COMPUTER AIDED SYNTHESIS OF DIGITAL SIGNAL PROCESSING ASICS

H. De Man, J.Rabaey, P.Six, L.Claesen

Interuniversity Microelectronics Center, Heverlee, Belgium

1. The challenge of the future : conceptual CAD.

Most ASICs designed today are either of the gate array or standard cell type. They are designed in much the same way as the traditional TTL/CMOS MSI boards and mostly they are intended to be economical substitutes for low performance, low complexity glue **logic** between LSI components on PCB's. The actual trend is more towards the generation of MSI/LSI structured building blocks by the use of **module generators**, often (erroneously) called silicon compilers. This is a first step towards ASICs containing complete **systems** on a chip.
Many services now become available offering "complete" solutions to the system designer i.e. the CAD tools, the silicon library and the brokerage to a silicon foundry. In all cases the principle of the **meet-in-the-middle** [DeM86] design strategy is followed. Hereby the logic functions and the bounding boxes of the software silicon cells provide the necessary abstraction mechanism between the specialized silicon design effort and the system designer.
In the past five years we have indeed seen an enormous evolution of CAD tools in the areas of user interface, data base, open system architecture, simulation, timing verification, testing, module generation, floorplanning and place and route tools.
However impressive this progress may be, in a recent study [Rap86] the following figures have been compiled about a typical ASIC design process :

	Manual	CAD
conceptual part	65%	15%
verification part	25%	40%
layout part	10%	45%

This clearly shows that actual CAD tools have solved the **structural and physical** part of the design process but unfortunately leave the **conceptual** part of the design process untouched. This latter part involves the **behavioral decomposition** of a system specification into structure i.e. the interconnection of the silicon cells. Since this problem is untouched,, verification of the correctness of the human decomposition steps by simulation takes too many human and computer resources.

A second observation in the world of ASICs is the trend either to higher system complexity or higher performance. Fig. 1 shows the application areas of programmable logic components and actual standard cell and gate arrays in a clock frequency vs. complexity plane. As stated above these techniques will continue to serve the glue logic market. However the challenge of the coming 10 years is most probably in the area of **perceptual electronics**, which can roughly be

defined as **real time** communication with man or machines of sound (voice, music) and image information. This will especially be caused by the wideband ISDN "wiring" of homes opening an enormous market of sophisticated home electronics. Systems in that area are no longer simple logic ocntrol circuits but involve the realization of **sophisticated real time** digital signal processing **algorithms** in silicon.

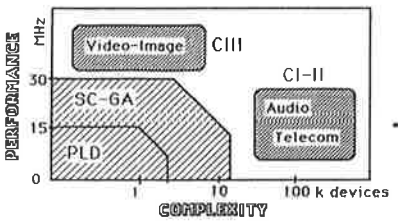

Fig.1

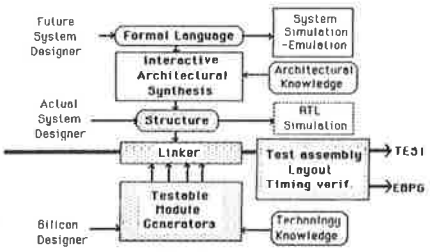

Fig. 2

Referring to Fig. 1 such systems on a chip are either highly complex algorithms in the audio spectrum or high performance video and image processing systems of lower complexity. Since they are data processing algorithms, they require **structured logic** and a lot of on chip **dedicated memory** and therefore can only efficiently be realized using parameterized module generation techniques realizing so called **compiled cells**.

Moreover, for such systems, the **time to the market** is of prime importance and, as such, design iterations can not be afforded.

In view of the above, future CAD systems adressing such applications should pay attention to the **conceptual part of CAD at the systems specification level.**

Very often these statements are immediately associated to the concept of behavioral **silicon compilation**. Hereby the system designer specifies the system in terms of a high level language which is **automatically** mapped into the hardware/firmware of a chip layout image. Many attempts in this direction are reported [Sou83, Bla85], Bra85,Mar86,Ros85]. In most cases however such systems are strongly oriented towards synthesis of Von Neumann type of computer architectures rather than the real time DSP algorithms discussed above and **lack of interactivity** causes usually very inefficient architectural solutions.

More efficient solutions, adressing also more restrictive applications such as digital filtering, have recently been proposed such as FIRST [Den82] starting from flow graphs and CATHEDRAL I [Jai86] starting directly from filter specifications and including algorithmic optimization techniques.

In this contribution, in section 2, we first want to state a few general principles which we believe will characterize future CAD systems for DSP ASICs.

We will then describe in section 3 how we have tried to realize these principles in a prototype CAD system called CATHEDRAL II [DeM86] adressing the complex algorithm audio application field. Some application examples are given.

2. General principles of future CAD systems for DSP ASICs.

2.1 From standard cells to module generators and block place and route.

Standard cell and gate array techniques use libraries at the gate and flip-flop level. They are mainly oriented to random logic design styles and, only recently, provision is made to include more structured logic blocks such as RAM, ROM and PLA. When one considers complete systems on a chip this level is too low. System designers are used to design with MSI/LSI building blocks at the level of ALU, MPY-ACC, ROM, RAM and even complete core microcoded processors. Usually such **modules** are themselves constructed from **functional building blocks (FBB)** (e.g. adders, counters, reg, mux...). In view of the enormous variability required from such a module library as well as due to the fast evolution of technology it is not realistic to store fixed layout patterns of all instances of the modules. This then leads to concept of so called module generators now being offered by the so called "silicon compiler" vendors.

