Automatic Design of VLSI Pipelined LMS Architectures

Anne-Claire Guillou, Patrice Quinton, Tanguy Risset
IRISA
Campus de Beaulieu, 35042 Rennes, France
{aguillou, quinton, risset}@irisa.fr

Daniel Massicotte
Université du Québec à Trois Rivières
C.P. 500, Trois rivières, Quebec, Canada
daniel_massicotte@uqtr.uquebec.ca

Abstract

We present the use of MMAlpha, a tool for the design of parallel VLSI *architectures, for the automatic generation of pipelined* LMS *adaptive filters. Starting from the equations of the applications, MMAlpha allows one to derive a* VHDL *description of an architecture at the register transfer level. We describe the design flow of MMAlpha, which goes through uniformization, scheduling, mapping and hardware generation. Results obtained for implementing a delayed* LMS *algorithm and a look-ahead delayed* LMS *algorithm on a* FPGA *Virtex XCV800 chip are shown.*

1. Introduction

High-level synthesis is now mandatory in order to obtain fast *time to market* delays for new VLSI implementations of signal processing algorithms. Flexible design tools are required to help control the design process, and possibly reuse designs.

Adaptive filters are ubiquitous in special-purpose signal processing hardware, and their VLSI implementation has therefore been extensively studied. The least mean square (LMS) algorithm is one of the most frequently used algorithms, because of its simple structure. Several methods using look-ahead transformations were recently proposed to obtain efficient pipelined architectures for this algorithm. In [12] the relaxed look-ahead transformation of the LMS algorithm was introduced, and a new relaxed look-ahead transformation was later on presented in [4].

The purpose of this paper is to show that VLSI hardware implementations of signal processing algorithms belonging to the LMS filter family can be automatically generated by high-level design tools, using formal transformations. More specifically, we present MMAlpha [3], a tool for the analysis and transformations of recurrence equation specifications down to hardware specifications. Recently, MMAlpha has been used as a design assistant for the synthesis of Kalman filters [7]. Among the transformations available in MMAlpha, *pipelining* and *scheduling* are inherited from the theory of systolic array synthesis and loop parallelization. In MMAlpha, the design process is guided by the designer and the main goal of this tool is to provide automatic program transformations in order to perform complex and error prone tasks.

The content of this paper is as follows: in section 2, we present the equations of the LMS algorithm and of its delayed version (DLMS) as described in [4]. Section 3 describes the MMAlpha tool and illustrates its operation on the design of an architecture for the DLMS algorithm. We first explain how to express the specifications of the DLMS algorithm and how to simulate it. Then by using pipelining and scheduling, we show how we obtain automatically a register transfer level description of an architecture from the original specification. We detail the scheduling step which is critical in the high-level synthesis: during this step, parallelization is performed and the bandwidth of the resulting hardware is chosen. The result of the scheduling step is automatically translated into synthesizable VHDL, from which a VLSI chip is generated. Section 4 presents the synthesis of the DLMS for various values of the parameter N on the Xilinx Virtex XCV800. We also present the synthesis of the look-ahead LMS on the same architecture. Finally, section 6 summarizes related works.

2. The LMS algorithm family

This section describes two algorithms of the least mean square algorithm family and there pipelined implementation. Least mean squares adaptive filters are commonly used in signal processing applications such as interference cancellation, system identification, speech coding and channel equalization. Their success among other adaptive filters is due to the relative simplicity of their hardware implementation. Unlike the fixed coefficient FIR or IIR digital filters, the coefficients of the adaptive digital filters such as LMS filter are adapted at each iteration until they converge. However, recursive and adaptive digital filters cannot be easily pipelined due to the presence of a feedback loop. Thus techniques such as delay insertion and look-ahead have to be used in order to overcome this difficulty.

