VLSI HIGH LEVEL SYNTHESIS OF FAST EXACT LEAST MEAN SQUARE ALGORITHMS BASED ON FAST FIR FILTERS

Jean-Philippe Diguet[1,2] *Olivier Sentieys*[1] *Daniel Chillet*[1] *Jean-Luc Philippe*[3]

[1] LASTI - ENSSAT - Université de Rennes, 6 rue de Kérampont, 22300 Lannion, France
eMail: sentieys@enssat.fr; http://www.enssat.fr/RECHERCHE/ARCHI
[2] Currently at IMEC, Leuven, Belgium
[3] LESTER - Université de Bretagne Sud, 10 Rue Jean Zay, 56100 Lorient, France

ABSTRACT

This paper relates experiences of algorithmic transformations in High Level Synthesis, in the area of acoustic echo cancellation. The processing and memory units are automatically designed for various equivalent LMS algorithms, in the FIR case, with important computational load. The results obtained with different filter lengths, give an accurate prototyping of new fast versions of the LMS algorithm. It also show that a theoretical arithmetic reduction must be correlated to the associated increase of memory requirements.

1. INTRODUCTION

One of the most important issue of High Level Synthesis is to provide the designer (e.g. in the signal processing area) with a tool that enables him to explore rapidly a part of the design space. For a given application, algorithmic transformations represent one of the most powerfull way [1] to approach the optimal design solution. In this paper, we focuse on algorithmic transformations applied to the VLSI implementation of the LMS algorithm. The High Level Synthesis is performed by the CAD tool: GAUT that optimizes area of both datapath [2] and memory unit [3], under a throughput constraint. Adaptive filters are widely used in many applications [4], [5]. In the special case of acoustic echo cancellation, the large size of finite impulse response requires filters about 1000-4000 taps. The computational load of such applications justify the efforts spent to reduce the arithmetic complexity, our goal is to test, through 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 [6] 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 [7], it is composed of short length FIR subfilters derived from the original FIR algorithm with the Chineese Remainder Theorem (CRT). Contrary to the block versions, these kind of fast algorithms generate a processing delay independant of the filter length (see [8], [9]). The five kind of synthesized algorithms are described in the second part.

For each algorithm and for five different filter sizes (1024, 1300, 1600, 1800, 2048 taps) we have carried out synthesis of both datapath and memory units. The sample frequency constraint is set by the application type to 16 KHz, the word length is 16 bits and and the component library has been built with a $1\mu m$ technology. The results, given in the third part, will be discuted, we will see the significance of memory aspects inherent in fast FIR techniques.

2. ALGORTIHMS

2.1. Reduction of arithmetic complexity

The first algorithm is the classic LMS in the FIR case. The principle of the 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 [8]. 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 lengfth L/N_i. Despite of the pre and post-processing operations, it results in a significant reduction of the arithmetic complexity. By repeating the process [8] 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 intepolations points and a $F(3,3)$ algorithm.

2.2. $F(2,2)$ - first version

The first version of the $F(2,2)$ is built by choosing the following interpolation points in the pre-processing pro-

cess: $\{0, 1, \infty\}$. If the initial FIR filtering is defined as: $Y(z) = H(z).X(z)$, we obtain the following equations:
Filtering part:

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

Updating in accordance with the FELMS algorithm:

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

The eq. 2 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 figure 1 depicts the structure of the algorithm.

2.3. $F(2,2)$ - 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:

$$\begin{cases} Y_0 = X_0.H_0 + X_1.H_1.z^{-2} \\ Y_1 = X_0.H_0 + X_1 + H_1 - (X_0 - X_1)(H_0 - H_1) \end{cases} \quad (3)$$

Updating in accordance with the FELMS algorithm:

$$\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n+1)} = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_{(n-1)} + \mu \begin{bmatrix} X_0^t(e_n + e_{n-1}) - (X_0 - X_1)^t e_{n-1} \\ X_0^t(e_n + e_{n-1}) + (X_1.z^{-2} - X_1)^t e_n \end{bmatrix} \quad (4)$$

We also get two terms common to the computation of H_0 and H_1 and the substraction of the X_i terms results from the filtering part.
The reduction of arithmetic complexity is equal to the previous one: 25%

