

# TARGET ARCHITECTURES IN THE CATHEDRAL SYNTHESIS SYSTEMS: OBJECTIVES AND IMPACT

Francky Catthoor (1), Jan Rabaeys (2), Hugo De Man (1)

(1) IMEC Lab., Leuven, Belgium, Phone: +32-16-281201

(2) Currently at Cory Hall, U.C.Berkeley, CA94720, U.S.

This research has been sponsored in part by the ESPRIT97 project of the EC and the industrial partners Philips, Siemens, BTMC and Silvar/Lisco.

## ABSTRACT

The rapidly increasing complexity of DSP applications and the need for short design times in the consumer and telecom fields necessitate the use of CAD tools supporting high-level architectural synthesis in these domains. In addition, sufficient efficiency is crucial for competitive real-life systems. It is our belief that these objectives can only be accomplished in an acceptable way by abandoning the ideal of total generality, i.e. by adopting several architectural target styles each tuned to specific (but still relatively broad) application domains. At IMEC, currently five such styles have been defined. Their impact on the algorithmic and rule-based synthesis tools which are under development is the main topic of this paper. Both the overlaps and the differences between the five approaches are addressed. Finally, also the need for powerful, diversified mechanisms of user interaction are advocated, at all the stages in the design.

## 1 INTRODUCTION

State-of-the-art Digital Signal Processing applications typically involve a rapidly increasing arithmetic complexity which has to be combined in many cases with a need for flexible and powerful decision-making operations. Moreover, they are not only addressing word-oriented applications but also employ advanced vector and matrix operations. This large application domain ranges from audio, speech and user-end telecommunications exhibiting medium throughputs, to image, video and radar processing which require higher sample rates. For most of these complex systems, especially in the consumer or home markets, a sufficiently large design efficiency has to be achieved in terms of throughput, (IC and board) area, power consumption, and packaging. The results should approach the ones of manual full custom layouts. However, at the same time, the design cycle from algorithm to layout for these ASIC's should be reduced from a few years to a few weeks in order to follow the rapidly evolving market. These objectives can only be achieved by supporting the complete design path from algorithm to ASIC with effective high-level synthesis and powerful module generation tools which result in efficient architectures and layouts. At IMEC we believe the latter is only possible when the synthesis strategies are based on a well-controlled user interaction and especially on clearly defined and well-chosen target architectures ("templates"), as demonstrated in our CATHEDRAL project [DeM86, DeM87].

## 2 ARCHITECTURAL STYLES

Currently, we have restricted ourselves to application-specific approaches (because of the efficiency needed) and the control flow option (as opposed to pure data flow concepts) because of the fixed sample rates in typical DSP applications [Cat87]. Within this still broad architectural domain, five different styles have been defined in our research projects (Fig.1) and more are in the making. Each of them is tuned towards a specific class of (sub)algorithms with well-defined properties, all within the signal processing field. In addition, they are all supported by a separate CATHEDRAL synthesis approach and CAD environment (as indicated).

The current styles include:

1. microcoded multi-processors with application-specific execution units and a powerful multi-branch controller [Cat88a] for algo-

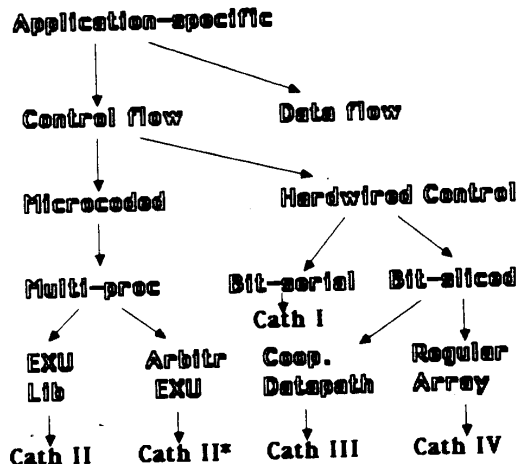


Fig.1 Overview of architectural approaches and corresponding CATHEDRAL synthesis systems.

