Architectural Exploration And Optimization for Counter Based Hardware Address Generation

Miguel Miranda Martin Kaspar[1] Francky Catthoor‡ Hugo De Man‡

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
‡Professor at the Katholieke Universiteit Leuven

Abstract

A set of automated system level techniques is presented for architectural exploration and optimization of counter based address generation units in real time signal processign systems. The goal is to explore different architectural alternatives available when mapping array references in order to select the most promising ones in area cost. The techniques are demonstrated on realistic test-vehicles, showing that architectural decision at early stages of the design process, can have a very large impact on the resulting area figure.

1 Introduction

The increasing requirements in throughput for memory intensive signal processing applications and the use of distributed memory architectures to provide enough bandwidth in the memory access, impose a tight cycle budget for memory address evaluation. Therefore, address manipulation forms a crucial component of any architecture which deals with data-dominated algorithms. In that case, efficient access to the memories within real-time constraints requires an optimized mapping of the initial array-indexed expressions onto hardware. Some architectural decisions can be taken to alleviate this problem. Most of them are based on the use of specialized, although still programmable, hardware address generation units (pAGUs) [1, 2]. These units especially benefit from methodologies targeted to efficiently reduce indexed array references to address pointer references for fast post-modification [3, 4]. However, they are normally quite costly in terms of area due to the arithmetical blocks, address register files and/or counters included to provide enough programmability. In order to cope with the bandwidth requirements in distributed memory architectures, a number of such structures are then needed. As the number of units grows, their cost and especially the cost associated to the program ROMs and associated instruction decoders also grows, reaching a point where the overhead introduced becomes a dominant factor in the overall data path related design cost. This is true in terms of area and power [5], as demonstrated by the large memory management units (MMUs) which reside on MPEG type designs.

An alternative to this approach is to use many small application-specific address generation units. In the most straightforward and usually inefficient case, this leads to a direct mapping of each individual array index reference, onto dedicated hardware, e.g [6]. This strategy can also lead to a large cost overhead if not executed well, although in this case the overhead only depends on the overall contribution of the local cost in the address generators. Furthermore, as we show in this paper, this overhead can in practice be significantly reduced by an appropriate architectural style selection (counter/table or custom data-path based). Moreover, significant further optimization by advanced transformations for address equation merging and multiplexing for both type of architectures [7, 8] are possible while still meeting the timing constraints imposed.

Most of the work found in literature is concentrated on bit-level optimizations for counter based architectures [9, 6] with some interactive support for multiplexing hardware at that level [10]. However, time-sharing at the address sequence level which we have shown [7] to be crucial to reduce the overall cost are almost not considered for this architecture style. In this paper, we will improve on our earlier work and add several new contributions to support efficient exploration of the large search space when implementing the original index array references, as a complementary step to our work in custom data-path address calculation units [8].

2 Architectural Strategies

Address optimization/sharing is a very error-prone and tedious task if done manually for real-time mul-

[1] At the time of this research, on leave from the Electrical Engineering Dept. of the Czech Technical Univ.

ED&TC '97 on CD-ROM

tidimensional signal processing (RMSP) applications. Therefore, a formalized methodology has been developed which is supported by our **Ad**dress Expression **Opt**imization and Realization Environment (ADOPT) by a mix of automated and interactive tools [8].

2.1 Target Architectural Styles

For address generation, a fixed storage ordering has to be selected when multi-dimensional signals are presented. In our environment, we first optimize this order for in-place cost. Then, an **address equation** (AE) for every manifest index expression of a signal located in a standard linearly addressed RAM is generated. Typically, in our target application domain of RMSP, every index expression I_i defines an affine mapping of the iteration space to each dimension of the storage space. This leads to an AE which is a linear expression of the loop iterators: $it_1 + C_1 * it_2 + \ldots + C_{n-1} * it_n + C_0$, where $C_0, C_2, \ldots C_n$ are constants, and $it_1, it_2, \ldots it_n$ represent the loop iterators, it_1 being the innermost iterator.