2.4. $F(3,3)$

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

$$\begin{cases} Y(z) = \left(Y_0 + Y_1.z^{-1} + Y_2.z^{-2}\right) \\ X(z) = \left(X_0 + X_1.z^{-1} + X_2.z^{-2}\right) \\ H(z) = \left(H_0 + H_1.z^{-1} + H_2.z^{-2}\right) \end{cases} \quad (5)$$

$$\begin{cases} Y_0 = (X_0.H_0 - X_2.H_2.z^{-3}) + ((X_1 + X_2)(H_1 + H_2) - X_1.H_1)z^{-3} \\ Y_1 = ((X_0 + X_1)(H_0 + H_1) - X_1.H_1) - (X_0.H_0 - X_2.H_2.z^{-3}) \\ Y_2 = ((X_0 + X1) + X_2)(H_0 + H_1 + H_2) - \\ ((X_0 + X_1)(H_0 + H_1) - X_1.H_1) - ((X_1 + X_2)(H_1 + H_2) - X_1.H_1) \end{cases} \quad (6)$$

Updating in accordance with the FELMS algorithm:

$$\begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}_{(n+1)} = \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix}_{(n-2)} + \mu \begin{bmatrix} X_0^t(e_{n-2} - e_{n-1} + e_n) - \\ (X_0 + X_1)^t(e_{n-2} - e_{n-1}) + \\ (X_1 + X_2)^t e_{n-2} \\ -X_0^t(e_{n-2} - e_{n-1} + e_n) + \\ (X_0 + X_1)^t(e_{n-2} - e n_1) - \\ (X_2.z^{-3} + X_0)(e_{n-1} - e_n) + \\ ((X_2.z^{-3} + X_0) + (X_0 + X_1))^t e_{n-1} \\ X_0^t(e_{n-2} - e_{n-1} + e_n) + \\ (X_2.z^{-3} + X_0)^t(e_{n-1} - e_n) + \\ (X_1.z^{-3} + X_2.z^{-3})^t e_n \end{bmatrix} \quad (7)$$

The complexity is now a about a third less than the original. The eq.7 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. 6).

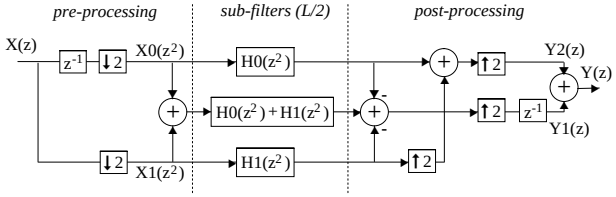

Figure 1. Structure of fast FIR F22-version 1

2.5. LMS-F22F22

This fourth kind of algorithm is obtained by applying the F(2,2) method to each sub-filter of the initial F(2,2) structure. The theorical complexity saving is about 44%. In fact, we will see that its memory requirements fully lead to a prohibitiv global cost.

3. RESULTS

3.1. Estimation

Firstly we compute, with the Probabilistic Estimation Module [10] of GAUT, the distribution of the area cost in time. This kind of estimation is charaterized by a high level of abstraction, it is computed without any assumption on the synthesis algorithms choice. By associate average and standard deviation we obtain a good estimator to compare the quality of ressource utilization in the datapath. The abstraction level prevents from having a great accuracy. However the estimation provides a cost from operators and registers that enables the designer to make comparisons. Let note that the cost evolution due to multiplexers, demultiplexers and tristates is linked to the register probable numbers. The evolution of such a probable cost as transformations are applied, enable to correctly convey the futur cost of the datapath. Thus the effects of transformations may be appreciate in order to take specification decisions.

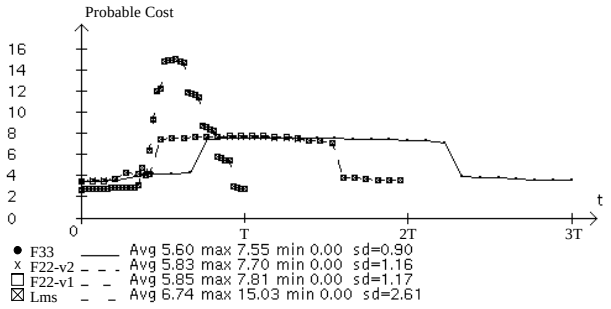

Figure 2. Functional units and registers probabilistic cost estimation

On the figure 2 the cost of functional units and registers is given for $N = 2048$ taps, the LMS may use T time units to compute one output, the $LMS - F22$ uses $2T$ times units to compute two outputs in parallel and the $LMS - F33$ may use $3T$ time units to compute three outputs in parallel. We observe that the reduction of the arithmetic complexity enables the $LMS - F(N, N)$ algorithms to get a better area cost distribution, the averages and standard deviation give a reduction about 25% for the $LMS - F22$ algorithms and about 30% for the $LMS - F33$ algorithm. The table 3 sums up the results obtained from probabilistic estimation and synthesis. The estimation is close to the theory, but the memory cost does not appear here, we will see that final results will be different.

