Journal of VLSI Signal Processing 25, 261–284, 2000

A Framework for High Level Estimations of Signal Processing VLSI Implementations

J. Ph. DIGUET
LESTER–Université de Bretagne Sud, 10 Rue Le Coat Saint Haoen, 56100 Lorient, France

D. CHILLET AND O. SENTIEYS
LASTI–ENSSAT–Université de Rennes, 6 rue de Kerampont, 22300 Lannion, France

Received November 24, 1998; Revised June 8, 1999

Abstract. This paper deals with the presentation of a framework for the rapid prototyping of Digital Signal Processing applications. The BSS framework enables both synthesis of dedicated VLSI circuits and cost, performance estimation. The latter can be used at different accuracy levels and can help the designer in selecting a proper algorithm in order to improve the global performance of its implementation. The cost estimation takes into account both the processing unit, including operators, registers, interconnections, and memory units. The implemented estimation techniques are presented and include functional unit number bound calculation, probabilistic cost estimation of processing unit components, and memory unit area evaluation. We demonstrate, on a real application, that cost/signal processing quality trade-offs can be achieved while changing the type of algorithm and the number of filter taps for a given algorithm specification.

1. Introduction

Recent advances in VLSI technologies and Computer-Aided design (CAD) enable us to design systems on a single chip. Programmable Digital Signal Processors and VLSI hardware for signal processing have matured rapidly during the last decade, and cover a large part of the integrated circuit market. We can see that this market is now driven by the explosive growth of digital mobile communication and multi-media [1].

Due to the fast evolution of digital signal processing (DSP) applications and due to the limited life time of new telecommunication products, the *time-to-market* is one of the most important figure of merit, with the design of *first-time-right* complex VLSI systems. Moreover, these systems must optimize design constraints such as signal processing quality, cost, performance and power dissipation. Therefore, the designer faces a huge multidimensional design space bounded by the constraints. In order to find his way through this maze, he must use indications that trade-off metrics provide him.

An *ideal* prototyping framework for DSP designers must deal with:

- Rapid prototyping of DSP applications. The tool must quickly return an estimation (cost, performance, power) of the implementation on the target architecture, to compare different algorithmic solutions. The transformations are not only limited to the algorithmic level, but could be applied at the functional level (e.g. choice of a hardware or software solution for a given function) and at the structural level (e.g. retiming, associativity). The estimation must be accurate, independent from the CAD tools and located at the highest level of abstraction.
- Choice between programmable processors or circuits, dedicated VLSI, or flexible architecture such

as ASIP. The tool must accept a generic model of the architecture. It also must be interfaced with the design environment of all the target architectures (High Level Synthesis, DSP compilers, ...).
- The memory problem which can be, for most of DSP applications, the critical part of cost and power dissipation. Most of the tools neglect the memory unit and focus on the processing part.
- Targeting different VLSI technologies (FPGA, Standard Cells, ...).
- An associated DSP co-simulation tool in order to verify the DSP quality constraints (error, convergence rate, filter parameters, ...).

The methodology that we present in this paper tries to answer most of the problem of rapid prototyping environment for DSP. It enables both estimation and synthesis under cost, time, power dissipation and SNR constraints. This paper will not deal with the two last criterions [2, 3] which will be detailled in a futur paper.

The area/time measurements are obtained in different ways based on various synthesis or estimation techniques. They give an area estimation of the Processing and Memory Units for a given real time constraint. This constraint is generally linked to the sample frequency of the DSP system and is a throughput constraint. The delay between the inputs and the corresponding outputs may or not be constrained by a number of pipeline stages. By using the methodology, the designer can find his optimal trade-off by changing input parameters and iterating the process. This parameters can be either the throughput constraint, or the digital signal processing quality and performance criteria. The latter can be modified by changing the algorithm with transformations (e.g. block processing, passing through the frequency domain, etc.), or the size of the data to be processed (e.g. size of the Impulse Response of an adaptive filter, etc.), or even the bit size of the data. All of these can improve the time performance but can have an effect on the quality of the DSP. It is for this reason, that the estimation must be correlated with the simulation in the same framework.

Different trade-off curves may be built. For a given algorithm, area/time curves are obtained by changing the time constraint T, on which the estimation is based upon. For a given time constraint, quality signal processing/cost curves are established by changing the algorithm or by modifying the characteristics of the same algorithm (see §5).

Figure 1 sums up the different ways which can be followed in order to obtain estimations, and then trade-off curves. The §2.2 presents a method that provides the optimal number of functional units (FU) and pipeline stages of the VLSI architecture that will be used later by the synthesis. This area estimation is dependent on the time constraint. In §2.3, we propose metrics to estimate the number of FU, registers and interconnection components of the processing unit (PU). This method is based on a probabilistic calculation of the DFG operation repartition in time and is linked with hardware selection (see §3) of the high level synthesis system GAUT integrated in the BSS framework [4]. It aims the guidance in the transformation space. The §4 deals with the memory problem. We will show in this section how the memory unit (MU) can be taken into account in the framework. A cost estimation tool, based on bus transfer optimization, number of bank calculation, and memory selection, is presented.

The ultimate way to accomplish trade-offs is through the high level synthesis itself. The PU and MU synthesis integrated in this framework are not presented in this paper, but can be found in [5–7]. Some of the results presented in the section §5 have been achieved with accurate estimations. The application studied deals with new-fast LMS algorithms.

2. Processing Unit Cost Estimation

2.1. Introduction and Previous Works

In this section, we deal with the two kinds of estimation involved in the GAUT framework. Estimations are carried out under a time constraint (iteration bound: T)

The first one aims at providing, the synthesis tool, with optimal numbers of functional units and pipeline stages. This accurate estimation must predict the less expensive set of resources stemming from the more efficient use of the high level synthesis phases. Like these tasks, resource estimation is a NP-complete problem [8] that involves searching the optimized distribution of operations from a DFG into a limited number of time steps.

Previous works deal with upper and lower bounds. They can be classified into two classes, estimation either taking or not taking the precedence constraints into account, between operations in the DFG. The first one contains fast methods that give very optimistic lower bounds and pessimistic upper bounds [9, 10].

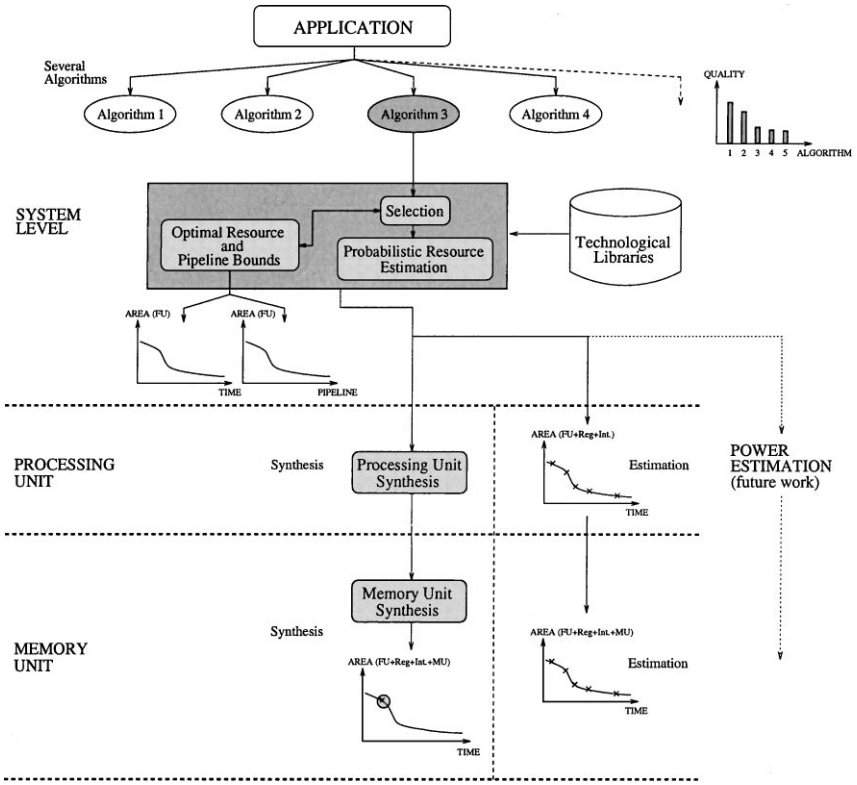

Figure 1. Global framework.

The second category computes the estimation by using the time frames [asap ; alap] of operations. Upper bounds are computed by extracting maximum parallelism from an oriented graph $F(N, R)$ where N_i is a node representing the operation i and $R_{i,j}$ a vertex indicating a precedence relationship between N_i and N_j. A solution is presented in [11] with a $O(N^3)$ complexity and it is suggested in [12] that an algorithm with a $O(N^3)$ complexity is feasible. Lower bounds correspond to the designer objective. In [13] a method is described concerning functional units, data-transfers and registers, its complexity is $O(N^2C)$ for the active resource and $O(NC^2)$ for the interconnects. They compute the minimum concurrency in a given time from a set of time frames of present operations. With such a method, precedence constraints are underestimated, since the overlapping of [asap; alap] intervals may hide dependence constraints as shown in Fig. 12. The lower bounds are solved on the same scheme. Another method is detailed in [10] for the functional units, the authors propose to evaluate the critical path T_c of the application for a given set of resources, if T_c is greater than $T_{\text{constraint}}$ the number of the tested resources is increased, and so on until the time constraint is reached. $T_{\min}$ is computed with a list scheduling algorithm, equivalent to [14], its complexity is $O(N \log(N))$. The algorithm, is fast but can lead to underestimation. It actually depends on a scheduling algorithm that does not always give the best solution, in fact the priority given to the list-scheduling is $1/(alap - asap)$, and, as said before, such time frames do not fully reproduce the precedence constraints between operations. An equivalent algorithm is presented for the interconnection bounds and heuristics are described for register bounds.

The method we shall present, is independent from any scheduling algorithm and combines the estimations of functional units with associated pipeline stages. It avoids exhaustive searching while being accurate, through an analysis of the usual causes of over and under estimations.

The second kind of resource estimation fits in a new approach of high level synthesis. It addresses the cost characterization of the tested application in order to

guide the designer in his specification choices. The estimation may be defined in three points.

Firstly, it's important to note here, that the aim is not the exact prediction of the final circuit features. Due to the level of abstraction, it would be illusory and useless to focuse on accurate area/time estimations of the final circuit. However, it still remains possible to provide reliable metrics which are sufficient to enable comparisons between different specifications. So, the objective is to get relative estimations which express the optimization potential of different algorithmic solutions.

Secondly, it provides an estimation of the distribution of the different kinds of resources in the available time steps. The aim is to get the metrics to analyze the complexity of the tested application, in order to understand and apply transformations.

Finally, it is independent from the synthesis tool.

Few works are aimed towards this kind of estimation. In [15] the authors dealing with the extraction of the properties of an algorithm to guide its synthesis by reducing the design space. The main metrics exposed are the concurrency to get an estimation of the number of resources, the temporal density that represents an approximation of the lifetime of the variables and the regularity that measures the ratio between the whole description of the algorithm and the smallest pattern, from which the algorithm can be reproduced. This work and ours fits in the same bracket of estimation. But, in our opinion, it's more interesting to firstly evaluate all these properties through a probabilistic calculation that includes the dependence relationships of nodes; and secondly to keep the links between the nodes and their localization in the time steps. Finally we can mention [16] where an original method is used, involving entropy to evaluate the repartition of nodes in the different pipeline stages.

The properties of this kind of estimation as a guidance tool in the space of transformations will not be described here. We will report its principle and its use for the estimation of interconnections cost in order to guide the module selection.

