Physical synthesis for CPLD architectures

Sid-Ahmed Senouci
Mentor Graphics, Grenoble, France

Abstract—In this paper, we present a new synthesis feature namely, "Xor matching", and the foldback product term synthesis for Complex Programmable Logic Devices (CPLD) architecture that is based on PAL-like macrocells. Our goal is to use the Xor gate and the foldback terms, (or shareable expander – Altera equivalent terminology [17]), available in each macrocell for minimizing the number of macrocells required to implement a circuit. We propose two innovative approaches: the first, is a very fast algorithm which always gives a match for a function onto the Xor gate of the CPLD device, when one exists; the second approach, is based on answering a fundamental problem: determine if a given foldback cluster can be assigned to a PAL block. A foldback cluster is defined as a set of functions and sub-functions that result in the same foldback which is created by the foldback decomposition algorithm. A suite of test cases (MCNC) were tested with device-fitting algorithms targeting the Atmel CPLD device (ATF15xx series) which implemented the corresponding hardware resources.

I. Introduction

The growth of the programmable logic market is a non-debatable success in the IC world over the last decade. CPLDs and FPGAs share this success in a balanced way and seem to be perfectly adapted to complementary needs. Most FPGAs have logic blocks based on look-up-tables (LUTs) , and some have multiplexer-based logic blocks. CPLDs are based on Programmable Array Logic style macrocells. Each macrocell can implement any Boolean function of up to *k* inputs and with no more than *m* product terms. Figure 2 shows the structure of an Atmel's CPLD macrocell. It consists of Xor gate, foldback (or shareable expander – Altera equivalent terminology [17]), cascade (or parallel expander – Altera equivalent terminology [17]) and a flip-flop (Architecture of the macrocell is detailed in section II.1). An FPGA offers high logic density, high capacity and somewhat unpredictable delays, as a critical path may need to go through multiple levels of logic cells connected by programmable interconnections. On the other hand, CPLD offers lower logic density and predictable delays, as a critical path may need to go through fewer levels of macrocells. It is well known that fast decoders, finite state machines are the favorite application field of CPLD whilst FPGA products have become aggressive in offering efficient high complexity logic and in embedding RAM or hard blocks in their architectures.

In the last decade, the synthesis problem for FPGAs has been widely addressed. Many LUT-based technology mapping, placement and floor-planning algorithms have been presented [5,6,7]. However, only some studies have been proposed on the synthesis problem for CPLDs, and consequently, very few mapping algorithms have been published [1,2,3,4].

[1] presented an approach that allows existing multi-level synthesis techniques to be adapted to implement circuits that are well-suited for CPLD architectures. The TEMPLA flow includes three phases: optimal tree mapping, heuristic partial collapsing–, and bin packing. The objective of this algorithm is to minimize the number of PLAs. In [2] PLAmap was developed to minimize the delay of mapped circuits. The PLAmap algorithm breaks the technology mapping problem into three phases: labeling, mapping and packing. [3] proposed *k_m_flow* as a technology mapper for single-output PLA-like macrocells, with both inputs and product term. The *k_m_flow* algorithm consists of two phases: labeling the network and mapping the network into *k/m*-macrocells. In [4] area/depth is the primary goal where the algorithm takes advantange of existing LUT mapper for single-output PLAs and packing for multiple-output PLAs. Most these algorithms are based on PLA architecture, without taking into account the macrocell architecture.

This paper focuses on innovative techniques in the synthesis flow for complex programmable logic devices (CPLDs) especially for those containing foldbacks, which are like an inverted product term, and an Xor block such as (Pterm $\oplus$ Sum of Pterms). For Xor synthesis, the detected Xor templates become "don't touch" subfunctions similar to other macros such as arithmetic blocks generated by parameterized macrogenerators. Furthermore the foldback decomposition, which is a pseudo factorization, is performed if and only if a global cost evaluation in terms of the potential number of macrocells shows a gain. As foldback product terms can only be used within a Programmable Array Logic (PAL), the combinational logic using a foldback product term has to be put in the same PAL (Fig 2). Thus the foldback cluster is created. This cluster assignment is somwhat similar to the cone-based approach used for timing-driven floorplan [8], but here the clusters give priority to foldback gains. The aim of the Xor and foldback detection procedures is to reduce the number of macrocells required to implement a circuit. We propose a very fast Xor matching algorithm based on algebraic theory[9]. We have also successfully reduced the number of macrocells by applying the foldback decomposition. Finally, we address the assignment problem on the PAL block.