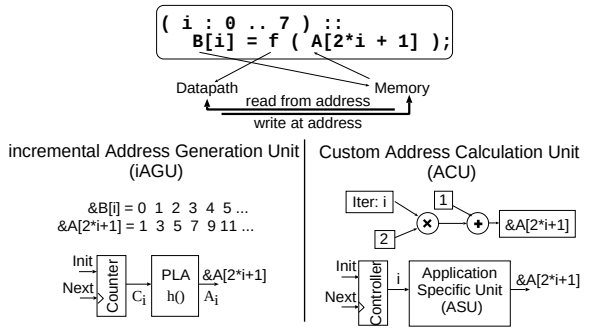

Figure 1: Illustration of counter based (iAGU) or custom data-path based (ACU) architectures in ADOPT.

ADOPT supports two different architectural styles for the implementation of an AE (see Figure 1). One possibility is to expand the AE in sequences of address values (A_i) which are then realized with modulo/binary counters with the outputs (C_i) modified by custom logic ($h()$) [9, 11]. In this way, the address generation process can be formulated by a recurrence equation: $A_i = h(C_{i-1} + M) = g(A_{i-1}) : i = 2 \ldots$, where M equals the modulus value of the modulo counter. Therefore, the hardware unit computes every next address incrementally, hence resulting in an **incremental Address Generation Unit** (iAGU). This architecture can simplify the hardware realization, e.g. because of the simpler control needed, but it requires implicit expansion of loops, so it is mainly intended for loops with limited iterator ranges ($\leq 10^5$).

The other possible architecture supported by ADOPT is motivated by the fact that in RMSP applications, the address expression constitute a direct function of the loop iterators, thus it can be mapped onto arithmetical operators. The resulting architecture is a **customized Address Calculation Unit** (ACU) realized with custom arithmetic building blocks selected from a library [8]. Here, use is made of a lowly multiplexed data-path style and script [12, 13], optimized for these ACUs, and of algebraic and space-multiplexing (merging) optimization techniques [14]. For both architectural styles, the resulting hardware is describe at the RT-level using VHDL modules that are then synthesized using commercial logic synthesis tools.

2.2 Architectural Trade-Offs

In ADOPT we have selected a control strategy for addressing based on the use of distributed local controllers completely decoupled from the system controller with address generators and data-paths synchronized by the **system clock** [8]. In this way, we localize the addressing hardware, hence reducing routing cost overhead. For iAGUs two control lines are generated from the local controller to obtain the next memory address, and to provide the initial state. However, for ACUs the complete set of iterator states has to be supplied by the local controller. Therefore it can be identified as a separate component apart from the actual customized arithmetical operator thus contributing to the individual address unit cost.

Depending on the characteristics of the address sequence, sometimes it is the least expensive to implement the address generator as a combination of a modulo/binary counter and a few logic gates. This is the case for memory accesses with a high regularity degree in their address sequence [7]. Furthermore, the size of this logic remains almost constant as the the length of the sequence increases (e.g., a butterfly FFT application [7]), and only the area associated to the counter increase slightly (logarithmically) with respect to the sequence length. If we compare the cost associated to the combinational logic in both styles, the area consumption in iAGUs can be about one or two orders of magnitude smaller (typically $0.001 \sim 0.01 mm^2$ in a $0.7 \mu m$ CMOS technology) than the average size occupied by just the arithmetical operator part of an ACU. Nevertheless, many other address sequences with very irregular access patterns are much more area "hungry" (up to $1 mm^2$ for the same technology), depending heavily on the sequence length. In this cases, the same address equation can be very efficiently implemented using an ACU, thus saving up to one order of magnitude in combinational area. Furthermore, the arithmetical operator part of the ACU cost only increase slightly (logarithmically) with respect to the

sequence length. The detection of such properties at the system level is addressed more formally in section 4.

