1

Boolean Optimization Using Implicit Techniques

F. Poirot, G. Tarroux and R. Roane
Compass Design Automation
505, Route des Lucioles - 06560 Sophia-Antipolis, France.
tel: (+33) 92.94.46.21 - fax: (+33) 93.65.39.21
e-mail: franck@compass.fr

Abstract

The increasing complexity of industrial designs, and especially with the extensive use of high level languages such as VHDL or Verilog, progressively obsoletes classical Boolean optimization techniques. Thus, the application of Binary Decision Disgrams in Logic Synthesis becomes an attractive alternative to push forward the limit of design complexity. In this paper, we have developed new Optimization techniques only based on implicit techniques, and all classical steps were fully reformulated to provide a powerful system providing better results than other existing techniques.

Keywords

Logic Synthesis, Binary Decision Diagrams, Boolean Optimization, Implicit Techniques.

1. INTRODUCTION

The generalization of hardware description languages such as VHDL or Verilog and improvements in technologies allow the designer to consider the possibility of defining bigger designs. Also, to make these descriptions more readable and easily debuggable or back-annoted, designers generally introduce local variables or signals to express their logic functions. These descriptions yield to huge redundant and unoptimized Boolean networks which will be under optimized or impractical to process with classical optimization algorithms.

To push forward these limitations, Binary Decision Diagrams have been introduced. BDDs are dense representations of Boolean function based on Shannon decomposition (Bryant 86). If a fixed input order for the successive Shannon decompositions is imposed, they become a canonical representation (ROBDDs) widely used for verification purpose (Madre 90, Touati 91, Jeo 91), testing (Akers 78), or direct technology mapping (Besson 92, Murgai 92).

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*
© IFIP International Federation for Information Processing 1995

In this paper, after some definitions, we are preliminary concerned with the description of our implicit Boolean optimization system. This system completely redefines conventional methods and pushes forward the complexity of designs to allow better optimizations. Efficient and novel ideas are presented using dynamic manipulation of implicit structures during the optimization process to allow the consideration of powerful techniques such as Boolean division with a constant control of the memory usage. These techniques have been evaluated on standard MCNC benchmarks and show a strong improvement compared to other published methods.

2. DEFINITIONS

2.1. Binary Decision Diagrams

Given a Boolean function $F(x_1, x_2, ..., x_n)$ the positive and negative cofactors of f with respect to x are defined by $F_x = F(1, x_2, ..., x_n)$ and $F_{\bar{x}} = F(0, x_2, ..., x_n)$. The Shannon expansion with respect to variable x is given by $F = x.F_x + \bar{x}.F_{\bar{x}}$. A Binary Decision Diagram is a Direct Acyclic Graph (DAG) representing a Boolean function. The construction of a BDD is based on the successive Shannon decompositions of the function according to a splitting variable as shown in figure 1. Each non terminal node n rooted by a variable x, corresponds to a function F(n) and is the origin of two arcs $F(n)_0$ and $F(n)_1$. The graph rooted at $F(n)_0$ represents the negative cofactor $F(n)_{\bar{x}}$ and is called the *Else Edge* of n. The graph rooted at $F(n)_1$ represents the positive cofactor $F(n)_x$ and is called the *Then* Edge of n.

For an Ordered BDD (OBDD) a global ordering "<" over the set of variables is imposed. On any path from a root to a terminal node the variables occur in the given order. The graph may have redundant vertices and duplicated subgraphs. They are eliminated by repeatedly applying transformations rules (Bryant 86). The BDD shown in figure 1 is in fact an OBDD with the order a<b<c<d.

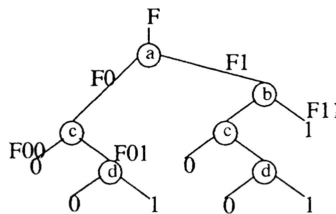

Figure 1: BDD for F=ab+cd

The maximally reduced graph is referred as Reduced Ordered BDD (ROBDD). For a given Boolean function and a given ordering, this graph is unique and therefore constitutes a canonical representation. This property presents crucial advantages for functional equivalence testing. Figure 2 shows the ROBDD of F= a **xor** b **xor** c.