The rest of the paper is organized as follows: after having introduced the CPLD features we are dealing with, the global flow is presented in section 2. In section 3, we give basic definitions, notations and problem formulation of

the Xor detection. Definitions, foldback decompostion algorithm and assignment problem are presented in section 4. Experimental results and conclusions are presented in sections 5 and 6.

II. General approach

II.1 CPLD features

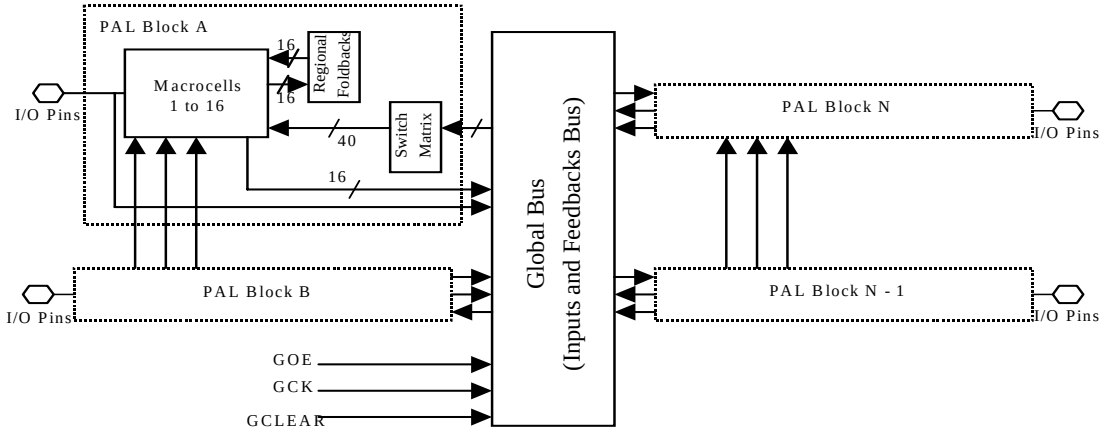

Fig 1: ATF15xx Family of CPLD

Atmel's ATF15xx family of CPLDs [16] will be used as an example throughout this paper. The architecture is a classical CPLD structure partitioned into PAL blocks. Each PAL consists of a set of macrocells and a switch matrix block. All PAL blocks are linked together via the global bus (Fig 1), which contains all input and I/O pin signals as well as the buried feedback signals from all macrocells. The switch matrix in each PAL block receives as its inputs all signals from the global bus in their true and inverted form. These signals can be selected as inputs to the individual PAL blocks by the fitter software. Each input signal or feedback signal has access to a few multiplexers from switch matrix block, thus greatly limiting the available opportunities to route a global bus signal.

