GAUT: An Architectural Synthesis Tool for Dedicated Signal Processors

E. Martin, O. Sentieys, H. Dubois, J.L. Philippe

ENSSAT - LASTI, University of Rennes I
6 Rue de Kérampont 22300 Lannion, France

ABSTRACT

This paper attempts to describe a pipeline architecture synthesis tool dedicated to signal processing applications. This approach relies on the use of a design strategy and of a generic architectural model, using optimized control of resources. GAUT[1] takes a VHDL description of an application as input, and generates the optimal structural and functional VHDL description of a dedicated architecture. The results obtained by GAUT are intended for an application in acoustic echo cancellation

1 Introduction

Technological advances force today's designer to consider a new work method. This method consists of automating the materialization of the component, or of the system which enables the execution of a software solution to a particular problem. One may therefore, methodically exploit the parallelism of the application in order to satisfy the algorithm execution time constraints.

Architectural synthesis allows the search for the optimal architectural solution relative to the specified constraints. To be optimal, the synthesis must rely on a design method which takes into account the specificity of the application domain. We have focused on the domain of Signal Processing in real time and we have formalised a specific method of conception for this type of application. This method is original on several points: - to avoid feddbacks in the design strategy, the units which are not yet synthesized are considered transparent; - to optimize the use of the treatment units a multi-phase time clock is used; - Data Flow Graph transformations dedicated to Signal Processing Algorithms is done.

2 Architectural Synthesis in Signal Processing

Dedicated architectural synthesis automates the different algorithm materialization steps (figure 1). The loop which appears in this cycle is particularly critical in real time applications [1]. Generally, it is necessary to open this loop either by adopting an appropriate strategy ('meet in the middle' design [2]), or by defining, in minute detail, the architecture which guarantees the validity of the synthesis (top-down design). In the domain of Signal Processing, numerous architectural synthesis tools are described in a variety of publications: SPAID [3], MIMOLA [4], HAL [5], EASY [6], CADDY [7], FIRST [8], SHEWA [9], YSC [10], AMICAL [11], OSYS [12]...

These tools generally integrate a descending approach, which is hierarchized into three primary steps:

- exhibition of the parallelism;
- materialization and optimization of a dedicated architecture, via the definition of the different functional units of which it is comprised. These tasks process NP-complete problems, and necessitate the definition of an **architectural model,** an ordering of materialization tasks, thus defining a **design strategy** of the architecture and the implementation of **optimization techniques;**
- generation of the description of the synthesized architecture, which is adapted to logic synthesis tools, and other logic CAD tools, that are currently available.

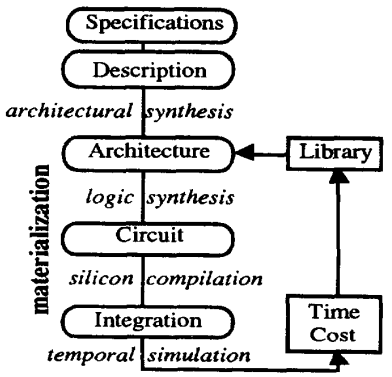

Fig. 1: Materialization Cycle of an Algorithm

[1] GAUT is partially supported by CNET Meylan

0-8186-4350-1/93 $3.00 © 1993 IEEE

3 GAUT: A Synthesis Tool Dedicated to Signal Processing.

3.1 Introduction

GAUT is a pipeline architecture synthesis tool, dedicated to Signal and Image Processing applications under real time execution constraints. It works from the behavioral domain to the structural domain. The constraint which must be satisfied is the calculation frequency, which may be linked to a sampling frequency. The processing is described with the assistance of behavioral VHDL. GAUT accepts the description of a formal library of operators. The compilation phase, enables the extraction of the parallelism of the application and some transformations. The DFG obtained is synthesized according to a generic model of core of Digital Signal Processor. Finally, the synthesis leads to the generation of a structural and functional VHDL description of the designed architectures.