ROBDD with negative edge is a more compact representation (Madre 88). In this representation, not only common subtrees corresponding to the classical notion of subfunctions (SF) are identified, but the complemented subtrees ($\overline{SF}$) are also detected. The subtree and its complement will be represented only once and a negative edge can be considered as an inverter. Figure 3 shows the ROBDD with negative edges of F= a **xor** b **xor** c. In this paper the term BDD denotes ROBDD with negative edges.

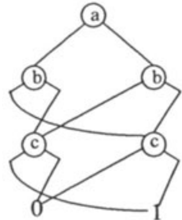

Figure 2: ROBDD of F=a xor b xor c

Figure 3: ROBDD with negative edges of F

2.2 Dynamic Improvement of the Ordering

After each garbage collection, before starting operations, the variable ordering is dynamically improved with the algorithm proposed in (Rudell 93). This step reduces the current node count, and makes the future operations faster to perform. It also allows to build BDDs which could not be built with any fixed ordering, as any given ordering fails during the construction. This swapping is performed on adjacent variables v_i and v_{i+1}. This allows to only change the part of the DAG containing v_i and v_{i+1}, instead of re-building the whole graph. Figure 8 gives the general case of the method when F_0 and F_1 are printed more than once. Note that new built nodes have to be made canonical and be looked-up in the node hash table before physically creating them. The interesting property of an adjacent swapping is that the top and bottom parts of the DAG, respectively built from v_1 to v_{i-1} and from v_{i+2} to v_n are unchanged before and after the transformation. Indeed, referring to figure 4, considering F, F_0, F_1 as inputs, top part of the DAG denotes the same BDDs with the same input ordering, before and after the swapping. Thus, the canonicity property of BDDs involves that the DAG obtained after swapping v_i and v_{i+1} is homomorphic to the initial one. Now considering F_{ij}, $(i,j) \in \{0,1\}^2$, as outputs, the bottom part of the DAG denotes the same BDDs with the same ordering before and after swapping.

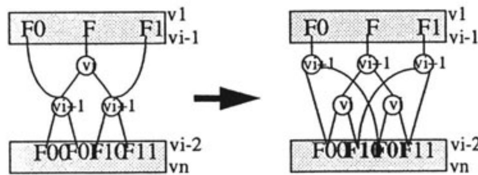

Figure 4: Adjacent swapping

2.3. Cube Set Manipulation and Implicit minimization

Classical approaches are based on Sum-of-Products (SOPs) structures which quickly appears to be very greedy in memory usage and unsuitable for logic operations like inverting or equivalence testing. So, in conjunction with BDDs which lead to a more efficient representation of Boolean functions than logic trees, an efficient cube set representation was also necessary to perform the minimization process. Madre and Coudert (Coudert 92) have proposed a BDD based meta-product structure, which requires two variables (o_i, s_i) to denote the occurrence and

the sign of each input x_i. In a such BDD, each 1-path starting from the root node denotes a cube. The paths rooted by the Then-Edge of o_i contains an occurrence of x_i. Then, s_i expressed if x_i occurs in its direct (Then-Edge) or complemented form (Else-Edge). This structure has the advantage to be compact and canonical for prime sets, and its size is not related to the number of cubes it denotes. In (Coudert 92) a breakthrough method was given to compute primes and essential primes of so large functions that they have never been handle before, and (Coudert 93b) provided a new logic minimization algorithm resolving cyclic core of all the hard Espresso problem. Nevertheless, for sparse networks like some ISCAS benchmarks the meta-products structure could blow up, even if the BDDs of these networks can be built. Minato proposed in (Minato 93a) a new cube set structure called Zero-Suppressed BDDs, or ZBDD (for Cube Set), based on new reduction rules on BDDs. This structure is definitively more suitable for cube set manipulation than meta-products, especially for sparse networks. In this structure, one variable is used to denote in which cube an input x_i appears, and an other variable denote in which cube $\overline{x_i}$ appears. The major improvement consists on a new reduction rule which eliminates from the graph the variables which does not appear in the cubes to whose path they belong, namely the variables whose Then-Edge points to the "0" terminal node. Figure 7 gives the algorithm described with meta-products in (Coudert 93a), adapted here with ZBDDs using the principle of Morreale's algorithm (Morreale 70).