Once the best target architecture style has been decided for each AE, further optimization can be done by merging hardware modules [14], or by sharing different address equations onto the same hardware unit [7, 8]. In general, merging optimizations do not change the benefits of performing architectural selection as a first step. However, it is not yet fully clear whether architectural decisions taken before time sharing will properly steer the search towards a global optimium. For instance, two AEs could be decided to be implemented as iAGUs based on some regularity criteria, resulting that their ACU shared version is less costly than the iAGU shared one. Nevertheless, architectural exploration can/must be used as method to prune the huge search space when order(s) of magnitude of difference between different target styles are present, which is indeed the motivation for our work.

3 System Level Optimization

Complex RMSP applications typically require hundreds of different AEs. Exploring all the possible candidates up to implementation level becomes clearly infeasible in terms of design time. Furthermore, if only one architectural style is assumed for the complete application, e.g. an ACU style, it could happen that a large part of the area overhead is being misused as many of the AEs could be much more efficiently mapped using iAGUs. Therefore, it is necessary to have a set of automated system level techniques to explore the different architectural trade-offs and to predict the impact of these decisions in an early stage of the design process.

```
/*** 2D-DCT KERNEL ***/
for (u=0; u<=7; u++) {
 for (v=0; v<=7; v++) {
  xtot = 0;
  for (x=0; x<=3; x++) {
   ytot = 0;
   for (y=0; y<=3; y++) {
    sum = In[x][y]+In[7-x][y]+In[x][7-y]+In[7-x][7-y];
    ytot =+ (cst_u_v_x_y * sum);
   }
   xtot =+ ytot;
  }
 }
}
/*** AUTO-CORRELATION KERNEL ***/
for (k=0; k<=63; k++) {
 r[k][k] = In[0]*In[k];
  for (j=k+1; j<=63; j++)
    r[k][j] = r[k][j-1] + In[j-1]*In[j];
}
```

Figure 2: Auto-correlation and 2D-DCT kernels used as test vehicles

Two algorithmic examples are used as test vehicles for our system level exploration methodology. Figure 2 shows the C-code descriptions of a two dimensional Discrete Cosine Transform (DCT) kernel extracted from a videophone[2] application [15], and of an Auto-Correlation kernel used in a mobile voice coder application [11]. They are relatively small examples so that they are still suitable for manual exploration and consequently manageable for illustration purposes. However, the possibilities for area saving are even more significant in full applications containing a much larger number of index expressions, especially in image and video processing. Therefore, a larger demonstrator will be discussed briefly in section 5.

3.1 Optimizing Techniques for iAGUs

The accurate selection of a suitable modulo value for counter based architectures has been one of the techniques proposed in the literature for improving area efficiency [9]. This selection is based on the analysis of the stream of individual address bits and it is important since the size of the mapping logic block depends on it. However, we believe that it is possible to achieve better area efficiency when raising the level of abstraction for analysis from bit to word level. For this purpose, we have developed within ADOPT a fully novel optimization technique called **sequence reduction**. We base our technique on the observation that many of the address sequences found in realistic applications exhibit (pseudo)periodic behaviors. Therefore, by detecting such properties and reducing the original sequence to a primitive one, which is being repeated (pseudo)periodically, it is possible to exploit a higher degree of regularity at the bit-level than in the original one. The reduction of the sequence is done at the word level which is suitable for our system level exploration methodology. The benefit is that the resulting hardware structure is smaller for two reasons: the counter is smaller, and the reduced sequence can be more regular, consuming much less mapping logic. The regularity in the address sequence arises from the regularity of the loop hierarchy in the source code where the sequence is generated by loop simulation.

Besides, we also benefit of bit-level optimization but over the reduced sequence instead of over the original one. Actually, the algorithm implemented in our iAGU synthesis tool (an extended version of HWAG [16]) is similar to the one proposed by Grant for optimal selection of modulo counters [9]. However, we concentrate on one modulo counter during the moduli determination algorithm. This leads to a significant reduction in CPU time for the analysis, so that larger sequence lengths can be treated. Optimal bit level assignment and the most efficient logic

[2]Due to symetry, only the code related to the first quadrant of the `x-y` plane is shown

block synthesis are left to the subsequent logic synthesis tools.