2.2. Optimal Resource and Pipeline Estimations for Synthesis

2.2.1. Principle. Within a High-Level Synthesis framework, such as GAUT tool [6], the estimation task follows the hardware component selection that provides the functional units (FU) computation delays and

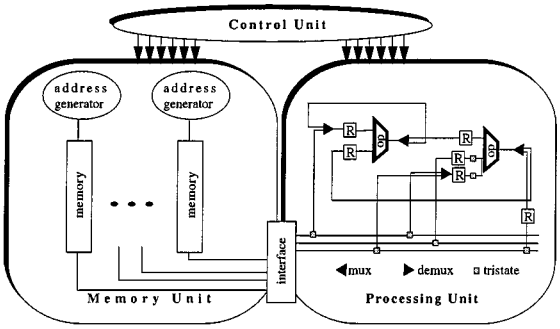

Figure 2. Architectural model of GAUT synthesis tool.

the optimal clock time, the component selection can be modified according to the estimation results [17]. The method presented here, aims for the dual estimation of accurate numbers of FU and pipeline stages. It is based on a preliminary analysis of the error causes in a pipeline architecture whose model is defined in Fig. 2. The data flow graph compiled from the VHDL behavioural specification is completely unrolled. Consequently the only restrictions to resource optimization through pipeline transformations is due to internal feedback loops. In fact, without feedback loops, the average number of FU (Eq. (1)) enables us to execute the algorithms if the appropriated number of pipeline stages is used (see Fig. 3).

With feedback loops, two issues cause estimation errors. The first one is the ignorance of real numbers of the time steps available for the different kinds of FU. The second one is the unavoidable parallelism that has to be done in order to respect the iteration period constraint. In [10] the authors rapidly suggest to take into account the unusable time steps but finally leave the idea because they think it is inefficient in many cases. In fact this information is a usefull part of the solution, we propose here a formulation of the problem in a loop context.

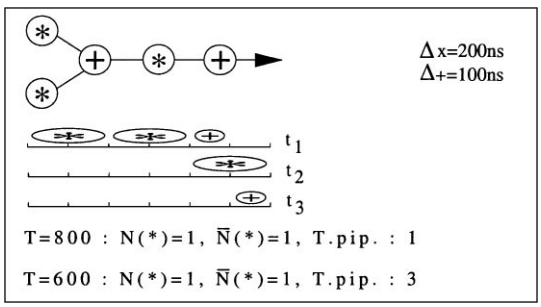

Figure 3. Accurate average number with a pipeline.

Firstly we will not take into account the number of delays in feedback loops. We see later how to benefit from the increase of mobility that the multi-delay graphs allow. This work allows the NP-complete problem to be avoided. Contrary to other techniques, the pipeline possibilities are introduced from the beginning of the estimation.

Bounds are useful for the limitation of the design space, but inefficient for the provision of exact number of resource and pipeline stages that are necessary to correctly guide the scheduling and binding tasks. Our tool GAUT doesn't perfom an hardware exploration, on the contrary it rapidly builds an architecture based on the available FUs and time steps. That's why an accurate estimation is required. Then, the datapath synthesis optimization focuses on minimizing controler, bus, registers costs.

The estimation is made up of two steps. The minimum number of FU is firstly computed with unlimited pipeline, the associated minimum number of pipeline stages is then calculated.

The method will be carried out in four main sections. (ii) computation of a number of FU in a feedback loop, (iii) computation of the global number of FU in the DFG, (iv) computation of the minimal number of pipeline stages, (v) optimisation with a multi-delay feedback loop.

$$\bar{N}(i) = \left\lceil \frac{N\mathrm{op}(i).\Delta(i)}{T} \right\rceil \qquad (1)$$

with: $\Delta(i)$ the computation delay of the FU of type i and $N\mathrm{op}(i)$ the number of operations of type i in the DFG.

2.2.2. Computation of the Number of FU in Feedback Loops. The method may be divided into four main steps for a given type i of operators. Firstly, the maximum mobility is allowed to the operation in the loops. Then we compute the improved average number of FU which is based on the utilization of the exact available time steps: $\bar{N}'(i)$. We treat the problem of local strong parallelism requirements by computing minimum parallelism rate: $\hat{\Phi}_{N\mathrm{par}}(i)$. Finally the number of FU is obtained by keeping the maximum between $\bar{N}'(i)$ and $\hat{\Phi}(i)$.

Improved Average Number. If $N_{\mathrm{op}}^{\mathcal{F}}(i)$ is the number of operations of type i in the feedback loops, $N_{\mathrm{op}}^{\mathcal{F}}(i).\Delta(i)$ time units are required to execute them. In Eq. (1) it is assumed that a whole set (T) of time units is available for the execution of the operations of type i. This assumption is of course, seldom true, the available time steps must belong to, at least, one of the time frames [asap ; alap] of an operation of type i. The number of time steps, in this subset, is defined as $T'(i)$.

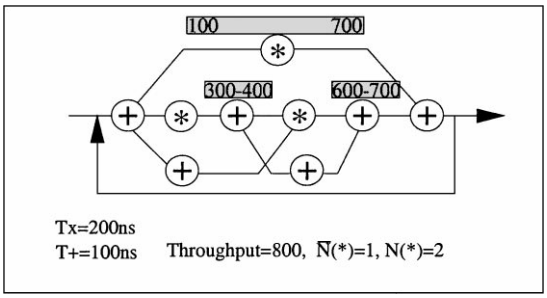

Figure 4. Unavailable time units for multipliers because of split intervals.

Another cause of error remains in the computation of the average number. It is due to a part of time, apparently available for operations of type i, that can not yet be used because of intervals of length lower than the computation delay of operators of type i. The Fig. 4 gives an example, where the apparent number of required multipliers is one and the exact number is two. In this case, the multiplication in the upper branch can not use the free 200 ns because they are split into two intervals of 100 ns. This part of time is due to dependences with the other types of operations. It is defined as $T''(i)$ and must be considered as time units required for the execution of operation of type i. In order to compute the numbers $T''(i)$, we firstly exhibit in each branch of the graph, the maximum time frames that separate two operations of type i. Then we add the intervals from the different branches, and keep only the intervals that have a length divisible by $\Delta(i)$. The construction of the final intervals look like the *Tetris* game. Finally a $mod\,\Delta(i)$ operation, applied to remaining intervals, provides the desired $T''(i)$ numbers. Details of the formulation are given in [18].

Finally we obtain the improved average number with the Eq. (2).

$$\begin{aligned} \bar{N}'_r(i) &= \frac{N\mathrm{op}^{\mathcal{F}}(i).\Delta(i) + T''(i)}{T'(i)} \\ \bar{N}'(i) &= \lceil \bar{N}'_r(i) \rceil \end{aligned} \qquad (2)$$

In [10], a notion of lost time is also tackled. However the authors did not consider neither the estimation in the pipeline optimisation nor the problem of

feedback loops. They do not propose a formulation of the problem because they notice that such a method can not detect errors due to strong and local requirements of parallelism. They finally choose a method based on a scheduling algorithm, as previously stated.

It is true that the problem of parallelism is not pointed out by the improved average number, this is the reason why we complete this by a computation of a parallelism rate. We will see that the two techniques must be associated to get the accurate estimation.

Minimum Parallelism Rate. The aim is now detect the parallalism requirements that impose the limitation of the mobility in the feedback loops. So, we now have to compute the minimum number of FU of type i, which must be active in the same time step, into the whole set of feasible schedulings associated with the operations of the feedback loops. The first idea consists of creating a density of parallelism for each operation, that, in a way, expresses the *needs* of FUs associated with the operation node. Thus, through adding these densities for each time step t common to operations located on different parallel branches of the DFG, it is possible to obtain the number of FU required. The density is defined (Eq. (4)), for a given operation node k of type i, as the inverse of the integer number of operations of type i that can be executed consecutively during the time frame: $[asap(k), alap(k)+\Delta(i)[$. In other words, the wider the time frame, the more it can contain the following intervals of length $\Delta(i)$, and hence the less the operation k will be constrained to be scheduled in the time step t.

There then remains the problem of knowing the densities that must be added in a given time step, without judging what the scheduling may be.

How and When Must We Add the Densities? The key problem is the sum of *needs* in FU linked to the operations that may be scheduled in a given date t. The principle is based on the addition of densities of operations that can be executed simultaneously. Actually, the sum can not be done on the whole [asap ; alap] interval for each operation, because in this case the overlappings of intervals from the following nodes lead to over-estimations. The difficulty associated to the overlappings intervals can be avoided. In fact, the over-estimation results from the addition of densities associated with operations that can not be executed in common time steps because they follow one another. However, it is important to retain the information on the whole $[asap(i), alap(i)]$ interval concerning a possible presence of the operation i or one of its predecessors or successors. Therefore, the solution for the avoidance of unfeasible superpositions of densities while covering the whole set of mobility intervals, consists in adding, in the given time step t, only the densities $1/Ps(i)$ of operations belonging to separate parallel branches of the DFG. In a given branch, the choice between a node and its predecessors and successors will be done by keeping the node with the greater density. In the case of nodes common to different branches, the density is normalized by the number of its successors having a density greater or equal to their common time frame (Eqs. (5) and (7) of the Algorithm 1). Figure 5 illustrates the method on a subgraph assumed to be inserted in feedback loops.

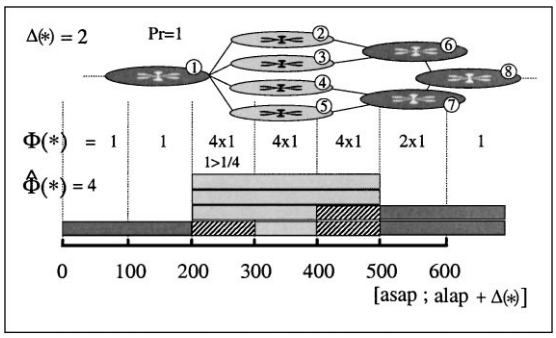

Figure 5. Minimum parallelism method. $\Delta(*)$: delay of $*$, $Pr = 1/Ps(*)$: density of $*$, $\Phi(*)$: minimum of parallelism rate for $*$, $\hat{\Phi}(*) = \max_t \Phi(*)$.

We have seen how the operation in a same branch of the DFG is selected. The other difficulty is linked to the choice of densities, belonging to parallel branches, that have to be taken into account. If we analyse the problem, we notice that the operations involved in local strong parallelism have a reduced mobility. Such operations possess less scheduling opportunities than the average of operations belonging to the same type. Therefore, a simple method for carrying out the selection of critical operations of type i consists of adding only the nodes in parallel branches that have densities greater than the average density of FU of type i (see Eq. (2)).

For instance, we can imagine, in the case of Fig. 5 that an other multiplication may exist in parallel with the operations 2,3,4,5. Assuming that an infinitely small density is associated with this extra operation. If all the parallel branches are taken into account, without distinguish critical nodes from the others, the

parallelism rate computed, would over-estimate the number of FU required (4). If the extra operation with lots of other scheduling opportunities than the only three dates 200, 300 and 400 is selected the exact parallelism is obtained.

$$\bar{P}r(i) = \frac{1}{\bar{N}'_r(i)} \tag{3}$$

where $\bar{N}'(i)$ is the integer just greater than $\bar{N}'_r(i)$.

The Algorithm 1 sums up the different steps of the computation of the minimum parallelism rate.