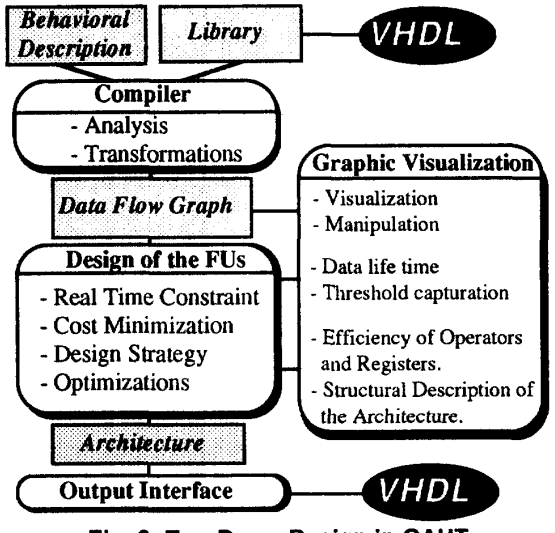

Fig. 2: Top-Down Design in GAUT

3.2 Design Strategy

The design stage which we follow is 'top-down', and our main contribution consists in defining the conception strategy. This strategy has to organise sequentially the design of the different Functional Units (FU) which are to be integrated in the architecture. When computing a Functional Unit, the chosen strategy has to take into account only the constraints of the application itself and of the Functional Units which have already been computed. The other units which are to be computed are considered as totally transparent. That is to say, that these units are supposed to satisfy the constraints of the unit currently synthesiz&red without disturbing its operation.

So this strategy organises the conception without any feedback on the functional units in this way:

- Design of the most complex Functional Units: these units undergo the strongest constraints of the application and represent a preponderant complexity of the architecture to be created. The quality of the created architecture depends principally on their optimization.
- Design of the most autonomous Functional Units: the suppression of the interaction constraints between units involves a reduction of architecture optimization. Thus, the first Functional Units to be designed, are those which undergo the least interaction constraints from the other units.

First of all, GAUT synthesizes the Processing Unit, then the Memorization Unit and the Communication Unit (which has not yet been integrated into the tool). The Control Unit is simply described in order to be synthesized by a finite state machine design tool.

During the design of the Processing Unit, GAUT initially processes arithmetic operators and targets their maximum use. Then come the registers and memory banks, which are part of the Memorization Unit. Because of no feed-back in the conception, the registers optimization, which is done before the memory optimization, is based on prediction techniques. The communication paths will then be optimized, followed by the optimization of the address generators of the memory banks dedicated to the application being considered (see figure 3).

The optimal design of a processing unit integrates the following tasks: resource allocation and selection, operation scheduling, and the assignment of operations to the various operators. First, GAUT executes the allocation task, and then simultaneously executes the tasks of scheduling and assignment.

Allocation consists of implementing the types and the number of operators which satisfy the average parallelism of the application extracted from the DFG dated by a As Soon As Possible scheduling. The average parallelism is calculated separately for each type of operation and for each pipeline slice N of the DFG, comprising the set of the date operations belonging to $[N.Tr, (N+1).Tr]$. Tr is the time constraint.

During the scheduling phase, a supplementary pipeline slice may be created if necessary, subsequent to operation

scheduling on the previously allocated operators.

Operation scheduling is a list scheduling with mobility heuristic which also depends upon the availability of allocated operators. The operations are scheduled as soon as the operator is available (allocated for a pipeline slice covering the considered date, free of all execution, and free during the time clock period in progress). The optimal **assignment** of a candidate operation on an available operator responds to the minimization of inter-connections between operators. The pipeline control of each operator is managed by a complementary priority on assignment. When an operator is allocated, but as yet not used, its use is primarily inferior to that of an operator already utilized. Furthermore, if a candidate operation has a positive mobility, then the scheduling is delayed. Finally, if an operator allocated from the beginning of the period is never used during the entire period, its allocation interval is delayed for one clock period.

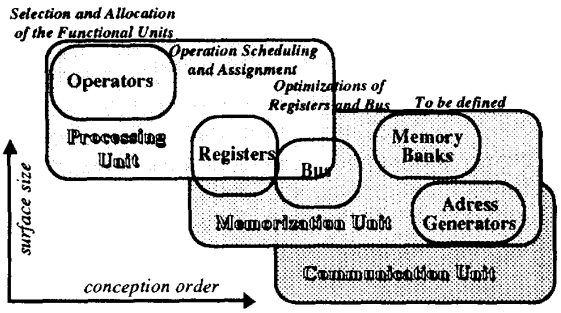

Fig. 3: Design Strategy

3.3 Input and Compilation Language

A VHDL input language at a behavioral level was retained. The total determinism in the DFG to be executed lead to the code transformation and the optimization of the description:

- Fixed iteration loops (FOR or WHILE) are unrolled.
- Variable propagation.
- Conditonal assignments are resolved by creating multiplexed values.

