technology Independent Symbolic Layout Tools*, Proceedings of the IEEE International Conference on Computer-Aided Design, p. 12-13, September 1983.

[63] MANUEL, T., The Pell-Mell Rush into Expert Systems, *Electron.* (1985) 54-59.

[64] DE MAN, H., *Mixed-Mode Analysis and Simulation Techniques for Top-Down MOSVLSI Design*, Proceedings of the 1981 European Conference on Circuit Theory and Design, 5-10, 1981.

[65] DE MAN, H. *et al.*, Dialog : An Expert Debugging System for MOS VLSI Design, *IEEE Trans. Computer-Aided Design*, **CAD-4,** No. 3 (1985) 303-311.

[66] LOB, C. *et al.*, *Circuit Verification using Rule-Based Expert Systems*, Proceedings of the International Symposium on Circuits And Systems, p. 881-884, Kyoto, Japan, June 5-7, 1985.

[67] KIM, J. *et al.*, *TALIB : An IC Layout Design Assistant*, Proceedings of the AAAI Conference, p. 197-201, Washington, 1983.

[68] KOLLARITSCH, P., WESTE, N., Topologizer : An Expert System Translator of Transistor Connectivity to Symbolic Cell Layout, *IEEE J. Solid-State Circuits* **SC-20,** 3 (1985) 799-804.

[69] FUJITA, T. *et al.*, *Knowledge base and Algorithms for VLSI Design*, Proceedings of the 1985 International Symposium on Circuits and Systems, p. 877-880, Kyoto, Japan, June 5-7, 1985.

[70] FREHEL, J., *A Formalism for Logical and Electrical Design*, Technical Digest of Papers of the 11th European Solid-State Circuits Conference, p. 177-178, Toulouse, France, September 16-18, 1985.

[71] ELMASRI, M., *Digital VLSI Systems*, (IEEE Press) 1985, part II, p. 120-240.

[72] NEWTON, A. R., *Techniques for Logic Synthesis*, Proceedings of the VLSI-85 Conference, p. 27-42, Tokyo, Japan, August 1985.

[73] GOLDBERG, A. *et al.*, Approaches Toward Silicon Compilation, *IEEE Circuits Devices Mag.* **1,** No. 3, (1985) 29-39.

[74] SOUTHARD, J., Mac Pitts : An Approach to Silicon Compilation, *IEEE Comput.*, **16** (1983) 74-82.

[75] BLACKMAN, T. *et al.*, *The SILC SILIcon Compiler : Language and Features*, Proceedings of the 22nd Design Automation Conference, p. 232-237, Las Vegas, Nevada, June 23-26, 1985.

[76] KAHRS, M., *An Overview of SILI (a Silicon Compiler)*, Proceedings of the VLSI-85 Conference, p. 43-53, August 1985.

[77] DENYER, P. *et al.*, *A Silicon Compiler for VLSI Signal Processors*, Proceedings of the 1982 European Solid-State Circuits Circuits Conference, ESSCIRC 82, September 1982.

[78] DE MAN, H. *et al.*, *Custom Design of a VLSI PCM-FDM Transmultiplexer from System Specs to Layout using a CAS System*, Proceedings Esprit Technical Week, September 1985.

[79] DE MAN, H. *et al.*, *Development of a CAD Methodology for VLSI Signal Processing Devices using multiprocessing Architectures*, Proceedings Esprit Technical Week, September 1985.

[80] RABAEY, J. *et al.*, *An Integrated Automated Layout Generation System for Digital Signal Processing Circuits*, Proceedings of the Custom Integrated Circuits Conference, p. 217-220, May 1985.

[81] RAJAN, J., THOMAS, D. E., *Synthesis by Delayed Binding of Decisions*, Proceedings of the 22nd ACM/IEEE Design Automation Conference, p. 367-373, Las Vegas, Nevada, June 23-26, 1985.

[82] BRAYTON, R. *et al.*, *The Yorktown Silicon Compiler*, Proceedings of the 1985 International Symposium on Circuits and Systems, ISCAS-85, p. 393-394, June 1985.

[83] ROSENSTIEL, W. *et al.*, *Synthesizing Circuits from Behavioral Level Specifications*, Proceedings of the 7th Conference CHDL-85, p. 391-403, Tokyo, Japan, August 1985.

[84] MARWEBEL, P., *The MIMOLA Design System : Tools for the Design of Digital Processors*, Proceedings of the 21st ACM/IEEE Design Automation Conference, p. 587-593, Albuquerque, new Mexico, June 25-27, 1984.