Algorithm 1. Computation of the minimum parallelism rate for FU of type i

- Definitions:
  - Let $Ps(O_k)$ be the potential of operations having the same type as O_k, that can be executed consecutively in the mobility interval of O_k;
  - Let $Nb_{\text{par}}(t)$ be the number of parallel branches in the time step t.
  - Let $\mathcal{I}_{i,j}(t)$ be the set of operations O_k of type i in the branch j whose interval $[asap(i), alap(i) + \Delta(i)[$ contains t.
  - Let $Ps_N(O_k) = Ps(O_k) * N_{\text{succ}}(O_k)$ where N_{succ} is the number of branches coming from O_k, belonging to the feedback loops and containing some operations O_p of type i, with a density $1/Ps(O_p)$ ower or equal to $1/Ps(O_k)$ until a date t' such as $t' < alap(O_k) + \Delta(O_k)$. *If an operation has a density $1/Ps(O_k)$ greater than those of its successors, the branches coming from O_k will use the normalized density until t' in order to avoid computing the density of a same node, several times*
- Increase the mobility of all nodes being in the feedback loops to the maximum;
- Compute the normalized potential associated to consecutive operations of type i in the DFG:

$$\forall O_k \in \mathcal{I}$$

$$Ps(O_k) = \left\lfloor \frac{alap(O_k) + \Delta(O_k) - asap(O_k)}{\Delta(O_k)} \right\rfloor \tag{4}$$

$$Ps_N(O_k) = Ps_N(O_k).N_{\text{succ}}(O_k) \tag{5}$$

- Filter the critical nodes for parallelism

$$\forall O_k \in \mathcal{I}_i$$

$$Pr_N(O_k) = \begin{cases} \dfrac{1}{Ps_N(O_k)} & \text{if } \dfrac{1}{Ps_N(O_k)} > \bar{P}r(i) \\ 0 & \text{else} \end{cases} \tag{6}$$

- Compute the minimum parallelism rate of FU of type i in step t, $\Phi_{N\text{par}}(t)$:

$$\Phi_{N\text{par}}(i,t) = \sum_{j=1}^{Nb_{\text{par}}(t)} \max_{O_i \in \mathcal{I}_{i,j}(t)} Pr_N(O_i(t)) \tag{7}$$

- Extract the global minimum parallelism rate:

$$\hat{\Phi}_{N\text{par}}(i) = \max_{t \in \mathcal{T}'} \Phi_{N\text{par}}(i,t) \tag{8}$$

Necessity of Use of Both the Improved Average Number and the Minimum Parallelism Rate. Two criteria must be taken into consideration in order to carry out the accurate number of FU. The first one, the improved average number, considers the real time allowed to operations of a given type, but can not detect the local necessities of parallelism. The latter computes the minimum parallelism rate, but considers for each operation the possibility of using the entire time frame $[asap(i), alap(i) + \Delta(i)[$, without otherwise taking care of the possible consequences on the time frames of the other operations that can be, thus, potentially reduced. For this reason, it is necessary to check, with the improved average number of operators, that the number of FU provided by the minimum parallelism rate is really sufficient to execute the whole set of operations. Such cases may appear with DFG that are not strongly constrained, as in the case of Fig. 6.

Consequently the number of FU required by the operation of type i in feedback loops is:

$$\bar{N}_{\text{opt}}(i)^{\mathcal{F}} = \text{MAX}\{\hat{\Phi}_{N\text{par}}(i);\ \bar{N}'(i)\} \tag{9}$$

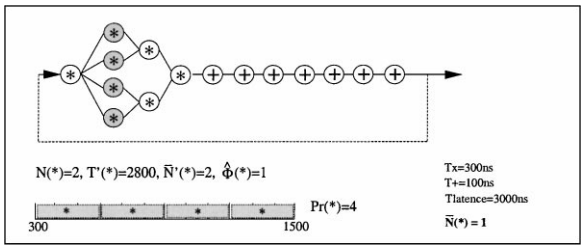

Figure 6. Case where the solution is given by the average number.

2.2.3. Estimation of the Number of FU in the Whole DFG

Problem Related to the Redistribution of Operations Located out of Feedback Loops. Following the estimation of FU in the feedback loops, there still remains the redistribution of operations located out of feedbacks loops in the free time steps. If required, pipeline stages are used.

If the global improved average number: $\bar{N}'^{\mathcal{G}}(i)$ is lower than the number of operators required by the feedback loop nodes, we know that the number of free time steps is quantitatively sufficient. However, these time steps can not always be used to execute the remaining operations. When these are split up into intervals of length, lower than the processing delay of the given type of FU, the redistribution is impossible. Therefore, the estimation may take into account the real available time for the operations located out of the feedback loops: $T''(i)^{\mathcal{G}}$, if this one is insufficient for the remaining operations, one or more other FU will be allocated. Figure 7 shows an example, where the redistribution of the three multiplications located out of loops is not feasible with the optimal number of multipliers computed for the operations inside the feedback loop ($\bar{N}_{\text{opt}}(*)^{\mathcal{F}}$), despite a global average number of multipliers $\bar{N}(*)^{\mathcal{G}}$ equal to $\bar{N}_{\text{opt}}(*)^{\mathcal{F}}$.

Principle of Complete Estimation. The method is similar to the previous one used for computing the number of FU in the feedback loops. The only difference is related to the particular processing of the mobility of the operations that are located out of loops. Actually, these operations may be scheduled anywhere in the interval $[0; T]$, thanks to pipeline transformations. Consequently, they are supposed to have a maximal mobility.

The method is described in Algorithm 2.

Algorithm 2. Computation of the global number of FU of type i required

- the mobilities of out of feedback loops operations are extended to [0 ; T]
- compute $N_{\text{opt}}(i)^{\mathcal{F}}$ with Algorithm 1
- compute $\bar{N}'(i)^{\mathcal{G}}$ with all operations of the DFG:

$$\bar{N}'(i)^{\mathcal{G}} = \frac{N\text{op}(i)^{\mathcal{F}}.\Delta(i) + T''(i)^{\mathcal{G}}}{T}$$

  ($T''(i)^{\mathcal{G}}$ and $T''(i)^{\mathcal{F}}$ are computed simultaneously)
- $N_{\text{opt}}(i)^{\mathcal{G}} = \max\{N_{\text{opt}}(i)^{\mathcal{F}}; \bar{N}'(i)^{\mathcal{G}}\}$

2.2.4. Computation of the Minimal Number of Pipeline Stages

Principle. The numbers of FU have been computed without any restrictions on pipeline. The aim is now to find the minimal number of pipeline stages that enable us to only use the minimal number of FU.

The pipeline transformation is required to modify the mobility [asap; alap] of the operations in order to associate them to free and usable time steps. This modification can only be done towards the future dates. It must keep the mobility of operations lower than T in order to use the estimation based on the improved average numbers and minimal parallelism rates. This condition will be respected by applying a recursive method that moves operation intervals without an assumption on scheduling. The initial mobilities are fixed by the absolute minimum of pipeline stages, computed with asap dates and respect of feedback loops scheduling constraints.

The principle is based on the parallelism rate that is locally (for an operation) compared to the optimal numbers of FU. Following this comparison, the possible extra time steps required, not exceeding the optimal number of FU, are passed on to the successors. If necessary the intervals of the successors are, consequently, brought forward. Thus, the whole set of operations intervals are recursively moved forward from the beginning of the DFT to its end. During recursion, the initial mobilities remain unchanged. When all the DFG is fully treated, the improved average number is computed to check the validity of the assumed available

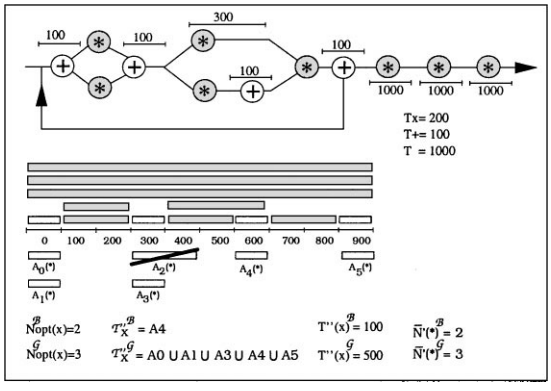

Figure 7. Unfeasible complete redistribution of outside loops operations.

time. Finally, the intervals of mobility are adjusted according to the final number of pipeline stages while respecting the feedback loop constraints. The accurate number of pipeline stages $Nb_{\text{pip}}^{\mathcal{G}}$ is finally obtained with the maximum on alap dates (Eq. (10)).

$$T_{\text{alap}}\ max = \max_{O_k \in \mathcal{G}} \{alap(O_k) + \Delta(O_k)\}$$
$$Nb_{\text{pip}}^{\mathcal{G}} = \left\lceil \frac{T_{\text{alap}} max}{T} \right\rceil \quad (10)$$

The critical task of the method resides in the recursive processing of the forward moves of the operation intervals. This point is detailed in the Algorithm 3.

Limited Pipeline Constraints. Some applications may impose constraints on the pipeline in order, for instance, to limit the response delay of signal processing. In such cases, the method remains unchanged. The algorithm (Algorithm 3) is applied until the limit on pipeline stages is not reached. If the process is stopped before the end, the number of FU required will not be optimal.

2.2.5. Optimization with the Multi-Delay Feedback Loops

Optimization Opportunities. A number of N_{D} delays in a feedback loop permits the distribution of operations

Algorithm 3: Recursive forward move of operation intervals for pipeline depth minimization

---

♦ definitions :
- $Tab_N[t,i]$: parallelism rate of FU of type i in the time step t.
- *adaptation_N* : updates $Tab_N[t,i]$ according to the previous forward moves.
- *compute_move(op)* : returns the number of extra time steps required to execute the current operation according to the resource constraints.
- *recursive_move(op)* : carries out the minimal moves of the intervals [asap ; alap[.
- influential : characterizes an operation which has the greatest normalized density in its branch, for a given date (cf. algo. 1).
- pipeline checking : the interval of *op* is moved forward if its length is lower than $D(op)$ in a pipeline stage.

♦ *compute_move(op)*
$\Delta N = 0$
for $t \in [asap(op)\ ;\ alap(op) + D(op)[$ do {
  if $Pr(op)$ *influential* in t {
    $\Delta N = \Delta N + (Tab_N[t \bmod T, i] - N_{opt}(i))\}$
  }
}
$t_1 = alap + D(op)\ ;\ t_2 = 0$
while $t_2 < \Delta N$ {
  if $Tab_N[t_1 \bmod T, (type(op)] > N_{opt}(type(op))$ {
    $\Delta N = \Delta N + 1$
  }
  $t_2 = t_2 + 1$
}
return ΔN

♦ *adaptation_N*$(Tab_N[t,i], Pr(op), \Delta t)$
for $t \in [asap(op)\ ;\ asap(op) + \Delta t[$ do {
  if $Pr(op)$ *influential* in t {
    $Tab_N[t \bmod T, i] = Tab_N[t \bmod T, i] - Pr(op)$
  }
}
for $t \in [asap(op) + \Delta t + 1\ ;\ alap(op) + D(op) + \Delta t[$ do {
  if $Pr(op)$ *influential* in t {
    $Tab_N[t \bmod T, i] = Tab_N[t \bmod T, i] + Pr(op)$
  }
}

♦ **recursive_move(op)** :
$\Delta t = \Delta t + \max_{k \in Pred(op)}$ *recursive_move(op)*
if $op \neq Nœud_{fin}$ {
  if $Pr(op) > \bar{P}r(type(op))$ then {
    if $\Delta t > 0$ then {
      adaptation_N $(Tab_N[t], Pr(op), \Delta t)$
      $asap(op) = asap(op) + \Delta t$ $alap(op) = alap(op) + \Delta t$
      pipeline checking
    }
    if $op \notin \mathcal{B}$ then {
      $\Delta t = \Delta t +$ *compute_move(op)*
    }
  }
}
return Δt

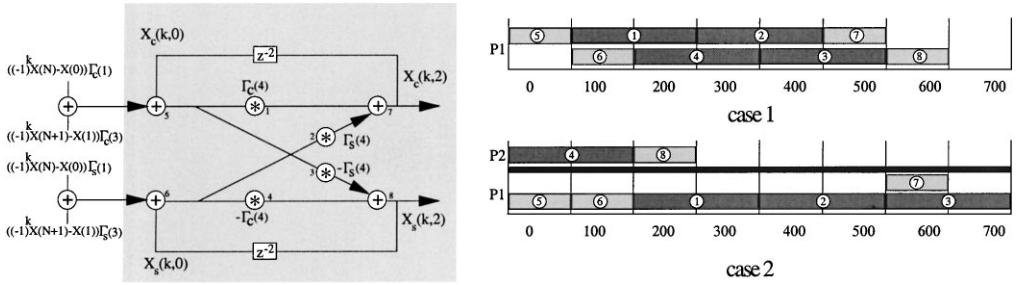

Figure 8. Multi-delays optimization example—case of a recursive DCT with block size = 2.

on N_D pipeline stages. This increase of mobility favors the optimization of the scheduling task. An example of recursive DCT [19] with two delays in feedback loop is given in Fig. 8. In this case, a multiplier is saved thanks to the utilization of the whole mobility. The techniques used to reduce the critical loop, by insertion of extra delays [20], increase algorithms complexity, therefore the optimization of scheduling may lead to significant hardware minimization.

Principle. The multi-delay opportunities in the feedback loop are taken into account in the system MARS during the scheduling task [21]. The authors use two kinds of mobility, in order to check the scheduling possibilities of a given operation: the mobility of the loop it belongs to and the intrinsic mobility of the operation. Our method is independent from scheduling techniques. The aim is to exhibit the mobility intervals in order to compute the required numbers of FUs. The method will not be detailed here (see [18]), we will just describe the main steps.

