HAL
open science

Advances in Bit Width Selection Methodology

David Cachera, Tanguy Risset

To cite this version:

David Cachera, Tanguy Risset. Advances in Bit Width Selection Methodology. [Research Report] RR-4452, INRIA. 2002. inria-00072136

HAL Id: inria-00072136
https://inria.hal.science/inria-00072136v1

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Advances in Bit Width Selection Methodology

David Cachera , Tanguy Risset

N°4452

Avril 2002

THÈME 1

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4452--FR+ENG

Advances in Bit Width Selection Methodology

David Cachera* , Tanguy Risset†

Thème 1 — Réseaux et systèmes
Projet COSI

Rapport de recherche n°4452 — Avril 2002 — 15 pages

Abstract: We describe a method for the formal determination of signal bit width in fixed points VLSI implementations of signal processing algorithms containing loop nests. The main advance of this paper lies in the fact that we use results of the (max,+) algebraic theory to find the integral bit width of algorithms containing loop nests whose bound parameters are not statically known. Combined with recent results on fractional bit width determination, the results of this paper can be used for 1-dimensional systolic-like arrays implementing linear signal processing algorithms. Although they are presented in the context of a specific high level design methodology (based on systems of affine recurrence equations), the results of this work can be used in many high level design environments.

Key-words: VLSI, co-design, systolic arrays, bit width selection, high level synthesis, (max,+) algebra.

(Résumé : tsvp)

* Irisa/ENS-Cachan, Antenne de Bretagne, `David.Cachera@irisa.fr`.
† Inria/Ens-Lyon, `Tanguy.Risset@ens-lyon.fr`.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Une nouvelle méthode pour déterminer la largeur des chemins de données

Résumé : Nous décrivons une nouvelle méthode pour la détermination symbolique des largeurs de chemins de données dans les implémentations VLSI d'algorithmes contenant des nids de boucles. L'originalité principale de ce travail réside dans le fait qu'il utilise des résultats de la théorie de l'algèbre (max,+) pour résoudre ce problème pour des nids de boucles paramétrés (c'est-à-dire des nids de boucles dans lesquels les bornes des indices dépendent de paramètres qui ne sont pas nécessairement connus statiquement). Combinés avec des résultats récents sur la détermination des tailles de bits fractionnaires des données codées en virgule fixe, les résultats de cet article peuvent être utilisés pour des réseaux monodimensionnels de type systolique implantant des systèmes linéaires de traitement du signal. Bien qu'ils soient présentés dans le contexte d'une méthodologie particulière de synthèse de haut niveau (fondée sur la manipulation de systèmes d'équations récurrentes affines), les résultats de ce travail peuvent être utilisés dans tout autre environnement de synthèse de haut niveau.

Mots-clé : VLSI, conception conjointe, réseaux systoliques, largeur de chemin de données, synthèse de haut niveau, algèbre (max,+).

1 Introduction

With the combined explosion of the electronic appliances market and of the complexity of systems implemented on a chip today, system designers face the challenge of silicon compilation. The long and error prone manual design of VLSI systems will be replaced by a step by step compilation from high level code in order to speed up and secure the design process. In this design process one difficult stage is the determination of the data-path bit width. As pointed in [15], when processor memory was expensive, programmers paid attention to this problem even for software implementations. The problem however did not receive much attention from the software community because of the standardization of the 32 bits (and then 64 bits) architectures. In silicon design the area of a circuit is always very important because the price of the chip is directly proportional to it.

The goal of the bit width determination stage in the design process is to find a compromise between a large data-path ensuring correctness and precision, and a smaller data-path yielding low power and cheap circuits. When going from a high level algorithmic description to a hardware implementation, the designer has to transform variables with infinite precision and unbounded values into bit vectors of fixed size. In this process, two distinct problems may arise: overflows, that appear if the size of the integral part is too small, and precision loss, if the fractional part is too small. These problems are solved during the determination of, respectively, the *range* of the signal and the *precision* of the signal. This process is very dependent on the global design methodology which makes it difficult to precisely formalize it.

In this paper, we consider this problem in the framework of the derivation of an architectural description from a high level (functional) description. Starting from an algorithmic description of a system, we refine it down to an architectural description in synthesizable VHDL. We focus on computation intensive algorithms, i.e., loop nests. Moreover, our descriptions are generic, i.e., handle symbolic parameters. In order to ensure reusability of the design, these parameters must be instantiated as late as possible. Parametric loop nest manipulations uses the *polyhedral model*. The existing tools for high level synthesis do not handle parameterized loop nest because of the complexity of this model. In this paper we propose new techniques for the symbolic determination of integral bit width in VLSI implementation of programs containing loop nests.