The description of the library used in the synthesis will be performed with the assistance of a generic VHDL, whose the grammar is shown in figure 4. A generic library can be parameted by time and cost to become a technolgy driven library.

The Gaut library permits the use of multifunctional FUs (such as ALU) and the use of pipeline operators. This allows to a hierarchical design startegy by using operators resulting from a previous synthesis.

The implemented operators may have the following characteristics:

- multiplicity of operations for a given operator [14];
- multiplicity of operators for a given operation. This case is typical of the equivalence of function for different costs, or the equivalence of format;
- multiplicity of characteristics for a class of operators.

<library>::	PACKAGE <ident> IS (<generic>)* END <ident>
<generic>::	COMPONENT <oper> GENERIC '(' <generic>)* ')' ; END COMPONENT
<generic>::	<charac>: INTEGER := <integer> | FUNCTION:<type>:=<function>(, <function>)*
<charac>::	AREA | PIPE_STAGE | DELAY | INPUTS | N_BIT | LATENCY | ACCESS_TIME | <ident>
<oper>::	REGISTER | MULTIPLEXER | MEMORY | DEMULTIPLEXER | BUS | <ident>
<function>::	ADD | SUB | MUL | DIV | LT | LE | GT | GE | EQ | NE | MUX | AND | ••• | <ident>

Fig. 4: Grammar of the library

3.4 Modelizations of the Generic Architecture

GAUT is a pipeline architectural synthesis tool which allows the full definition of the core of a DSP which is dedicated to a specified application. The figure 5 illustrates the generic model of the structure of the synthesized architectures.

The topology of the Processing Unit model is based on an elementary cell which includes several multiplexers, registers, and demultiplexers interconnected for the requirements of the application around an operator. This cell is dedicated to the processing to be executed. These cells are attached to the operators implemented during the allocation phase, in quantities sufficient to satisfy the constraints. After the scheduling of the operations and optimization of the registers, a cell may include several registers or may share a register with other cells.

The various cells communicate via a parallel multi-bus network. The topology of the parallel multi-bus network is determined after the optimization of the data transfers, which attempts to reduce the multiplexers implemented in the network.

The control model integrates two features: the use of the pipeline in order to control the operators and the registers; and the use of a multi-phase timeclock for controlling the events. The number of timeclock phases is not determined in advance, and depends on the results of the scheduling task, which dates the execution of the operations in order to provide a maximum use of the operators. GAUT

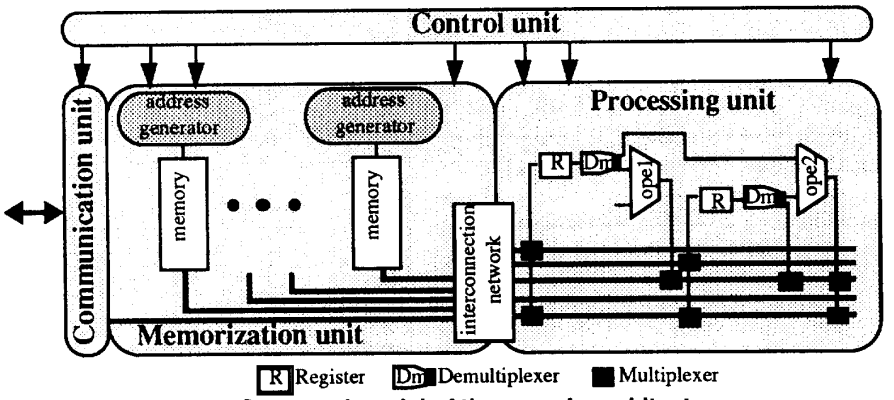

Fig. 5: Structural model of the generic architecture

integrates scheduling technique which enables the user to control the number of timeclock phases to be implemented.

