1173

UNIVERSITE DE NEUCHÂTEL
INSTITUT DE MICROTECHNIQUE

# On the Design and Implementation of DSP Algorithms: An Approach using Wave Digital State-Space Filters and Distributed Arithmetic

THESE

PRESENTEE A LA FACULTE DES SCIENCES
POUR OBTENIR LE GRADE DE DOCTEUR ES SCIENCES

PAR

Ulf Sjöström

# IMPRIMATUR POUR LA THÈSE

On the Design and VLSI Implementation of DSP Algorithms: An Approach using Wave Digital State-Space Filters and Distributed Arithmetic.

de Monsieur Ulf Sjöström

---

## UNIVERSITÉ DE NEUCHÂTEL

FACULTÉ DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel sur le rapport des membres du jury,

Messieurs F. Pellandini, A. Shah, M. Kunt (EPF-Lausanne), D. Mlynek (EPF-Lausanne) et L. Wanhammar (Linköping)

autorise l'impression de la présente thèse.

Neuchâtel, le 9 février 1993

Le doyen :

A. Robert

# TABLE OF CONTENTS

## chapter 3. Digital Signal Processing and Wave Digital Filters



## chapter 4. State-Space Representation

# On the Design and Implementation of DSP Algorithms:

## An Approach using Wave Digital State-Space Filters and Distributed Arithmetic


## SUMMARY

*A new methodology for design and implementation of digital filters is presented in this thesis. The presentation stretches over the domains of digital signal processing, VLSI architecture and CMOS design. Wave digital filters are introduced, specifically the method to obtain wave digital filters in state-space form. Furthermore, an hardware architecture based on distributed arithmetic is proposed. To support the methodology a vertically sliced synthesis system has been developed. The used programs and algorithms in this system are discussed. It turns out that the presented approach leads to very regular and efficient designs and implementations for a large class of digital signal processing algorithms.*

# *Conception et Réalisation des Algorithmes de Traitement Numérique du Signal:*

## *Une méthode utilisant des filtres numériques d'onde représentés dans l'espace d'état et réalisés en arithmétique distribuée*


## *RÉSUMÉ*

*Nous présentons dans cette thèse une nouvelle méthodologie de conception et de réalisation de filtres numériques, dont le domaine touche au traitement de signal, aux architectures VLSI et à la conception de circuits CMOS. Nous y traitons en particulier de filtres numériques d'onde et de la méthode permettant de les représenter dans l'espace d'état. Nous y proposons en outre une architecture matérielle en arithmétique distribuée; à cet effet, nous avons développé une approche de synthèse de circuit par tranches verticales. Nous discutons dans cette thèse le détail des programmes et des algorithmes mis en œuvre. Il apparaît que l'approche proposée conduit à une réalisation très efficace et régulière, et ce pour un grand nombre d'algorithmes de traitement de signal.*

# chapter 1. INTRODUCTION

*This chapter gives a survey and a general introduction to DSP, VLSI and ASICs. The background in these fields and some of the more important developments made over the last few years are discussed. The needs for adequate design methodologies and CAD tools are explained and different existing methodologies and synthesis systems are briefly mentioned. The main outline of the contents of this thesis is also presented.*

## 1.1. Motivation

### 1.1.1. Background

The motivation behind this work has been found in the ever evolving domains of *Digital Signal Processing* (DSP) and *Very Large Scale Integration* (VLSI). Due to this development it is possible to solve problems today using digital techniques in fields where it was impossible only some years ago. Hence, the range of applications is as well continuously growing. It has been observed that the design and implementation tasks for DSP algorithms become more and more complex with all the new possibilities offered and due to the increased complexity of applications. Especially, the tasks that are related to both the DSP and the VLSI domain are difficult to solve.

Clearly, it is indispensable to use good strategies to quickly arrive to efficient, cheap and correct solutions. This thesis will treat one very specific approach to attack these problems. It will be presented as a methodology supported by a *vertically sliced synthesis system.*

This thesis will emphasis the feasibility of Wave Digital State-Space Filters (WDSSF) and of Distributed Arithmetic (DA) as the title proposes. Some new aspects on the design of these filters will be discussed. Tools and algorithms to support their design will also be introduced. A new architecture based on the DA processor is proposed. All the necessary components have been carefully designed and implemented in a CMOS technology. A simple way of mapping DSP algorithms onto this architecture will also be proposed.

The presented approach is the outcome of a very stimulating time at the university of Neuchâtel. This work has led to a number of publications [Defi89, Sjös89a, Sjös89b, Bals90, Defi90, Sjös90, Anso90, Anso91a, Anso91b] and some postgrade courses [ Pell89, Pell90] to which the author has participated. Clearly, the presented results are also influenced from the time the author made at the University of Linköping [Sjös86] and related to the work of Wanhammar [Wanh81] and Sikström [Siks86].

Evidently, there exist parts that have not been covered. It should be noted that the whole work has been performed in a very small group of three people being involved in other projects as well. Therefore, it is not any complete product that is presented, but a collection of ideas, algorithms and tools to speed up and to simplify the design and implementation task.

### 1.1.2. Outline of the Text

In the rest of this first chapter some important aspects of DSP and VLSI will be viewed. Some related synthesis systems will also be mentioned.

The concepts and the motivation of the used approach are introduced in Chapter 2. The aspects of the DSP domain are discussed in general in Chapter 3. Wave digital filters and related structures are treated in detail. Some new ideas concerning the synthesis of WDSSF are shown in Chapter 4 and 7. The concept of state-space representation, which has been chosen as a basic description of DSP algorithms, will also be discussed in detail in Chapter 4. The architecture synthesis, including the definition of the used architecture, is treated in Chapter 5. Moreover, the used mapping and scheduling are discussed in Chapter 7.

The final part concerns the VLSI implementation, especially the development of the necessary cell-libraries and the required modules are discussed in Chapter 6 where a detailed description of each block is presented. The part concerning the automatic generation of the blocks is given in Chapter 7, together with the description of the used tools.

The conclusion in Chapter 8 gives a summary of the main advantages and features derived using the proposed approach. Some propositions for future extensions and research in this field are made as well.

## 1.2. DSP and VLSI Domains

### 1.2.1. The DSP Field

Comparing the DSP domain to the analog counterpart one finds that problems such as tuning, ageing and temperature drift can be totally circumvented. A much wider range of structures and algorithms can be found in the this domain. It is possible to take advantage of results from many scientific fields including mathematics, signal theory, computer science, digital electronics, etc. Programmable, or adaptive systems, can easily be designed using digital techniques while this is not always possible in the analog domain.

Large and complex DSP systems can be integrated on a few integrated circuits. If these chips are designed exclusively for a specific application they are called *Application Specific Integrated Circuits* (ASICs). The presented approach is clearly directed towards ASIC implementations.

Telecommunication is one of the largest and most important application field today. It includes television systems, consumer electronics, satellite communication systems, data transmission, telephony, telegraphy, etc. Other important application fields include for instance, measurement, radar, sonar, speech and image processing. In general, processing-speed requirements range from some fractions of Hz up to GHz.

### 1.2.2. Classes of DSP Algorithms

Several different algorithms can be found in the DSP domain. They can be classified in many different ways, but two fundamentally different groups can be identified:

- ***Fixed algorithms:***
  This class of algorithms is assumed to be *constant and periodically repetitive*. This includes *Linear and Time Invariant* (LTI) DSP algorithms and all other algorithms that do not vary with the time and are not data dependent. LTI digital filters, both recursive and non-recursive, are typical examples of this group. These algorithms are interesting from an implementation point of view. Neither decisions nor conditional branching is needed, only cyclically repeated *data*

*independent operations* are required. Normally, *data-flow architectures* and *fixed scheduling* can be used for the implementation.

- ***Adaptive algorithms***:
  This class of DSP algorithms has in general either a data-driven or a time dependent behavior. Implementation architectures should usually allow both decisions and conditional branching. It seems as if the best suited architectures for these applications are still based on variations of the classical von Neuman architecture. Typically these algorithms are implemented using *of-the-shelf digital signal processors* or ASICs based on the classical processor architecture.

Naturally, a much finer classification within these groups can be made. For the first group this will be discussed in Chapter 3, while the second is outside the scope of the text and will not be further considered.

### 1.2.3. The VLSI Field and ASIC Implementation

Another subject treated in this text is the VLSI domain, and therefore a short introduction to this follows. The impressing growing-rate in this field is the outcome of the improvements in process technology and in design methodologies.

An ASIC today can contain approximately 20-30 times more devices and can be clocked with speeds 10 times faster than they could 10 years ago. A further improvement is possibly also gained using more efficient and better adapted architectures. This development is expected to continue in the future.

The reason to use hardware implementation is usually that the commercial processors do not furnish high enough performance. Custom hardware implementation could be done using discrete components on a printed circuit board, but usually also low power dissipation and a more compact format are desired [Swar86]. This leads to VLSI implementations and ASICs.

A design task for ASIC implementation includes many steps. First of all an appropriate architecture should be selected, then the elements of this architecture should be designed, i.e., processors, memories, etc., and finally the DSP algorithm should be mapped and scheduled on the hardware.

If only one or very few processors are used for the implementation the scheduling can become very similar to the program used in the software

implementation. On the other hand, a very simple fixed scheduling can be used if there is a one-to-one correspondence between the DSP operations and the architecture components.

The implementation on silicon of a specific architecture can be done using a wide range of methods. The ASIC design methodologies can be divided into two main categories [Fey 87], the *Semi-Custom* designs and the *Custom* designs.

The *semi-custom* categories have all in common that a part of the design is made beforehand. In some cases also part of the chip is prefabricated. Only some few mask layers have to be designed and processed for the chip afterwards. The main drawback is a limited efficiency and possibly limitations in complexity and performance.

Using *custom* designs, on the other hand, extremely efficient high performance layouts can be obtained. The approach can be compared to developing software using an assembler language, which often is the case for critical DSP applications. However, this strategy implies that the designer has a deep knowledge in integrated circuit design and the used technology [Mead80, West85].

## 1.3. Methodologies and Synthesis Systems

A design *methodology* can be defined and viewed many ways. Here it is interpreted as the overall strategy suggesting how to solve subtasks on each level. The architecture and the realization should also be defined by the methodology.

Considering the ASIC implementation of DSP algorithms, a complete methodology should start with the choice and synthesis of the DSP algorithms. It should also describe how to translate the DSP algorithm into processing requirements for a given architecture including scheduling. Finally, the implementation strategy should be defined.

### 1.3.2. CAD Frameworks

Two specific classes of CAD frameworks can be distinguished. The first group of special purpose CAD tool is the so-called *synthesis systems*. These are specialized systems for a specific field of application including tools for all phases in the design such as synthesis tools, verification tools, optimization tools, etc. If the CAD tool is specially targeted for a system it is sometimes referred to as a *vertically sliced* synthesis system. The outline of

one such system will be described throughout this thesis. Secondly, the so-called *expert systems* where artificial intelligence techniques have been used to introduce knowledge and learning in the system. These expert systems will probably evolve enormously in the future while they are fairly simple today and they are not used to a large extent yet.

### 1.3.3. Existing Synthesis Systems

Research in these fields has been going on in many places the last few years and different solutions have been proposed. Synthesis systems have also been defined. Some of these are oriented specially towards DSP design and implementation while others are more general so-called *silicon compilers*. Systems belonging to the last group are often bulky and too inefficient to give satisfactory results for DSP ASIC integration. Another important fact making some of them less suitable for DSP is that they might reschedule and rearrange the DSP algorithms such that the numerical properties are altered. For these reasons they are not further considered here.

The specialized DSP synthesis systems, on the other hand, are of great interest here. A large number of systems have been proposed over the last decade. Some of them include all phases, i.e., parts for synthesize, realization and implementation, while others only emphasize on the realization phase. Some of these systems are listed here below. This list might not be extensive or complete but it includes some tools that clearly have influenced the work presented here.

- **Cathedral I:**
  This is the first of a series of synthesis systems developed at Catholic University of Leuven and at IMEC, Leuven, Belgium [Jain86]. It is also the first of the Cathedral systems. The objectives behind this system were to treat fixed DSP algorithms and implement them on an optimized bit-serial architecture. Multiplications are implemented by a shift-add and shift-subtract approach using *canonic signed digit* (CSD) coded coefficients. Very few basic building blocks are required this way, and the methodology is impressively efficient from an area point of view [VanG86, Jain86, Clae88]. All the main design steps are treated in this system. However, it only covers applications with a low to moderate speed, and only fixed algorithms. Tools from Cathedral I are used in the presented approach.

- **Cathedral II:**
  This system exists in two versions [DeMa87] where the second one is called *Cathedral Second*. Here an architecture based on the classical processor concept is used [DeMa86, DeMa88]. Multiple data-paths can be generated and different solutions can be directly compared. The whole system is built up around the SILAGE language used both for synthesis and realization [Sche88]. The final DSP algorithm described in this language is mapped using strict mapping rules onto the architecture. A large amount of CAD tools has been developed around Cathedral II and currently work is going on to commercialize the system under the name of *DSP station* [DSP 91].

- **Cathedral III:**
  Cathedral III was designed to meet requirements of very high speed applications, e.g., image processing. The approach is, in principle, limited to fixed DSP algorithms. The architecture used is based on *systolic arrays* and *wavefront arrays* [Note88]. Furthermore, pipelining and maximal resource allocation are used to achieve maximal throughput.

- **First:**
  This is an approach using a bit-serial architecture to implement different DSP algorithm developed at the University of Edinburgh, UK, [Deny85]. The methodology is not complete in the sense that it does not include the DSP synthesis step. It emphasizes on the mapping and the implementation. Different special primitives to form signal processing elements are used. In *Second* [Smit88] this concept is extended with a macro-cell compiler. The main feature of this system is simplicity.

- **Lager:**
  This system has been developed at UC Berkeley, USA, [Raba85], and it has been continuously improved over the years (it now exists in versions I, II, III and IV). The basic idea is to generate a specially tailored micro-processor type of architecture with special data-paths and specialized micro-instructions. Communications are bit-serial while data-paths are bit-parallel. The wordlengths used in different parts can be exactly tuned. This system has been a model for many other systems. The entry to this system is an assembler-like language to describe the DSP algorithm. However, no synthesis on the DSP level is considered.

- **Moval:**
  This is a system that has been developed and that is used at AT&T Bell Labs [Ligt86]. It is based on the symbolic VLSI system MULGA [West81]. It includes analysis, layout and verification tools. Four levels of representation are used, behavioral, data, space/time and hardware. It makes use of a bit-parallel data-flow architecture.

- **Spaid:**
  Spaid has been developed at the university of Waterloo, Canada, and is a part of their VLSI CAD tool package [Haro89]. This tool only covers the part of mapping and scheduling of DSP algorithms. The underlying parts of DSP synthesis and VLSI design are left out to other systems. However, the system is flexible and allows, a large amount of user interaction, addition of user specified functional units, etc. The synthesized architectures are based on multiple buses and functional units normally based on parallel arithmetic and a register bank (memory) assigned to each bus.

A methodology and a supporting synthesis system will be presented throughout this text. Due to the many design tasks involved and the rather limited resources for development, i.e. a small group of people, etc., this synthesis system is purely experimental. It is far from being any complete product and from being a fully automatic system. Instead, all the tasks, from filter approximation to layout generation are treated. It consists of a collection of already existing tools, both commercial and university software, and some special purpose tools written within the frame of the project. The complete framework and all the tools therein are presented in Chapter 7.

# chapter 2. THE PROPOSED METHODOLOGY

*The outline of a methodology for design and VLSI implementation for DSP algorithms is discussed in this chapter. All required steps are introduced and various necessary choices, criteria and limitations are explained. The approach based on the wave digital state-space filters and the distributed arithmetic is introduced.*

## 2.1. Introduction

An appropriate design strategy is needed to reduce the design effort, which otherwise can be prohibitive. It will also bring down the complexity and thus reduce the probability for design errors. Knowledge of the design process in both the DSP and the VLSI field is essential when defining such strategy. This knowledge is sometimes translated into expert-systems or into specialized synthesis systems. Only the second type of system will be considered here.

By introducing limitations in both the DSP and the VLSI field, the task will be significantly simplified. Before doing this the purpose and the goal of the strategy must clearly be defined.

### 2.1.1. Goals and Criteria

The basic aim of the presented methodology is to show a simple but very powerful way for ASIC implementation of a class of DSP algorithms. Efficiency is obtained by introducing some restrictions. Otherwise, it would be impossible to define a precise strategy. Furthermore, used architectures

could not be limited to a few different configurations, and the ASIC implementation task would be very cumbersome. There exists a strong relation, or trade-off, between generality and efficiency. Strong limitations have been introduced here, without therefore making the approach too limited.

The first step to take is to define the goals and the criteria for the methodology and to find out the necessary limitations. The main goals and criteria that have been used here are listed below.

- **Generality:**
  The used approach should be as broad as possible without making the methodology too complex. The range of application fields covered by the strategy should be as large as possible, both concerning speed and types of DSP tasks. This criterion should be kept in mind both when finding suitable DSP algorithms and selecting architectures. However, it is here limitations or constraints will be introduced.

- **Flexibility:**
  This point is in some sense similar to the one above, but here the flexibility for a given task is considered. For instance, if a digital filter is designed it is essential that different solutions can be generated all with slightly different parameters, both considering synthesis of the DSP algorithm and the implementation. This includes accuracy, speed, I/O-formats, etc.

- **Efficiency:**
  This is perhaps the most indisputable goal. Especially today when the competition from commercial available DSP chips is very hard, it is extremely important to design efficient ASICs. This does not only imply efficiency of the used DSP algorithms and on the hardware level, but also efficiency of performing the design task itself. Commonly, the efficiency is measured as the performance of a developed DSP system compared to its relative cost.

- **Simplicity:**
  Keeping the strategy coherent and well structured, i.e., using the same steps for a wide range of applications will simplify the design task. Restricting the width, or the span, of the methodology will directly simplify all algorithms and routines used in different steps. If the strategy itself is simple and well structured, it will also be easy and simple to use.

- **Robustness:**
  It is desirable for the methodology to be robust or even follow a so called *"correct by construction"* approach. This means that strictly following the strategies defined by the methodology, implementations for all the included range of applications should be correct. This requires that the methodology is appropriately defined from the beginning. Another aspect of robustness is that the resulting hardware should act correct under all possible conditions. The chosen classes of architectures and DSP algorithms will have an enormous influence on the robustness.

- **CAD suitability:**
  Last but not least it is important that the methodology is supported by CAD tools. A design and implementation approach that is not supported by a CAD environment, especially in this field, cannot claim to conform to any of the points given above. The methodology must be defined such that it is feasible and simple to define and develop these tools. It is as well important that already existing CAD tools and systems can be adapted to, and incorporated in, the system.

The choice will clearly be a delicate trade-off between generality and efficiency. Clearly, taking other factors into account, such as simplicity, and CAD suitability, it is feasible to introduce relatively strict limitations. The exact choices that have been made on different levels will be described later on, but first an overview of the complete design and implementation task follows.

### 2.1.2. Design and Implementation Task

The complete synthesis task consists of three basic levels, *synthesis*, *realization* and *implementation*, figure 2.1. A large amount of sub-tasks, that are dependent on the used approach, should be solved on each level. Generally the following tasks should be solved in an ASIC synthesis system:

- **Synthesis**
  Here different sub-tasks should be performed such as, the approximation of a transfer function, realization of a precise description of a DSP algorithm. Transformation, tuning optimization and verification of this DSP algorithm is a very involved process that usually follows. Basically a precise and correct description of the algorithm, i.e., a *behavioral description*, should be the final result. For the definition of the methodology, the decision to make is not how to synthesize but which DSP algorithms to synthesize.

- **Realization**
  The functional description of the DSP algorithm should be translated into a *structural description* constructed by the *architectural components* given by the methodology. This translation or *mapping* contains operations and procedures such as transformations, optimization, tuning, binding, scheduling and verification. To perform this efficiently the target architecture and all its implementation parameters must be clearly defined in advance by the methodology.

- **Implementation**
  Finally, the implementation follows where the structural description is transformed into a physical, or *geometrical description* in case of an ASIC approach. If the structural description only contains components that exist in a library, as basic cells or as modules, the first part of this task is straightforward. This corresponds to the binding of each structural component to a geometrical. The scheduling defines the sequence and the timing that will be used. Module generators are used to construct the composed modules and finally, a placement and routing procedure completes the implementation step. Simulation, verification and test-vector generation are other important sub-tasks. A non-negligible task is the development of cell libraries, concerning both design and verification.

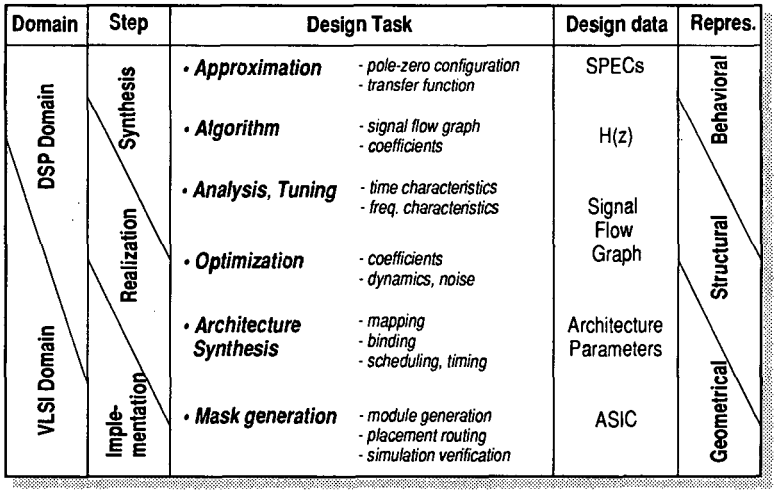



*Figure 2.1 The design and implementation task.*

Unfortunately, it is sometimes very difficult to situate some of the sub-tasks, they can be considered members of more than one level, cf. figure 2.1. For instance, the development of the DSP algorithm can be placed in both the synthesis and the realization step. The *architectural synthesis* is sometimes considered to be part of the realization step and some times it is considered to be a pure implementation step. In fact, the whole realization step can be considered to be a part of both the DSP domain and the VLSI domain.

At the very top and the very bottom of figure 2.1, i.e., in fields that are clearly within either the DSP domain or the VLSI domain, many existing strategies and methodologies can be found. The problem, however, is that they are usually defined independently of each other and it is not possible to find any natural relation between them. For this reason, the main effort of defining a new methodology should be made to bridge these two domains.

Figure 2.1 shows a very simplified model, in fact very often an iterative process is required. The model described by Gajski's *Y-chart* [Gajs83, Buri85] gives a more precise view, figure 2.2.

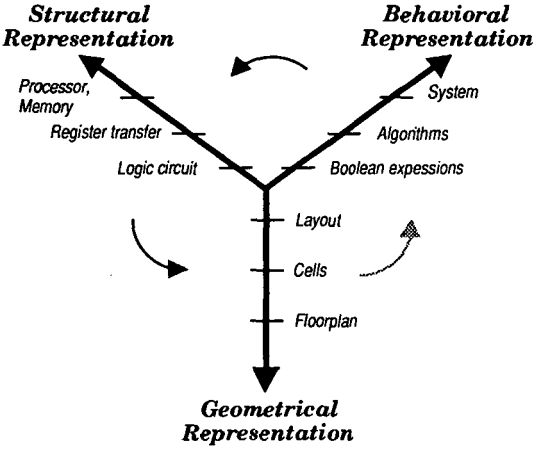



***Figure 2.2*** *Gajski's Y-chart describing the design task.*

In this model the design starts in one branch, i.e., the *behavior branch*, moving towards the center corresponds to a *top down* approach. Iteration and feed-back can be performed as well. Once the initial task or tasks are finished in the behavior branch one moves over to the *structural branch* and

finally to the *geometrical*. This corresponds to the situation described above, but very often the design and implementation task can be viewed as a process moving in a spiral towards the center of the chart. This is especially the case when defining a methodology. The treated DSP algorithm depends on the target architecture that depends on the structural description that depends on the DSP algorithm, etc.

## 2.2. DSP Algorithms

It is possible to start on any of the branches in the Y-chart when defining a methodology. Here all choices made have been outgoing from the DSP domain. The criteria defined earlier can be used for choosing specific classes of DSP algorithms. Adaptive DSP algorithms are directly disregarded to enhance simplicity and efficiency. The choice made here is to emphasize only on the fixed DSP algorithms.

### 2.2.1. Definitions and Limitations

The definition of fixed DSP algorithms used here refers to algorithms that are *constant, linear and periodically repetitive*. They should also be *realizable* [Fett76] or *computable* [Croc75]. This roughly requires that any directed loop in the algorithm should have a non-zero positive algorithm delay. Furthermore, a restriction to treat only one-dimensional signal processing has been introduced. Extensions to the multidimensional case can be made especially if the algorithms are *separable* in the dimensions.

An even more appropriate description here would be, all algorithms that have a *constant state-space description*. That is, that they can be described by a state-matrix with fixed coefficients or elements. The last constraint can sometimes even be released to discretely alternated set of coefficient. It means that an alternation between some predefined set of coefficients can be allowed. Sometimes this later class of algorithms is referred to as *weakly programmable*.

The range of fixed algorithms includes many important and well-known classes of DSP algorithms. Fixed digital filters, discrete transforms, in fact, all linear and time invariant algorithms are examples thereof. The main property of those algorithms is that the exact behavior is known in advance and there are no data dependent or time dependent decisions, operations or change of algorithm (branching). The scheduling can be exactly defined beforehand and will not change, in other words only *fixed scheduling* is needed.

### 2.2.2. Choice of fixed DSP Algorithms

A given candidate of filter class is the *Finite duration Impulse Response* (FIR) filters. These filters have several advantages, they cannot sustain any parasitic oscillations, they can be designed to have theoretical linear phase response and they are widely used in different applications. They fulfill most of the defined criteria, they are generally useful, flexible, simple to design (many methods exist) and they are robust. Still, the FIR filters have some profound drawbacks. They are not efficient from a hardware point of view. Very high filter orders and thus a large number of multiplications are often necessary.

A recursive filter algorithm of *Infinite duration Impulse Response* (IIR) type, is generally more efficient. The only problem is the robustness. In the digital domain all arithmetic operations are unavoidably performed in a finite precision and thus not carried out exactly. This means that error or noise sources are introduced. In a majority of the recursive algorithms this can result in disastrous phenomena such as *limit-cycles* or *parasitic oscillations*. These malfunctions can be avoided by carefully choosing certain filter types.

The best kind of IIR filters which is known are the so called *Wave Digital Filters* (WDFs) [Fett71]. These filters can be designed having a guarantied stable response even under *looped conditions*. All known parasitic oscillations can be suppressed. They also perfectly imitate the behavior of the best classical filters, namely the doubly terminated lumped element filters. Generally they are also efficient from a hardware point of view. They have an exceptionally complete theory developed around them so they can be used in a large range of applications. Finally, they are relatively simple to synthesize, using results and methods form the classical network theory.

Another important group of DSP algorithms, that are very similar to the LTI filters, is the *discrete transforms*, or to be more precise, the linear discrete transforms. The most commonly used transforms today are the *Discrete Fourier Transform* (DFT) and the *Discrete Cosine Transform* (DCT).

The chosen DSP algorithms will be discussed in Chapter 3, specially the WDFs. Some of their properties rely on some basic concepts that are important to consider for the rest of the methodology.

## 2.3. Architecture

The application field is essential for the selection of architecture. Given that only fixed DSP algorithms are treated, it is a considerably simpler task to define the architecture.

### 2.3.1. Criteria and Constraints on the Architectural Choice

The main strengths of ASICs are their very high potential performance, their reliability, their suitability for a high volume production (low discrete component count) and the difficulty to copy them. On the other hand, considering the complexity of development and design, Micro- or DSP-processor based implementations usually represent both cheaper and simpler solutions. However, the *performance* has been the most decisive factor here.

The *cost* is usually defined as a measurement of the required chip area and/or a complexity measurement for the architecture. This is normally the most common and important cost factor. Other cost measures to take into account are the development cost (i.e., the design time), the cost for testing and verification and perhaps a calculated cost for redesign of some faulty parts of the chip. The only systems considered here are those that can fit on one or on a few chips. This influences the choice of architecture.

Some *physical constraints* can also be imposed on the architecture. Power dissipation is one typical constraint. Robustness and reliability in general and more specifically under conditions such as cosmic radiation, can be other criteria. Should dynamic logic or static logic be used? Of course many of these criteria or constraints are also depending on the implementation and the available process technology.

*Modularity* and *regularity* are additional factors that improve the costs and the flexibility. It is also much simpler to design supporting CAD tools for a regular architecture.

### 2.3.2. Chosen Architecture

Comparing and measuring the criteria against each other it appears that architectures based on bit-serial arithmetic units and bit-serial communications are favorable from many points of view. They are selected here because of the propitious ratio between size and performance. All kind of *carry-ripple-chains* or *carry-look-ahead* logic can be completely avoided resulting in a high basic clock-rate. The main drawback of the bit-serial

architecture is normally the flexibility, e.g., the problem to implement decision logic. Due to the fixed nature of the treated DSP algorithms, this is of no significance here.

Another observation is that all the selected DSP algorithms can be represented in form of *innerproducts* (IP), i.e., they all have a state-space representation accordingly to the used definition. Taking the IP as the basic arithmetic operation has some advantages. First a minimum number of explicit operations are required and if they can be performed with adequate numerical precision, a minimum of quantization noise is achieved.

These last points can be provided using a technique called *Distributed Arithmetic* (DA) proposed by Croisiere et al. [Croi73] and Peled and Liu [Pele74]. Using a realization based on a *Carry-Save Adder* (CSA) keeps all the advantages of bit-serial arithmetic. The hardware requirement is extremely low for a wide range of applications. A higher amount of parallelism is also obtained when the innerproduct is the basic operation instead of explicit multiplications and additions. All the details on DA and an architecture based on the DA processor will be discussed in Chapter 4.

## 2.4. Realization

The next part, and probably the most important, is to define how the selected DSP algorithms should be mapped onto the target architecture. Given the restriction to treat only fixed DSP algorithms, the fact that they all can be represented in state-space form and the chosen DA architecture, this step is straightforward in the simple case. Each state-equation represents an IP that can directly be mapped onto a *Distributed Arithmetic processor* (DAP). The calculation of the IP is scheduled once per sample-interval on this processor.

### 2.4.1. State-Space Transformation

The first step of the realization is to translate the DSP algorithm into a suitable and general form. The *state-space representation* has been chosen, since the selected DSP algorithms were defined to be describable in the *state-space* (SS) form, and given that the chosen architecture supports IP operations. Of course, many DSP algorithms do not directly appear in SS form. Therefore, a transformation step is necessary after the initial synthesis step.

The state-space representation of any realizable fixed algorithm can be obtained in a straightforward manner. A common description of a fixed DSP

algorithm is the *signal flow graph* (SFG), figure 2.3a. This system can be viewed as a network containing adders and coefficient multipliers, figure 2.3b. The variables or signals stored in the delay-elements together with the inputs and outputs are called *state variables*. The new values of these variables can now be calculated as a linear combination of the old values. Representing this as one difference equation per variable the *state-space matrix* is obtained, figure 2.3c.

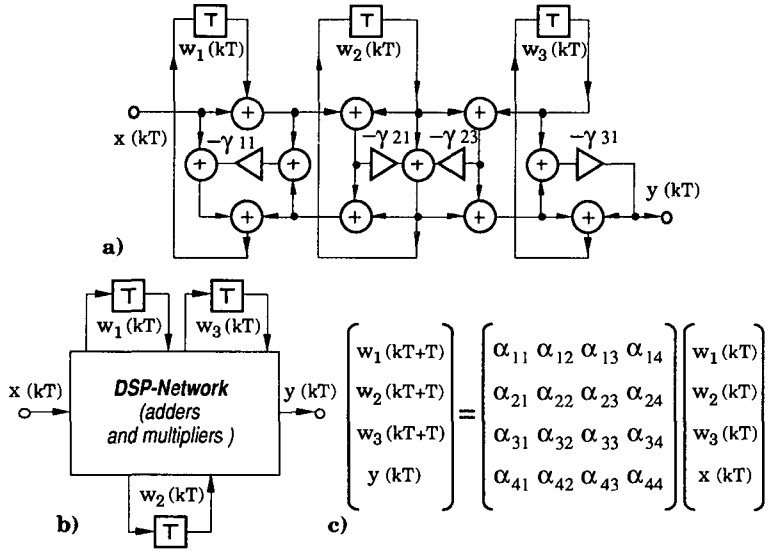



***Figure 2.3*** *a) Signal flow graph representation of a digital filter*
*b) State-space view*
*c) State-space matrix representation*

### 2.4.2. Features using the State-Space Representation

The number of variables in a canonic SS representation is always the minimum to fully describe the system. With one IP per variable where a IP corresponds to a single operation, it immediately follows that the number of arithmetic operations is minimal. Furthermore, starting from a situation where all the old states are known, all the subsequent states of the system, i.e., all the new values of the state-variables can be calculated concurrently. This means that the state-space representation is maximally parallel.

State-space representation offers a uniform, consistent and general description of various DSP algorithms. It also considerably simplifies the mapping and scheduling of the DSP algorithms. Using an implementation based on DAPs where only a final quantization is performed the representation is also very accurate.

### 2.4.3. Constraints on the SS Transformation

It is often assumed that transforming a DSP algorithm from one form to the other preserves all the characteristics. This is true if the DSP algorithm is linear and constant. Any DSP algorithm having the same transfer function is equivalent under these conditions. If quantization, overflow and other inevitable non-linearities are introduced it is not any more necessary that they are equivalent. For example, it has often been shown that realizations of the same filter using different kinds of second order sections result in completely different properties regarding noise, dynamic range [Jack89] and limit-cycles.

To preserve the essential part of the original DSP algorithms numerical properties it is important to follow certain rules when changing or transforming the representation. To accomplish this some methods and algorithms have been introduced. This is one of the most important part of the presented methodology and it will be discussed in Chapter 4 and 7.

## 2.5. The Proposed Methodology

### 2.5.1. Outline of the Methodology

When the basic limitations, constraints, choices and definitions have been made, the outline of the proposed methodology can be presented. A model the design and implementation task specific to the methodology is given in figure 2.4.

A short summary of the sub-tasks used in this methodology follows. This way it is easier to see the context of the separate presentations of different parts in other chapters.

A general remark is that even if not explicitly stated, simulations and verification on each level are essential. Furthermore, the presented framework should not serve as a *silicon compiler*, but rather as a special purpose tool-box for the designer. Hence, human interaction remains vital on each level.

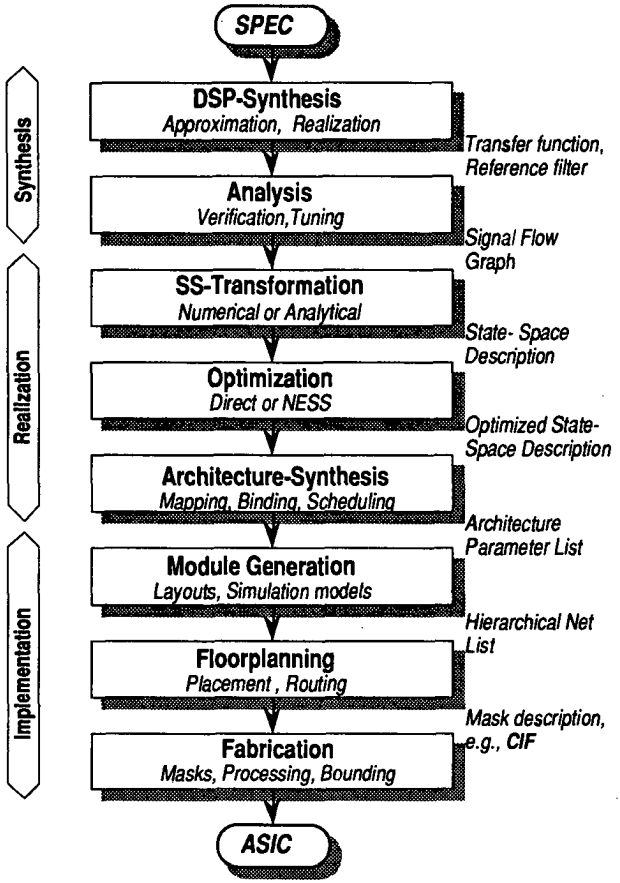



***Figure 2.4*** *Overview of the framework*

### 2.5.2. DSP Synthesis Tasks

The first part considered is the synthesis step, where specifications are translated into *approximation tolerances*. In the process that follows, a transfer function that stays within these tolerances is searched for using an appropriate algorithm. Normally a transfer function with the corresponding pole-zero configuration is the result of the approximation phase.

Many classical or more recent algorithms and software tools can be used. Concerning the WDFs, analog or classical synthesis tools are often required. The only inconvenience is that most of these programs use different formats both for input and output.

After the approximation the realization of a structure representing the found DSP algorithm follows. The SFG is the most commonly used representation.

In the analysis, verification and tuning step, the synthesis is verified by explicitly checking that different responses are within the given tolerances. This can seem to be superfluous if a proper approximation and realization algorithm is used. On the other hand, when dividing the problem and designing subsystems it is necessary to check global specifications. Sometimes an approximation algorithm only handles one type of tolerances, the others must be treated explicitly afterwards. Other special properties and characteristics of the obtained DSP algorithm can also be investigated and tuned. If something is found not to be satisfactory the previous synthesis step is repeated.

The analysis of interest can be divided into time analysis and frequency analysis. For the first kind, effects and responses of different wave forms can be investigated, for instance the impulse and the unit step responses. Naturally it is more interesting if there exists a possibility to make a *bit-true* simulation where possible non-linearities are considered. Different finite wordlength effects can then be investigated. The frequency domain analysis usually includes magnitude, phase, group-delay, noise and sensitivity analysis.

Due to the transformation following the synthesis in this framework it will be important to perform analysis after each design phase. This step, or procedure should contribute to the achievement of a correct and well-tuned description of the DSP algorithm.

### 2.5.3. Realization Tasks

DSP algorithms that are not already represented in form of a state-space description will be transformed into this form. In the general case, this transformation can be performed numerically. In this case, the values of the state-coefficients are calculated from the numerical values of the coefficients in the original structure.

To obtain a *Numerically Equivalent SS-transformation* (NESS) the state-coefficients must be derived from the quantized coefficient values of the original DSP algorithm. If optimization of the state-coefficient should follow the transformation explicit relations between the state-coefficients and the original coefficients are required. For this purpose, it has been necessary to develop some special tools, SFGkernel [Sjös89b] and SFD [Anso91a]. These will be described in Chapter 7.

The coefficient optimization is very dependent on the previous step. The objective of the optimization is to find state coefficient that can be represented using few bits and thus minimizing the hardware requirement. In the general case this can be solved. Many existing tools and algorithms can be used. The usual constraint on the optimization is that the responses of the DSP algorithm should stay within the tolerances given by the specifications.

When a numerically equivalent state-space representation is used, the optimization is performed under additional constraints. The original algorithm coefficients should be quantized such that minimal SS-coefficients are obtained. This guaranties that the same numerical properties as the ones of the original structure are maintained. Some specialized tools and algorithms are discussed in Chapter 7.

The *architecture synthesis* implies a translation of the final detailed DSP algorithm description (the behavioral description) into terms of hardware components (the structural description). In this case each innerproduct could be mapped onto a DAP. Normally, this one-to-one mapping is not optimal, it is advantageous to map more than one IP onto each processor. A scheduling step is then required to determine in which order IPs should be calculated. An advantage of the methodology is the facility and straightforwardness of this mapping. This can otherwise be the most crucial part of a synthesis system.

However, if multiplexing between channels and multi-rate systems are considered this task is no longer trivial. This demands that some *mapping rules* are defined to describe the different possibilities to bind operations (IPs) to the processors (DAPs) and the limitations that exist. To evaluate the feasibility of a found solution another part is needed, namely *estimators*. Mapping rules and estimators are the basic components of a tailored tool under development, but still a fair amount of heuristics and user interaction is required. The final structural description used in this framework has been named the *Architecture Net List* (ANL).

### 2.5.4. Implementation Tasks

The final *implementation* and *layout generation* step follows. These are fairly straightforward steps, considering the chosen architecture. Here full custom layout have been used for designing each basic building block or module. Globally an unconstrained layout strategy is used, which basically signifies that each module can have any size or any shape. The task of implementation can be divided into three parts:

- **Library development:**
  A non-negligible point is the development of the cell library and of the simulation models. This has, in this case, been performed using a full custom design approach. All cells have been developed giving an equal significance to the required area and to the obtained performance. Other criteria, such as for example power dissipation, have been given a secondary importance in the development. The library development task is not shown in figure 2.3 because it is outside the main stream of the design and implementation tasks.

- **Module generation:**
  The derived ANL is translated into input parameters for the *module generators*. Using the developed cell library these module generators produce all required modules, e.g., DAPs, memories, quantization units, I/O logic and a control unit. The ANL can also be translated into a net-list usable to the actual VLSI CAD system and can then be simulated using a software-model generated by one of the module generators.

- **Floorplanning:**
  The net-list also serves as the input to the floorplanning, which is the phase where the final chip layout is generated. Floorplanning includes both placement and routing and this can be done using either automatic, semi-automatic or manual approaches. Currently a semi-automatic strategy is used, this gives a reasonable trade-off between efficiency and complexity.

# chapter 3. DIGITAL SIGNAL PROCESSING AND WAVE DIGITAL FILTERS

*In this chapter the general properties of fixed DSP algorithms are discussed and different finite arithmetic effects are explained. Some families of carefully selected DSP algorithms are also discussed; in particular, the Wave Digital Filters will be introduced. The basic concept of the WDF will be treated and some interesting sub-classes are also viewed.*

## 3.1. Aspects on DSP Algorithms

In the preceding chapter, the criteria for the choice of DSP field and related algorithms were defined. Here, the field of signal processing will be treated and the chosen classes of DSP algorithms will be discussed in detail.

### 3.1.1. Constant, Linear and Periodically Repetitive Systems

The work presented here has been focused on *constant, linear and periodical repetitive DSP algorithms* defined as fixed DSP algorithm in the previous chapter. In particular the *one-dimensional* algorithms are treated. The fixed DSP algorithms include two of the most important groups of DSP algorithms namely; digital filters and discrete transforms. These classes of algorithms are well established today considering both theory and design. It is also believed that they will represent an important part of the future DSP systems

Many of the presented results could be extended to other types of fields such as *multi-dimensional* DSP, weakly programmable algorithms, non-linear applications, etc. However, it has not been the aim to cover the extension to such applications here.

### 3.1.2. Data Representation

The numerical behavior [Claa76] and the implementations are both strongly influenced of the data representation used in the DSP algorithm. Therefore it is of great importance to select a representation well suited for the application field.

The *floating-point representation* gives a high dynamic range for DSP algorithms, but it has some drawbacks. DSP algorithms often require high accuracy resulting in a long mantissa of the floating-point representation. Furthermore, the finite wordlength effects are difficult to analyse, complex models of the behavior are required. More important, the hardware requirements are very large compared to fixed-point arithmetic. Arithmetic operations, such as additions and multiplications are complex to perform and as a result of this, the speed and the performance will suffer.

The *fixed-point representation* is much simpler. Here analysis can be made using simple models of the behavior [Jack89] and the hardware requirements are moderate. The dynamic range is normally limited, but a sufficient *Signal-to-Noise Ratio* (SNR) can be obtained by increasing the data wordlength. In principle 6 dB of dynamic range for each extra bit added to the wordlength is gained. For most applications the wordlengths can thus be kept reasonably short. In fact, it is more often the required accuracy that determines the wordlength. Only fixed-point arithmetic will be considered in the rest of the text.

Other representations such as *Residue Number Systems* (RNS) have been proposed. They have show to have some very interesting properties especially for FIR filters and discrete transforms [Blah85] or for *block processed* recursive algorithms.

Fixed-point arithmetic can be represented using different kind of interpretations of the numbers [Flor63]. *Sign-magnitude, one's-complement* and *two's-complement* representations are some of the most used. The two's complement arithmetic has gained the largest interest due to its simplicity, both concerning arithmetic operations and implementation or hardware requirements. Multiplication is simple to realize, and overflow due to addition can be allowed in intermediate results if only the final result lies

within the given range. From a numerical point of view it is sometimes preferable to use another representation, e.g., sign-magnitude. Concerning quantization characteristics, rounding is often preferred over simple truncation due to a lower statistical noise. However, magnitude truncation has shown to be superior when trying to suppress parasitic oscillations.

## 3.2. Finite Wordlength Effects

Various phenomena occur inside a structure realizing a DSP algorithm because of the finite representation of the numbers. The effects of the *finite wordlength* or *finite arithmetic* can be divided into different groups.

First there are effects that can be modeled as noise, these concern quantization and truncation of signals. Moreover, signals are limited in magnitude and scaling aimed to reduce the probability for overflow will affect the dynamic rage. Other effects are also introduced by the finite representation of coefficients. Unfortunately, there are also more severe problems concerning the stability under finite wordlength considerations, i.e., non-linear parasitic oscillations.

### 3.2.1. Quantization Effects

The first effect is the *quantization effect*. All signals have unavoidably to be quantized to a discrete value when they are converted from an analog to a digital signal. This is made using for instance rounding or truncation characteristics and due to this a small error is produced. Under normal conditions these errors can be modeled as white uncorrelated noise. If rounding characteristics is used the noise will have zero mean value and a variance of $\sigma_q^2 = q^2/12$, where $q = 2^{-w}$ is the quantization step. If $h(pT)$ is the system impulse response then the contribution of this noise source to the output can be calculated as:

$$\sigma_{yq}^2 = \sigma_q^2 \sum_{p=0}^{N} h^2(pT) . \tag{3.1}$$

The noise for the stationary case is obtained if $N \rightarrow \infty$. From this expression the needed accuracy for the *analog-to-digital converter* (ADC) can be derived.

### 3.2.2. Round-Off Effects

The next kind of non-linearity is the *round-off effect* or sometimes also called *truncation effect*. In recursive algorithms the signal has to be

truncated or rounded at certain points. For example, after a multiplication between a coefficient of wordlength $w_c$ and a signal with wordlength $w_d$, both coded using two's complement, the result will consist of $w_c+w_d-1$ bits. In a recursive system it is absolutely necessary to truncate or round the result to $w_d$ bits, otherwise a continuous growth of the signal wordlength would be the result. Usually the same signal representation is desired at the input and output of a system, therefore this effect is also relevant in case of non-recursive algorithms.

The multiplier can be replaced by an ideal multiplier with an addition of a noise signal e(kT) at the output. This noise signal is normally modeled as a white noise source. According to the used quantization, the mean value and the variance can be calculated. This kind of noise is normally referred to as *round-off noise*.

The contribution from the noise source to the output can then easily be calculated as a variance at the output, similarly to the quantization noise. The variance for the total amount of round-off noise at the output can be calculated as

$$\sigma_{ye}^2 = \sum_{i=1}^{M} [\, \sigma_{ei}^2 \sum_{p=0}^{N} g_i^2(pT)] \tag{3.2}$$

where $g_i(pT)$ is the impulse response from the i:th location of a truncation operation (the i:th noise source) and M is the number of noise sources.