In the remaining of this paper, we use the following notations:

- V^T denote the transpose of vector V
- N is the filter length
- $d(n)$ is the desired signal
- $e(n)$ is the error signal
- μ is the adaptation constant
- $x(n)$ is the input signal and $\boldsymbol{x}(n)$ is the input vector denoted by :

$$\boldsymbol{x}(n) = [x(n), x(n-1), ..., x(n-N+1)]^T$$

- $\boldsymbol{w}(n)$ is a filter coefficient vector denoted by:

$$\boldsymbol{w}(n) = [w_0(n), w_1(n), ..., w_{N-1}(n)]^T$$

Equations of the LMS filter

The structure of the LMS algorithm is shown in figure 1. By assuming that the adaptive digital filter is an FIR filter whose impulse response is denoted by $w_i(n)$, the output signal $y(n)$ is given by:

$$y(n) = \boldsymbol{x}^T(n).\boldsymbol{w}(n) = \sum_{i=1}^{N} x(n-i)\, w_i(n) \quad (1)$$

The weight update equations of the LMS are given by:

$$\begin{aligned} \boldsymbol{w}(n+1) &= \boldsymbol{w}(n) + \mu \times e(n)\, \boldsymbol{x}(n) \quad (2) \\ e(n) &= d(n) - y(n) \quad (3) \end{aligned}$$

Equations of the delayed LMS

It is possible to obtain a pipelined implementation by inserting delays in the recursive loop of the weigth update formula. The corresponding algorithm is called the *delayed least means square* (DLMS) algorithm (see figure 2). By moving these delays and placing them at every tap of the filter, pipelining of the DLMS is accomplished.

The weigth update equations of the DLMS become:

$$\begin{aligned} \boldsymbol{w}(n+1) &= \boldsymbol{w}(n) + \mu \times e(n-D)\, \boldsymbol{x}(n-D) \quad (4) \\ e(n) &= d(n) - y(n) \quad (5) \end{aligned}$$

The computation of $y(n)$ given in equation (1) is unchanged.

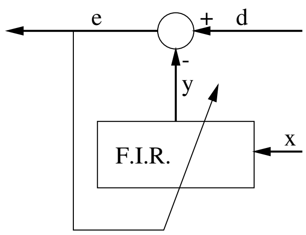

Figure 1. Standard LMS algorithm structure.

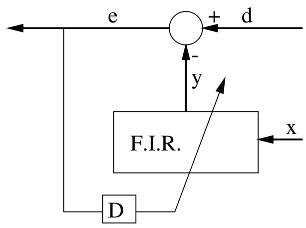

Figure 2. Delayed LMS structure.

3 Design in MMAlpha

MMAlpha is a programming environment for manipulating and transforming Alpha programs. Alpha is a functional language that has been designed in 1989 [6, 5] to express both recurrence equation specifications and hardware descriptions of systolic arrays. The language and the transformations implemented in MMAlpha are inherited from the research on automatic systolic synthesis [9]. MMAlpha was used to design applications in signal processing, multimedia, and bio-informatics. Applications to channel equalization are presented in [7, 14].

MMAlpha targets both Asics and FPGAs. The output of the compilation process is a synthesizable VHDL program that can be compiled with vendors' tools. The interest of targeting FPGA results from the observation that regular designs are well suited to the internal architecture of most FPGA which are made of locally connected configurable logic blocs (CLB)s.

The design flow with MMAlpha is illustrated in figure 3. A computation intensive part of a program (usually a loop) is translated into Alpha (manually). A hardware description at the register transfer level is automatically obtained using MMAlpha (we illustrate this derivation process on the DLMS algorithm hereafter). This description is divided into three parts. The hardware part represents the implementation of the loop on the FPGA. The software part is what replaces the loop in the original program. The hardware/software part handles the communication of data between the host and the FPGA. In this paper, we focus on the derivation of the hardware part, which can be done au-