1. microcoded multi-processors with application-specific execution units and a powerful multi-branch controller [Cat88a] for algorithms at low- to medium throughputs involving complex decision-making and block or matrix manipulation as in audio, speech, telecom, algebraic processing and even back-end image and video processing (CATHEDRAL-2 [Rab88] - CATHEDRAL-2nd [Lan89]).
2. hard-wired bit-serial architecture for linear DSP [VGi83], i.e. digital filter algorithms and the like in audio, speech and so on (CATHEDRAL-1 [Jai86]).
3. bit-sliced multiplexed data-paths cooperating under a hard-wired hierarchically decomposed controller [Cat88b], both optimized in terms of critical timing paths, for recursive high-speed algorithms as in medium-level image, video and radar processing applications (CATHEDRAL-3 [Not89]).
4. regular array architectures with a regular network of processing elements, mostly localized communication and distributed storage [Cat88b], suited especially for the front-end modules in image, video and radar processing (CATHEDRAL-4 [Ros89]).
5. the communication in between the architectural styles mentioned above [Cat87, DeC89] and also the I/O communication is an important issue and has to be addressed separately. Synthesis of I/O "algorithms" from behavioural description is needed too (I/O-CATHEDRAL [VBe89]).

## 3 DESIGN METHODOLOGY: OVERLAPS AND DIFFERENCES

In the design path from a behavioral signal flow-graph (SFG) description to an efficient ASIC architecture, several tasks can be identified which are relatively independent of the specific target architecture. In the current development of the CATHEDRAL environments mentioned in section 2 it can indeed be seen

that most (sub)tasks occur for more than one style. Some of them can be mostly or partly shared, such as partitioning into large processing modules. (some) data flow transformations, multi-rate and multiple-precision decisions, type alignment, and bus-, multiplexer- or register optimization. Still, most activities labeled in an identical way have to satisfy different boundary conditions and have to support distinct SFG properties dependent on the class of applications to be mapped (section 2). As a result they are tuned to specific contexts and they employ either different algorithmic techniques or fully "matched" expert systems. Examples include operator type selection, hardware allocation, operator assignment, time scheduling, control flow optimisation, memory management (both foreground and background), and inter-module communication. However, the expertise to construct a tool for a task in one case can frequently bootstrap building a second and a third version. The main overlaps will be identified in the sequel of this section. At the same time, the distinction between the actual implementation of some of the key tasks in the different CATHEDRAL's will be discussed in more detail.

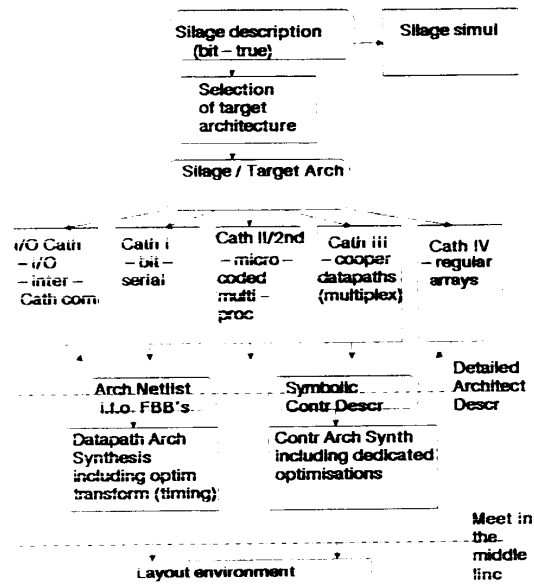


Fig.2 Overall design methodology for the CATHEDRAL's.

A global view of the CATHEDRAL tool-boxes is presented in Fig.2. They all start from a common behavioural description in the applicative SILAGE language [Hil85]. This specification is split up in parts suited for the different synthesis systems. The final combined results from the architectural synthesis phase are the structure (architectural net-list or ANL) and the ordered execution of operations on this ANL (which can be seen as a more detailed SFG), i.e. the controller description. In Fig.3, as an example, the "anatomy" of the CATHEDRAL-2nd [Lan89] and CATHEDRAL-3 [Not89] synthesis environments is illustrated. An attempt has been made to stress the resemblance as much as possible, although this does not mean they are completely similar. These two will be employed as the test cases to demonstrate our claims. It should be noted though that similar arguments apply for the other architectural tool-boxes.