The paper is organized in the following manner: section 2 gives an overview of related work, section 3 presents the modeling we use, section 4 illustrates our methodology on a simple example, section 5 presents our main contribution, and section 6 gives concluding remarks.

2 Related work

Starting from a floating point (or theoretical infinite precision) representation of a system, one must transform it into fixed point representation. One important preliminary issue in this process is the determination of the coding system used for the values manipulated. We do not adress this problem here, we will assume that the numbers are coded in the two's complement system which is widely accepted as the "default" standard. We also assume that we are able to design new algorithms, hence we are not in the position of implementing *bit true* specifications given by normalizing comities where the exact bitwidth of each signal is already given (IEEE, IETF, see [4] for this problem).

A real value r will be represented in two's complement system by a value $\tilde{r}$. $\tilde{r}$ is a $Q_{n,m}$ signal if it is composed of n binary digits before the dot (integer bit width needed to code the *range* of the signal) and m binary digits after the dot (fractional bit width used to set the *precision* of the implementation). In order to avoid overflow during an operation between two signals, operands can be shifted (divided by a power of two): as a consequence, the range of the result will be reduced. This shift information must be carried along the data path computation in order to restaur the correct range of the final result. If overflow still occurs, the resulting signal is *saturated* to fit with the available dynamic offered by its own representation (the designer may choose not to do any treatment in that case).

Once the range of each signal is set, one can reduce the precision by truncating the results of operations. This operation reduces the number of bits necessary to represent the result, but degrades the precision of the computation. The VLSI designer chooses these different bit widths from information obtained by static analysis tools (like the computation of the *signal to noise ratio*) or extensive simulations. As the important information is the total number of bits, the determination of the range and precision should not be handled independently. However, the complexity of the process is greatly reduced if they are treated in sequence. This determination becomes even more difficult when the algorithm contains loop nests for which some parameters (number of iterations for instance) are fixed later in the design process. The integral bit width determination of parameterized loop nests algorithms was the major motivation of the present work.

One can classify the bit width determination methods into two categories: the *formal methods* attempt to statically extract bit width information from the code and the *simulation based methods* try to reach an admissible bit width by successive simulations. It turns out that a combination of both types of methods is mandatory. Formal methods prove very useful to orient the design space exploration for the simulation phase. The approach presented here should be classified in the formal methods category.

Upon the methodologies using simulation, one can retain the following: [9, 8, 13, 3, 10] and the proposed inmplementations in Ptolemy [6], DSP Station or SPW. The Ptolemy environment in addition provides an interesting typing mechanism that defines a hierarchy of types, and allow data refining within this hierarchy. As these methods are based on simulation, input samples have to be carefully chosen, and no general symbolic result can be obtained for parameterized architectures.

The Gaut synthesis tool [14] uses a formal strategy to derive signal bit width in DSP applications. This strategy defines a model for operators and computes noise standard deviation using algorithms that operate on the data-flow graph of the architectural description. Given constraints such as the output signal-to-noise ratio, Gaut is able to symbolically compute the signal bit widths. The main restriction of this approach is the way it handles loops: loops are unrolled, preventing us from treating specifications that contain loops with parameterized bounds.

An interesting methodology was presented by Amarasinghe et al. [15] and implemented in a silicon compiler. Once again, a typing mechanism is proposed for finite precision numbers and a refinement algorithm that allows the shortening of signal bit width by a bidirectional value range propagation method, and a heuristic classification of variables appearing in loops. The bit widths of variables appearing in loops are set according to their type (constant, linear, polynomial,...) and to the loop bounds, but these bounds are not handled as symbolic parameters.

HP labs [11] propose a formal method for integral bit width selection in the PiCo system. Their method is, as our, oriented towars high level synthesis of loop nests. Their algorithm executes a static iterative constraint propagation until convergence (hence unapplicable if the loop bounds are not statically known).

Recently a very interesting novel result was presented by Menard and Sentieys in [12]. They used mathematical tools from the signal processing theory to compute the noise resulting of a linear[1] system. By using the transfer function of the linear system they explain how to express as a closed function the noise produced at the output of the system. Their method, used in conjunction with the one presented here will enable us to derive the complete bit width of signals for hardware implementation of loop nest programs representing linear systems (the case of non linear system will still needed a simulation phase to derive the bit width of the fractional part).

3 Signals and operators modeling

As mentioned above, our aim is to determine the bit widths that are necessary to both range and precision by means of formal techniques. We model the behavior of each operator with respect to these two concerns: range and precision. The major emphasis of the paper is the derivation of the integer bit width of signals. However, for illustrating purposes, we propose a possible simple model for fractional bit width determination.