Referring to Fig. 2, module generators are **software procedures** generating **views** of instances of parameterizable module types by assigning **actual parameters** to the formal parameter list of the module generating procedures. These procedures have been designed by the silicon specialists of the foundry service.

By views, we mean the complete **characterization** of an instance of a module to be used in a system design. This means that, besides the layout information, we also need a **timing** view (in function of parameters and loading conditions) , a **functional** view for RTL level simulation and last but not least a **testing** view (the latter is usually missing in the actual commercial CAD systems!).

Most of the module generators provide some means to be **"technology independent"** (read within a given mask sequence and for limited changes in the spacing rules...). Usually this is achieved by describing layout at the transistor level by procedures. A disadvantage of this technique is that it is **virtually impossible to guarantee correctness** under technology changes. We believe that, in the future, cell correctness under limited technology changes and pitch matching requirements can only be guaranteed by using **symbolic layout and compaction techniques** at the transistor cell level and that procedural layout at the cell composition level [We81,Six86].

The CAD system discussed sofar corresponds to the **dotted** part of Fig. 2 which, today, is becoming **commercially** available. Such systems are **structural** in nature. The system designer conceptually defines his/her system at a schematics editor as an interconnection of instanciated module calls (65% of the design time!). This is followed by intensive **verification** of behavior by **simulation** using the instanciated functional view of the modules (RTL simulation). The appropriate views are generated by the **linker** which is a message passing interface between structural/constraint description and the module generator library.

Notice that the **bold line** in Fig. 2 symbolizes the strict separation between the system design activity and the silicon design activity, which, in view of the specialized nature of it must be left in a reusable way to the scarce number of silicon specialists. This is the meet-in-the-middle design methodology.

Once simulation has been completed, **layout** can start. Layout of chips with modules generated

by module generators requires general block place and route systems which only recently have become commercially available.
Fig. 3 shows an example of an automated place and route of modules as generated from the module generator in CATHEDRAL II [Six86].

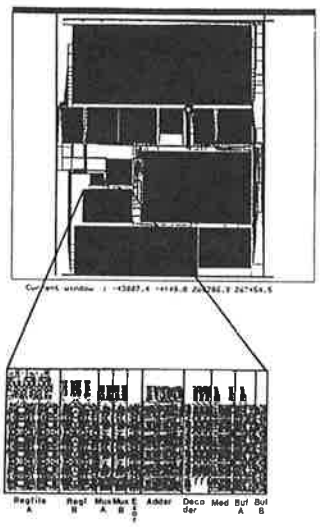

Fig. 3 : Block place and route of parameterized modules in CATHEDRAL II

Only after layout is completed we know the correct impact of wiring and fanout delay and a full chip **timing analysis** will be necessary potentially leading to a redesign of buffers, use of other FBB's in modules or even a complete redesign of the chip. There is a growing trend towards the use of timing analyzers rather than reliance on simulation. [Szy86,VdM86]. However recent work indicates that extreme care must be taken against logically impossible (false) paths which can introduce overly pessimistic results. In [Ben87] a solution to the false path problem is given.
Last but not least, and here is the weak spot of nearly all commercial systems, one needs the generation of the test patterns for the complete chip based on the testing views of the constituting modules. **Very little work has been published on systematic ways to assemble complete test patterns for ASICs based on a parameterized cell library.**

2.2 Towards architecture specific synthesis systems

Referring again to Fig. 2 let us now concentrate on the conceptual part of the design process.
In order to further shorten the design effort we will have to help the system designer in the translation of the system specification into structure. Most DSP systems adressing the audio and image processing fields are of an **algorithmic** nature handling synchronously generated blocks of data (e.g. speech samples, image pixels, spectral amplitude-phase pairs etc...). Therefore such algorithms, as well as their real time **constraints,** can be described by very high level **formal behavioral languages** rather than by an ambigious inexecutable natural language. An example will be given in section 3. The advantage of such a description is that it can be simulated (or even better : emulated) to check the correctness. Once this is done, theoretically no further simulation at lower level is required when using an automated synthesis system. The key CAD tool therefore is the **architectural synthesis** program.
A carefull study of the DSP application field has made us to believe that "the silicon compiler" will probably never exist ! This is due to the fact that the trade-off between area-time-power in silicon as well as the real-time requirements are much more complex than the trade-offs in software compiler construction. This is clearly indicated in Fig. 4 which shows that, even for DSP applications, many alternative **architecures** have to be considered dependent on the

complexity and data throughput of the algorithms considered. Therefore we believe that, in the future, a variety of **vertically sliced synthesis systems** will exist which potentially will support different design styles to be combined in a single system on a chip. Example : array type processors for front-end image processing and multiprocessor system to do the back-end processing of the compressed image data. In the ESPRIT 97 program we have developed synthesis systems for bit-serial filters (CATHEDRAL I) [Jal86] as well as a system for multiprocessor systems (CATHEDRAL II) [DeM86] and we are working on a system for high speed bit-parallel concurrent datapaths (CATHEDRAL III). Ultimately this will be extended to array type concurrent processors and then an integration of this system in an **open architecture type CAD system** [Bro87] is envisioned.