Given a certain storage order and, e.g. a two level loop hierarchy, there are four types where sequence reduction can be applied. These types are shown in Figure 3 for a simple example. Note that the number of different reduction types grows as 2^{nl}, nl being the depth of the loop nest where the AE is located. Sequence reduction techniques impose the need to define flexible counter descriptions for both: detection of specific reset states and flexible initialization schemes. Therefore, the extended HWAG tool makes use of VHDL to describe such flexible structures. Later on, they will be synthesized using existing RTL logic synthesis tools. Note that this implementation style doesn't increase the complexity of the global controller since the iAGU itself takes care about the detection of the specific reset state.

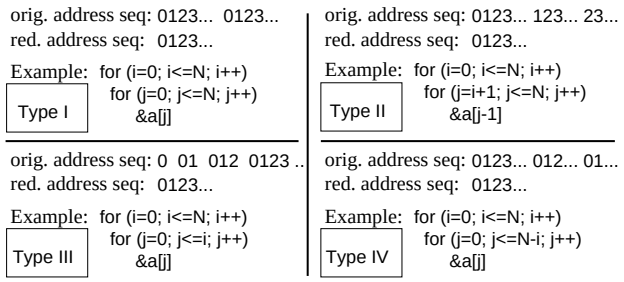

Figure 3: Types of reducible sequences for sequence reduction

Table 1 shows synthesis results for both test vehicles containing the index expressions where sequence reduction could be applied[3]. The area figures are in mm^2 and the target technology is $0.7\mu m$ CMOS standard cell. The reduction types are given between brackets in the *Length* column. The overall area savings obtained using sequence reduction are clearly demonstrated for both applications (59% for the 2D-DCT and 27% for the Auto-correlation). For the 2D-DCT the saving is mostly obtained due to a reduction in the counter bitwidth and not in the mapping logic since the original sequence is already very regular. However, for the Auto-correlation the overall saving is mostly obtained in the mapping logic, since by reducing two of the address sequences of the application (In[j] and In[j-1]) we can convert them into two periodic and highly regular sequences, thus considerably reducing the need for mapping logic. In this case, the effect on the counter is not so visible since although the reduced sequence is much shorter than the original one (64 instead of 2016 addresses), also more logic is required in order to implement the checking of the reset state and its associated increment. This shows the dual effect of sequence reduction techniques.

Expression	Before/After Seq.Reduction			
	Length	Counter	Logic	Total
2D-DCT				
In[x][y]	1024/16 (I)	**.05/.02**	.00/.00	.05/.02
all address	—	**.20/.07**	.01/.01	.21/.08
Auto-Correlation				
In[j]	2016/63 (II)	.05/.06	**.65/.01**	.70/.07
all address	—	.30/.30	**2.3/1.6**	2.6/1.9

Table 1: Area savings using sequence reduction techniques for the test vehicles.

4 Architectural Selection

This subsection describes the basic considerations and practice of how to select an appropriate target architecture for a given AE. We will show for both test-vehicles that the selection of the right architectural style for each AE can be crucial for arriving at area efficient solutions. In order to avoid an exhaustive search up to the implementation level, the choice should be guided very early in the exploration. For this purpose we define a set of heuristics based on two types of measures: a regularity measure [7] ($Regul$), and a sequence reduction measure ($Reduc$). The regularity measure is used to project the need of logic in the mapping between counter and address bits. The $Reduc$ measure is used to measure the possibilities of sequence reduction for the AE under analysis. These numbers, $Regul$ and $Reduc$ must be computed for each address sequence, by first selecting the $Reduc$ number, and over the resulting reduced sequence, the $Regul$ number. In practice this can happen very fast. Note that the $Reduc$ measure is produced already after sequence reduction analysis (see section 3.1).

$$Reduc = 1 - \frac{new\_\#cnt\_bits}{orig\_\#cnt\_bits};\ Regul = \frac{\#regular\_bits}{\#address\_bits} \quad (1)$$