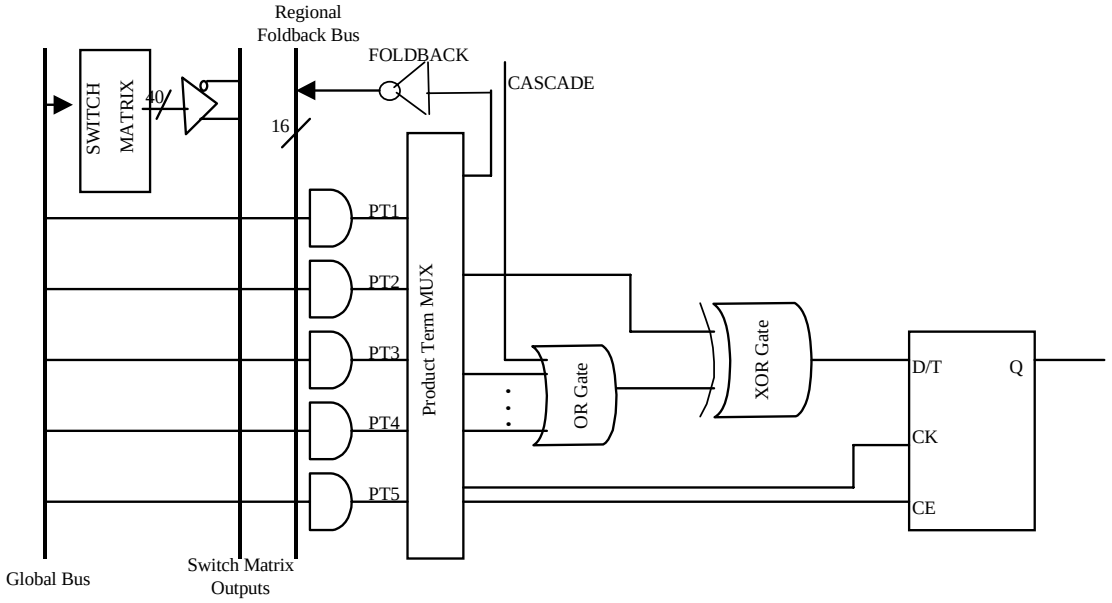

Fig 2 : ATF15xx Macrocell

Figure 2 illustrates the macrocell architecture. Each macrocell consists of product terms, a product term select multiplexer, OR gate, Xor gate, Cascade logic (or parallel expander), foldback bus, a flip-flop/latch and an output buffer. The product term select multiplexer (PTMUX) allocates five product terms as needed to the macrocell. Within a single macrocell, all the product terms can be routed to the OR gate. For the Cascade capability, the number of product terms can be extended from neighboring macrocells, up to 40 as long as space is available in the same PAL. An Xor gate exists at the output of the macrocell for each Boolean function. One input to the Xor gate

comes from the OR sum product terms (SOP). The other Xor gate input can be a product term (PT). Thus he the macrocell's Xor gate allows efficient implementation of any Boolean function that can be expressed as $(PT \oplus SOP)$. Each macrocell can also generate a foldback product term. This signal goes to the regional bus and is available to all the macrocells in a given PAL block. The foldback is an inverse polarity of one of the macrocell's product terms. Lastly, the flip-flop has very flexible data and control functions and can be configured for D and T operation.

II.2 The global design flow

The global design flow takes as input a netlist which is the output of a higher level compiler (VHDL, CUPL, Verilog or ABEL Compiler) as well as user constraints (Fig 3).