Numbers of FU. Firstly the DFG is broken into independent subsets. A subgraph contains nodes of inter dependent feedback loops. The maximal mobility of operations is then computed according to the subgraph they belong to.

When a operation belongs to different feedback loops, several solutions of pipeline stage sets may appear. Such cases are resolved by applying two criteria of choice. The most constrained loop (with the lowest number of delays) has got the priority, and the solution leading to the minimal number of pipeline stages is retained. The numbers of FU are computed in the same way as before. However, the result (Eq. (9)) can not be used if mobility lengths are greater than the time constraint T, whereas now mobility intervals may exceed T. This difficulty is overcome by using normalized densities:

$$Ps'(O_k) = \frac{1}{N_p} \cdot Ps(O_k)$$

where N_p is the number of pipeline stages where (O_k) may be scheduled and $Ps(O_k)$ the density computed without pipeline. This method means this simple fact that the usable time is N_p greater than previously.

Number of Pipeline Stages. Two methods may be used to compute the minimal number of pipeline stages. The first one is still valid, however the task of pipeline checking performed while the intervals are moved forward, requires particular attention.

A second one becomes feasible because the intervals of mobility are allowed to be greater than the time constraint T. It consists in increasing the number of pipeline stages from the minimal value until the FU bounds are reached. For each step, the mobility intervals are worked out. However, in the case of pipeline estimation, the subgraphs are not studied independently. The mobility of the entire graph is now limited by the current number of pipelined stages tested, and the dependences between subgraphs must be taken into account.

We prefer the second method in order to avoid the recursivity that may require a huge program stack in the case of an important DFG. As previously, the designer can impose a constraint on the pipeline stage number.

Example. Figure 9 gives some results obtained in the case of an IIR7 filter. The characteristics of the DFG *multi-delays*, *seven loops*, *four sets of loops* make it as very suitable benchmark for our estimation based en loop opportunities exploration. By increasing the time constraints, trade-off curves are performed for FU area and numbers of pipeline stages. A retimed version of

IIR7 - FU and Pip. Estimations under various Time constraints

Before retiming

T	400	500	600	700	800	900	1000	1100	1200	1300	1400	1600	1700	1800	3000	3500
Fu Cost	34	24	21	20	17	14	14	14	11	11	10	10	10	7	7	4
Pip.	5	4	3	2	2	2	2	1	1	1	1	1	1	1	1	1

After retiming

T	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	2900	3000	3100
Fu Cost	28	21	18	17	17	14	11	11	11	11	10	7	7	7	4	4
Pip.	4	4	4	3	3	2	2	1	1	1	1	2	1	1	2	1

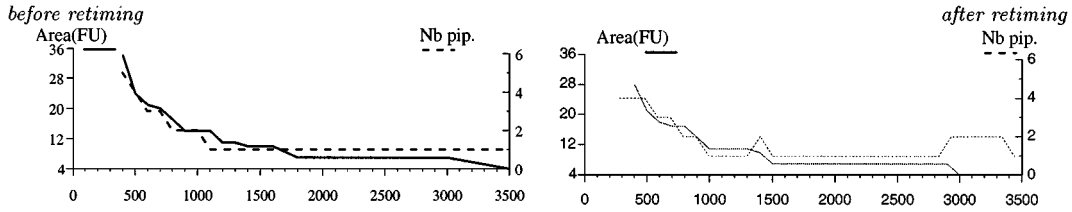

Figure 9. IIR7 trade-offs—Area and pipeline/time constraint.

the algorithm is proposed, we notice that the transformation enable a better use of pipeline in order to reduce area cost.

2.2.6. Conclusion. We have seen the method included in the GAUT framework to compute bounds on FU and pipeline numbers. The complexity of the estimation is lower than $O(N.\max_i\{alap(O_i) - asap(O_i)/\Delta t\})$. The method is independent from a given technique of scheduling, it is based on the two causes of error in resource estimation: the misappreciation of available time and the local requirements of strong parallelism. We will see in §3 how it is used to select the right components for estimation or synthesis. Using the right number of components and pipeline stages, the high level synthesis focuses on the minimization of both interconnection and controller cost during scheduling and binding tasks. Then a VHDL harware description is carried out for the logic synthesis tool [22].

2.3. Probabilistic Resource Estimation

2.3.1. Introduction. The first aim of the method is to provide the designer, with a dynamical estimation of cost in the interval [0 ; T]. The estimation is based on the probabilistic numbers of FU, registers, bus and interconnections in each time step. What is aimed isn't not an high accuracy of cost prediction that can't not be addressed at a high level of abstraction. The main issue is to be able to compare the effects of transformations (algorithmic/structural/fonctionnal) in order to make choices based on analysis and relative measures. Such metrics are handled in global design methodology [18] that associate estimations and transformations in order to guide the optimization of the *algorithme-architecture-adequacy*. While analysing the results and observing the effects of specifications choices, the designer gets keys to reduce the transformations space. The objective is also to complete the component selection choices by an estimation of registers, bus and interconnections complexity.

2.3.2. Principle

Functional Units. Given a DFG and a selection of components, we look for the probable number of FU of type n in the time step $t : N_i(t)$. Firstly, $\tilde{N}_n(t)$ the probable number of FU of type n *scheduled* t, may simply defined as:

$$\tilde{N}_n(t) = \sum_{i=0}^{\tilde{N}_{\max}} i.P_n(\tilde{N}_n = i, t) \qquad (11)$$

where $P_n(\tilde{N}_n = i, t)$ is the probability that i FU of type n are scheduled to the date t. Such a formulation would lead to a very complex computation. It can be demonstrated that Eq. (11) may be simplified as:

$$\tilde{N}_n(t) = \sum_{i \in E\,\mathrm{op}_n} P_{O_i}(t) \qquad (12)$$

where $E\,\mathrm{op}_n$ is the set of operations of type n and $P_{O_i}(t)$ the probability that the operation i would be scheduled in the time step t. After introducing the delay Δ of the associated FU, we obtain:

$$N_n(t) = \sum_{i \in E\,\mathrm{op}_n} \sum_{k=t-\Delta(0_i)+1}^{t} P_{O_i}(k) \qquad (13)$$

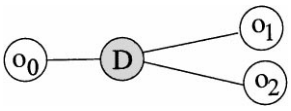

Figure 10. Data dependence between operations.

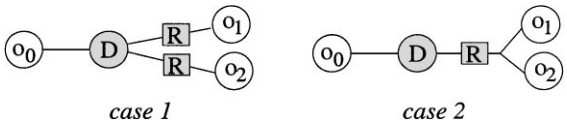

Figure 11. Without and with data sharing.

Resources for Data Transfers. The architectural model of GAUT (Fig. 2) enables three kind of data transfers:

- memory → FU (input data);
- FU → memory (results);
- FU → FU (temporary data). If the data life is greater than the *memory threshold*, the data is stocked in memory: FU/register/memory/register/FU, else the data remains in a register: FU/register/FU.

We will comment the estimation of registers for a data transfer between FU, in a particular case where a data D is produced by an operation O_0 and consumed by two operations O_1 and O_2 (see Fig. 10). A generalization will be done later.

Probability of Transfer. The scheduling probabilities of the producer and consumers are now both involved:

$$P_{\mathrm{R}}(t,i) = \begin{cases} P\{(O_0,t_0);(O_i,t_i)\} & \text{if } t \in I_{\mathrm{R}} \\ 0 & \text{else} \end{cases} \qquad i = 1,2 \tag{14}$$

Where $P\{(O_0,t_0);(O_i,t_i)\}$ is the probability that O_0 and O_i would respectively be scheduled in the time steps t_0 and t_i. I_{R} represents the duration interval where the data D is stocked in a register. Two cases may considered:

- $t_i - t_0 \geq$ *memory_threshold*: $I_{\mathrm{R}} = [t_0 + \Delta Op_0, t_0 + \Delta Op_0 + \Delta\mathrm{reg} + \Delta\mathrm{mem}] \cup [t_i - \Delta\mathrm{reg},\ t_i + \Delta Op_i]$ The data D is firstly written and then read in memory.
- $t_i - t_0 <$ *memory_threshold*: $I_{\mathrm{R}} = [t_0 + \Delta Op_0, t_i + \Delta Op_i]]$ D remains in the datapath.

The events, O_1 *is scheduled to* t_1 and O_2 *is scheduled to* t_2 are not independent, consequently the associate as a probability of transfer is defined as:

$$\begin{aligned} &P\{(O_i,t_i)\ ;\ (O_0,t_0)\} \\ &\quad = P((O_i,t_i) \mid (O_0,t_0)).P_{O_0}(t_0) \\ &\quad = P((O_0,t_0) \mid (O_i,t_i)).P_{O_i}(t_i) \end{aligned}$$

$$= \frac{p_{O_0}(t_0)}{\sum_{asap(O_0)}^{t_i - \Delta Op_0} P_{O_0}(t)} \cdot P_{O_i}(t_i) \tag{15}$$

The partial sums $S_0(t') = \sum_{t=asap(O_0)}^{t'} p_{O_0}(t)$ are constituted while $p_{O_0}(t)$ are computing.

Probable Number of Registers. The principle is identical to the FU's one, it is based on the sum of data transfers probabilities. However, it depends also on the use (case 2 of Fig. 11) or not (case 1 of Fig. 11) of data sharing optimization. In the case of Data Sharing, a single register (resp. bus) can store the data for the two successors:

Thus the probable number of register du to the data D is, without data sharing:

$$N_{\mathrm{R}}(D,t) = P_{\mathrm{R}}(t,1) + P_{\mathrm{R}}(t,2) \tag{16}$$

and with data sharing:

$$N_{\mathrm{R}}(D,t) = P_{\mathrm{R}}(t,1) + P_{\mathrm{R}}(t,2) - P_{\mathrm{R}}(t,1).P_{\mathrm{R}}(t,2) \tag{17}$$

The probable number of registers computed on the whole DFG is finally:

$$N_{\mathrm{R}}(t) = \sum_{D} N_{\mathrm{R}}(D,t) \tag{18}$$

Generalization. The previously described method is similar for the estimation of bus and interconnections between FU. The difference lies in the definition of associated intervals: I_{R}, I_{B}, $I_{\mathrm{C}[i,j]}$, where C$[i, j]$ designates a connection between the FU of types i and j. The method is finally generalized to transfers between an operation and any number of successors.

Computation Complexity. The complexity is higher, for the estimation of FU. Without data sharing, we get: $O(N.max_i^2\{(alap(i) - asap(i))/\Delta_t\})$. The square on time term may lead to slow estimations when the lengths of [asap ; alap] intervals increase. In such cases, the operations probability of scheduling in a given time

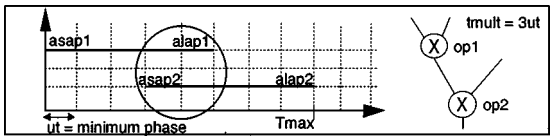

Figure 12. Unfeasible parallelism.

step become infinitely small. Therefore, an equivalent estimation is obtained by applying an interpolation. Considering a limited number of scheduling possibilities N_s, uniformly distributed in the intervals [asap ; alap], the estimation is computed with a maximal complexity equal to: $O(N \cdot N_s^2)$.