### 3.2.3. Overflow Effects

A signal expressed using finite accuracy has a maximal magnitude that can be represented. Once this allowed range is exceeded an *overflow* has occurred. This is one of the more severe finite wordlength effects. The overflow errors or the noise caused by overflow can not be modeled as easily as the truncation errors. The produced noise is obviously highly correlated to the signal, it corresponds to a large distortion of the output signal.

The overflow effects can be very severe for the numerical stability of the algorithm due to the strong correlation to the signal. Due to this it is necessary to avoid overflow. The use of an appropriate scaling can reduce this problem. Scaling on the other hand tends to reduce the dynamic range of the algorithm or to increase the required wordlength, especially if overflow should be totally avoided. Hence, a reduced dynamic range can be viewed as an indirect effect of overflow. Even if the DSP algorithm is scaled so that the zero-state response cannot cause an overflow one can never

guarantee the complete absence of overflow. In fact, hardware errors, i.e., faulty bits caused by external disturbances, can also cause overflows, even if the DSP algorithm is *safely scaled*.

Different overflow characteristics can be implemented. The most commonly used is the *saturation characteristic*. There, in case of overflow, the signal is replaced by the saturated value, i.e., a value of maximum magnitude. In case of two's complement arithmetic, if no caution is taken, a wrap-around characteristic is obtained. The smallest excess of the number range will cause a large error.

### 3.2.4. Coefficient Truncation Effects

The last of the finite wordlength effects is the coefficient truncation effect. This is a deterministic effect, i.e., once a set of coefficients is found the behavior of the filter will not change later on. The finite representation usually only leads to a static deviation of the transfer function or the filter response. However, it is feasible to use low sensitive filter structures to allow short coefficients, and thus saving hardware. Low sensitivity has also shown to be closely related to low noise [Jack76], which makes the choice to use these structures even more important.

Optimization of the coefficients can often be a large problem. It usually represents a problem where the global optimum cannot be found without searching the complete discrete-parameter space. Choosing a structure or a class of DSP algorithm with low sensitivity can be a first step to simplify the coefficient optimization.

The truncation of the coefficients can have an important influence on the numerical stability properties of the filter algorithm. Some classes of DSP algorithms are structurally bounded, which applies certain constrains on the coefficients. This will be discussed more in detail later.

### 3.2.5. Stability under Finite Arithmetic Conditions

Usually the statistical noise model described above can be used under the assumption that the noise is white and uncorrelated to the signal [Jack89]. The assumption is true for most signals, i.e., signals having sufficient amplitude and spectral content. On the other hand if the input signal is sinusoidal or of low level, the noise cannot any more be said to be white or uncorrelated, especially in case of constant input signals. So-called *parasitic oscillations* can occur from the round-off non-linearity in this case. Although these *limit-cycles oscillations* are small, they can be troublesome in some

applications and they always cause a loss of dynamic range. Different behavior can be found depending on the number representation, and whether rounding or truncation is used. It is important to observe that the *bounded-input bounded-output* (BIBO) stability requirement is no more sufficient to ensure a stable DSP algorithm. A list of different phenomena that concerns the stability under finite arithmetic conditions follows.

- **Small Scale Limit-Cycles:**
  This group of correlated noise is often referred to as *small-scale limit-cycles*, or *oscillations due to granularity*. They are simply caused by the non-linear round-off or truncation operation. Some models have been developed to understand the behavior, but these models are usually only valuable for some limited classes of filter structures [Fett75a, Barn77, Jack79]. Sign-magnitude truncations applied at certain locations in the algorithm tends to be efficient for suppressing these oscillations. Introducing randomness in the quantization characteristics can also be enough to avoid the limit-cycles in some structures.

- **Large Scale Limit-Cycles:**
  The second group, which is even more troublesome, is the *large scale limit-cycles* or the *overflow oscillations*, caused by overflows, round-off effects or by combinations thereof. These limit-cycles oscillate over the whole number range causing overflows. A reset or a power down of the system is needed to force the filter back into normal behavior. As mentioned before, scaling can reduce the probability for overflows but never totally prevents the occurrence of them. The overflow characteristics have a large influence on the behavior of these errors. In some filters the oscillations can be avoided simply using saturation arithmetic.

- **Periodic-Input Limit-Cycles:**
  If periodic input signals are considered the situation becomes even more complicated. The frequencies of periodic signals are usually irrational numbers, except for some trivial cases, and therefore they cannot be exactly represented. It is inevitable that the output will differ from the theoretical and this is a *periodic input limit-cycle*. For the simplest cases, i.e., constant input and input of period 2, these oscillations can sometimes be avoided by *controlled rounding*. A special case of these oscillations is the *zero-input limit-cycles*.

- **Forced Response Stability**
  This corresponds to a restrictive stability measure, but it is reasonable

to require the *forced response stability* for any DSP algorithm used in an application where absolute stability is mandatory. The forced response stability implies that any limit-cycle due to an overflow will start to vanish as soon as the input is decreased to a level where under normal linear condition no overflow would occur [Claa75, Samu82]. This stability is also sometimes referred to as *asymptotic stability*. Even if stability accordingly to the earlier points can be ensured for a large number of structures and algorithms, the forced response stability is not automatically obtained. In fact it exists very few classes of recursive DSP algorithms where this is fulfilled.

- **Stability Under Looped Conditions**
  Even if precautions are taken to avoid oscillations in a filter, the external environment can play an important role of causing non-desired oscillations. The digital filter or the DSP algorithm might be situated in a larger system with a global external feed-back, e.g., a telephone transmission link. If the system still is stable it has been named *stability under looped conditions* [Fett84]. It is generally very difficult to ensure stability under these conditions since the stability is dependent on the external environment.

Satisfactory analysis, or simulation tools, on a bit-true level, are needed to analyse and study these different undesired effects in a filter. However, the oscillations might not be that easy to produce in a simulation. Adequate knowledge and understanding are needed to be able to interpret the results and to find the most critical oscillations. Clearly the best way to avoid all of this is to select DSP algorithms where all these stability aspects are met. In the section 3.4, a class of filters is presented, where all the stability problems due to finite wordlength effects can be avoided, if some simple rules are applied.

## 3.3. Fixed DSP Algorithms

### 3.3.1. FIR filters

The simplest group of fixed filter algorithms are the FIR filters. They are interesting from many aspects especially the robustness. Filters of order $m$ can be described by the non-recursive difference equation:

$$y(kT) = \sum_{i=0}^{m} A_i \, x(kT - iT) \tag{3.3}$$

$A_i$ being real constants, or as a transfer function in the z-domain

$$H(z) = A_m z^{-m} + A_{m-1} z^{-m+1} + \ldots + A_0 \qquad (3.4)$$

FIR filters are often used because of the linear phase and the guaranteed stability under finite arithmetic conditions. The stability comes from the fact that these filters have a bounded impulse response and can be realized with non-recursive structures.

However, the problem is that often very high filter orders are needed to achieve the specified frequency selectivity. In fact, only transmission zeros in the z-domain are used to shape the magnitude function (3.4). In case of linear phase filters it can be shown that the impulse-response is always symmetric or anti-symmetric around the half of the filter length, that is, $A_{M-n} = \pm A_n$. Hence, the number of multiplications can be reduced to the half by pre-adding or pre-subtracting. Due to this symmetry it is possible to preserve the linear phase property also for a filter with quantized coefficients.

A common problem is the high overall delay through the filter due to the elevated filter order. In many cases this can be a prohibitive factor.

The classical method to synthesize FIR filters is the *window-function* technique. It starts with an inverse *Fourier-transform* of the desired (ideal) transfer function. The derived impulse response is shifted forward in time and then multiplied with a window function, which ensures that a finite duration impulse response is derived. However, the final frequency response is distorted due to the truncation of h(kT). These effects can be tuned using different window functions, e.g., Hamming, Hanning, Blackman, etc. [Oppe75, Jack89]. However, the effects will always exist.

Efficient FIR realizations can also be obtained using the Discrete Fourier Transform. For example, *frequency sampling* techniques, represent a method to synthesize FIR filters describing the impulse-response in term of its DFT coefficients, [Rabi75].

Other more direct methods are based on *equiripple* designs. McClellan and Parks have proposed the most commonly used algorithm for equiripple approximation [McCl73]. It is based on Remez's exchange algorithm. The use of these algorithms usually leads to better solutions, concerning complexity, than window techniques.

Non-standard solutions for FIR filters have been published, for example the *Interpolating FIR*, IFIR filters [Sara85]. The so-called recursive FIR filters are another class, which are efficient from a hardware point of view

but they also have all the draw-backs of recursive filters, such as stability problems due to finite wordlength effects.

### 3.3.2. IIR Filters

Infinite impulse response filters are preferable to use where it is possible, due to their superior frequency selectivity. Usually IIR filters can be implemented very efficiently, with a low number of arithmetic operations and few memory elements. The recursive difference equation describing a IIR filter is a generalization of (3.3). ($A_i$ and $B_j$ are constants)

$$y(kT) = \sum_{i=0}^{m} A_i\, x(kT - iT) + \sum_{j=1}^{n} B_j\, y(kT - jT) \tag{3.5}$$

The transfer function can now be expressed as a ratio between two polynomials in z.

$$H(z) = \frac{A_m z^{-m} + A_{m-1} z^{-m+1} + \ldots + A_0}{1 - B_n z^{-n} - B_{n-1} z^{-n+1} - \ldots - B_1 z^{-1}} \tag{3.6}$$

The main argument against IIR filters is the occurrence of stability problems under finite wordlengths conditions. This problem can be solved using certain classes of filters where the stability can be ensured. IIR filters can not be designed to have exactly linear phase, they will always have some phase distortion. This second problem can be partly solved by cascading group delay equalizers with the original filter or by using special approximation techniques when synthesizing the filters [Kuno88]. Then an approximately linear phase in some desired parts of the frequency band can be obtained.

The most straightforward realization of IIR filters is the direct form [Rabi72]. The polynomial coefficients in the numerator and denominator of (3.6) are directly used in the realization. However, a better solution from a numerical point of view is to subdivide the transfer function into a product form [Jack70]. The order of the sub-transfer functions should be as low as possible. It can be shown that cascaded sub-filters have superior sensitivity properties over direct form filters. The sensitivity is also closely related to the noise level in the filter. Thus, it is important to use low sensitivity structures. The minimum order of these sub-transfer functions will be one and two, if real coefficients are desired.

$$H(z) = A_0 \prod_{i=1}^{L} H_i(z) \tag{3.7}$$

$$\text{where} \quad H_i(z) = \frac{(1 + a_{1i}z^{-1} + a_{2i}z^{-2})}{(1 - b_{1i}z^{-1} - b_{2i}z^{-2})} \quad \text{or} \quad H_i(z) = \frac{(1 + a_{1i}z^{-1})}{(1 - b_{1i}z^{-1})} \tag{3.8}$$

A large number of permutations giving different numerical behavior can be found when cascading the sections. First of all, the poles and zeros can be paired differently to form the sections. For a N:th order filter, N/2! combinations exist. The ordering of N/2 sections can be made in N/2! ways. Thus, $N/2!^2$ different combinations can be found, given that the same structures for the first and second order sections are used everywhere. It is not obvious to find the correct combination even if some heuristics and sub optimal methods exist.

It is also possible to realize the sub-filters in parallel form. The transfer function can then be expanded into a partial fraction (3.9) where $H_i(z)$ is a first or second order section similarly to (3.8). Usually the parallel form suffers from poor stopband sensitivity while the passband sensitivity is low. For some implementations the parallelism inherent in this form is of interest.

$$H(z) = C_0 + \sum_{i=1}^{L} H_i(z) \tag{3.9}$$

A suitable filter should be insensitive to changes in the coefficient values and it should give low noise even for reasonable wordlengths. In fact, it can be shown that these two properties are closely related [Jack76].

### 3.3.3. Other kinds of Digital Filters

Many other digital filter structures exist. One particular class that has shown to be efficient and to have low sensitivity to coefficient changes is the so called *lattice structures*. The most known of these are the *Gray and Markel filters* [Gray73]. However, these will not be discussed here since it can be shown that similar and even better structures can be derived using an even more suitable class of filter.

The classical filter structures that are known to be best from sensitivity point of view are the doubly terminated ladder LC filters. A digital counterpart to them is desirable. It exists one class of digital filters that exactly imitates the behavior of the classical filters conserving the desired properties. These are the so called *Wave Digital Filters* (WDFs) [Fett84].

### 3.3.4. Discrete Transforms

The discrete transforms especially the *Discrete Fourier Transform* (DFT) have shown to be very useful in the DSP domain. It is most commonly used for spectral analysis, but it can be used in a number of other fields. To mention some examples, it is applicable to compute convolution, correlation, for estimation, etc.

Due to the importance and to the relatively high computing requirements of the DFT numerous *fast algorithms* have been developed, so-called *Fast Fourier Transform* (FFT) algorithms. Most of these have only been developed to reduce the number of multiplications. This aspect does not have any direct importance, for methodology presented here, since the DFT would be computed using a different direct method using innerproducts and not discrete multipliers. Nevertheless, FFTs have shown to be very important, it is due to these algorithms discrete transforms have gained their popularity. Fast transform schemes can be useful especially for higher order transforms.

In other application fields it has shown to be better to use other transforms. This is typically the case in bandwidth compression of speech and images [Jaya84]. Here it has been shown that the *Karhunen-Loeve Transform* (KLT) is statistically optimal, but unfortunately this transform is adaptive, or dependent on the data (or to be exact dependent on signal statistics) and therefore very difficult to realize in hardware.

Instead, a transform that has gained a large interest in this domain is the *Discrete Cosine Transform* (DCT) [Rao 90]. The DCT has shown to have the properties that come closest to the KLT of all known linear transforms. The DCT can be implemented very efficiently using the architecture and the methodology presented here [Defi89, Sjös90, Defi90].

## 3.4. Wave Digital Filters

The WDFs were first proposed by Fettweis in 1971 [Fett71]. These filters are based on an indirect synthesis. First an analog reference filter, or a model filter, is designed. The structure of the reference filter is later on transformed into a digital structure. In fact, WDF is a class of digital filters that are closely related to classical networks. It is due to this relationship these filters have shown to inherit many interesting properties.

Also other digital filters related to classical networks have been proposed, but they sometimes suffer from realization problems, i.e., *directed delay-free loops* and finite wordlength stability is not always guaranteed.

Several properties of the classical filter networks are dependent on the *passivity* and the *losslessness* of the two-port itself. A WDF design, can be shown to preserve these properties even under finite wordlength conditions. The theory of WDFs will only briefly be introduced here. It is important to discuss some of their basic properties in order to show how to obtain WDF in state-space form. For a complete description [Fett86] is recommended.

### 3.4.1. Basic Concepts, Frequency Variables and Signal Quantities

The correspondence between a WDF and its reference filter can be established in the frequency domain. Therefore, a complex frequency variable $\psi$, sometimes referred to as *Richards' variable* [Bele68] is defined:

$$\psi = \frac{z - 1}{z + 1} = \tanh(\frac{sT}{2}), \qquad z = e^{sT} \tag{3.10}$$

This variable is dimension free, but this is not restrictive, since most filters anyway are synthesized using normalized frequencies. The correspondence in frequencies can now be derived as:

$$\phi = \tan(\frac{\omega T}{2}), \qquad s = j\omega, \ \psi = j\phi \tag{3.11}$$

The property (3.11) implies that real frequencies in the reference filter domain correspond to real frequencies in the s-domain. Furthermore, the Nyquist range is precisely mapped in a one-to-one correspondence into the reference domain.

The main difference, compared to other approaches that are simulating classical filters in the digital domain, is the choice of signal quantities. Instead of using voltage and current, a concept from the classical circuit theory, the *scattering parameter theory* [Bele68, Bahe84] using waves, is introduced. The waves can either be *voltage waves, current waves*, or *power waves*. It turns out that there are no real differences between the first two. Only a dual circuit is derived if current waves are used instead of voltage waves. It can also be shown that the use of power waves is less suitable. From here on, only voltage waves will be considered.

### 3.4.2. The Port

A port is characterized by its impedance, i.e., the current flowing through the port and the voltage over the port. In Wave Digital Filters the port is characterized by an *incident wave*, A, a *reflected wave*, B, and its *port-resistance*, R, figure 3.1.

The definition of the port is valid for the instantaneous case, as well for the steady-state case.

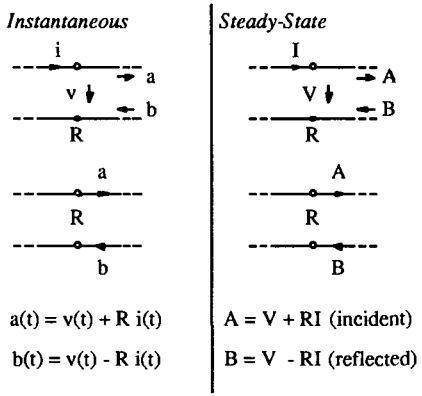



***Figure 3.1*** *Representation of a port, reference domain and digital domain.*

### 3.4.3. The Reference Filter

It can be shown that the sensitivity to changes in coefficient values and the noise behavior are closely related. Therefore, it is important to select reference filters with low sensitivity, to achieve a digital filter with short coefficients and with low noise. For classical filters, the *doubly terminated reactance networks* have the best sensitivity properties [Bahe84, Sedr78, Teme77], and should therefore be selected as reference filter.

Another important point when choosing reference filter is the realization criteria, it should not contain any sub circuit that creates delay-free loops in the digital domain. Fortunately, this is fulfilled for many classical circuits.

### 3.4.4. Element Transformation

The first relation to establish is the digital counterparts to the reference filter elements. The corresponding digital network elements are easy to find using the definition of the voltage waves. Take a capacitor, it is described in the reference domain by the steady-state equation:

$$V = RI/\Psi \tag{3.12}$$

Using the bilinear transformation defined in (3.10) yields:

$$zV - V = zRI + RI \Leftrightarrow z(V - RI) = V + RI \tag{3.13}$$

Using the definition of the voltage wave this can be identified to be:

$$zB = A \Leftrightarrow B = z^{-1}A \tag{3.14}$$

Finally describing (3.14) in the discrete time domain gives:

$$b(kT) = a(kT-T) \tag{3.15}$$

This can be derived directly using instantaneous equations.

In Table 3.1 some elements in the reference domain are shown with their counterparts in the digital domain. Similarly to the capacitor the indictor becomes a delay element but with a sign inversion, the resistor a wave-sink where all the energy is consumed. For the short-circuit a sign inversion is obtained, this can be viewed as a phase shift of 180°, while the open circuit reflects the wave without phase shift. The resistive voltage source becomes a wave-sink at one terminal and a wave-generator at the other.

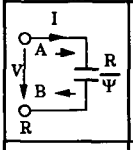
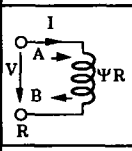
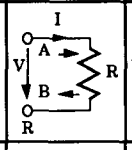
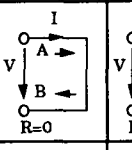
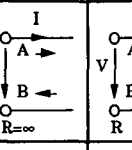
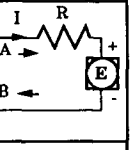
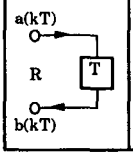
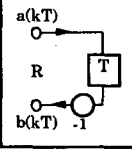
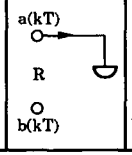
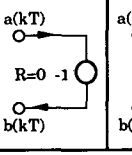
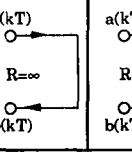
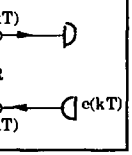
| Capacitance | Inductance | Resistance | Short-circuit | Open-circuit | Resistive Source |
|---|---|---|---|---|---|
| $V = RI/\Psi$ | $V = RI\,\Psi$ | $V = RI$ | $V = 0$ | $I = 0$ | $V = E + RI$ |
|  |  |  |  |  |  |
| $b(kT)=a(kT-T)$ | $b(kT)=-a(kT-T)$ | $b(kT)=0$ | $b(kT)=-a(kT)$ | $b(kT)=a(kT)$ | $b(kT)=e(kT)$ |
|  |  |  |  |  |  |

***Table 3.1*** *Element transformations, one ports*

Some other useful elements are shown in Table 3.2. The first is the so-called *Unit-Element*(UE) from the micro wave filter domain. This element has a characteristic resistance of R and a delay in each direction of T/2, i.e., half a sampling period. The QUARL is related to the UE and is sometimes more practical to use. By choosing $\Delta = T/2$, a full delay of T in one direction and zero delay in the other is obtained. Also the gyrator is very simply realized in the digital domain.

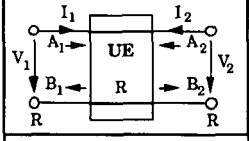
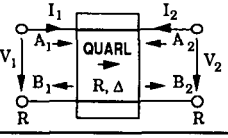
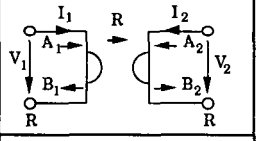
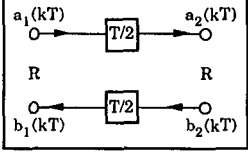
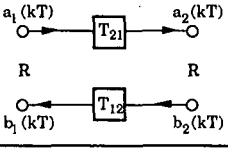
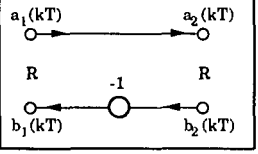
| Unit Element | Quasi Reciprocal Line | Gyrator |
|---|---|---|
| $K=\frac{1}{\sqrt{1-\Psi^2}}\begin{pmatrix}1 & \Psi R\\ \Psi/R & 1\end{pmatrix}$ | $K=\frac{e^{-s\Delta}}{\sqrt{1-\Psi^2}}\begin{pmatrix}1 & \Psi R\\ \Psi/R & 1\end{pmatrix}$ | $K=\begin{pmatrix}1 & -R\\ 1/R & 1\end{pmatrix}$ |
|  |  |  |
| $b_1(kT)=a_2(kT-T/2)$<br>$b_2(kT)=a_1(kT-T/2)$ | $b_1(kT)=a_2(kT-T_{21})$, $T_{21}=T/2+\Delta$<br>$b_2(kT)=a_1(kT-T_{12})$, $T_{12}=T/2-\Delta$ | $b_1(kT)=-a_2(kT)$<br>$b_2(kT)=a_1(kT)$ |
|  |  |  |

***Table 3.2*** *Element transformations*

The *circulator* is translated into simple interconnections, Table 3.3. One possible configuration of the *ideal transformer* is also shown.

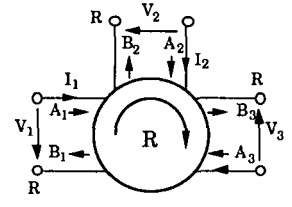
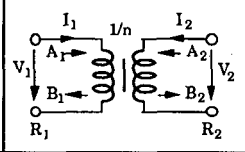
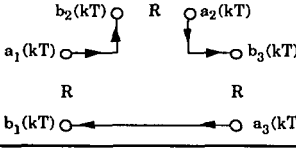
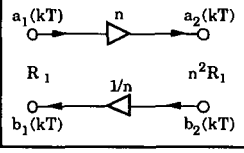
| Three-Port Circulator | Ideal Transformer |
|---|---|
|  | $V_2=n\,V_1$ $\quad I_1=-n\,I_2$  |
| $b_1(kT)=a_3(kT)$, $b_2(kT)=a_1(kT)$,<br>$b_3(kT)=a_2(kT)$ | $b_1(kT)=1/n\,a_2(kT)$<br>$b_2(kT)=n\,a_1(kT)$ |
|  |  |

***Table 3.3*** *Element transformations*

All the elements have a characteristic resistance, R, whose value is derived from the reference element. The consequence of R is that the elements cannot be directly connected. An interconnection network is needed to adapt the port-resistances in different interconnected ports to each other.

### 3.4.5. Interconnections, The Adaptor

To interconnect elements in the digital domain both terminals of connected ports should be connected. The terminal for the incident wave of one element must be connected to the terminal for the reflected wave of the other and vice versa. Furthermore, the port resistance must match for both ports. Usually the last criterion is not directly fulfilled and an adapting network is needed. This network should simulate Kirchhoff's laws of different configured interconnections. Therefore, it is necessary to define a new element called an *adaptor*.

### 3.4.6. The Parallel Adaptor

The ports of the elements can be connected either in parallel or in serial. A parallel connection of N elements can be represented in the reference domain as in figure 3.2.a, and in the digital domain as in figure 3.2.b.

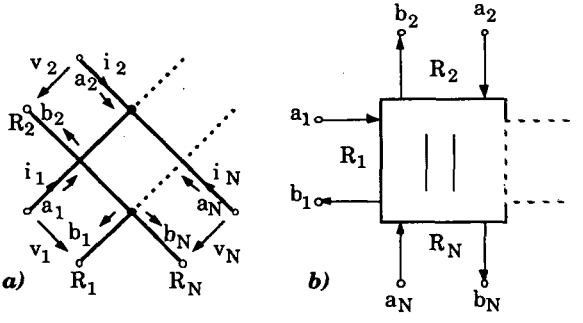



***Figure 3.2*** *Parallel connection of N ports.*
***a)*** *Reference Domain*
***b)*** *Digital Domain*

Kirchhoff's laws for currents and voltages give the following relations for the parallel connection:

$$v_1 = v_2 = \ldots = v_N\,, \qquad i_1 + i_2 + \ldots + i_N = 0 \tag{3.16}$$

Using the wave equations the following relations between the reflecting waves and the incident waves can be derived:

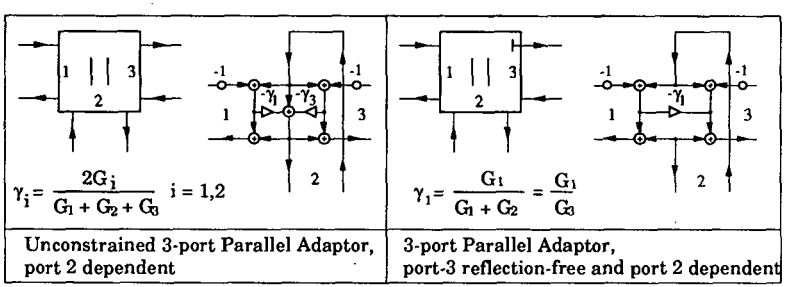
$$b_i = (\gamma_1 a_1 + \gamma_2 a_2 + \cdots + \gamma_N a_N) - a_i, \quad i = 1 \text{ to } N \tag{3.17}$$

$$\text{where} \qquad \gamma_i = \frac{2G_i}{G_1 + G_2 + \ldots + G_N}, \quad G_i = \frac{1}{R_i} \tag{3.18}$$

$$\text{and} \qquad 2 = \gamma_1 + \gamma_2 + \ldots + \gamma_N \tag{3.19}$$

This shows that the interconnections and the adaptation of the port-resistances, can be done by a digital network consisting simply of adders/subtractors and coefficient multipliers. Equation (3.19) shows that all the coefficients do not have to be implemented explicitly, one of them can be implicit and then the corresponding port will be called *dependent port*. Generally, for an n-port adaptor, n-1 multipliers, and 3n-3 adders are needed to implement the adaptor.

More concretely, for a three-port parallel connection, the digital network will have the structure shown in table 3.4. This network is called a *three-port parallel adaptor*.




| Unconstrained 3-port Parallel Adaptor, port 2 dependent | 3-port Parallel Adaptor, port-3 reflection-free and port 2 dependent |
|---|---|

***Table 3.4*** *Three-port parallel adaptors*

The unconstrained three-port adaptor contains only two coefficients since the third one is dependent, according to (3.19), and does not have to be implemented explicitly. Different types of *constrained* adaptors can also be derived. One of the most interesting adaptor is that with a *reflection-free port*. A reflection-free port is a port *i* where the reflecting wave $b_i$ is independent of the incident wave $a_i$, thus the parameter $\gamma_i$ becomes exactly one in that case. The constraint for this adaptor is that the port conductance in the reflection-free port is equal to the sum of all other port conductances for the adaptor. The reflection-free port is denoted with a vertical bar on the arrow corresponding to the reflected wave, cf. table 3.4.

### 3.4.7. The Serial Adaptor

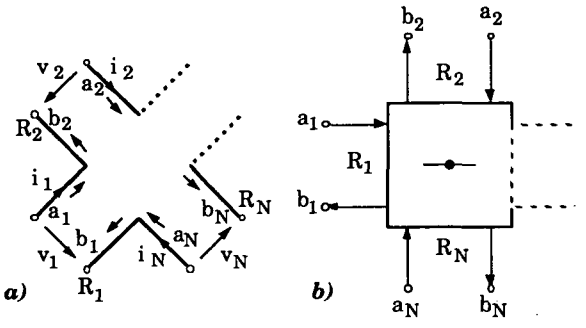



***Figure 3.3*** *Serial connection of N ports.*
***a)*** *Reference Domain*
***b)*** *Digital Domain*

A similar discussion can be made for the serial interconnection of ports. The main difference is that the sum of the voltages is zero and all currents are equal for serial interconnections.

$$b_i = a_i - \gamma_i(a_1 + a_2 + \cdots + a_N), \quad i = 1 \text{ to } N \tag{3.20}$$

$$\text{where} \qquad \gamma_i = \frac{2R_i}{R_1 + R_2 + \ldots + R_N} \tag{3.21}$$

Still equation (3.19) holds for the serial adaptor. The necessary number of circuit elements is also similar to the parallel adaptor.

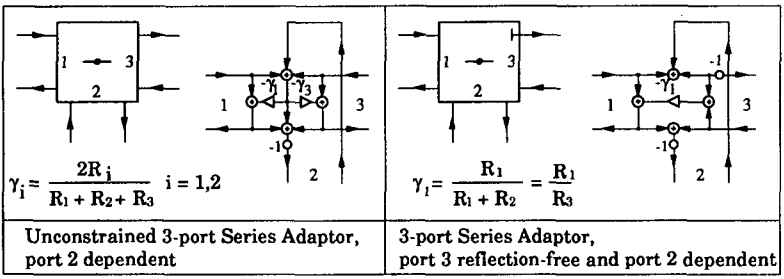



***Table 3.5*** *Three-port serial adaptors*

The flow-graphs for the three-port serial adaptor are given in table 3.5, showing both an unconstrained three-port adaptor and an adaptor with

reflection-free port. The reflection-free port $i$ is in this case obtained by calculating $R_i$ as the sum of the two other port resistances.

### 3.4.8. The Two-Port Adaptor

In some cases it is convenient to use two-port adaptors, i.e., in resonance circuits or in WDFs derived from unit-element reference filters. In the case of a two-port connection no real distinction can be made between series or parallel connection. Thus, serial, parallel or even serial-parallel two-port adaptors have been proposed [Fett75c]. They have some differences in the numerical behavior. Choosing one of the four basic two-port adaptors, shown in table 3.6, that gives $0 \leq \gamma \leq 1/2$, scaling for sinusoidal signals is automatically obtained [Gazs85]. Additional variants of the series/parallel two-port adaptor can also be derived.

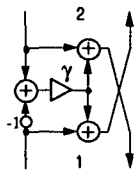
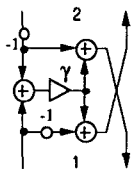
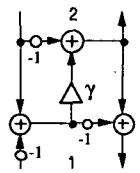
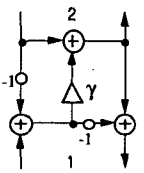
| Series/Parallel 1 | Series/Parallel 2 | Series | Parallel |
|---|---|---|---|
|  |  |  |  |
| $\gamma = \frac{R_1 - R_2}{R_1 + R_2}$ | $\gamma = \frac{R_2 - R_1}{R_1 + R_2}$ | $\gamma = \frac{2R_1}{R_1 + R_2}$ | $\gamma = \frac{2R_2}{R_1 + R_2}$ |

***Table 3.6*** *Two-ports adaptors*

## 3.5. Design of Wave Digital Filters

### 3.5.1. Synthesis

To design a WDF, classical filter synthesis and approximation methods can be used. First the digital filter specifications must be translated into an analog filter specification. This can be done using the simple relation from the bilinear transform in (3.10). An analog filter can then be realized. This should be a doubly terminated lumped element filter. An analogy can be used to transfer the filter into the reference domain. The reference filter can now be transformed element by element into the digital domain and the interconnections establishing Kirchhoff's laws are replaced by adaptors. Finally the coefficients in the adaptors can be calculated and a wave digital filter is derived.

The concept can be viewed like in figure 3.4. From a normal s-domain design an *analogy* can be used to transfer it to the reference domain. The reference filter can also be transformed into the micro wave filter domain with commensurate delay lines. Of course delay line filters can as well be used as reference filters.

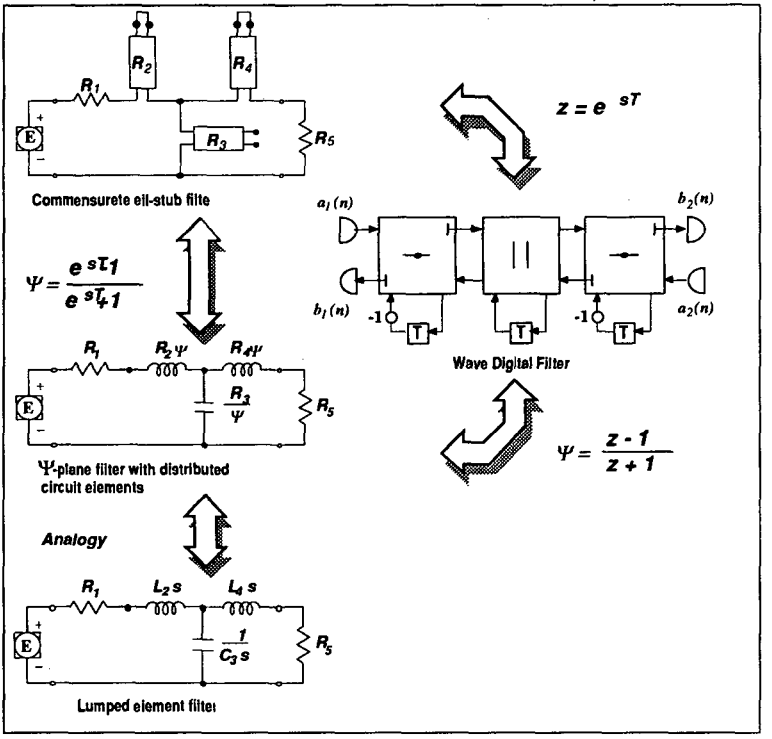



***Figure 3.4*** *Principles of Wave Digital Filter synthesis*

### 3.5.2. Transformations

Once a filter is derived in the reference domain, the transformation into a WDF can be undertaken. All the discrete elements are replaced by their WDF equivalents and the interconnections are replaced by the adaptors. Using the adaptor equations, the coefficient values can be calculated. Certain rules must be followed to avoid delay-free loops and to ensure the desired properties of WDFs.

One way to avoid delay-free loops is to make use of constrained adaptors with one reflection-free port. This clearly breaks up the closed delay-free loops in the structure. It can be done using the extra freedom choosing some of the port resistances. On the other hand if the reference filter is designed as a delay line filter with unit elements [Meer77], no delay-free loops will occur in the digital structure. Using the possibility to insert UEs or QUARLs, into the reference filter using *Kuroda's-identities* [Bahe84, Erik79] will also break up delay-free loops at an increase of hardware requirements. However, this method does not make use of the UEs filtering capability. It is better to use a synthesis directly leading to alternating RL components and UEs.

### 3.5.3. WDF Related to Symmetrical Lattice Filters

A class of wave digital filters, which has shown to be especially interesting, corresponds to the filters derived from *symmetrical lattice* reference filters [Fett74]. In figure 3.5 $Z_1$ and $Z_2$ correspond to the canonic (lattice) impedances and the network is doubly terminated with $R_0$. For reason of generality one voltage source is placed at each port.

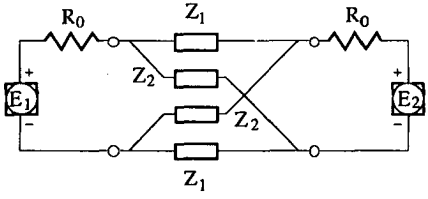



***Figure 3.5*** *Doubly terminated symmetrical lattice filter*

Calculating the voltage wave scattering two-port parameters one finds:

$$s_{11} = s_{22} = \frac{S_1 + S_2}{2} \tag{3.22}$$

$$s_{12} = s_{21} = \frac{S_1 - S_2}{2} \tag{3.23}$$

where

$$S_1 = \frac{Z_1 - R_0}{Z_1 + R_0} \tag{3.24}$$

$$S_2 = \frac{Z_2 - R_0}{Z_2 + R_0} \tag{3.25}$$

The parameters $S_1$ and $S_2$ are reflectances and $Z_1$ and $Z_2$ are pure lossless reactance functions then $S_1$ and $S_2$ clearly are all-pass functions. The relation between the waves and the scattering parameters gives:

$$B_1 = \frac{S_1}{2}(A_1 - A_2) + \frac{S_2}{2}(A_1 + A_2) \tag{3.26}$$

$$B_2 = \frac{S_1}{2}(A_2 - A_1) + \frac{S_2}{2}(A_1 + A_2) \tag{3.27}$$

This can directly be realized by a WDF shown in figure 3.6a. If $E_2 = 0$ in figure 3.5 it follows that $A_2 = 0$ and if the complementary output $B_2$ is not calculated the realization in figure 3.6b is obtained.

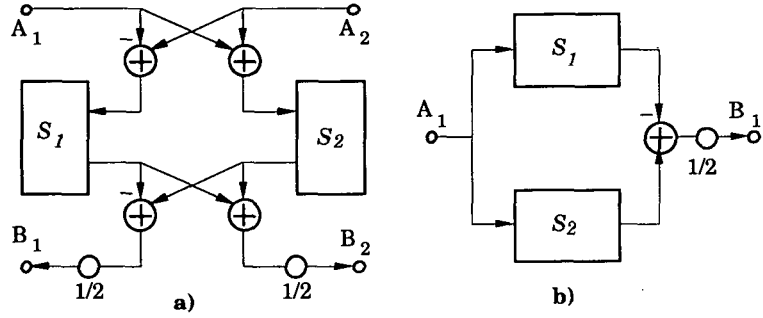



***Figure 3.6*** *Wave digital filter realizations of symmetrical lattice filter.*

Due the two parallel branches in this kind of WDF an increased sensitivity for the stop band is obtained while the passband sensitivity is extremely low. It has been shown that they still can be realized using very simple coefficients. An advantage of these WDFs is the parallelism given by the two separate branches.

The canonical impedances for the lattice reference filter can be derived from, for example, a symmetrical ladder structure using the Bartletts' theorem [Sedr78]. A large number of different transformations and expansions of the canonical reflectances can be done to derive the final WDF. The most commonly used *Wave Digital Lattice Filters* (WDLF) (or lattice wave digital filters) are based on decomposing the reflectances into first and second order sections using circulators. A description of explicit formulas for synthesizing the WDLFs can be found in [Gazs85]. These WDFs always result in signal flow graphs with a minimum number of multiplications. In general, for a N:th order filter only N multiplications are required.

## 3.6. Pseudopassivity

One of the more important properties of the WDFs is their assumed passivity. A digital network cannot be defined as passive in a normal sense, therefore Fettweis has defined a concept using pseudopower [Fett72]. This is only possible because of the close relation between the WDFs and the classical circuit theory and the use of the scattering concept.

### 3.6.1. Power Concept

For an n-port network the instantaneous *pseudopower* can be defined as the power entering the network minus the power leaving it:

$$p(kT) = \sum_{j=1}^{n} [a_j^2(kT) - b_j^2(kT)] \frac{1}{R_j} \tag{3.28}$$

The steady-state pseudopower absorbed by the network can then be defined by:

$$P = \sum_{j=1}^{n} \left( |A_j|^2 - |B_j|^2 \right) \frac{1}{R_j} \tag{3.29}$$

If $P \geq 0$ for $\mathrm{Re}\{\Psi\} \geq 0$ then the network is *pseudopassive.*

If $P = 0$ for $\mathrm{Re}\{\Psi\} = 0$ then the network is *pseudolossless.*

If $P = 0$ for all $\Psi$ then the network is *nonenergic.*

Under linear conditions all adaptors, ideal transformers, gyrators, and circulators, can be shown to be nonenergic [Fett72]. This means that exactly all the energy that enters these elements is also reflected back. Capacitors, inductors, UEs, QUARLs are all pseudolossless under the same conditions. This can be interpreted as that these elements might only store a certain amount of the energy temporary. Using *incremental pseudopassivity* it has also been shown that a network consisting exclusively of pseudopassive elements is also itself pseudopassive [Meer80].

### 3.6.2. Constraints on the Design

To conserve the pseudopassivity also under finite arithmetic condition, certain constraints have to be introduced. It has been proven that no overflow correction is allowed before the multiplications in the different types of adaptors [Fett75a]. No quantization is normally allowed inside the

adaptors. A simple sufficient condition can then be defined, saying that quantization and overflow correction can only be made at the output of each block, particularly at the output of the adaptor. It should, however, be observed that this condition is sufficient but not necessary. Hence in some cases a different and less restrictive quantization scheme can be used.

The used quantization characteristic must ensure that no additional pseudoenergy is produced. Clearly for this reason rounding or two's complement truncation cannot be used. Sign-magnitude truncation on the other hand always results in a smaller or equal magnitude, figure 3.7a.

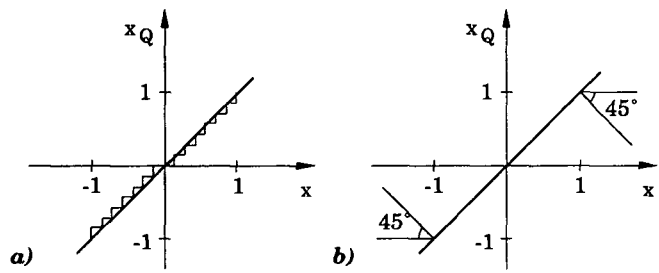



***Figure 3.7*** *a) Quantization characteristics*
*b) Overflow behavior*

The requirement on the overflow behavior corresponds to figure 3.7b, i.e., any continuous curve whose slope stays within ± 45° from saturation is sufficient [Fett85]. Again this can be shown using the incremental pseudopassivity [Meer80]. If two different inputs, where at least one is outside the maximal range of the signal, differ with the smallest possible amount, i.e., one quantization step $q = 2^{-wd}$. Then the corresponding outputs from the quantization element should differ with an amount less or equal to the quantization step $q$. It directly follows that the proposed characteristic is allowed. Normally, the saturation arithmetic is the simplest to implement.

In some WDFs both two's complement truncations and overflow wraps are allowed [Fett85]. In other filters is sufficient to use "energy consuming" quantizations only in some few locations. For the bireciprocal wave digital lattice filters two's- complement truncation is sufficient [Noss83].

From the pseudopassivity discussion it follows that the WDFs are guaranteed to be stable. All elements are pseudopassive or nonenergic even after the introduction of the nonlinearities. It is thus, impossible that any energy is generated. In fact WDFs are *guaranteed forced response stable.*

That is, the deviation from an ideal linear system should converge to zero after a certain time. They have also be shown to be stable even under the very severe *looped conditions* [Fett86].

### 3.6.3. Simplified Constraints

Since all filters in the proposed approach will be implemented using the state-space form, a simplified set of constraints can be used. Only signals outside adaptors might be accessible in the state-space form. A restriction that all the nonlinearities only occur outside the adaptors is therefore introduced.

- No quantization of the signals are allowed inside the adaptors, or inside any other element.
- The quantized signal should have a magnitude less or equal to the original signal.
- Overflow is not allowed inside the adaptors. The slope of the overflow characteristic should not differ more than 45° from saturation arithmetic.

These constraints are sufficient but not necessary, in fact they are more restrictive than the ones proposed earlier. The constraints that are used here have been chosen here for the simplicity and it will be shown that with this set of requirements it is also possible to transform the structures into an essentially numerical equivalent state-space form.

## 3.7. Different WDF families

The WDFs can be designed using various methods, which are more or less dependent on the chosen reference filter and on the implementation strategy. Some simple but important approaches are discussed here, complete presentations are found in [Wanh81, Fett86].

### 3.7.1. WDF using Constrained Adaptors

This is the first and perhaps the most straightforward type of WDFs. A direct translation of the elements of the ladder reference filter is made. Thereafter the elements are connected using constrained adaptors with one of the ports reflection-free.

Having synthesized the ladder reference filter, figure 3.8a, one can identify the connections and the elements, figure 3.8b, and finally by using

the freedom to select the implicit port resistances $R_8$ – $R_{13}$ a realizable network figure 3.8c is obtained. These resistances are calculated such that the port coefficient for the corresponding port of one of the adaptor connected to this port will be one. It is the use of these types of adaptors that makes it possible to fulfill the realizability requirement, i.e., absence of delay-free loops. One of the adaptors will always be unconstrained. By placing this as close as possible to the center of the structure a minimal computational path is obtained, and it results in a better natural scaling of the filter [Fett85].

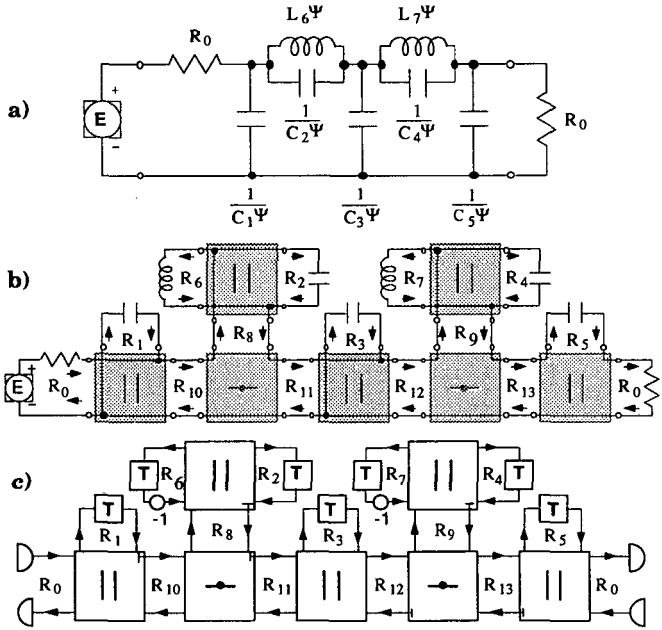



***Figure 3.8*** *Directly connected constrained adaptor structure derived from a ladder reference filter.*

A canonic WDF can always be found for each canonic reference filter using this approach by identifying capacitive or inductive loops or cutsets [Fett75b]. For example, it can easily be shown that the sum of all incident waves in a capacitive loop is zero. Due to this, one of the incident waves can be calculated implicitly from the others. It has been found that it reduces the noise and at the same time simplifies the stability constraints [Fett86].

There exist some variations of this method for realizing WDFs. The resonators in the example of figure 3.8.a, i.e., the two serial branches with an inductor and a capacitor in parallel, could be realized using two-port adaptors instead. This equivalent one port is obtained using unit element equivalents to the resonators [Wanh81]. However, care should be taken so that capacitive or inductive loops and cutsets still can be reduced such that a canonic realization is obtained.