As the purpose of the regularity measure ($Regul$) is to project the need of logic to implement the mapping function (in case of iAGUs), a simple *regularity* heuristic is then expressed as the fraction of address bits generated directly by some of the counter bits (no logic needed) with respect to the total number of address bits [7]. A value of 1.0 of $Regul$ indicates that all address bits can be generated directly by the counter bits. A value of 0.0 indicates a need of special logic to generate each address bit. This is indeed the worst case, although no information is given about the amount of logic needed. In order to obtain at the system level the proportion of counter bits that actually need mapping, we have implemented a very fast

[3] Results for indexed accesses In[7-x][y], In[x][7-y], In[7-x][7-y] and In[j-1] are not shown since they are identical to In[x][y] and In[j] respectively.

and simplified version of the algorithm proposed by Grant. In [9] the intention was to reduce the amount of mapping logic by finding optimal assignments between counter and address bits. The first simplification we assume for our estimate is to take as a reference the sequence generated by a binary counter. Then we check whether any stream of individual address bits fits into any bit-stream of a binary counter. The second simplification is that we don't proceed in looking for XOR-ed combinations of counter bits since this process takes too much CPU time. Instead, the *Reduc* measure is used to express the fraction of counter bits that are saved by sequence reduction with respect to the number of counter bits needed to implement the original sequence (see equation 1).

A set of rules based on design considerations for the correct interpretation of the measures are given below:

i. If the *Regul* measure is high enough, typically ≥ 0.3, then the selected AE is a good candidate to be implemented by means of an iAGU, irrespective of the vale of *Reduc*. Indeed, address sequences having ~30% of regularity at the bit level do not require much logic for mapping the non-regular bits. Note that we are not dealing with fully random scalar-level allocation schemes that might lead to non-linear address spaces and thus very random memory access references.

ii. Clearly, values of $Regul \sim 0.0$ indicates that the ACU style is preferred, because the address sequence is already quite irregular.

iii. When the sequence can be reduced drastically ($Reduc \geq 0.5$), and the resulting sequence length is small, then it is recommended to implement it using an iAGU, even if the *Regul* number is low (typically $\leq 0.1 \sim 0.2$).

iv. Special care has to be taken when some address equations (AE_j) have already been decided to be implemented using ACUs. If the current regular AE_i is belonging to the same loop scope as AE_j, and AE_i is simple enough (e.g., containing just one iterator in its expression: $AE_i = C_i * it + C_0$), then the preferred target architecture is an ACU even for very regular access. The reason is that the local controller needed to feed AE_i is much costly than its data-path and it can be reused from AE_j.

The need for the rules is motivated by the fact that we can now explore directly at the system level whether a certain AE can be efficiently mapped onto an iAGU. Note though that no information is available yet about the real cost difference between both target styles. For ACUs, it would be only feasible to obtain an estimate of its associated cost if part of the synthesis trajectory is performed. The same applies to iAGUs, where an estimation of the cost of the PLA could be developed by examining the minimum number of minterms of its associated table, but this would imply to perform part of the logic synthesis trajectory which takes too much CPU time.

5 Exploration results

In this section, we illustrate the benefits of architectural selection for hardware addressing. Experiments over both examples were targeted to compare area costs for architectural styles and to check the accuracy of the measures and rules. The *Regul* and *Reduc* numbers have been computed for every sequence of both test vehicles in Figure 2. Table 2 shows the rule (between brackets) applied in each case as well as the area consumption for each target architecture. The iAGU area figures are given after sequence reduction. A clear match (shown in bold) can be seen between the choice suggested by the rules and the actual cost for both styles after detailed realization. Hence, our rules can be used to decide the choice.