We have to consider four types of duration:

- the timing of each cell: *Tfunct*
- the cycle time of the control clock: *Tcycle*
- the desired execution time for the algorithm, that is to say the real time constraint: *Tr*
- the propagation time of the architecture: *Tp*.

Because of the pipeline, *Tp* can be greater than *Tr*. These times verify the following inequation:

$$Tfunct \leq Tcycle \leq Tr \leq Tp.$$

Each cell works on a set of data of the algorithm during a time interval *n.Tcycle*, *(n+k).Tcycle*, with *k=Tr/Tcycle*, *n* depending on the allocation.

The use of the multi-phase time clock permits to optimize the efficiency of each cell. The efficiency is defined as the ratio between the time when the cell is really active and the total duration of its allocation.

We now show by an example, the advantages of the multi-phase clock by calculating the efficiency of cells controlled by a mono-phase clock and by a multi-phase clock: let us consider two cells with two different timings: T_{f1} and T_{f2} with $T_{f1} < T_{f2}$ and $T_{f2} / T_{f1} = r$ (r is truncated).

The use of the mono-phase clock leads us to choose either one mono-cycle control clock with $Tcycle=T_{f2}$ or the other multi-cycle control with $Tcycle=T_{f1}$.

The two efficiencies of the architecture are:

$$\frac{N.(T_{f1}+T_{f2})}{2.Tr} \text{ or } \frac{N.\left(T_{f1}+\frac{T_{f2}}{r+1}\right)}{2.Tr} \text{ where} N = \frac{Tr}{Tcycle}$$

The use of a multi-phase clock permits us to drive the first cell *N1* times per cycle and the second cell *N2* times per cycle. The optimum cycle time is either $Tcycle=N1.T_{f1}$ or $Tcycle=N2.T_{f2}$. The efficiency is:

$$\frac{k.(N1.Tfonc1 + N2.Tfonc2)}{2.Tr} \text{ where} N1 \approx r.N2$$

This efficiency is always greater than the one with the mono-phase clock. The maximum lost time is equal to the GCD of the two function time.

The model for the generic Memorization Unit permits us to take into account the functional constraints of the application domains and also the functional constraints to ensure the transparency of the data access for the processing unit.

Registers are a particular case because they are both in the model of processing unit to storage data during processing as well as for the intermediate calculation data, and in the model of the storage unit to store certain types of variables.

The memory banks are used to store data, or constants, which have a relatively long life time. Their number is determined in order to respond specifically to each application.

3.5 Registers and Busses Optimization

Because of no feed-back in the design, the registers optimization has to be done during the conception of the processing unit. The choice of the location of a variable in a register or in memory, has to be done according to the minimization of two contradictory cost criteria:

- the cost of a register is higher than the cost of a memory point.
- the cost to access data in a register is lower than to access data in memory (because of the necessity to compute the address).

Two criteria are used to chose the location of memorisation of the data:

- A variable whose life time is inferior to a locality threshold is stored in a register;

- The location of memorisation depends on the class of the variable.

These are classified into four categories: - temporary processing data (declared or undeclared); - constant data (read-only); - recursive data (which serves to express the recursivity of the algorithm to be synthesized, via their assigment after having been utilized); - input/output data declared in port (these belong to the Communication Unit). An intermediate type of data is a result which does not correspond to either a declared variable, signal, or port.

The optimal storage of a given data element depends upon its declaration and its life time. It can be in either in memory or register except for the I/O ports. The remaining difficulty lies in selecting an optimal locality threshold which results in minimizing the cost of the storage unit. The synthesis tool leaves the choice of the value of the locality threshold up to the user. In order to help the user, GAUT proposes a histogram of the life time of the variables, normalized by the utilization frequency, which is calculated using the scheduled DFG.

During or after the assignment task GAUT optimizes both registers aund busses by adding multiplexers and 3-state demultiplexers or by time and space bus partitionning. This is done with Branch&Bound techniques using heuristic cost calculation.