Now, the different tasks will be discussed one by one:

1. First, all CATHEDRAL's need some type of partitioning in large processing modules. The core of appropriate techniques can

### Cathedral 2nd Cathedral III

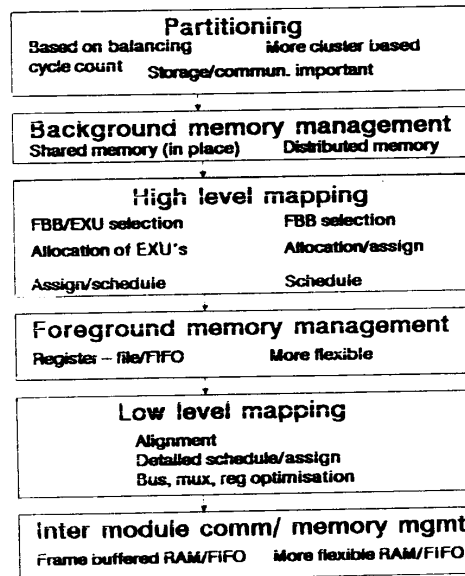


Fig.3 Overview of tasks addressed in the CATHEDRAL-2nd and CATHEDRAL-3 tool-boxes tuned, towards microcoded multi-processor respectively cooperating data-path architectures.

probably be shared, although the detailed objectives of the division process and also the actual goal functions to be optimized differ. This is mainly a consequence of the way in which inter-module communication is achieved [Cat87]. In the CATHEDRAL-2/2nd style for instance, frame-buffering is typically introduced between the microcoded "processors" [DeC89]. This allows for rigorously independent synthesis of each processor but results in larger buffers and a decreased overall area efficiency. In contrast, much more flexibility is desired for inter-module communication within CATHEDRAL-3, but then the final solution (luckily) also resembles the original signal flow much more due to the high throughputs required (see e.g. [Cat87]).

This partitioning task has not received much attention up to now in the literature (except in [Tho88]) but it is very important to arrive at a fully supported design path. It is a topic of future research.

2. Next, in most cases some type of high-level memory management has to be foreseen. Typically, high-level decisions concerning background memories are taken. For CATHEDRAL-2nd this is e.g. oriented towards matrix manipulation [Vba89], or towards handling variables with many indices in general. But a similar step has to occur for the other target architectures, except for the bit-serial case where the application domain is limited to applications without loops (and indices). It is however very questionable whether this step can be unified for all of the CATHEDRAL's. Indeed, many of the "rules" will probably have to be application-domain specific for efficiency. For instance, in the microcoded multi-processor case (CATHEDRAL-2/2nd) the hardware is shared intensively so the detection of variables (arrays and matrices) which can be stored in-place is crucial to arrive at a reasonable chip area. For the cooperating data-paths (CATHEDRAL-3) which are aimed at (much) lower ratios between achievable clock-rate and throughput, sharing memory locations is not that difficult any more. Moreover, the emphasis has

then shifted from memory sharing (for area reasons) to arriving at the desired throughput with the defined storage organisation. Hence, multi-port memory structures such as pointer-addressed memories [DeC89] play an important role. Consequently, only the general frame-work and methodology plus some of the mapping rules can be fully shared between these CATHEDRAL's. In the literature this task has hardly been mentioned up to now.

3. Then, a first (high-level) architecture mapping step is needed. Here the restriction has been made that mostly rule-based activities occur. The reason is that at this highest level, the decisions which have to be taken are still so general that an exhaustive exploration of the design space would be infeasible, as many degrees of freedom are present. We believe this kind of flexibility cannot be handled very well by algorithmic techniques alone, as the problems are NP-complete or even NP-hard. Therefore, pruning has to occur making use of what is known about the specific application and the target architecture. Moreover, in order to restrict the flexibility a little, usually only relatively "high-level" operations are employed (see e.g. [Lan89]), i.e. the expansion to the primitive FBB operations is postponed (step 4).

Typical examples of (mostly) rule-based high-level tasks include many high-level transformations on the signal flow graph, handling multi-rate behaviour, decisions on the use of multiple precision operations, allocation of a number of (clustered) hardware operators (i.e. deciding on the composition and the amount of each type), operation to operator type binding (type selection), control flow optimizations/transformations and so on.

Most of the "matured" synthesis systems (see [McF88] for an overview) currently deal with a few of these tasks, but none deals with the complete list.

In some cases algorithmic techniques are needed as "subroutines" though, called from within a rule-based system to provide more information concerning subproblems. A typical example are cycle count estimates obtained by "crude" scheduling. Sometimes a (costly) iteration is needed involving the complete "bottom" part (steps 4-6). The control remains however within the expert system.

It should be pointed out that many of the major differences between the five CATHEDRAL's in section 2 are located at this level. This observation is also partly related with the discussion why rule-based approaches are needed: the search spaces are so large that a "unified" approach would not lead to efficient implementations. Hence, at IMEC we have rigorously chosen for diversification in terms of both architectures and the corresponding high-level mapping.