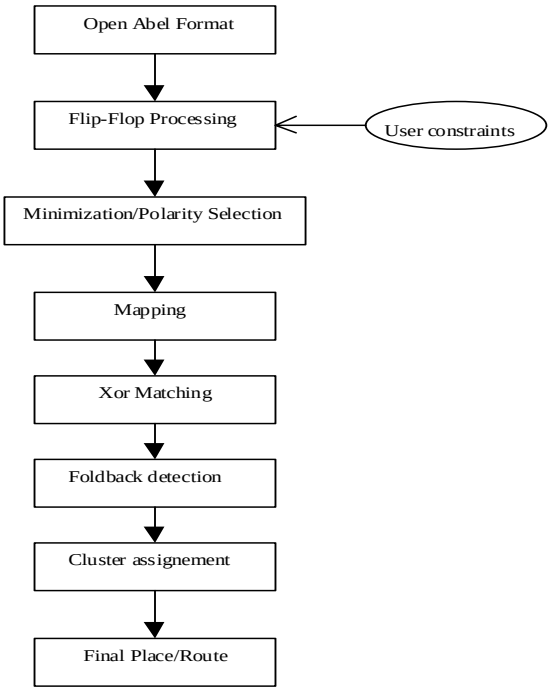

Fig 3: Global design flow

After a classic polarity selection phase based on product term number minimization (ESPRESSO), a mapping step is called. Thereafter, the Xor matching and Foldback decomposition stages are considered. The first step detects Xor expressions that are of interest for CPLD synthesis, namely expressions having the following form $(PT \oplus SOP)$, where *PT* is a Product Term and *SOP* a Sum of Product terms Boolean expression. The second step is a Foldback product term detection/selection. It will be decided at this point if the use of a foldback product term appears to be suitable for the initial functions or subfunctions. These approaches will be explained in detail in this paper. The Xor matching and Foldback decomposition are performed if and only if a global cost evaluation in terms of potential number of macrocells shows a gain. After the foldback detection step, foldback clusters are created. As foldback product terms can only be used within a PAL, the logic utilizing a foldback product term has to be put in the same PAL. Unfortunately, the presence of foldback clusters may lead to a difficult fitting process. Nevertheless, it is interesting to use the foldback physical pattern and to perform minimization on logic as required. The assignment of foldback clusters consists of two phases: first, control and size limitation of the foldback clusters are processed. The second phase is to check if each cluster input is affected to one and only one input of the PAL block. This will lead to the final placement/routing that will not be considered at all in this paper.

III. Xor Matching for CPLD architectures

Xor operators commonly appear in high level descriptions and are preserved through the RTL synthesis path. It may happen, particularly in the CPLD world, that Xor expressions are flattened and one of the tasks of this paper is to reintroduce them in order to take full advantage of the available Xor gates of the CPLD device.

This study is therefore completely different from the basic problem of expressing Boolean functions using the Xor operator. It is well known that a large amount of published literature exists on Reed-Muller expressions [15]. In this section we present a matching algorithm for the Xor gate of the CPLD device based on the necessary

conditions theory [9].

III.1 Problem formulation

Definition: Consider the Xor_CPLD expressed in the following form $PT \oplus f$, where *PT* is a product term of a macrocell and f is a sum of product terms of a macrocell (Fig 2).

Example : Let *F* be an Xor Boolean function :

$$F = ab \oplus (cd + gh) \qquad \text{(i)}$$

After flattening and minimization , *F* can be expressed as :

$$F = \bar{a}cd + \bar{b}cd + \bar{a}gh + \bar{b}gh + ab\bar{c}\bar{g} + ab\bar{c}\bar{h} + ab\bar{d}\bar{g} + ab\bar{d}\bar{h} \qquad \text{(ii)}$$

The macrocell architecture is based on five product terms.

In (i), use 1 macrocell to implement *F*, but in (ii), use 2 macrocells to implement *F*.

After the Xor extraction, we reduce the number of macrocells required to implement *F*, and the gain is equal to one macrocell.

The problem selected here as expressed above is rather a rewriting of the Boolean function equivalent to $PT \oplus f$ in which such a form has been hidden by a flattening process.

Given a Boolean function, *F* is a sum of product terms expression. We assume that *F* is expressed in minimum support. In this section we try to use the new approach to detect if *F* can be implemented by an Xor_CPLD. *F* can be expressed as follows: $F = Pt \oplus g$ (1)

Where g is a sum of product terms expression.

The input variable supports of Pt and g are disjoint. This corresponds to the CPLD physical pattern. By applying this restriction, we obtain a much faster algorithm. The aim of the procedure presented in this subsection is to extract a product term *Pt* having a maximal number of literals called "Xor cofactors" from the Boolean function *F*.

III.2 Necessary conditions verified by literals of a XOR cofactor

Let *F* be a minimized sum of product terms expression.

$$F = \sum_{i=1}^{n} m_i \quad \text{where } m_i \text{ is the product term.} \qquad (2)$$

Let us suppose that F can be implemented by an Xor_CPLD, then F has a decomposition as shown in (1).

Where *Pt* is Xor cofactor.

The problem reduces to finding *Pt*, which is an Xor cofactor.

$$Pt = x_1 x_2 ... x_p \qquad (3)$$

$$F = \left(x_1 x_2 ... x_p \right) \oplus g \qquad (4)$$

Proposition III.2.1: *If Pt is an Xor cofactor, then all the inputs of Pt have the same occurrence in a decomposition of F as in (2).*

Let x_i, x_j be two inputs of Pt and $Occ(x_i), Occ(\overline{x_i})$ be respectively the occurrence of the literals $x_i, \overline{x_i}$ in (2).