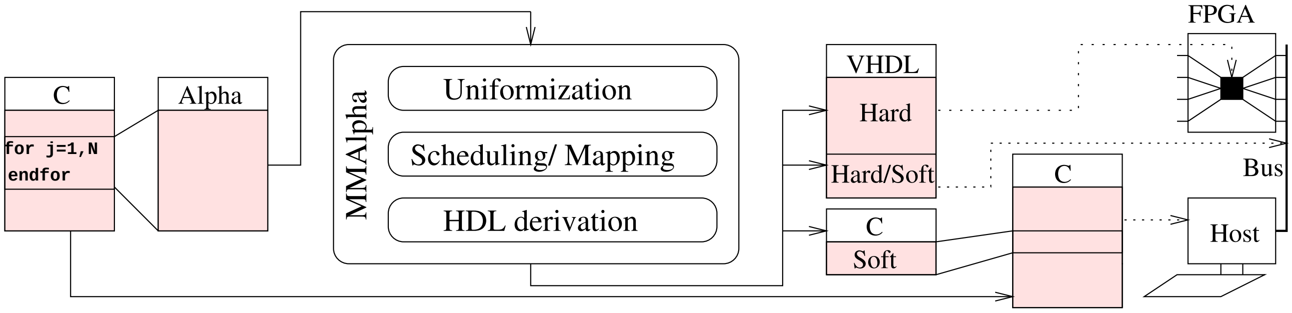

Figure 3. Design flow with MMAlpha. Square shaded boxes represent programs (languages of different specifications are indicated), round boxes represent transformations.

tomatically from a high-level description of the algorithm. The hardware/software part generation is explained in [10].

We now illustrate the MMAlpha design flow on the DLMS algorithm. Equations (1), (4), and (5) represent a high-level specification of the DLMS. In order to obtain an architectural description from this specification, one must go through several program refinements. The first step is to obtain a *functional specification* as equations (1), (4), and (5) cannot be directly executed. This correspond to writing the Alpha source program. Then come a series of program manipulations which end up with a *behavioral hardware description* which corresponds to the usual notion of register transfer level description (RTL). This description (also called AlpHard) is translated into synthesizable VHDL. Finally, logic synthesis of this VHDL can be done. To be complete, we should mention that MMAlpha provides facilities for simulation. In particular, a VHDL test-bench can be automatically derived from the high-level functional simulation of Alpha programs.

Functional specification

First, note that equations (1), (4), and (5) do not specify initializations. The computation of the *definition domain* (i.e. set of indices n for which the variable is defined) of each variable is not easy because we are manipulating recursive expressions. The Polylib library [15] of MMAlpha provides a very efficient help. In our example, we can find that, if the data $x(n)$ is available for $n \geq 0$, the results $y(n)$ will be available for $n \geq N$ and the auto-adaption can start at $n = N + D$. Moreover, notice that this equationnal specification involves two loops although only one loop index (n) is explicitly mentioned. A second loop index (i) must be used to describe the $\mathbf{x}(n).\mathbf{w}(n)$ dot product. Taking these remarks into account, we get an equivalent specification below which can be syntactically transformed in Alpha. For the sake of clarity, we use throughout this paper this equation formalism which is more intuitive than the Alpha language [6] (the Alpha program for the DLMS algorithm is shown in appendix).

$$\begin{aligned}
w(n,i) &= \text{if } (n < N + D) \text{ then } 0 \\
&\quad\ \text{else } w(n-1,i) + \mu e(n-D)x(n-i-D) \\
e(n) &= d(n) - y(n) \\
y(n) &= \text{if } (n \geq N) \text{ then } \sum_{i=0}^{N-1} x(n-i)w(n,i)
\end{aligned}$$

Designing in MMAlpha

We need to refine this specification in order to describe more precisely how it is implemented. For instance, we need to indicate how we choose to implement the $\sum$ operator. A natural choice for that is to serialize it. In MMAlpha, it is very simple to replace the last equation above (defining $y(n)$) by:

$$\begin{aligned}
Y(n,i) &= \text{if } (n \geq N) \text{ then} \\
&\quad\ \text{if } (i = -1) \text{ then } 0 \\
&\quad\ \text{else } Y(n,i-1) + x(n-i)w(n,i) \\
y(n) &= \text{if } (n \geq N) \text{ then } Y(n, N-1)
\end{aligned}$$

Note that the choice is here to serialize with i increasing from 0 to $N - 1$. Serializing in the other direction would give a well known architecture where the data flows of y and x go in opposite directions [9].