In CATHEDRAL-2nd for instance [Lan89], the concept of "execution units" [Cat88a] with many possible operations but with a limited topological flexibility plays an important role. This choice has been made to arrive at efficient microcoded architectures for complex algorithms with much decision-making, operating at relatively low throughputs. It allows to restrict the valid search space for allocation considerably. However, the assignment of the operations to the heavily shared operators is still very hard and has to be combined with scheduling information to arrive at a reasonable overall solution.

In CATHEDRAL-3 on the other hand, the required rates are much higher and therefore, the topology of the data-paths should be virtually unrestricted (except for storage which is addressed separately) [Cat88b,Not89]. At the same time however, some form of hardware-sharing is still necessary for efficiency. In the microcoded case above, operations which are locally clustered in the SFG are typically "collapsed" onto one operation or a small group of operations (i.e. an execution unit). The disadvantage of this type of collapsing is that it leads to relatively large area overheads in terms of foreground memory and control. Still, it is the only way to arrive at the heavy sharing required for those applications. For the cooperating data-paths, the situation is quite different in

that the sharing ratio is typically relatively limited (about 10 or so). The throughput is of major importance but fixed. Hence, the data-path and controller area (and power) costs have to be restricted by all means to arrive at practical IC's. Hence, in most cases it has proven to be more effective to try to identify more or less repetitive node structures in the SFG which can then be "folded" and mapped directly onto the same (large) cluster of building blocks [Not89]. This type of hardware sharing can lead to smaller area overheads in practice [Cat87, Not89], if applied carefully. As a result, the allocation and assignment tasks are now interacting heavily, and the corresponding synthesis tools must necessarily differ from the ones in CATHEDRAL-2nd. Moreover, the requirements on the scheduler are very limited for the cooperating data-paths, as the graphs to be "ordered" in time contain only a few dozens of nodes compare to orders of magnitude more for the microcoded applications. This allows to select scheduling techniques which are more likely to arrive at fully optimal results. Several other differences and also overlaps exist but will not be treated here.

4. As a preparation for the next step, typically an expansion of the SFG information has to take place after the high-level mapping [Lan89]. The major effort in this step can be shared by all CATHEDRAL's.

5. Next, the organization of the "foreground" memory has to be considered. Typically, this involves coming up with some kind of clustering of the distinct registers (each allocated for a single or a few variables) into more efficient structures like register-files or pointer-addressed memories if needed [Lan89]. At this stage, the decisions on the back-ground memory above have to be refined too.

In most cases we will probably have to backtrack (or come back) to this decision because the low-level mapping changes some of the boundary conditions for this memory management. Therefore, a user-controlled iteration is taking place. At present, it looks like most of these efforts can be shared for all the CATHEDRAL's except for the bit-serial case.

A number of techniques have been proposed already at this level (see [Lan89] for a discussion).

6. Now the "low-level" architecture mapping occurs. At this level the flexibility in the design space is more restricted because in the CATHEDRAL's we are approaching the "detailed architecture" line (Fig.2). Indeed, due to the restrictions in the library features of the different target architectures, tasks like connection (e.g. bus and mux) optimization, type alignment [Pau89], the final detailed assignment and scheduling and the optimisation of registers based on life-times [Goo87] become somewhat more tractable. This is very fortunate because typically a very large amount of SFG nodes is present after the expansion step. Thus, these tasks would be virtually impossible to solve even near-optimally with algorithmic tools if no restriction would be imposed on the underlying FBB library. As a desirable "side-effect" of the common library for all CATHEDRAL's and due to the nature of these subtasks, many of them can be shared among the different CATHEDRAL's, especially the hardware optimizations.

Most of the effort on high-level synthesis (see [McF88] for a discussion) is concentrated at this stage so far. Here, the most advanced results have already been obtained.

7. The last step involves combining the divided "modules" again into a complete architecture and the incorporation of the inter-module communication network. This step is for the moment only solved for CATHEDRAL-2nd [DeC89], but we believe similar approaches apply for the other environments. In view of the comments in item 1, they will have to be steered by architecture dependent expertise (i.e. rules). Especially the necessary flexibility in CATHEDRAL-3 requires powerful communication methods [Cat87].

So far, this step is not supported in other synthesis systems. It should be stressed that arriving at a flexible reuse of subtasks between the CATHEDRAL's hinges upon the availability of a number of standardized exchange formats between the tools (see also [Lan89]). Moreover, a complete characterisation of the libraries with the Functional Building Blocks has to be available. In order to restrict the maintenance and support effort to a minimum, everything is currently being done to share them for different CATHEDRAL's.