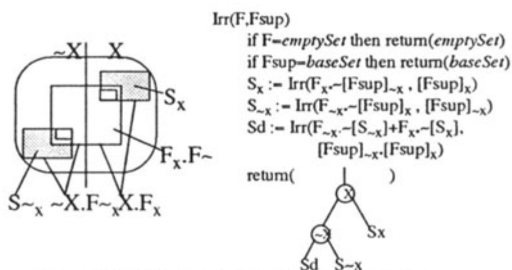

Figure 7: Irredundant Prime Cover Generation

3. BOOLEAN NETWORK DECOMPOSITION

3.1. State of the Art

The decomposition of the Boolean network is of prime importance in logic synthesis as it directly impacts on the quality of the synthesized circuit. Even if the quality criteria is measured in terms of area, speed, power or testability, it has been proven that the literal count is a good predictor. Algebraic methods to optimize combinational circuitry have been already proposed in the early 1950's. These methods were only suitable for small networks not exceeding 10 variables. They were nicely enhanced in the 1980's by more sophisticated techniques (Brayton 84).

Nevertheless, all these research studies were mainly driven by initial works targeted on PLA implementation. Clearly, the sum-of-products representation was very attractive to deal with Boolean networks as the programming data structure was compact, and the algorithms may use the efficiency of classical programming languages (mask on operators, and, or, xor between bytes representing the covers). Therefore, most designs of that period were handled in an efficient way as long as PLAs was an interesting issue, and the design complexity relatively small.

But the introduction and the generalization of hardware languages such as Verilog, VHDL, and the improvements in the technologies, significantly increases the size of Boolean networks. Then, industrial applications became more and more impractical to process with standard SOPs approaches. They usually consume huge memory and take large CPU time which is not compatible with the time to market criteria becoming a key of success for CAD companies. Of course, numerous heuristics solve these weaknesses but they seriously impact on the quality of the final circuitry.

The introduction of implicit techniques was a complete breakthrough in this field and creates new motivations to reconsider conventional techniques with these powerful structures. They represent the next generation in synthesis. Minimization was already successfully addressed by (Coudert 92) and (Minato 92), but the decomposition still remained a bottleneck in the flow.

Minato (Minato 93b), introduced algebraic techniques but they were very limited. In fact, Brayton's kernel extraction (Brayton 87) method was rewritten with implicit techniques and was based on the most occurring literals. Also, only algebraic division was applied, and the algorithm only considered one single function avoiding the sharing of identical logic within several functions.

Recently, Stanion (Stanion 94) proposed a more sophisticated method by producing a more complete set of divisors. Unfortunately, the weighting of these candidates was based on BDD's 1 path which is neither prime nor irredundant. So, they could not correctly link the candidate extraction to their final goal. Moreover, they provided a more sophisticated division than Minato with Boolean consideration by using cofactor techniques, but these methods could not provide the same quality than a fully Boolean division. Finally, as in the previous method, they only deal with single functions.

The method that we propose tries to cover the previous weaknesses by taking advantage of the efficiency of our new BDD and ZBDD package presented before. In fact, our goal is to develop implicit techniques allowing to consider a complete Boolean network as input of our BDD optimizer. Then, it may run sophisticated decomposition techniques (divisor extraction, exact weighting, Boolean division) on industrial networks with fast execution time.