The next step, *uniformization*, consists in pipelining the broadcast. This task has received much attention from the theoretical point of view during the last twenty years. MMAlpha provides a procedure for automating this step. After uniformization, the specification is the following one

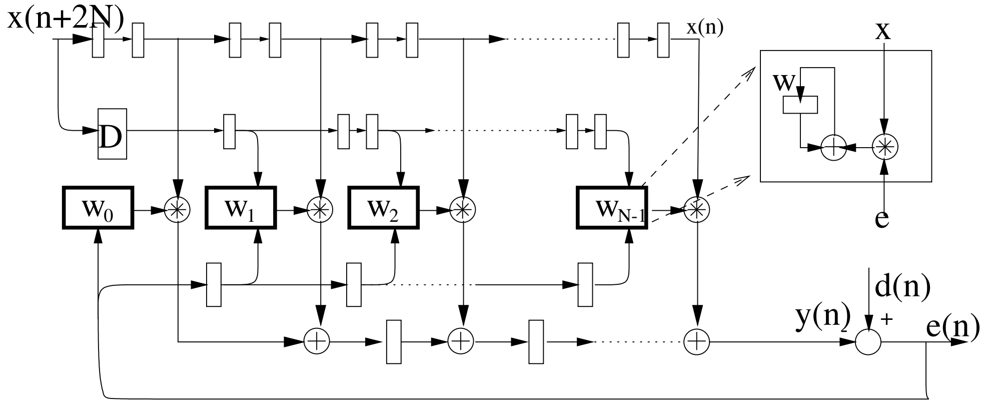

Figure 4. Snapshot of the architecture obtained after the DLMS design process of MMAlpha at $t = n+1$.

(we did not write the initialization for each signal) :

$$
\begin{array}{rcl}
w(n,i) & = & \text{if } \ (n \geq N+D; \ \ i \geq 0) \ \ \text{then} \\
 & & w(n-1,i) + \mu E(n,i) x_D(n,i) \\
x_D(n,i) & = & \text{if } \ (n \geq N+D+1; \ \ i \geq 1) \ \ \text{then} \\
 & & x_D(n-1,i-1) \\
x(n,i) & = & \text{if } \ (n \geq N+1; \ \ i \geq 1) \ \ \text{then} \\
 & & x(n-1,i-1) \\
E(n,i) & = & \text{if } \ (n \geq N+D+1; \ \ i \geq 1) \ \ \text{then} \\
 & & E(n,i-1) \\
Y(n,i) & = & \text{if } \ (n \geq N; \ i \geq 0) \ \ \text{then} \\
 & & Y(n,i-1) + x(n,i) w(n,i) \\
y(n) & = & \text{if } \ (n \geq N) \ \ \text{then } \ Y(n,N-1)
\end{array}
$$

The simple structure of the dependencies between variables of the above program allows the computation of y to be pipelined, which is essential for a fast implementation. We could also choose not to pipeline the computation of y and obtain an architecture as well, but the important point here is that the uniformity of the above program allows the designer to choose very easily between these different implementations. This choice is driven by the user during the next step called *scheduling/mapping*.

Scheduling this program attempts to give an execution date for each computation and tries to provide as much parallelism as possible. The mapping step allocates the computations to the processors. After the scheduling/mapping step, the above equations are automatically transformed into the following ones, which can be directly interpreted by the architecture presented on the left of figure 4:

$$
\begin{array}{rcl}
w(t,p) & = & \text{if } \ (t \geq p+D+1; \ \ i \geq 0) \ \ \text{then} \\
 & & w(t-1,p) + \mu E(t-1,p) x_D(t-2,p) \\
x_D(t,p) & = & \text{if } (t \geq p+D; \ \ p \geq 1) \ \ \text{then} \\
 & & x_D(t-2,p-1) \\
x(t,p) & = & \text{if } (t \geq p+1; \ \ p \geq 1) \ \ \text{then} \\
 & & x(t-2,p-1) \\
E(t,p) & = & \text{if } (t \geq p; \ \ p \geq 1) \ \ \text{then} \\
 & & E(t-1,p-1) \\
Y(t,p) & = & \text{if } \ (t \geq p+2; \ p \geq 0) \ \ \text{then} \\
 & & Y(t-1,p-1) + x(t-1,p) w(t-2,p) \\
y(n) & = & \text{if } \ (n \geq N) \ \ \text{then } Y(n+1,N-1)
\end{array}
$$

Again, remember that we did not write any initialization, neither on the above program nor on the architecture of figure 4. The real architecture is slightly more complicated because of the initialization of w, x and x_D. The corresponding initialization signals are automatically generated during the design process, hence the work of the designer is much simplified. The architecture obtained here is the one that was proposed in [12].