Although this method to obtain WDFs are one of the simplest and most intuitive it has some drawbacks concerning the implementation. The computational paths are relatively long and a long sequence of operations must be performed each sample interval. Transforming this structure to the state-space form, as suggested in this thesis, will show that the obtained state-coefficients state-space are relatively long and that the matrix generally has no zero elements.

### 3.7.2. WDF with Inserted Unit Elements and Kuroda Identities

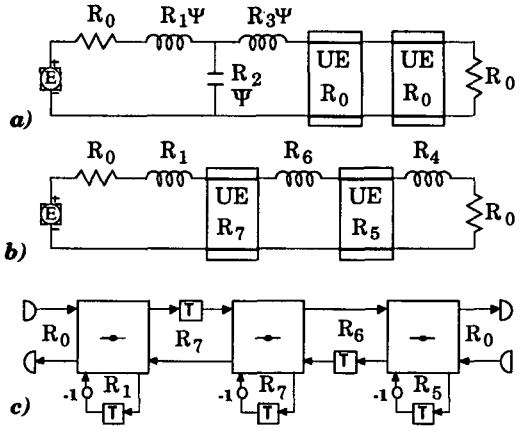



*Figure 3.9* *Simple example of reference filter*
*a) Original structure cascaded with UEs*
*b) Structure with inserted UEs*
*c) Corresponding WDF*

Unit elements can be used to avoid delay-free loops. The basic idea is to insert a number of UEs at one or both sides of the reference filter, figure 3.9a. Giving the UEs the same characteristic resistance as the terminators of the original filter leaves the transfer function unchanged. However, the

delay is increased. Using *Kuroda Identities* the UEs can be inserted in the filtering network itself by interchanging places with the LC elements [Wanh81]. The procedure is repeated until there is one UE in between each circuit element, figure 3.9b. Now a simple element by element transformation can be performed to obtain the WDF.

Normal unconstrained three-port adaptors can now be used when the UEs are transformed into delays. Equivalence transforms [Fett86] can be used to obtain full delays in either the upper or the lower branch in each intersection of figure 3.9c, or the UEs are simply replaced by QUARLs with $\Delta = T/2$. The main drawback of the method is that a non-canonical solution is always derived. On the other hand a large amount of parallelism is obtained. This can be utilized for the implementation.

Using state-space representation of the filters, as proposed in the presented methodology, these structures have some advantages. The number of state-variables increases but the state equations contain fewer terms since many coefficients are zero and the state-coefficients will often be simpler.

Clearly it would be better if the filtering capability of the UEs in figure 3.9 would be used. This could be obtained by direct synthesis of the filter in figure 3.9b.

### 3.7.3. WDF derived from Richards' Structures

Here synthesis and approximation theory from the *micro wave domain* are used [Bahe84]. A structure of cascaded UEs, figure 3.10a, can directly be used as reference filter for the WDFs. In this case only two-port adaptors and delays are needed for realizing the WDF.

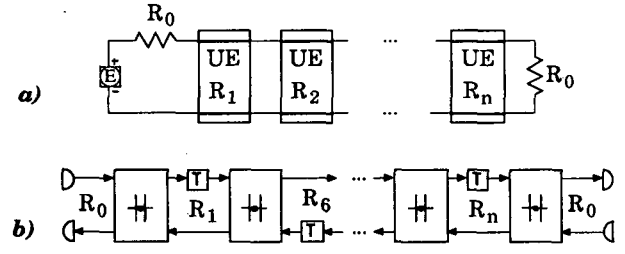



***Figure 3.10 a)*** *Richard reference filter*
***b)*** *Corresponding WDF if QUARLs are used*

Using this kind of reference filters some restrictions are imposed on the transfer function [Bahe84] H(Ψ). This can be inconvenient for some applications.

Likewise, with the class described above, the half delays are replaced by full delays using QUARLs instead of UEs. Here again, a large amount of inherent parallelism is the result. A state-space description will only contain state-equations with four terms, if the delays are placed alternately in the upper and in the lower branches as suggested in figure 3.10b.

### 3.7.4. WDF Derived from Lattice Reference Filters

This has shown to be one of the most interesting classes of WDFs. A canonic solution can always be found, and a very low complexity realization is obtained. The derived WDF structure is normally modular and regular. The name lattice comes from that the filters are derived from symmetrical lattice filters in figure 3.5 that are realized in the reference domain by two parallel reflectances, cf. figure 3.6.

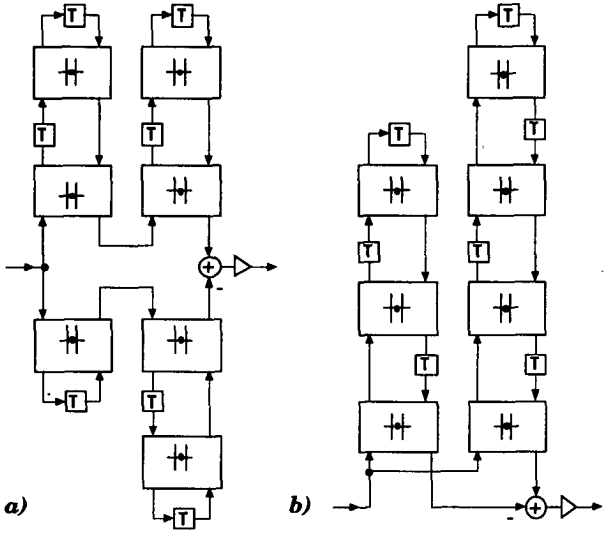



***Figure 3.11 a)*** *WDLF realized by cascaded resonators*
***b)*** *WDLF realized by Richards' structures*

These filters have never succeeded to gain a large interest in the analog domain, except for the case of crystal filters, because of the poor stop-band

sensitivity. However, in the digital domain the stopband sensitivity is not a serious problem. Any arbitrary stop-band specification can be fulfilled by choosing a correct set of coefficients.

To realize the reflectances in figure 3.6 a method using circulators can be used [Gazs85]. Any lossless reactance function can be realized by cascading first and second order one-ports (reactance functions) using this concept. The corresponding WDF will then only consist of cascaded first and second order all-pass sections preferably realized using only two-port adaptors, figure 3.11a. The parallelism obtained due to the two separate branches is of special interest for the implementation. Furthermore, if bireciprocal WDLFs are synthesized every second adaptor coefficient will be zero and a much simpler structure is obtained [Gazs85].

If instead the reactances are realized by higher order Richards' structures the realization in 3.11b is obtained. This structure would be even better suited for the methodology here. Similarity to the structure in figure 3.10b short state equations are obtained. However, the first structure can be pipelined by introducing delays between the cascaded sections.

### 3.7.5. Summary

Only some very few basic possibilities have been shown here. There exist many more possibilities regarding both reference filters and WDFs. For example, a given WDF can be altered by using adaptor transformations, i.e., a parallel adaptor can be transformed into a series and vice versa [Fett75c]. This can be used to scale the filter. WDF can also be scaled using the ideal transformer in Table 3.3. By using $n = 2^k$ where k is an integer the transformer can be inserted in any port.

The largest advantage of the WDFs is of course their guaranteed stability. The complete theory of the WDFs makes them very general and flexible. A specially tuned WDF can be found for any filter application. For a much more extensive presentation of the WDFs [Fett86] is recommended. An extensive reference list of related papers can also be found there.

WDF in state-space form, or *Wave Digital State-Space Filters* (WDSSF) are used in the presented approach. They can be derived by transforming any of the discussed WDF to a state-space form or by direct realization of an n-port generalized adaptor. Different proposed approaches for the synthesis and the realization of the WDSSF will be discussed in the next chapter.

# chapter 4. STATE-SPACE REPRESENTATION

*In the first part of this chapter a discussion around the state-space description of DSP algorithms is made. A generalized numerically equivalent state-space representation of fixed DSP algorithms is defined. Different strategies to obtain WDFs in state-space form are presented and some extensions are proposed for the design of Wave Digital State-Space Filters.*

## 4.1. The State-Space Representation

A uniform way of representing DSP algorithms is to use the *state-space representation* (SSR). It corresponds to a specific form of difference equation describing the DSP algorithm. The SSR describes the internal state of the algorithms as well as the input/output relationship. Due to the convenient mathematical form, it is also very useful to determine different additional properties of the DSP algorithm. The computational requirement is sometimes presented as the drawback of the SSR, the number of multiplications and additions are not minimal, but this can be turned into an advantage using innerproduct processors. Furthermore, only a minimum number of new signal values have to be computed each sample interval. These values can in principle be calculated concurrently, i.e., they are not dependent on each other. Maximally parallel realizations are always obtained using the state-space description.

Apart from that state-space filters can be designed directly using model-based approximation [Cand86], they can also be derived from other normal

filter structures. Any one-dimensional fixed DSP algorithm having positive delays, i.e., a causal algorithm, and which fulfills the fundamental *realizability conditions* [Fett76, Fett84] (the *computability conditions* [Croc75]) can be described in state-space form. This includes the constant linear periodically repetitive DSP algorithms as defined earlier. Other DSP algorithms not covered by the definitions above might as well be represented in the state-space form. This makes this representation very useful as a general description of DSP algorithms.

### 4.1.1. Representation of Filter Algorithms

Independently of its topology, a given fixed DSP algorithm can be seen as a network consisting exclusively of multipliers, adders, delay-elements, inputs and outputs, figure 4.1.

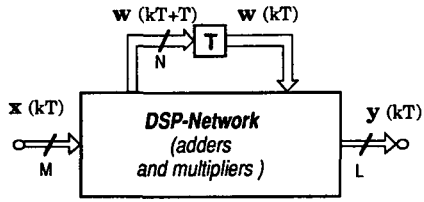



***Figure 4.1*** *State-Space view of a digital filter*

This system can then be described by *state-space equations* assuming that all matrices have real valued constant elements:

$$\mathbf{w}(kT + T) = \mathbf{A}\,\mathbf{w}(kT) + \mathbf{B}\,\mathbf{x}(kT)$$

$$\mathbf{y}(kT) = \mathbf{C}^T\,\mathbf{w}(kT) + \mathbf{D}\,\mathbf{x}(kT) \qquad (4.1)$$

Given that $N$ is the order of the system and given $M$ inputs and $L$ outputs, the vector $\mathbf{w}(kT)$ contains $N$ *state variables*, $\mathbf{x}(kT)$ $M$ *input variables* and $\mathbf{y}(kT)$ $L$ *output variables*.

- $\mathbf{A}$ is the $N \times N$ system matrix,
- $\mathbf{B}$ is the $M \times N$ *input matrix*,
- $\mathbf{C}^T$ is the transposed $L \times N$ *output matrix*
- $\mathbf{D}$ is a $M \times L$ matrix.

This definition could also be used for complex valued matrices or for matrices being dependent on the time index kT. Only constant fixed DSP algorithms are considered here, and they can all be described using a fixed state representation leading to real valued constant state-matrices.

The most common class of DSP algorithms are the single-input single-output filters. Here **B** and $\mathbf{C}^T$ become column respectively row vectors and **D** a simple scalar. The representation can be presented in form of a flow graph, figure 4.2.

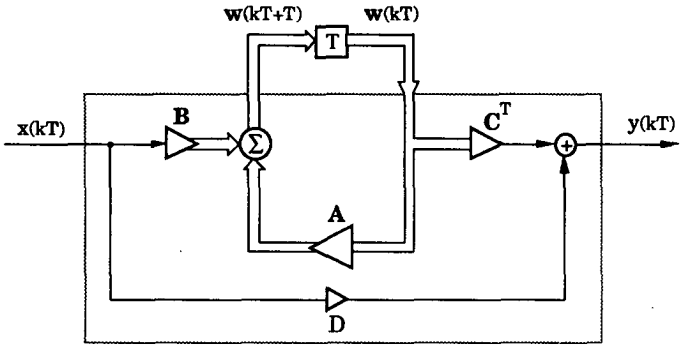



***Figure 4.2*** *State-Space flow graph*

In order to derive the *system function* or the *transfer function* from a state-space description one can calculate the impulse response, i.e., the response at the output y(kT) when the input is a *Dirac* function, $x(kT) = \delta(kT)$. Let all initial states $\mathbf{w}(0) = 0$. Clearly, $\mathbf{w}(T) = \mathbf{B}$, $\mathbf{w}(2T) = \mathbf{AB}$, $\mathbf{w}(3T) = \mathbf{A}^2\mathbf{B}$ and $\mathbf{w}(kT) = \mathbf{A}^{k-1}\mathbf{B}$. This leads to

$$h(kT) = D\delta(kT) + \mathbf{C}^T\mathbf{A}^{k-1}\mathbf{B}u(kT - T) \tag{4.2}$$

where u(kT) is the *unit step*. Taking the z-transform of (4.2) gives

$$H(z) = D + \sum_{k=1}^{\infty} \mathbf{C}^T\mathbf{A}^{k-1}\mathbf{B}z^{-k} = D + z^{-1}\mathbf{C}^T\Big[\sum_{k=0}^{\infty} \mathbf{A}^k z^{-k}\Big]\mathbf{B} \tag{4.3}$$

If all the eigenvalues of **A** have a magnitude less than $|z|$ the sum above converges to $(\mathbf{I} - z^{-1}\mathbf{A})^{-1}$ and the whole expression can be rewritten as

$$H(z) = D + z^{-1}\,\mathbf{C}^T(\mathbf{I} - z^{-1}\mathbf{A})^{-1}\,\mathbf{B} \tag{4.4}$$

where **I** is the diagonal *unity matrix*. Alternatively it can be expressed as

$$H(z) = D + \mathbf{C}^T(z\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{B} \tag{4.5}$$

The poles of H(z) correspond to the eigenvalues of **A**, [Jack89].

The derivation of the transfer function can be extended to a multi-input multi-output system where H(z) would be an *MxL transfer matrix* in the relation $\mathbf{Y}(z) = \mathbf{H}(z)\,\mathbf{X}(z)$.

The state equations can also be written in a more compact form as a single matrix ***M***.:

$$\begin{bmatrix} \mathbf{w}(kT+T) \\ \mathbf{y}(kT) \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} \mathbf{w}(kT) \\ \mathbf{x}(kT) \end{bmatrix} \tag{4.6}$$

The DSP algorithm described by this matrix can now fully be realized using *(N+L)* innerproducts and *N* delay-elements.

### 4.1.2. Similarity Transformations

The state-variables describing the system are not unique, i.e., another set of variables can be found giving the same external behavior. One way of obtaining this is to use so-called *similarity transformations*. If a non-singular *(NxN) transformation matrix* T is used, then a new set of state-variables is obtained by

$$\hat{\mathbf{w}}(kT) = \mathbf{T}\,\mathbf{w}(kT) \tag{4.7}$$

the corresponding state matrices are then obtained as

$$\hat{\mathbf{A}} = \mathbf{T}\,\mathbf{A}\,\mathbf{T}^{-1}, \qquad \hat{\mathbf{B}} = \mathbf{T}\,\mathbf{B}$$

$$\hat{\mathbf{C}}^{T} = \mathbf{C}^{T}\mathbf{T}^{-1}, \qquad \hat{\mathbf{D}} = \mathbf{D} \tag{4.8}$$

If **T** is a diagonal matrix a *scaling transformation* is obtained. This is useful to use for avoiding overflows. Another interesting special case is derived if **A** has distinct eigenvalues $\lambda_k$ it can be expressed as

$$\mathbf{A} = \mathbf{P}\,\Lambda\,\mathbf{P}^{-1} \tag{4.9}$$

where $\Lambda$ is diagonal matrix having the $\lambda_k$ as its diagonal elements and **P** is built up by the corresponding *eigenvectors* $\mathbf{v}_k$. By letting $\mathbf{T} = \mathbf{P}^{-1}$ the following decomposition can be obtained

$$\hat{\mathbf{A}} = \mathbf{T}\,\mathbf{A}\,\mathbf{T}^{-1} = \mathbf{P}^{-1}\mathbf{P}\,\Lambda\,\mathbf{P}^{-1}\mathbf{P} = \Lambda, \qquad \hat{\mathbf{B}} = \mathbf{P}^{-1}\mathbf{B}$$

$$\hat{\mathbf{C}}^{T} = \mathbf{C}^{T}\mathbf{P}\,, \qquad \hat{\mathbf{D}} = \mathbf{D} \tag{4.10}$$

This corresponds to a diagonalization of **A**. Unfortunately the $\lambda_k$ are often complex-valued that makes it impractical for the realization. However, it

has been shown that if they appear as complex conjugated pairs, as they always do if h(kT) is real-valued, a diagonal-block matrix can be obtained instead [Barn77]. In this so-called *coupled form* the blocks of either 1x1 or 2x2 are obtained. This form directly corresponds to the partial-fraction expansion of H(z) [Jack89].

Although the computational requirements can be decreased by similarity transformations the state-variables are changed and so also the numerical properties. On the other hand these transformations are of great interest for the analysis and the tuning of DSP algorithms.

### 4.1.3. State-Space Transformation

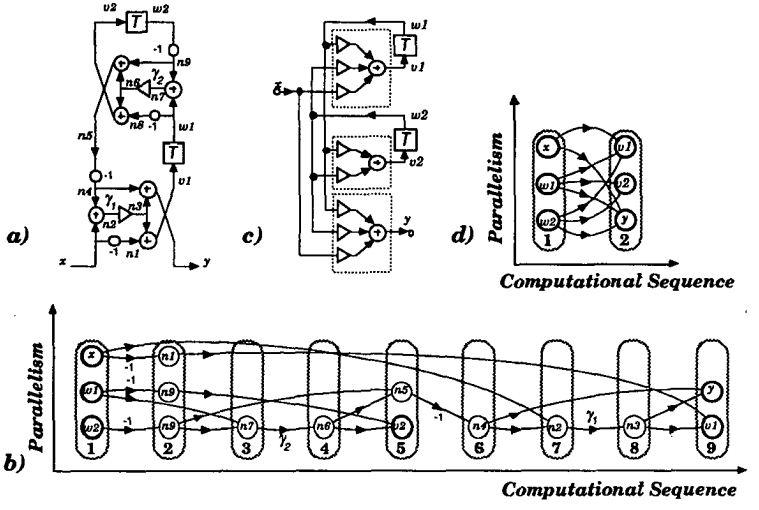



***Figure 4.3*** *State-space transformation of a simple filter.*
- ***a)*** *Original signal flowgraph*
- ***b)*** *Corresponding precedence graph*
- ***c)*** *State-space flowgraph*
- ***d)*** *Precedence graph for the state-space structure*

To derive a *state-space representation* (SSR) from a digital filter or fixed DSP algorithm, it is useful to start with the signal flow graph, for example figure 4.3a. By using a method proposed by Crochiere and Oppenheim [Croc75] a *precedence graph* for the SFG can be obtained, figure 4.3. From this description it is simple to define all equations that are required to

describe the system in a computable order. This representation is also useful to analyse the scheduling, the parallelism and the computational paths of a DSP algorithm. All tough it should be noted that better representations making use of *interperiod scheduling* exists [Wanh91], but this is of no direct concern here.

To derive a precedence graph all branches containing delays in the SFG are eliminated. Then all inputs, i.e., all nodes with only outgoing arcs or branches are identified and these form the first set of nodes, cf. figure 4.3b. The SFG is further reduced by eliminating these nodes and all outgoing branches from these. The new input nodes are identified and form the next group of nodes. This is repeated until the SFG has been completely eliminated. The arcs describing the relations between the nodes are now introduced to complete the precedence graph. A similar graph can also be extracted for the arithmetic operations from this precedence graph [Croc75]. This is sometimes very useful when implementing DSP algorithms using conventional arithmetic blocks such as adders and multipliers.

Clearly, the obtained precedence graph shows the amount of parallelism in each step of the sequence, i.e., how many variables that can be calculated concurrently in each step. In the vertical direction the sequence is shown and it is relatively easy to find the critical path for each output variable.

Transforming the filter into state-space form, figure 4.3c, a maximum of parallelism is obtained and the sequence is reduced. Using the innerproduct as a basic operation the length of the computational path will be minimal, figure 4.3d.

### 4.1.4. Features of the State-Space Description

A canonical form of the state-space description can always be found with a minimum number of state-variables. This can be verified by checking if $\boldsymbol{M}$ (4.6) is linearly independent. If not, one of the dependent variables can be eliminated by updating the columns of the other dependent variables in $\boldsymbol{M}$ and deleting the row and the column for the eliminated variable.

As long as the innerproduct is the basic operation a canonic state-space description will require a minimum number of quantization operations. Rounding or truncation needs only to be performed after completing the calculation of each innerproduct, or in a SFG description at the input of each delay element. The noise level will thus be essentially lower than in other filter structures, especially if minimal norm state-space filters are considered [Mull76a, Mull76b, Hwan77, Barn77].

The state-space structure includes a large number of multiplications (N+L)(N+M) but this is not a problem if vector multipliers are used for the implementation. In many practical cases there will be a number of zero state-space coefficients giving a sparse matrix. Shorter innerproducts are obtained and this reduces the computational requirements which lead to a simpler implementation. In some cases when a sparse matrix can be reordered to a block matrix it is also possible to improve the scheduling and the sequence of the computations.

Generally, all the innerproducts can be calculated concurrently and thus a maximum of parallelism is inherent to the structure. Another property of the state-space structure is the modularity. Generally for a N:th order single-input, single-output system, N+1 innerproducts each with N+1 terms, should be calculated.

Many basic properties can directly be obtained using state-space representation. This form is very convenient for analysis and tuning of DSP algorithms. Due to the mathematical formulation it is fairly simple to develop software for performing these tasks.

## 4.2. Wave Digital State-Space Filters

### 4.2.1. Design Strategies

Due to the passivity of the reference filter many of the desired properties of WDFs are obtained. This is made by a structural transformation to a digital network where passivity or pseudo-passivity is maintained due to the structural relation. This transformation is performed in two steps; first the elements are transformed and then the connections are replaced by adaptors. Some strict rules must be followed to obtain a realizable filter.

Mainly two methods exist to derive *Wave Digital State-Space Filters* (WDSSFs). First a WDSSF can be derived using the state-space transformation of a WDF. It is essential to follow certain rules to ensure that the derived SS-filter still remains a WDF with all numerical properties. The second possibility is to consider a reference filter as an n-port network and to describe it by a voltage-wave scattering matrix.

The first possibility is to use the numerically equivalent state-space transformation. Any WDF can be considered as a n-port where delays, inputs and outputs are connected to the ports. By relying on the quantized parameters from the original filter to derive the WDSSF the pseudo-passivity under finite wordlengths consideration are automatically fulfilled.

This is perhaps the most indirect method, but it has shown to workout fairly well and it is straightforward in the sense that only classical well-known methods can be used.

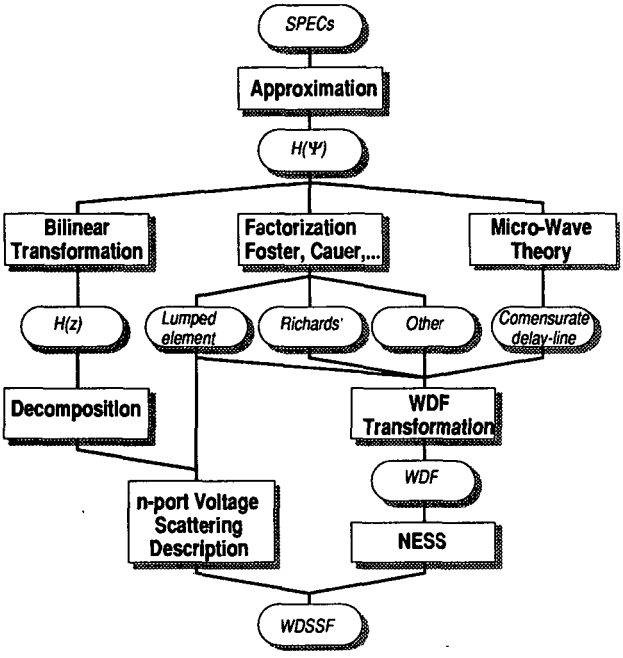



***Figure 4.4*** *Different strategies to derive wave digital state-space filters*

Instead of doing this it is possible to view the reference structure as a n-port network where two ports, the source and the load, are resistively terminated. This network can be described by means of a voltage wave scattering matrix. The elements can directly be translated into WDF equivalents but the interconnections will not be decomposed into individual parallel or serial connections, but realized by a general n-port SS-adaptor. This is only interesting if the constraints for the quantization of the discrete parameters of this adaptor can be found. This is equal to derive a scattering parameter description, i.e., the voltage wave scattering matrix describing the n-port.

A more direct method would correspond to use the bilinear transformation. H(z) is derived from H(Ψ), which should of course be

positive bounded and real such that it can be realized by passive or even lossless structures [Bahe84]. Using different results from scattering matrix and WDF theories H(z) can be decomposed into the normal WDF building blocks. A similar approach is used for the WDLF in the method proposed by Gazsi [Gazs85] but the drawback is that the method is limited to a specific class of structures. Trying to generalize this concept could lead to the direct synthesis of WDSSF. However, the problem is that by giving up the structural relation it is no more simple to maintain the simple proofs of pseudo-passivity. This could be made using a strict mathematical approach. It might also be difficult to retain the low sensitivity properties of the WDFs.

### 4.2.2. Constraints

If the state matrix defined in (4.1) is used, then the passivity requirement on the WDSSF can be expressed as that the matrix formed by

$$\mathbf{I} - \mathbf{R}^{1/2}\mathbf{A}^{T}\mathbf{R}^{-1}\mathbf{A}\,\mathbf{R}^{1/2} \tag{4.11}$$

is non-negative definite or even positive definite [Fett86]. This can be expressed in a simpler way:

$$\mathbf{R}^{-1/2}\mathbf{A}\,\mathbf{R}^{1/2} < \mathbf{I} \tag{4.12}$$

where $\mathbf{R}$ is the diagonal matrix formed by the port resistances in each of the internal ports, i.e., all ports except for the source and load. This leads to WDSSF for which all the stability properties are true except for possibly the looped condition stability. To conserve also this property, $\mathbf{A}$ in the expression above should be replaced by the complete voltage wave scattering matrix $\mathbf{S}$ for the n-port. Then the n-port is truly pseudo-passive.

For a lossless WDSSF (4.12) can be replaced by

$$\mathbf{R}^{1/2}\mathbf{S}^{T}\mathbf{R}^{-1}\mathbf{S}\,\mathbf{R}^{1/2} = \mathbf{I} \tag{4.13}$$

or

$$\mathbf{S}^{T}\mathbf{G}\,\mathbf{S} = \mathbf{G} \tag{4.14}$$

where $\mathbf{G} = \mathbf{R}^{-1}$, the diagonal matrix of all port conductances [Mart80].

### 4.2.3. Optimal WDSSFs

Independently on the strategy used to obtain the WDSSF the global goal is to obtain a filter with all the required properties of a WDF. It should also result in an as simple as possible implementation. The first goal is automatically achieved using some of the proposed approaches, while the

second objective is dependent on the used strategy. A simple or cheap implementation might require short state coefficients, if possible even some zero coefficients, few IPs and perhaps short IPs. Normally, it is not possible to obtain all these objectives in the same solution, a tradeoff must be made.

Similarity transformations (4.8) could be used to optimize some of the criteria but the requirements given by (4.12) or (4.14) must still be fulfilled. Choosing to use an appropriate reference filter, is the most important action for obtaining an optimal WDSSF.

## 4.3. Numerically Equivalent State-Space Transformation

Already in Chapter 2 it was proposed how to transform normal WDFs into state-space form. It was mentioned that certain rules have to be followed if the numerical properties should be conserved. This is of paramount importance when deriving a WDSSF this way. A concept of *numerically equivalent state-space* (NESS) representation is introduced.

### 4.3.1. Numerically Equivalent Representation

In general two different fixed DSP algorithms are equivalent if their system functions are the same. That is, if only the input-output relation is considered and they operate with infinite precision. When quantization of signals and parameters are introduced and the finite arithmetic aspects are considered, numerical equivalence must be introduced. Nonlinearities such as quantization and overflow are usually unavoidable. Then it is no longer sufficient to compare the I/O relation.

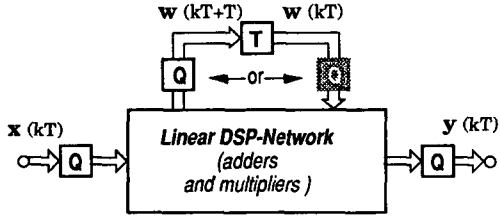



***Figure 4.5*** *State-Space view of a digital filter*

Two DSP algorithms represented by, for instance, two different SFGs, are numerically equivalent if and only if the state-variables are identical and the errors introduced by the nonlinearities result in the same excitation at the positions of the state-variables. If the same nonlinearities are used in both SFGs it is then necessary that they are performed at the same relative

locations (nodes). This is very difficult to fulfill for two SFGs with different topology. Often they do not contain the same nodes or internal variables.

Considering a canonic state-space transformed DSP algorithm, the number of distinctive nodes is reduced to a minimum. Only the nodes corresponding to the state-variables are accessible. Hence, quantization and other nonlinearities can only be introduced at these locations, figure 4.5.

If this structure should be numerically equivalent to the original structure, it exists two possibilities. The first, where all nonlinearities in the original structure are performed at the positions of the delay elements, at the inputs and at the outputs. The second possibility, where the nonlinear operations in the SS-transformed structure are altered so they give exactly the same contribution to the state variables as the nonlinearities in the original structure. This can be extremely complicated, or even not possible, in a practical case and this method have thus been rejected here.

### 4.3.2. Essential Numerically Equivalent Representation

A less strict requirement is obtained using the *essential numerically equivalent state-space representation*. Here the state-variables should be identical, possibly except for a change of sign and a linear scaling factor, and the nonlinear operations performed only at the positions of these variables. The difference from the equivalent form is that the original structure is allowed to contain nonlinearities at some additional positions.

The *essential* signifies that the structure is equivalent except for the nonlinearities. The state-space coefficients are derived from the quantized coefficient values in the original filter. Since quantization and overflow corrections are made only after the complete evaluation of the state equation the SS-structure will enhance some of the finite arithmetic properties, e.g., the round-off noise. However, it is not necessarily sure that this transformation conserves or improves all aspects of the numerical behavior for every fixed DSP algorithm. This should be carefully verified from case to case.

More important is that this concept is very useful for the wave digital filters. The nonlinearities cannot be performed inside the adaptors or any other element according to the simplified rules defined in Chapter 3. However it will always be possible to perform them at the locations of the delays, inputs and outputs. The SS-transformed structure remains a WDF with all what that implies. It will even be an improved WDF due to the minimum noise representation of the SS form.

### 4.3.3. Equivalent State-Variables

Identical state-variables are obtained, when the state-space matrix is calculated from the original structure according (4.1) and the state-space coefficients are exactly derived from the original truncated DSP coefficients. If $\alpha_{ij}$ are the state-space coefficients, $\gamma_k$ the original filter coefficients, $[\cdot]_Q$ the quantization operator and $F(\gamma_1, \gamma_2, ..., \gamma_M)$ a linear combination of the coefficients, then $\alpha_{ij}$ should be calculated according to (4.16) and not as in (4.15).

$$[\alpha_{ij}]_Q = [\, F(\gamma_1,\gamma_2,..., \gamma_M)\,]_Q \tag{4.15}$$

$$[\alpha_{ij}]_Q = F([\gamma_1]_Q, [\gamma_1]_Q,...,[\gamma_1]_Q) \tag{4.16}$$

Using this relation it is possible to find an equivalent set of quantized state-space coefficients to each original filter with quantized coefficients, and vice-versa. This guarantees that the state variables are exactly equivalent and that an essential numerically equivalent state-space representation is obtained.

### 4.3.4. Tradeoff Parallelism/Wordlength

Clearly, there is a strong correlation between the inherent parallelism and the required wordlength for the obtained equivalent set of state coefficients. If a variable is calculated as a long sequence of operations including many temporary variables then the $\gamma_k$ will appear as products of each other in (4.16) leading to a very long wordlength. Generally, the wordlength increases to the sum of the wordlength of all terms in the product when multiplying discrete fixed-point numbers. The situation would correspond to an algorithm with a long *computational path*.

If a higher amount of inherent parallelism would be present in the original DSP algorithm or shorter computational paths in the flowgraph, then the coefficients would appear single-handed or as sums of each other in the coefficients for the state-equation. Summing fixed-point binary numbers does not necessarily increase the wordlength, and if it does, it will be increased only by a few bits.

To shorten the computational paths between the state-variables, new variables can be introduced. This corresponds to pipelining of the DSP algorithms or to introduce delays in certain parts of the algorithm. Of course this cannot be done in any arbitrary place, e.g., introducing delays in any directed loops it would totally change the system function. Furthermore, it can only be performed if the delay through the system is allowed to increase.

These delays will correspond to new state-variables and the DSP algorithm will no more be canonical. Extra variables can also be used to get a more sparse state matrix, in fact one will often obtain a block state matrix.

Another possibility is to use so called *auxiliary variables*. This corresponds to explicit calculation of intermediate nodes. These have not to be stored and reused in the subsequent sampling interval but will on the other hand reduce the parallelism since all state equations cannot be calculated in parallel any more. The use of auxiliary variables reduces the parallelism, everything cannot be calculated in a single phase anymore. On the other hand in certain architectural configurations this is not too important and the resulting innerproducts to calculate have fewer terms and simpler coefficients.

This leads to the conclusion that is feasible to use DSP algorithms with a large amount of native parallelism if the numerically equivalent state-space transformation is to be used. The consequence of this is that WDLFs usually are more interesting than other WDFs, since they contain to parallel branches giving a sparse matrix and due to that they can easily be pipelined by introducing extra state variables in both branches. Ladder based WDFs designed using inserted UEs are also interesting. Inserting UEs may in some cases directly correspond to pipelining.

The search for a globally optimal solution should also concern the amount of inherent parallelism. It is no more sure that a better or more efficient solution is achieved using the canonic and minimal original DSP algorithm. Instead non-canonical or pipelined structures might be more optimal.

### 4.3.5. WDSSF Obtained by Using NESS Transformation

The first method proposed here for obtaining WDF in state-space form is based on the numerical equivalent state-space form described above. The starting point is a WDF derived using any of the methods described in Chapter 3. It is not worthwhile spending too much effort on tuning and transforming this WDF in the original form when most of the obtained solutions will lead to exactly the same state-space description. Moreover, many parameters are more easy to tune directly for the WDSSF. It is more important to select a WDF that contains as much parallelism as possible and with as short computational paths as possible.

Using a SFG description of the WDF, a so called *Wave Flow Graph* (WFG), a set of difference equations can be extracted. The state-space equations can be obtained by reducing the system equations until they only

contain the state-variables. This is an involved work especially for a WDF derived from a ladder structure. Some software tools using graph algorithms have been developed to support this transformation [Sjös89a, Anso91a].

The state-coefficients $\alpha_{ij}$ of the resulting WDSSF are expressed explicitly as linear combinations of the original WDF adaptor coefficients $\gamma_k$. This is absolutely necessary if they should be quantized and optimized. The objective of the coefficient optimization will now be to find a set of quantized $\gamma_k$ such that the $\alpha_{ij}$ are as short as possible and such that the requirements given by the specifications still are fulfilled. The only constraint on the coefficients is that $0 \leq \gamma_k \leq 2$, for an n-port parallel or series adaptor where $n > 2$. This prevents port-resistances or port-conductances from becoming negative. A slightly different constraint is valid in case of two-port adaptors, according to Chapter 3.

The low sensitivity property of the WDFs together with the fact that they contain relatively few adaptor coefficients simplifies the optimization process considerably. An algorithm and a tool [Sjös89a] to accomplish this optimization is presented in Chapter 7.

Once a feasible quantized WDSSF has been obtained it can be scaled by using $\hat{\mathbf{S}} = \mathbf{T}\ \mathbf{S}\ \mathbf{T}^{-1}$, where $\mathbf{T}$ is a diagonal scaling matrix and $\mathbf{S}$ is the complete state matrix. The elements in this matrix are necessarily simple powers of two otherwise $\mathbf{T}^{-1}$ would not be possible to represent in binary form and the resulting $\hat{\mathbf{S}}$ would not be essential numerically equivalent to $\mathbf{S}$.

### 4.3.6. PCM Filter Example

To demonstrate the method a fifth order PCM filter according to CCITT specification G712 PCM is considered. This filter has been used as a benchmark to compare different implementation strategies [Clae88, Yazd90a]. The filter is realized by a WDLF obtained using FALCON [Gazs86].

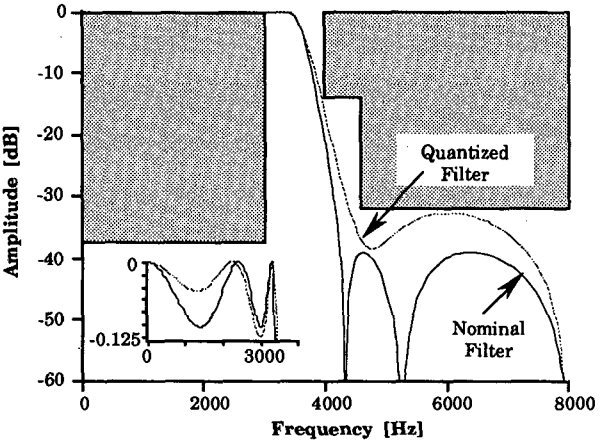



*Figure 4.6 PCM filter requirements with plotted magnitude responses*

A SFG for the filter is shown in figure 4.7. The filter consists of two parallel branches and only contains five adaptor coefficients.

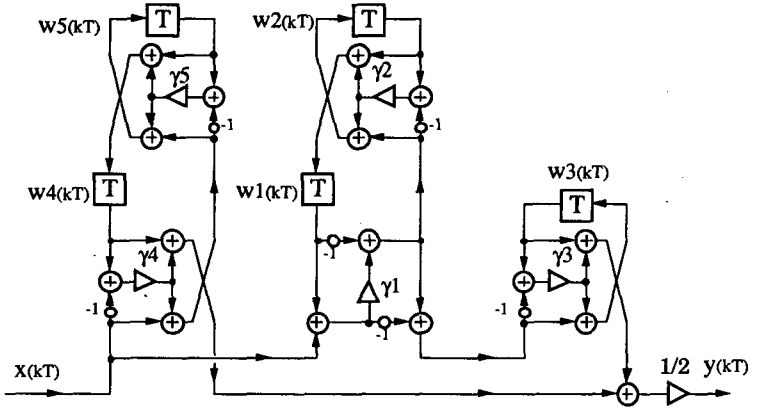



*Figure 4.7 SFG of the fifth order WDLF obtained by FALCON*

By using, for instance SFGkernel [Sjös89] the following state equations are obtained:

$$
\begin{aligned}
w_1(kT+T) &= w_1(kT)\,\gamma_2(1-\gamma_1) + w_2(kT)(1+\gamma_2) + x(kT)\,\gamma_1\gamma_2 \\
w_2(kT+T) &= w_1(kT)\,(1-\gamma_2)(\gamma_1-1) + w_2(kT)\,\gamma_2 + x(kT)\,\gamma_1(\gamma_2-1) \\
w_3(kT+T) &= w_1(kT)\,(\gamma_1-2)(1-\gamma_3) + w_3(kT)\,\gamma_3 + x(kT)\,(1-\gamma_1)(1-\gamma_3) \\
w_4(kT+T) &= w_4(kT)\,\gamma_4\gamma_5 + w_5(kT)\,(1+\gamma_5) + x(kT)\,(1-\gamma_4)\gamma_5 \\
w_5(kT+T) &= w_4(kT)\,(\gamma_5-1)\gamma_4 + w_5(kT)\,\gamma_5 + x(kT)\,(\gamma_4-1)(1-\gamma_5) \\
y(kT) &= w_1(kT)\,\tfrac{1}{2}\gamma_3(2-\gamma_1) + w_3(kT)\,\tfrac{1}{2}(1+\gamma_3) - w_4(kT)\,\tfrac{1}{2}(\gamma_4+1) \\
&\quad + x(kT)\,\tfrac{1}{2}(\gamma_3(\gamma_1-1)+\gamma_4)
\end{aligned}
\tag{4.17}
$$

At maximum the adaptor coefficients appear in products of two for this specific example. After discrete optimization of the coefficients the following values were found.

$$\gamma_1 = \frac{3}{16} \quad \gamma_2 = \frac{5}{32} \quad \gamma_3 = \frac{1}{4} \quad \gamma_4 = \frac{3}{8} \quad \gamma_5 = \frac{5}{16}$$

Using a scaling matrix with the diagonal elements $t_1 = t_2 = t_4 = t_6 = 1$, $t_3 = 2$ and $t_5 = 1/2$ and dividing the output of the filter with 2 the complete WDSSF is described by:

$$
\begin{pmatrix} w_1(kT+T) \\ w_2(kT+T) \\ w_3(kT+T) \\ w_4(kT+T) \\ w_5(kT+T) \\ y(kT) \end{pmatrix}
= \frac{1}{2^9}
\begin{pmatrix}
65 & 592 & 0 & 0 & 0 & 15 \\
-351 & 80 & 0 & 0 & 0 & -81 \\
-348 & 0 & 128 & 0 & 0 & 156 \\
0 & 0 & 0 & 60 & 336 & 100 \\
0 & 0 & 0 & -264 & 160 & -440 \\
58 & 0 & 320 & -176 & 0 & 22
\end{pmatrix}
\begin{pmatrix} w_1(kT) \\ w_2(kT) \\ w_3(kT) \\ w_4(kT) \\ w_5(kT) \\ x(kT) \end{pmatrix}
\tag{4.18}
$$

A maximal wordlength of 11 bit, 10 bit + sign, is required for any of the state coefficients. Evidently, the matrix contains many zeros. This is due to the two separate branches of the original SFG in figure 4.7. It can even be recognized that the recursive part of (4.18), i.e., the **A** matrix (4.1), consists of two separate blocks corresponding to the filter order of the upper and lower branch, one 3x3 and one 2x2.

## 4.4. Voltage Wave Scattering Parameter Derivation

### 4.4.1. Scattering Parameters of Classical Circuits

The state matrix representation of a WDF is closely related to the *scattering matrix* from the classical circuit theory [Bahe84, Bele68]. The

scattering matrix has been shown to exist for every classical network in conjunction to impedance or admittance matrices. This has given the *scattering parameters* an important role in classical circuit theory. The scattering parameters can be defined by using *wave quantities* as signal carriers at each port of a network. Here, voltage waves are used rather than the classical power waves and this results in a slightly different scattering matrix. Fortunately, the only difference is a linear scaling of each parameter as will be demonstrated later on.

Using an equivalent definition of the incident and reflecting voltage wave as was used for the WDF's, one gets

$$\begin{aligned} \mathbf{a} &= \mathbf{v} + \mathbf{R}\,\mathbf{i} \\ \mathbf{b} &= \mathbf{v} - \mathbf{R}\,\mathbf{i} \end{aligned} \tag{4.19}$$

where $\mathbf{v}$ and $\mathbf{i}$ are the vectors of voltage and currents of the ports and $\mathbf{R}$ is a diagonal matrix consisting of the port-resistances. If $\mathbf{a}$ and $\mathbf{b}$ are the vectors of the incident and the reflected waves the scattering matrix $\mathbf{S}$ is defined by:

$$\mathbf{b} = \mathbf{S}\,\mathbf{a} \tag{4.20}$$

where the separate elements of $\mathbf{S}$ are the so called *scattering parameters*.

To give a simple physical interpretation of the scattering matrix and parameters a two-port is first considered. Writing out (4.20) explicitly for the two-port gives:

$$\begin{aligned} b_1 &= S_{11}\ a_1 + S_{12}\ a_2 \\ b_2 &= S_{21}\ a_1 + S_{22}\ a_2 \end{aligned} \tag{4.21}$$

The scattering parameters can now be defined as

$$\begin{aligned} S_{11} &= \left.\frac{b_1}{a_1}\right|_{a_2=0}\ , \quad S_{12} = \left.\frac{b_1}{a_2}\right|_{a_1=0} \\ S_{21} &= \left.\frac{b_2}{a_1}\right|_{a_2=0}\ , \quad S_{22} = \left.\frac{b_2}{a_2}\right|_{a_1=0} \end{aligned} \tag{4.22}$$

Each parameter is calculated as the ratio of a reflected signal to an incident signal under the condition of zero incident signal at the other port. That the incident signal at the other port is zero, $a_i = 0$, simply corresponds to $v_i = -R_i\,i_i$. This, on the other hand, corresponds to the situation where port i is terminated in its port reference resistor $R_i$.

The scattering parameter $S_{11}$ can now be interpreted as the *input reflection coefficient* of the one-port network formed by terminating the second port by its reference resistor. Similarly $S_{22}$ is the *output reflection coefficient*. The *forward transmission coefficient* $S_{21}$ describes the transfer function of the network. Taking the squared absolute value of this coefficient, $|S_{21}|^2$, gives the network's *voltage gain*. Similarly $S_{12}$ equals to the *reverse transmission coefficient* and $|S_{12}|^2$ to the *transducer voltage gain*.

All the above discussion can of course be extended to a n-port network. Instead of a 2x2 scattering matrix a nxn matrix is obtained. Furthermore the definition of the scattering parameters has to be extended:

$$S_{ij} = \left.\frac{b_i}{a_j}\right|_{a_k=0\ \forall\ k\neq j} \qquad \text{when } i \neq j \tag{4.23}$$

The scattering parameter $S_{ij}$ is thus defined as the ratio between the reflected wave in port i and the incident wave at port j when all other incident waves are zero. This means that all other ports are terminated with their port resistances. Similarly the diagonal elements in the scattering matrix $S_{ii}$, or the reflectances are described by:

$$S_{ii} = \left.\frac{b_i}{a_i}\right|_{a_k=0\ \forall\ k\neq i} \tag{4.24}$$

Basically, $S_{ii}$ corresponds to reflectance of the one port formed by terminating all other ports than port i with their port resistances, while $S_{ij}$ would correspond to the two-port formed by terminating all other ports than i and j with their port resistances. These two last results clearly show that there is no difference in the definition of scattering matrix for a 2-port or a *n*-port network.

### 4.4.2. Relation between Power and Voltage Wave Scattering Parameters

It should be noted that the classical scattering parameter theory makes use of power waves rather than voltage waves giving a slightly different meaning of the parameters. The power waves are defined as

$$a_i = \frac{v_i}{\sqrt{R_i}} + \sqrt{R_i}\, i_i \quad b_i = \frac{v_i}{\sqrt{R_i}} - \sqrt{R_i}\, i_i \tag{4.25}$$

Fortunately this difference will only appear as a scaling factor for the scattering parameters. If the power wave scattering matrix is denoted $\hat{\mathbf{S}}$ the following relation between $\mathbf{S}$ and the power wave scattering matrix exists:

$$\mathbf{S} = \mathbf{R}^{1/2}\,\hat{\mathbf{S}}\,\mathbf{R}^{-1/2} \tag{4.26}$$

For a simple two-port this gives

$$S_{11} = \hat{S}_{11}\,,\; S_{22} = \hat{S}_{22}\,,\; S_{12} = \hat{S}_{12}\sqrt{\frac{R_1}{R_2}}\,,\text{ and } S_{21} = \hat{S}_{21}\sqrt{\frac{R_2}{R_1}} \tag{4.27}$$

Clearly the scattering parameters only differ by a linear scaling factor and the results from the classical scattering parameter theory can be used directly in most cases. The diagonal elements are equivalent in both descriptions. Clearly, (4.26) shows that this is true for any dimension of **S**.