An input occurrence be expressed as follows: $SumOcc(x_i) = Occ(x_i) + Occ(\overline{x_i})$

From (4) we have:

$$F = \left(x_i x_j ... x_p \right)\bar{g} + \overline{\left(x_i x_j ... x_p \right)}g \qquad (5)$$

With g being independent of all inputs of Pt :

$$g = \sum_{i=1}^{k} p_i \quad \text{and} \quad \bar{g} = \sum_{i=1}^{h} l_i$$

where p_i and l_i are product terms then : $SumOcc(x_i) = SumOcc(x_j) = k + h$

Proposition III.2.2: *Xor cofactor Pt exists if,* $\left\{ \forall x_i, x_i \in Lt(Pt), F_{x_i} = g \right\}$.

Where $Lt(Pt)$ *is a set of literals* of Pt . *We compute* F_{x_i} *or* $F_{\overline{x_i}}$ *according to this literal* x_i *appears under a direct or complemented form in Pt .*

We know g is independent of all inputs of Pt and using (4) we then have: $F_{x_i} = F_{x_j} = ... = F_{x_p} = g$

III.3 Xor_CPLD matching algorithm

The approach used here consists of two phases. If $F = Pt \oplus g$, then obvious necessary conditions exist dealing with the occurrence of the literals of *Pt*. A first phase aims at detecting these necessary conditions of identifying the *Pt* literals. Having identified the variable candidates, a second phase based on ROBDDs (Reduced Ordered Binary Decision Diagram) verifies if *F* is equivalent to a $Pt \oplus g$ expression.

III.3.1 Binary decision diagram

We assume here that the reader is familiar with
binary decision diagrams and all related numerous
BDD packages [9][10].

Example1: The ROBDD templates of $a \oplus b$ (Fig 4)

Example2: We begin the construction of the ROBDD following an order of variables which starts with the Pt variables. The ROBDD templates of $Pt \oplus g$ is shown in (Fig 5).

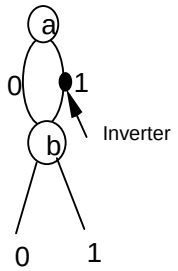

Fig 4

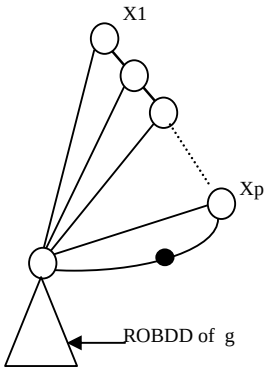

Fig 5

III.3.2 Algorithm

Based on this theory, we describe the algorithm that checks whether *F* can be implemented by an Xor_CPLD gate. The Boolean function F is minimized once. Let *S(F)* be the set of inputs of *F*. We perform this check only if we need more than one macrocell to implement *F*. The matching steps are done as follows:

1. $\forall x_i \in S(F)$, compute $SumOcc(x_i)$ and use *SumOcc* value to sort *S(F)* in classes.
   Each class is an Xor cofactor candidate.

2. Select Xor cofactor candidate as one with greater number of inputs, then we construct a ROBDD for this candidate using a variable ordering such that the inputs of the Xor cofactor candidate appear on the top of the ROBDD (Fig 5).

3. Check for each input x_i of the Xor cofactor candidate if $F_{x_i} = g$. This check is performed by verifying in the ROBDD template (Fig 5) if the node *g* is reached by only direct (complemented) from the nodes $x_i, ..., x_{p-1}$ and by direct edge and an inverted edge from the node x_p .

4. To check if F is equivalent to a $F = Pt \oplus g$ expression. If so, report a match and quit.
   Otherwise return to step 2.

IV. Foldback synthesis

A foldback is a product term *PT* of a macrocell which is inverted and fed back into the PAL logic array for use by any or all of the macrocells in the same PAL block (Fig 2) (i.e this signal is available to all macrocells within the same PAL block). The foldback terms in each PAL can also generate additional fan-in sum terms with small additional delays.

IV.1 Basic definitions:

Definition 1: Cost of a Boolean expression

Consider a Boolean function *F* expressed as a sum of product terms. The cost of the function *F,* denoted as Cost (*F*), is the number of product terms.