Scheduling Probabilities. The probabilities model could be uniform: $P_{O_i}(t) = 1/(alap(O_i) - asap(O_i) + 1)$ if $asap(O_i) \leq t \leq alap(O_i)$ and 0 otherwise. We did not retain such a distribution in order to avoid the wrong parallelism problems due to time frame overlappings (see Fig. 12). The distribution law we use, takes into account dependences on predecessors and successors. The scheduling probability of O_k in the time step t is computed according to the number of associated scheduling possibilities of the predecessors and successors that allow the scheduling of O_k in t. (Eqs. (19) and (20)).

$$P_{O_n}(t) = \frac{1}{N_0} \cdot \prod_{i=1}^{N_{\text{Pred}}} \max[1, t - (asap(p_i + \Delta(p_i)) + 1] \cdot \prod_{j=1}^{N_{\text{Suc}}} \max[1, asap(s_i) - (t + \Delta(O_n) + 1)] \tag{19}$$

with:

$$N_0 = \sum_{t=asap(O_n)}^{alap(O_n)} \prod_{i=1}^{N_{\text{Pred}}} \max[1, t - (asap(p_i + \Delta(p_i)) + 1] \cdot \prod_{j=1}^{N_{\text{Suc}}} \max[1, asap(s_i) - (t + \Delta(O_n) + 1)] \tag{20}$$

Where N_{Pred} and N_{Suc} are respectively the numbers of predecessors and successors. $P_{O_n}(t)$ and N_0 may be computed simultaneously. The complexity is linear with the number of operations N:

$$C < O(N.max_n\{(alap(n) - asap(n))/\Delta_t\} \cdot max_i\{N_{\text{Pred}}(i)\} \cdot max_j\{N_{\text{Suc}}(j)\})$$

2.3.3. Estimation Properties. The global cost distribution is obtained with :

$$C_{\text{total}}(t) = \sum_i \text{Cost}(FU_i).N_i(t) + \text{Cost(reg)}.N_R(t) + \text{Cost(bus)}.N_B(t) + \sum_{j,k} \text{Cost(interconnect)}.N_{C[jlk]} \tag{21}$$

In the design methodology the Eq. (21) is used, at the upper level, to perform algorithmic selection. For $C_{\text{total}}(t)$ like for the others curves, two main metrics are significant:

- The average of $N_X(t)$: means the lower bound of resource or cost of X.
- The irregularity of $N_X(t)$: means the difficulty to reach the minimal bound. In practice, the standard deviation is used to quantify the regularity of the resource distribution.

We associate the two values to get a global measure of complexity that is called Probable Cost. The estimation is independent from the algorithm involved in the high level synthesis tool. Consequently, the probable values do not return an exact estimation of the results obtained after synthesis. However, the metrics of average and regularity provide relative values that enable the classification, according to the cost, of the transformed specifications.

2.3.4. Examples. In Fig. 13 a table provides some estimations concerning each two versions of a given algorithm. Actually the issue of high level estimation is the comparison between different kinds of specifications. Each problem must then be analyzed in detail through the whole set of resource estimation. The Fig. 13(a) and (b) exhibit a possible use of estimation for algorithmic transformations. On Fig. 13(a) are given the curves of cost distribution in time steps [0 ; T] for successive retiming transformations of a 2nd order Volterra filter. Figure 13(b) provides the evolution of global cost from estimation and synthesis according to the consecutive transformations. While transformations are performed, we notice an associated regularity enhancement.

2.3.5. Conclusion. The probabilistic cost lies with a design methodology as an analysis tool. We take benefit from its properties to guide the designer in the transformations space including algorithmic, functional and

Algo.	Volt.		FFT		IIR7		Ellip.		Ad.Latt.		LMS		Fir16 1 2		Fast LMS	
	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b
Prob. Cost	3.14	2.91	23.6	21.2	17	13.5	8.1	7.2	22	13.4	5.9	11.7	11.9	8.6	13.8	11.7
GAUT	3.18	2.93	23	20.1	14.4	11.8	9.3	7.4	19.3	13.3	4.1	12.1	10.9	7.6	13.6	11.2

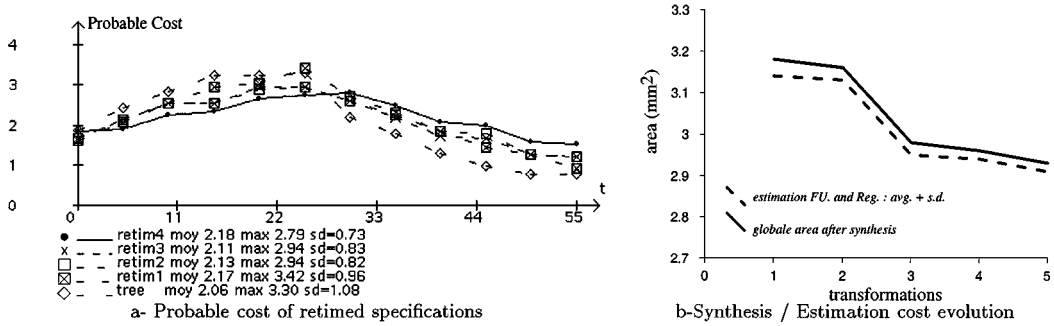

Figure 13. Cost evolution of a Volterra filter with successive retiming transformations.

structural transformations. We also use it to get metrics about interconnection for the hardware component selection.

3. Hardware Component Selection and Allocation

3.1. Introduction

The input requirements of the component selection are:

- A DFG built from the behavioral specification (VHDL) of the application;
- A library of specified components (VHDL) and a time constraint. For a given function, several FU are available with different area/delay trade-offs. The library also contains multi-functional units.
- A time constraint;

Using this data, the aim is to select the set of components that enable to respect the time constraint while minimizing the final cost architecture.

3.2. Example

We use a DIF2 algorithm to show how the selection of a multifunctional component may evolve with the time constraint. The Fig. 14 gives the selected components, it may be noticed that an Adder_Subtractor is not always required.

3.3. Method

The process is composed of four steps, in order to reduce the search space according to the computing complexity.

- Selection of a subset of components based on the average numbers. A reduced number of solutions is picked out with a branch and bound method. The following tasks will be performed on the subset solutions.
- The optimal numbers of FU and pipelines are computed.

FFT 32pts	Time constraint			
Components	4500ns	4200ns	3750ns	2200ns
mult.	2	3	3 (pip)	4
add.	1	/	1	1
sub.	1	1	2 (fast)	1
add/sub	/	1	/	1

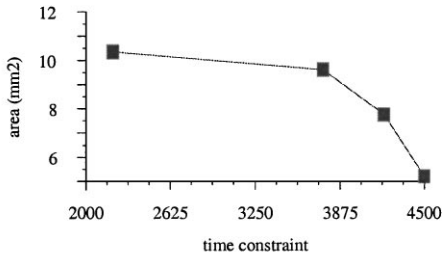

Figure 14. FFT DIF2—multifunction selection.

- Multifunctional optimization is performed for underused FU. (see [17])
- The estimation of numbers of registers, interconnections and bus are computed for the subset solutions. The metrics used to carry out the final selection are:

$$\bar{N}_{\mathrm{R}} = \frac{1}{T_r} \cdot \sum_{t=1}^{T_r} N_{\mathrm{R}}(t)$$
$$\bar{N}_{\mathrm{B}} = \frac{1}{T_r} \cdot \sum_{t=1}^{T_r} N_{\mathrm{B}}(t) \qquad (22)$$
$$\bar{N}_{\mathrm{C}} = \frac{1}{T_r} \cdot \sum_{t=1}^{T_r} N_{\mathrm{C}}(t)$$

4. Memory Unit Cost Estimation

Recent works [23, 24] have shown that during the system design, the memory can take a considerable part of the system cost. This is particular true in digital signal processing because the quality criterion for DSP applications is obtained either by suggesting new algorithms (mostly with a block vector computing), or by extending the response size of known filters. In the first case, it is necessary to evaluate the implementation for these algorithms by a prototyping stage. In the second case, the processing unit evolves little, however, the number of data to store rise considerably, in this condition, ASIC could be impossible to achieve. So it becomes important to develop some tools which allow us to estimate the memory cost before the synthesis step. To be interesting, these tools must give a good estimation in a very small computation time. If this condition is satisfied, then it is possible to explore numerous design solutions for an application.

4.1. Previous Works

To estimate accuracy the cost of the system implementation, we need to take into account the memorization problem. In literature, we can find many articles concerning these problems. Generally, these articles suggest the hardware solution which strongly depends on the data types used in the description and the memory types used in the library. Most of these articles deal with the synthesis of a particular memory organization which are detailed in the following points: (1) The allocation problem of the storage units for scalar or vectorial data [25, 26]; (2) The problem of address generation [27, 28]; (3) The hierarchical organization (to ensure the rhythm of the processor access) [29, 30].

These previous points are not very interesting for the estimation problem because the cost of the solution is known after you have built the hardware. On the other hand, some studies give methods to compute a memory estimation.

In [30], a memory selection technique is presented. It is based on specific memory unit organization which is based on four possible methods. A horizontal selection which associates several memory units, ensures that the width of the data corresponds to the width of the memory selected. A specific memory size can be achieved through vertical selection. Furthermore, to increase the transfer frequency, a selection of interleaved memories can be used. Finally, another one can virtually increase the number of memory ports.

The selected set of memories is a good estimation of the memory unit complexity. In [31], the suggested method enables them to provide the number and the type (RAM or ROM), the word width, and the number of ports of each memory. The selection rests on the evaluation of a silicon-area-cost function.

In [32], some High Level Transformations are presented. The goal consist in making transformations on the algorithm loops in order to try to minimize the data stored and thus, the number of transfers to make between processing and memory units. These two parameters are good estimators of the memory complexity. The methods revealed are based upon data structures of matrix form and suggest transformations with a view to reduce the number of memory points and transfers.

In [33], a technique of storage estimation for arrays of signals is presented. An integer linear formulation is given to compute an estimation of the number of storage locations. Also, in [23], the technique presented allow to obtain a evaluation of memory area for a chip. It is based on data flow analysis and give informations about total number of memory cells.

4.2. Our Approach

The memory estimation takes place after the processing unit estimation as showed in Fig. 1. In this context, the constraint we have to ensure is defined as an access sequence (see Fig. 15). The methodology we have developed consists of three estimation levels: the first level computes the number of buses for implementation between PU and MU; the second computes the number of necessary memory banks needed to store all the data; and the third level selects, in a library, a set of memory banks. The first two levels are not accurate

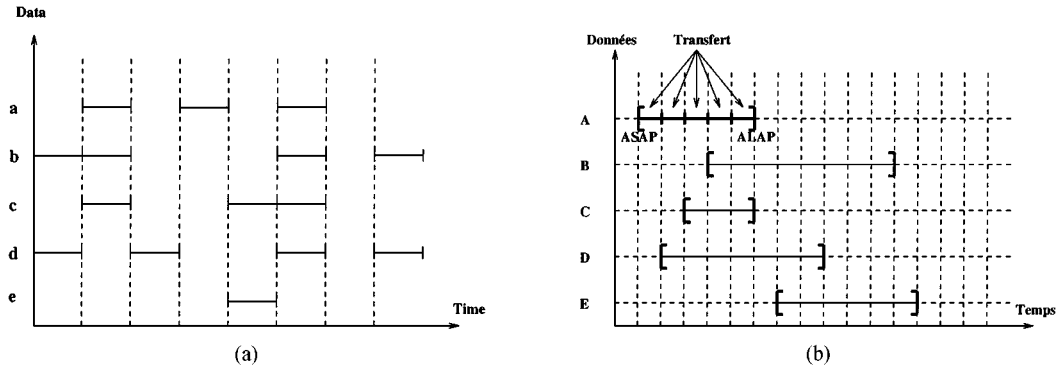

Figure 15. (a) Access sequence example (b) Access interval between PU and MU.

enough to evaluate correctly the MU cost. On the other hand, the third level give a good estimation of the MU area.

To compute the number of banks, we first need to compute the number of buses implemented between PU and MU, this step is called the smoothing task.

4.2.1. Smoothing of the Access Sequence. The processing unit estimation gives a list of access-time intervals which represent the possible dates for each access. This step consists in computing an estimation of the best dates for each access (see Fig. 15). This estimation gives the number of buses between PU and MU. The ideal goal of the smoothing operation is to produce a constant distribution of the number of transfers for all the possible access times. On the Fig. 16(a), we can see a bad distribution of accesses because, to ensure this sequence, it is necessary to implement 6 buses between the PU and the MU. On Fig. 16(b), we can see a good distribution of the same sequence, only 3 buses are implemented for the same total number of transfers.

Mathematical Smoothing Formulation. Hypothesis: We consider that all data are transferred just once. When the data is transferred N times, we create new data, $N-1$, with on transfer for each.

Let A_{it} be integer variable to indicate if the ith data is transferred at the time t.

Let ASAP_i be variable to indicate the "as soon as possible" date for the transfer of the ith data.

Let ALAP_i be variable to indicate the "as late as possible" date for the transfer of the ith data.