### 4.4.3. Multi-Port

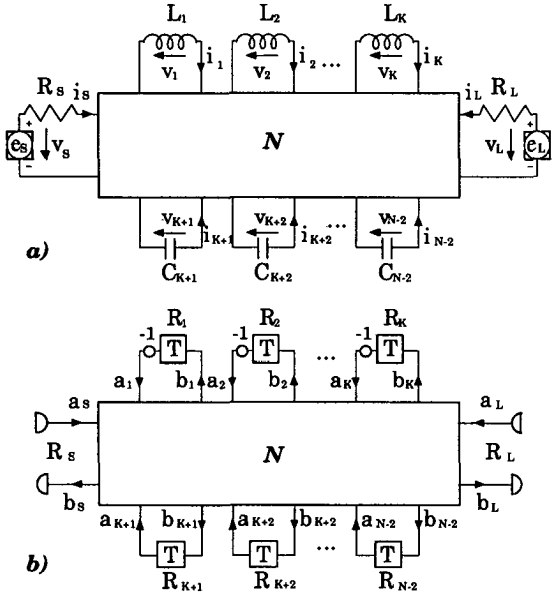



***Figure 4.8*** *a) Multi-port representation of reciprocal lumped element network **N**.*
*b) Corresponding digital network.*

Using the basic concepts of the scattering parameter theory discussed in the last sections, it is possible to view any lumped element network as a *multi-port N* to which the elements are connected. The multi-port *N* represents the connection topology. Here it is only interesting to consider

lossless networks. Using the voltage wave scattering relation established in (4.19) and (4.20) clearly $N$ in figure 4.8b can be described by the scattering matrix, and if $N$ is lossless it also implies that $\mathbf{S}$ is lossless. In fact, this concept is directly used for the adaptors proposed by Fettweis [Fett75c] but only n-ports containing exclusively serial or parallel connections are considered there.

To derive the scattering parameters different methods can be used. The simplest or the most straightforward is to use Kirchhoff's laws together with the wave definitions.

Due to the losslessness of $N$ only steady-state relations are necessary, but to avoid confusions between some symbols later in this chapter lowercase letters are used for the signal quantities here.

### 4.4.4. Calculation of the Scattering Parameters

The next problem is to calculate the scattering parameters for a given network. Due to (4.19) and (4.24) we have:

$$S_{ii} = \frac{v_i - R_i\, i_i}{v_i + R_i\, i_i} = \frac{\frac{v_i}{i_i} - R_i}{\frac{v_i}{i_i} + R_i} = \frac{Z_i - R_i}{Z_i + R_i} \qquad \forall\, a_k = 0,\ k \neq i \tag{4.28}$$

where $Z_i$ is the input impedance at port i when all other ports are terminated. Furthermore, combining (4.19) and (4.23) gives:

$$S_{ij} = \frac{-2R_i\, i_i}{v_j + R_j\, i_j} = \frac{i_i}{i_j}\,\frac{-2R_i}{Z_j + R_j} \qquad \forall\, a_k = 0,\ k \neq j \tag{4.29}$$

The remaining problem is now to calculate the current ratio $i_i / i_j$. This has been found to be equal to the following expression:

$$\frac{i_i}{i_j} = -\frac{Z_j}{Z_{ji}} \tag{4.30}$$

where $Z_{ji}$ corresponds to the transfer impedance formed by

$$\frac{v_j}{i_i}\ \forall\, i_j = 0 \tag{4.31}$$

Now (4.29) can be rewritten as:

$$S_{ij} = \frac{-2 Z_j R_i}{Z_{ji} (Z_j + R_j)} \qquad \forall a_k = 0, k \neq j \tag{4.32}$$

The above expressions for the scattering parameters can now easily be calculated for a given lumped element network. It is simple to calculate the different impedances automatically and thus also the parameters themselves.

### 4.4.5. Relation to the State-Space Matrix

If classical doubly terminated lumped element filters are considered, the state-space matrix is similar to the scattering matrix. The state variables $w_i$ are defined at each input to a delay element. Using the capacitor and the inductor as the only frequency dependent elements it can be verified that the new value of the state variable can be calculated as $w_i(kT) = b_i(kT)$ and that the incident waves equal the old values, $a_i(kT) = \pm w_i(kT-T)$, positive in case of a capacitor and negative in case of an inductor, figure 4.8b.

The scattering matrix for the n-port contains the complete description for both the source and the load. Thus, both a complementary input and output are defined. This is usually not the case for the state matrix. For a N:th order ladder filter a $(N+2)^x(N+2)$ scattering matrix is obtained while the required state matrix is of the dimension $(N+1)^x(N+1)$. The column corresponding to the reflecting wave in the generator port should normally be deleted together with the row corresponding to the incident wave at the output port, if the complementary signal is not used.

If a non-canonic reference filter is used, e.g., a ladder filter containing second order reactance functions in either serial or shunt ports such that capacitive loops or inductive cutsets are formed, some of the state variables will be linearly dependent. In the discussion about WDFs it was mentioned how these dependent variables could be eliminated in a wave flow graph, this is also possible in the state-space matrix. For example if variable $j$ is dependent on variables $k$ and $l$, then row and column $j$ are deleted and column $k$ and $l$ updated according to the dependency.

### 4.4.6. Constraints

One problem using a multi-port voltage-wave scattering description is to find the rules to apply when quantizing the corresponding state coefficients. Using the concepts of pseudo-passivity or pseudo-losslessness directly only leads to relations containing squares and ratios of scattering parameters and port resistances, as for example (4.33).

$$\sum_{j=1}^{n} S_{ij}^2 \frac{R_i}{R_j} = 1 \quad \text{for } i = 1..n \tag{4.33}$$

These are not practical or useful when trying to find discrete quantized parameters. In fact, the obtained constraints are similar to what one would find using power waves. Fortunately, Martens and Meerkötter [Mart76] have found a decomposition of the voltage wave scattering matrix, making it possible to find simpler and more useful constraints [Mart80].

## 4.5. Decomposed Voltage Wave Scattering n-Port

### 4.5.1. Basic Concepts

Using the multi-port concept described in figure 4.8 specifically for ladder networks, some useful decompositions of the scattering matrix can be derived, [Mart76]. Any well-formed lumped element ladder structure can be divided into *l link ports* corresponding to series ports in the ladder structure, and *t tree ports* related to the shunt ports, where $l + t = n$. Then the relations between currents and link currents respectively voltages and tree voltages can be expressed as

$$\mathbf{i} = \mathbf{B}^T \mathbf{i}_l, \quad \mathbf{v} = \mathbf{Q}^T \mathbf{v}_t \tag{4.34}$$

where $\mathbf{B}^T$ is the *circuit matrix* and $\mathbf{Q}^T$ the *cutset matrix*. These matrices describe Kirchhoff's laws in a convenient way. They only contain 1, -1 and 0, or if an extension is made to include ideal transformers in the ports they can also contain the transformer-ratio values $n_k$. It can also be shown that these matrices are orthogonal leading to $\mathbf{Q}\ \mathbf{B}^T = 0$, which is equal to the losslessness of the topology of the n-port [Mart80]. Applying these matrices to the definition of the voltage waves of (4.19) yields

$$\mathbf{a} = \mathbf{Q}^T \mathbf{v}_t + \mathbf{R}\,\mathbf{B}^T \mathbf{i}_l, \quad \mathbf{b} = \mathbf{Q}^T \mathbf{v}_t - \mathbf{R}\,\mathbf{B}^T \mathbf{i}_l \tag{4.35}$$

where $\mathbf{R}$ is the diagonal matrix formed by the port resistances $R_k$. (Observe that lowercase letters are used to denote incident and reflecting waves here in order not to confuse them with the system and circuit matrices.) Using $\mathbf{G} = \mathbf{R}^{-1}$ and the orthogonality of $\mathbf{Q}$ and $\mathbf{B}$ gives

$$\mathbf{Q}\,\mathbf{G}\,\mathbf{a} = \mathbf{Q}\,\mathbf{G}\,\mathbf{Q}^T \mathbf{v}_t = \mathbf{Y}\mathbf{v}_t \tag{4.36}$$

and

$$\mathbf{v}_t = \mathbf{Y}^1\,\mathbf{Q}\,\mathbf{G}\,\mathbf{a} \tag{4.37}$$

Now the relation between $\mathbf{a}$ and $\mathbf{b}$ can be established using

$$\mathbf{b} + \mathbf{a} = 2\,\mathbf{v} = 2\,\mathbf{Q}^T \mathbf{v}_t = 2\,\mathbf{Q}^T \mathbf{Y}^{-1}\,\mathbf{Q}\,\mathbf{G}\,\mathbf{a} \tag{4.38}$$

leading to

$$\mathbf{b} = (2\,\mathbf{Q}^T \mathbf{Y}^{-1}\,\mathbf{Q}\,\mathbf{G} - \mathbf{I})\,\mathbf{a} \tag{4.39}$$

where $\mathbf{I}$ denotes the unity matrix. Clearly relation (4.39) is equal to the voltage wave scattering matrix $\mathbf{S}$.

### 4.5.2. Decompositions

By observing that

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_l \\ \mathbf{B}_t \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_l \\ \mathbf{Q}_t \end{bmatrix} \tag{4.40}$$

where $\mathbf{B}_l$ and $\mathbf{Q}_t$ are unity matrices and by partitioning $\mathbf{G}$ as

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_l & 0 \\ 0 & \mathbf{G}_t \end{bmatrix} \tag{4.41}$$

one obtains

$$\begin{bmatrix} \mathbf{b}_l \\ \mathbf{b}_t \end{bmatrix} = \begin{bmatrix} 2\,\mathbf{Q}_l{}^T\mathbf{K} - \mathbf{I} & 2\,\mathbf{Q}_l{}^T(\mathbf{I} - \mathbf{K}\,\mathbf{Q}_l{}^T) \\ 2\mathbf{K} & \mathbf{I} - 2\,\mathbf{K}\,\mathbf{Q}_l{}^T \end{bmatrix} \begin{bmatrix} \mathbf{a}_l \\ \mathbf{a}_t \end{bmatrix} \tag{4.42}$$

where

$$\mathbf{K} = \mathbf{Y}^{-1}\,\mathbf{Q}_l\,\mathbf{G}_l = (\mathbf{G}_t + \mathbf{Q}_l\,\mathbf{G}_t\,\mathbf{Q}_l{}^T)\,\mathbf{Q}_l\,\mathbf{G}_t \tag{4.43}$$

Different decompositions of $\mathbf{S}$ can now be obtained

$$\mathbf{S} = \begin{bmatrix} -\mathbf{I} & \mathbf{Q}_l{}^T \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} -\mathbf{I} & 0 \\ -2\,\mathbf{K} & \mathbf{I} \end{bmatrix} \begin{bmatrix} -\mathbf{I} & \mathbf{Q}_l{}^T \\ 0 & \mathbf{I} \end{bmatrix} \tag{4.44}$$

All tree matrices of (4.43) are self inverse and thus also

$$\mathbf{S} = \mathbf{S}^{-1} \tag{4.45}$$

Using the definition of $\mathbf{K}$ it can be shown that $\mathbf{S}^T\,\mathbf{G} = \mathbf{G}\,\mathbf{S}$ and (4.44) gives

$$\mathbf{S}^T\,\mathbf{G}\,\mathbf{S} = \mathbf{G} \tag{4.46}$$

This corresponds to the losslessness and reciprocity of the reference network topology. This was also described by (4.13) and (4.14).

Other mathematically interesting decompositions that are expressed by the negative and positive eigen-values and the corresponding eigenvectors can be found in [Mart76, Mart78, Mart80]. In (4.47) the first matrix corresponds to the eigenvectors.

$$\mathbf{S} = \begin{bmatrix} \mathbf{I} - \mathbf{Q_l}^T\mathbf{K} & \mathbf{Q_l}^T \\ -\mathbf{K} & \mathbf{I} \end{bmatrix} \begin{bmatrix} -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{Q_l}^T \\ \mathbf{K} & \mathbf{I} - \mathbf{K}\mathbf{Q_l}^T \end{bmatrix} \tag{4.47}$$

### 4.5.3. Quantization Algorithm

Clearly, the relation of (4.46) should hold also when the elements of **S** are quantized to binary values. By using the results of this method the problem to quantize **S** is reduced to quantize **K** thus $t \times t$ parameters instead of $k \times l$. **K** cannot be truncated randomly since the elements can be linearly dependent. In fact the $t \times t$ elements of **K** are dependent on $l$ parameters, namely the port resistances, and for the truncated **K** it is necessary to find a corresponding set of rational valued **R** or **G**.

As a comparison, the adaptors defined by Fettweis have the property that they all can be described by $n-1$ parameters, i.e., the adaptor coefficients $\gamma$. Because these are fewer than the $n$ port-resistances, they can be quantized within the range $0 < \gamma < 2$.

Another problem with quantization of **K** is that the **G** cannot be chosen arbitrarily. As shown in (4.43) to calculate **K** from **G** includes a matrix inversion of **Y**. A method to cope with the truncation of **K** has been proposed by Martens [Mart80].

- To start, the port conductances of $\mathbf{G_l}$ are quantized to fixed point binary numbers. This can be made using discrete optimization. A matrix $\mathbf{C_l} = [\ \mathbf{Y}\ |\mathbf{Q_l}\ \mathbf{G_l}\ ]$ is formed. The diagonal elements of **Y** will contain the unknown $\mathbf{G_t}$ elements.

- The following step is repeated $t$ times, starting with $i = 1$. Truncate $1 / C_{ii}$ to a binary number, this gives $G_{ti}$. Multiply row i by the obtained $1 / C_{ii}$. Use elementary row operations to zero all elements in column i except for $C_{ii}$. This gives the new matrix $C^{i+1}$. By repeating these operations until $C^{t+1}$ is obtained all $\mathbf{G_t}$ elements are known.

- The resulting matrix is $\mathbf{C}^{t+1} = [\ \mathbf{I_t}\ |\ \mathbf{K}\ ]$ where **K** will only contain binary elements.

Unfortunately, this method has shown to result in rather long coefficient wordlengths in some cases. It is also hard to find a global optimum. Each step has to be reevaluated each time a value for $\mathbf{G_l}$ is changed and the optimization procedure would be slow. It would be preferable to find an orthogonal set of parameters to quantize. These parameters can all be extracted from **K** since the full system can be described only in terms of **K**

and $\mathbf{Q}_1$ where $\mathbf{Q}_1$ is a unity matrix. A result of this is that the square of the minimum number of tree ports $t^2 \geq n - 1$, i.e., the minimum number of orthogonal parameters describing the n-port. Otherwise, $\mathbf{K}$ could not fully define the n-port.

## 4.6. Conclusions

The state-space representation of DSP algorithms offers many advantages. It is a general and compact form of representation where all the inherent parallelism can be utilized for the implementation. For the proposed methodology where the implementation is based on a innerproduct processor the state-space form is especially suitable.

The proposed method using the numerically equivalent state-space transformation has shown to be highly flexible. The method can be employed for any realizable digital filter algorithm. Multiple input and output filter algorithms can as well be used. A straightforward strategy is used where the complete filter algorithm is completely developed first and afterwards this algorithm is transformed into a state-space form. This is followed by a constrained optimization of the coefficients.

The NESS form combined with the choice of WDF leads to WDSSF that have optimal behavior under finite arithmetic considerations. The optimization of the coefficients might sometimes be unnecessary restrictive but this is the cost that has to be paid to keep the method simple and straight forward.

Various possibilities to extend the method exist. The filter algorithm can be decomposed into substructures each transformed into NESS form. Auxiliary variables can be introduce to improve coefficient wordlengths and the computational requirements. A number of transformations can be carried out on the state matrix. However, it should be pointed out that if a true WDSSF is required the rules for pseudopassivity must be followed.

The decomposed voltage wave scattering n-port is an interesting method to obtain general n-port adaptors for WDFs. However, this method is limited to true ladder configurations for the reference filter. The method has not been fully exploited and no real optimization procedures have been developed. Apparently it is a difficult problem to find a simple and efficient optimization procedure. Experience made by manually quantizing the parameters have shown that the method leads to long state coefficients. A more sophisticated parameter optimization procedure needs to be developed to make the method useful.

# chapter 5. AN ARCHITECTURE BASED ON DISTRIBUTED ARITHMETIC

*In this chapter the architecture used for the implementation is presented. First, a discussion of the possible partitionings is made and then different options concerning distributed arithmetic are discussed and compared. In the last part the details of a specific realization of a bit-serial architecture based on a distributed arithmetic processor is shown. All necessary building blocks to complete the architecture are discussed and the concepts of a global multi-processor architecture is presented.*

## 5.1. Architectural Options

A large variety of VLSI architectures are available for the implementation. Each of these architectures have numerous options at different levels. This makes the choice of architectures a difficult task. However, first the main class of architecture need to be chosen. This can be either programmable computer and data-path architectures or hardwired data-flow architectures. The control structure and the exact configuration should also be determined, e.g., single or multiple data and control streams, single or multiple processors etc.

### 5.1.1. Options Concerning the Computational Part

Given that the architecture should calculate innerproducts using distributed arithmetic, still various things on a more detailed level should be defined. There exist numerous ways to decompose and calculate the

innerproduct. The choices can be viewed as a *decision tree*, figure 5.1. It should be noticed that this is a simplified view since it is more a *hyperspace* than a tree. Many sublevels and many more choices exist, e.g., even if fixed-point arithmetic is selected various interpretations of the numbers can be made, two's-complement, one's-complement, sign-magnitude, canonic signed digits, etc. The ordering of the levels is not unique, however it corresponds to a top-down decomposition.

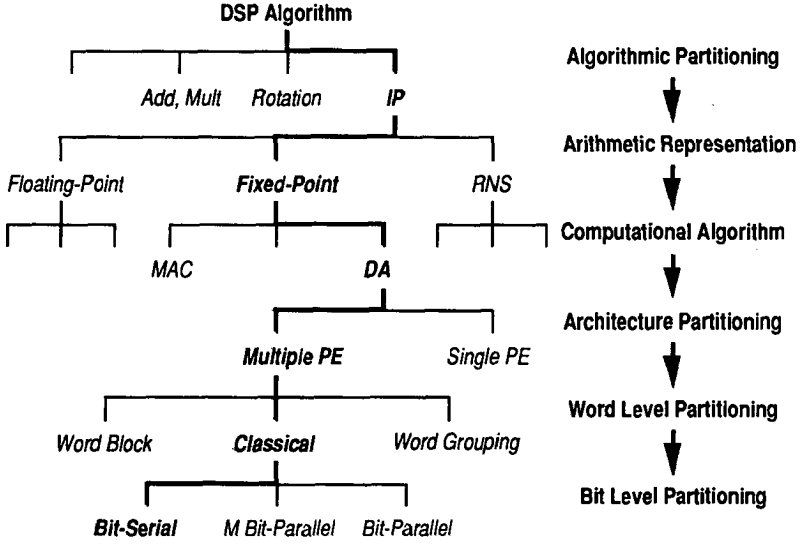



***Figure 5.1*** *A selection of architectural choices.*

In the process of choosing the architecture, decisions at many levels have to be taken. The process must start with an algorithmic partitioning where the DSP algorithm is decomposed into basic operations that can be used to fully realize the algorithm.

The next step is to determine the representation of signals and parameters. Knowing the operations and the arithmetic representation used, the word-level algorithm showing how to calculate these operations is defined. Thereafter, partitioning can be made to obtain different types of realizations concerning the time-space mapping. Finally, partitioning on word- and bit-level exactly determines the low level computations.

In DSP domain, especially fixed digital filtering, the typical operations are additions, subtractions, coefficient multiplications, shifts and delays.

Many algorithms can be decomposed in other basic operations, e.g., inner-products, vector based operations, complex multiplications, divisions and rotations, etc. The reason to use those operations can be that they naturally appear in an algorithm and therefore will simplify the implementation step. Other important aspects for the architectural choice are, required speed, latency, dynamic range, accuracy, sometimes also power consumption, and compactness.

### 5.1.2. Choices

To achieve a maximal speed, all possible parallelism of the algorithm should be extracted and exploited for the implementation. As it has been shown earlier this is achieved using a state-space description of the DSP algorithm. Another way of improving parallelism on the algorithmic level is to introduce pipelining. Considering these aspects it has been found that the *innerproduct* (IP) is a good choice for the basic operation.

Still several other decisions concerning the basic calculations must be made. First the number representation has to be determined. Floating-point representation and *Residue Number Systems* (RNS) have been disregarded here due to their implementation complexity. Often in DSP applications the precision is more important than the maximal dynamic range and this is another fact favoring fixed-point arithmetic.

The innerproduct can be calculated in a "lumped" fashion, i.e., as separate *Multiply-Accumulate* (MAC) operation or more implicitly using *Vector Multiplication* (VM). For the innerproduct considered here one of the vector multiplicands is constant. In this case VMs based on *Distributed Arithmetic* (DA) [Croi73, Pele74, Burr77, Bütt76] have been chosen due to the efficient implementation.

Different partitionings of the DA algorithm can be made. These can be classified in different ways [Burr77, Whit89, Burl89]. Burleson proposes a very systematic way dividing DA into *regular iterative algorithms* and *tree algorithms* by using transformation indexes [Burl89], while Burrus derives different variations, e.g. word and bit grouping, word block, etc., using convolution decompositions [Burr77]. The last classification is adopted here.

Some interesting variations derived using different choices on the last levels will be discussed in this chapter. These can be used to reduce the memory or the arithmetic hardware, or to increase the throughput. Nevertheless, the classical bit-serial DA algorithm is preferred in most situations.

### 5.1.3. Cost Parameters

A set of parameters or cost measures can be defined such that it is possible to compare different architectural variations. Given that the data wordlength is $w_d$ and the clock-cycle time is $T_{cl}$, a set that has shown to be convenient is:

- **Area A,** corresponding to the required chip area for the VLSI implementation. The area can be the restricting factor for the physical implementation, and it is thus an important parameter for comparing architectures. The importance of this parameter is less restrictive for modern VLSI technologies.

- **Cycle Time $T_C$,** is the most straightforward measure of the performance that defines the throughput of the architecture. Using bit-serial arithmetic $T_C = w_d\ T_{cl}$. The maximal speed can normally be calculated as the inverse of the minimal cycle-time.

- **Latency Time $T_L$,** is defined as the difference in between the time when the first part of the output is obtained and the time when the first part of the input was available, i.e., $T_L = T_{output} - T_{input}$. This measurement is especially important when recursive DSP algorithms are considered since calculated data will directly be reused in the subsequent sampling period. In these cases $T_L$ will determine the maximal algorithmic speed, i.e., sampling period.

- **Precision P,** in terms of, for instance, the *Signal-to-Noise Ratio* (SNR) is sometimes also given as a parameter to compare different realizations. The precision is usually proportional to $w_d$. This is a less important factor here when comparing DA architectures since the precision can also be controlled by changing DSP realization parameters.

Power consumption could be a cost measurement when comparing the architectures. However, this has not been considered here when it is very dependent on the technology used for the implementation. The power consumption is to some extent proportional to the area and the required clock-cycle time.

### 5.1.4. Arithmetic Bit-Level Choices

The first important choice to make is between bit-serial and bit-parallel arithmetic. A bit-serial system needs $w_d$ clock-cycles and one bit arithmetic

units to perform an operation, while a bit-parallel system needs only one clock-cycle but $w_d$ bit wide arithmetic units. The clock-cycle duration is proportional to the maximum number of gate-delays in the arithmetic units. This delay will obviously be substantially longer for bit-parallel arithmetic. This problem can be avoided using speed-up logic, for instance *carry look-ahead* circuitry [Hwan79], but then, the area and the complexity increases. The area time product is therefore not constant.

Yet, in some high-speed applications bit-parallel logic might be the only possible solution. Furthermore, bit-parallel implementations are of course more efficient for data dependent operations, such that, comparisons, divisions, etc.

Dependent on the application, the optimal arithmetic choice might not be a pure bit-serial or bit-parallel solution. Instead it can be advantageous to process M bits in parallel or more general to define a *grain size* M [Smit89] where $M < w_d$. The arithmetic would then become M-bit-parallel with two extremes, bit-serial and bit-parallel arithmetic. Having an architecture that supports different values of M clearly increases the flexibility. The drawback is that the control will undoubtedly be more complex.

The communication problem is also related to the grain size. A large overhead in area is necessary due to interconnections for bit-parallel architectures, especially in case of multi-processor architectures. Similarly the memory bandwidth is dependent on the format of the data. For instance, using a bit-serial implementation $w_d$ clock-cycles are required for each arithmetic operation, and using *serial-parallel conversion* before storing any data in the memories clearly reduces the required memory speed with a factor of $w_d/2$, if one read and one write is performed each period. The arithmetic data-format does not necessarily have to match the memory or the communication data-format.

Considering all these aspects, a true bit-serial architecture has been chosen here.

## 5.2. The Basic Distributed Arithmetic Processor

The basic concepts of DA seem to have evolved at many places independently. However, [Croi73] and [Pele74] are some of the more known early descriptions but other different derivations of DA can also be found. The innerproduct can be viewed as a two-dimensional convolution and by "mixing" the operations of convolution and multiplication distributed arithmetic is derived [Burr77]. Another similar derivation is obtained by

considering the innerproduct being an operation containing three indices $i, j$, and $k$, corresponding to the bits of coefficients and data and the terms in the innerproduct, respectively. Normally, a projection in $k$ is used leading to the multiply-accumulate approach. By projecting in $j$ instead distributed arithmetic is obtained [Ans89a].

### 5.2.1. Basic Concepts

Using distributed arithmetic a full innerproduct, i.e., a vector-multiplication between a coefficient vector **a** and a data vector **x**, can be performed in one operation. The result y is computed as:

$$y = \mathbf{a}\,\mathbf{x} = a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_Nx_N \tag{5.1}$$

The innerproduct in (5.1) can be denoted on a summation form:

$$y = \sum_{i=1}^{N} a_i x_i \tag{5.2}$$

When two's-complement representation is used for the signals and the coefficients, the sum can be written as:

$$y = \sum_{i=1}^{N} a_i \left[ \sum_{j=1}^{wd-1} x_{ij}2^{-j} - x_{i0} \right] \tag{5.3}$$

If the summation order is interchanged, the summation is first made over the terms between the j:th bit of $x_i$ times $a_i$, and then over the bits. Finally for bit 0, the sign bit, a subtraction is made.

$$y = \sum_{j=1}^{wd-1} 2^{-j} \sum_{i=1}^{N} a_i x_{ij} - \sum_{i=1}^{N} a_i x_{i0} \tag{5.4}$$

If $F_j$ is defined as a function of the j:th bit of each $x_i$

$$F_j = a_1x_{1j} + a_2x_{2j} + a_3x_{3j} + \ldots + a_Nx_{Nj} \tag{5.5}$$

the following result is derived:

$$y = \sum_{j=1}^{wd-1} F_j\, 2^{-j} - F_0 \tag{5.6}$$

In fact $F_j$ can only take $2^N$ different values, which can be pre-calculated and stored in a *look-up-table,* e.g., in a ROM. The j:th bit of $x_1$ to $x_N$ is then

used to address this table. To compute (5.6) *Horner's* form can be used:

$$y = ( \ldots (( 0 + F_{w_d-1})2^{-1} + F_{w_d-2})2^{-1} + \ldots + F_1)2^{-1} - F_0 \qquad (5.7)$$

Equation (5.7) clearly shows that only three different operations are required for calculating the innerproduct. First a *table look-up* to obtain $F_j$ where the j:th bit of every input is used to form the address, then an *addition* (subtraction) and finally a *divide by two*, that can be realized as a simple (single-bit) shift operation.

### 5.2.2. Hardware Requirements

To implement distributed arithmetic, an adder/subtractor with an accumulator register, and a ROM table are needed, figure 5.2. It is not necessary to implement a full $w_d$ bit wide accumulator. In fact, it only has to operate on $w_c$ bits, where $w_c$ is the ROM coefficient wordlength, i.e., the wordlength of the look-up code-world $F_j$. This is under assumption that a carry-save adder is used, if a carry-propagate adder or a parallel adder would be used the required width would be $w_c + 1$ bit. This reduces the hardware considerably since the coefficient wordlength is typically smaller than the data wordlength in most applications. The division by two is implemented as a shift at the input of the accumulator register.

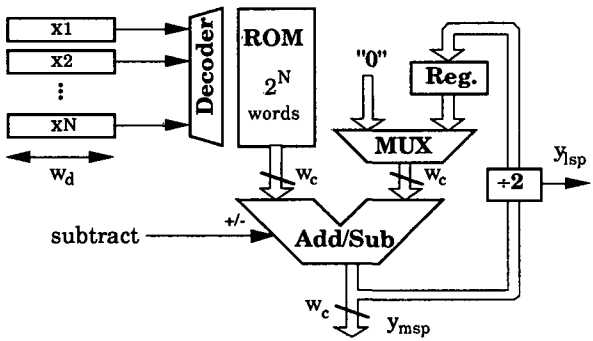



***Figure 5.2*** *Block Diagram of Distributed Arithmetic Processor*

The inputs $x_1$ to $x_N$ are shifted in with the least significant bit first. This generates the address of the ROM where $F_j$ is stored. $F_j$ is added to the contents of the accumulator register and the result is stored back into this register after a shift, i.e., a division by two. This is repeated for all bits of the inputs with exception for the sign-bit, for which instead a subtraction is performed. The *least significant part* (lsp) of the result is generated bit-

serially while the *most significant part* (msp) is available at the output of the adder in bit-parallel form.

The *Shift-Accumulator* (SA), i.e., the arithmetic part of the processor can be realized using different configurations. By using a *Parallel-to-Serial Conversion Register* (PSCR) for $y_{msp}$ and using a *multiplexor* (MUX) to select the correct part of the result, $w_d - w_c$ bits from $y_{lsp}$ and $w_c$ bit from $y_{msp}$, an externally fully bit-serial *Distributed Arithmetic Processor* (DAP) is obtained, figure 5.3a. On the other hand, using a *Serial-to-Parallel Conversion Register* (SPCR) for $y_{lsp}$ a fully parallel result $y_p$ can be formed combining it with $y_{msp}$, figure 5.3b. The first configuration is preferred here since it corresponds to a truly bit-serial solution. The term SA used hereafter, will refer to figure 5.3a.

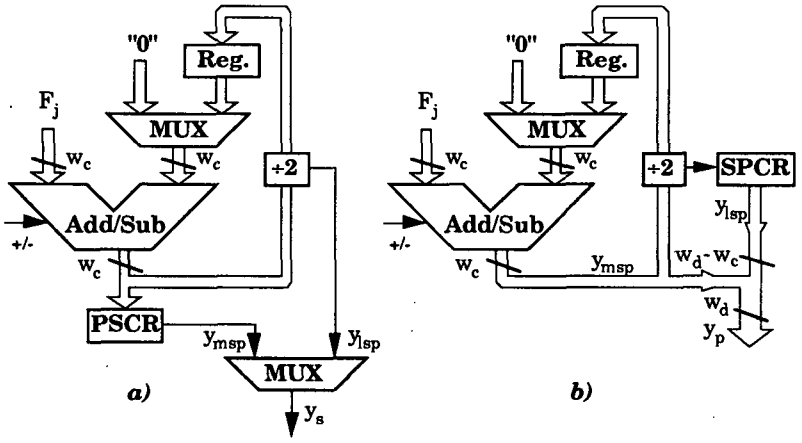



***Figure 5.3*** *Shift-accumulators with bit-serial or bit-parallel output.*

## 5.3. Variations to Reduce the Hardware

Several variations on the distributed arithmetic architecture have been proposed over the years. Here follows a short summary of some interesting deviations of the normal DAP. Some of them have been included and adapted for the presented methodology. The variations are mainly of two types, those that reduce the area and those that enhance the throughput.

### 5.3.1. Word Grouping and Innerproduct Decomposition

Reducing area often consists of trying to reduce the ROM size. For some applications the look-up-table size can be critical, e.g., for direct realization

of FIR filters with N taps the ROM size will be $2^N$. For the currently used technology and cell library the limit of N is between 10 and 11 corresponding to 1k respectively 2k word of ROM. This is the limit at which the maximum speed can be maintained. Of course by increasing the clock-cycle larger N can be allowed.

An obvious solution is to break up the innerproduct in L parts with N/L terms in each *partial innerproduct*, figure 5.4a. The arithmetic part increases L times and L-1 extra bit-serial full-adders are needed. The number of ROM words in each DAP is reduced to $2^{N/L}$. In figure 5.4a a decomposition with L = 4 is shown. By choosing an appropriate grouping of the terms in the IP, i.e., terms with equal wordlengths and similar magnitude, the best result is obtained. The width of the DAPs can in general be reduced to $w_c - {}_2\log L$ bit where $w_c$ would correspond to the direct realization. The total ROM size can thus be reduced from $w_c\, 2^N$ bit to approximately $L(w_c - {}_2\log L)\, 2^{N/L}$ bit. The throughput remains unchanged compared to the direct realization as well as the latency, that can be computed as $T_L = (w_d + w_c - {}_2\log L + {}_2\log L) = w_d + w_c$.

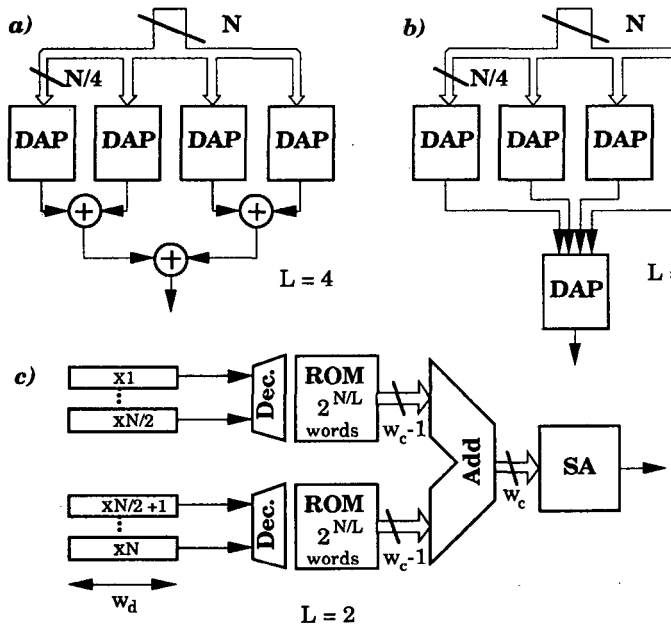



***Figure 5.4*** *Innerproduct decompositions.*

Another regular way to realize the decomposition is to use an additional DAP instead of an adder tree, figure 5.4b. By observing that the words in the ROM look-up table will equal the number of ones in the input vector a simplification can be made. Thus, the look-up table should work like a finite-state machine or more specifically as a counter. This can be compared to the *quasi-serial multiplier* described by Schwartzlander [Swar73]. The ROM for the *"summing"* DAP would only contain L+1 words and be $_2$logL wide, in the example of figure 5.4b the stored values would be (0,1,2,3 and 4). However, some irregularities in handling the sign-bit would occur.

Another possibility is to make a decomposition of the coefficients to make better use of the summing DAP. By factoring the terms in the partial innerproducts calculated by the first row of DAPs non-trivial coefficients for the last DAP are obtained. This should be realized using a normal full look-up table with $2^N$ words. Instead the value of all coefficients can be distributed such that the last DAP will no longer contain trivial coefficients. If this is correctly made, it is possible to reduce the total number of shift-accumulator bits. Using a proper optimization the alternative in figure 5.4b could become more economical than that of figure 5.4a. It also corresponds to a very modular and regular solution. The cycle time will remain the same, while the latency time will increase to approximately $2\,(w_d + w_c)\,T_{cl}$. The required coefficient wordlength $w_c$ would in most cases be shorter.

A third way to implement the decomposition is to modify the shift-accumulator, figure 5.4c. Instead of computing partial innerproducts and summing them to obtain the final result, the look-up entries are first summed together and then added accumulated in the shift-accumulator. The drawback of this method is that parallel adders are required in the first adder stages. Thus, this configuration is best suited for a complete parallel solution using only carry-propagate adders. Due to wiring between ROMs and ALUs a large area would be wasted.

The IP decomposition can be used whenever a reduction in area can be obtained, especially when the number of terms in the innerproduct is high. Very often applications such as FIR filters and discrete transforms contain symmetries in the coefficients. It is thus possible to directly decrease the number of terms in the IP. This is accomplished by simply pre-adding/pre-subtracting inputs pair-wise for linear phase FIR and for DCT [Defi89]. This directly decreases the look-up-table size to $2^{N/2}$ words. Another way to avoid the problem is to make a decomposition of the DSP algorithm such that it can be realized as cascaded sub-algorithms or sub-filters.

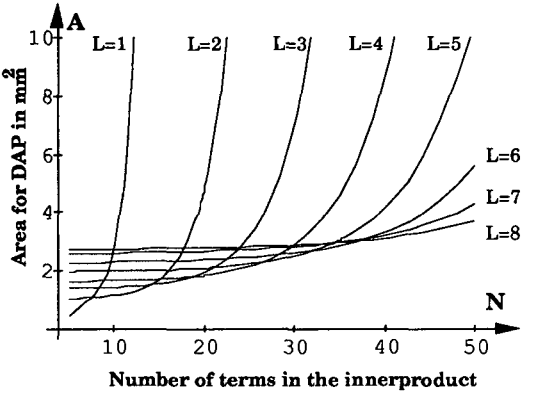



***Figure 5.5*** *Area versus different decompositions*

To give an idea of how the decomposition of figure 5.4a works for different parameters of N and L figure 5.5 is shown. The calculations are based on measures from the implementation described in Chapter 6 using a $w_c$ of 15 bits. These discrete functions that have been approximated by curves using a worst-case estimation. Much better solutions are obtained at the discrete points where the ratio N/L is an integer or when N and/or L are powers of two. Still it is remarkable to see the savings that can be obtained using this method. Clearly there are two regions in the diagram, one in the lower part where the area of the SA is important and another region dominated by the ROM area. It seems by choosing an appropriate value of L that the area becomes approximately proportional to N.

### 5.3.2. Offset Binary Coding

A technique to reduce the memory size with a factor of two was already proposed 1973 by Croisiere et al. [Croi73]. This technique has also been described as *Offset Binary Coding* (OBC) of the coefficients [Smit88a]. The goal is to reduce the redundancy in the ROM look-up-table. It can be accomplished by letting the bits represent (-1, 1) instead of (0, 1) as in a conventional binary code.

Using the observation

$$x_i = \frac{1}{2}[\, x_i - (-x_i)\,] \tag{5.8}$$

the two's complement representation of $x_i$ becomes

$$x_i = -\hat{x}_{i0}\,2^{-1} + \sum_{j=1}^{w_d-1} \hat{x}_{ij}\,2^{-j-1} - 2^{-w_d+1} \tag{5.9}$$

where $\hat{x}_{ij} = ( x_{ij} - \bar{x}_{ij} )$ and $\bar{x}_{ij}$ represents the complement of $x_{ij}$, i.e., a bit inversion. Now (5.4) can be rewritten as

$$y = \sum_{j=1}^{w_d-1} 2^{-j-1} \sum_{i=1}^{N} a_i \hat{x}_{ij} - 2^{-1} \sum_{i=1}^{N} a_i \hat{x}_{i0} + A_0 \tag{5.10}$$

where

$$A_0 = -2^{-w_d+1} \sum_{i=1}^{N} a_i \tag{5.11}$$

Here $\hat{x}_{ij}$ can only take the values -1 or + 1 instead of 0 and 1 for normal two's-complement digits. Defining $\hat{F}_j$ using (5.10) for the OBC the same way as $F_j$ was defined in (5.5), it can still take $2^N$ different values but only $2^{(N-1)}$ different magnitudes, cf. table 5.1. The ROM look-up-table can now be reduced to the half, i.e., $2^{(N-1)}$ values.

| Row | $x_{3j}$ | $x_{2j}$ | $x_{1j}$ | $F_j$ | $\hat{F}_j$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $-1/2\,(a_3 + a_2 + a_1)$ |
| 1 | 0 | 0 | 1 | $a_1$ | $-1/2\,(a_3 + a_2 - a_1)$ |
| 2 | 0 | 1 | 0 | $a_2$ | $-1/2\,(a_3 - a_2 + a_1)$ |
| 3 | 0 | 1 | 1 | $a_2 + a_1$ | $-1/2\,(a_3 - a_2 - a_1)$ |
| 4 | 1 | 0 | 0 | $a_3$ | $+1/2\,(a_3 - a_2 - a_1)$ |
| 5 | 1 | 0 | 1 | $a_3 + a_1$ | $+1/2\,(a_3 - a_2 + a_1)$ |
| 6 | 1 | 1 | 0 | $a_3 + a_2$ | $+1/2\,(a_3 + a_2 - a_1)$ |
| 7 | 1 | 1 | 1 | $a_3 + a_2 + a_1$ | $+1/2\,(a_3 + a_2 + a_1)$ |

***Table 5.1*** *Look-Up-Tables for N=3*

It turns out that the only difference between the column for $\hat{F}_j$ and $F_j$ is a constant offset of $-1/2\,(a_1 + a_2 + a_3)$, hereof the name offset binary coding. Moreover, it easy to verify that the column for $\hat{F}_j$ is anti-symmetrical around the middle of the table, i.e., the entries are pair-wise equal in magnitude but with a different sign. The table can then be reduced by the letting one term, in this example $x_{3j}$, determines the sign.

The hardware can be realized according to figure 5.6. One input is used to control whether to complement the other inputs or not, and to determine whether to subtract or not. This is realized by a set of XOR gates before the decoder. Instead of adding zero to the look-up entry at lsb time, the last term $A_0$ (5.11) is added. The overhead in hardware is normally small compared to

the gain in ROM size. Concerning cycle and latency time they remain unchanged.

For a more detailed description of OBC [Bütt76] and the tutorial by White [Whit89] are recommended.

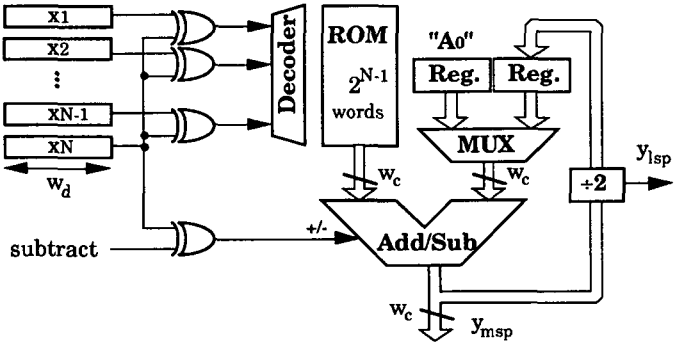



***Figure 5.6*** *Block Diagram of OBC Distributed Arithmetic Processor*

## 5.4. Throughput Enhancement

The above discussions have been focused on methods to reduce the hardware. Instead, if the throughput should be increased, several other options exist. Some classical variations on this theme are discussed below.

### 5.4.1. Parallel Form

To improve the throughput without having any constraints on the latency is fairly straight-forward. With a linear increase of the hardware, a linear increase of the speed is accomplished. Simply, having M processors available for one innerproduct, each processor is used for calculating every M sample, figure 5.7. For a bit-serial approach the limitation $M \leq T_L / T_{cl}$ is obtained, otherwise the DAPs would not be fully utilized.

The advantage with this variant is that the basic DA processor remains exactly the same. It is possible to combine this method with almost all other variants proposed here. It is even possible to obtain a throughput higher than one sample/clock-period even when the processors are bit-serial. The drawback is that the latency does not decrease at all, it remains the same, or even increases slightly because of the switches. This method is therefore less suitable for recursive DSP algorithms, the period cannot be reduced and thus, the maximally achievable sampling frequency is not improved.

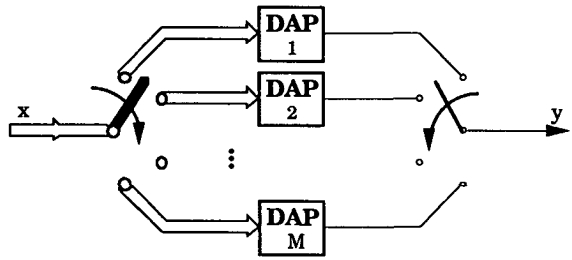



***Figure 5.7*** *Parallel architecture for distributed arithmetic.*

To overcome this problem *inter-period dependencies* can be explored, and if possible *inter-period scheduling* of the DSP algorithm can be made [Wanh91]. Another possibility would correspond to use *block-processing* methods as proposed by [Burr71]. Block processing is a generalization of the scheme in figure 5.7. However, for conventional binary arithmetic a slight overhead would be the outcome because of the necessity to perform some overlap-add calculations.

### 5.4.2. Bit Grouping

An alternative often proposed is to use M bits at the time [Pele74, Arjm81, Whit89]. This has the effect that the ROM requirement increases from $2^N$ to $2^{NM}$ word and this can be very restricting. On the other hand the cycle-time decreases to $T_C = (w_d / M)\, T_{cl}$ and even the latency time is improved to $T_L = (w_d + w_c)/M\, T_{cl}$ in the M-bit-parallel case. However, the clock-period, $T_{cl}$, will increase when M-bit parallel adders are required.

Apart from the increased ROM look-up-table, this method requires a basic structure very much like the normal DA processor of figure 5.2. The difference is that a division with a factor of $2^M$, or a shift of M bit, is needed instead of a simple one bit shift. To avoid redundancy $w_d$ should be a multiple of M. This is normally impractical from an implementation point of view. It should be observed that $w_c$ will grow compared to a conventional approach.

The arithmetic function would correspond to (5.12). Only $w_d/M$ iterations are necessary but the term $a_n x_{nk}$ can take $2^{NM}$ different values.

$$y = \sum_{k=1}^{\frac{w_d}{M}-1} 2^{-jM} \sum_{n=1}^{N} a_n x_{nk} - \sum_{i=1}^{N} a_n x_{n0} \qquad (5.12)$$

This technique can also be combined with offset binary coding and with some other options. It is especially advantageous to combine it with the word block structure of figure 5.4. This method reduces the latency as well as it increases the throughput. Due to this a reduced sampling period for recursive DSP algorithms is obtained. The solution is suitable for implementations where memory (ROM) is cheap, e.g., in realizations using discrete components. However, this is not always the case for ASIC CMOS implementations.

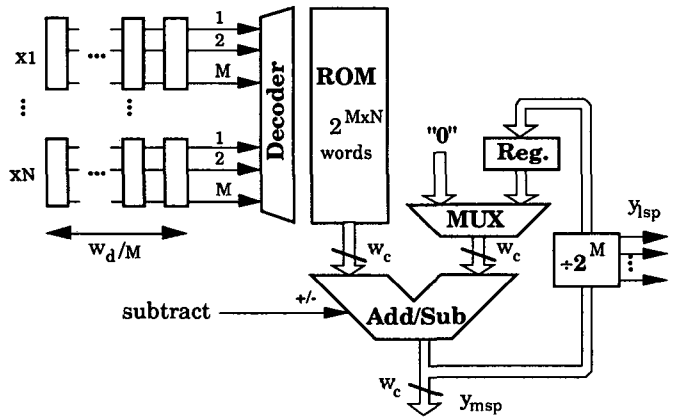



***Figure 5.8*** *Bit grouping architecture for distributed arithmetic.*

This method has been disregarded in the methodology mainly because of the irregularity with M-bit shift that would change the realization of the shift-accumulator. Only solutions using the same basic SA have been considered further.