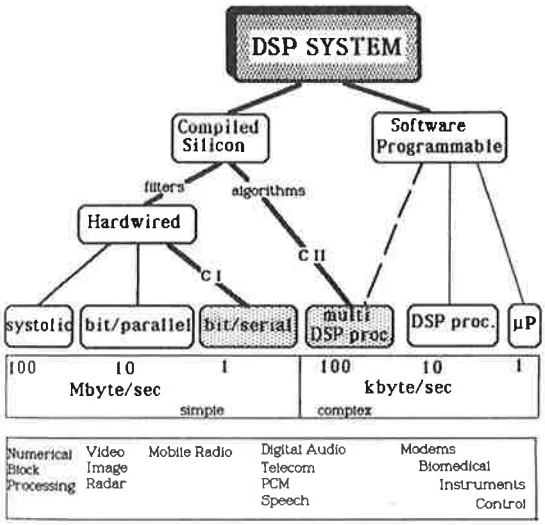

Fig.4 : different architectures for DSP depend on throughput and application.

As will be discussed in section 3 and illustrated in Fig. 2, the strong link of a synthesis system to the **target architecture,** requires that the synthesis system is driven from a set of system designer defined rules describing the **architectural knowledge** i.e. the possible structural hardware compositions, the microcode of the modules and the rules for optimal mapping of the typical application algorithms on the harware.
Moreover, it is our experience that **natural intelligence** is still beating by far artificial intelligence and therefore the key to succes in this field is that succesful synthesis systems must allow for **user interactivity** to explore and evaluate the design space for a fixed specification in a fast way whereby all detailed optimization work of the alternatives is taken care off by the CAD system.
Hence the term : **computer-aided-synthesis** rather than silicon compilation.
In order to make these points clear, we describe in section 3 the example of CATHEDRAL II as a representative example of the principles stated above.

3. An example of a conceptual CAD system : CATHEDRAL II

3.1 A dedicated multiprocessor architecture

CATHEDRAL II is a synthesis system adressing complex DSP algorithms in the audio spectrum. Typical applications are : modems, echo-cancellers, digital audio, spectrum analysis, speech processing, servo-systems, robotics (matrix calculations), data-compression etc... At sample rates of 100kHz and clock rates of 10MHz a programmable module can spend 100 cycles on the algorithm for wordlengths varying between 8 and 32 bits. Since most algorithms require many more than 100 cycles, one has to consider **concurrently operating processors** , which in order to keep silicon area reasonable have to be **dedicated** to the particular algorithmic requirements.

A careful study in the ESPRIT97 project has shown that signal processing algorithms can be described as structured programs whereby a set of independent **functions** execute subtasks of the algorithm. These functions exchange global variables through a synchronous protocol requiring some data buffering. This leads to an architecture as proposed in Fig. 5 [Cat86].

Fig. 5a gives the outline of the chip architecture. Each subtask is taken care of by a **processor** consisting of a **dedicated datpath** and its associated **local microcoded controller** . Global variables are transferred over **buffer elements** executing the **interprocessor communication protocol**. Finally the complete processor is controlled by a **master controller** firing start and stop conditions to the processors and their communication network.

Fig. 5b illustrates how each datapath is composed of a cluster of strongly connected **execution units** (EXU's) such that it optimally executes its part of the algorithm. Each EXU is a four terminal block with two parameterizable input registerfiles with up to 8 registers and one or two tri-state output buffers to be connected to the **customizable bus structure** (max. 5 busses can run over the EXU bit slice).

In view of the reduction of the variability we have found that, for DSP datapaths, the following EXU's in all cases have lead to efficient realizations :

1) General purpose EXU's : ALU-SHIFT , Adress Calculation Unit (ACU)

2) Accelerator functions : MPY-ACCumulate, DIVider, COMParator, NORMalizer.

The ACU unit is extremely important for DSP in view of cyclic memory traversals (filtering, correlation, convolution) as well as decimation and interpolation. The COMP function is important in decision making algorithms. The NORMalizer is a fixed-point to floating point (and vice-versa) transformer. This becomes more and more important in many advanced DSP algorithms involving matrix computation.

All these EXU's are parameterizable in wordlength, register-file size, 1-5 busses, 1-2 buffers of different strengths, type of adders, max. shift, pipeline in MPY a. o.

Accelerator functions are used in critical performance parts when general purpose parts (less silicon) are insufficient for the job.

Fig.5c shows the general architecture of the multibranch controller used. It has the flexibility to support decision making as well as regular repetitive algorithms in an efficient way.

Finally buffer structures are necessary to provide interprocessor communication. In CATHEDRAL II first the microcode of the processors is scheduled independently. When the schedules of the

processors are known one also knows the cycle numbers in a program **frame** at which the global variables of a processor a are **produced** and when these variables are **consumed** by processor b and whether the communication is uni- or bidirectional.