Definition 2: foldback candidate

A foldback candidate is a Boolean expression having the form $(x_1 + x_2 + \overline{x_3} + ... + x_k)$ where x_i is an input variable or a subfunction (shared or not) under a direct or complemented form and denoted by the term $Node_{FB}$ which is always inverted.

Definition 3: Cost of foldback

The cost of a foldback subfunction selected as an admissible foldback candidate is 1. This means that it corresponds to the foldback physical pattern.

We detect a foldback factor of two or more product terms of a set of functions.

Example1: Let *F* be a minimized sum of the product terms expression.

$F = abde + abce + abe\overline{f} + gh + \overline{h}lm + np$ and Cost (*F*) = 6

yields a foldback factoring as follows:

$F = abe(d + c + \overline{f}) + gh + \overline{h}lm + np$, $Node_{FB} = (\overline{cd}\, f)$ and Cost ($Node_{FB}$) = 1

then: $F = abe\overline{Node_{FB}} + gh + \overline{h}lm + np$ and Cost (*F*) = 4

After the foldback decomposition the gain is equal to 2 product terms.

Example 2:

The following case does not generate a foldback factor.

$F = abe + abcd = ab(e + cd)$

In this case we do not have the foldback form since, the expression $(e + cd)$ is not a foldback factor.

IV.2 Foldback detection algorithm

The algorithm proceeds like a "pseudo" factorization with a filtering of all the factors which are not the foldback form $(x_1 + x_2 + \overline{x_3} + ... + x_k)$, where x_i is an input variable or a subfunction (shared or not) under a direct or inverted form. We describe the foldback detection algorithm as follows:

1. Form cliques of functions by specifying number of literals in common.
   For each clique of functions.

2. For each function in clique,
   Create all foldback product terms for both onset and offset expressions and for each such foldback product term
   a. Determine if the identical foldback product term is derived from other product terms of this function or from product terms of other functions of the clique.
   b. Form an association with this foldback of each function of the clique which will factor into this foldback.

When all foldback product terms of the clique are determined,

3. For each foldback in the clique, For each function in the clique,
   a. Determine if the function factors into that foldback, onset, offset, or both onset and offset.
   b. If the use of the foldback is indicated, determine whether or not a reduction in the number of product terms occurs by the use of the foldback for instance, if foldback factoring is designated for the onset specification, compare the onset implementation with the expander against the offset implementation without foldback to determine if use of the foldback results in a reduction in the number of product terms.
   c. If there is a reduction in the number of product terms through use of the foldback, accumulate the number of saved product terms for all functions of the clique.

4. Select that expander which yields the optimal savings in number of product terms in the clique
   a. Create a foldback node for that foldback.
   b. For each function in the clique which yields a product term reduction by use of that foldback, form the appropriate function implementation having that foldback node as a literal, both onset, offset when appropriate.
   c. Remove the designated expander from the list of clique foldbacks.
   d. Perform step 2 on newly created function implementations, that is, determine if the newly factored function, with foldback implementation, can be factored again into foldbacks which may have a common use in the clique, allowing factoring into multiple foldbacks.

5. Return to step 4.

IV.3 Foldback cluster assignment

The preparation of the fitting process needs to answer a basic question whether a given foldback cluster candidate can be assigned to the PAL block.

IV.3.1 Definitions

Definition 1: The PAL block architecture is defined as the number of inputs, denoted as *I*, the number of macrocells, denoted as *M* and the number of outputs, denoted as *O*.

Definition 2: In a PAL block, each pin signal is routed to X Muxs. This means that there are X possibilities for routing each pin signal into a PAL block.

Definition 3: In a PAL block, each feedback is routed to Y Muxs , providing Y possibilities for routing this signal in to the PAL block.

Definition 4: The number of inputs of the PAL block is equal to the number of Muxs of the switch matrix block.

Definition 5: The foldback cluster, denoted as CL_{FB}, is defined as the set of functions and subfunctions that have the same foldback. Let $IN(CL_{FB})$ be the set of inputs or subfunction of CL_{FB}, $MC(CL_{FB})$ as the number of macrocells and $OUT(CL_{FB})$ as the number of outputs.