3.2. Synthesis

All results concerning the datapath, memory and overall areas for the different algorithms, are given in the table 1. The bold values represent the lower cost in each category. So we note that the fast algorithms did not produce the attended results, worse, the basis LMS is sometimes better than the optimized versions. Let analyse deeply the results.

PU area	Gaut (PU)	% /lms (G)	Prob. cost (op.+reg.)	% /lms (PC)
LMS	20,5	-	9,35	-
LMS-F22-v1	13,6	33	7,02	25
LMS-F22-V2	13,1	36	6,9	25,2
LMS-F33	11,2	45	6,5	30

Figure 3. PU area savings from synthesis and probabilistic estimations

The five algorithms seem to have the same memory requirements: $2.L$ memory points for the signal vector and the filter coefficients. However we note in the table 1 that faster is the algorithm higher is the memory cost. The extra memory cost is due firstly, to the M new samples vectors of length L/N that have be, in practice, created in order to compute in parallel the M sub-filters. In [9] 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 loaded in memory or registers.

		Area in mm^2				
Nb taps		1024	1300	1600	1800	2048
LMS	PU	8,6	8,8	12,2	15,3	20,5
	MU	**3,9**	**3,9**	**5,8**	**7,2**	**7,8**
	Tot	12,5	**12,7**	18	22,5	28,3
LMS-F22-v1	PU	**4,5**	**8,6**	9,8	8,8	13,6
	MU	6,5	6,2	11,5	8,6	11,6
	Tot	**11,1**	**14,3**	21,3	**18,3**	25,3
LMS-F22-v2	PU	4,9	9	**8,6**	9,6	13
	MU	6,8	7,5	11,5	9,2	11,8
	Tot	11,7	16,5	**18,8**	18,9	**24,8**
LMS-F33	PU	5,2	8,9	9,3	**9,5**	**11,2**
	MU	8,1	12,2	15,5	10,9	13,7
	Tot	13,3	21,2	24,7	20,5	24,9
LMS-F22-F22	PU	5,9	8,9	9,7	11,1	11,4
	MU	19,1	18,3	22,1	20,5	27,7
	Tot	25	27,2	31,8	31,6	39,1

(PU: processing unit, MU: memory unit)

Table 1. Design area from behavorial synthesis

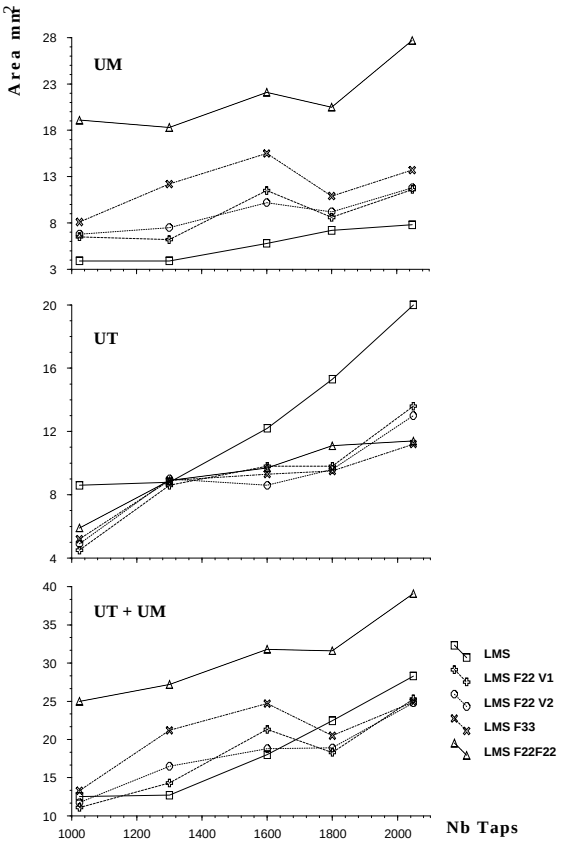

Figure 4. LMS cost vs taps

The memory cost is not the only responsible for the extra cost observed. For instance, we note that the $LMS-F33$ datapath becomes less expensive than the $LMS-F22$ datapath only for a number of taps equal to 3072, wheras it should have always been cheaper. In fact, the memory transfers also modify the datapath cost, because they generate multiplexers, demultiplexers and tristates. When the data transfers become important, the cost of these small components, often neglected in High Level Synthesis, result in a significant extra cost.