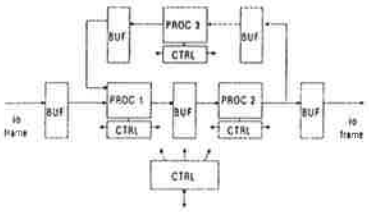

Fig.5a : example of multiprocessor architecture.

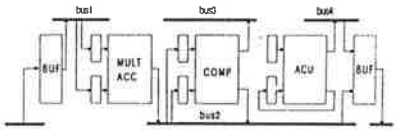

Fig.5b : single processor datapath example

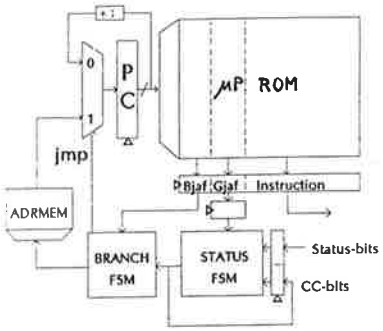

Fig. 5c : general multibranch controller

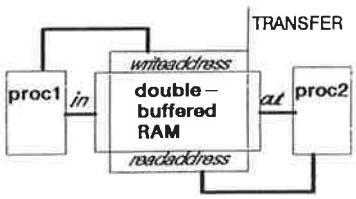

Fig.5d : interprocessor communication example

Bidirectional communication is possible using a **switched RAM**. However, this seldom occurs and is very expensive in terms of decoding and ACU necessary to control it. In most cases either a **FIFO** with **wired** pointer read and write adressing is much more efficient (Fig. 5d). In most cases processors are interconnected as a linear array and automated procedures have been developed to compute the **optimal skew** between processors so as to minimize the storage capacity of the buffers.

It is important to notice that a very concize definition of an architecture is of outmost importance to be able to create a conceptual CAD system. As an example we will describe in the sequel the architectural synthesis parts of CATHEDRAL II.

3.2 A pragma based interactive synthesis system

3-2-1 : Outline of the CAD system.

The architectural synthesis problem can be defined as follows :

"Given the behavioral description of a synchronous DSP algorithm, generate within

*the timing constraints an area optimal structure of the **datapaths, their local controllers with their microcode and the interprocessor buffers** using only instances of the modules belonging to the architectural definition"*

Fig. 6 shows an outline of the synthesis CAD system of CATHEDRAL II.

In this system one can roughly distinguish the following (not entirely independent) parts :

-specification and verification by simulation (emulation)
-partitioning in independent subprograms
-mapping subprograms on datapath structure and generation of associated microcode.
-optimization loop
-interprocessor buffer optimization and generation
-supervising controller generation.

These different parts are strongly related to the architectural definition and therefore the system described below is a **rule based** system calling procedural optimization **controllable by the designer** by the **pragma** concept. We will describe these parts in more detail below.

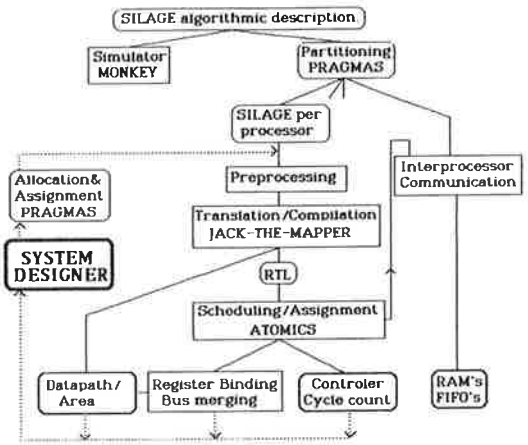

Fig. 6 : Outline of the synthesis CAD system of CATHEDRAL II.

3-2-2 : The specification language and the pragma concept : SILAGE.

The specification of a DSP algorithm for CATHEDRAL II consists of two parts :

- **The first part** describes the behavior by means of the applicative language SILAGE [Hil85]. The main idea of SILAGE is to capture the signal flow graph of the algorithm. It does not contain any structural information and does not enforce any degree of concurrency. It does not contain information about the implementation of the control flow. In SILAGE signals can be reals or fixed point, finite wordlength types which are infinite arrays in time. Relations between variables are expressed by explicit, time discrete simultaneous equations of which therefore the ordering is irrelevant. Provision is made for the delay operator z^{-1} - @ and operators exist for decimation and interpolation. Loops can be used as a compact notational

format but they do not imply any control flow.
The SILAGE description of the algorithm can be considered as a system specification and should therefore be debugged and verified in a rigourous way. For the system designer, SILAGE also serves as the algorithm development medium. Hance, efficient simulation or even emulation tools are of prime importance. Therefore a SILAGE simulator providing simulation both at floating point and fixed point precision is provided in CATHEDRAL II.

```
part 1    #define  W num<20,0>
          #define  a11 0.625

          func  main(In:W) Out: num=
          begin
              Section2 = biquad(Section1,a21,a22,b21,b22);
              Section1 = biquad(In,a11,a12,b11,b12);
              Out = FirstOrder(Section2,a31,b31);
          end;
          func biquad(input,a1,a2,b1,b2: num) : num =
          begin
              State = input + W(b1*State@1)-W(b2*State@2);
              return = State + W(a1*State@1) + W(a2*State@2);
          end;

part2     % assign (biquad(In,_,_,_,_),(_),alu,2);
          % assign (biquad(Section1,_,_,_,_).(_),alu,3);
          % assign (FirstOrder(),(_),alu,4);
```

Fig. 7 : Silage description of a 5th order PCM filter

Fig. 7 shows an example of a SILAGE description of a 5th order PCM filter. This example illustrates that SILAGE only contains behavior and does not impose any structure. The structural realization depends entirely on the compilation of SILAGE code. In the interactive concept of CATHEDRAL II however, the designer is able to enforce structural decisions at the highest level.