Expression	Measures	ACU		iAGU	
	regul/reduc	ctl/asu	Tot	cnt/pla	Tot
2D-DCT					
In[x][y]	1.0/.60 (i)	.05/.00	.05	.02/.00	**.02**
In[7-x][y]	.67/.60 (i)	.05/.00	.05	.02/.00	**.02**
In[x][7-y]	.60/.60 (i)	.05/.00	.05	.02/.00	**.02**
In[7-x][7-y]	.33/.60 (i)	.05/.00	.05	.02/.00	**.02**
Auto-Correlation					
r[k][k]	1.0/.00 (i)	.03/.01	.04	.03/.01	**.04**
In[k]	1.0/.00 (i)	.03/.02	.05	.03/.00	**.03**
r[k][j]	.00/.00 (ii)	.07/.01	**.08**	.05/.78	.83
r[k][j-1]	.00/.00 (ii)	.07/.01	**.08**	.05/.78	.83
In[j]	1.0/.45 (iv)	.07/.00	**.07**	.06/.01	.07
In[j-i]	1.0/.45 (iv)	.07/.00	**.07**	.06/.01	.07

Table 2: Area costs and *Reduc* and *Regul* measures for the different index expressions.

The benefit of architectural exploration can be also obtained even for already optimized designs by merging techniques. In this context, we performed new experiments merging the counters and local controllers of those AEs that belongs to the same loop scope. For the 2D-DCT kernel the selected architecture was counter based (see Table 2) and the overall cost was 0.08 mm^2, **2.5 times** less than using only ACUs. However, the Auto-correlation kernel leads to a mixed architecture, although very close in cost to an ACU based one, consuming 0.37 mm^2 (0.39 for an only-ACU case) which is about **4 times** less than usign only iAGUs. Therefore, for both applications the rules steered very accurately the selection process even after merging.

Nevertheless, the relative figures are only important

here as the absolute figures are much larger in full applications. To illustrate this, a set of experiments over a larger application was performed. In this case, we selected index expressions for one of the two arrays present in full-search 2D Motion Estimation kernel. The algorithm is part of a video encoder application targeted to realistics frame sizes of 256x256 pixels, with a 23x23 pixels neighborhood size, and using a DCT block of 8x8 pixels.

Access	Measures		ACU	iAGU
	regul/size	secs	ctl/asu	cnt/pla
A_269	**.35/64** (i)	0.1	.04/.02	**.03/.01**
A_73	**.29/68** (i)	0.1	.04/.02	**.03/.01**
A_115	.18/448 (ii)	0.2	.06/.03	.04/.24
A_17	.18/476 (ii)	0.2	.06/.03	.04/.24
A_151	.12/448 (ii)	0.2	.06/.08	.04/.35
A_53	.12/476 (ii)	0.2	.06/.08	.04/.35
A_239	**.35/1024** (i)	0.4	.07/.02	**.05/.01**
A_3	.18/4352 (ii)	0.9	.07/.12	.07/.19
A_133	.12/3136 (ii)	0.7	.07/.10	.06/2.1
A_35	.12/3332 (ii)	0.7	.07/.10	.06/2.3
A_101	.12/4096 (ii)	0.8	.07/.13	.06/.51
A_183	.18/7168 (ii)	1.3	.09/.03	.06/3.8
A_219	.12/7168 (ii)	1.3	.09/.09	.06/6.7

Table 3: Area costs and measures for the different index expressions for the Motion Estimation application

Table 3 shows the results after architectural exploration for this application. The first column displays the regularity[4] values for each AE, the size of the sequence, and the rule (between brackets) applied for architectural selection. Also, the time spent on the computation of the measures using a HP-715/50 workstation are displayed. CPU numbers clearly demonstrate that the proposed measures are very suited for architectural exploration since they are suficiently small and their dependency with the size of the address sequence remains linear. Finally, the area figures for the two types of architectures are also shown. The Table is clustered in 5 groups of AEs with approximately the same sequence length. Inside each cluster the AEs are sorted according to the *Regul* values, listing first those AEs with higher values. Note that the heuristic allows to make the right decisions even for AEs of different sizes, as demonstrated by accesses: A_219, A_73 and A_239.

6 Conclusions

In this paper, we exploit regularity properties at the bit level and (pseudo)periodic properties at the sequence level to significantly reduce area cost for counter based address realization architectures. Moreover, we define a set of easy computed system level criteria for a quick selection of the best suited target architecture (data-path or counter based) which is based on two types of measures. This is essential due to the huge search space present in hardware address generation. The principles are demonstrated on two small but realistic test-vehicles, obtaining significant reduction on area overhead. Also, very promising results are indicated for larger applications too.