The less is regular the algorithm the higher are memory costs. The best example is given by the algorithm LMS-F22-F22 whose the memory requirements lead to worst solution whatever is the taps number.

Let note that the reduction of the 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 multiplations 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 synthesis is an increase of the number of registers, multiplexers, demultiplexers 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 1300, 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 interconnections 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 four times out of five, despite of the increase of memory resources. The difference between the two versions of the $F22$ algorithm is based on the number of substractions and additions that are better balanced in the second kind of specification. In the case of 2048 taps (see fig. 5) it enables to save one adder while sharing the time constraints between the two kinds of operators Finally it lead to a cheaper cost for the LMS-F22-v2 algorithm.

4. CONCLUSION AND PERSPECTIVES

In this paper we described an experience in High Level Synthesis applied to a real life application of acoustic echo cancellation. The designs carried out include functional and memory units. The results obtained point out the problem of the increase of memory unit inherent in methods of arithmetic complexity reduction. They also hightlight the importance of algorithmic transformations in High Level Synthesis.

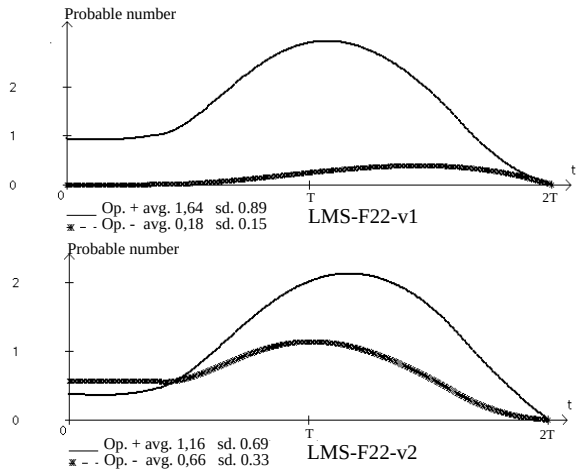

Figure 5. Estimations of Add. and sub. number for the LMS-F22 algo. with 2048 taps

The study such fast algorithms would require, in futur, more efforts on regularity in order to reduce memory costs. Anyway, the best architectural solutions would be the results of trade-offs between arithmetic complexity savings and increasing memory requirments.

REFERENCES

[1] M. Potkonjak and J.M. Rabaey, *VLSI Design Methodologies for Digital Signal Processing*, chapter Exploring the Algorithmic Design Space Using High Level Synthesis, pp. 131–167, International Series in Engineering and Computer Science: VLSI,. Kluwer Academic Publishers, 1994.

[2] 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.

[3] J.L. Philippe, D. Chillet, O. Sentieys, and J.Ph. Diguet, "Memory aspect in signal processing and HLS tool: some results," in *EUSIPCO*, Trieste, Italy, Sept. 1996.

[4] S. Haykin, *Adaptive Filter Theory*, Englewood Cliffs, NJ: Prentice Hall, 1986.

[5] O. Macchi, "Le Filtrage adaptatif en Télécommunications," *Annales des Télécommunications*, vol. 36, no. 11-12, 1981.

[6] J. Benesty and P. Duhamel, "A Fast Exact Least Mean Square Adaptive Algorithm," *IEEE Trans. on Signal Processing*, vol. 40, no. 12, pp. 2904–2920, Dec. 1992.

[7] R.E. Blahut, *Fast Algorithms for Signal Processing*, Addison-Wesley, Reading, MA, 1985.

[8] Z.J. Mou and P. Duhamel, "Short length FIR filters and their use in fast no recursive filtering," *IEEE Trans. on ASSP*, 1989.

[9] 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.

[10] J.Ph. Diguet, O. Sentieys, J.L. Philippe, and E. Martin, "Probabilistic resource estimation for pipeline architecture," in *IEEE Workshop on VLSI S.P.*, Sakai, Japan, Oct. 1995, pp. 217–226.