- **The second part** of a description indeed contains optional directives for the compilation. These directives are called **pragmas**, through which the designer is able to give (incomplete) structural hints. As is also indicated in Fig. 5 three types of pragmas are provided :

***Pragmas for partitioning** the SILAGE functions over processors : "*biquad, processor,2*" forces the function "*biquad*" to be implemented on processor 2.

***Allocation** pragmas : "*alloc(alu,3)*" allocates 3 ALU's in the datapath.

***Assignment** of an expression to an EXU instance :

" *% assign (biquad(input,_,_,_,_).(b1*_),alu,2)*" forces all multiplications involving variable b1 local to any function call of "*biquad*" with first argumant "*input*" to be executed on ALU instance 2.

The splitting of the description in two parts is of prime importance , because it gives the system designer the possibility to explore different implementations of the same behavioral description. The verification of the correctness of the behavior can be done once by the high level simulator while the exploration of the design space is carried out by means of the pragmas whereby the behavior is guaranteed to be correct by construction.

3-2-3 : The mapping problem : allocation and assignment : JACK_THE_MAPPER

Once the system designer has indicated by partitioning pragmas how the SILAGE functions are to be partitioned into processors, the **mapping** problem consists in the **allocation** of the EXUs of the **datapath** of each processor and the **assignment** of the SILAGE operations to primitive operations on these EXUs. The latter results in a translation of the SILAGE code into a set of unordered **register transfer** instructions to be run on the datapath hardware.
In CATHEDRAL II this is achieved by a rule based program called JACK_THE_MAPPER. It uses a mixture of automated tools and user interaction based on the pragmas in order to solve this extremely complex optimization problem.
It is our experience that a system designer has often a good insight in the computational bottlenecks of an algorithm. Therefore, he/she is well capable to estimate the required amount of parallelism and the acceleration units needed. The most time consuming and error prone job is not located in the allocation task but in the optimum operator assigment, the controller generation, the optimization of register usage and bus count.
First, as in any general compiler system, in a **preprocessing** step , JACK parses the SILAGE description, determines the datatypes of all signals, performs a number of local transformations (e,g. elimination of common subexpressions, optimal ordering of commutative operations , function expansion etc...) and sets up the **data precedence graph** containing only **primitive** SILAGE operations which can be executed by the EXUs. If present, also the user defined pragmas are read in.
The proper **translation** step is performed by an architecture independent **expert system shell** written in PROLOG which contains architecture dependent **rules**. If no pragmas are present then JACK will always first try the **cheapest** silicon area **allocation first**. (Usually an ALU-ACU solution). If after scheduling (see 3-2-4) it is found that too many cycles are needed, (see also Fig. 6) the designer can state **pragmas** e.g. to add accelerator EXUs to the datapath , to add parallel EXUs or to repartition the algorithm.
Basically the task then consists of **assignment** of the primitive SILAGE operations to the EXUs, to define the bus structure and to assign intermediate variables to register files and background memory. This task can be divided into a translation and a number of optimization subtasks.
The translation step transforms behavioral primitives into architectural primitives. This step is of extreme importance since it will determine how efficient the architectural properties are exploited. In order to cope with architectural changes and expansions, this tool has to be flexible and expandable by system architects. This is the reason of the choise for a declarative rule based system.
These rules may be straightforward for a simple addition on an ALU but others are far more complicated e.g. rules for multiplication (parallel, parallel-serial, constant*variable, CSD add-shift...) , division on DIV or ALU, algorithmic delays, array- or loop operations, floating point operations, decimation and interpolation etc...
A second set of rules implements the interconnection strategy. It generates the necessary busses, input multiplexers and tri-state output buffers.
The current rule base contains some 120 rules but it is in continuous expansion and modification as our experience grows. The implementation of the **inference mechanism** is done in a

demand driven way, i.e. the mapping starts from the output of the precedence graph and works its way backwards, driven by the variables needed to compute the currently mapped operation. This leads to a fast execution time which is of prime importance in an interactive CAD system. This flexibility has led us to the development of a user-friendly knowledge acquisition system to ease the introduction of new rules.