Modeling w.r.t. precision. Given any signal sequence $X(n)$ of M successive values, we note $\tilde{X}(n)$ the approximated value of $X(n)$ obtained, for instance, in a VLSI implementation of a function computing $X(n)$. We call *noise* associated with the signal $X(n)$ the value $e_X(n)$: $e_X(n) = X(n) - \tilde{X}(n)$. The noise is abstracted by its standard deviation σ_X. The *signal to noise ratio* (SNR): $R_X = 10\ \log_{10}\left(\frac{1}{\sigma_X^2}\right)$ for a normalized signal is commonly used for evaluating the validity of an implementation, it allows to derive static results concerning the necessary bit width of algorithm.

For each operator, we give an abstract version that models its behavior with respect to the noise standard deviation. This abstraction is taken from [16]. When considering an operator, noise has two sources: (i) generated errors, that result from truncation and appear at a local level, and (ii) propagated errors that result from the accumulation of errors upstream.

Generated noise is a consequence of truncation in the fixed point format. If a signal X is encoded in the two's complement fixed point format with the fractional part encoded on b bits, the truncation generates an error for each value of X, bounded by $q = 2^{-b}$. Assuming this noise follows a uniform probability density on the interval $[-q, 0]$, mean and standard deviation are given by the following equations.

$$\mu_X = -\frac{q}{2} \qquad \sigma_X^2 = \frac{q^2}{12}$$

[1] Here, the word *linear* has to be understood within the context of signal processing theory.

Propagation of noise depends on the nature of the considered operator. For instance, an addition operator sums the input noise standard deviations. Let X and Y be two signals with respective noises σ_X^2 and σ_Y^2. The resulting noise on output has a standard deviation given by

$$\sigma_{X+Y}^2 = \sigma_X^2 + \sigma_Y^2$$

Modeling multiplication is a little bit more tricky because it requires the knowledge of a probabilistic estimation of signal powers. For a more precise modeling of noise (including, in particular, possible correlation betweens signals) see [12, 8].

Modeling w.r.t. range. The integral bit width for signal X will be noted L_X (by convention we include the sign bit of the two's complement representation in this length). As for the precision, the range of signal can be abstracted by probabilistic distribution [8], but in that case the result should systematically be validated by extensive simulation. A conservative approach is usually chosen to prevent overflows, which leads to the following equations[2]:

$$L_{X+Y} = \max(L_X, L_Y) + 1 \quad \text{and} \quad L_{X*Y} = L_X + L_Y - 1$$

These equations define, for each operator *op*, an abstract model $\tilde{op}$ which specifies the behavior of this operator with respect to integral bit width (e.g. $a\tilde{+}b = \max(a, b) + 1$). This abstract model is called "Opcode transfer function" in [11]. This abstraction is naturally extended to complex arithmetic expressions.

Solving these equations. Given a particular implementation of an algorithm, for each variable X, we have to compute two quantities: σ_X and L_X. L_X will give the integer bit width of the signal and σ_X will indicate to the designer if the result is precise enough. For simple algorithms (non parameterized loops), equations defining σ_X can be solved with traditional algebraic techniques [16] and equations defining L_X can be solved by iterative constraint propagation [11]. But if we want to implement a loop for which we do not exactly know the number of iteration (parameterized loop), L_X determination is difficult due to the presence of $\max$ operations. We explain how to overcome this difficulty in the following.

4 Parameterized nested loops: running an example

In this section, we explain on a simple example how we propose to determine the bit width of each variable in a high level design flow from a nested loop program. The example we chose is the Finite Impulse Response (FIR) filter which is a simple convolution filter used in many digital signal processing algorithms (correlation, adaptive filtering, etc.).

[2]Other modeling may lead to slightly different integral bit width equations [11]. However these equations will always be constructed with $\max$ and $+$ operations.

The problem is the following: given a possibly infinite sequence of values $x(n)$, ($n \geq 0$) and N weights $w(i)$ ($0 \leq i \leq N-1$), compute sequence $y(n)$, ($n \geq N-1$) where

$$y(n) = \sum_{i=0}^{N-1} x(n-i)w(i) \tag{1}$$

If we want to realize a VLSI circuit for this computation, we are given some information about the inputs (w and x) and some requirements about the output (y). During the design of the circuit, we will have to determine how to encode our signals.

For our example, we will assume that input signals x and w are composed of rational values which are encoded in a specific finite precision format and are associated with a null noise: $\sigma_w = \sigma_x = 0$, and a similar bit width $L_w = L_x = L_0$. Provided there is no overflow during the computation, the validity of the result y given by the circuit depends on whether the signal to noise ratio is more than a particular value R_{min}, given by the specifications of the problem.