IV.3.2 Problem formulation

The assignment problem for the foldback cluster can be formally stated as follows: given a cluster of the functions and the basic PAL block , the cluster is assigned to a PAL block. This problem is solved by checking two conditions: first that the PAL block capacity is respected. Secondly, that if each input (or output, or feedback) of cluster is affected by one and only one input (or output, or feedback) of the PAL block. Is means finding a free Mux of the switch matrix in the PAL block to route the input (or output, or feedback). We denote an input (or output, or feedback) of the foldback cluster as an element.

Definition: Let $ELT(CL_{FB})$ be the set of elements of CL_{FB}.

Property: If a cluster of functions has been assigned to a PAL block, then all the functions of this cluster have the same foldback.

Proposition: A CL_{FB} is said to be assigned to the PAL block if and only if:

(1). $|IN(CL_{FB})| \le I, |MC(CL_{FB})| \le M$ and $|OUT(CL_{FB})| \le O$

(2). Each element of CL_{FB} is assigned to one and only one Mux of the switch matrix in a PAL block.

Proposition (1) is a quick and easy check. For proposition (2) we propose the problem formulation. We have a set of elements of CL_{FB} and a set of Muxs of the PAL block. These are the two kinds of nodes in our bipartite graph. We know which Mux can be routed with elements of flodback cluster *(Definition 2 and Definition 3)*. This defines the edges of our bipartite graph. Our aim in the maximum cardinality matching problem is assigning elements to Muxs in such a way that as many Muxs as possible are used by an element that can handle it. One element can be assigned to at most one Mux and we can assign at most one element to one Mux. At this point, the problem can be expressed more naturally in graph theory terms. A bipartite graph *G = (U, V, E)* with vertex sets *U* , *V* and edge set *E*, where *U* is a set of inputs of CL_{FB}, *V* is a set of Mux of switch matrix in PAL block and $E = \{(u,v) | u \in U, v \in V\}$. If *(u, v)* is an edge, i.e, $(u,v) \in E$, then there exists a possibility of routing the vertex *u* by the vertex *v (Definition 2 and Definition 3)*. Obviously, the problem reduces to a classical bipartite matching problem and the solution consists in performing a maximum matching algorithm in a graph *G* [10,11]. In conclusion, the foldback cluster can be assigned to the PAL block if there exists a maximum cardinality of a matching in a graph *G* equal to $|ELT(CL_{FB})|$.

V. Experimental results

The techniques presented above have been implemented in C language and incorporated in new Atmel EDA tools. To assess the results produced by Atmel fitters, these were compared to Altera's MAX+PLUS II tool (version 9.5). Table 1 shows the analysis of the results for MCNC benchmark circuits. For each benchmark, 2 indications are given. For Atmel, the first one gives the number of macrocells (**NberMC**) and the second gives the number of foldbacks (**NberFB**). The results of Altera are obtained by trying two different synthesis options with optimization for area (index 0) : Normal and Fast. **NberLC** values and **NberSE** values indicate respectively the number of logic cells and the number of the shareable expanders. For comparison we used the ATF15xx device from Atmel [16] and the MAX7000 device from Altera [17]. The two devices have the same logical capacity. Then, a logic cell (**LC**) in a MAX7000 is basically equivalent to a macrocell (**MC**) in a ATF15xx (Fig 2).

Bench	ATMEL synthesis		ALTERA					
			Normal synthesis			Fast synthesis		
	NberMC	**NberFB**	**NberLC**	**NberSE**	**Gain(%)**	**NberLC**	**NberSE**	**Gain(%)**
5xp1	12	9	16	9	33	15	7	25
alu4	81	30	141	29	74	149	26	84
apex3	142	34	154	43	8	168	41	18
apex4	207	36	248	23	20	250	21	21
clip	16	26	22	3	37	19	10	19
cps	163	44	175	88	7	157	62	- 3
duke2	45	25	52	28	16	51	10	13
ex5	64	18	63	23	-1,5	67	8	4
misex3	96	30	225	77	134	220	54	129
rd84	26	17	32	06	23	32	18	23
seq	173	36	266	122	54	237	67	37
t481	7	6	6	6	-14	17	5	143
table3	104	28	139	102	34	127	25	22
table5	90	23	125	77	39	118	34	31
vg2	10	13	16	4	60	16	0	60
Total avrage Gain					**34%**			**39%**

Table 1: Comparison with Altera tool

On average, when the circuits in Table1 are fitted using the Altera tool, they require 34% to 39% more logic cells than the Atmel fitters targeted to the ATF15xx.