### 5.4.3. Other Configurations

The options discussed in the two earlier sections are not the only possibilities. They can be combined with each other or with some other solutions. Many theoretically interesting configurations can be generated, but from a practical point of view the variations discussed earlier and the plain DAP are usually the most interesting. Surveys treating and comparing more possibilities are found in [Burr77, Whit89, Burl89, Arjm81].

An interesting variation arises using the encoding of data or coefficients by means of *Content Addressable Memories* (CAM) [Anso90]. A decomposition very similar to distributed arithmetic is used, and the concept is

especially attractive to use whenever an adjustable coefficient set is required. Unfortunately, the realization and implementation proposed in this thesis is not very well adopted for this alternative.

### 5.4.4. Used Options

The classical DA processor has been chosen as the best candidate for this methodology and it forms the basis for the arithmetic part of the proposed architecture. The bit-serial version from figure 5.3a is exclusively used, even if the bit-parallel configuration has been studied [Bals87].

The option corresponding to word grouping in figure 5.4a and 5.4b can be used whenever the overall complexity can be reduced, e.g. for long FIR filters. This can very well be controlled by the diagram in figure 5.5. Offset binary coding has been included, the extra hardware options that are required have been implemented. This option can be used in all cases except for when processors are time multiplexed. The correction term of (5.11) would case to many problems if the same hardware unit is used to calculate different innerproducts.

The parallel form of figure 5.7 can be used without altering the basic processor. The reversed scheme where DAPs are time multiplexed is also used. Finally, the bit grouping option has been excluded simply because of layout reasons. A completely different processor would be required.

## 5.5. Global Modules

Given that distributed arithmetic will be used, a large number of other architectural aspects still need to be defined. First of all, other components are required. The main blocks are shown in figure 5.9. The first part is the *processing unit*.

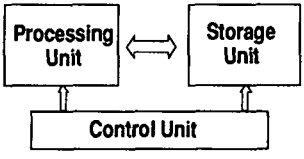



***Figure 5.9*** *Global modules.*

A *storage unit* containing data storage in form of registers or RAM, is required to retain different signal-values or variables. A *central control unit* is as well needed to generate the global clock- and control-signals. Test logic

to facilitate testing of the architecture can as well be included. Finally, pad drivers, level shifters and so called glue logic are required in most implementations even if they are not architectural blocks.

### 5.5.1. Processing Unit

The processing part, i.e., the *Processing Unit* (PU), of the architecture is mainly based on the DAP described earlier. The choice of using a true bit-serial DAP automatically simplifies the interconnections and gives a maximal clock speed, i.e., a minimal $T_{cl}$.

Other bit-serial processors can in principle also be included in the methodology. This can be a serial-parallel multiplier, that is a special case of the DAP. Basic processors such as bit-serial full-adders, subtracters, half-adders, scalers, etc., are also useful in some cases. The only constraint on any included element, is that it uses bit-serial in- and outputs.

To obtain other characteristics than two's-complement truncation and two's-complement overflow, a special unit to make a correction of the values can be used. This unit can be seen as a special processing element that can be cascaded with other processors.

### 5.5.2. Memory Unit

The storage unit can be implemented using a RAM or some simple shift-registers. The choice strongly depends on the used global architecture. Using the concept of a bit-serial data-flow architecture it is certain that data should be accessible in bit-serial form and it is also probable that multiple memory or storage elements are required.

Using shift-registers, data remain in bit-serial format and no conversion is needed. The length of the register, i.e., the delay, can be precisely adjusted on the level of a single clock-cycle. However, if the desired delay corresponds to a much longer period, e.g., K $w_d$ clock-cycles, a RAM of K words of $w_d$ bit is more convenient. Using a serial-to-parallel conversion at the input, and a parallel-to-serial conversion at the output of the RAM, allows the memory to operate as a long shift-register of K $w_d$ bit. In certain cases it might as well be possible to use a mixed solution where both types of storage units are utilized.

### 5.5.3. Control Unit

In fixed DSP algorithms, according to the definition from Chapter 3, all sequences are computational cyclic, i.e., the same sequences are repeated

over and over again and do not change. The sequencing can typically be controlled by a *finite-state machine*. The sequence can be programmed in a PLA or simpler stored in a ROM. Using a feed-back register to the ROM it will form the finite-state machine, this will be viewed in Chapter 6. The length of the sequences will depend on the global architecture.

Another kind of control unit is needed to generate the addresses for the storage modules if they are realized by RAMs. Each storage element can in principle have its own addressing unit but normally only one unit can be shared between all SEs. For the same reasons that the control sequences would be periodical, the sequence of addresses will also be periodical.

## 5.6. System Architecture

To exactly formulate the requirements on these modules the global architecture must be defined. This should show and specify how different parts can be interconnected. It should also specify the sequences, the control structure and what kind of variations can be applied or used.

Earlier it was mentioned that concepts from data-flow architectures will be used, and more specifically hardwired architectures. Still, the choices are numerous and a much finer definition will be made.

### 5.6.1. Multi-Processor Architecture

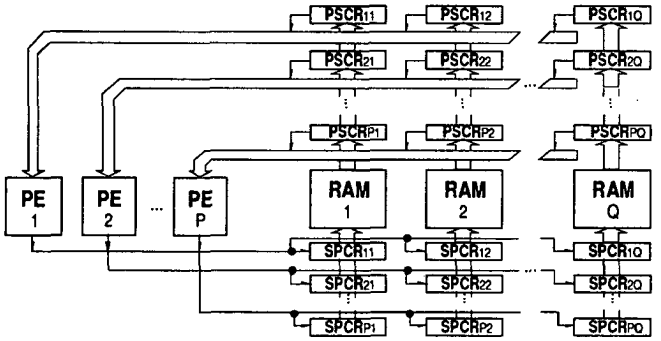



***Figure 5.10*** *Global system architecture*

It is important to choose an architecture that is regular and modular to simplify the final layout design. It also simplifies use of automatic layout tools. The choice of the global architecture has partly been based on the multi-processor architecture proposed in [Dinh84a, Dinh84b]. There, P

processing elements and Q RAMs are used in a maximal configuration, figure 5.10. By using *Serial-to-Parallel Conversion Registers* (SPCR) at the input of the RAM and *Parallel-to-Serial Conversion Registers* (PSCR) at the output, it is possible to store data in parallel form in the RAMs. For the sake of simplicity it is assumed that the same wordlength $w_d$ is used everywhere.

The number of processors P are chosen so that

$$P = \left\lceil \frac{N_{ip}\, T_{cl}\, w_d}{T_{sampl}} \right\rceil \tag{5.13}$$

where $N_{ip}$ is the number of innerproducts to calculate, $T_{sampl}$ the sampling time, $T_{cl}$ the minimum clock-cycle time, $w_d$ the data wordlength and $\lceil \bullet \rceil$ is the ceiling function. Equation (5.13) simply corresponds to the ratio between the required and the available throughput. However, this is under the assumption that no constraints due to latency time are present. It will be shown later on, that the situation is much more complicated.

The number of RAMs Q is chosen as the maximum number of terms in any of the innerproducts. Using P PSCR, in the figure 5.10, for each RAM, it is possible to supply all the processors with all necessary data. Using the same number of register at the input of the RAMs all the P results can be stored in each of the Q RAMs. Consequently, it is possible to use a redundancy where every signal is stored in every RAM. However this is seldom necessary and a reduction of the SPCR, in figure 5.10, can often be made to a single register for each RAM.

To simplify the configuration and to relax speed constraints on the RAMs, it is better to skew the operation of the PEs in time by a number of clock cycles. Then it is possible to load one row of the PSCR at the time, e.g., *row i*. After the data been loaded, $PE_i$ can start to process these, meanwhile *row i+1* of the registers are loaded. Once this is finished $PE_{i+1}$ starts to process and *row i+2* is loaded, etc. The resulting data will then appear at the SPCRs skewed in time by the same amount used for the PSCRs. Each RAM can then store these results one by one in a sequence.

The only limitation found using this architecture is that

$$(T_{read} + T_{write})\, P \leq w_d\, T_{cl} \tag{5.14}$$

This means that the time to make P read and write accesses of the RAM cannot be longer than the macro cycle-time, $T_{mc}$, which is $w_d$ clock-cycles long. However, by using several cascaded or parallel sub-systems of this kind this problem can be circumvented.

It has been demonstrated how this architecture can be used for different applications, [Dinh84a, Dinh84b]. The type of processor is not necessarily limited to a DAP, any kind of bit-serial processor can be included, if the latency time is exactly a multiple of $w_d T_{cl}$. Instead of giving the number of terms in an innerproduct, simply the number of data inputs is given.

An inconvenience is that each processor will work skewed one or many clock-cycles, and thus, a special set of control-signals should be generated for each PE. Each RAM will also in general require its own *Address Counter* (AC). A third inconvenience is related to the latency delay of the RAM and register banks. Due to the latency delay of the PEs and it is sometimes even possible that a zero delay is required from the *Storage Elements* (SE) and this is not possible strictly following the specifications of this architecture.

### 5.6.2. Proposed System Architecture Model

Due to the mentioned inconveniences a different system architecture model is proposed accordingly to figure 5.11. The architecture is simply divided into two parts, a *Processing Unit* (PU) containing P bit-serial PEs and the *Storage Unit* (SU) containing S storage elements. The SU should have M outputs and P inputs, it is assumed that each processor only produces one bit-serial output, even if this could be extended.

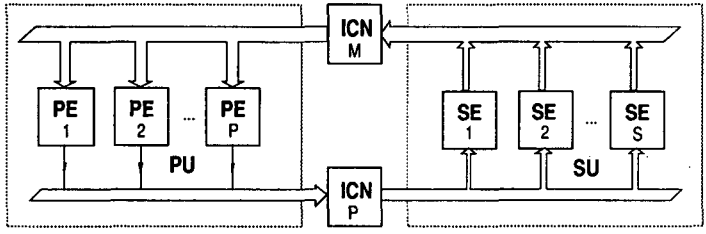



***Figure 5.11*** *System architecture model*

The SEs can all have different numbers of inputs and outputs, only the total sum of inputs should be greater or equal to P and the total sum of outputs should be greater or equal to M. Storage element $SE_k$ has $I_k$ inputs and $O_k$ outputs. The SU and the PU are interconnected through a hardwired *Interconnection Network* (ICN) in figure 5.11. In the normal case this is simply a fixed bit-serial bus between the two units. However, it can be a programmed switching network in some special cases.

The delay through the SU can be a multiple of $T_{mc}$, also including zero delay. From the scheduling point of view it is allowed to assume that any

stored data can be made available at any time, before it has been erased of course, and the scheduling should be periodic with the periodicity a multiple of $w_d T_{cl}$. This clearly simplifies the scheduling of a DSP algorithm on this architecture.

The storage elements can be implemented using mainly two different approaches. The simplest solution is to use registers. RAM can be used whenever the required life-times of the data are long. Also when the number of variables to store is high, as it would be in a multi-channel implementation, the RAM solution is convenient. The number of SEs S is given by some constraints that differs between the two different realizations. In case of RAMs it will be the access time and in case of a register implementation it will be the number of processors P.

### 5.6.3. Storage Element Realized by Registers

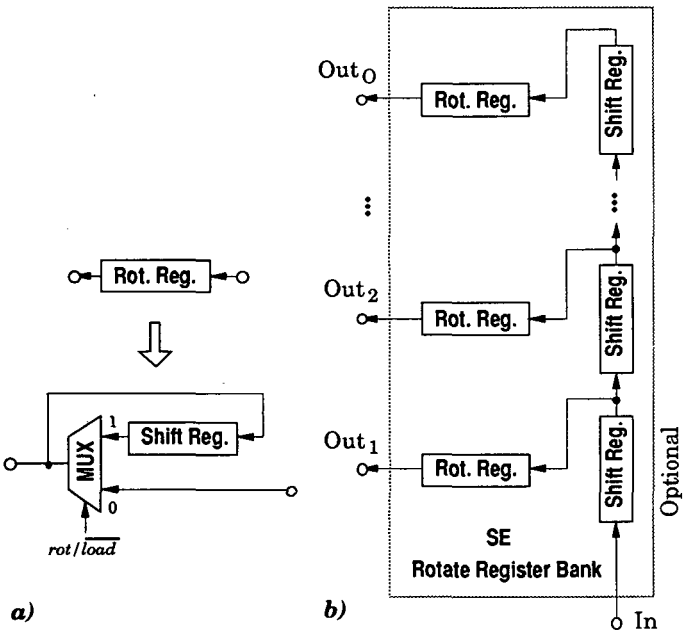



***Figure 5.12*** *Rotate Register based SE*

The first and simplest possibility for the realization is using shift-registers. The problem is that data sometimes need to be repeated during different *macro-cycles*, where a macro-cycle is $w_d$ clock-cycles long. This can

be viewed as the *life-time* of the data-samples is longer than the macro cycle-time $T_{mc}$. Then it is practical to use the realization shown in figure 5.12a. It is simply a shift-register with feedback. The signal *rot* / $\overline{load}$ decides whether a new input should be selected or not. When this is made the new value will be saved until a new load is performed.

The complete SE is constructed by combining a number of *rotate-registers* with normal shift-registers, as shown in figure 5.12b. The lengths of the registers in this configuration are a multiple of $w_d$, zero length included. In a normal case, they will all be exactly $w_d$ bit long except for the first shift-register that can be any number of bit long dependent on the application. The number of storage elements required, S, using exclusively register-based SEs, is always greater or equal to P.

This kind of SE always has a single input, while any number of outputs can be used. A special case is obtained when no repetition of the stored signal values is required. All rotate-registers would require zero length, i.e., they would be replaced by a wire. Only the shift-registers are left in this case. The simplest possible SE of this type corresponds to a single shift-register.

### 5.6.4. Storage Element Realized by RAM

The other possibility is to use a RAM. The problem here is that the data is preferably stored in parallel form, while the PU only contains bit-serial elements. According to the first system architecture discussed, conversion-registers are required. In this case two *Conversion-Register Banks* (CRB) are used, $CRB_A$ and $CRB_B$ in figure 5.13a. These two register banks work alternatively as input and output registers. When $CRB_A$ is used as input register then $CRB_B$ is used for the output and vice versa.

The CRBs are realized as suggested by figure 5.13b. They simply consist of K bi-directional registers that can be shifted in horizontally or vertically. It is not necessary that the clock used for the shifts in the two directions is the same. The clock-cycle for the vertical load can be increased to match the RAM-cycle time $T_{RAM}$. The number K corresponds to the maximum of I and O. By using an implementation of the registers where the inputs and outputs are in *high-impedance* state when not used, all multiplexers in figure 5.13a can be avoided.

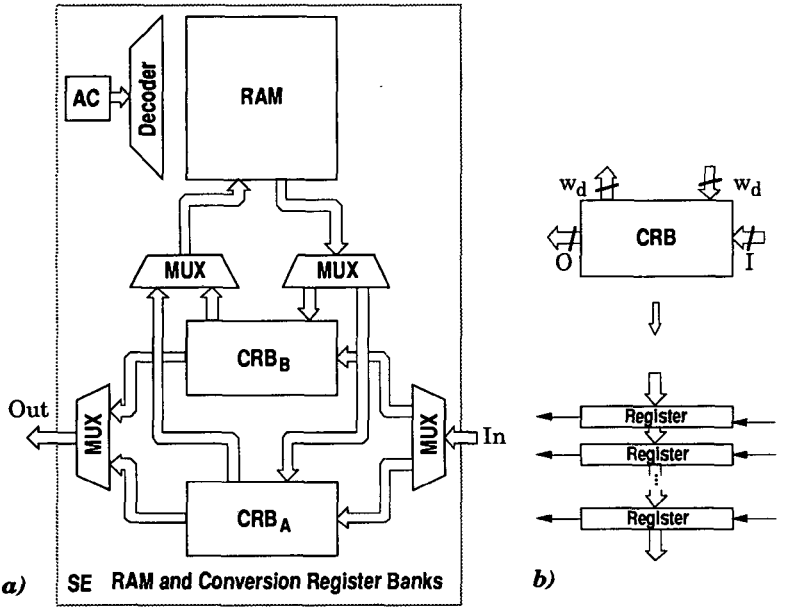



***Figure 5.13*** *SE based on RAM with conversion-register banks*

A sequencer realized by a counter is used to generate the address for the RAM, it is denoted *Address Counter* (AC) in figure 5.13a. Due to the fact that all DSP algorithms that will be realized by this architecture are periodically repetitive and the address sequence will also be periodical.

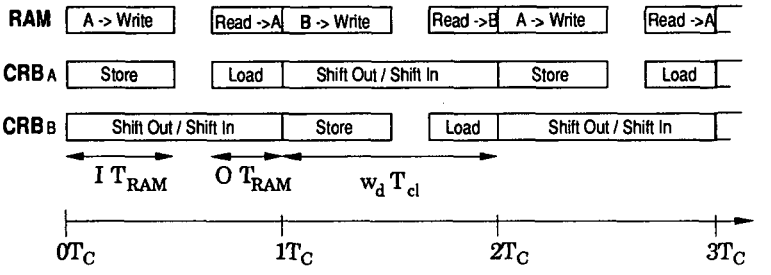



***Figure 5.14*** *Sequence*

The operation sequence starts with that the data in $CRB_A$ is stored in the RAM during I RAM-cycles. A RAM-cycle, $T_{RAM}$, is a multiple of the clock-cycle $T_{cl}$. For simplicity it is assumed that the used $T_{RAM}$ corresponds to the

maximum of $T_{read}$ and $T_{write}$. Meanwhile storing these data, $CRB_B$ is used as a shift-register. While data is shifted out on one side, new data is simultaneously shifted in at the other side. At the time I $T_{RAM}$ all data in $CRB_A$ have been stored. Then the next phase can start when O data is loaded from the RAM to $CRB_A$ during O RAM-cycles.

When one macro-cycle has elapsed, the data in $CRB_B$ is fully replaced by new data to be stored. $CRB_A$ will now be used as the input/output register while $CRB_B$ will be used to store to and load from the RAM. This sequence is represented in the diagram of figure 5.14.

A solution of special interest is obtained when I = O = 1 and S = P = $N_{IP}$, where $N_{IP}$ is the number of innerproducts to calculate. Then one SE and one PE per state-variable are used. Each SE will contain two registers only, figure 5.15a. This configuration is especially useful for a multi-channel configuration where one SE is used for each state-variable. A simplified version is shown in figure 5.15b where one dedicated SPCR and one PSCR is used. However this structure requires that the RAM is faster, i.e., $T_{RAM} = T_{cl}$. This exactly corresponds to the solution found in the single chip 32-channel PCM filter implementation presented in [Renf84a, Renf84b, Sjös86].

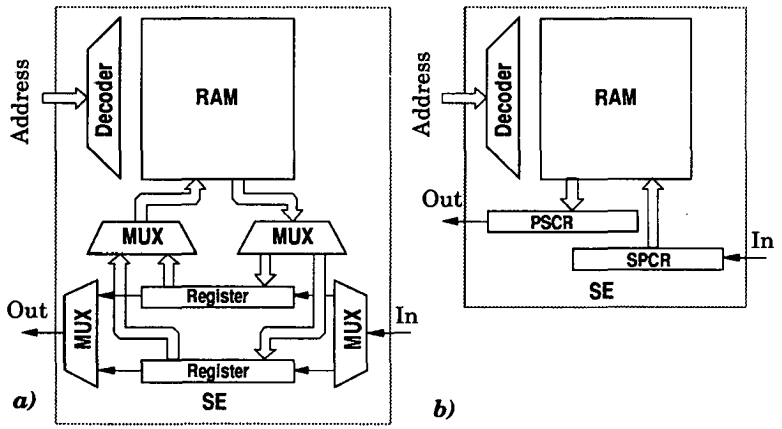



***Figure 5.15*** *SE with only one input and output*

The sum of the number of inputs and outputs to the RAM based SE are limited by:

$$T_{RAM}(I + O) \leq w_d T_{cl} \qquad (5.15)$$

If (5.15) is not fulfilled I or O must be reduced. This can only be done by storing some variables on another SE. Accordingly, the minimum number of SE of this type used in a storage unit can be calculated as:

$$S = \left\lceil \frac{T_{RAM}(P+M)}{w_d T_{cl}} \right\rceil \tag{5.16}$$

It is possible to increase S to simplify the scheduling, the sequence or sometimes even to reduce the area. If multiple SEs based on RAMs are used in a storage unit, it is also possible to share the address counter among the SEs at least if the cycle-time of the address sequence is the same.

### 5.6.5. Quantization and Saturation Unit

Constraints on the type of quantization and overflow characteristics is often given from DSP domain. In this methodology constraints given by the WDFs are used. A module that performs the necessary corrections, called the *Quantization and Saturation Unit* (QSU), is used. Sign-magnitude truncation characteristic instead of two's-complement truncation and saturation characteristic in case of overflow is implemented by the QSU.

This unit could be viewed as a special processing element. However, this is not optimal when uncorrected values will be stored in the SU. Uncorrected values contain an extra *guard-bit* that can be canceled after correction by the QSU. This is explained in detail in Chapter 6. The latency could also be increased by placing the QSU in the processing unit. Data will circulate twice in the loop before it is ready to be reused.

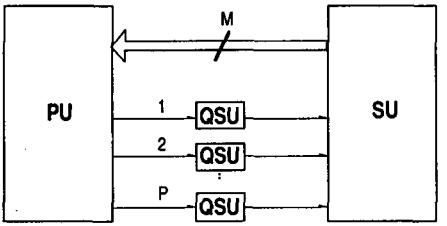



***Figure 5.16*** *Placement of the quantization and saturation unit*

In most cases it is better to perform the correction directly after the PE, i.e., before the value is stored and reused. The configuration in figure 5.16 is obtained. Clearly the same number of PEs and QSUs will be required. The overall delay is of course increased by these units but this is unavoidable. It can be compensated by reducing the delay in the SU by the same amount.

### 5.6.6. Timing Constraints of the Architecture

Clearly there is a difference between the timing constraints of recursive and non-recursive DSP algorithms. In the first case the minimum cycle-time is determined by the latency time in any recursive loop of the data-flow. Unfortunately, the latency time is high in a bit-serial system.

The minimal possible cycle-time for a non-recursive algorithm $T_{Cmin}$ will be equal to the macro cycle-time, $T_{mc} = w_d T_{cl}$. This time can (in the one channel case) be decreased by using a higher number of processors than the number of innerproducts to be calculated, i.e., $P \geq N_{IP}$. Generally, if $N_C$ channels with $N_{IP}$ innerproducts should be processed using P processors, the minimal achievable non-recursive cycle-time for the architecture can be expressed as:

$$T_C = \left\lceil \frac{N_{IP}}{P} \right\rceil N_C T_{mc} \tag{5.17}$$

if the calculation of the channels cannot overlap in time. If they can overlap, the cycle-time can be reduced to:

$$T_C = \left\lceil \frac{N_{IP} N_C}{P} \right\rceil T_{mc} \tag{5.18}$$

The minimal sampling time $T_{sampl}$ for the non-recursive algorithms equals the cycle-time, either the one in (5.17) or the one in (5.18).

In case of recursive algorithms the situation is different. The minimum cycle-time corresponds to the overall latency-time, $T_L$. This can be calculated as the sum of the latency time for all elements in the loop of the architecture, figure 5.11. In the most common case, one PE, one QSU and one SE are included in the loop. The latency-time can then be calculated as:

$$T_L = T_{Lpe} + T_{Lqsu} + T_{Lse} \tag{5.19}$$

For the developed hardware the minimum processor latency-time is $T_{Lpe} = (w_d + w_c)$, the QSU latency-time $T_{Lqsu} = w_d$ and the minimal storage unit latency-time $T_{Lse} = 0$. This gives a total latency of $T_L = (2\ w_d + w_c)T_{cl}$ for this specific architecture.

To compute the minimal sampling period for the recursive case is more complicated. However it can be divided in two parts:

$$T_{sampl} = T_C + T_{Leff} \tag{5.20}$$

where $T_C$ corresponds to (5.17) or (5.18). $T_{Leff}$ is the *effective latency time*, i.e., the effective time that is lost due to the latency in the loop. For the proposed architecture it has been found that this can be calculated as:

$$T_{Leff} = \mathrm{Max}\left[\ 0,\ T_L - T_{mc} - \left(\ T_C - \left\lceil \frac{T_C}{N_C} \right\rceil\ \right)\right] \qquad (5.21)$$

Clearly, the effective latency time must be non-negative.

The expression in (5.20) is valid both for the recursive and the non-recursive case since $T_{Leff}$ always will be zero for a non-recursive algorithm. It also implicitly gives the required value of P. Furthermore, the exact required clock-cycle time $T_{cl}$ can now be calculated.

In non time-critical applications the total latency can be increased to

$$T_L = \left\lceil \frac{T_{Lpe}+T_{Lqsu}+T_{Lse}}{w_d} \right\rceil w_d\, T_{cl} \qquad (5.22)$$

i.e., a multiple of $w_d$ clock-cycles. This simplifies the control-signals and introduces the possibility to pipeline the interconnections. The decision to use (5.19) or (5.22) is made on basis of the DSP constraints.

### 5.6.7. Shimming Delays

To equalize the latency-time of each loop in the architecture accordingly to (5.22), shimming delays are used. These are simple shift-registers. Naturally, it would be simplest to include these delays directly into each module such that a latency time for any unit would be $T_{Lunit} = K\, T_{mc}$. Doing this is not optimal since each unit will increase its delay to a multiple of $T_{mc}$ and the overall delay in the loop could increase with more than $T_{mc}$.

It has been found to be better to distribute the shimming delays between units and to increase the loop delay only to the nearest higher multiple of $T_{mc}$. The shimming delays can also be used to pipeline interconnections, i.e., to introduce clocked elements in long wires connecting global blocks. This makes it possible to use smaller bus drivers without limiting the speed.

### 5.6.8. Testability

An important aspect of an architecture is its testability. Clearly a DSP algorithm with very few inputs and outputs has low *observability*. Most of the internal states are not directly observable at the output. Concerning the *controllability* the same is true. Only the inputs can be directly controlled externally.

Both these aspects can be improved by adding the possibility to measure and to change internal signals. The proposed multi-processor architecture is well adopted to this concept. Figure 5.17 shows a simple configuration that allows testing of the PEs and the SEs separately.

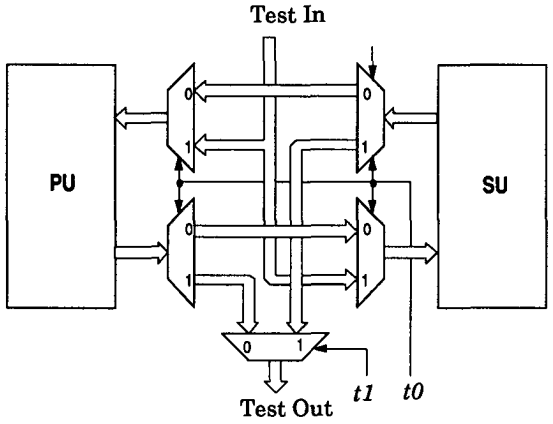



***Figure 5.17*** *Test configuration*

If the first control signal *t0* is low the architecture will work in normal processing mode. When *t0* is switched high then the outputs from either the SU or the PU can be observed. This is controlled by *t1*. At the same time external test inputs can be fed to the units.

The scheme is much simpler than it appears to be in the figure. By using the possibilities of CMOS implementations, the switches can be implemented with an almost negligible overhead. The bit-serial buses in figure 5.17 will have the width of Max { M,P } or P.

# chapter 6. CMOS IMPLEMENTATION

*In this chapter details of the CMOS implementation for the distributed arithmetic architecture are shown. Different implementation options are discussed and compared. Some layouts and strategies for the design are presented. Finally, a summary of the obtained results is given in form of figures of merit and simulation results.*

## 6.1. The Implementation Task

### 6.1.1. Parameterizable Modules

The architecture described in the previous chapter is based on some few basic blocks. The first main component is the *Processing Unit* (PU) formed by the processing elements. The primary used PE is the *Distributed Arithmetic Processor* (DAP). The DAP should be *parameterizable* for various configurations and available with different options. The same flexibility should also be provided for the other main components like the *Quantization and Saturation Unit* (QSU), the *Storage Unit* (SU) and also for the *Control Unit* (CU). They have all been designed to support different options. This is necessary to make these modules useful for a large range of applications.

### 6.1.2. Design Strategy

The full custom design approach has been adopted for the implementation. This choice has been made for many different reasons as explained in Chapter 2, but principally for improving the speed as much as possible and for reducing the required area.

The principle that has been followed is to first design some very elementary structures such as, inverters, transmission-gates, dynamic register cells, XOR gates, full-adder, etc. These structures are used to form the basic cells, e.g., the shift-register cell and the shift-accumulator bit-slice. A complete library of cells has been developed for each module, i.e., DAP, QSU, etc. The modules are then built-up by connecting those cells directly to each other, i.e., using interconnection by *abutment*.

A large amount of time was spent to design the library cells and to optimize the speed using an iterative process involving SPICE simulations and redesign. Geometries with 45° angles are also employed to shorten wires and to reduce the area. Most layouts have been fully hand-crafted. In some few cases symbolic layout tools have been used in a first step, followed by a manual compaction of the obtained layout.

### 6.1.3. Used Logic

Mostly all cells are realized by a mixture of dynamic and static parts. All clocked elements such as latches and registers are implemented using the well-known dynamic structures from [Mead80, West85]. Only non-critical dynamic structures are used, to reduce problems with charge redistribution, etc. The non-overlapping two-phase clock scheme is employed everywhere. Other parts are realized using pass-transistor structures and transmission-gates.

In an effort to reduce the complexity and the area, precharged dynamic logic is used in larger structures where the reduction of complexity is significant. This is most evident in RAM and ROM structures and in the decoders. Approximately half the area is gained.

Everywhere full advantages have been taken of the logic possibilities in CMOS, i.e., alternative ways of forming and combining logic functions. Full boolean expressions are reduced to a single-stage structure and not realized by cascaded gates.

### 6.1.4. Technology

The technology used so far for all implementations and layouts is the CMN20a from VLSI Technology Inc. (VTI). This is a polysilicon-gate twin metal 2 μm CMOS process. The goal when designing the cell library was that all modules should work at the same maximal speed, that naturally should be as high as possible.

In the present version of the module library all modules are expected to work at a maximal clock-rate of 100 MHz, with exception for a RAM memory that works at half of this speed. However this has only been possible to verify by detailed simulations, although different prototype circuits have been integrated. No high speed tests have been possible until this moment. The only real speed bottle-neck left are pad-drivers and I/O level-shifters. These all originate from the standard cell library of VTI. However, by converting data from serial to parallel form before generating the chip outputs and from parallel to serial form for the chip inputs, this problem can be avoided.

Fortunately, there exist new technologies today from VTI what are a direct down scaling of the used technology. The CMN16 corresponds to a 1.5 μm and the CMN10 to a 1.2 μm technology. Thus it will be possible to directly reuse the designed cells and libraries in these technologies. Of course, supplementary simulations and verifications should be made when moving over to a different technology.

## 6.2. Distributed Arithmetic Processor

### 6.2.1. DAP Module

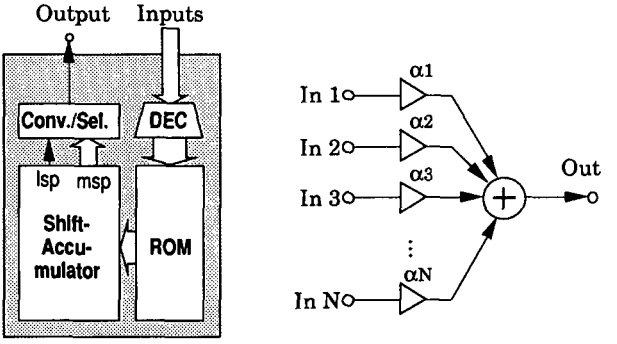



***Figure 6.1*** *Distributed arithmetic processor, logic diagram and signal flow graph symbol.*

The basic distributed arithmetic processor, DAP, consists of a *decoder*, a *Read Only Memory* (ROM) and a *Shift-Accumulator* (SA), figure 6.1. The processor should work on bit-serial input data and generate bit-serial output data. In principle all the calculations should be performed exactly, i.e., no quantization or overflow is allowed internally.

It should be possible to generate the DAP with different options. This concerns wordlength, content of memory, decoder and ROM for offset binary coding, etc. Therefore, the processor has been defined with a number of parameters.

### 6.2.2. Parameterization

The complete DAP can be parameterized using a set of parameters. The most important are listed below.

- **The accumulator wordlength $w_c$** that is determined by the required wordlength for the precalculated partial product $F_j$. This will directly determine the width of the DAP and indirectly the size of both control-signal drivers and ROM buffers.

- **The number of innerproducts $N_{ip}$** that are calculated on the same DAP. If this number is larger than one the decoder must be extended to select the currently calculated IP.

- **The number of terms $N_{ti}$** in the innerproduct $IP_i$. The total number of terms in all IPs will be rounded to the nearest higher multiple of eight, due to row-column decoding of the ROM. If $N_{ti}$ differs between the IPs calculated on the same DAP a special *address mapping unit* should be added.

- **The coefficient set of each IP** that obviously should be programmed into the ROM matrix. In fact the first parameter $w_c$ could be implicitly derived from the coefficients.

- **An option OBC** that tells if OBC is used, and thus, determines the type of decoder that is used, the input buffers and the control signals.

- **Option HM and VM** that indicates that *horizontal* and/or *vertical mirroring* will be used. This is useful to save space when two DAPs are placed close to each other in a floorplan. Border cells can be shared and sometimes also control signal drivers, input drivers, power supplies, decoders, etc.

- **Output driving capability $D_c$** that indicates the size of the output driver.

- **Options UL, LL and LR** determine where connectors for the control-signals should be available. This can be *upper left*, *lower left* and/or *lower right* corner.

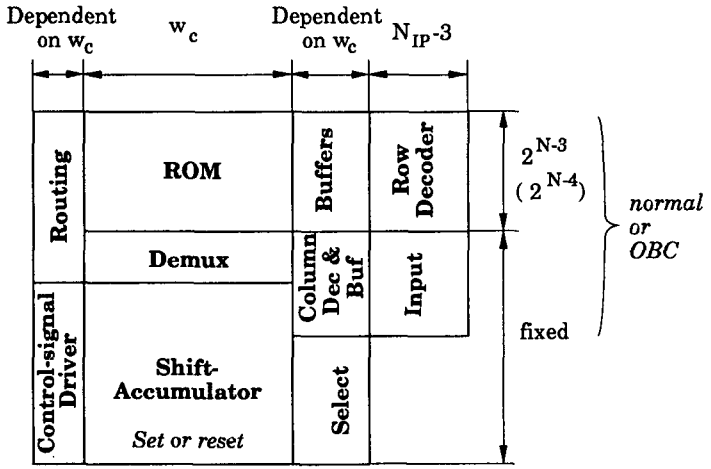



***Figure 6.2*** *Floorplan of parameterized DAP*

The figure 6.2 shows the consequence that some of the parameters have to the floorplan of the DAP. In the current implementation the ROM buffers are of fixed size with sufficiently driving capability for $w_c \leq 20$. In future versions the drivers should be parameterized and specifically tuned from case to case.

### 6.2.3. Signals

The complete set of signals used in the DAP is shown in the timing diagram in figure 6.3. The diagram is drawn for a processor with $w_c = 4$ and for $w_d = 7$. A schematic of the corresponding DAP can be found in figure 6.11. The basic clock in the system is the non-overlapped two-phase clock realized by $\phi_1$ and $\phi_2$. The input is synchronized on the signal *sub* and is then clocked in on the clock-phase $\phi_1$. The output is clocked out on phase $\phi_2$.

The signal *sub* is used to force a subtract instead of an add in the SA and *reset1* and *reset2* to reset the internal registers before starting a new calculation. The complementary signals *rp2* and *lp2* are used to control whether to shift or to load the parallel-to-serial conversion register, cf. figure 5.3a. Finally *msp2* and *lsp2* are used to select the correct part of the result.

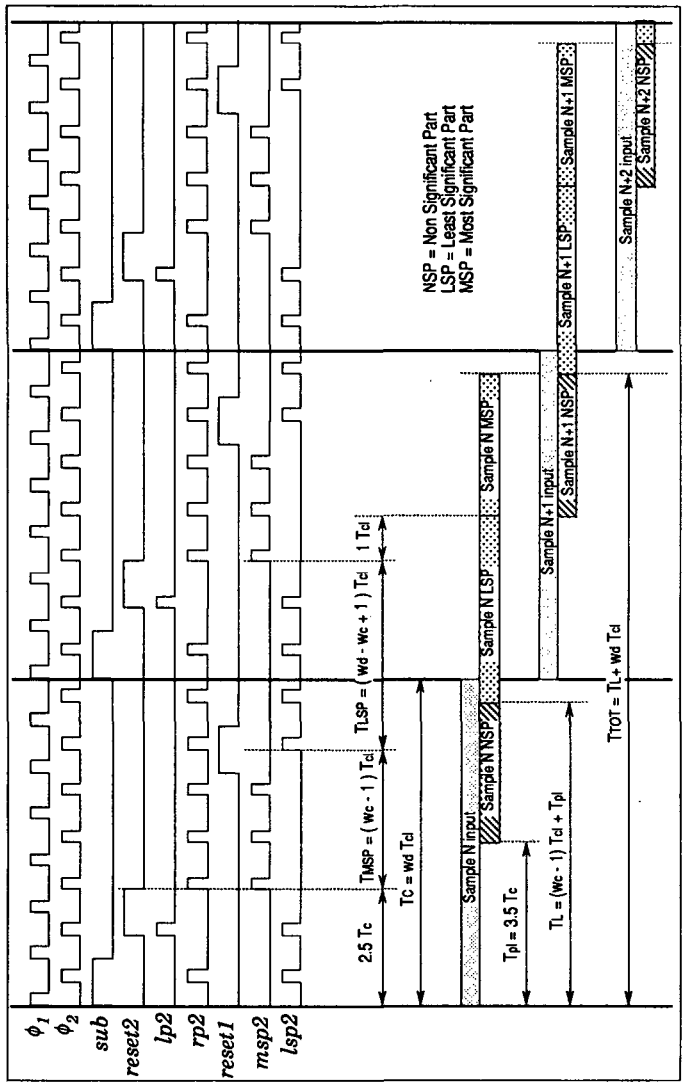



***Figure 6.3*** *Timing diagram for the DAP*

The cycle-time $T_C$ and the latency-time $T_L$ can be extracted from the diagram. It is also possible to define a number of timing parameters. For

instance, the required processing time for the least and most significant parts, $T_{LSP}$ respectively $T_{MSP}$, the pipelining-time $T_{pl}$ and the total delay-time $T_{TOT}$. The last parameter $T_{TOT} = T_{pl} + (w_c + w_d)\ T_{cl} = 13.5\ T_{cl}$ where $T_{cl}$ is the clock-cycle time.

### 6.2.4. Layout

The layout of a complete 4-bit DAP is presented in figure 6.4. The number of inputs to this processor is seven and thus the ROM contains 128 words. The parts of figure 6.2 can easily be recognized. The obtained layout is very dense and can be automatically generated using the parameters described above and the developed cell library.

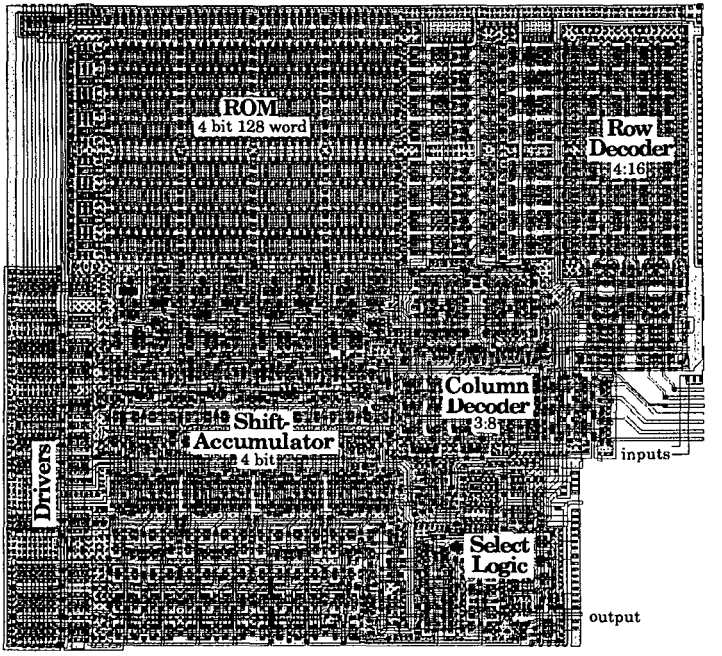



***Figure 6.4*** *Layout of a four-bit shift-accumulator*

The overall size for this unit is 622 μm x 670 μm and the DAP contains 1050 transistors given that half of the ROM matrix is filled with transistors. This gives an overall square density of 2520 MOS/mm$^2$ including the white space in the lower right corner.

To explain more details about the actual realization of the shift-accumulator and the ROM table, separate sections describing the implementation of them follows.

## 6.3. The Shift-Accumulator

The arithmetic part of the DAP is the shift-accumulator. The SA consists of an adder/subtractor, an accumulator register with built-in right-shift and repeater for the most significant bit. To be conformed with the bit-serial input/output it should also contain a parallel-to-serial conversion register for the most significant part of the result as discussed in Chapter 5. Finally, the correct part of the result should be selected, because it will consist of two different parts, the *most significant part* (msp) containing $w_c$ bit and the *least significant part* (lsp) the remaining $w_d$ - $w_c$ bit.

For the implementation of the SA two main options exist. It can be based on a *Carry-Save Adder* (CSA) corresponding to a bit-serial approach, or it can be based on the *Carry-Propagate Adder* (CPA), which then is the bit-parallel approach. These two alternatives differ both in performance and complexity. The CSA allows a minimal clock period but leads to an elevated latency time. The CPA on the other hand, demand a longer clock period but minimizes the latency. In the following two sections, realizations of both types are described.

### 6.3.1. Carry-Propagate Adder

The carry-propagate adder, sometimes also called *carry-ripple-through adder*, is normally used in arithmetic units working on bit-parallel data. When the input data is in bit-serial form as in distributed arithmetic, the CPA can still be used since the partial sums (the look-up-table entries) are available in bit-parallel format. Intuitively a bit-parallel accumulator should be faster than a bit-serial one, but the CPA suffers from the carry-propagation delays which limit the clock-rate. On the other hand the latency time, or the total delay time for an operation, is minimal since the result is directly available in parallel form. This can be used such that the overflow and quantization corrections can be combined with the SA itself [Bals87].

The required hardware for a basic shift-accumulator would be relatively simple, figure 6.5a. This structure generates the result in parallel form. To obtain a fully parallel result a shift-register of length ($w_d$ - $w_c$) should be added at the lsp output. By reading this register in parallel with the msp result a completely bit-parallel result can be obtained. An R in the register cell indicates a reset and # indicates register with parallel load.

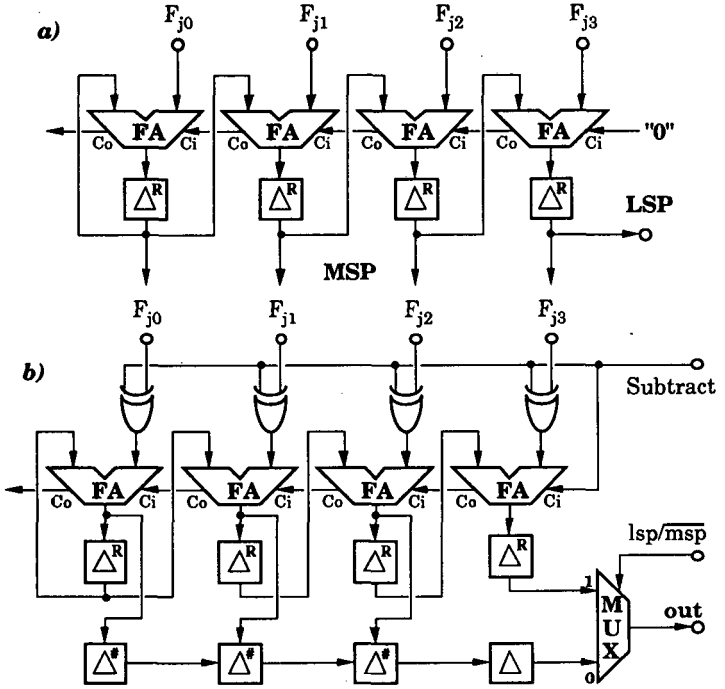



***Figure 6.5*** *Shift-accumulator based on a Carry-Propagate Adder*

On the other hand when a bit-serial solution is desired the structure in figure 6.5b should be used. A MUX is used to select the correct part of the result, the ($w_d$ - $w_c$) lsb bits will occur in the upper branch and the $w_c$ msb bits in the lower. The XOR gates at the input are used for bit-wise inversion to generate the negative two's-complement number in case of subtraction.

To circumvent the problem of the carry-propagation-delay different solutions exist. Carry-look-ahead schemes [Anso89] can be employed to reduce the delay. However these modifications usually drastically increase the complexity and area of the shift-accumulator. The decision whether to use them or not must be based on the area/speed trade-off that is achieved in different technologies.

### 6.3.2. Carry-Save Adder

The CSA is usually used for realizing serial-parallel multipliers. The main

feature with this adder is that no carry-propagation is required between different stages since the carry is saved and reused in the same stage in the subsequent clock-cycle. The carry-propagation delay is minimized this way. The basic structure for a 4-bit adder is shown in figure 6.6. It consists of one *Full-Adder* (FA) and two registers (latches) per bit.

The computational path between the delay elements is minimal. Another useful feature, is that this adder has a built-in *add-shift-accumulate* function. This exactly corresponds to the needs of distributed arithmetic. The problem, however, is that the result is not available directly after the last operation, due to the pipelining. All the contents of the registers must be derived at the output before the subsequent innerproduct can be calculated. This can only be done by adding zeros to the contents of the registers and propagating the result towards the output, one bit at the time. It will require $w_c$ extra clock-cycles to derive the complete result at the output. During this time, the shift-accumulator is occupied.