The smoothing operation consists of the minimization of the number of accesses for each access date:

$$\text{Min}\left(\sum_{i=1}^{\text{ND}} A_{\text{it}}\right) \quad \forall t = 1, \ldots, \text{NA}$$

$$\text{Constraint 1: } \sum_{t=1}^{\text{NA}} A_{\text{it}} = 1 \quad \forall i = 1, \ldots, \text{ND}$$

$$\text{Constraint 2: } \forall t \mid A_{\text{it}} = 1 \Longrightarrow \text{ASAP}_i \leq t \leq \text{ALAP}_i$$

With ND as the number of data and NA as the number of access time.

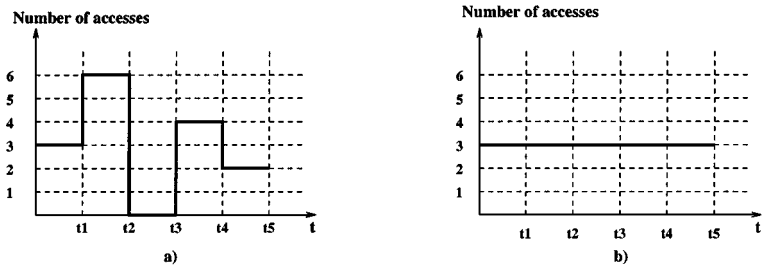

Figure 16. Examples of transfer sequences.

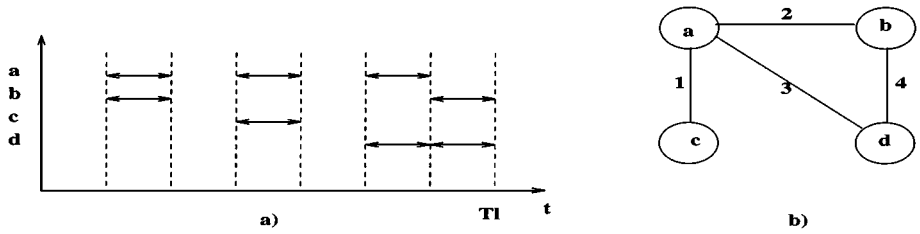

Figure 17. Example of building a conflict graph from a transfer sequence.

Constraint 1 ensures us that all the transfers are completed, and constraint 2 ensures us that the date of each transfer is in the ASAP-ALAP interval. The problem could be formulated as follows:

$$\text{Min}\left(\sum_{t=1}^{\text{NA}}\sum_{i=1}^{\text{ND}}|A_{\text{it}}-\overline{\text{TNA}}|\right)$$

$$\text{With } \overline{\text{TNA}}=\frac{\text{ND}}{\text{NA}}$$

$$\text{Constraint 1: } \sum_{t=1}^{\text{NA}}A_{\text{it}}=1 \quad \forall i=1,\ldots,\text{ND}$$

$$\text{Constraint 2: } \forall t \mid A_{\text{it}}=1 \Longrightarrow \text{ASAP}_i \le t \le \text{ALAP}_i$$

With TNA as the total number of accesses of the algorithm, and $\overline{\text{TNA}}$ as the average number of accesses per time interval.

Because of the absolute sign, this formulation is non linear, so it is impossible to solve it with the classical ILP (Integer Linear Programming) techniques.

This problem is similar to a data-flow-graph scheduling, so it is solved width a scheduling algorithm under the constraint of minimum parallel accesses.

4.2.2. Compute the Number of Memory Banks. Here, the important factor is to ensure the coherence of data and the transfer simultaneity for all the access sequence. The coherence of data is defined using the following three points:

- temporal coherence: two data which are transferred at the same time must not be found in the same memory bank;
- spatial coherence: the reading and writing of the same data must be carried out in the same memory bank;
- functional coherence: in the case of a pipeline structure, the writing at a pipeline stage must not erase data which are still of use for other pipeline stages.

We have used a graph $\mathcal{G}\{\mathcal{N},\mathcal{A}\}$ to represent the conflicts between data, (see Fig. 17(b)). The set of vertices N is made up of the set of data to be memorized ($N=\{a,b,c,d\}$); whilst the set of edges A, represents the conflicts between data ($A=\{1,2,3,4\}$). There is conflict between data during a discrete time if these data are simultaneously transferred. Should this be the case, an edge is created between the two vertices. The graph of Fig. 17(b) is therefore directly built from the sequence of Fig. 17(a). The research concerning the minimum number of necessary banks N_{B} to memorize the set of data is equivalent to the research of the chromatic index of graph $\mathcal{G}$. It involves researching the minimum number of colours needed to colour the vertices of the graph $\mathcal{G}$, under the restriction that two vertices connected by an edge cannot carry the same colour. The colouring problem of graph $\mathcal{G}$ is equivalent to a clique partitioning problem of the complementary graph $\overline{\mathcal{G}}$ (a clique is a sub graph, as each couple of vertices from this sub graph are connected by an edge).

The solution to this problem thus concerns isolating the most of data as possible, as each pair of isolated data incorporates an edge between the two. We can express such a calculation in the following way:

$$\bullet\ \text{Maximization} \quad \text{NB}=\sum_{i=1}^{\text{ND}}\text{DM}_i$$

$$\text{Constraint:} \quad \forall i,\ j \mid \text{DM}_i=\text{DM}_j=1 \Rightarrow C_{ij}=1$$

With DM_i as the variable that indicates if the ith data is isolated. The objective function NB is correct from the point of view of its ILP expression. On the other hand, the constraint must be expressed by way of a linear inequation. The constraint $\forall i,\ j \mid \text{DM}_i=\text{DM}_j=1$ means that *each pair of the two variables i and j belong to the biggest group if, and only if, there is a conflict between the two variables*. This constraint can not be expressed in the form of inequality, however, we can explain the complementary constraint: *each pair of two*

variables does not belong to the largest clique, if there is no conflict between the two variables.

$$\mathrm{DM}_i + \mathrm{DM}_j \leq 1 \quad \forall i, j \mid C_{ij} = 0$$

The problem involving the research of the number of memory banks is shown in the following way:

$$\bullet \text{ Maximize} \quad \mathrm{NB} = \sum_{i=1}^{\mathrm{ND}} \mathrm{DM}_i$$

$$\text{Constraints:} \quad 0 \leq \mathrm{DM}_i \leq 1 \quad \forall i$$

$$\text{Constraints:} \quad \mathrm{DM}_i + \mathrm{DM}_j \leq 1 \quad \forall i, j \mid C_{ij} = 0$$

The resolution of this problem can be achieved by three types of methods:

- exhaustive: although the complexity is usually important, some heuristics allow us to limit the research space;
- maximization or minimization: from the ILP formulation, research by a simplex method supplies good results, however the calculation time is long and often unsatisfactory in an algorithm prototyping stage;
- linear: this type of research is non-optimal, but quickly supplies an evaluation of the complexity.

The method developed to carry out the number of banks is linear.

The calculation times necessary to obtain the solution are relatively small.

4.2.3. Selection of Memory Components. To have a fast estimation of the memory cost, we suggest a selection of memory components for a fixed model of organization (Fig. 18). This selection, in turn, consists of searching through a library for the memory set best suited to the data storage, all according to the requirements expressed by the PU estimation. We can notice that when we reach the best selection, the higher the number of memory type is, the higher the number of explored solutions is.

This model is composed of NB memory banks associated with NG address generators (with NG $\leq$ NB). The memory unit can be made up of varying time access memory.

In the first approach, the types of data and memory are ignored (note that these types could be taken into account with some small modifications). The problem

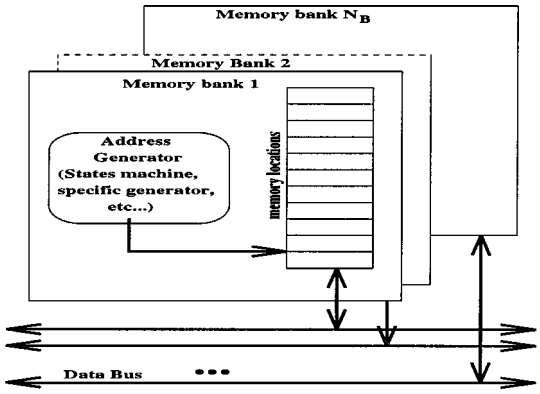

Figure 18. Memory unit model.

is to select a set of minimum number of memories, which can be formulated by the following expressions:

$$\mathrm{Min}\left(\sum_{i=1}^{\mathrm{NM}} S_i\right)$$

$$\text{Constraint 1:} \quad \sum_{i=1}^{\mathrm{NM}} S_i \geq \mathrm{NB}$$

$$\text{Constraint 2:} \quad \sum_{i=1}^{\mathrm{NM}} S_i \cdot \mathrm{NP}_i \geq \mathrm{ND}$$

$$\text{Constraint 3:} \quad \sum_{i=1}^{\mathrm{NM}} S_i \cdot \frac{T_L}{TA_i} \geq \mathrm{TNA}$$

With NM the number of memories present in the library, NP_i the number of memory points in the ith memory bank. Constraint 1 ensures transfer parallelism and data coherence. The first is obtained by a simple analysis of the transfer sequence, while the second calls for the research of a number of memory banks (N_B) which is done before.

Constraint 2 ensures the storage of all data. In the first step, each data needs a memory location. This can be optimized if one envisages the sharing out of memory locations by several data which have disjointed life times.

Constraint 3 ensures that all the transfers can be done.

The number of available access times in a memory bank during latency time is represented by T_L/TA_i. The total amount of possible accesses is therefore the sum of all possible accesses in the selection.

Feasibility of Internal Memory. For a given technology and foundry, the internal memory must not be

larger than a specific size. This maximum size is illustrated by the product $NumberOfBits \cdot NumberOfWords$. Thus, we can explain a constraint for internal memory, see constraint below.

$$\forall i = 1, \ldots, N_{\mathrm{MI}} \Longrightarrow \mathrm{NP}_i \cdot NBits_i \leq Max_i$$

4.2.4. Conclusion. The memory estimation we have developed is based on memory selection. This method gives a good estimation of the MU area based on a realistic memory component (because the component is selected in a silicon-foundry library).

After this estimation, it could be possible to start the synthesis step of the MU. The main tasks of the synthesis are: the data distribution in memory banks, the data placement in each memory bank and the synthesis of the address generators (for more details, see [34]).

5. Area/Time Trade-offs of New Fast LMS Filters

5.1. Introduction

The GAUT framework provides two ways of achieving trade-off curves: with or without achieving architectural synthesis. While performing estimation and selection tasks we obtain quick results, without building the architecture. This solution is independent from the scheduling and binding algorithmic choices. The following results have been obtained by this way. The number of Fu has been obtained through the optimal estimation described in §2.2. The estimation of interconnect cost has been performed through a probabilistic estimation. Finally, the memory cost estimation has been carried out.

The aim of this study is to estimate the cost of different algorithms for a given time constraint. The iteration frequency is fixed to 16 kHz, while modifying the number of filter taps associated to the specification of the application. Therefore, we finally obtain Cost/(DSP quality) trade-offs curves for each kind of tested algorithm.

5.2. Description of the Application.

The application fits in the context of acoustic echo cancellation, for which the large size of finite impulse response requires filters about 1000–4000 taps. The computational load of such applications justifies the efforts spent to reduce the arithmetic complexity, our goal is to test, through estimation in high level synthesis, the efficiency of some recent methods in a VLSI context.

The LMS algorithm may be turned into updating and fixed filtering parts. For the updating task, we use the method described in [35] for the FELMS algorithm that keeps the convergence properties of the LMS while allowing significant reduction of the computational complexity. The fixed part uses the fast FIR filtering described on [36], it is composed of short length FIR sub-filters derived from the original FIR algorithm with the Chinese Remainder Theorem (CRT). Contrary to the block versions, these kind of fast algorithms generate a processing delay independent of the filter length (see [37, 38]). The four kind of algorithms will be detailed.