3-2-4 : Scheduling and optimization : ATOMICS

As a result of the above described translation step, the SILAGE description has been transformed into a datapath **structure** and a register transfer (RT) description of the algorithm.
In CATHEDRAL II the RT-language produced by JACK has a form which relates directly to the architecture. As seen in the example statement below, in a first part (before the | separator) the (conditional) transfer of variables between registers is given while the second part describes the EXU type on which the operation is executed, the mode (**symbolic microcode**) of the EXU, and the **bus-mux** on which the transfer takes place :

if c **then** s : reg1 <- a : reg2, b : reg3 | alu = add, bus2=s, mux1 = bus2;

In the RT description, no timing is assigned to the operations and a number of the hardware assignments (e.g. the binding of an operation to a particular ALU instance) are also left open. As a result still a number of **procedural optimization** tasks have to be performed which can be summarized as follows :

-find the optimal ordering of RT-operations on the time axis such that the execution of the algorithm takes a minimum number of cycles (**scheduling**).

-bind the undefined assignments so that the allocated hardware is used in an optimal way (best load balancing leading to minimum number of cycles).

-**register binding** and **bus merging** : during mapping each intermediate variable has been assigned to a register in the EXU register files and each transfer is originally executed over its own bus. Obviously not every variable needs to be stored in a register location during a complete program execution (**frame**). In a **lifetime analysis** the minimum number of register is searched to store all variables during the minimum time they are needed during a frame.
Bus merging similarly consists in a reduction of the number of busses by reusing them for different transfers during a frame. Alternatively one can also exchange a number of cycles to obtain a maximum **merger of busses** , which of course will require a **rescheduling** of the RTs.

A very powerful heuristic optimizer-scheduler program ATOMICS [Goo87] has been developed capable of scheduling repetitive programs with nested loops taking I/O constraints into account. Every RT statement can be considered as a node in a graph. The statement is executed on a potentially to be assigned EXU under a required mode of operation. If the statement is conditional then the condition is computed from **status** bits in the datapath which have to pass through the controller **pipeline** before the condition bit is computed.
Scheduling can then be stated as finding a **level labeling** of the nodes in the graph such that :

-the number of labels (potentials) is minimal and :
-the precedence ralations are respected and
-there are no resource allocation conflicts : i.e. no two different modes of an EXU are scheduled in the same potential (have the same label) and
-the controller pipeline- and the I/O-timing constraints are satisfied.

should be noted that scheduling is an N-P complete hard problem due to the occurrence of resource allocation conflicts. In ATOMICS, techniques similar to the ones used in operations research have been adapted to the problem of nested repetitive programs. Due to the heuristic nature of the problem only near optimal solutions are obtained. However extensive investigations have shown that in nearly all cases either optimal or very near optimal solutions have been obtained. For more details see [Goo87].

3-2-5 : controller generation and interprocessor communication

Once scheduling, register optimization and bus merging have been done, we know the sequence of synbolic microcode. It is then a simple task to translate the symbolic microcode into a **symbolic state diagram** which is basically a large symbolic case statement.
Then, after **state assignment** and using the **boolean microcode** of each EXU we can translate the state diagram into the PLA planes and the microcode ROM content of the processor controllers. The layout of these elements is obtained using the PLASCO [Bar85] environment.
Finally, it is then possible to synthesize the interprocessor communication hardware and to derive the structure and contents of the central controller.
The tasks of this tool, which is currently under development, are to select the cheapest communication protocol(FIFO, RAM with single or double buffering), to dimension the buffer arrays and to determine the exact timing of the needed control signals. It must be mentioned that the selection of a certain protocol can result in a number of extra constraints on the processor timing or hardware, so that a reiteration on the processor synthesis process may be necessary.

3.4 Application examples

1. The PCM filter.

As an illustration of CATHEDRAL II we synthesize the PCM filter described in the SILAGE code in Fig. 7. In a first attempt, a single ALU implementation is automatically generated by JACK. It is shown in Fig. 8a. It consists of a RAM for I/O , one ALU to perform all arithmetic operations and one ROM (rom_ctrl) forming the link to the controller over which immediate adresses can be fetched. (Note that the multiplications with constants have been expanded automatically in a minimal number of add-shifts on the ALU). In this first solution, the number of machine cycles found by ATOMICS is **36**. This is the fastest possible execution time using the minimum number

of EXUs. However, when using the bus merging algorithm we get after ca. 1 minute of CPU (Apollo DN3000) a new solution as in Fig. 8b where all busses have been merged into a single one at the expense of three additional cycles (39).

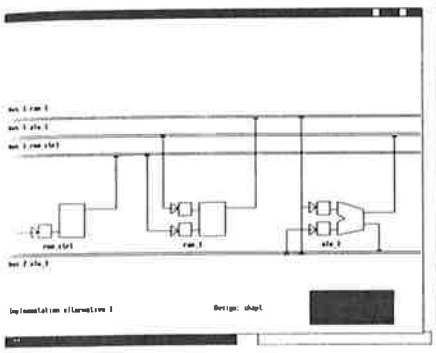

Fig. 8a. 1 ALU- 4 bus solution (36 cycles)

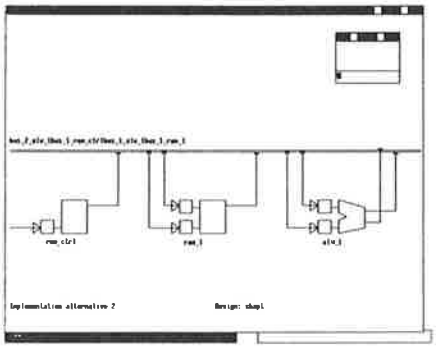

Fig. 8b. 1 ALU- 1 bus (bus merger, 39 cycles)