Hardware generation

The Alphard description of the architecture of figure 4 is precise enough to be translated directly into VHDL and synthesized with commercial tools. Beforehand, the designer only has to assign value to the N and D parameters.

4. Results and discussion

The complete design flow for the DLMS architecture can be set up in half an hour for a first prototype architecture.

Re-executing the complete design (for changing the parameter N for instance) takes a few minutes. The VHDL has been simulated using MMAlpha facilities for generating stimuli files. The resulting design has been synthesized using the Synopsys "FPGA compiler II" tool to obtain a net-list and Xilinx "design manager" tool for mapping it on the FPGA. The target FPGA was a Xilinx Virtex XCV800. We measured performance in terms of clock cycle estimated by Xilinx and area (percentage of the total number of CLBs) for each design, as well as synthesis time (measured on a Sun Sparc station). Integers are coded on 9 bits. The results are presented in table 1.

One can easily check that the clock cycle is roughly independent of the size of the architecture (this is one of the nice properties of systolic architectures). Moreover, the area of the architecture is an affine function of the size parameter N. It means that we are able to "guess" the size of the architecture for N=32 (44 %) without synthesizing it. In MMAlpha, the design for any value of N is the same, because N is kept symbolic. The information on the regularity of the design is lost when we use Xilinx tools, hence the place and route time increases more than linearly with the size of the array. We are currently working for having the regularity of the design taken into account during the place and route phase. However, the place and route time is still acceptable. These result prove that (i) the complete design path can be covered by MMAlpha and commercial logic synthesis tools and (ii) the use of MMalpha is justified by the genericity of the design and the ease of switching from one value of N to another.

value of N	clock rate	area	place and route time
8	72.8 MHz	10%	10 mn 40 s
10	71.5 MHz	14%	14 mn 41 s
16	73.1 MHz	23%	28 mn 13 s
32	72.0 MHz	44%	68 mn 26 s

Table 1. Results of the design of the DLMS obtained with MMAlpha for various value of N

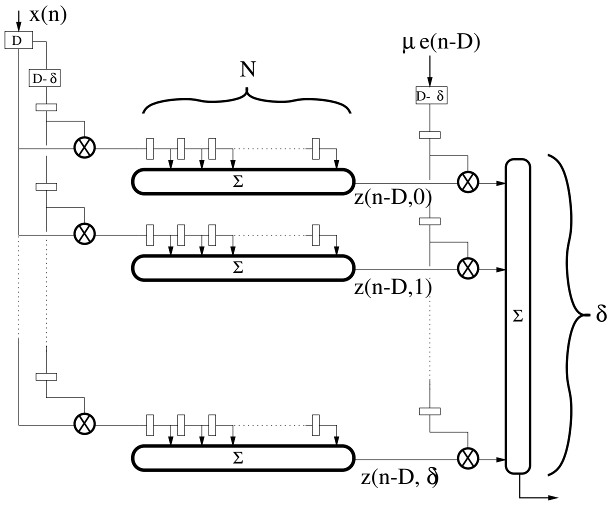

Figure 5. Snapshot of the additional hardware proposed in [4] for improved look ahead.

The DLMS is well adapted to MMAlpha because it fits exactly the model targeted by MMAlpha (regular, systolic architectures). To give an idea of the power of MMAlpha, we choosed to generate a less regular architecture: the look ahead pipelined architecture described in [4]. This architecture is composed of a DLMS plus a mechanism for reducing the delay from D to $D - \delta$. This mechanism is illustrated on the right of figure 4. With a few more work (2-3 days), we could generate the architecture presented on the right of figure 4 with the restriction that the VHDL for the cascade of adders where not automatically generated (the user had to provide it manually). This proves that, even if MMAlpha is not efficient for generating non regular design, it can be integrated into a set of co-design tool for designing only the regular part of the application. The result where the following: the size of the additional hardware of [4] for N=8 is 5% of the XCV8000, the clock rate is 176 MHz. This also proves that the architecture proposed in [4] does not decrease the clock rate and that it adds roughly 50% to the area of the architecture.