[4] This application does not contain (pseudo)periodic memory accesses references, therefore $Reduc = 0.0$ for all entries.

References

[1] K.Kitagaki, T.Oto, T.Demura, Y.Araki, and T.Takada. A new address generation unit architecture for video signal processing. *Visual Communications and Image Processing'91:Image Processing*, Vol. 1606:891–900, 1991.

[2] Texas Instruments. *TMS320C5x User's Guide*, 1993.

[3] C.Liem, P.Paulin, and A.Jerraya. Address calculation for retargetable compilation and exploration of instruction-set architectures. In *Proc. 33rd. ACM/IEEE Design Automation Conf.*, pages 597–600, 1996.

[4] G.Araujo, A.Sudarsanam, and S.Malik. Instruction set design and optimizations for address computation in DSP architectures. In em Proc. 9th ACM/IEEE Intl. Symp. System Synthesis, pages 102–107, 1996.

[5] S.Wuytack, F.Catthoor, L.Nachtergaele, and H.De Man. Power exploration for data dominated video applications. *IEEE Intl. Symp. on Low Power Design*, 1996.

[6] P.Lippens, J.Van Meerbergen, A.Van der Werf, W.Verhaegh, B.McSweeney, J.Huisken, and O.McArdle. Phideo: A silicon compiler for high speed algorithms. In *Proc. 2nd ACM/IEEE Europ. Design Automation Conf.*, pages 436–441, 1991.

[7] M.Miranda, F.Catthoor, and H.De Man. Address equation multiplexing for real time signal processing applications. In *VLSI Signal Processing, VII*, pages 188–197, New York, 1994. J. Rabaey, P. Chau and J. Eldon (eds.). IEEE Press.

[8] M.Miranda, F.Catthoor, M.Janssen and H.De Man. ADOPT: Efficient hardware address generation in distributed memory architectures. In *Proc. 9th ACM/IEEE Intl. Symp. System Synthesis*, pages 20–25, 1996.

[9] D.Grant and P.Denyer. Address generation for array access based on modulus m counters. In *Proc. 2nd ACM/IEEE Europ. Design Automation Conf.*, pages 118–122, 1991.

[10] D.Grant, J.Van Meerbergen, and P.Lippens. Optimization of address generator hardware. In *Proc. 5th ACM/IEEE Europ. Design and Test Conf.*, pages 325–329, 1994.

[11] B.Vanhoof, I.Bolsens, S.De Troch, L.Philips, J.Vanhoof, and H.De Man. Design of a voice coder with the cathedral-II silicon compiler. In *Proc. of the User Forum and EuroASIC prizes, EDAC-EuroASIC-93*, pages 150–153, 1993.

[12] W.Geurts, F.Catthoor, and H.De Man. Time constrained allocation and assignment techniques for high throughput signal processing. In *Proc. 29th ACM/IEEE Design Automation Conf.*, pages 124–127, Anaheim, CA, June 1993.

[13] S.Note, W.Geurts, F.Catthoor, and H.De Man. Cathedral-III: Architecture driven high-level synthesis for high throughput DSP applications. In *Proc. 28th ACM/IEEE Design Automation Conf.*, pages 597–602, 1991.

[14] J.M.Janssen, F.Catthoor, and H.De Man. A specification invariant technique for operation cost minimization in flow-graphs. In *7th ACM/IEEE International Symposium on High-level Synthesis*, pages 146–157, 1994.

[15] S.Vernalde, P.Schaumont, I.Bolsens, and H.De Man. Synthesis of high throughput DSP ASICs using application specific data-paths. *DSP & Multimedia Technology*, 3(6):13-21, June 1994.

[16] M.Kaspar, P.Schaumont, I.Bolsens, and H.De Man. Incremental hardware address generation. In *Proc. Workshop on Design Methodologies for Microelectronics*, pages 153–159, 1995.