The first step of the design process is to write a program for the FIR in a high level language. Our design methodology uses the Alpha language [5, 18] which is dedicated to the synthesis of highly parallel (systolic-like) architectures. The FIR specification written in Alpha is shown on figure 1 (N is the width of the convolution filter, weights are denoted by $w(i)$, and M is the number of inputs $x(n)$). The architecture in figure 2 is obtained with the high level design process using the MMAlpha environment [5].

```
system fir : {N,M | 3<=N<=M-1}
  (x : {n | 1<=n<=M} of integer;
   w : {i | 0<=i<=N-1} of integer)
     returns (res : {n | N<=n<=M} of integer);
var
   Y : {n,i | N<=n<=M; -1<=i<=N-1} of integer;
let
   Y[n,i] = case
            { | i=-1} : 0[];
            { | 0<=i} : Y[n,i-1] + w[i] * x[n-i];
            esac;
   res[n] = Y[n,N-1];
tel;
```

Figure 1: Specification of the FIR filter: high level description in Alpha. This specification is directly obtained by serializing the summation in equation (1).

The methodology associated with the MMAlpha tool [5] leads to the *space-time* representation of the program displayed on figure 3. The original specification of the FIR shown on figure 1 is close to its mathematical formulation: in particular there is no specified order for computations. The equations of the Alpha program represented on figure 3 exactly express the same computation (we didn't represent the inputs and output of the program which are identical to the ones of figure 1), but in addition they explicitly define *when* and *where* each computation should take place. Indeed,

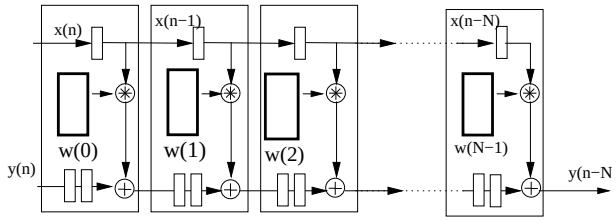

Figure 2: RTL description (in VHDL) semi-automatically derived from this specification of figure 1 with the systolic design methodology implemented in the MMAlpha environment.

by interpreting index `t` as time and index `p` as space (processor number), we have a very precise description of a pipelined systolic execution of the FIR filter on a SIMD-like machine. We still have to apply some transformations to refine this specification until we reach a complete description of the architecture of figure 2, but we will show that all the information needed to derive the bit width of each signal is present in the program of figure 3.

```
W[t,p] = case
         { | t=p+1} : w[t-1];
         { | p+2<=t} : W[t-1,p];
         esac;
XP[t,p] = case
          { | p=0} : x[t+N-1];
          { | 1<=p<=N-1} : XP[t-2,p-1];
          esac;
Y[t,p] = case
         { | p=-1} : 0[];
         { | 0<=p<=N-1} : Y[t-1,p-1] + W[t-1,p] * XP[t-1,p];
         esac;
res[n] = Y[n+1,N-1];
```

Figure 3: Body of the Alpha program of figure 1 after scheduling and mapping in the MMAlpha environment.

Consider for instance the equation defining `XP` (pipelined value of `x`) on figure 3. In each processor, this equation represents a signal, hence we can associate with it : a noise $\sigma_{XP}(p)$ (related to its fractional part), and a range $L_{XP}(p)$ (related to its integral part) which both depend on the processor number. The equation defining `XP` indicates that this value is simply propagated from one processor to the next with a delay of two clock cycles. This equation directly leads to the equations defining σ_{XP} and L_{XP}:

$$\sigma^2_{XP}(p) = \begin{cases} \sigma^2_x & \text{if } p = 0 \\ \sigma^2_{XP}(p-1) & \text{if } p > 0 \end{cases} \quad \text{and} \quad L_{XP}(p) = \begin{cases} L_x & \text{if } p = 0 \\ L_{XP}(p-1) & \text{if } p > 0 \end{cases} \tag{2}$$

These equations are very easily solved, giving $\sigma^2_{XP}(p) = 0$ and $L_{XP}(p) = L_x$ for any processor p. Moreover, we know from our assumption about the input signal x that $\sigma^2_x = 0$ and $L_x = L_0$. Similarly, the same quantities may be computed for `W`: $\sigma^2_W(p) = \sigma^2_w = 0$ and $L_W(p) = L_w = L_0$ for any value of p.

Observe now the equation defining `Y` in figure 3. We will first assume that signal `Y` has the same fractional bit width b in each processor. The equations defining the standard deviation of the noise of `Y` $\sigma^2_Y(p)$ (we note $q = 2^{-b}$) and the range $L_Y(p)$ of `Y` are the following:

$$\sigma^2_Y(p) = \begin{cases} 0 & \text{if } p = -1 \\ \sigma^2_Y(p-1) + \frac{q^2}{12} & \text{if } p > 0 \end{cases} \tag{3}$$

$$L_Y(p) = \begin{cases} 0 & \text{if } p = -1 \\ \max(L_Y(p-1), L_W(p) + L_X(p) - 1) + 1 & \text{if } p > 0 \end{cases} \tag{4}$$