5.3. Description of New-Fast LMS Algorithms

5.3.1. Reduction of Arithmetic Complexity. The first algorithm is the classic LMS in the FIR case. The principle of fast filtering applied to the fixed part of the LMS is a transformation of the FIR filter equation into two finite degree polynomials computed by the CRT. It decomposes the computation into three main parts. The first part is concerned with the interpolation of the polynomial products in different points deriving different algorithms in the same $F(N_i, N_i)$ class [37]. The second is the filtering operation and the final one is the reconstruction of the filtered signal. An algorithm of the class $F(N_i, N_i)$ applied to a FIR of length L computes N_i outputs in parallel using m_i sub-filters of length L/N_i. Despite pre and post-processing operations, it results in a significant reduction of the arithmetic complexity. By repeating the process [37] on sub-filters, the length of the final filters, and consequently the arithmetic complexity, can be considerably reduced.

The algorithms discussed in this paper will be two $F(2, 2)$ algorithms based on different interpolation points and a $F(3, 3)$ algorithm.

5.3.2. F(2, 2) - First Version. The first version of the $F(2, 2)$ is built by choosing the following interpolation points in the pre-processing process: $\{0, 1, \infty\}$. If the initial FIR filtering is defined as:
$Y(z) = H(z).X(z)$, we obtain the following equations:

Filtering part:

$$\begin{cases} Y(z) = (Y_0 + Y_1.z^{-1}) \\ X(z) = (X_0 + X_1.z^{-1}) \\ H(z) = (H_0 + H_1.z^{-1}) \end{cases}$$
$$\begin{cases} Y_0 = X_0.H_0 + X_1.H_1.z^{-2} \\ Y_1 = (X_0 + X_1)(H_0 + H_1) - X_0.H_0 - X_1.H_1 \end{cases} \quad (23)$$

Updating in accordance with the FELMS algorithm:

$$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n+1)} = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n-1)} + \mu \begin{bmatrix} X_0^t(e_n - e_{n-1}) + (X_0 + X_1)^t e_{n-1} \\ -X_0^t(e_n - e_{n-1}) + (X_1.z^{-2} + X_1)^t e_n \end{bmatrix} \quad (24)$$

Equation (24) has been modified to reduce its complexity: two terms are common to H_0 and H_1 and the sum of X_i result from the filtering part.

The initial filtering requires L multiplications and additions, the $F22$ algorithm needs $3L/4$ multiplications and additions for the filtering part and $2 + (3/2)(L/2 - 1)$ additions/subtractions for the pre and post processing parts. The reduction of arithmetic complexity is therefore about 25%. The Fig. 20 depicts the structure of the algorithm.

5.3.3. F(2, 2) - Second Version. The second version of the $F(2, 2)$ uses an other set of interpolation points: $\{0, -1, \infty\}$. It results in the following equations:
Filtering part:

$$\begin{cases} Y_0 = X_0.H_0 + X_1.H_1.z^{-2} \\ Y_1 = X_0.H_0 + X_1 + H_1 - (X_0 - X_1)(H_0 - H_1) \end{cases} \quad (25)$$

Updating in accordance with the FELMS algorithm:

$$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n+1)} = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n-1)} + \mu \begin{bmatrix} X_0^t(e_n + e_{n-1}) - (X_0 - X_1)^t e_{n-1} \\ X_0^t(e_n + e_{n-1}) + (X_1.z^{-2} - X_1)^t e_n \end{bmatrix} \quad (26)$$

We also have two terms which are common to the computation of H_0 and H_1 and the subtraction of the X_i terms results from the filtering part.

The reduction of arithmetic complexity is equal to the previous one: 25%

5.3.4. F(3, 3). The $F(3, 3)$ algorithm computes three parallel outputs. In order to avoid complex interpolations points, it is carried out by applying the $F(2, 2)$ algorithm twice. It results in the following equations:
Filtering part:

$$\begin{cases} Y(z) = \left(Y_0 + Y_1.z^{-1} + Y_2.z^{-2}\right) \\ X(z) = \left(X_0 + X_1.z^{-1} + X_2.z^{-2}\right) \\ H(z) = \left(H_0 + H_1.z^{-1} + H_2.z^{-2}\right) \end{cases} \quad (27)$$

$$\begin{cases} Y_0 = (X_0.H_0 - X_2.H_2.z^{-3}) \\ \quad +((X_1 + X_2)(H_1 + H_2) - X_1.H_1)z^{-3} \\ Y_1 = ((X_0 + X_1)(H_0 + H_1) - X_1.H_1) \\ \quad -(X_0.H_0 - X_2.H_2.z^{-3}) \\ Y_2 = ((X_0 + X_1) + X_2)(H_0 + H_1 + H_2) \\ \quad -((X_0 + X_1)(H_0 + H_1) - X_1.H_1) \\ \quad -((X_1 + X_2)(H_1 + H_2) - X_1.H_1) \end{cases} \quad (28)$$

Updating in accordance with the FELMS algorithm:

$$\begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}_{(n+1)} = \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}_{(n-2)} + \mu \begin{bmatrix} X_0^t(e_{n-2} - e_{n-1} + e_n) - \\ \quad (X_0 + X_1)^t(e_{n-2} - e_{n-1}) + \\ \quad (X_1 + X_2)^t e_{n-2} \\ -X_0^t(e_{n-2} - e_{n-1} + e_n) + \\ \quad (X_0 + X_1)^t(e_{n-2} - en_1) - \\ \quad (X_2.z^{-3} + X_0)(e_{n-1} - e_n) + \\ \quad ((X_2.z^{-3} + X_0) + (X_0 + X_1))^t e_{n-1} \\ X_0^t(e_{n-2} - e_{n-1} + e_n) + \\ \quad (X_2.z^{-3} + X_0)^t(e_{n-1} - e_n) + \\ \quad (X_1.z^{-3} + X_2.z^{-3})^t e_n \end{bmatrix} \quad (29)$$

The complexity is now a about a third less than the original. Equation (29) has also been formulated to enable the re-use of terms, shared by the equations of the H_i and the additions of the X_i, and previously computed in the FIR part (Eq. (28)).

5.4. Results and Conclusions

The four algorithms seem to have the same memory requirements: $2.L$ memory points for the samples vector and the filter coefficients. However we note in the faster the algorithm is the higher the memory cost is. The extra memory cost is due firstly, to the M new samples

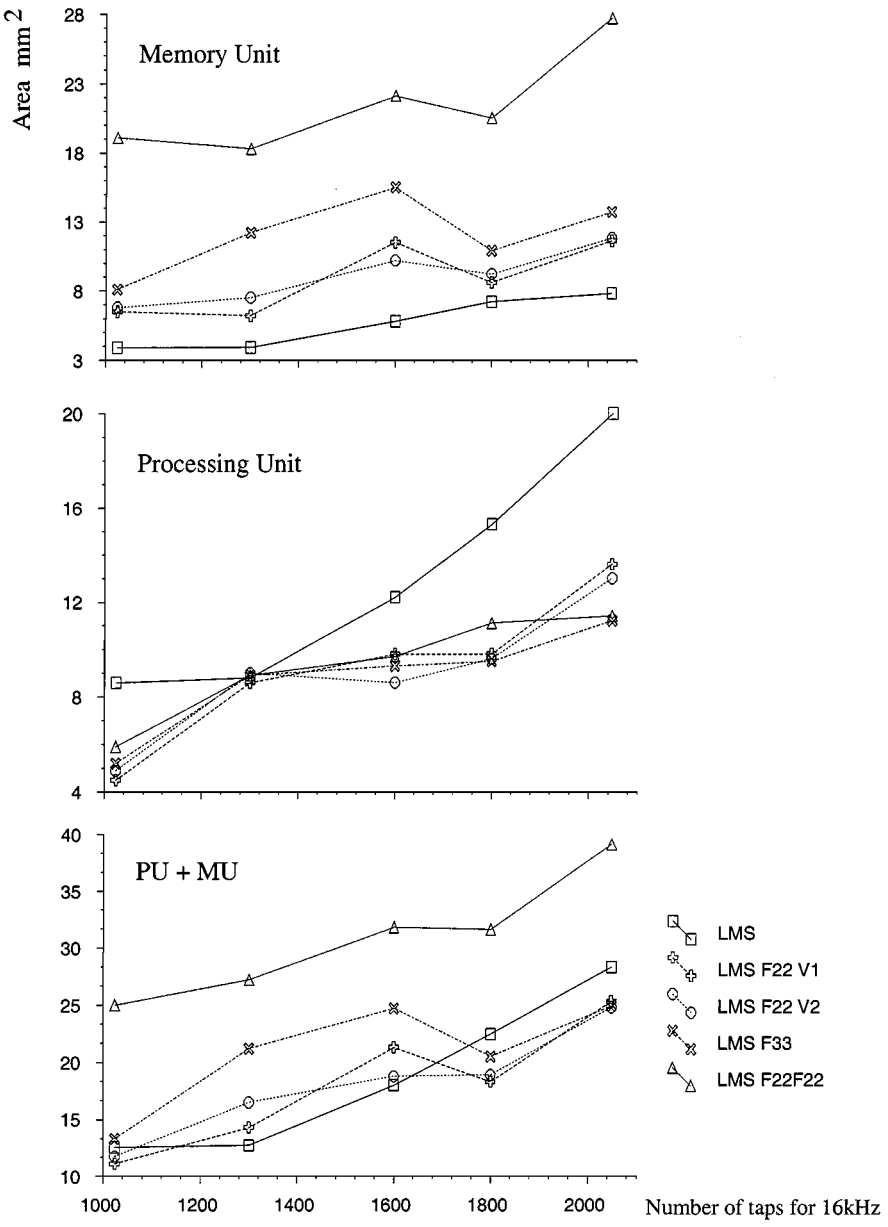

Figure 19. Structure of fast FIR F22-version 1.

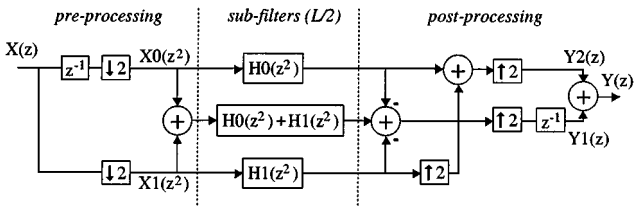

Figure 20. Trade-offs Area/perfomance of LMS algorithms.

vectors of length L/N that have be, in practice, created in order to compute in parallel the M sub-filters. In [38] this problem is discussed for the implementations of such fast FIR filters on DSP, the authors explain that the number of pointers registers is proportional to the number of FIR sub-filters. An efficient methodology is developed to reduce this number of memory registers. Otherwise, the pre and post-processing task generate lot of temporary data that must be stock in memory or registers.

The memory cost is not the only responsible of the extra cost observed. For instance, we note that the *LMS-F*33 datapath becomes less expensive than the *LMS-F*22 datapath only for a number of taps equal to 3072, whereas it should have always been cheaper. In fact, the memory transfers also modify the datapath cost, because they generate mux, demux and tristate. When the data transfers become important, the cost of these small components, often neglected in High Level Synthesis, result in a significant extra cost.

Let note that the reduction of number of operations is not always related to a reduction of the number of components, in High Level Synthesis. If a fast algorithm improves the rate of multiplications by unit of time from 2.5 to 2, it saves 20% of operations but 0% of multipliers. Moreover, a fast algorithm often breaks partially the regularity of the initial algorithm, the consequence on the architecture is an increase of the numbers of registers, mux, demux and tristate required.

In fact the efficiency of the algorithmic reduction of arithmetic complexity depends strongly on the rate between the initial number of operations and the thoughput constraint. In the case of a filter length equal to 1500, we observe that the reduction of the number of multiplications is not sufficient to save enough multipliers to make up for the increase of memory and interconnection resources. Consequently the classic LMS gives the smallest area.

The results show that algorithmic transformations in High Level Synthesis are essential to guide the designer towards a good specification choice, for a given case study.

Finally, we observe that the fast algorithms give the best design three times out of four, despite of the increase of memory resources. The advantage of the second version of the *F*22 algorithm is due to better balance between the number of additions and substractions that make easier the overall synthesis.

To conclude we can say that after estimation, its up to the designer to make a choice. According to quality of DSP he needs for his application, he will set a minimum number of taps. Giving this constraint he will, by observing the results, choose one of the proposed algorithms. For instance, if a high quality is required, he will take a 2048 taps LMS for which the algorithm LMS-F22-v2 provides the lowest cost.