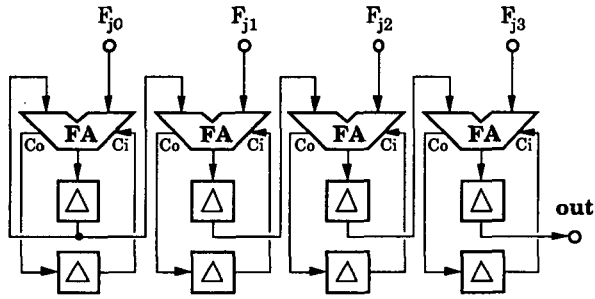



***Figure 6.6*** *Carry-Save Adder*

To increase the throughput in the CSA an extra set of registers can be added, figure 6.7. When the last bit of an innerproduct has been computed, the contents of the *sum-* and *carry-registers* are down-loaded into these registers. Then the computing of the successive innerproduct can thus start immediately. Parallel-to-serial conversion registers are used. Another full-adder with a carry latch is used for summing the contents of the sum- and carry-registers. This full-adder is also used for adding a one to the lsb when subtracting. To complete the shift-accumulator two multiplexors are used to select the desired part of the result.

To be able to perform subtractions in two's complement an XOR gate is added at each input, figure 6.7, similarly to the configuration used for the CPA. Using these gates the input can be inverted by letting the control

signal *"Subtract"* high and a subtraction instead of an addition is performed. If OBC is used, the *"Subtract"* signal is formed by XOR of the normal subtract and the term that controls the sign. To correct for the small offset that is generated by OBC the carry-registers can be programmed with reset or set. This is indicated as an **R** and an **S** in the matching registers in figure 6.7. This corresponds to add the $A_0$ term from (5.11) at lsb time.

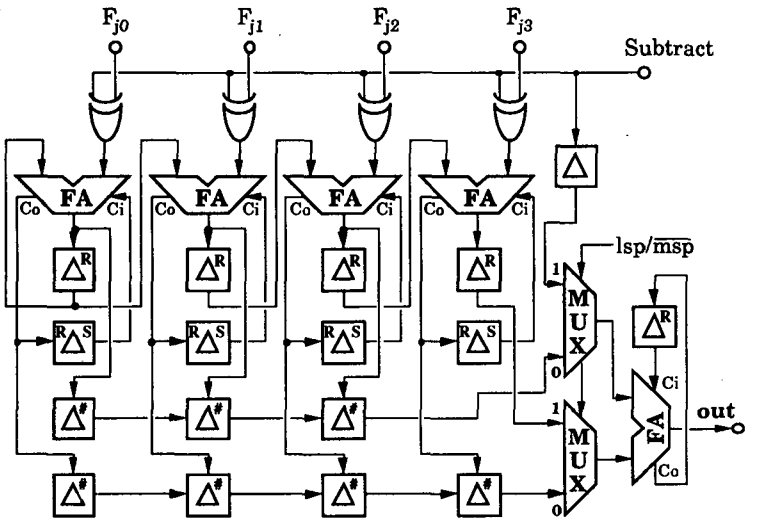



***Figure 6.7*** *Modified CSA shift-accumulator with pipeline registers*

The choice how to implement the adders can also affect the speed [West85]. Dynamic logic leads to simpler configurations and usually to faster solutions than static logic. Because of the critical carry-ripple-chain of a CPA, good care should be taken when choosing the logic. A feasible strategy is to employ dynamic logic for the carry-save adder, and the more robust static logic for the carry-propagate adder. The CSA will then be useful for high speed applications, and the slower CPA can be used in applications where a simpler solution is preferable. Another argument for this strategy is that dynamic logic cannot be used below a certain clock-frequency due to discharging of the storage nodes.

The choice in this methodology has fallen on the CSA due to the throughput and due to a purely bit-serial system was desired. For comparison a CPA solution has also been evaluated in a test integration.

### 6.3.3. Full-Adder

The central and most complex part of the shift-accumulator is the full-adder. The FA should be as compact, regular and fast as possible. Many different structures for full-adders exist [Mead80, West85, Reus83]. A straightforward CMOS implementation is to use normal logic gates such as XOR, NAND, NOR, INVERTERS, etc.

The combinational full-adder in figure 6.8a is not very well suited for CMOS implementation since it contains non-inverting gates. Then it is better to make use of the possibilities of CMOS design. This leads to the CMOS full-adder in figure 6.8b, that requires 28 transistors. Unfortunately it results in a rather irregular structure.

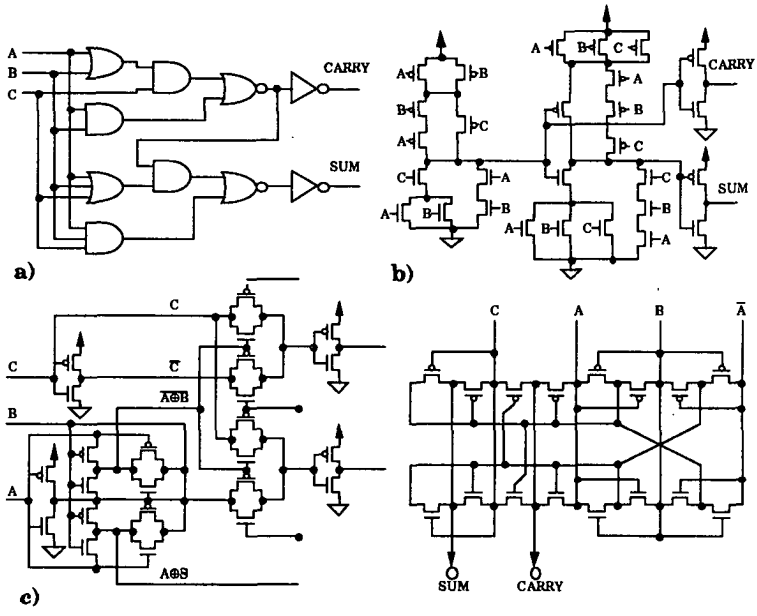



***Figure 6.8*** *a) Combinational full-adder schematic*
*b) CMOS full-adder transistor diagram*
*c) Transmission-Gate full-adder transistor diagram*
*d) Pass-Transistor full-adder transistor diagram*

A better solution can be found using the so-called transmission gate full-adder [West85]. This adder is fast and the number of transistors reduced to

24, figure 6.8c. However, this structure is still not very regular and it requires a high amount of wiring.

A more interesting structure is found in [Reus83]. This is based on pass-transistor logic. It contains only 16 transistors and it is extremely regular, figure 6.8d. The speed of this full-adder is relatively high. It corresponds to only two gate delays according to Reusens. Since this FA does not contain any driven gates, it needs input and output buffers of either static or dynamic type. It should be noted that an additional inverter is needed to generate $\bar{A}$ and that two more inverters should be included at the outputs to give the structure driving capability to compare it to the other FAs. Still in that case only 22 transistors are required and due to the regularity of this FA, i.e., the complete symmetry between n- and p-devices, a considerable smaller area is required. In addition, the full-adder require no power supply wires and this simplifies the layout considerable. Different configurations having inverted inputs and/or inverted outputs can be found by slightly modifying the adder [Anso86].

A suitable FA realization for the CPA, on the other hand, should first of all have a minimum carry-propagate delay. Reusens FA would not be very well adapted because it is basically a non-regenerating device. It could however be modified for a CPA configuration but still other better adopted full-adder structures can be found [West85].

### 6.3.4. Registers

Registers play an important role in implementations of DSP algorithms, especially using bit-serial arithmetic. It is therefore important to have a good strategy for their realization. It has been found that using the non-overlapping two-phase clock and a dynamic register-cell a good speed-area trade-off is obtained. The cell can be implemented using only 8 transistors, figure 6.9a. Using the tri-state inverter in figure 6.9b saves one connection between each p- and n-diffusion.

Dynamic registers do not suffer from charge redistribution problem if correctly configured, but still the information will get lost due to discharging if the register is not clocked at a sufficient high speed. However, experience has indicated that the registers can be clocked at a remarkable low speed without these problems, i.e., in the range of 500 Hz to 1 kHz. Furthermore, in an efficient bit-serial implementation of digital filters, almost all registers will be clocked at nearly maximum speed.

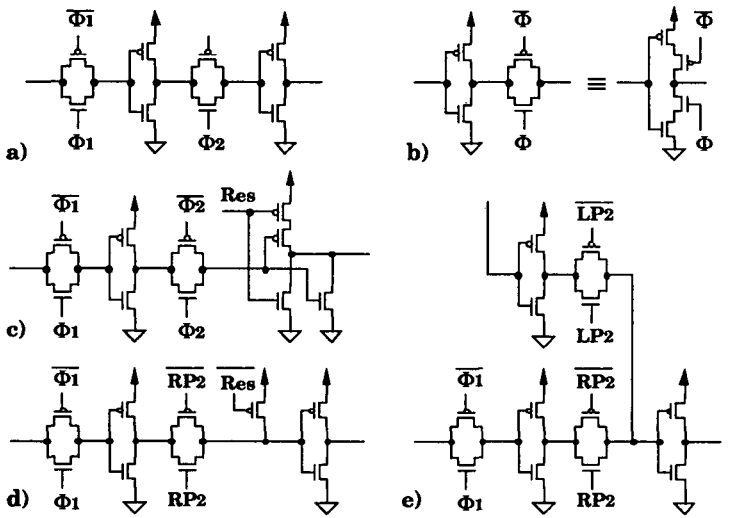



***Figure 6.9*** *Dynamic register cells realized by;*
***a)*** *two inverters and transmission gates*
***b)*** *tri-state inverter*
***c)*** *register with reset*
***d)*** *simplified reset*
***e)*** *parallel load register*

For all the implementations a strategy using dynamic registers and static or pass-transistor logic has been used.

As discussed before it is necessary to have registers that can be set or reset. Reset can be obtained by replacing the last inverter in figure 6.9a with a NOR gate, figure 6.9c. However, this has been found to be to slow in some cases. The solution that results in lowest complexity is the one proposed in figure 6.9d where a single pull-up p-transistor is used for the reset. It is necessary to use a masked $\phi_2$ clock *RP2*, figure 6.3. A set function is obtained instead by simply replacing the p-transistor by a pull-down n-channel transistor and using the complementary control signal.

A straightforward way to implement the parallel-to-serial conversion registers is to use multiplexors. However a much simpler solution to obtain this function is to let the control-signals control the transmission gates, figure 6.9e. Then the dual masked $\phi_2$ clocks, *LP2* and *RP2* in figure 6.3, are used to obtain the desired function.

### 6.3.5. Output Selection Logic

To combine the least significant and the most significant part of the result and to add the contents of the sum and carry pipelining registers some additional hardware is required.

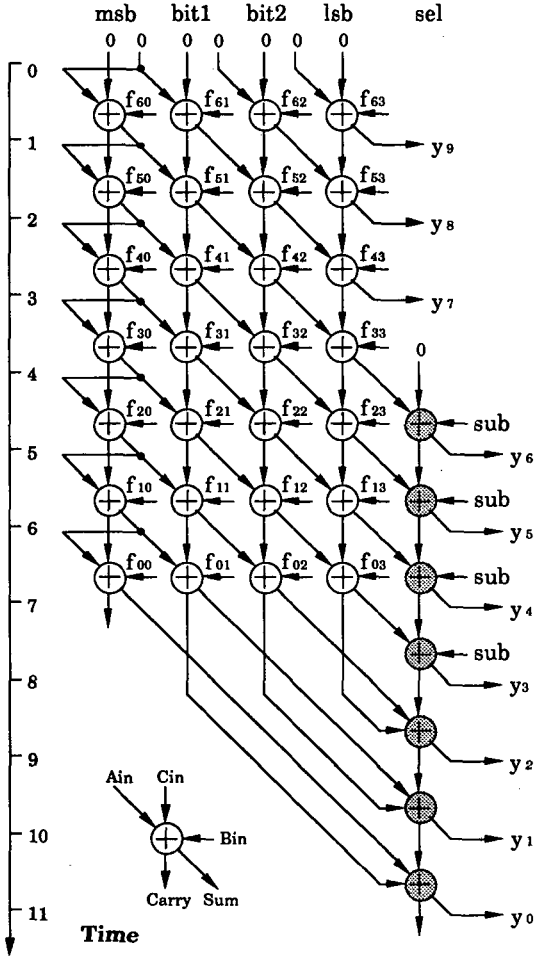



***Figure 6.10*** *Time-Space diagram for a four-bit accumulator with seven bit data.*

In figure 6.10 a complete *time-space graph* for a shift-accumulator is shown. The graph shows a four bit SA ($w_c = 4$) and for calculation of seven bit data ($w_d = 7$). $F_{ij}$ in the graph corresponds to the i:th bit of the j:th look-up entry for the innerproduct. The four SA full-adders are used in seven consecutive clock periods for the basic calculation. By using an extra full-adder (shaded in the figure) in the following four cycles ($w_d - w_c$) the complete result can be formed. This adder can be fully utilized in case of offset binary coding when a one will be added to the lsb in case of a subtraction. The calculation for two consecutive samples overlaps $w_c$ clock-cycles in this configuration.

Naturally between each time-step, vertical in this graph, a register is required. In the four last steps these registers correspond exactly to the pipelining registers.

A schematic of the final version of the shift-accumulator is shown in figure 6.11. Some simplifications have been done to further reduce the hardware. For instance the carry of the msb is never used so the register cell has been deleted.

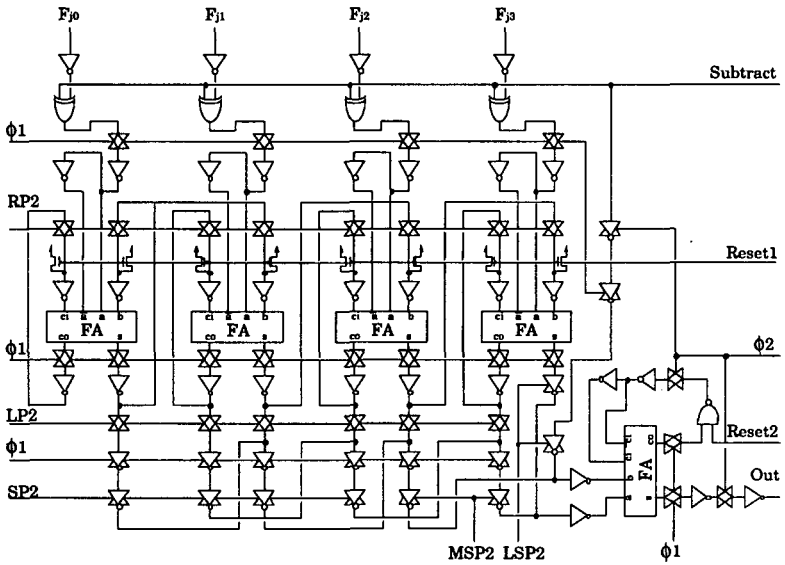



***Figure 6.11*** *Schematic of a complete four-bit shift-accumulator for normally coded coefficients.*

The SA can be constructed using four different bit-slices. A simplified msb-slice that abuts directly to a clock- and control-signal cell. Then follow $w_c$-3 normal bit-slices. The second least significant bit has a modified carry register (another control-signal is used). Finally, there is the lsb-slice that together with the selection logic makes up the unit to form the result. The number of different bit-slices increases to eight in case of OBC when each carry bit should be either set to one or zero, an n- or a p-transistor is required for this.

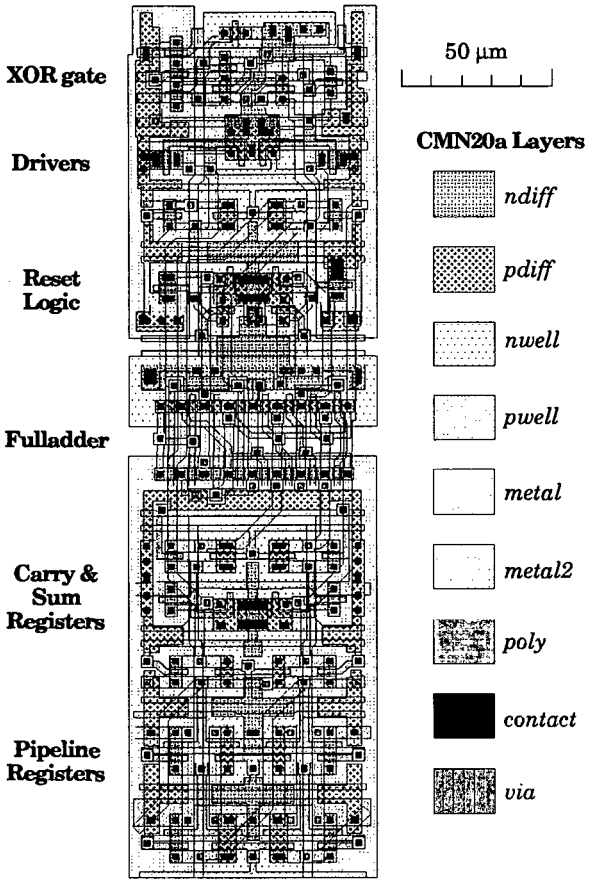



***Figure 6.12*** *Layout of shift-accumulator normal bit-slice with reset*

A detailed layout of a complete bit-slice is presented in figure 6.12. The placement of the different blocks corresponds directly to the schematic in the figure above. A layout-style using polysilicon mainly in the horizontal direction passing through vertical strips of n- and p-wells, has been used. The first metal layer is used mainly in horizontal direction to interconnect different clock- and control-signals, while metal 2 is used in vertical direction to connect internal signals. A minimum of diffusion and polysilicon have also been the goal of the layout-style in order to achieve a maximal speed.

It can be noted that alternative realizations for the registers have been used. In some places the registers have been built up using inverters and transmission gates, while in other places tri-state inverters have been used. The only reason for this is layout considerations. The layout is the result of an iterative design and simulation process where parts of the layout have been changed and optimized to improve the performance. The very compact part in the center of the layout corresponds to the pass-transistor full-adder discussed earlier.

In total the bit-slice cell contains 66 transistors and the size of the abutment box is 70.0 μm wide, and 281.5 μm high, resulting in an area of 0.019705 $mm^2$ and a density of 3349 MOS/$mm^2$. SPICE simulations have shown that this unit can be clocked at higher speeds than 100 MHz. Lack of appropriate test-equipment and problems to measure a separate bit-slice has prevented this from being verified by measurement.

## 6.4. The ROM

### 6.4.1. Constraints and Options

The look-up-table is implemented using a *Read Only Memory* (ROM) where the pre-calculated partial bit sums will be stored. The size is dependent on two factors. First, the needed coefficient wordlength (or the wordlength of the partial sums) denoted $w_c$ here, secondly of the number of terms in the innerproduct to be calculated with the actual distributed arithmetic processor.

Typically ROM-coefficient wordlengths of 5-20 bits are usually required in most DSP applications. To vary $w_c$ is simple, it is only to use additional bit-slices of both the arithmetic part and the ROM. However, some physical constraints limit the maximum number of bits. The word-lines in the ROM are proportional to the coefficient wordlength. A large wordlength will thus slow down the whole architecture. The SA itself does not have this

constraint because it is of carry-save type and no signals are propagated through the structure except for the control-signals, but these can be driven by sufficiently sized drivers.

The required number of words in the ROM can simply be calculated as two to the power of the number of terms in the innerproduct. For OBC this number is divided by two. If a greater look-up-table than 1k - 2k words is used, the surface of the ROM will be totally overwhelming and the architecture becomes less area efficient. However, the methods described in the Chapter 5 can be used to circumvent the problem in some cases.

### 6.4.2. ROM Structure

The basic structure for the ROM is a usual array of single CMOS transistors. A "one" is coded by the presence of a transistor, while a "zero" is symbolized with no transistor, figure 6.13. A parallel structure, i.e., a NOR structure, with only n-channel transistors has been chosen since this solution is faster.

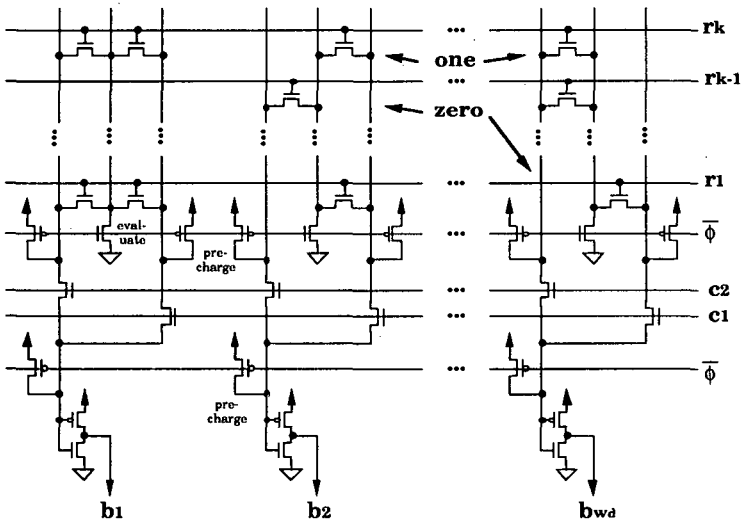



***Figure 6.13*** *Basic ROM structure with pre-charge logic*

To obtain a single transistor cell, some precharging scheme must be employed. In fact all outputs are precharged to a high level and if the accessed bit in the ROM contains a transistor this device will discharge the

output node and a low level value is obtained. To fit into the overall architecture, the ROM should directly abut to the shift-accumulator. It must thus be designed with exactly same pitch as the rest of the DA-processor. The width of a shift-accumulator bit-slice is equal to 70 μm and multiple ROM cells can be implemented on this pitch. A configuration using a column decoder is used for this reason. The block diagram for a ROM with a two to one demultiplexor is shown in figure 6.13.

The inputs are split into two groups, first the *row address*, that forms the normal addresses for the row decoder. A second group, the *column address*, is used to form an address for the column decoders.

An eight bit-lines per accumulator-bit configuration has shown to be convenient in the used technology. A layout of a cell containing 2x8 ROM cells is shown in figure 6.14. Metal is used both for word-lines and bit-lines.

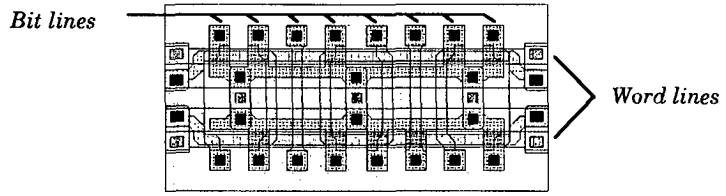



***Figure 6.14*** *Layout of a basic cell containing 2x8 ROM cells*

The abutment size of this cell is 70 μm wide and 24 μm high. A fairly dense ROM matrix can thus be obtained, in fact, the required pitch per word will be as low as 24 μm /16 = 1.5 μm / word. A 1k word ROM will require 150 + 30 = 180 μm which is still less than the height of the SA. The obtained density of the ROM is 8465 bit/mm$^2$ that is fairly good considering that there were constraints on a fixed width for the ROM cell.

### 6.4.3. ROM Decoder

Fully complementary logic with NAND-structure for the n-transistor switching-net and an OR-structure for the p-network could be used. It is usually more economical to use a precharged configuration then one of the two complementary networks can be avoided. N-transistors are the better choice from a speed point of view.

To further increase the speed it is advantageous to use a parallel NOR-structure rather than the serial NAND-structure. The problem is that the output must be inverted twice to obtain the desired function where only one

of the word-lines should be selected. To increase the speed, pipelining of a half cycle is used for the word-line buffers. Figure 6.15 shows a complete three-to-eight decoder with both input and output buffers.

The same structure is used both for row and column decoding. The column-decoder however is of fixed size, three-to-eight that allows eight columns per accumulator bit.

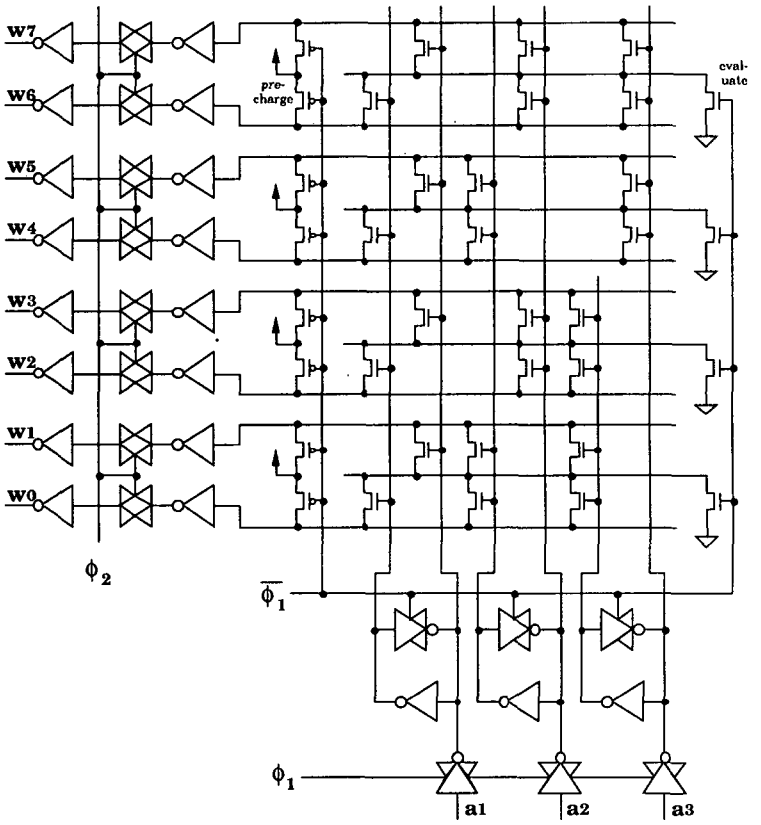



*Figure 6.15 ROM decoder*

### 6.4.4. ROM Decoder Adapted to Offset Binary Coding

In the case of offset binary coding of the look-up-table, the ROM is reduced to half the number of words as described in Chapter 5. The decoding

is obtained by using one of the inputs to control the sign. This input is then used to decide whether or not the other inputs should be complemented (bit-wise inverted). This can be done by using an XOR gate before the decoder. An even simpler implementation is achieved by integrating this function into the decoder as suggested in figure 6.16.

It is relatively easy to verify that the functionality of the decoder proposed in the figure is identical to taking the XOR of the inputs $a_2$ and $a_3$ with $a_1$ and then using a normal two-to-four decoder. An additional feature of this solution is that basically the same decoder and drivers can be used it is only to add the special cell for the control input. For the column decoder, on the other hand, a conventional solution has been used strictly because of layout reasons. The size of the column decoder is constrained in both vertical and horizontal dimensions and can not be modified. This can be seen in figure 6.2 and 6.4.

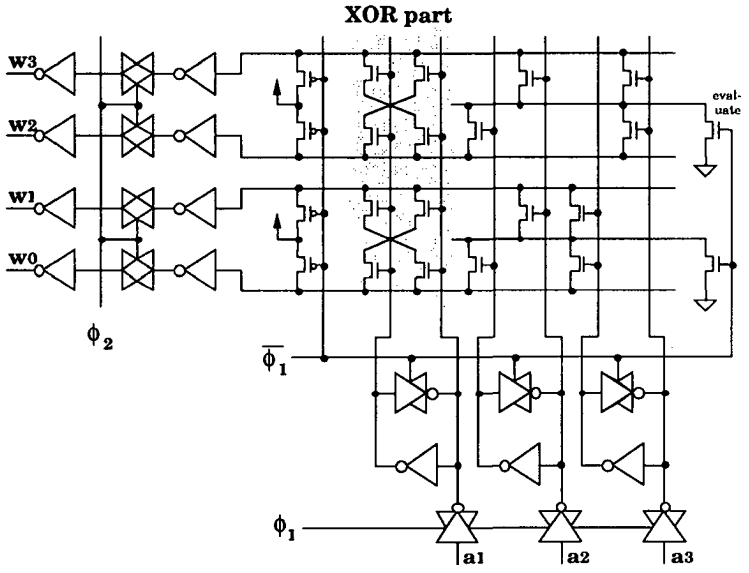



***Figure 6.16*** *ROM decoder for OBC*

## 6.5. Quantization and Saturation Unit

To simplify the arithmetic units, two's-complement representation is used to represent the signals. However, from a DSP point of view, or more

specifically given by the WDF other characteristics than two's complement are required. A unit performing sign-magnitude truncation and saturation characteristic on overflow has been developed. This is the *Quantization and Saturation Unit* (QSU).

### 6.5.1. Overflow Detection

To implement the saturation characteristic one needs to detect and handle eventual overflows. This can be done by comparing the carries of the two most significant bits after the final addition or subtraction. However, in a carry-save adder the carries are generated only from partial results. Thus, the decision whether or not an overflow has occurred cannot be done before the complete result is calculated. Using an auxiliary sign-bit, a so-called *guard-bit,* is another possibility. Obviously if the sign-bit and the guard-bit differ, an overflow has occurred. Nevertheless, independently of method, the drawback is that an extra bit should be handled. This results in an extra clock-cycle or in an additional interconnection path. Since only bit-serial blocks are used the method using the guard-bit has been adopted here.

It should be noted that using this guard-bit, which is forcibly necessary if a quantization and saturation unit is required, the wordlength will increase to $w_d + 1$ bit. The guard-bit can be dropped immediately after it has been used in the QSU. However, the cycle-time will anyway be increased to $w_d + 1$ clock-cycles. Using a RAM for storing the signals, this guard-bit is not stored while it is a copy of the sign-bit. It is regenerated by copying the sign-bit while reading the RAM. The data wordlength $w_d$ will here after refer to the extended wordlength if not otherwise stated.

### 6.5.2. Realization

| Input | | | | | | Computed | | | | | | Output | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| g | s | 1 | 2 | ... | q | g | s | 1 | 2 | ... | q | g | s | 1 | 2 | ... | q |
| 0 | 0 | $x_1$ | $x_2$ | ... | $x_q$ | 0 | 0 | $x_1$ | $x_2$ | ... | $x_q+x_g2^{-q}$ | 0 | 0 | $x_1$ | $x_2$ | ... | $x_q$ |
| 0 | 1 | $x_1$ | $x_2$ | ... | $x_q$ | 0 | 0 | 1 | 1 | ... | $1+x_g2^{-q}$ | 0 | 0 | 1 | 1 | ... | 1 |
| 1 | 0 | $x_1$ | $x_2$ | ... | $x_q$ | 1 | 0 | 1 | 1 | ... | $1+x_g2^{-q}$ | 1 | 1 | 0 | 0 | ... | 0 |
| 1 | 1 | $x_1$ | $x_2$ | ... | $x_q$ | 1 | 1 | $x_1$ | $x_2$ | ... | $x_q+x_g2^{-q}$ | 1 | 1 | $x_1$ | $x_2$ | ... | $x_q+x_g2^{-q}$ |

***Table 6.1*** *Quantization and saturation characteristics*

To obtain sign-magnitude characteristics from a two's complement truncated value a simple algorithm can be used. It is necessary to know the sign, or more precisely, the correct sign of the signal. Here the guard-bit is used. The value of this guard-bit is added to the lsb position of the signal.

To correct an overflow, i.e., to generate the saturated value of the signal, all fractional bits are set to one, and the sign-bit to zero. By adding the guard-bit to the lsb the correct function is obtained.

Clearly these two tasks can be joined to a simple algorithm [Sjös86, Sjös88]. Both require that the correct sign-bit, or the guard-bit, is added to lsb. Table 6.1 shows how these two tasks are merged for the four possible different cases. The indices g and s correspond to the guard-bit respectively the sign-bit, while index 1 to q are the fractional bit of the signal, q being the lsb. The four cases that appear are

**(g,s) = (0,0)** that means a positive number without overflow, no correction is made.

**(g,s) = (0,1)** this is a positive overflow. Bits 1 to q are set to 1 while the sign is set to zero.

**(g,s) = (1,0)** this is a negative overflow. Bits 1 to q are set to 1 while the sign is set to zero, and 1 is added to lsb.

**(g,s) = (1,1)** this is a negative number without overflow, 1 is added to lsb.

To implement the characteristic of table 6.1, an XOR gate is needed for detecting overflow. A *Half-Adder* (HA) is sufficient for adding the guard-bit to the least significant position. Additional logic in form of an AND gate is used in order to generate the *set/reset* signals. The complete quantization logic can then be realized as shown in figure 6.17.

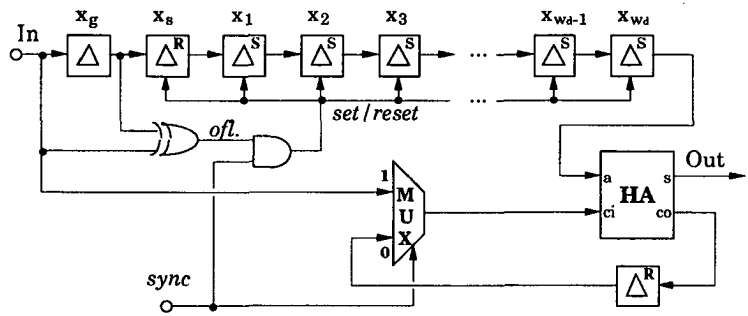



***Figure 6.17*** *Quantization logic with saturation characteristics*

The exact numerical behavior of this unit is slightly modified compared to the sign-magnitude characteristic, figure 6.18.

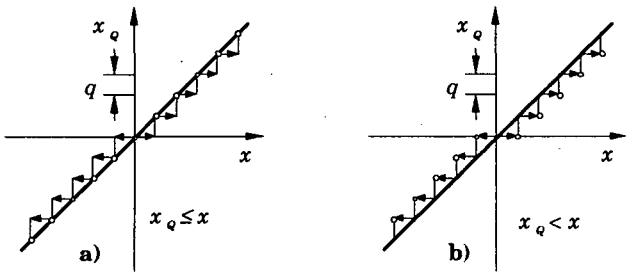



***Figure 6.18*** *a) Normal sign-magnitude characteristic*
*b) Modified sign-magnitude characteristic*

The difference appears at the discrete points where the precisely represented signal is equal to a value that can be represented by the binary representation. With the realized quantization unit here, the quantized value will be one quantization step lower. Fortunately this is not any significant problem. For WDFs this modified sign-magnitude characteristic is sufficient to preserve pseudo-passivity even if a slightly larger error will occur every now and then. Using the normal statistical noise model no distinction can be made between the two models in figure 6.18.

### 6.5.3. CMOS Realization

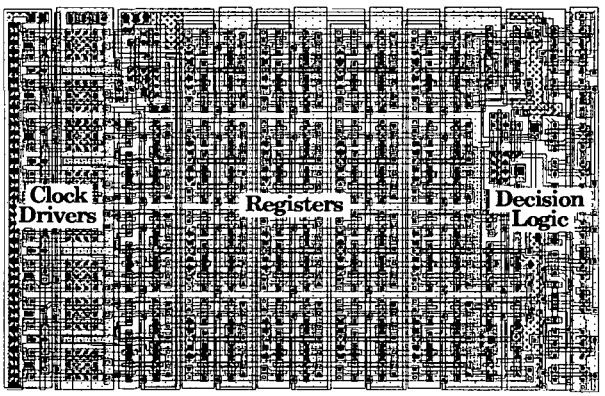



***Figure 6.19*** *Layout of a 16+1 bit quantization unit*

The quantization unit has been implemented accordingly to the schematic in figure 6.17. Certain things had to be changed to adapt this unit to the overall design strategy. To allow a maximal speed pipelining has been introduced. The overflow detection and the setting of the carry are made in separate cycles. A complete unit for 16 + 1 bit data is shown in figure 6.19.

To avoid that the unit would grow and be very long and narrow for increased wordlengths a layout strategy with folded register has been used. The register is folded modulo 4 that has shown to be convenient for the current implementation.

## 6.6. Storage Unit

### 6.6.1. Storage Elements

Accordingly to the previous chapter about the architecture, the storage unit can be based on two different types of storage elements. The SE can be realized using shift registers or RAM, figure 6.20a. This solution will require two identical conversion register banks $CRB_A$ and $CRB_B$. The register based structure uses the rotate registers that have an internal feedback to store signal values over multiple cycles, figure 6.20b. Both these units will be described in the following two sections.

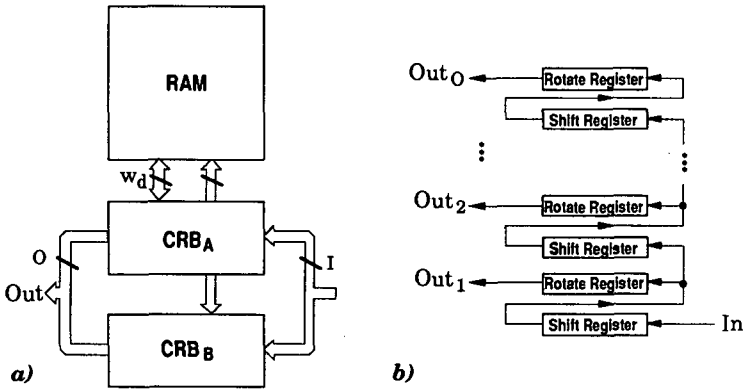



*Figure 6.20* *a) RAM based storage element*
*b) Register based storage element*

The storage unit, as it is defined, can contain multiple storage elements, and in principle both types of SEs can be mixed. However, in the normal case only one of the two types is used.

## 6.7. RAM Based Storage Element

In contrast to general purpose *Random Access Memories* (RAM) the memory required here is not really of random access type. Basically it works in a predefined sequence where all addresses will be accessed periodically. The memory should rather be named *periodical accessed memory*. It is no longer necessary that the decoding and the read or the write is made in the same phase. Therefore pipelining can be introduced to speed up the memory.

### 6.7.1. The Cell

A dynamic RAM can be used since the address sequence is periodical and relatively short. This is the direct result of that the treated DSP algorithms are periodically repetitive. RAMs are usually implemented using either one, three, or four transistors per memory cell [West85]. One transistor cell is more or less exclusively used by semiconductor companies specialized on manufacturing memories. It is not advisable to try to implement this kind of cell in a normal CMOS technology, at least not without a profound knowledge of the technology parameters. One is left with the choice of three or four transistors per cell.

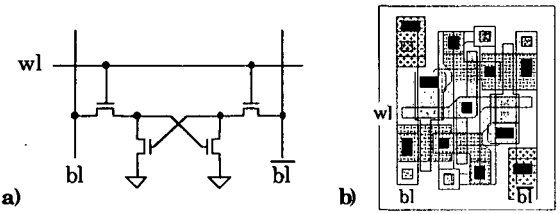



***Figure 6.21*** *Four transistor dynamic RAM cell, transistor diagram and layout.*

The four transistor RAM cell in figure 6.21a has been chosen here for different reasons. The main advantage of this is the completely symmetric structure. This symmetry can be used when designing the layout of the cell, resulting in a circuit that is both compact and not too vulnerable to errors in the manufacturing. The symmetry makes it possible to use differential amplifiers everywhere. As a result, the two *bit-lines* (BL) are complementary and they will be shared both for reading and writing. The cell is also self-refreshing, i.e., each access to a cell will restore the data.

A very compact layout of the basic cell was designed using one metal layer for the *word-lines* (WLs) and the other for the BLs, figure 6.21b. This is

important when a high speed solution is desired. The obtained cell has the measures 24 μm x 23.5 μm that gives a RAM core density of 1.78 kBit / mm$^2$.

### 6.7.2. Pipelining

Another result of the periodic address sequence is the possibility to use a completely pipelined memory. A block-diagram is shown in figure 6.22 where pipelining latches are represented by a shaded bar. Using the pipelining, all different parts of the RAM are decoupled. A much higher clock-rate can then be used.

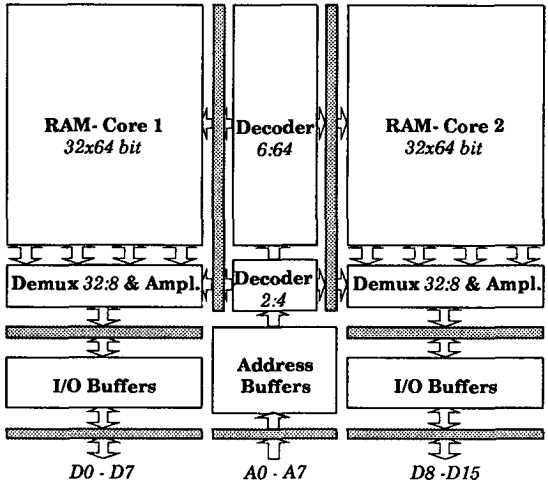



***Figure 6.22*** *Block diagram of pipelined 256 x 16 bit RAM.*

The RAM has been divided into two core-areas according to figure 6.22. Row and column decoding are also used. A fairly squared outline of the RAM is obtained this way, figure 6.23. The length of word-lines and bit-lines are also minimized this way.

A complete layout of a small RAM is shown in figure 6.23. The size of this RAM corresponds to 16 words of 4 bit. This is not a realistic size because of the overhead of I/O circuitry.

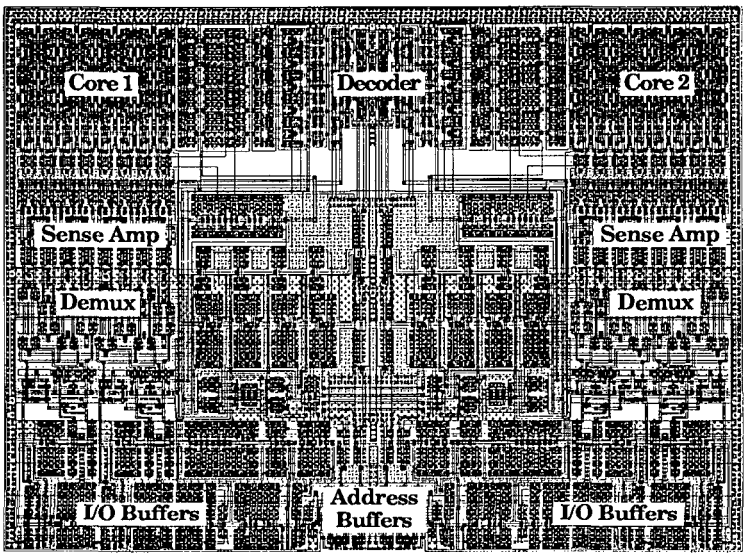



*Figure 6.23 Layout of a 16 x 4 bit RAM.*

### 6.7.3. Serial Parallel Conversion by CRBs

Since a bit-serial architecture is used and the RAM preferably stores data in parallel form, a conversion between these formats is needed. As suggested before this problem is solved using two *Conversion Register Banks* (CRB) for each RAM.

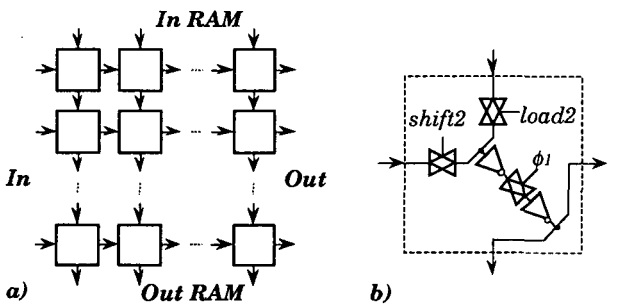



*Figure 6.24 Transistor diagram of conversion register bank.*

The conversion registers are in principle a four-port bi-directional shift-

register, figure 6.24a. Shift can be made in the horizontal or in the vertical direction. Vertical shift can be viewed as a parallel load.

Simple dynamic shift-register cells with a multiplexed input are used for the realization. The width of the RAM port corresponds to the signal wordlength $w_d$. The number of bits in the I/O port depends on the number of variables stored in the RAM. A schematic diagram of the register cell is shown in figure 6.24b.

Two different clocks are used, *load2* and *shift2*. These are complementary phase 2 clock-signals, when one clock is active the other one is not and vice versa. If a longer period is required during load, the $\phi_1$ clock should also be replaced by a clock having longer periods during load.

## 6.8. Register Based Storage Element

The shift-registers are all implemented using the dynamic cell shown in figure 6.9a and 6.9c. On the layout level the shift-register cells have been designed such that the structure can be folded in case of a longer register.

### 6.8.1. Rotate Register

If a variable is needed in more than one time-slot, i.e., more than one period of $w_c$ clock-cycles, a repeat function or a memory function must be incorporated into the shift-register. The simplest solution is to let the data rotate in the shift-register, i.e., by feeding back the output to the input. This corresponds to the *rotate register* shown in figure 6.20. The input must be switched by a multiplexor to be able to feed new data into the register. The configuration of figure 6.25 implements this function.

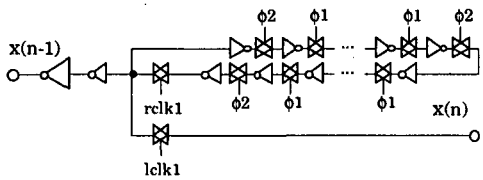



***Figure 6.25*** *Transistor diagram for register with feedback.*

The two control-signals *lclk1* and *rclk1* are complementary so the old data is either fed back (rotated) or new data are clocked in (loaded). The register module has been designed such that many registers can be stacked on the top of each other and clock- and control-signals can be shared.

### 6.8.2. Register Bank

By combining the rotating register with normal shift-registers the complete SE is formed. The layout of this unit has been made such that it is possible to increase $w_d$ simply by adding a new slice of the whole unit. This gives a highly regular structure. The only problem occurs in a multi-channel solution when the register lengths are different multiples of $w_d$. Then it might be necessary to use dummy wire-through cells in some places.

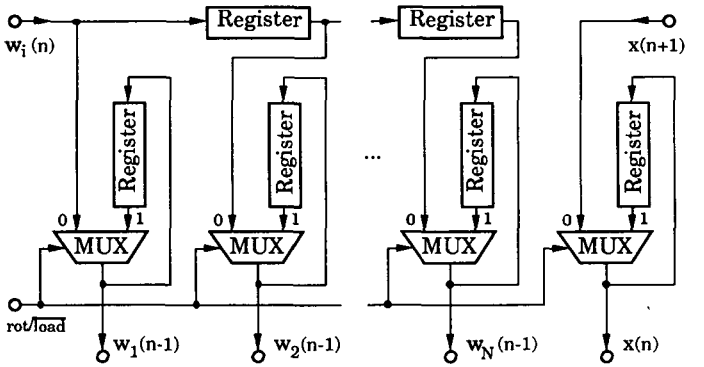



***Figure 6.26*** *Shift-register bank for single processor architectures.*

This is efficient for the realization of a storage unit for a single processor implementation or more generally when $N_{PE} < N_{IP}$. Each new value of the state variables is calculated in a sequence. They are fed into $N_{IP}$ shift-registers of $w_d$ bits and at the same moment all new values are loaded into the rotating registers once the previous sample period is terminated. The necessary configuration is viewed in figure 6.26.

All registers are $w_d$ bits long, but in case of multi-channel processing with $N_C$ channels the feed-in registers at the top of figure 6.26 are extended to $N_C$ $w_d$ bits. The advantage with this configuration of registers is the large number of variations and the regularity and modularity.

## 6.9. Control Unit

The presented methodology is restricted to fixed algorithms and thus the controlling of the architecture is relatively simple. All computations are known a priori, they are not data dependent, and they will all be periodical. Thus, the *Control Unit* (CU) can be based on a *Finite-State Machine* (FSM).