#### 4 USER INTERACTION

Effective mechanisms for user-interaction constitute an important issue. Due to the very high cost of system verification by simulation, even at a high behavioral level, the algorithm verification step should be repeated as few times as possible. Therefore, the hints to the high-level synthesis systems as proposed by the designer during the architectural exploration stage, should be entered and interpreted in such a way that the correct-by-construction principle remains valid. This cannot always be accomplished in full, but still, much design time can be saved in this way. For this purpose, several mechanisms have been defined (see also [DeM89]). At the highest level we allow "pragma" constructs, i.e. structural hints, in our behavioral specification language SILAGE [Hil85]. In some of the steps above the expansion (items 1-3) some more powerful editing mechanisms are desirable, such as for guiding the background memory management [Vba89]. Finally, below the expansion more indirect ways of interacting by means of "tolerances" on cycle count and/or area appear to be more attractive.

#### 5 DEMONSTRATORS

Developing a suitable design methodology should be approached from the point of view of system and architectural designers. The experience with several realistic demonstrator applications has been crucial to come up with efficient and useful methodologies. They have been selected from many different application domains in order to clarify the distinction between the CATHEDRAL's in terms of both architectural and synthesis issues. These demonstrators will not be explained here due to lack of space. Moreover, many of them have already been published elsewhere (see e.g. Rab88, Cat88a, Cat88b). They substantiate our efficiency claims presented above, in the sense that these (mostly) manual exercises have resulted in architecture realizations which would be acceptable in a practical i.e. industrial context.

#### 6 DISCUSSION

Between the different CATHEDRAL systems developed at our laboratory, a relatively large overlap exists, although some of the crucial tools differ substantially in their approach. Many of the design tasks to be supported overlap heavily such as partitioning, signal type optimization, decisions on multi-rate organization and multiple precision, and also the final optimization of hardware units (registers, busses, multiplexers etc.). Others pop up at different locations but differ in that they exploit the features of the architectures for which they are tuned. Examples include memory management (both foreground and background), hardware allocation, operator assignment, and time ordering (leveling or scheduling). This has led to a huge diversity of software programs which are being developed at our laboratory. Founding the synthesis on a firm architectural basis is one of the main reasons for the acceptable efficiency experienced in realistic (complex) test cases.

Many other environments are currently under development (see [McF88] for a recent overview). Some share our views, others follow completely different paths. It is difficult to predict which approach is the most effective. This typically depends heavily on the demonstrator application used. Still, the fact that a complete algorithm of practical size should be handled is essential. Within this constraint though, there is room for a large variety of approaches for the vast range of different application domains.

A link with a common high-level description language (applicative SILAGE [Hil85] with extensions) for all the CATHEDRAL's allows to compare realizations of the same application on different target architectures (if required). The architectural exploration within a particular target class is supported by means of powerful interaction mechanisms at the higher levels which allow in principle to tune the entire architecture. The resulting architectures for all these CATHEDRAL's are constructed as a netlist of compact, highly parameterisable building blocks, available in a precharacterized macrocell library (meet-in-the-middle design strategy) [DeM86]. These building blocks are assembled into data-paths in a module generation environment. In addition, powerful control architecture synthesis approaches are being integrated. The individual descriptions are then sent to a floor-planner. A complete environment of this type has been developed for the CATHEDRAL-2 style in the PIRAMID system [Hui88].

#### Acknowledgements.

This research has been conducted mostly in the context of an EC project with many partners and groups involved. Major contributions to the material described here have been made by Jef Van Meerbergen, Gert Goossens, Jan Vanhoof, Dirk Lanneer, Rajeev Jain, Stefan Note, Johan Van Ginderdeuren, Mark Pauwels, Ingrid Verbauwhede, Peter Van Bekbergen, Jan Decaluwe and Lee Chen Yi.