6. Conclusion

In this paper we present a global framework that enables the prototyping of applications. The complexity of the trade-off exploration requires the use of different techniques that depend on the needs of the user. The area/time trade-offs are obtained while repeating the estimation with different time constraints. For a given signal processing application, this method enables to estimate quickly various algorithms, thus DSP quality/area/time trade-offs are produced.

Probabilistic cost estimations focus on the analysis of cost distribution within a constrained number of time steps. The main issues of the method are the dynamic cost estimation based on conditional probability law that tackles real dependencies, the computation simplification that allows fast estimations and the estimation of the whole datapath resources: FUs, registers, bus and interconnections.

The results analysis is involved in an estimation/ transformations methodology. The aim is to guide the designer to find out the best specification while applying algorithmic/data-flow/functional transformations in order to correctly distribute resource uses and to flatten the cost curve for the designer.

The other datapath estimation is dedicated to the following synthesis optimization. The originality of the method is firstly the dual computation of minimal numbers of FUs and associated Pipeline stages. Secondly, it proposes an association of a new parallelism rate, that avoids usual over-estimations, and an improved FU average number that rejects unusable time units. Finally, the method addresses the entire mobility opportunities which allow the number of delays in feedback loops. The knowledge of operation-node mobilities and the number of FUs enable the reduction of the synthesis complexity. Consequently, the synthesis task principally addresses the minimization of the interconnect and controller costs during scheduling and binding tasks.

The two kind of estimations are involved in a global hardware component selection.

The memory unit estimation is presented to complete the evaluation of the ASIC cost. For a more accurate estimation, the estimation processing is decomposed on three levels. The method is based on the mobilities of the read/write operations. The probabilistic cost estimation give us the ASAP and ALAP times for the access operations. These times are used to evaluate the minimum number of memory banks that we need to implement. After that, our tool selects in a library the best set of memory components for the storage of all data.

This method gives a good and a fast estimation of the MU area based on realistic memory components. The number of memories in the library is very important. Indeed, the higher the number of memory component is, the better the solution could be. Moreover, the memory can be internal or external.

A real life application has been studied in the case of accurate estimations applied to cost/DSP quality tradeoffs (see §5). The results show how a designer can use the estimation framework to carry out a specification choice. This estimation is not linked to any synthesis tool, and then can guide the designer in the optimisation of his application implementation by using transformations.

Only three axes of the complete design space exploration framework have been presented here. The methodology is now extended to two other criterions. The first one addresses the estimation and optimization of ASIP and ASICs power consumption [2]. The second one is related to the optimal choice of FU bit lengths in order to respect SNR constraint [3].

References

1. P. Lippens, V. Nagasamy, and W. Wolf, "Cad Challenges in Multimedia Computing," in *IEEE Int. Conf. on Computer Aided Design*, 1995.
2. N. Julien, E. Martin, S. Gailhard, and O. Sentieys, "Area/Time/Power Space Exploration in Module Selection for dsp High-Level Synthesis," in *Int. Workshop PATMOS'97*, Louvain-la-Neuve, Belgique, 1997, pp. 35–44.
3. J.M. Tourreilles, C. Nouet, and E. Martin, "A Study on Discrete Wavelet Transform Implementation for a High Level Synthesis Too," in *Eusipco'98*, Sept. 1998.
4. "Breizh Synthesis System Framework," Tech. Rep., ENSSAT—Univerisity of Rennes, `http://archi.enssat.fr/bss`, 1998.
5. D. Chillet, J.P. Diguet, J.L. Philippe, and O. Sentieys, "Memory Unit Design for Real Time dsp Applications," Submited to *Integrated Computer-Aided Engineering*, 1996.
6. E. Martin, O. Sentieys, H. Dubois, and J.L. Philippe, "Gaut, an Architectural Synthesis Tool for Dedicated Signal Processors," in *EURO-DAC*, Hamburg, Oct. 1993, pp. 85–94.
7. E. Martin, O. Sentieys, and J.L. Philippe, "Synthèse Architecturale de Coeur de Processeurs de Traitement du Signal," *Techniques et Sciences Informatiques*, vol. 1, no. 2, 1994.
8. B.M. Pangrle, "On the Complexity of Connectivity Binding," *IEEE Trans. on Computer Aided Design*, vol. 10, no. 11, 1991, pp. 1460–1465.
9. R. Jain, A.C. Parker, and N. Park, "Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs," *IEEE Trans. on Computer Aided Design*, vol. 11, no. 8, 1992.
10. J.M. Rabaey and M. Potkonjak, "Estimating Implementation Bounds for Real Time DSP Applications Specific Circuits," *IEEE Trans. on Computer Aided Design*, vol. 13, no. 6, 1994.
11. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, New York: Academic Press, 1980.
12. D.L. Springer and D.E. Thomas, "Exploiting the Special Structure of Conflict and Compatibility Graphs in High-Level Synthesis," *IEEE Trans. on Computer Aided Design*, vol. 13, no. 7, 1994, pp. 843–856.
13. A. Sharma, Estimation and Design Algorithms for The Behavorial Synthesis of Asics, Ph.D. Thesis, University of Wisconsin, Madison, Dpt. of Elec. and Comp. Eng, Dec. 1992.
14. M. Rim and R. Jain, "Lower-Bound Performance Estimation for the High-Level Synthesis Scheduling Problem," *IEEE Trans. on Computer Aided Design*, vol. 13, no. 4, 1994, pp. 451–458.
15. L. Guerra, M. Potkonjak, and J. Rabaey, "System-Level Design Guidance Using Algorithm Properties," in *IEEE Workshop on VLSI S.P.*, San Diego, Oct. 1994, pp. 73–82.
16. H. Jun and S. Hwang, "Design of a Pipeline Datapath Synthesis System for Digital Signal Processing," *IEEE Trans. on VLSI Systems*, vol. 2, no. 3, 1994.
17. O. Sentieys, J.Ph. Diguet, J.L. Philippe, and E. Martin, "Hardware Module Selection for Real Time Pipeline Architectures Using Probabilistic Cost Estimation," in *IEEE ASIC Conference*, Rochester, USA, Sept. 1996.
18. J.Ph. Diguet, Estimation de Complexité et Transformations d'Algorithmes de Traitement du Signal pour la Conception de Circuits VLSI, Ph.D. Thesis, Université de Rennes I, Oct. 1996.
19. K.J.R. Liu and C.-T. Chiu, "Unified Parallel Lattice Structures for Time- Recursive Discrete Cosine/Sine/Hartley Transforms," *IEEE Trans. on Signal Processing*, vol. 41, no. 3, 1993, pp. 1357–1377.
20. K.K. Pahri and D.G. Messerschmitt, "Pipeline Interleaving and Parallelism in Recursive Digital Filters—part I: Pipelining Using Scattered Look-Ahead and Decomposition," *IEEE Trans. on Computer Aided Design*, vol. 37, no. 7, 1989, pp. 1099–1117.
21. C. Wang and K.K. Parhi, "High-Level dsp Synthesis Using Concurrent Transformations, Scheduling, and Allocation," *IEEE Trans. on Computer Aided Design*, vol. 14, no. 3, 1995.
22. J.L. Philippe, O. Sentieys, J.Ph. Diguet, and E. Martin, "From Digital Signal Processing Specifications to Layout," *Novel Approaches in Logic and Architecture Synthesis*, Chapman & Hall, 1995, pp. 307–313.
23. F. Balasa, F. Catthoor, and H. De Man, "Exact Evaluation of Memory Size For Multi-Dimensional Signal Processing Systems," in *IEEE International Conference on Computer Aided Design*, 1993.
24. T. Kim and C.L. Liu, "A New Approach to the Multiport Memory Allocation Problem in Data Path Synthesis," *Integration, the VLSI Journal*, 1995, pp. 133–160.

25. M. Balakrishnan, A.K. Majumdar, D.K. Banerji, J.G. Linders, and J.C. Majithia, "Allocation of Multiport Memories in Data Path Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 7, no. 4, 1988.
26. F. Depuydt, G. Gossens, and H. De Man, "Scheduling with Register Constraints for dsp Architectures," *Integration, the VLSI Journal*, vol. 18, 1994.
27. F. Grant, P.B. Denyer, and I. Finlay, "Synthesis of Address Generators," in *IEEE Int. Conf. on Computer Aided Design*, Nov. 1989, pp. 116–119.
28. R. Reynaud, E. Martin, and F. Devos, "Design of a Bit Sliced Address Generator for fft Algorithms," in *Signal processing IV: Theories and Applications*, 1986.
29. J.L. Hennessy and D.A. Patterson, *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., Version francaise par D. Etiemble et M. Israel, Mc Graw-Hill, 1992.
30. S. Bakshi and D.D. Gajski, "A Memory Selection Algorithm for High-Performance Pipelines," in *EURO-DAC Brighton*, Sept. 1995.
31. F. Balasa, F. Catthoor, and H.De Man, "Dataflow Driven Memory Allocation for Multi-Dimensional Signal Processing Systems," in *IEEE International Conference on Computer Aided Design*, Nov. 1994, pp. 31–34.
32. H. Samsom, F. Franssen, F. Catthoor, and H. De Man, "Verification of Loop Transformations for Real Time Signal Processing Application," in *IEEE Workshop on VLSI Signal Processing*, Oct. 1995.
33. I.M. Verbauwhede, C.J. Scheers, and J.M. Rabaey, "Memory Estimation for High Level Synthesis," in *ACM/IEEE Design Automation Conference*, 1994, number 31.
34. O. Sentieys, D. Chillet, J.P. Diguet, and J.L. Philippe, "Memory Module Selection for High Level Synthesis," in *IEEE Workshop on VLSI Signal Processing*, 1996.
35. J. Benesty and P. Duhamel, "A Fast Exact Least Mean Square Adaptive Algorithm," *IEEE Trans. on Signal Processing*, vol. 40, no. 12, 1992, pp. 2904–2920.
36. R.E. Blahut, *Fast Algorithms for Signal Processing*, Addison-Wesley, Reading, MA, 1985.
37. Z.J. Mou and P. Duhamel, "Short Length FIR Filters and Their use in Fast no Recursive Filtering," *IEEE Trans. on ASSP*, 1989.
38. A. Zergaïnoh and P. Duhamel, "Implementation and Performance of Composite Fast FIR Filtering Algortihms," in *IEEE VLSI Signal Processing*, Osaka, Japan, Oct. 1995, pp. 267–276.

Jean-Philippe Diguet received the Engineer degree in Electronics from ESEO, France, in 1992 and the PhD degree in Signal Processing from University of Rennes in 1996. He was working toward his thesis with the Lasti laboratory, Lannion, France, then he joined IMEC in 1997 for a postdoctoral year. He is now an assistant professor of electrical engineering at UBS university, Lorient, France and a member of the LESTER laboratory. His current research interests are in architecture application matching and high level estimations for system design methodologies applied to signal-processing and multimedia domains.

Daniel Chillet received the engineering degree in Electronics and Computer Engineering from the graduate School in Applied Science and Technology (ENSSAT) of the Rennes University. After his DEA graduation, he received, in January 1997, the Ph.D. degree in Signal Processing from the Rennes University. The topic of his Ph.D. concerned the definition of high level synthesis for memory unit in the context of signal processing. In September 1997, he was nominate as Assistant Professor at the University of Rennes (ENSSAT). His current research interests concern the topic of the methodology definition for ASIC design in the context of signal processing. He is imply in the developement of the GAUT tool which is a high level synthesis tool developped at LASTI since 1986.

Olivier Sentieys graduated from ENSSAT as a senior executive engineer in Electronics and Signal Processing Engineering in 1990, and received the DEA degree in 1990 and the Ph.D. degree in Signal Processing and Telecommunications in 1993 from the University of Rennes. He is currently an Associate Professor of Electrical Engineering at ENSSAT-University of Rennes and is the head of the Signal-Architecture Group of the LASTI Laboratory. His research interests include high level synthesis of VLSI Signal Processing systems, finite arithmetic effects, and low power methodologies for ASIC and DSP. He is involved in research projects including the development of the design framework BSS, the definition of behavioural IP for telecomm, and joint research programs in codesign and biomedical. He is a member of IEEE and ACM.
sentieys@enssat.fr