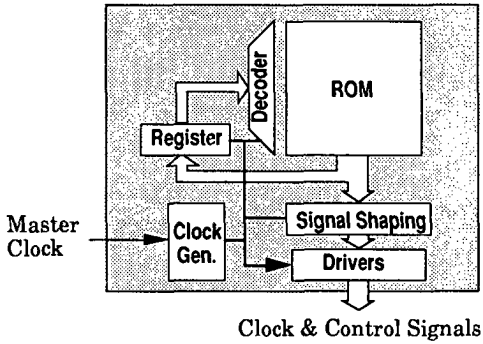



***Figure 6.27*** *Control Unit*

### 6.9.1. Finite-State Machine

A finite-state machine can be implemented in different ways, but one regular and simple way is to use a ROM, figure 6.27. The ROM will contain *micro-instructions* containing two fields, the next address and the control-signals. The address field is latched and used for generating the address for the succeeding clock-cycle. The value of the control-signal field is used to determine the different control-signals in the next cycle. They are input to the signal-shaping-logic that generates the wanted pulse-forms. Finally, large global control-signal drivers are used to distribute these signals to different parts of the chip. These are designed using staged drivers.

A clock-generator generating the non-overlapping two-phase clock from a single-phase master clock can also be included in the unit. Internal versions of these clocks are used in the CU. These clocks are buffered the same way as the rest of the signals before they are distributed on to the chip. It is important that the gate depth for all signals is exactly the same otherwise skewing between clock-signals and control-signals could appear [Mead80].

### 6.9.2. Sequencing

The signals to be generated can be divided into different types, depending on their periodicity, their shape and their purpose. Here follows a short description of signals needed for this architecture:

- **Clock signals**, these are the basic clock signals with the shortest period. A non-overlapping two-phase clock scheme is used in the architecture. It means that two clocks are needed $\phi_1$ and $\phi_2$, and the four different phases $\phi_1$, $\overline{\phi}_1$, $\phi_2$ and $\overline{\phi}_2$ are normally generated locally.

The distribution of these signals is very important, they should not be skewed to each other. A strategy where all clock- and control-signals are locally buffered has been used.

- **Word synchronization signals**, these signals are used to align and synchronize the data words. The periodicity is always $w_d$ clock cycles which also can be called one macro-cycle.

- **Channel synchronization signals**, here the period is a multiple of the word synchronization signal. The signal is used to switch from processing one channel to another in a multi-channel system. For a single channel application, word and channel synchronization are equivalent.

- **Sample synchronization signals**, are multiples of the channel synchronization signals. They are used for events that are synchronous to the sampling period. Of course, if a single-channel and single-rate system with one processor per innerproduct is considered, one cannot distinguish between these signals and the word synchronization signals.

- **Masked clock signals,** these are shaped exactly as the basic clock signals with the exception that one or more pulses are masked off each period, where the period can be one of the three earlier described. These kinds of signals are useful where ever multiplexing or switching is needed. Two main classes can be defined, signals shaped as $\phi_1$ or $\phi_2$ clock signals.

- **Pulsed signals,** these are shaped as normal data signals. The pulse width corresponds to a multiple of a clock cycle. The signal changes value either synchronously to $\phi_1$ or to $\phi_2$.

- **Special shaped signals,** these include for instance, signals that rise synchronously to one clock and fall synchronously to the other clock phase. Sometimes it is also necessary with signals that have a controlled slope or delay, e.g., control signals for the RAM sense-amplifiers. All these signals should be locally generated in the unit where they are used.

### 6.9.3. Shaping

Only two types of pulse shaped global signals are allowed, the masked clock pulses and the pulsed signals. Two further variations are needed to

make these pulses synchronous to either $\phi_1$ or to $\phi_2$. Four simple units are used for the shaping, they are shown in figure 6.28 with the corresponding pulse-form they generate ( *A, B, C, D* ). It is assumed that the input *S* is synchronous to $\phi_1$.

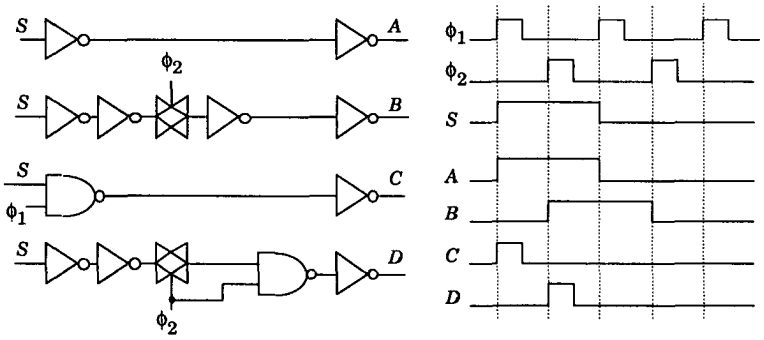



***Figure 6.28*** *Pulse shaping logic*

### 6.9.4. Drivers

To avoid skewing between clock- or control-signals the buffering of these signals is important. The overall strategy used is that each control-signal is locally buffered. This reduces the difference in load capacitance between different blocks as well as the overall load. The local buffers can be controlled and sized accordingly to the needs.

A worst-case strategy is used for the transistor-sizing of the global drivers. The total capacitive loads for all control-signals, except for the clocks that are treated separately, are calculated. The drivers are then sized sufficiently large for the signal with the greatest load. A staged design has shown to give the shortest rise- and fall-times for the signals. By using exactly the same number of stages for all signals a minimum of skewing between different control-signals is introduced.

# chapter 7. FRAMEWORK

*In this chapter the framework for the design and implementation of fixed DSP algorithms is presented. The methodology is summarized, some developed CAD tools and algorithms are discussed. Together with some already existing programs and tools these form a vertically sliced synthesis environment that supports the presented methodology.*

## 7.1. CAD Environment

### 7.1.1. CAD Outline

The synthesis system and its CAD environment presented here should not be viewed as a complete product or compared with any commercial system. Instead some of the CAD tools are very experimental. They have only been developed to support the presented approach for ASIC realizations of DSP algorithms [Sjös89a]. Especially the interface between different design steps and between some of the tools is far from being optimal. The intention behind the development has not been to create a commercial product, but to verify that the used strategies are correct and to show how parts of a design system can be realized.

The framework can serve as a special purpose *tool-box* for the designer where different specially tailored tools can be found for each phase of the design and implementation task. The emphasis is put into the special purpose tools and algorithms needed. These have been mainly developed for state-space transformation, optimization and for architecture synthesis. The overall CAD framework is shown in figure 7.1. The darker shaded tasks are the parts where the main effort has been invested.

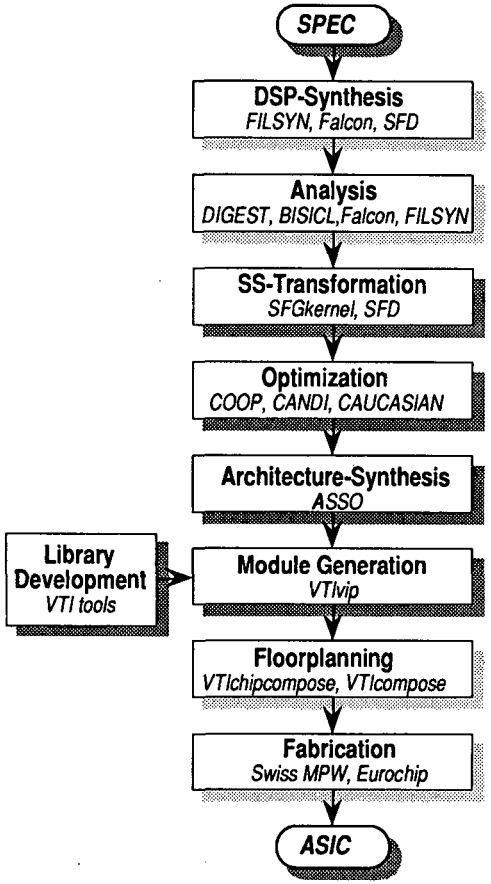



***Figure 7.1*** *Overview of the framework*

### 7.1.2. DSP Domain Tools

This part of the framework can essentially be solved using already existing tools according to figure 7.2. A clear distinction should be made between WDF synthesis and realization, to the left of the figure, and the normal DSP algorithms, to the right. WDFs are of course designed using the analogy to the classical analog filter domain. They will thus require different tools and methods compared to other digital filters and DSP algorithms.

For the synthesis of a digital filter, or a DSP algorithm, any tool is convenient in this framework as long as specifications can be given as an input and it produce a description of a complete DSP algorithm, preferably in form of a signal flow graph. The problem, however, is that most tools have their own interfaces. Some tools accept specifications described by a language, other specifications in a parameter-file and some interactively. The output can be generated in form of another language, a netlist or simply coefficient values for a given class of algorithms with given structure.

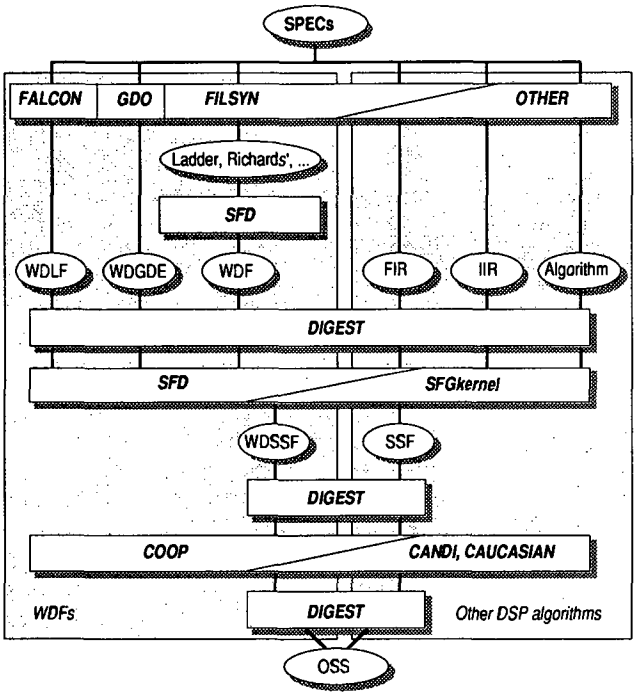



***Figure 7.2*** *DSP synthesis system*

As indicated in figure 7.2, FILSYN [Szen77] is used as the basic platform for the synthesis step. This software is a fairly complete package including tools for the synthesis of many types of filters. Directform IIR digital filters can be realized and composed in either cascade or parallel decompositions. FIR filters can be synthesized using both window and equiripple design. They can furthermore be decomposed into cascade form or be realized directly in transversal form. Group delay equalizers can also be synthesized

for different filter classes. Other features such as transformation, simulation and analysis of filters are as well included in FILSYN. For the design of WDF, FILSYN offers a fairly complete solution. It supports synthesis of many different classical filters, including *micro-wave filters, lumped element filters, Richards' structures*, etc. Some basic manipulations and transformations of an obtained filter can also be made.

The tool FALCON [Gazs86] is used to synthesize and realize wave digital lattice filters [Fett74]. The obtained WDLFs consist of two branches, each with cascaded first and second order allpass sections [Fett74, Gazs85]. FALCON supports design of cascaded subsystems and mixture of WDFs and FIRs. Another useful feature is that output in form of a netlist for the DIGEST tools [Clae84] can be generated.

A special tool, called *Group Delay Optimizer* (GDO) has been developed to equalize the group delay for any filter by cascading it with a *wave digital group delay equalizer* (WDGDE). The WDGDEs are realized the same way as one of the branches of the WDLF. The tool does not use an approximation algorithm. Instead a random optimization based on simulated annealing [Kirk83] is used to obtain a minimum overall ripple in the delay. Coefficient wordlengths are also directly optimized. The GDO program has, for example, been used successfully to realize group delay equalizers for a very strict requirement for a digital audio system [Yazd90b].

Analysis and simulations are important verification procedures on all levels, not least in the DSP domain. In fact, these tasks can be performed at each step of figure 7.1. Characterization and tuning are other related procedures that often can be made using the same tools. To accomplish all this the *DIGital filter Evaluation and Simulation Tool* (DIGEST) [Clae84] is used in this framework. This is in fact a collection of a large number of individual tools and utilities, but mainly the simulation kernel and some coefficient optimization tools, CANDI and CAUCASIAN have been used.

Finally, different approximation algorithms have been coded using *Matlab* [MAT 89] and *Mathematica* [Wolf88]. These tools are also well suited to perform some basic analysis.

### 7.1.3. Realization of WDSSFs

Wave digital state-space filters can be derived by transforming normal WDFs using the essential numerically equivalent transform described in Chapter 4. They can also be derived by using the concepts of an n-port voltage wave scattering description discussed in the same chapter.

To support the first approach three tools have been partly developed and are under further development, SFD, SFGkernel and COOP in figure 7.2. The first tool SDF [Anso91a, Anso91b] can be used to transform analog reference structures into WDFs and to transform WDFs into state-space form.

The second tool SFGkernel [Sjös89b] was developed mainly to have a uniform description of filters and other fixed DSP algorithms. They can be entered in schematic form using this tool and they can be transformed to the NESS description.

The third tool COOP [Sjös88, Sjös89b] is a special purpose coefficient optimization program to optimize state coefficients for WDSSFs. All these tools and the algorithms used will be described in separate sections.

Concerning the realization of WDSSFs using the scattering matrix approach no tool has yet been developed. Some experimental methods to derive them have been evaluated using the *Mathematica* system from [Wolf88]. For the proposed method by Martens and Meerkötter [Mart78] some routines are under development.

### 7.1.4. VLSI Domain Tools

Apart from a basic set of tools such as layout tools, design rule checkers, simulators, etc., tools to generate, compose and route the global modules are needed. This can be found in many modern VLSI CAD systems. Currently all tasks in this step are completely based on the COMPASS tools, i.e., the CAD System from VLSI Technology Inc. (VTI) [VTI 87]. Also some other tools, e.g., TILT [Riem90] and ALADDIN [Pigu86] have been used for the cell library development.

Basic leaf cells such as bit-slices of the distributed arithmetic processor, and memory cells have been developed using mainly hand-crafted layout methods and VTIlayout. Using these cells, logical blocks like processing, storage, and control units are generated from a parameterized model. This has been done using a specialized utility of VTI, namely the language VTIvip.

The complete implementation step is viewed in figure 7.3. In the *Architecture Net List* (ANL), generated in the previous architecture synthesis step of the framework (this will be described in a later separate section), all necessary modules and their parameters are described. Parameters for the module and model generators can be extracted from this

netlist. Also a HNL can be extracted (this is the old standard hierarchical netlist used in VTI) for use in chip composition and verification.

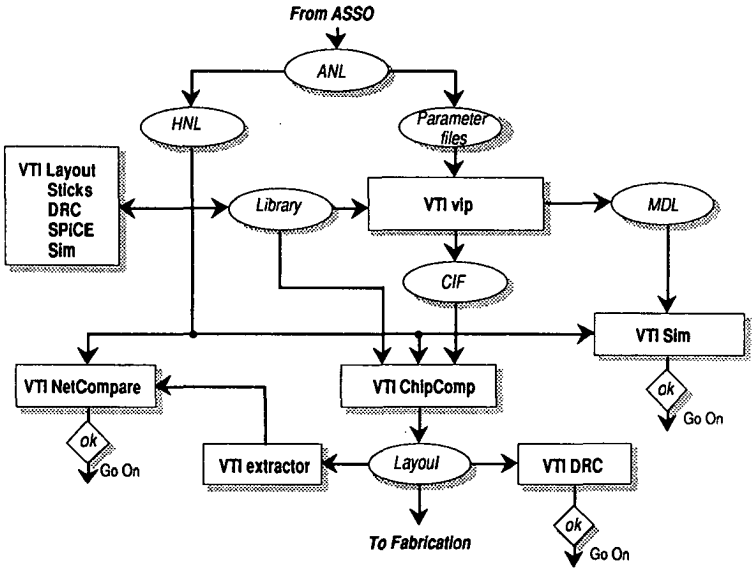



***Figure 7.3*** *VLSI Implementation system*

The VTIvip module generators then produce the actual layout for the required modules and their corresponding simulation models, MDL. Before going further, a simulation should be made to verify the correctness of the mapping and the generated models. In principle this should not be necessary once the framework is complete and verified, then a *correct by construction* approach can be followed. If the simulation turns out to be satisfactory the next step is to compose the complete chip. For this the netlist, the modules and some additional cells are needed.

Using the net extractor the final layout can be compared to the initial netlist. The *Design Rule Checker* (DRC) can be used to verify that the layouts agree with the design rules.

Some remarks must be made concerning the used implementation system:

- Even if this seems to be a straightforward and simple system, some difficulties have been noticed when trying to realize the described ideas. First of all it has been observed that the VTI (Compass) tools

have some limitations and some faulty behavior. For instance, the very vital chip composition using tool VTIChipComp has shown to be very troublesome. So far, not a single chip within this project has been realized using this tool, simply because it has not managed to route even the simplest chip.

- What is even worse is that the standards within the system have changed. Net lists in the new and old format are not compatible which is truly remarkable. Backwards compatibility is not maintained. The approach used in figure 7.3 can no more be used due to this. Furthermore it seems that full custom tools, like for instance the sticks editor, are no more fully supported.

## 7.2. Architecture Synthesis

Since the filter algorithm is described in form of innerproducts and these are realized by distributed arithmetic processors, it is fairly straightforward to translate the DSP algorithm into hardware requirements.

The mapping of the filter algorithm into the proposed architecture can be performed in various ways. On one extreme, a separate processor unit can be used for each innerproduct, i.e., $P = N_{IP}$. This is convenient in applications where the required speed is high. In case of non-recursive algorithms or block processing, this can even be extended to M processors per innerproduct accordingly to Chapter 5. On the other extreme, in some applications the speed requirement is lower and all innerproducts can be calculated using a single time-multiplexed processor. Normally, solutions found between these two extremes are used.

Usually the number of possible solutions is relatively large and it is time consuming to evaluate all of them by hand. Therefore, some strategies are proposed. These will be included in the *Architecture Synthesis and System Optimization* (ASSO) tool that is under development [Sjös89b], figure 7.1.

### 7.2.1. The Target Architecture

The system architecture was presented in Chapter 5. To summarize some important concepts and parameters figure 7.4 is shown. The processing unit contains P processors and in the used implementation these are bit-serial distributed arithmetic processors. Each processor $P_{Ei}$ will require $M_{PEi}$ inputs and one output. The storage unit, on the other hand, contains S storage elements. These are realized either by register banks or RAMs. Storage elements $SE_j$ have $I_{SEj}$ inputs and $O_{SEj}$ outputs.

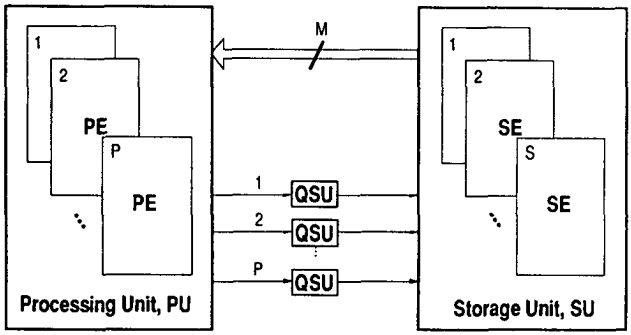



***Figure 7.4*** *The bit-serial target architecture*

The two units are connected by an M bit wide bit-serial bus directed from the SU to the PU. It is possible to place quantization and saturation unit (QSU) at the output of the PU if this is necessary. The PU have P outputs. In principle this gives the number of inputs to the SE. No inter-processor communication is allowed inside the PU.

### 7.2.2. Mapping

The proposed strategy for finding an efficient mapping of the DSP algorithm onto this architecture, can be presented accordingly to figure 7.5. The first step corresponds to calculate the required number of processors P. It is speed requirements, variable dependencies from the DSP domain and speed and latency constraints from the VLSI domain that gives the minimum number of processors P. This number implicitly also gives the number of QSUs.

The next step is the binding where each IPs is bound to a specific PE, or in other words, where parts of a PEs are allocated for the innerproducts. This is made under the constraint that the area should be minimized, and that other additional VLSI constraints are not exceeded. Typically IPs with the same coefficient wordlength and the same dependency on the variables are mapped on the same processor. This step will give the resource allocation for the PU.

The scheduling of the DSP algorithm can be made under the assumption that no constraints are imposed by the SU. The proposed storage unit is very flexible, i.e., it can be configured different ways and it supports in principle any periodical time-schedule. Thus, only constraints from the DSP algorithm and constraints given by the processors are considered. The scheduling

should be performed such that the number of input variables required at any time, M, and the cycle-time $T_C$ are minimized.

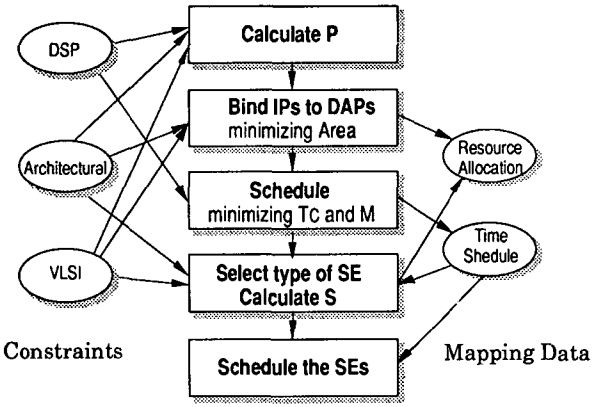



***Figure 7.5*** *The proposed mapping strategy*

After these steps the exact time-schedule, $T_C$, M and P are known. This is all what is required to configure the SU. An estimation of the total required storage capacity in terms of bits and storage-time should be followed. This can be used to decide whether to use RAM or register based SEs. Different criteria in these two cases are used to calculate the number of required SEs.

Finally, a local scheduling of the complete SU should be made. First it should be decided on which SE a variable should be stored and then to find the correct the sequence for the SEs. The last step is trivial since the time-schedule is already given.

### 7.2.3. Allocation and Scheduling of the PEs

Using the constrains from the DSP domain and from the VLSI domain, the architecture P can be derived from:

$$T_{sampl} \leq T_C + \mathrm{Max}\left[0,\ T_L - T_{mc} - \left\lceil \left(T_C - \frac{T_C}{N_C}\right) \right\rceil \right] \qquad (7.1)$$

where

$$T_C = \left\lceil \frac{N_{IP}}{P} \right\rceil N_C\, T_{mc} \qquad (7.2)$$

if no overlap in the processing of channels is allowed. This is obtained using the definitions in Chapter 5. (7.1) will also give the required clock-cycle $T_{cl}$.

However, all the calculations there were based on the assumption that the state-matrices are dense, i.e., they contain only non-zero elements. This corresponds to a worst case situation. If the full possibilities offered by sparse state matrices should be used, the first three phases in figure 7.4 will be an iterative optimization process. The cycle-time can be improved by binding variables that are non-dependent on each other on the same processors.

One example is a WDLF realized using two or three PEs. Due to the parallelism offered by the two branches in the WDFs the recursive part of the state matrix A in (4.1), will consist of two separate blocks. By binding innerproducts from both of these blocks to all PEs and using a special scheduling the effective latency time can be reduced. This can be compared to processing two separate channels, due to the data independence between the two branches.

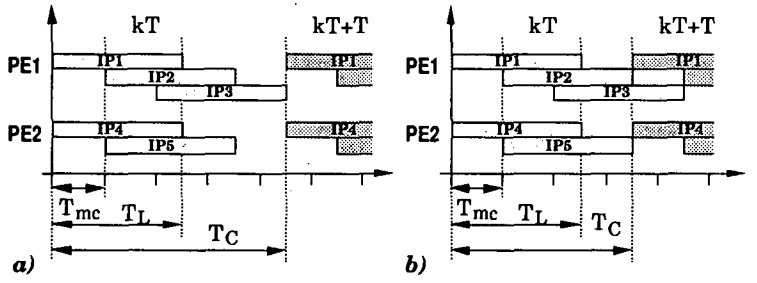



***Figure 7.6*** *Example of scheduling where the cycle-time can be reduced.*

A general rule that always can be applied is to map the output variables on the PEs that should calculate the highest number of innerproducts. By scheduling these variables last on these processors, a reduced cycle-time can be achieved, since these variables correspond to a non-recursive part and will not be needed in the subsequent sampling period. An example of this is shown in figure 7.6. Five innerproducts are scheduled on two processors, figure 7.6a. $IP_3$ corresponds to the calculation of a non-recursive output variable and the processing of the next sampling interval, kT+T, can start earlier, figure 7.6b.

I general a simple *as-soon-as-possible* (ASAP) scheduling [Wanh91] is proposed. Whenever it is possible the time-scheduled can be compacted by analyzing the scheduling diagram, figure 7.6. The three first steps of figure 7.5 will then be an iterative process where optimization could be applied.

### 7.2.4. SE Allocation and Scheduling

After the exact time-schedule of the DSP algorithm is obtained the requirements for the storage unit can be extracted. This is simplest achieved using a *life-time table* of the variables [Wanh91]. The required length of this table corresponds to the longest sequence given by the time-schedule, in other words the cycle-time $T_C$.

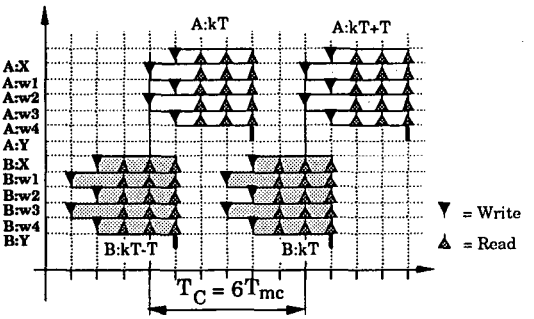



***Table 7.1*** *Variable life-time of a two-channel fourth order filter*

A simple example of a life-time table is shown in table 7.1. The example there corresponds to a fourth order two-channel ( channel A and B ) recursive filter. This could typically be a WDSSF obtained from a ladder reference filter. Three PEs are used in this case, i.e., P = 3, and the overall cycle-time is 6 macro-cycles long. In this specific example it is not required to store the output y, a vertical bar in the diagram denotes this.

The total storage capacity can be derived from this diagram by summing the life-time of all variables during the full cycle time $T_C$. The total number of variables that needs to be stored at any time instance is found by vertically scanning the table. In this example a maximum of 7 variables are stored and a total storage-time of 34 $T_{mc}$ is required.

Table 7.1 can directly be used as a resource allocation table for a single RAM based SE. The maximum number of input variables required in any time instance is found to be three. This also corresponds to the number of processors P. The maximum number of outputs is five. Five registers are then required in both CRBs. The total storage capacity of this RAM would be five words and not seven, because of the pipelining by the conversion registers. The writing of one variable and the reading of one other can overlap $T_{cm}$ in time. In this example it is not realistic to use a RAM SE.

Another solution is to use register based SEs. The number of required SEs is given by P, i.e., tree in this example. Using register SEs the reading and writing can overlap any number of cycles by increasing the length of the input register. A possible resource allocation for the example, using three SEs, is shown in Table 7.2.

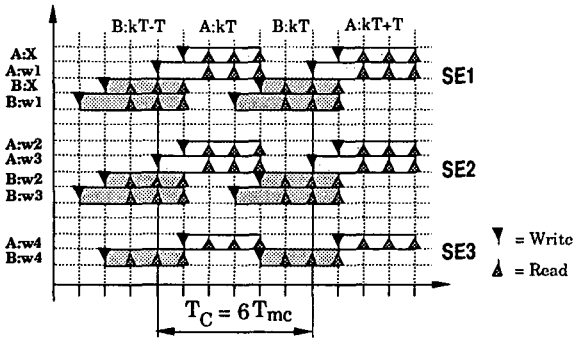



***Table 7.2*** *Resource allocation table.*

The two first storage elements, SE1 and SE2 will require two rotate-registers and two shift-registers each. The third will require one rotate- and one shift-register. An overall requirement of 10 registers in the complete SU is obtained for the example. This solution is clearly simpler.

Finally, when the resource allocation and the configuration of the SEs are determined, a local scheduling should be performed. However, this scheduling is completely given by the used resource allocation tables.

### 7.2.5. ASSO

The CAD tool ASSO is still under development, no parts concerning the scheduling have yet been implemented. Instead, estimators to quickly evaluate different resource allocations have been developed. This is useful for the second step of figure 7.5 where an allocation that minimizes the area should be found. The evaluation of (7.1) can also automatically be performed to give the bounds on the number of processors, the number of channels, the sampling period, etc.

Estimators to calculate the areas of different configured storage elements have been written. This is very important for a quick and accurate evaluation of different configurations.

The scanning of the variable life-time table could as well be included in this tool. Different solution could be generated and evaluated automatically using some very simple algorithms.

All this concerns the mapping and the system optimization. The other part of ASSO, namely, the architecture synthesis should generate the necessary parameters and netlists for the implementation system. These parameters are all included in the *Architecture NetList* (ANL) in figure 7.3. However, due to hesitations to change the implementation system, no real effort has been put on this part. Anyway this is a straightforward translation of the mapping results into a netlist and to PE and SE parameters.

## 7.3. SFGkernel

A special purpose software, SFGkernel, has been developed. The signal flow graph is entered to the system using a graphical editor supporting hierarchy. It has been found that this is a convenient way of describing DSP algorithms. Using a schematic representation it is more convenient then it would be using for example a procedural language. It is easier for a human to control the correctness of the description this way. Using hierarchy the filter algorithms can be described using, for instance, *wave flow graphs* instead of the normal SFG. This saves time and it eliminates the probability for errors.

The second motive to develop SFGkernel was to obtain an automatic way to transform DSP algorithms into state-space form. This has shown to be very time consuming and cumbersome to perform by hand. For the general case it is sufficient to make a numerical state-space transformation. However, when WDFs are concerned, if coefficient optimization is required an analytic or symbolic *State-Space Representation* (SSR) is needed. SFGkernel was developed to support the coefficient optimization program COOP.

An important feature of SFGkernel is the possibility to generate outputs in different netlist formats. In this framework it is especially interesting to generate DIGEST netlist description both for the original DSP algorithm and for the state-space representation of the algorithm.

### 7.3.1. SFG Editor

The chosen input format of SFGkernel is a parametric *Hierarchical Netlist Language* (HNL) file. This corresponds to the general netlist description used in the COMPASS (VTI) tools until version 7 [VTI 87]. This

netlist format was chosen for the simple reason that a schematic editor supporting hierarchy and parameters was directly available. This is the VTIschematic editor from the COMPASS (VTI) tools, figure 7.7. A complete parser for the HNL description was written in Pascal.

Parameterization and hierarchy are important features that allow a modular and systematic construction of complex filter schematics (SFGs) starting from basic building blocks.

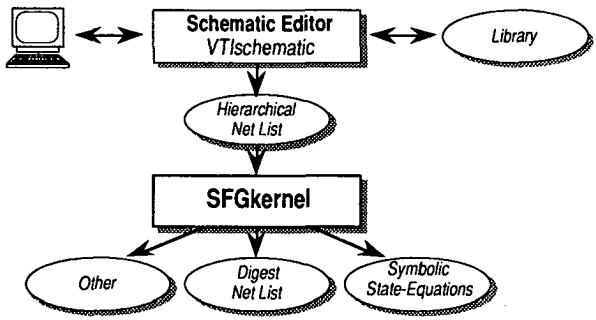



***Figure 7.7*** *SFGkernel environment*

A small library of parameterized DSP modules has been developed using this tool. This contains, different two- and three-port adaptors for WDFs, second order directform IIR sections, FIR transversal blocks, etc. It also includes basic operators such as two-input adders, multipliers and quantization operations, delays, etc. Preferably, the HNL files would be directly be available after the synthesis step.

Once the hierarchical netlist is generated, SFGkernel parses this description and constructs an internal flattened flow graph, figure 7.8. This graph can directly be used to generate outputs in many different formats. Currently *DIGEST netlist* and a symbolic state-space description are supported. For each variable a delay element is introduced. The delay elements are the only signal terminals used in the program. This means that delay elements with zero-delay (T = 0) are introduced for inputs and outputs and in some places of the SFG schematics to obtain auxiliary variables. These last types of variables are useful for different decompositions of the DSP algorithm.

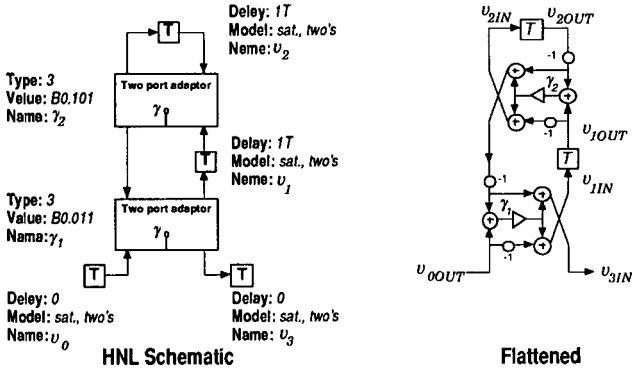



***Figure 7.8*** *Example of SFG schematics and flattening*

### 7.3.2. State-Space Transformations

By identifying all output nodes, i.e., all inputs to delay elements, a reversal parsing of the graph is made. Starting from one output, $v_{iIN}$, and back tracking all the branches connected to this node until one reaches all inputs with paths to this output, a dependency graph is constructed. One dependency graph for each output variable is created, figure 7.9. This is the reverse process compared to the precedence graphs introduce in Chapter 4.

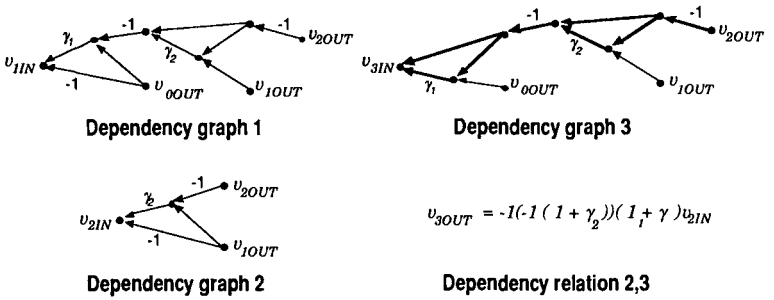



***Figure 7.9*** *Dependency graphs and relations extracted from the example in figure 7.8*

By following the path from an input, $v_{jOUT}$, in dependency graph $i$ until one reaches the output, the explicit symbolic dependency can be determined, this corresponds to state-space coefficient $\alpha_{ij}$. In figure 7.9 an example is shown in dependency graph 3. Using this algorithm a factorized form is

automatically obtained. Some efforts have also been made to reduce this expression further, e.g., to eliminate repetitive multiplication with -1. In case numerical values are required instead, the coefficients should simply be calculated from the value parameters given by the HNL.

SFGkernel generates two different state-space representations, one in form of a DIGEST netlist for simulation and verification and one in symbolic form to be used by the coefficient optimization tool. The last one is generated as a source module (in Pascal) expressing explicitly the state-space coefficients $\alpha_{ij}$ in terms of SFG coefficients $\gamma_{kl}$. This module is compiled and linked with COOP, cf. section 7.5, where it is used to compute the cost function based on the SS coefficient wordlengths. Clearly, faster evaluation of wordlengths is obtained for the optimization this way compared to using data files.

The HNL is important for verification of the SS transformation. It directly grants the possibility for detailed simulations and analysis of the SS represented algorithm. The SFGkernel can be seen as a general utility to automatically translate the filter algorithm between different representations. Unfortunately the HNL and the VTIschematics are no more supported in newer versions (v8r1 and higher) of the COMPASS tools and therefore other possibilities have to be investigated.

## 7.4. SFD

When synthesizing reference filters for WDFs, tools to synthesize classical analog networks are required. The obtained reference filter should be transformed to a WDF using the rules given from the theory of WDFs. To simplify this straightforward but time consuming procedure a new utility the *Symbolic Filter Design* (SFD) tool has been developed by Ansorge, [Anso91a, Anso91b].

SFD generates a *Hierarchical Signal Flow Graph* (HSFG) describing the generated WDF as a result. In a second phase, similarly to the SFGkernel tool, this filter representation can be translated into the numerically equivalent state-space form. That is, the symbolic state-space representation that is necessary for the state-coefficient optimization by COOP.

### 7.4.1. Ladder Network Decomposition

In the present version of SFD, WDF derived from doubly terminated ladder reference filters can be realized. They are realized using the direct connected serial and parallel n-port adaptors [Fett86]. Presently work to

extend the tools for other types of WDFs is made. This includes structures containing unit elements and filters with other topology than the ladder network.

The tool is written in Common Lisp [Stee90] and is based on symbolic processing methods using a rule-based approach for the application of exact behavior-preserving transformations. The input to the tool is for the moment a spice-like netlist based on the prefix-oriented list notation used in Common Lisp. This netlist is first parsed, the syntax is verified and the correctness of the ladder network is checked.

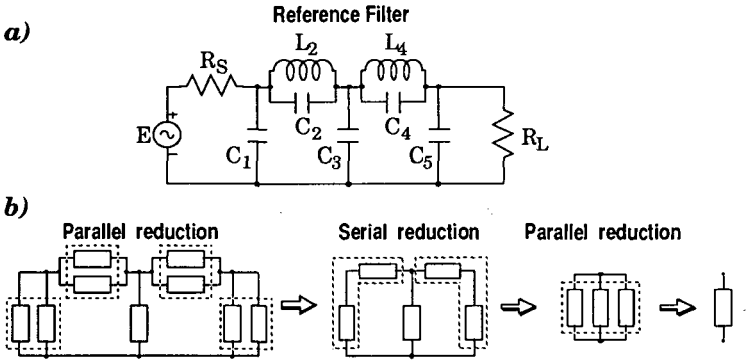



***Figure 7.10*** *Example of reference filter and serial and parallel reductions.*

A ladder network basically consists of alternating parallel and serial interconnections, i.e., shunt (tree) ports and series (link) ports. Thus, using a recursive algorithm the structure is reduced using repeatedly serial and parallel reductions. This is best demonstrated by the example in figure 7.10. The reference filter shown corresponds to a fifth-order elliptic low-pass filter derived using for instance FILSYN.

Starting with a parallel reduction the structure is reduced as proposed in figure 7.10b. Using a serial reduction followed by one more parallel reduction finally a single impedance is obtained.

While these reductions are performed a tree structure is obtained, figure 7.11. This representation is very useful for making some optimization of the filter. For instance purely capacitive or purely inductive loops or cut-sets can be detected. One element in each of these can be suppressed [Fett75b]. This is a feature that has been built in SFD. Another optimization to make is to

try to obtain a balanced tree that gives a minimal computational path for the WDF.

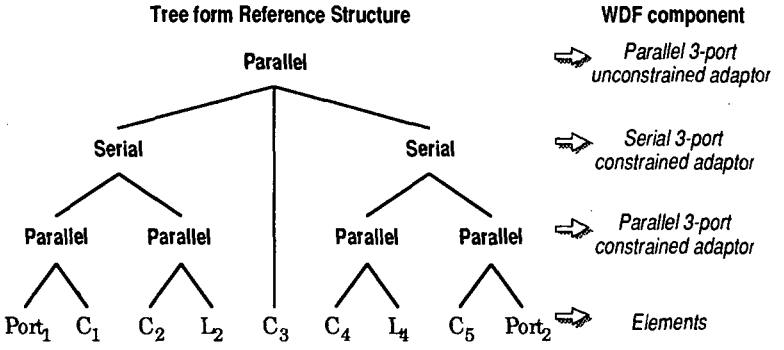



***Figure 7.11*** *Tree representation.*

### 7.4.2. Realization of WDFs

When the best tree-form has been found, i.e., a balanced tree, the WDF transformation step can take place. Clearly the tree leafs (the elements) can directly be mapped to their WDF counter parts accordingly to the description in Chapter 3. The parallel and serial nodes in the tree correspond to n-port adaptors where one port is reflection free.

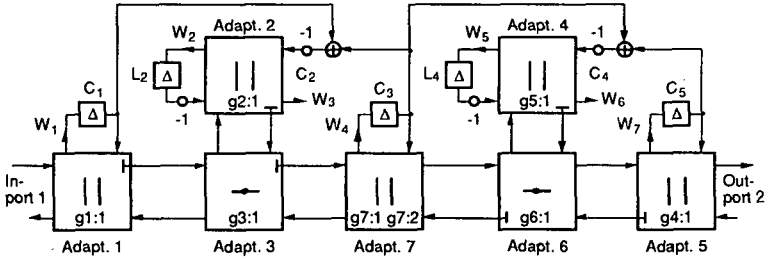



***Figure 7.12*** *Obtained WDF.*

At the bottom of the tree, all parameters (port resistances) are given. On the next level of nodes the port resistances can be chosen such that this will correspond to a reflection-free port. Delay-free loops would otherwise occur in the WDF. At the final level, i.e., for the root node all port resistances are known and an unconstrained adaptor is used.

An HSFG describing the WDF, figure 7.12, is obtained using a netlist syntax presented in [Anso91a]. The parameters in this netlist are given both in symbolic and in numerical form.

A second part of SFD parses this netlist and flattens it. It is flattened until it only contains assignment statements based on elementary algebraic operations. This directly gives the polynomials describing the state-space coefficients symbolically, although a numerical derivation is also supported. This symbolic state-space representation can afterward be used in a coefficient optimization.

## 7.5. COOP

An optimization kernel called COOP, (COefficient Optimization Program) has been developed. COOP is based on *simulated annealing* [Kirk83] and *multivariable feedback representation* [Jain85].

Many objective functions can be defined, but basically the hardware requirements should be minimized. Due to the specific architecture used here, a set of minimum length state-space coefficients, or to be more exact a set of minimum length ROM coefficients, will also result in a minimal hardware configuration.

### 7.5.1. Simulated Annealing

Coefficient optimization problems have been found to be NP-complete. No analytic methods can generally be found. To find the global optima a full evaluation of the complete discrete parameter search space must be performed. Some rather heuristic methods based on experience are usually employed. However, in recent years a method called simulated annealing [Kirk83], has shown to be very useful for these kinds of optimization problems. Simulated annealing has been used with success in different parts of CATHEDRAL 1 for solving similar problems [Jain86].

Simulated annealing is an optimization based on a random algorithm. It is closely related to the physical process of annealing of a solid. At higher temperatures the behavior is random, atoms move up and down between different energy states. While slowly cooling down the solid this behavior becomes more and more controlled, it is less probable that a state with higher energy is taken. When reaching the absolute zero temperature all the atoms have taken the minimal energy state.

By letting the parameters to be optimized (the coefficients in this case) correspond to the atoms, and the cost function be the energy state a similar behavior is obtained. The probability to accept a change or a move is evaluated as $p = e^{-\Delta C/T}$ where $\Delta C$ is the difference in cost $\Delta C = C_{new} - C_{old}$ and T the current temperature. If $p \geq 1$, or the new solution has a lower or equal cost compared to the old, the new solution will always be accepted, if $p < 1$ there is a certain probability that the move will be accepted since it is compared to a normal distributed random, number in the range [0,1].

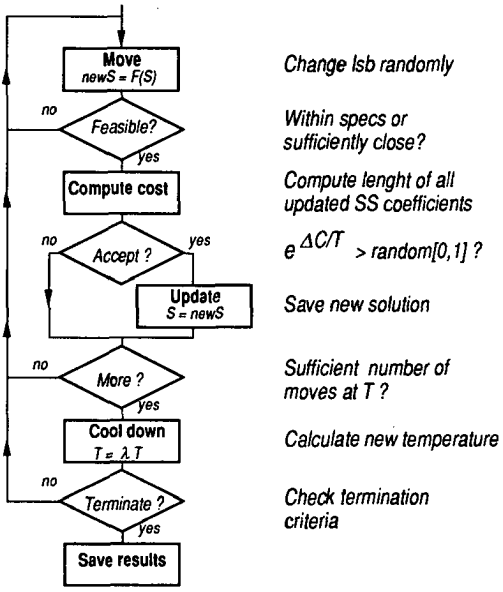



***Figure 7.13*** *Coefficient optimization flow-graph*

The main flow of COOP is shown in figure 7.13. One or more filter coefficients $\gamma_{kl}$ are selected for a move (change) where the least significant bits are randomly changed, i.e., the wordlength changes in the range of ± N bit. The multivariable feedback representation is used to rapidly check if the filter is still within the specifications, and thus, if the new solution is feasible. If so, a cost function is evaluated. Here the explicit represented state-space coefficients $\alpha_{ij}$ generated by SFGkernel can be used.

An acceptance test follows, here all new solutions with lower cost are automatically accepted and some solution with higher cost might be

accepted randomly with decreasing probability dependent on the simulated annealing temperature T. An inner-loop criterion is checked, i.e., if sufficiently numbers of moves have been made at the current temperature. The chosen number determines the annealing speed, if a too fast annealing is made the optimum is probably not reached.

When cooling is required the annealing temperature is exponentially decreased accordingly to the figure. The parameter $\lambda$ usually ranges from 0.8 to 0.99 and will also affect the annealing speed. Finally, the termination test checks whether the optimization should be terminated, otherwise the loop is repeated. The termination criterion checks whether the minimal temperature (the zero temperature) is reached or if no acceptances have been made at the last number of temperature steps.

### 7.5.2. Multivariable Feedback Representation

One of the more important part in the optimization process, whatever method is used, is to check the feasibility of the obtained solutions. Normally this demands that the requirements given as specifications are still fulfilled. In the optimization algorithm given above these feasibility checks should be carried out each iteration. The number of iterations is very large in simulating annealing, in the order of millions. The evaluation is also carried out at different frequencies. Thus, an efficient way to calculate the transfer function for different values of the coefficients is very important.

Normally this can be performed using *symbolic expressions*, *repeated network analysis* or *truncated Taylor series*. However, all these methods have some drawbacks concerning either complexity or numerical accuracy. Instead a method proposed by Jain et al. [Jain85] have been adopted. This is the *MultiVariable Feedback representation* (MVF).

By viewing the system $\tilde{\mathbf{N}}$ as having the normal inputs and outputs, simplified to one of each here, and multiple feedbacks containing the coefficients to be optimized, the MVF representation is obtained. Changing one of the parameters can be distinguished as using an additional multiplier $\Delta\gamma_j$ in parallel to the original $\gamma_j$, figure 7.14a or as adding a signal $x_j = y_j \Delta\gamma_j$ to $u_j$, figure 7.14b. A transfer matrix $\mathbf{T}(\gamma, \omega_k)$ can now be defined as

$$\mathbf{Y} = \mathbf{T}(\gamma, \omega)\,\mathbf{U} \text{ or } \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} t_{00}(\gamma, \omega) & \mathbf{t}_{01}^{T}(\gamma, \omega) \\ \mathbf{t}_{10}(\gamma, \omega) & \mathbf{T}_{11}(\gamma, \omega) \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{u} \end{bmatrix} \qquad (7.1)$$

where $\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_n]^T$, $\mathbf{x}$ and $\mathbf{y}$ are the vectors of the corresponding variables and $\omega$ is the frequency. By calculating one matrix $\mathbf{T}_k(\gamma)$ for each

frequency $\omega_k$ all the elements in the matrices will be constants. Clearly the system function at a fixed frequency $H_k(\gamma(i))$ is equal to $t_{00}(\gamma(i))$ at this frequency.