#### REFERENCES

- [Cat87] F.Catthoor, "Architectural design strategies for complex DSP-systems in an automated synthesis environment", Doctoral dissertation, ESAT, K.U.Leuven, Belgium, May 1987.
- [Cat88a] F.Catthoor, J.Rabaey, G.Goossens, J.Van Meerbergen, R.Jain, H.De Man, J.Vandewalle, "Architectural Strategies for an Application-specific Synchronous Multi-processor Environment", IEEE Trans. on Acoustics, Speech and Signal Processing, Vol.36, No.2, pp.265-284, Feb. 1988.
- [Cat88b] F.Catthoor, H.De Man, "Customized architectural methodologies for high-speed image and video processing", Proc. Int. Conf. on Acoustics, Speech and Signal Processing, New York, pp. 1985-1988, April 1988.
- [DeC89] J.De Caluwe, J.Rabaey, J.van Meerbergen, H.De Man, "Interprocessor communication in synchronous multiprocessor digital signal processing chips", accepted for publication in IEEE Trans. on Acoustics, Speech and Signal Processing, 1989.
- [DeM86] H.De Man, J.Rabaey, P.Six, L.Claesen, "Cathedral II: a silicon compiler for digital signal processing", IEEE Design and Test Magazine, pp.13-25, Dec. 1986.
- [DeM87] H.De Man et al., "Synthesis of DSP systems at Leuven", session in Proc. IEEE Int. Conf. on Computer Design, Port Chester NY, pp.134-145, 1987.
- [DeM89] H.De Man, F.Catthoor, G.Goossens, J.Van Meerbergen, J.Rabaey, J.Huisken, "Architecture-driven synthesis techniques for mapping digital signal processing algorithms into silicon", to be published in special issue on comp.-aided design of Proc. of the IEEE, 1989.
- [Go87] G.Goossens, J.Rabaey, J.Vandewalle, H.De Man, "An efficient microcode-compiler for custom DSP-processors", Proc. IEEE Int. Conf. Comp. Aided Design, Santa Clara CA, pp.24-27, Nov. 1987.
- [Hil85] P.M.Hilfinger, "A high-level language and silicon compiler for digital signal processing", Proc. IEEE Custom Integrated Circuits Conf., Portland OR, pp.213-216, May 1985.
- [Hui88] J.Huisken, H.Janssens, P.Lippens, O.Mc Ardlie, R.Segers, P.Zegers, A.Delaruelle, J.v.Meerbergen, "Design of DSP systems using the PIRAMID library and design tools", Proc. Internl. Workshop on Logic and Architecture Synthesis, Grenoble, France, May 1988.
- [Jai86] R.Jain, F.Catthoor, J.Vanhoof, B.De Loore, L.Claesen, J.Van Ginderdeuren, H.De Man, J.Vandewalle, "Custom Integration of a PCM-PDM Transmultiplexer using a Computer-Aided Design Methodology", IEEE Trans. on Circuits and Systems, Vol.CAS-33, pp.183-195, Feb. 1986.
- [Lan89] D.Lanneer, M.Pauwels, F.Catthoor, J.Van Meerbergen, G.Goossens, H.De Man, "System-oriented high-level synthesis of microcoded architectures for DSP", submitted for publication.
- [McF88] M.C.McFarland, A.C.Parker, R.Camposano, "Tutorial on high-level synthesis", Proc. 25th ACM/IEEE Design Automation Conf., San Francisco CA, pp.330-336, June 1988.
- [Note89] S.Note, F.Catthoor, J.Van Meerbergen, H.De Man, "High-throughput multiplexed data-path synthesis", submitted for publication.
- [Pau89] M.Pauwels, F.Catthoor, D.Lanneer, H.De Man, "Type-handling in bit-true silicon compilation for DSP", submitted for publication.
- [Rab88] J.Rabaey, H.De Man, J.Vanhoof, G.Goossens, F.Catthoor, "CATHEDRAL II: A Synthesis System for Multi-processor DSP Systems", in "Silicon Compilation", D.Gajski (ed.), pp.311-360, 1988.
- [Ros89] J.Rosseeel, F.Catthoor, H.De Man, "Assignment, sequencing and projection for regular array synthesis", submitted for publication.
- [Tho88] D.E.Thomas, E.Dirkas, R.Walker, J.Rajan, J.Westor, R.Blackburn, "The system architect's workbench", Proc. 25th ACM/IEEE Design Automation Conf., San Francisco CA, pp.337-343, June 1988.
- [Vba89] I.Verbauwhede, F.Catthoor, J.Vandewalle, H.De Man, "Background memory management for the synthesis of algebraic algorithms on multi-processor DSP chips", submitted for publication.
- [Vba89] P.Van Bekbergen, J.Van Meerbergen, F.Catthoor, H.De Man, "Race-free time-optimized synthesis of asynchronous interface circuits", submitted for publication.
- [VGI83] J.Van Ginderdeuren, H.De Man, M.Goncalves, W.Van Nuijse, "Compact CMOS building blocks and a methodology for dedicated digital filter applications", IEEE J. Solid-state Circ., Vol.SC-18, pp.306-316, June 1983.