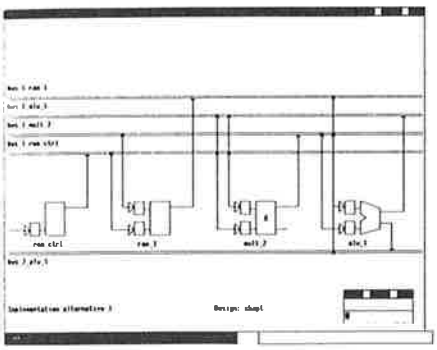

Fig. 8c. ALU-MPY solution pragma(18 cycles)

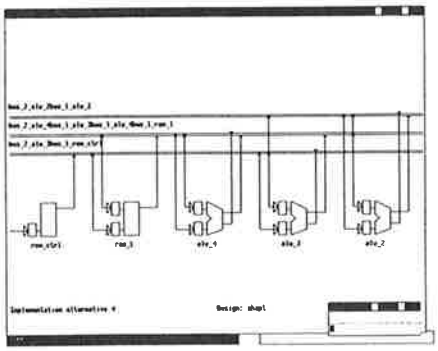

Fig. 8d. Repartitioning over 3 ALUs (20 cycles)

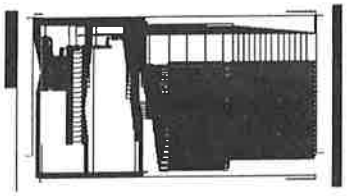

Fig. 9a. Generated layout of Fig. 8b

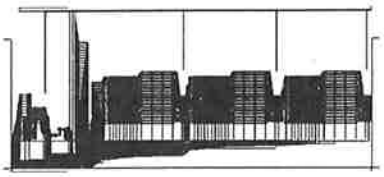

Fig. 9b. Layout of Fig. 8d.

When the designer is satisfied with this result he/she can call the module generators through the linker environment to generate the necessary instances of the modules, which then can be interactively or automatically placed and routed on the floorplanning tool. For the single bus

solution, the floorplan is shown in Fig. 9a. The area of the filter is 6.7 mm^2 in a 2.5 µm CMOS double metal process.
On the other hand if the speed is not sufficient the designer can quickly generate new solutions by adding pragmas to the original SILAGE description. The pragma:

pragma "assign(_*_,mult,1)"

states that all multiplications should take place on a single multiplier. The solution is shown in Fig. 8c shows the new datapath (5 min CPU). ATOMICS shows that it executes in 18 cycles which is about twice as fast as the original solution.
Finally Fig. 8d shows the result of a **repartitioning** pragma as given with Fig. 7. Hereby the designer assigns one ALU per filter section which leads to a 20 cycle solution. Fig. 9b gives the layout after module generation and place and route.
Notice from this example how such a synthesis system allows for a very rapid scan through the design space since, in manual design, each debugged microcode redesign costs about one week of effort instead of ca. 5 min. CPU in CATHEDRAL II.

2. Adaptive interpolator for compact disc error correction

To demonstrate a somewhat more representative example for the complexity of algorithms that can be handled in CATHEDRAL II, the example of an adaptive interpolator for the correction of burst errors in compact discs [Vel83] will be discussed. The algorithm includes the computation of a 512*512 correlation matrix, the inversion of a 51*51 Toeplitz matrix using the Levinson-Durbin algorithm, the computation of the interpolation coefficients and the inversion of a full 16*16 matrix. Initially, a four processors solution was tried. A study of the complexity of the processors and the amount of interprocessor memory requirements showed however quickly that a single processor with a complex datapath as shown in Fig. 10 was preferrable.

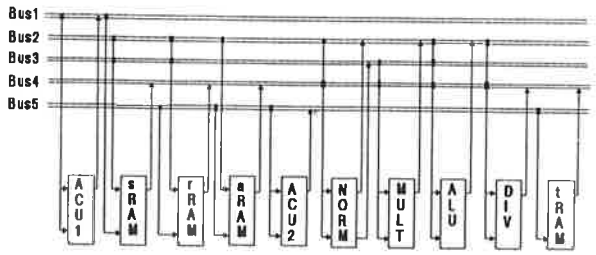

Fig. 10. Single processor solution for adaptive interpolator. All EXUs except COMP are used.

The ATOMICS scheduling of the RT description reveals that the complete algorithm can be executed in 77000 cycles on this datapath. This easily fits into the allotted time frame of 11.6 msec. This example shows clearly the power of conceptual CAD at the architectural level.

4. Conclusions

We believe that, due to the the evolution towards micron technology and the abundance of system level designers, future ASIC design will move to complete **perceptual electronics** systems on a chip. However, due to the scarcity of silicon level designers silicon in ASICS must be reusable and must provide modules at the EXU level. This will inevitably lead to the "meet-in-the-middle" design concept. Since already today good solutions for **structural design** at the **block place-and-route** level do exist, the bottleneck is at the **conceptual design** level.

In this paper we have introduced the principles of such system and we have illustrated it on the hand of the prototype CAD system CATHEDRAL II for the design of complex DSP systems in the audio spectrum. We have demonstrated the necessity of the existence of a number of such systes each adressing its own **application field** by an appropriate **flexible but concize architectural definition** supported by powerful, technology updatable module generators.