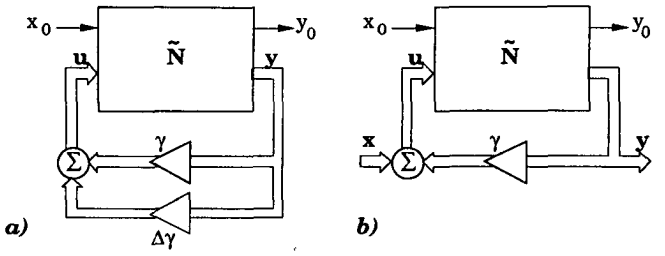



***Figure 7.14*** *Multivariable feedback representations of* $\tilde{N}$

By letting $\Gamma$ be the diagonal matrix formed by $\Delta\gamma$ and

$$\mathbf{V} = [\,\mathbf{I} - \mathbf{T}_{11}(\gamma(0))\,]^{-1} \tag{7.2}$$

and the new set of coefficients are $\gamma(i) = \gamma(0) + \Delta\gamma$ then it can be shown [Jain85] that the new values of $\mathbf{T}_k(\gamma(i))$ can be calculated as

$$t_{00}(\gamma(i)) = t_{00}(\gamma(0)) + \mathbf{t}_{01}^{T}(\gamma(0))\,\Gamma\,\mathbf{V}\,\mathbf{t}_{10}(\gamma(0)) \tag{7.3}$$

$$\mathbf{t}_{01}^{T}(\gamma(i)) = \mathbf{t}_{01}^{T}(\gamma(0)) + \mathbf{t}_{01}^{T}(\gamma(0))\,\Gamma\,\mathbf{V}\,\mathbf{T}_{11}(\gamma(0)) \tag{7.4}$$

$$\mathbf{t}_{10}(\gamma(i)) = \mathbf{t}_{01}^{T}(\gamma(0))\,\Gamma\,\mathbf{V}\,\mathbf{T}_{11}(\gamma(0)) \tag{7.5}$$

$$\mathbf{T}_{11}(\gamma(i)) = \mathbf{t}_{01}^{T}(\gamma(0))\,\Gamma\,\mathbf{V}\,\mathbf{T}_{11}(\gamma(0)) \tag{7.6}$$

This method is useful for any fixed DSP algorithm independently of topology and the deviation of the parameters, i.e., the magnitude of $\Delta\gamma$. It has also shown to be useful for sensitivity analysis and to calculate bounds for the parameters.

### 7.5.3. Feasibility Test in COOP

In COOP the calculations of the frequency responses are made at *m* discrete frequencies. This means that *m* transfer matrices $\mathbf{T}_k(\gamma(0))$ have to be calculated initially. Fortunately, this is an option in DIGEST used for the CANDI and the CAUCASIAN optimization tools and can be directly used here. When some coefficients are changed a simplified algorithm is used to obtain $t_{00}(\gamma(i))$. This is described as algorithm *Af* in [Jain85].

- Only the rows and columns in $\mathbf{T}_{11}(\gamma)$ corresponding to the coefficients that are changed are involved in the calculations. Let $\gamma_c = \gamma_c(0) + \Delta\gamma_c$ be the new changed values and accordingly $\Gamma_c$ the diagonal matrix thereof.

- Then an auxiliary matrix $\mathbf{A}_c = [\,\Gamma_c^{-1} - \mathbf{T}_{c11}(\gamma(0))\,]^{-1}$ is computed.

- The next step is to solve the linear equations $\mathbf{A}_c\mathbf{u}_c = \mathbf{t}_{c10}(\gamma(0))$ by *LU-decomposition* and to compute the scalar $w = \mathbf{t}_{c01}^{T}(\gamma(0))\,\mathbf{u}_c$.

- Finally, $H_k(\gamma_c(i)) = t_{00}(\gamma_c(i)) + w$ is calculated.

Different responses, magnitude, phase and group delay can then be compared to the specifications after computing $H_k(\gamma_c(i))$ for all frequencies. In COOP a specification cost factor is used to allow a small deviation from the specifications, it has been found that the optimizations converge faster due to this in some cases. This penalty factor should be chosen sufficiently high to ensure that the final optimized DSP algorithm fulfills all the specifications.

If the new solution is found to be feasible, a new $\mathbf{T}_k(\gamma(0))$ can be calculated accordingly to (7.3 -7.6). Naturally only the rows and columns corresponding to the changed coefficients are needed in the calculations, all other elements remain unchanged. Another possibility would be to store the update in a new vector containing accumulated updates rather than recalculating $\mathbf{T}_k(\gamma(0))$. However, when the calculation of a new move of a coefficient in COOP is dependent directly on the value of this coefficient the first method is used.

### 7.5.4. Cost Function

Different objective functions can of course be defined and the final choice is dependent on the architecture. To overcome the problem to define a fixed cost function that is valid for all cases a number of different cost parameters have been introduced.

$C_b$ = the bit cost that is multiplied to the total number of bits. This optimizes the overall wordlength.

$C_l$ = the maximal length coefficient bit cost that is multiplied to the longest coefficient and it tends to equalize all coefficients in length.

$C_m$ = The minimum magnitude coefficient cost used to try to force coefficients to zero. It is multiplied to the coefficient with smallest magnitude.

$C_s$ = the specification cost, this is the cost to exceed the tolerances given by the specifications. It is multiplied to a total accumulated error from all specification points used.

All these costs are added together to form the global cost $C_{TOT}$ in (7.7).

$$C_{TOT} = C_b + C_l + C_m + C_s \tag{7.7}$$

The costs are simply defined in a parameter file and by giving one of them a high value the corresponding objective will be highly weighted in the optimization. By using the $C_s$ parameter, the response of the filter can temporarily exceed the specifications. This was added to improve the randomness at higher temperatures and it seems to have improved the speed of the optimization.

### 7.5.5. Limitations

Currently two versions of COOP exists, one in Pascal for VAX/VMS and one written in C on a UNIX workstation. The program has been successfully used in a number of application examples for the direct optimization of filter coefficients. It has shown to be noticeable faster than for instance CANDI and CAUCASIAN.

The link between SFGkernel and COOP, i.e., the generation of the source code for the state-equations, was never fully established. A decision to cancel further development of SFGkernel was made when the netlist format was no longer supported by VTI. The symbolic state-space representation from SFD is so far only given as a netlist. If a direct optimization of state-coefficients should be made the source code for the equations must be entered by hand. In fact it is the distributed arithmetic ROM coefficients that should be used for the optimization. These can simply be calculated as sums of the symbolic state equation coefficients accordingly to the definition.

The optimization of ROM coefficients will require a much higher accuracy during the optimization. Already when optimizing the direct filter coefficients, it has been noticed that the 32-bit integer format used in COOP to represent the coefficients is sometimes to low. For the final optimized set of coefficients this is sufficient, but due to the randomness of simulated annealing it is necessary to accept solutions that sometimes contain very long coefficients. For the ROM coefficient optimization, some experience has

been made using the 55-bit mantissa of a double-real floating-point number to represent the coefficients. Even this can be limiting in some cases, and naturally it slows down the optimization.

Optimization of the original coefficient values will give a sub-optimal solution, and is perhaps the simplest possibility for avoiding accuracy problems. It has also been found that using simulated annealing the simulation parameters must be precisely tuned. Otherwise the optimization will not work efficiently. Presently other methods are under consideration.

# chapter 8. CONCLUSIONS

*Some obtained results are briefly discussed including two different application examples. This is followed by some conclusions concerning the developed methodology, its framework, the usefulness and the performance. Finally, propositions for different improvements and suggestions of some fields to for future research are given.*

## 8.1. Performance

### 8.1.1. Achieved Results

To start with the performance, it has been found by simulations that the developed hardware units work at a clock-rate of approximately 100 MHz using the 2 μm CMN20a technology. In principle, this means that a maximal achievable sampling rate of 3 - 12 MHz can be obtained, since the data wordlength typically varies between 8 - 32 bit. Each processor occupies approximately 1 $mm^2$ of chip area that makes it possible to use up to 20 or 30 of the processors on a single chip. This gives an indication of the upper limit of complexity that can be achieved with the developed hardware.

### 8.1.2. PCM Benchmark

A simple application example is the benchmark PCM filter [Yazd90a] described in Chapter 4. A single DAP processing unit was used together with a single register based SE. A chip die-size of approximately 1.1 mm x 1.1 mm is required when a wordlength of 21 bit is used. The maximal sampling-rate in this example should be very close to 5 MHz. This has not yet been verified by measurement due to lack of appropriate test equipment. A prototype chip

has been realized, and that it will be analyzed in the near future. A plot of a complete layout for this WDF is shown in figure 8.1.

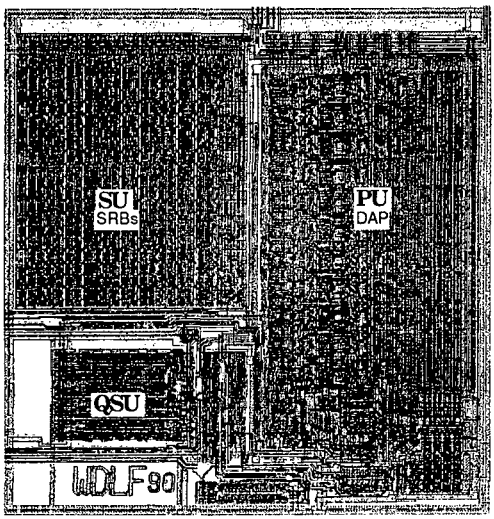



***Figure 8.1*** *Layout of a PCM filter implementation.*

The chip compares well to what is achieved by other approaches implementing the same benchmark filter. Table 8.1 shows a comparison between our approach, **imt**, and implementation approaches found in [Clae88]. These are denoted C–I, C–II and C–III for Cathedral I, II and III, respectively. The implementations are compared in terms of maximal clock-frequency, wordlength, maximal achievable sampling frequency, area, and required number of clock-cycles. Finally, a *figure of merit* taking into account the used area over the maximal performance is calculated.

The figures for the Cathedral implementations have been found in [Clae88]. The C-I implementation is based on a bit-serial data-flow architecture using CSD multiplications. Different architectures for the C-II are listed whereas it is possible to use a wide range of different *data-paths* for the Cathedral II system. The C-III corresponds to a systolic array implementation. Three different alternatives for our system are also listed. It consists of using either one two or three *distributed arithmetic processors* (DAPs).

All the values given for the area and speed correspond to a 3 μm technology. Since the presented architectures have exclusively been implemented in a 2 μm technology a linear scale factor has been used to calculate the corresponding values for a 3 μm technology. This gives that, 1 $mm^2$ of area correspond to 2.25 $mm^2$ and 100 MHz clock speed to 66 MHz. The required wordlength too meet the specifications was 20 bit but when our approach is based on a bit-serial system an additional guard-bit is required to detect overflow. (In principle this should also be true for the C-I implementation.)

| System | Data path | Clock freq. MHz | Word-length bit | Cycles | Sampl. freq kHz | Area $mm^2$ | A/Fs $\frac{mm^2}{MHz}$ |
|---|---|---|---|---|---|---|---|
| C-I | bit-serial | 20 | 20 | 20 | 1000 | 2.3 | 2.3 |
| C-II | 1 alu | 10 | 20 | 12 | 833 | 6.8 | 8.2 |
| C-II | 2 alu's | 10 | 20 | 8 | 1250 | 10.6 | 8.5 |
| C-II | 1 alu +mult | 10 | 20 | 9 | 1111 | 22.8 | 20.5 |
| C-III | bit-parallel | 5 | 20 | 1 | 7500 | 14.8 | 2.0 |
| **imt** | **1 DAP** | **66** | **21** | **126** | **523** | **2.5** | **4.8** |
| **imt** | **2 DAP** | **66** | **21** | **63** | **1046** | **3.6** | **3.4** |
| **imt** | **3 DAP** | **66** | **21** | **42** | **1596** | **4.7** | **2.9** |

***Table 8.1*** *Comparison of different implementations*

The presented approach compares fairly well to the three Cathedral systems. The clock frequency is more than tree times faster compared to the other bit-serial approach C-I. This is the result of the very careful design of the cells.

The implemented 1 DAP solution from figure 8.1 requires a relatively small area. Due to the multiplexing of the DAP, i.e., all six innerproducts are calculated on the same DAP, the achievable speed is not as high as for some other architectures. Anyway the obtained figure of merit is in parity with the corresponding one for C-I that also is a bit-serial solution.

The best figure of merit obtained for our approach comes from the 3 DAP solution. Here the relative overhead for the storage unit is smaller and the sampling frequency can be three times higher than the 1 DAP solution. The figure of merit compares well to the best of the Cathedral systems.

Due to constraints imposed by latency it is not useful to use more than three DAPs. Surprisingly it seems that the C-I implementation does not suffer from this latency problem.

It should also be noted that the chosen benchmark is possibly not the best for comparing VLSI architectures since the filter could very well be implemented on a DSP processor and ASIC approaches should aim at targets with higher requirements. Even if the presented approach is competitive with the Cathedral systems it would perform better for more complex benchmark filter. The approach is better suited for higher speed and longer innerproducts and it is especially efficient for non recursive applications such as FIR filters and discrete transforms.

### 8.1.3. 2-Dimensional DCT for Real-Time Video Applications

The second application example concerns the design and implementation of a DCT chip for real-time video applications [Defi89, Sjös90, Defi90]. The obtained results are remarkable. Using a chip die-size of 4.5 mm x 5.0 mm and approximately 70'000 transistors, a 16-point 2-dimensional DCT can be implemented, figure 8.2. The solution uses a processing-unit based on 16 distributed arithmetic processors and a storage unit containing a single storage element based on a 256 x 16 bit RAM.

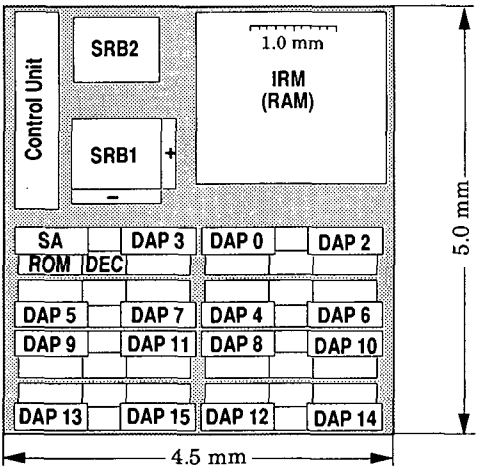



***Figure 8.2*** *Floorplan of a 2-D DCT chip.*

The performance of this chip corresponds to impressive 268 Mops where one operation is one innerproduct. If the equivalence in more normal multiply and add operations are considered, a figure of 503 Mops is obtained. All this is achieved at a moderate clock speed of 16.8 MHz. In principle this clock speed could be increased six times accordingly to the specifications on the designed hardware. This gives some indication of what can be achieved using the proposed approach and implementation.

This DCT chip compares extremely well, considering both the speed and area, to some other developments from for instance, SGS Thomson, Bell Core, Philips, MIT, AT&T Bell Labs, etc. [Rao 90]. It is also believed that the development time is considerable shorter using the presented methodology.

### 8.1.4. Limitations

Because of the "distributed" nature of distributed arithmetic algorithms, it is rather complex to change or update filter coefficients, however it exists some solutions [Anso90]. Hence, the presented architecture cannot directly be used for adaptive algorithms. By extending the architecture, different sets of precalculated coefficients can be used, but these types of applications were not considered in this work.

The rather complex coefficient optimization that has to be used to obtain numerically equivalent solutions can be restrictive in some cases. One might obtain state-space coefficients that are longer then the data wordlength and this results in a clear penalty in performance using the chosen realization of the DAP. A redesign of the DSP algorithm is a possible work-around, using an algorithm with more inherent parallelism will improve a better result.

At lower speeds the proposed architecture will become less efficiently utilized. Practical examples have shown that realizations of simple filters at, for example, telephony rates, only would use a fraction of the potential performance of the architecture. This kind of applications has not been the target of the methodology, implementations based on commercial available DSP processors are usually possible in these cases. However, it should be pointed out that the concept of WDSSFs described in this methodology can very well be used also for DSP implementations [Bals90].

The framework is currently running in a mixed environment on different types of platforms, VMS (micro-VAX), UNIX (APOLLO), and MacOs (Apple Macintosh). This is a limitation in many cases since design data have to pass from one system to the other. In the future, it is only planned to support UNIX and X-Window environment.

## 8.2. Conclusions

### 8.2.1. Choice of Filter Structures

In the proposed approach the wave digital filters play an important role. They have been chosen due to their unique properties under finite wordlength conditions [Fett86], due to the efficient realization of these filters and their moderate hardware requirements. Finally, they are very flexible due to the enormous range possibilities offered and the complete theory developed. WDFs are one of the very few IIR filters that can be used for any kind of digital filter application, since stability under all conditions can be ensured.

However, the implementation approach can be extended to include other kind of IIR digital filters. Since they do not add any new features, practically no effort has been made to study them in more detail. On the other hand, the FIR filters and the linear discrete transforms are important groups of fixed DSP algorithms that have been included.

The choice of initial filter structure should mainly be based on the signal processing properties, since the implementation tasks remain the same independently of this choice. However, the choice will have a substantial influence on the obtained performance and efficiency of the implementation. Structures with inherent parallelism and low coefficient sensitivity have shown to be suitable for the used approach.

The signal processing properties are conserved during the state-space transformation. The use of the numerically equivalent state-space transformation even improves the signal-to-noise ratio and the dynamic range for the WDFs. It also results in very regular and modular structures that simplifies the mapping onto the proposed architecture.

### 8.2.2. Architecture and Implementation

Because of the chosen architecture, it has been possible to keep the whole methodology reasonable simple. Using the distributed arithmetic algorithm to calculate innerproducts is not only well adopted to the state-space filters, it also leads to simple and efficient implementations. Limitations on programmability are of course imposed but this is of no concern to the methodology and the class of applications considered here. The use of distributed arithmetic in combination with state-space representation also reduces the number of operations to a minimum and the parallelism to a maximum.

Using exclusively bit-serial components in the architecture, many things are simplified. For instance, the usual communication bottle-neck of data-flow architectures is, if not totally avoided, reduced to an acceptable level. Furthermore, the required memory bandwidth is reduced. Using bit-serial architectures a maximal clock-rate can be maintained due to the inherent pipelining on the bit-level. The area-speed trade-off compares very well to other architectures.

The definition of the system architecture proposed here, leads to some simple rules for the mapping and scheduling. The architecture is also flexible, it can be used for a wide range of applications using only some few basic components. Other bit-serial processors than the DAP can also be included without changing the definition.

On the implementation level only, optimized full-custom designed cells and modules are used. This both increase the speed and the reduces the area. Larger parameterized modules are compiled using module generators.

The reusability of the designed VLSI blocks is also satisfying in this methodology due to the restrictions on the architecture. The same blocks can be reused for implementations of various applications. As long as the same class of technology is used no redesign of the basic cells is required.

### 8.2.3. Synthesis System

Using the proposed synthesis system, it should be possible to arrive to correct and efficient solutions in a limited amount of time. Clearly, the used system is not automatic and the user interface is not yet optimal, but in any case all tasks of the design and the implementation process are treated. The algorithms used in these tools are simple and efficient and this is only possible due to the constraints built-in into the methodology.

The emphasis has been put on tools solving problems specific to this methodology. Although some of these tools might not be complete, it has been shown that the used algorithms are useful and that the tasks can be solved as proposed.

With some additional effort, it would surely be an important effort for this framework, a semiautomatic *correct-by-construction* synthesis system could be the ultimate goal. Instead, taking advantage of new coming systems and environments like DSP Station [DSP 91] the effort put on developing tools could be moved over to other more challenging tasks instead.

## 8.3. Future Improvements

### 8.3.1. DSP Algorithms

Concerning synthesis and design of WDSSFs the proposed methods could be extended. First of all, extensions to include multi-rate signal processing could be made. This will have some interesting effects on the state-space representation. State-decimation techniques could be employed to reduce computational requirements in some cases.

Some methods for partitioning of DSP algorithms could also be developed. It is in few cases that a complete DSP system can be described in a full state-space form efficiently. Normally it is better to partition the system in smaller parts where each part is described in state-space form. This usually leads to better implementations. A natural extension would be to investigate the optimal way of doing this partitioning.

The method to derive the n-port WDFs proposed by Martens and Meerkötter [Mart76] could be developed. It seems that it is possible to derive an orthogonal set of parameters similar to the adaptor coefficients used by Fettweis. If these could be defined, the synthesis of WDSSFs derived from ladder reference filters would become considerably simpler. This would probably also have some consequences on the implementation, since shorter coefficients would be expected in this case.

### 8.3.2. Design-Data Representation

In the DSP design process the passage between different steps includes changes and translations between different representations of the design-data. It would be desirable to avoid this inconsistency by using a specialized description language. In this case everything from the specifications to the DSP algorithms and the corresponding flow graphs could be represented in this language. This would also simplify the simulation, analysis, and optimization phase if the same language could be used also there. The language should be able to describe a detailed structure, preferably also using different levels of hierarchy, and additional constraints individual elements.

In fact the task is very similar to languages used for the description of hardware, i.e., so-called *Hardware Description Languages* (HDLs). To use one of these, for example the new standard VHDL [Lips89], would have the advantage that the architecture and the hardware could be described the same way. A language that has been developed for describing DSP algorithms is SILAGE [Sche88]. This seems to be a reasonable choice, since

simulators already exist. It has also been used as a vital part of the CATHEDRAL II - IV systems.

Verification and test are sometimes tasks that are poorly covered in design approaches. Unfortunately this is also the case here, verification in form of simulation has been given a fair share of the attention, while testing only has come to ideas or some ad-hoc strategies. This is a field that should be much more developed in a complete system or a complete methodology. Fortunately, many things concerning the testing are very simple for the proposed architecture, but still, some structured strategy should be developed possibly using self-tests.

Simulation as a verification on different design levels would be much simpler to perform if for instance SILAGE or VHDL would be used.

### 8.3.3. Technology Independence

One obvious enhancement of the implementation strategy would be to improve the portability of the layout blocks. This can be achieved by using some symbolic layout descriptions, where the real mask layers are generated only at the very last moment when the technology has been chosen. However, this would have consequences on other parts of the system, for instance the architecture synthesis. Some of the mapping rules depend directly on technology features, e.g., the maximal clock speed.

The realization of this idea depends heavily on the available VLSI CAD tools. While this is a natural built-in feature of some, it is not accessible in other systems. Based on the experience made using Compass (VTI) it is believed that one should not purchase a VLSI Design system that does not support features like symbolic layout, at least not if the tools should be used for full-custom designs.

# AKNOWLEDGEMENTS

*"I would like to thank everybody, especially myself and Mrs Albrecht"*
Anonymous chinese speaker at the conference dinner URSI-89, Erlangen.

First of all, I wish to express my gratitude to Professor Fausto Pellandini who offered me the possibility to come to the University of Neuchâtel and work under his supervision.

Furthermore, I wish to tank all the members of the jury for their effort to read and review the contents of this thesis.

A special thanks to Dipl. Ing. Michael Ansorge and Dipl. Ing. Ivan Defilippis for their contribution to this project. Without their help and collaboration and many suggestions it would not have been possible to conclude this work.

I am also very grateful to all other friends and colleagues that I had the opportunity to work with during these years. Dipl. électr. phys. Peter Balsiger, B.Sc. Javier Bracamonte and Dipl. électr. phys. Sandro Ponta, to mention a few.

I also want to thank my former colleagues from the University of Linköping, especially Dr. Lars Wanhammar and Dr. Björn Sikström for introducing me to this field.

Finally, I cannot express the value of the support, assistance, love and encouragement I have got from my companion and girlfriend M.Sc. Ellen Blikstad over these years.

The financial support of the work was partly provided by the *Swiss Foundation for Research in Microtechnology* (Grants FSRM CS 85/7, CS 87/11 and CS 88/14), and partly by the *Commission for the Promotion of Applied Scientific Research* (Grant CERS C-2014.1).

# REFERENCES


[Anso86] M. Ansorge : "Réalisation d'une cellule d'additionnuer complet", *Internal Communication*, 1986.

[Anso89] M. Ansorge : "Architectures VLSI Appliquées au Traitement Numérique du Signal : Architecture et Algorithmes", *Rapport IMT* 89, IMT Neuchâtel, 1989 (in french).

[Anso90] M. Ansorge, I. Defilippis, U. Sjöström, P. Balsiger, and F. Pellandini : "A Flexible Low-Power Digital Signal Processor Based on a Content Addressable Memory", *Proc. Europ. Signal Processing EUSIPCO'90*, Vol. 3, pp. 1399-1402, Barcelona, Spain, September 1990.

[Anso91a] M. Ansorge, U. Sjöström, I. Defilippis, P. Balsiger, and F. Pellandini : "On the Automated Symbolic Design of Wave Digital Filters", *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing ICASSP'91*, pp. 1561-1564, Toronto, Canada, May 1991.

[Anso91b] M. Ansorge, U. Sjöström, I. Defilippis, and F. Pellandini : "Symbolic Design of Ladder Wave Digital Filters", Accepted for publication at *GRETSI'91*, Juan-Les-Pins, France, September 1991.

[Arjm81] M. Arjmand, and R. A. Roberts : "On Comparing Hardware Implementations of Fixed Point Digital Filters", *IEEE Circuits and Systems Magazine*, Vol. 3 , No. 2, pp. 2-8.

[Bahe84] H. Baher : *"Synthesis of Electrical Networks"*, John Wiley & Sons, New York, 1984.

[Bals87] P. Balsiger: "Etude des filtres numérique d'ondes réalisés en arithmétique distribuée, en vue d'une implementation sur silicium", Projet de diplôme, Université de Neuchâtel, Switzerland, March 1987, (in French).

[Bals90] P. Balsiger, U. Sjöström, and F. Pellandini, "Digital Signal Processor Implementation of Lattice Wave Digital Filters", EUSIPCO'90, Barcelona, Spain, Sept. 1990, pp. 1531-1534.

[Barn77] C. W. Barnes, and A. T. Fam : "Minimum Norm Recursive Digital Filters that are Free of Overflow Limit Cycles", *IEEE Trans. Circuits and Systems*, Vol. CAS-24, No 10, pp. 569-574, October 1977.

[Bele68] V. Belevitch : *"Classical Network Theory"*, Holden-Day, San Fransico, USA, 1968.

[Blah85] R. E. Blahut : *"Fast Algorithms for Digital Signal Processing"*, Addinson-Wesley, USA, 1985.

[Buri85] M. R. Buric, and T. G. Matheson : "Silicon Compilation Environments", *IEEE 1985 Custom Integrated Circuits Conf.*, pp. 208-212, 1985.

[Burl89] W. P. Burleson, L. L. Scharf : "A VLSI Design Methodology for Distributed Arithmetic", *IEEE Trans. Computer Architecture*, Vol. 10, 1989.

[Burr71] C. S. Burrus : "Block Implementation of Digital Filters", IEEE Trans. Circuit Theory, Vol. CT-18, No. 6, pp. 697-701, November 1971.

[Burr77] C. S. Burrus : "Digital Filter Structures Described by Distributed Arithmetic", *IEEE Trans. Circuits and Systems*, Vol. CAS-24, No. 12, pp. 674-680, December 1977.

[Bütt76] M. Büttner, and H. W. Schüßler : "On Structures for the Implementation of the Distributed Arithmetic", *Nachrichtentechn*, Z. 29 , H. 6, pp. 472-497, 1976.

[Cand86] J. V. Candy : *"Signal Processing: The Model-Based Approach"*, MacGraw-Hill, Singapore, 1986.

[Claa75] T. A. C. M. Claasen, W. F. G. Mecklenbräuker, and J. B. H. Peek : "On the Stability of the Forced Response of Digital Filters with Overflow Nonlinearities", *IEEE Trans. Circuits and Systems*, Vol. CAS-22, No. 8, pp. 692-696, August 1975.

[Claa76] T. A. C. M. Claasen, W. F. G. Mecklenbräuker, and J. B. H. Peek : "Effects of Quantization and Overflow in Recursive Digital Filters", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-24, No. 6, pp. 517-529, December 1976.

[Clae84] L. Claesen, H. J. De Man, and J. Vandewalle : "DIGEST: A Digital Filter Evaluation and Simulation Tool for MOSVLSI Filter Implementations", *IEEE Journal Solid-State Circuits*, Vol. SC-19, No. 3, June 1984.

[Clae88] L. Claesen, F. Catthoor, D. Lanner, G, Goossens, S. Note, J. Van Meerbergen, and H. De Man : "Automatic Synthesis of Signal Processing Benchmark Using CATHEDRAL Silicon Compilers", *Proc. IEEE Custom Integrated Circuits Conf. '88*, pp.14.7.1-14.7.4, 1988.

[Croc75] R. E. Crochiere, and A. V. Oppenheim : "Analysis of Linear Digital Networks", Proc. IEEE ??, Vol. 63, No. 4, pp. 581-595, April 1975.

[Croi73] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Riso : "Digital Filter for PCM Encoded Signals", U.S. Patent 3777 130, December 4, 1973.

[Defi89] I. Defilippis, U. Sjöström, M. Ansorge, and F. Pellandini : "A 2-Dimensional 16 Point Discrete Cosine Transform Chip for Real Time Video Applications", *Proc. GRETSI'89*, Vol. 2, pp. 813-816, Juan-Les-Pins, France, June 1989.

[Defi90] I. Defilippis, U. Sjöström, M. Ansorge, and F. Pellandini : "Optimal Architecture and Time Scheduling of a Distributed Arithmetic Based Discrete Cosine Transform Chip", *Proc. Europ. Signal Processing EUSIPCO'90*, Vol. 3, pp. 1579-1582, Barcelona, Spain, September 1990.

[DeMa86] H. De Man et al. : "CATHEDRAL II: A Silicon compiler for digital signal processing", *IEEE Design & Test of Computers*, pp. 13-25, .

[DeMa87] H. De Man, J. Rabaey, P. Six, and L. Claesen : "Computer Aided Synthesis Systems for Digital Signal Processing", *Proc. Journées d'électronique*, Lausanne, Switzerland, October 1987.

[DeMa88] H. De Man, J Rabey, J. Vanhoof, G. Goossens, P. Six, and L. Claesen : "CATHEDRAL II - a Computer-Aided Synthesis System for Digital Signal Processing VLSI System", *Computer-Aided Engineering Journal*, pp. 55-66, April 1988.

[Deny85] P. Denyer, and D. Renshaw : *"VLSI Signal Processing: A Bit-Serial Approach"*, Addinson-Wesley, VLSI Systems Series, USA, 1985.

[Dinh84a] F. Dinha, B. Sikström, U. Sjöström, and L. Wanhammar : "A Multi- Processor Approach to Implement Digital Filters", *Proc. Nordic Symp. VLSI in Computer and Communications*, pp. 142-147, Tampere, Finland, June 1984.

[Dinh84b] F. Dinha, B. Sikström, U. Sjöström, and L. Wanhammar : "LSI Implementation of Digital Filters - A Multi-Processor Approach", *Proc. Int. Conf. on Computers, System and Signal Processing*, Vol. 3, pp. 1316-1320, Bangalore, India, December 1984.

[DSP 91] *"DSP Station Product Description"*, Mentor Graphics Corporation, Leuven, Belgium, 1991.

[Erik79] S. Eriksson : "Some Details Concerning the Wave Digital Filter Structures", Report LiTH-ISY-I-0323, Linköping University, Sweden, 1979.

[Fett71] A. Fettweis : "Digital Filter Structures Related to Classical Filter Networks", *Arch. Elek. Übertragungtech.*, Vol. 25, pp. 79-89, February 1971.

[Fett72] A. Fettweis : "Pseudopassitivity, Sensitivity, and Stability of Wave Digital Filters", *IEEE Trans. Circuit Theory*, Vol. CT-19, pp. 668-673, November 1972.

[Fett74] A. Fettweis, H. Levin, and A. Seldemeyer: "Wave Digital Lattice Filters", *Int. J. Circuit Theory Applications*, Vol. 2, pp. 203-211, June 1974.

[Fett75a] A. Fettweis, and K. Meerkötter : "Suppression of Parasitic Oscillations in Wave Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-22, No. 3, pp. 239-246, March 1975.

[Fett75b] A. Fettweis : "Canonic Realization of Ladder Wave Digital Filters", *Int. J. Circuit Theory Applications*, Vol. 3, pp. 321-332, 1975.

[Fett75c] A. Fettweis, and K. Meerkötter : "On Adaptors for Wave Digital Filters", *IEEE Trans. Acoust., Speech and Signal Processing*, Vol. ASSP-23, No. 6, pp. 516-525, December 1975.

[Fett76] A. Fettweis : Realizability of Digital Filter Networks, *Arch. Elek. Übertragungtech.*, Vol. 30, pp.90-96, February 1976.

[Fett84] A. Fettweis : "Digital Circuits and Systems", *IEEE Trans. Circuits and Systems*, Vol. CAS-31, No. 1, pp. 31-48, January 1984.

[Fett85] A. Fettweis, J. A. Nossek, and K. Meerkötter : "Reconstruction of Signals after Filtering and Sampling Rate Reduction", *IEEE Trans. Acoust., Speech and Signal Processing*, Vol. ASSP-33, pp. 893-902, August 1985.

[Fett86] A. Fettweis : "Wave Digital Filters: Theory and Practice", Proc. IEEE, Vol. 74, No. 2, pp. 270-327, February 1986.

[Fey 87] C. F. Fey, and D. E. Paraskevopoulos : "A Techno-Economic Assessment of Application-Specific Integrated Circuits: Current Status and Future Trends", Proc. IEEE, Vol. 75, No. 6, June 1987.

[Flor63] I. Flores : *"The Logic of Computer Architecture"*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1963.

[Gajs83] D. Gajski, and R. H. Kuhn : "New VLSI Tools", *IEEE Computers Magazine*, pp. 11-14, December 1983.

[Gazs85] L. Gazsi : "Explicit Formulas for Lattice Wave Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-32, No. 1, pp. 68-88, January 1985.

[Gazs86] L. Gazsi : *"FALCON Reference Manual"*, Ruhr University Bochum, W. Germany, 1986.

[Gray73] A. H. Gray and J. D. Markel : "Digital Lattice and Ladder Filter Synthesis", *IEEE Trans. Audio Electroocoust.* Vol. AU-21, pp. 491-500, December 1973.

[Haro89] B. S. Haroun, and M. I. Elmarsy : "SPAID: An Architectural Synthesis Tool for DSP Custom Applications", *IEEE Journal Solid-State Circuits*, Vol. SC-24, No. 2, pp. 426-435, April 1989.

[Hwan77] S. Y. Hwang : "Minimum Uncorrelated Unit Noise in State-Space Digital Filtering", *IEEE Trans. Acoust., Speech and Signal Processing*, Vol. ASSP-25, No. 4, pp. 273-281, August 1977.

[Hwan79] K. Hwang : *"Computer Arithmetic: Principles, Architecture, and Design"*, John Wiley & Sons, New York, 1979.

[Jack70] L. B. Jackson : "Roundoff-Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form", *IEEE Trans. Audio Electroacoust.*, Vol. AU-18, pp. 107-122, June 1970.

[Jack76] L. B. Jackson : "Roundoff Noise Bounds Derived from Coefficient Sensitivities for Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-23, No. 8, pp. 481-485, August 1976.

[Jack79] L. B. Jackson : "Limit Cycles in State-Space Structures for Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-26, No. 1, pp. 67-68, January 1979.

[Jack89] L. B. Jackson : *"Digital Filters and Signal Processing"*, Second Edition, Kluwer Academic Publishers, 1989.

[Jain85] R. Jain, J. Vandewalle, H. J. De Man : "Efficient and Accurate Multiparameter Analysis of Linear Digital Filters Using a Multivariable Feedback Representation", *IEEE Trans. Circuits and Systems*, Vol. CAS-32, No 3, pp. 225-235, March 1985.

[Jain86] R. Jain et al. : "Custom Design of a VLSI PCM-FDM Transmultiplexer from Systems Specifications to Circuit Layout Using a Computer-Aided Design System", *IEEE Journal Solid-State Circuits*, Vol. SC-21, pp. 73-86, February 1986.

[Jaya84] N. S. Jayant, and P. Noll : *"Digital Coding of Waveforms: Principles and Applications to Speech and Video"*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1984.

[Kirk83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi : "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671-680, May 1983.

[Kuno88] I. Kunold : "Linear Phase Realization of Wave Digital Lattice Filters", *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing, ICASSP'88*, pp. 1455-1458, New York, April 1988.

[Ligt86] A. Ligtenberg, M. Vetterli, and J. H. O'Neill : "MOVAL: A Framework for turning Digital Signal Processing Algorithms into Custom VLSI", *Signal Processing*, Vol. 11, No. 2, pp. 119-132, 1986.

[Lips89] R. Lipsett, C. Schaefer, and C. Ussery : *"VHDL: Hardware Description and Design"*, Kluwer Academic Publishers, 1989.

[Mart76] G. O. Martens and K. Meerkötter: "On n-Port Adaptors for Wave Digital Filters with Applications to Bridged-T Filters", *Proc. IEEE Int. Symp. on Circuits and Systems, ISCAS'76*, pp. 514-517, Munich, W. Germany, April 1976.

[Mart78] G. O. Martens, and H. H. Lê : "Wave Digital Adaptors for Reciprocal Second-order Sections", *IEEE Trans. Circuits and Systems*, Vol. CAS-25, No. 12, pp. 1077-1082, December 1978.

[Mart80] G. O. Martens : "Voltage Scattering Matrix Analysis of n-Ports with Application to Wave Digital Filters", *Proc. Int. Conf. on Circuits and Computers*, pp. 1180-1183, Port Chester, NY, USA, October 1980.

[MAT 89] *MATLAB User's Guide*, The MathWorks, Inc., Sherborn, MA, USA, 1987.

**References**


[McCl73] J. H. McClellan, T. W. Parks, and L. R. Rabiner : "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters", *IEEE Trans. Audio Electroacoust.*, Vol. AU-21, pp. 506- 526, December 1973.

[Mead80] C. Mead, and L. Conway: *"Introduction to VLSI Systems"*, Addison-Wesley, Series in Computer Science, USA, 1980.

[Meer77] K. Meerkötter : "Canonic Realization of Wave Digital Filters Involving Unit Elements", *Int. J. Circuit Theory Applications*, Vol. 5, pp. 359-402, 1977.

[Meer80] K. Meerkötter : "Incremental Pseudopassivity of Wave Digital Filters", *Proc. Europ. Signal Processing Conf. EUSIPCO'80*, pp. 27-31, Lausanne, Switzerland, September 1980.

[Mull76a] C. T. Mullis, and R. A. Roberts : "Synthesis of Minimum Roundoff Noise Fixed Point Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-23, No. 9, pp. 551-562, September 1976.

[Mull76b] C. T. Mullis, and R. A. Roberts : "Roundoff Noise in Digital Filters: Frequency Transformations and Invariants", *IEEE Trans. Acoust., Speech and Signal Processing*, Vol. ASSP-24, No. 6, pp. 538-550, December 1976.

[Noss83] J. A. Nossek, and H.-D. Schwartz : "Wave Digital Lattice Filters with Applications in Communication Systems", *Proc. IEEE Int. Symp. Circuits and Systems ISCAS'83*, pp. 845-848, Newport Beach, CA, May 1983.

[Note88] S. Note, J. Van Meerbergen, F. Catthoor, and H. DeMan : "Automatic Synthesis of a High Speed Cordic Algorithm with CATHEDRAL III Compilation System", *Proc. IEEE Int. Symp. Circuits and Systems ISCAS'88* , Helsinki, Finland, June 1988.

[Oppe75] A. V. Oppenheim, and R. W. Schafer : *"Digital Signal Processing"*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1975.

[Pele74] A. Peled, and B. Liu : "A New Hardware Realization of Digital Filters", *IEEE Trans. Acoust., Speech and Signal Processing*, Vol. ASSP-22, No. 6, pp. 456-462, December 1974.

[Pigu86] C. Piguet, M. Stauffer, and M. Joss : "AUTO_CELL Gate Matrix Cell Compiler, Users Guide", Centre Suisse d'Electronique et Microtechnique (CSEM), Neuchâtel, Switzerland, November 1986.

[Raba85] J. Rabaey, S. Pope, and R. Brodersen : "An Integrated Automated Layout Generation System for DSP Circuits", *IEEE Trans. on Computer Aided Design*, Vol. CAD-4, pp. 285-296, July 1985.

[Rabi72] L. R. Rabiner, J. W. Cooley, H. D. Helms, L. B. Jackson, J. F. Kaiser, C. M. Rader, R. W. Schafer, K. Steiglitz and C. J. Weinstein : "Terminology in Digital Signal Processing", *IEEE Trans. Audio Electroacoust.*, Vol. AU-20, pp. 322-337, December 1972.

[Rabi75] L. R. Rabiner, and B. Gold : *"Theory and Application of Digital Signal Processing"*, Prentice-Hall Inc., Englewood Cliffs, N. J., 1975.

[Rao 90] K. R. Rao, and P. Yip : *"Discrete Cosine Transform: Algorithms, Advantages, Applications"*, Academic Press, San Diego, CA, USA, 1990.

[Renf84a] M. Renfors, B. Sikström, U. Sjöström, and L. Wanhammar : "Design and Implementation of a 32 Channel Wave Digital PCM Filter", *Proc. Nordic Symp. VLSI in Computer and Communications* pp. 114-119, Tampere, Finland, June 1984.

[Renf84b] M. Renfors, B. Sikström, U. Sjöström, and L. Wanhammar : "VLSI Implementation of a Digital PCM Filter," *Proc. Int. Conf. on Computers, System and Signal Processing*, Vol. 3, pp. 1468-1472, Bangalore, India, December 1984.

[Reus83] P. P. Reusens : *"High Performance VLSI Digital Signal Processing Architecture and Chip Design"*. The Faculty of the Graduated School of Cornell University, Thesis, August 1983.

[Riem90] R. Riem-Viis, and G. Maliki : *"TILT, Users Manual"*, Version 2, IMT Rue A. L.-Breguet 2, CH-2000 Neuchâtel, December 1990

[Samu82] H. Samueli and A. V. Willson, : "Almost Period P Sequences and the Analysis of Forced Overflow Oscillations in Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-29, No. 8, pp. 510-515, August 1982.

[Sara85] T. Saramäki, Y. Neuvo, and S. K. Mitra : "Efficient Interpolated FIR Filters", *Proc. IEEE Int. Symp. Circuits and Systems, ISCAS'85*, Vol. 3, pp. 1145-1148, Kyoto, Japan, June 1985.

[Sche88] C. Scheers : "Functional Languages and a Compiler for Silage", March 1988.

[Sedr78] A. S. Sedra, and P. O. Brackett : *"Filter Theory and Design: Active and Passive"*, Matrix Publishers Inc., Champ. Il., 1978.

[Siks86] B. Sikström : *On the LSI Implementation of Wave Digital Filters and Discrete Cosine Transforms*, Linköping Studies in Science and Technology, Diss. No. 143, Linköping University, Sweden, May 1986.

[Sjös86] U. Sjöström : *"A modular and Regular Implementation Scheme for Digital Filters - A multiplexed Wave Digital PCM Filter"*, Linköping Studies in Science and Technology, Thesis No. 62, Linköping University, Sweden, January 1986.

[Sjös88] U. Sjöström : "On the Design and Implementation of Digital Filters : an Approach using Wave Digital Filters and Distributed Arithmetic", *Rapport IMT* No. 223 EC 09/87, IMT Neuchâtel, January 1988.

[Sjös89a] U. Sjöström, I. Defilippis, M. Ansorge, and F. Pellandini : "A Methodology for ASIC Implementation of Digital Filters", *Proc. GRETSI'89*, Vol. 2, pp. 797-800, Juan-Les-Pins, France, June 1989.

[Sjös89b] U. Sjöström, I. Defilippis, M. Ansorge, F. Pellandini : CAD Environment for Digital Filter Design and Implementation, *Proc. URSI Int. Symp. on Signals, Systems and Electronics, ISSSE'89*, pp. 601-604, Erlangen, FRG, September 1989.

[Sjös90] U. Sjöström, I. Defilippis, M. Ansorge, and F. Pellandini : "A Discrete Cosine Transform Chip for Real Time Video Applications", *Proc. IEEE Int. Symp. Circuits and Systems, ISCAS'90*, Vol. 2, pp 1620-1623, New Orleans, USA, May 1990.

[Smit88a] S. G. Smith and P. B. Denyer : *"Serial-Data Computation"*, Kluwer Academic Publisher, Boston, USA, 1988 .

[Smit88b] S. G. Smith, M. Keightley, P. B. Denyer, and S. Nagara : "SECOND: Synthesis of Elementary Circuits on Demand", *IEEE Journal Solid-State Circuits*, Vol. SC-23, No. 3, June 1988.

[Smit89] S. G. Smith, R. W. Morgan, and J. G. Payne : "ASIC Architectures for Digital Signal Processing Implementation", *Proc. GRETSI'89*, Vol. 2, pp. 805-808, Juan-Les-Pins, France, June 1989.

[Steel90] G. L. Steele Jr : *"Common Lisp: The Language"*, 2nd Edition, Digital Press, Burlington, MA, USA, 1990

[Swar86] E. E. Swartzlander Jr. : *"VLSI Signal Processing Systems"*, Kluwer Academic Publisher, Hingham, Massachusetts, 1986.

[Szen77] G. Szentirmai : "FILSYN - A General Purpose Filter Synthesis Program", *Proc. IEEE*, Vol. 65, No. 10, pp. 1443-1458, October 1977.

**References**


[Teme77] G. C. Temes, and J. W. LaPatra : *"Introduction to Circuit Synthesis and Design"*, McGraw-Hill, Tokyo, Japan, 1977.

[VanG86] J. Van Ginderdeuren, H. De Man, B. De Loore, H. Van den Wyngaert, A. Delaruelle, and G. Van den Audenaerde : "A High Quality Digital Audio Filter Set Designed By Silicon Compiler CATHEDRAL-1", *IEEE Journal Solid-State Circuits*, Vol. SC-21, No. 6, December 1986.

[VTI 87] VLSI Technology Inc. : *Manuals*, VTI, 1987.

[Wanh81] L. Wanhammar : *"An Approach to LSI Implementation of Wave Digital Filters"*, Linköping Studies in Science and Technology, Diss. No. 62, Linköping University, Sweden, April 1981.

[Wanh91] L. Wanhammar : "DSP Integrated Circuits", Prentice-Hall Inc., 1991 (in print).

[West81] N. Weste, and B. Ackland : "A Pragmatic Approach to Topological Symbolic IC Design", *Proc. 1st Int. Conf. VLSI*, pp. 117-129, Edinburgh, August 1981.

[West85] N. Weste, and K. Eshraghian : *"Principles of CMOS VLSI Design - A Systems Perspective"*, Addinson-Wesley, VLSI Systems Series, USA, 1985.

[Whit89] S. A. White : "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", *IEEE ASSP Magazine*, Vol. 6, No. 3, pp. 4-19, July 1989.

[Wolf88] S. Wolfram : *"Mathematica: A System for Doing Mathematics by Computer"*, Addison-Wesley, USA, 1988.

[Yazd90a] M. Yazdanpanah, "Conception et Implantation VLSI d'un Filtre Numérique d'Onde", *Rapport IMT* No 270 IC 01/90, IMT Neuchâtel, Janvier 1990 (in french).

[Yazd90b] M. Yazdanpanah, "Conception d'un Filtre Multicadence pour une Application Audio", *Rapport IMT* No 271 IC 01/90, IMT Neuchâtel, Janvier 1990 (in french).