3.6 Output Interfaces

The output interface describing the architecture generates two files: one for the structural description of the processing part of the synthesized architecture; and the other for the functional description of the control part of the synthesized architecture [16]. These two files are in the VHDL format. The processing unit is therefore described by an entity describing the ports, which enable its connection to the control part as well as to receive data and supply the results; and by an architecture describing it in the form of an instanciation of library components.

The control unit will be described by an entity describing the ports which allows its connection to the processing part as well as to receive a timeclock, and by an architecture describing it in the form of a process with the time clock in a sensitivity list.

4 Results of the Synthesis by GAUT

4.1 Results on a Benchmark for Synthesis

We synthesized an elliptic filter [15] and our results are compared with those of other CAD tools. Three libraries have been used and prove the avantage of controlling the processing unit by a multi-phase clock when the operators have very different functional times.

Register	Multiplexer	Adder	Multiplier
cost = 200	cost = 80	cost = 400	cost = 2400

	Library 1	Library 2	Library 3
Adder time	100 ns	100 ns	80 ns
Multiplier time	200 ns	160 ns	200 ns

Tr real time	HAL		SPAID	GAUT Lib 1	GAUT Lib 2	GAUT Lib3
1700ns	4 Mult 3 Add 12 Reg	EASY 3 Mult 3 Add	2 Mult 3 Add 17 Reg	3 Mult 3 Add 12 Reg	2 Mult 3 Add 14 Reg	2 Mult 2 Add 12 Reg
1900ns	2 Mult 2 Add 12 Reg	CADDY 1 Mult 2 Add	1 Mult 2 Add 19 Reg	2 Mult 2 Add 11 Reg	1 Mult 2 Add 11 Reg	2 Mult 2 Add 12 Reg
2100ns	1 Mult 2 Add 12 Reg			1 Mult 2 Add 11 Reg	1 Mult 2 Add 11 Reg	1 Mult 1 Add 11 Reg

Fig. 6: Results on the 5° order elliptic filter

4.2 Results in Acoustic Echo Cancellation

The problem of acoustic echo cancellation is apparent in applications such as teleconference or hands free telephony, which may be either fixed or radio mobile. This issue implicates complex Signal Processing algorithms, which are generally adaptative. Some adaptive algorithms are particulary complex because they require thousands of coefficients and a sampling period of 16 KHZ (62500 ns).

The results of the methodological approach using a synthesis tool were compared with those of manual design. We present the enhanced performance of the architectural synthesis tool for the rapid prototyping of the signal processing algorithms. Currently, specialists in the domain estimate the complexity of their algorithms by a simple count of the operations to be executed (primarily multiplications and divisions). However, this estimation is incorrect once it becomes necessary to use the parallelism, because a certain correlation may be noticed between algorithm regularity and the usage rate of the different resources implemented in the architecture.

Two signal processing algorithms have been retained for use in acoustic echo cancellation. The first algorithm is a 1000-point transverse filter adapted by the stochastic gradient, or LMS filter. The second filter is a 1000-point lattice filter adapted by a 30-cells.

Gaut generates the architecture of the figure 7 for the

LMS filter on 1000 points. The operators are used with a rate of 87% and the registers with a rate of 70%.

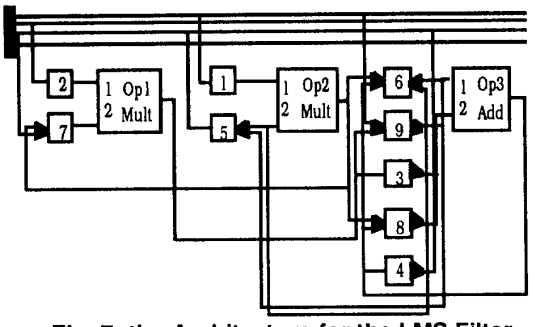

Fig. 7: the Architecture for the LMS Filter

Finally, the interpretation of architectures, with the assistance of the format described in paragraph 3.6, is subsequently generated in order to interface with data path generator type tools, and machine state synthesis tools.

A comparison can be done between results for different sizes of filters (for the LMS and for the lattice filter). This comparison permits us to know the real cost of the processing unit in each case; what is important is that these costs are rather different than those calculate from the number of operations [17].