Truncations of the result to b bits explain the two $\frac{q^2}{12}$ terms. We can solve equation 3 with a symbolic solver and the solution we get is $\sigma^2_Y(p) = (p+1)\frac{q^2}{6}$. As explained earlier, the result `Y[n+1,N-1]` of the circuit is valid if its SNR is greater than R_{min}:

$$10\ \log_{10}\left(\frac{1}{N\frac{q^2}{6}}\right) \geq R_{min} \quad \text{leading to :} \quad b \geq \frac{1}{2}\ \log_2\left(\frac{N}{6}10^{\frac{R_{min}}{10}}\right)$$

Hence, if we assume that the fractional part of variable `Y` has the same bit width in each processor, we can compute the exact minimal bit width that is necessary to obtain the required SNR for the FIR. This could be done because of simplistic assumptions on input signals (no input noise). In the general case, we will have to use the method presented in [12].

Consider now equation (4). We will explain in section 5 how to solve it for a given value of L_0. With traditional algebraic techniques, we can find the result if $L_0 = 0$. Indeed, in that case, we will necessarily have $L_Y(p) = p$. Now, if we are able to solve equation (4) for any value of L_0, we will really have an automatic procedure for determining L and σ^2 for loop nests programs as soon as we have a space-time representation of the architecture. Note now that equation (4) can be automatically *generated*. Indeed, equation (4) is very close to the Alpha program of figure 3 (intuitively, they are obtained by "removing" index `t` of the program). This property is very important since it implies that the *whole* process determining the bit width can be automatically driven from the Alpha operational specification of figure 3. This is the main contribution of this paper, and it is detailed hereafter.

5 Solving integral bit width equations for parameterized nested loops

In the previous section, we have shown how to generate integral bit width equations for the FIR example. In this section, we generalize this process and solve the equations for any algorithm containing loop nests implemented on a linear array architecture.

5.1 Generation of integral bit width equations

We start from an *operational description* of the linear architecture: the initial Alpha program has been scheduled, i.e., time and space indices have been identified. Such a scheduled Alpha program defining m variables $V_1, \ldots, V_m$ is composed of m recurrence equations of the following general form:

$$
\begin{array}{ll}
V_1[t,p] = & case \\
& z \in D_{1,1}(t,p) : F_{1,1}(V_1[f_{1,1,1}(z)], \ldots, V_m[f_{1,1,m}(z)]) \\
& z \in D_{1,2}(t,p) : F_{1,2}(\ldots) \\
& \ldots \\
& esac \\
V_2[t,p] = & case \\
& z \in D_{2,1}(t,p) : F_{2,1}(V_1[f_{2,1,1}(z)], \ldots, V_m[f_{2,1,m}(z)]) \\
& z \in D_{2,2}(t,p) : F_{2,2}(\ldots) \\
& \ldots \\
& esac \\
\ldots &
\end{array}
$$

Each variable V_i is defined over a polyhedral domain D_i and can be considered as a generalized array whose shape is not rectangular but polyhedral. In these equations, each $D_{i,j}$ is a sub-domain of domain D_i, and $F_{i,j}$ are arithmetic functions. Index t is the time index, while p is the processor index, and each $f_{i,j,k}$ is an *affine* function on indices called *dependency function*. These two features (affine dependencies between indices and polyhedral domains) are the basis for a rich set of formal transformations that are the core of the MMAlpha environment.

For our purposes, we restrict ourselves to the case of uniform programs, i.e., programs where all functions $f_{i,j,k}$ are translations on indices: $f_{i,j,k}(z) = z - d_{i,j,k}$ where $d_{i,j,k}$ is a constant value. Note however that any Alpha program can automatically be translated into an equivalent uniform one. Such a uniform Alpha program can be associated to its *reduced dependence graph* (dependence graph for short) composed of vertices corresponding to the variables V_i and arcs corresponding to dependencies between these variables (each dependency $f_{i,j,k}$ leading to an arc labeled by $d_{i,j,k}$).

From these equations, we deduce the integral bit width equations of the system by using the modeling introduced in section 3. To do that, we substitute each signal V_i by its integer bit width L_{V_i} and each arithmetic expression $F_{i,j}$ by its abstract version $\tilde{F}_{i,j}$. A variable $V_i(t,p)$ corresponds to a signal V_i in processor p at time step t. In the final architecture, $L_{V_i}(p)$ must be the maximum bit width of $V_i(t,p)$ over time. This can systematically be obtained by projecting every polyhedral domain and every dependency function on index p along the time axis (we note $d'_{i,j,k}$ for the projection of $d_{i,j,k}$ on index p). The action of projecting parameterized domains can be easily performed with the polyhedral library Polylib [17] which is the kernel of the MMAlpha environment.