It is our believe that for the wide class of synchronous machines it must be possible to design an architecture independent expert system shell whereby the particular architectural features are added as **declarative knowledge**. This leads to the fact that future ASIC CAD will consist of a software shell whereby the competition between system houses will be back where it always belonged in electronics : **system and architectural knowledge as the key to the future !.**

It is also our experience that the way to optimally exploit it can not be programmed very well into CAD systems but we have seen that system designers prefer the **pragma based interactive** idea far above automated synthesis systems on which they have no impact if they don't like the solution. The time savings really comes from the possibility to quickly evaluate the quality of design alternatives whereby the difficult bookkeeping and optimization tasks are taken care of by the CAD system.

We must realize that we are only in the beginning of this era. Indeed a lot of work will be necessary to adress the problem of high performance video- and image processing systems, the analog interfaces, automated synthesis of asynchronous communication and insuring testability of compiled silicon.

Last but not least we have to mention the problem of providing real time **emulation** capability to system designers using a high level specification language.

It is our experience that in the DSP area system designers want to experiment, if possible, in real time with their algorithms. Simulation on a traditional workstation is very awkward since reasonable test cases involve millions of samples. How this problem must be solved is still an open question which will depend strongly on the availability of powerful audio and video processors which can be assembled into "general" purpose programmable multiprocessor systems onto which a specification language such as SILAGE can be scheduled, perhaps in much the same way as it is now scheduled onto a dedicated architecture.

5. References

[Bar85] M. Bartolomeus et al. : " PLASCO : a procedural silicon compiler for PLA based systems", Proc. IEEE CICC-85 conference, pp. 226-229, may 1985.

[Ben87] J. Benkoski et al. : "Efficient algorithms for solving the false path problem in timing verifiers", Proc. ICCAD conference-87, Sta Clara, nov. 1987.

[Bla85] T. Blackman et al. ,: "The SILC silicon compiler : language and features", Proc. 22nd design automation conference, pp. 232-237,June 23-26, 1985.

[Bra85] R. Brayton et al.,: "The YORKTOWN silicon compiler", Proc. 1585 ISCAS conference, pp. 393-394,June 1985.

[Bro87] J. Brouwers et al. : "Integrating the electronic design process", VLSI systems design, june 1987, pp. 38-50.

[Cat86] F. Catthoor et al. :"General datapath, controller and communication architectures for the creation of a dedicated multiprocessor environment", Proc. IEEE ISCAS conference, May 1986, pp. 730-731.

[DeM86] H. De Man et al. : "CATHEDRAL II : A silicon compiler for digital signal processing ", IEEE Design &Test of computers", Dec. 1986, pp. 13-25.

[Den82] P. Denyer et al : " A silicon compiler for VLSI signal processors", Proc. 1982 ESSCIRC conference, sept 1982.

[Goo87] G. Goossens et al. : "An efficient micro-code compiler for custom multi-processor systems" , Proc. IEEE ICCAD conference nov. 1987.

[Hil85] P. Hilfinger, : :A high level language and silicon compiler for digital signal processing", Proc. IEEE CICC conference, Portland, May 1986, pp. 213-216.

[Jai86] R. Jain et al.:" Custom design of a VLSI PCM-FDM transmultiplexer from system specs to layout using a CAD system", IEEE J. Silid State Circuits, Vol. SC-21,1,,Feb.1986, pp. 73-85.

[Kow85] T. Kowalski et al. : "The VLSI design automation assistant : What's in a knowledge base ?",Proc. 22nd design automation conference, pp. 252-258.

[Mar86] P. Marwedel, "A new synthesis algorithm for the MIMOLA software system" , Proc. 23rd design automation conference , pp. 271-277.

[Rap86] A. Rappaport, : "Technological evolution of the semicustom design process", VLSI semicustom design guide by CMP publications Inc. Summer 1986, pp. 40-44.

[Ros85] W. Rosenstiel et al.,:"Synthesizing circuits from behevioral level specifications", Proc. of 7th CHDL conference, pp.391-403, Tokyo, August 1985.

[Six86] P. Six et al. :"An interactive environment for creating module generators", Proc. ESSCIRC-86 conference, pp.65-70,sept. 1986.

[Sou83] J. Southard, : " Mac-Pitts : an approach to silicon compilation", IEEE computer, Vol. 16, pp. 74-82, dec. 1983.

[Szy86] T. Szymansky : "LEADOUT : a ststic timing analyzer for MOS circuits", Proc. Int. Conf. CAD, Nov. 1986, pp. 130-133.

[Vel83] R. Veldhuis et al. : "Adaptive interpolation for digital audio signals : an integer implementation ", Philips Internal Report, 1983.

[Vdm86] E. Vanden Meersch et al. :"SLOCOP : a timing verification tool for synchronous CMOS

logic", Proc. ESSCIRC-86, Delft, pp. 205-207.

[Wes81] N. Weste : "MULGA-An interactive symbolic layout system for the design of ICs", The Bell system Tec. J., Vol. 60, 6, July-Aug 1981, pp. 823-857.