VI. Conclusion

The overall design flow for a CPLD is organized around processing for innovative features. The Xor synthesis based on algebraic matching and foldback processing creates initial foldback clusters. Once the clusters are defined, the clustering step explores a bipartite matching method of cluster assignment to PALs. This flow has been fully implemented and validated on the ATMEL flow.

In conclusion, it has been shown during the last two decades that successful topic synthesis is a combination of fundamental successful achievements (ROBDD ...) and device-specific features requiring permanently innovative heuristics.

VII. References

[1] J. H. Anderson, S. D. Brown, “Technology Mapping for Large Complex PLDs, ” in Proc. 35 th ACM/IEEE Design Automation Conference 1998, pp 698-703.

[2] D. Chen, J. Cong, M. Ercegovac and Z. Huang, “Performance-Driven Mapping for CPLD Architectures, ” IEEE Trans. on Computer-Aided Design, oct. 2003,Vol. 22, No. 10, pp. 1424-1431.

[3] J. Cong, H. Huang, and X. Yuan, “Technology mapping for k/m-macrocell based FPGA’s,” in Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays, San Jose, CA, Feb 2000, pp. 51-59.

[4] S.L. Chen, T.T Hwang and C.L Liu “A Technology Mapping Algorithm for CPLD Arcitectures, ” in Proc. IEEE. Int. Conf. on Fiel-Programmable Technology, Hong Kong, Dec. 2002, pp. 204-210.

[5] J. Cong and Y. Ding, “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs, ” IEEE Trans. On Computer-Aided Design, Jan. 1994, Vol. 13, No. 1, pp. 1-12.

[6] K. C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar, “DAG-Map: Graph-based FPGA technology mapping for delay optimization,” IEEE Design Test Comput., pp. 7–20, Sept.1992.

[7] H.Eisenmann and F.Johannes, “Generic Global Placement and Floorplanning,” in Proc.35 th ACM/IEEE Design Automation Conference 1998.

[8] S.A Senouci, A. Amoura, H. Krupnova and G. Saucier, “Timing-Driven Floorplanning on Programmable Hierarchical Targets,” in Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays, San Jose, CA, 1998, pp. 85-92.

[9] R. Murgai, B.K. Brayton, A.S Vincentelli, “An improved synthesis algorithm for multiplexor-based PGA’s,” in Proc.29th ACM/IEEE DAC 1992, pp 380-386.
[10] H. Alt, N. Blum, K. Mehlhorn, M. Paul, “Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}(m \log n)^{0.5})$,” Information Processing Letters, Vol. 37, No. 4, 237-240, 1991
[11] N. Sherwani, Algorithms for VLSI Physical Design Automation, Third edition. 1999 by Kluwer Academic Publishers.
[12] D. Kania, “A technology mapping algorithm for PAL-based devices using multi-output function graphs,” in Proc. 26th Euromicro Conf., Sept. 2000, pp. 146–153.
[13] V. Solovjev and M. Chyzy, “The Universal Algorithm for Fitting Targeted to Complex Programmable Logic Devices,” in Proc. 25th Euromicro Conf., Sept. 1999, pp. 286–289.
[14] S. Krishnamoorthy and R. Tessier , “Technology Mapping Algorithms for Hybrid FPGAS Containing Lookup Tables and PLAs, ”IEEE Trans. on Computer-Aided Design, may. 2003,Vol. 22, No. 5, pp. 545-559.
[15] V. Ciriani, “Logic Minimization using Exclusive OR Gates ,” in Proc 38st ACM/IEEE Design Automation Conference 2001.
[16] The ATF15xx *Family Data Sheet*, Atmel Corporation, 2000.
[17] MAX 7000B Programmable Logic Device Family, the Altera Data Book, Altera Corporation, 2000.