After this projection, the case branches are merged into a single expression which is the maximum of all the expressions inferred from each branch. The system of equations we get after suppression of time index has the following form (where each $\tilde{F}_{i,j}$ is an arithmetic expression involving only $\max$ and $+$ operations):

$$
\begin{aligned}
L_{V_1}(p) = \max(&\tilde{F}_{1,1}(L_{V_1}(p-d'_{1,1,1}),\dots,L_{V_m}(p-d'_{1,1,m})),\\
&\tilde{F}_{1,2}(L_{V_1}(p-d'_{1,2,1}),\dots,L_{V_m}(p-d'_{1,2,m})),\dots)\\
L_{V_2}(p) = \max(&\tilde{F}_{2,1}(L_{V_1}(p-d'_{2,1,1}),\dots,L_{V_m}(p-d'_{2,1,m})),\\
&\tilde{F}_{2,2}(L_{V_1}(p-d'_{2,2,1}),\dots,L_{V_m}(p-d'_{2,2,m})),\dots)\\
\dots
\end{aligned}
$$

5.2 Solving the bit width equations in the $(\max, +)$ algebra

To be able to solve this system, we will make two assumptions.

- The first assumption we make is that we have a linear monodirectional array: data are carried in the array from one processor to its neighbor following a unique direction. Each $L_{V_i}(p)$ depends on values $L_{V_j}(p-1)$. It can be easily shown that any system of uniform recurrence equations can be rewritten with unit dependencies by introducing temporary variables. Note, however, that it may happen that $L_{V_i}(p)$ depends on itself ($d'_{i,j,k}$ may be equal to 0). This happens, for instance, in auto-adaptive algorithms like the one of DLMS filter. In that case, we will not be able to find the integer bit width (note however that none of the existing techniques will be able to automatically find the fractional bit width either as DLMS is a non linear system).

- Our second assumption is that no multiplication occurs in a strongly connected component of the dependence graph (they only occur *between* strongly connected components of the graph). This situation almost never occurs in treatments implemented in VLSI (signal processing, coding, etc.). Indeed, this would lead to bit width growing too fast (values of order A^p after p operations). In the following, we will decompose the dependence graph in strongly connected components and treat them sequentially. This is exactly what we did in the example of section 4: our dependence graph contained three strongly connected components which where the three vertices `X`, `W` and `Y`. This decomposition into strongly connected components is also adopted in the work of Amarasinghe et al [15], they refer to that as *sequences*.

Under these assumptions, the system of bit width equations for variables contained in a single strongly connected component of the dependence graph is such that no addition between two bit widths occur in the right hand side of any equation. As the addition operation is distributive with respect to the $\max$ operation, we can extract all the $\max$ operations and keep only a single $\max$ operation applied on several sums. If two occurrences of a particular $L_{V_j}(p-1)$ appear in the definition of $L_{V_i}(p)$, one of them can be statically eliminated by taking the maximum of the two. The general form of the system can finally be restricted to:

$$
\begin{aligned}
&L_{V_1}(p) = \max(L_{V_1}(p-1)+\lambda_{1,1}, L_{V_2}(p-1)+\lambda_{1,2},\dots,L_{V_m}(p-1)+\lambda_{1,m},\lambda_{1,m+1})\\
&L_{V_2}(p) = \max(L_{V_1}(p-1)+\lambda_{2,1}, L_{V_2}(p-1)+\lambda_{2,2},\dots,L_{V_m}(p-1)+\lambda_{2,m},\lambda_{2,m+1})\\
&\dots
\end{aligned}
\tag{5}
$$

The idea that we introduce here is to use the $(\max, +)$ algebra theory [7, 1] to solve this system. $(\max, +)$ is an algebra where $\max$ stands for the additive operation and $+$ stands for the multiplicative operation. We will show that, with the help of the *Perron-Frobenius* theorem adapted to

$(\max, +)$, we will be able to solve these equations. If we call $L_V(p)$ the vector composed of the $L_{V_i}(p)$ values, we can write the definition of $L_V(p)$ in terms of $L_V(p-1)$ as a matrix vector multiplication in the $(\max, +)$ algebra. From now on, we will note $\oplus$ for the $\max$ operator, $\otimes$ for the $+$ operator and $\odot$ for the $(\max, +)$ matrix vector product (hence, $(A \odot b)_i = \underset{j=1,m}{\text{Max}}\,(A_{i,j} + b_j) = \bigoplus_{j=1,m} (A_{i,j} \otimes b_j)$). We denote ϵ the zero of $\oplus$ (usually $-\infty$).