5. Related work

There has been quite a lot of ephemeral tools based on systolic techniques. One of the reasons for that is the fact that these techniques are well suited to be integrated in an automatic design process (see [9], for instance). Most of these tools are not maintained today because (i) many of them where limited to systolic design and could not be easily integrated into a system design tool, and (ii) the demand for such tools was not strong enough to maintain them. Among the "survivors", are the design methodology developped at Imec [1], the tools derived from the work of TuDelft and Berkeley [2] and the recent publication of the work of Hewlett-Packard Laboratories [11].

The work at Imec [1] focuses on the minimization of the memory cost (mainly for power consumption purpose), the target architectures are usually not systolic arrays but coarser grain architectures. The PICO-N [11] system of Hewlett-Packard generates synthesizable VHDL for an array of customized VLIW processors. The target architecture is restricted but the authors obtain good results on loop programs extracted from various digital signal and image processing applications. These two quasi-industrial tools prove that the demand has been increasing and justify the amount of work realized in the development of MMAlpha. The Compaan [2] tool attempts to compile loop nests written in Matlab into a Process Network description which can be connected to Ptolemy.

6. Conclusion

In this paper, we have demonstrated the ability of MMAlpha to provide quickly efficient hardware design for the DLMS algorithm. Although the design proposed where not new, the ease for the generation and simulation of VHDL allows one to prototype an hardware implementation much faster than writing the VHDL code by hand. We also gave experimental results on the LAS architecture that were not published in the presentation of this architecture. The MMAlpha tool is a GNU-licenced software that should allow fast and easy design process. Our intention is to provide automatically the interface between the design and the host processor and obtain a complete circuit compiler for FPGAS.

References

[1] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology*. Kluwer Academic Publishers, 1998.

[2] E. F. Deprettere, E. R. a nd Paul Lieverse, and B. Kienhuis. Compaan: Deriving process networks from matlab for embedded signal processing architectures. In *8th International Workshop on Hardware/Software Codesign (CODES'2000)*, San Diego, CA, may 2000.

[3] F. Dupont de Dinechin and P. Quinton and S. Rajopadhye and T. Risset. First Steps in Alpha. Publication Interne 1244, IRISA, 1999.

[4] M. Katsushige, N. Kiyoshi, and K. Hitoshi. Pipelined LMS Adaptative Filter Using a New Look-Ahead Transformation. *IEEE Transactions on Circuits and Systems*, 46:51–55, Jan. 1999.

[5] P. Le Moenner, L. Perraudeau, S. Rajopadhye, T. Risset, and P. Quinton. Generating regular arithmetic circuits with AlpHard. In *Massively Parallel Computing Systems (MPCS'96)*, May 1996.

[6] C. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, Université de Rennes 1, IFSIC, Dec. 1989.

[7] A. Mozipo, D. Massicote, P. Quinton, and T. Risset. Automatic Synthesis of a Parallel Architecture for Kalman Filtering using MMAlpha. In *1999 IEEE Canadian Conference on Electrical & Computer Engineering*, Edmonton, Canada, May 1999.

[8] P. Le Moenner and L. Perraudeau and S. Rajopadhye and T. Risset and P. Quinton. Generating Regular Arithmetic Circuits with AlpHard. In *Massively Parallel Computing Systems (MPCS'96)*, 1996.

[9] P. Quinton and Y. Robert. *Systolic Algorithms and Architectures*. Prentice Hall and Masson, 1989.

[10] T. Risset and S. Derrien. Interfacing compiled FPGA programs: the MMAlpha approach. In *PDPTA2000: Second International Workshop on Engineering of Reconfigurable Hardware/Software Objects*. CSREA Press, june 2000. To appear.

[11] R. Schreiber, S. Aditya, B. R. Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. High-level synthesis of nonprogrammable hardware accelerators. In *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2000)*, Boston, july 2000.

[12] N. Shanbhag and K. Parhi. Pipelined Adaptive DFE Architectures Using Relaxed Look-Ahead. *IEEE Transactions on Signal Processing*, 43(6), June 1995.

[13] J. Thomas. Pipelined Systolic Architectures for DLMS Adaptive Filtering. *Journal of VLSI Signal Processing*, (12):223–246, 1996.

[14] Étienne Mémin and T. Risset. On the study of vlsi derivation for optical flow estimation. In *Parallel Image Processing and Analysis - Theory and Applications - (IWPIPA '99)*, Chenai, India, Jan. 1999.

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

A Alpha program for the DLMS algorithm

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