Number of points	TRANSVERSAL	LATTICE (30 cells)
800	2 Mult, 1 ±, 9 Reg, 10 Mux **1 / 0,8**	3 Mult, 1±, 19 Reg, 16 Mux **1,62 / 2,55**
900	2 Mult, 1 ±, 9 Reg, 10 Mux **1 / 0,9**	3 Mult, 1±, 25 Reg, 26 Mux **1,92 / 2,85**
1000	2 Mult, 1 ±, 9 Reg, 10 Mux **1 / 1 reference**	3 Mult, 2±, 33 Reg, 33 Mux **2,27 / 3,15**

Fig. 8: Relative Cost to the LMS Filter
area complexity / operation complexity

In the preceding table, the synthesis results are noted, as well as the comparative variations of architectural costs and complexities estimated with the number of arithmetic operations of these two filters, when the number of points changes (text in bold type). The synthesis tool can also help to chose the best algorithm for the particular constraints of an application.

5 Conclusion and Perspective

The domain of architectural synthesis in Signal Processing is undergoing numerous developments. It seemed vital to formalize not only the optimization techniques, but also the design strategies and generic architecture models which were employed. GAUT integrates an effective design strategy and an original generic architecture model, which allows us to achieve the optimal use of the resources implemented for a given application. The acoustic echo cancellation application illustrates the contribution made by a synthesis tool in the framework of rapid algorithm prototyping.

Work is currently underway concerning the synthesis of other functional units which are not as yet defined, allowing the design of complete processors dedicated to considered applications. Attention is also being focused on the problem of optimal selection of resources to be allocated, in the context of the utilization of full component libraries.

[1] H. De Man "Silicon Compilation for real Time Signal Processing systems" Tutorial on high Level Synthesis", EDAC, Galsgow 12-15 march 1990.

[2] B. Bowen & W. Brown "systems Design" Edition Prentice Hall, 1985.

[3] B. S. Haroun, M. I. Elmasry "Architectural synthesis for DSP silicon compilers", IEEE Transactions on computer-aided design, Vol 8, No 4, April 1989.

[4] P. Marwedel "Matching System and Component Behavoriour in MIMOLA Synthesis Tools", EDAC, Glasgow 12-15 march 1990.

[5] P. Paulin, J. P. Knight "Force-directed scheduling in automatic data path synthesis", 24th ACM/IEEE DAC, 1987.

[6] L. Stock, R. Van Den Born "EASY : multiprocessor architecture optimization", Workshop logic and architecture synthesis for silicon compilers, May 1988, Grenoble, France.

[7] P. Gutberlet & J. Müller & H. Krämer & W. Rosenstiel "Automatic Module allocation in high level synthesis" , EURO-DAC 92, Hamburg 7-10 September 1992.

[8] P. Denyer & D. Renshaw "VLSI Processing: A Bit Serial Approach" Edition Addison wesley, 1982.

[9] A. Parker "MAHA: aA program for Datapath Synthesis" 23th ACM/IEEE Design automation conference, Las Vegas Jully 1986.

[10] D. Gajski & all "Silicon compilation" Edition Addison-Wesley, 1988.

[11] M. Israel, J. Benzakki, M. Francois "OSYS: Tools for behavioral synthesis of ICs with VHDL" VIUF Spring 92, Scottsdale USA.

[12] A. Jerraya "VHDL and Architectural Synthesis" Spring 93 Innsbruck, Austria.

[13] E. Martin, O. Sentieys & J.L. Philippe. "Traitement du signal et architecture dédiée: GAUT une approche méthodologique en CAO de VLSI". Congrès AFCET, Paris 8-10 june 1993.

[14] R. Bergamaschi, R. Camosano & M. Payer " Area and Performance Optimizations in Path-Based Scheduling". EDAC, Amsterdam 25-28 Fev 1991.

[15] S.Y. Kung & H.J. Whitehouse & T. Kailath "VLSI and Modern signal processing", Prentice Hall.

[16] J.L Philippe, E. Martin "Prototyping Digital Signal Processing using VHDL and CAD Architectural Tool" Spring 93, Innsbruck, Austria.

[17] O. Sentieys "Analyse et synthèse d'architectures en TDSI: vers la conception d'architectures hétérogènes" Thesis Universisty of Rennes I 1993.