The good thing with matrix vector is that it is a linear operation and hence, algebraic theory provides solutions to equations of this form in a $(\max, +)$ algebra. In a first step, we assume that all the constant coefficients $\lambda_{i,m+1}$ are ϵ. We explain hereafter how to generalize the technique to the case where these coefficients are not equal to ϵ. Under this assumption, the integer bit width system (5) can be expressed as:

$$L_V(p) = M \odot L_V(p-1) \quad \text{where} \quad M = \begin{pmatrix} \lambda_{1,1} & \lambda_{1,2} & \cdots & \lambda_{1,m} \\ \lambda_{2,1} & \lambda_{2,2} & \cdots & \lambda_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{m,1} & \lambda_{m,2} & \cdots & \lambda_{m,m} \end{pmatrix}$$

Let us now consider the graph whose adjacency matrix is M. This graph is isomorphic to the subgraph of the dependence graph corresponding to the strongly connected component we study (a non-ϵ value in $M_{i,j}$ corresponds to a dependency between V_i and V_j). A property of adjacency matrices of strongly connected graphs is that they are *irreducible*. In classical algebra, the irreducibility of a matrix A is used in the Perron-Frobenius theorem to show that the spectral radius ρ_A of A is an eigenvalue of A and the associated eigenspace is a ray whose components are all positive. This result can be used to approximate the value of the vector recursively defined by $b(n) = A \cdot b(n-1)$ for large n (intuitively $b(n) \simeq \Theta(\rho_A^n)$).

In the $(\max, +)$ algebra, some result equivalent to the Perron-Frobenius theorem has been shown. It can be shown [7] that the spectral ray of matrix M above is the maximal geometric mean of weights of elementary cycles of the graph corresponding to the matrix. This gives us an easy way of computing the spectral ray ρ_M of matrix M. The concept of *cyclicity* is also useful for us. The cyclicity $c(A)$ of a matrix A is the the GCD of the length of elementary cycles of the graph associated to the matrix.

We can now use the extension to $(\max, +)$ algebra of the Perron-Frobenius theorem:

Theorem 5.1 $(\max, +)$ Perron-Frobenius. *Let $M \in \mathbb{R}_{max}^{n \times n}$ be an irreducible matrix in a $(\max, +)$ algebra with spectral ray ρ_M and cyclicity $c(M)$. There exists an integer N such that*

$$k \geq N \quad \Rightarrow \quad M^{k+c(M)} = (\rho_M)^{c(M)} \otimes M^k$$

With this theorem, we can express the asymptotic behavior of the bit width $L_{V_i}(p)$ for the variables V_i contained in one strongly connected component of the dependence graph. Their exact value is computed by hand for small values of p. In summary, we have the following result: for every 2-dimensional uniform loop nest which satisfy the following conditions,

- the loop nest contains parallelism (this imply that it will lead to a linear array with the standard systolic design methodology),

- the loop nest leads to a computable system of equations for the integral bit width (i.e., bit width $L_i(p)$ does not depend on itself as for auto-adaptive filters),
- the strongly connected cycles of the reduced dependence graph do not contain multiplication,

we can automatically find the bit width of each signal by keeping unknown the size of the array. The fact of having a parameterized bit selection is essential in a high level design process as well as for the re-usability of the design.

5.3 Solving the example

We illustrate how to use this technique on the very simple example of the FIR. In our example, we had to solve equation (4): $L_Y(p) = \max(L_Y(p-1)+1, 2L_0)$ (for $p \geq 0$) where L_0 is the initial integral bit width of the input signals w and x. Expressed in the $(\max, +)$ formalism, it leads to: $L_Y(p-1) = M \odot L_Y(p) \oplus (2L_0)$ where $M = (1)$. The graph whose adjacency matrix is M is composed of only one node and one arc (cycling on the node) labeled with the value 1. Hence, the maximal geometrical mean of elementary cycles on the graph is 1 and $\rho_M = 1$ (the cyclicity of matrix M is also 1).

As the system extracted from equation (4) is not linear but affine, we will have to perform a classical change in the specification. The constant will be considered as a variable: C(p), and we will manage to ensure that the value of C(p) is always 0. The new equation in $(\max, +)$ is (with $L_Y(-1) = 0$ and $C(-1) = 0$:

$$\begin{pmatrix} L_Y(p) \\ C(p) \end{pmatrix} = M' \odot \begin{pmatrix} L_Y(p-1) \\ C(p-1) \end{pmatrix} = \begin{pmatrix} M & 2L_0 \\ \epsilon & 0 \end{pmatrix} \odot \begin{pmatrix} L_Y(p-1) \\ C(p-1) \end{pmatrix}$$

The new matrix M' is not reducible anymore but, as the spectral ray of M is positive, the result of theorem 5.1 still holds with ρ_M (intuitively, as ρ_M is strictly positive, $L_Y(p)$ is increasing with p, hence, taking the maximum with a constant can change the behavior for small values of p but not asymptotically). Now, using theorem 5.1, we know that there exists N such that

$$p \geq N \Rightarrow L_Y(p+1) = \rho_M \otimes L_Y(p) = L_Y(p) + \rho_M.$$

This is the result that we had if $L_0 = 0$ (with corresponding N being equal to -1). In the general case, N can be bounded[2]. However, as we deal with small matrices and small integer values this bound is usually small in practice. Hence the most efficient way to find it is to compute by hand the first values of $L_Y(p)$ until we reach a p_0 such that $L_Y(p_0) = L_Y(p_0 - 1) + \rho_M$. For instance, if we take $L_1 = 5$, we compute by hand the initial values of L_Y: $L_Y(-1) = 0$, $L_Y(0) = 10$, $L_Y(1) = 11 = L_y(0) + 1 = \rho_M \odot L_Y(0)$. Hence we immediately get that: $L_Y(p) = L_Y(0) + p = 10 + p$.

6 Conclusion

We have presented a method for the integral bit width determination of signals for loop nest programs which are implemented on one-dimensional linear arrays. The main originality of the paper lies in

the fact that we use results of the $(\max, +)$ algebra theory to solve the problem for parameterized loop nests (i.e., loop nests for which the bounds of loop indices depend on parameters which are not necessarily statically known). The fact of having a parameterized bit selection is essential in a high level design process as well as for the re-usability of the design. We have demonstrated the method on a classical signal processing treatment: the FIR filter. To be complete the method presented here should be used in conjunction of the result of [12] for the determination of fractional bit width of linear signal processing systems.

Although our concept is presented in the context of high level design methodology of Alpha, the work presented in this paper can be used in any high level design environment using a different modeling for operators. However, its application requires an equational description of the behavior of the architecture. Once this description is obtained, the generation and the resolution of the bit width equations are similar to the one presented here.

References

[1] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[2] A. Bouillard and Bruno Gaujal. Coupling time of a (max,plus) matrix. Technical Report 4068, INRIA, 2001.

[3] R. Cmar, L. Rijnders, P. Schaumont, S.Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed-point refinement. In *ACM/IEEE Design and Test in Europe Conference*, Munich, March 1999.

[4] B. Dupont de Dinechin, C. Monat, and F. Rastello. Parallel execution of the saturated reductions. Technical Report 2001-28, ENS-Lyon, 2001.

[5] F. Dupont de Dinechin, P. Quinton, S. Rajopadhye, and T. Risset. First Steps in Alpha. Technical Report 1244, Irisa, 1999.

[6] E.A. Lee et al. Overview of the ptolemy project. Technical Report UCB/ERL No. M99/37, University of California, Berkeley, july 1999.

[7] S. Gaubert and M. Plus. Methods and applications of (max,+) linear algebra. Technical Report 3088, INRIA, January 1997.

[8] S. Kim. *A Study on the Fixed-Point Implementation of Digital Signal Algorithms*. Phd, Seoul National University, February 1996.

[9] S. Kim and E. A. Lee. Infrastructure for numeric precision control in the Ptolemy environment. In *Proceedings of the 40th Midwest Symposium on Circuits and Systems*, August 1997.

[10] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong. Automatic fbating to fixed point translation and its application to post–rendering 3D warping. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 240–248, Los Alamitos, CA, 1999. IEEE Computer Society Press.

[11] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber, and T. Sherwood. Bitwidth cognizant architecture synthesis of custom hardware accelerators. Technical Report HPL-2001-209, CAR, HP Laboratories Palo Alto, 2001.

[12] D. Menard and O. Sentieys. Automatic evaluation of the accuracy of fixed-point algorithm. In C. D. Kloos and J. da Franca, editors, *Design Automation and Test in Europe (DATE)*, pages 529–535, Paris, 2002. IEEE Computer Society Press.

[13] M. Pauwels, D. Lanneer, F. Catthoor, G. Goossens, and H. De Man. Models for bit-true simulation and high-level synthesis of DSP applications. In *IEEE Great Lakes Symp. on VLSI*, Kalamazoo, February 1992.

[14] O. Sentieys, J.P. Diguet, and J.L. Philippe. Gaut: a high level synthesis tool dedicated to real time signal processing application. In *EURO-DAC*, September 2000. University booth stand.

[15] M. Stephenson, J. Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *ACM SIGPLAN '2000 Conference on Programming Language Design and Implementation (PLDI)*, Vancouver, June 1999.

[16] Jean-Marc Tourreilles. *Conception d'Architectures pour Traitement du Signal en Précision Finie*. Thèse de doctorat, université de Rennes 1, January 1999.

[17] D. Wilde. A library for doing polyhedral operations. Technical Report 785, Irisa, Rennes, France, 1993.

[18] D. Wilde. The Alpha language. Technical Report 827, Irisa, Rennes, France, Dec 1994.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399