3.2. Decomposition flow

A typical decomposition flow is represented in the figure 8. The first phase consists of computing a set of candidates which can be used to divide one or more functions in the Boolean network and obviously, if no candidate is found the decomposition is ended. Afterwards, on or more candidates have to be selected as divisor and it is important to define an accurate selecting criteria which correctly represents the goal of the optimization. In the technology independent optimization the more suitable criteria is the number of literals in a Boolean function. Then, for each candidate we compute a weight which represents exactly the number of literals saved in the Boolean network when this candidate is used as divisor. Moreover, in order to have a more powerful decomposition, we develop and use in our flow a method to determine the best compatible set of candidates. This allows to divide the Boolean network by several candidates in the same iteration instead of using only one. To divide the Boolean network we have develop both Boolean and algebraic divisions based on ZBDD structure. The power of these two new divisions allows us to run both of them for each candidates and keep the best result without a large cpu time penalty. All algorithm used in our flow are described more precisely in the following paragraphs.

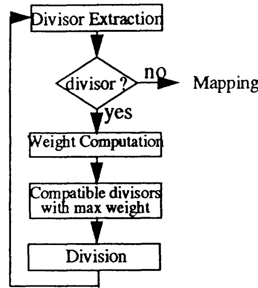

Figure 8: Decomposition flow

3.3. Divisor extraction

The most popular method for the identification of common sub-expressions in the Boolean network has been proposed by Brayton and Mcmullen at ISCAS'82. Here again, the problem is similar to the creation of optimal code for a compiler and the SOPs appear a very suitable representation for a limited complexity. In this paper, the notion of kernels of a logic expression was introduced to provide an efficient means of finding common sub-expressions. A kernel K of a function F is defined as the cube free quotient of F by a cube C.

In our flow, we have developed a new algorithm taking full advantage of the implicit structure to get all level-0 kernels of a function. One main advantage of this procedure versus classical SOP ones is that during the examination of the implicit structure graph only effective inputs are considered while SOPs require the evaluation of all inputs which allows to generate the set of level-0 kernel in a little cpu time. The pseudo code of the level-0 kernel extraction procedure is given in figure 9.

```
kernel0(zbdd, lit)
{    if nbLiterals(zbdd) < 2 then return
     while (lit:=nextInput(zbdd,lit))
     {    if not literalIsSeveralTimes(zbdd, lit) then continue;
          zbddVar := getZBDDVar(lit);
          zbdd1 := subSetWhenVarIs1 (zbdd, zbddVar);
          cubeFactor := largestCubeFactor(zbdd1, zbddVar);
          if cubeFactor=emptySet then continue;
          if cubeFactor<>baseSet
          then zbdd1 := divideAlgZBDD(zbdd1, cubeFactor);
          updateCandidatList(zbdd1);
          kernel0(zbdd1, lit);
     }
}
```

Figure 9: kernel 0 extraction procedure

Using only level-0 kernel candidates is not enough to have an efficient decomposition because some other candidates can be very good divisors like intersecting level-0 kernels or common cubes. Intersecting level-0 kernels are computed from the level-0 kernel set generated with the procedure previously described. For instance, if the two following level-0 kernels k1=a+b+c and k2=a+b+d are found, the *intersectKernel* procedure described below will find the

candidate a+b which will be put in the candidate list. In the same way, if in the Boolean network there are two functions f and g such as f=a.b.c.d + e.f.g and g=a.b.c.e + f.g.h, we can easily see that the cubes a.b.c and f.g are shared by f and g. The procedure *commonCube* extracting common cubes in the Boolean network is described below.

Therefore, our candidate computation phase based on ZBDD generates a list of candidates containing level-0 kernels, intersecting kernels and common cubes which allows us to chose the best candidate among a large set of candidates.

```
kernelIntersect()
{   for i:=1 to (nbKernel0-1)
    {   for j:=(i+1) to nbKernel0
        {   newKernel := intersection(kerneli, kernelj);
            if newKernel<>emptySet
            then updateIntersectList(newKernel);
        }
    }
}
```

Figure 10: kernel intersecting procedure

3.4. Divisor selection

After getting the set of candidates for decomposition, we compute the weight of each of them. The candidate's weight is the number of literals saved in the Boolean network when the network is divided by this candidate. During the weight computing phase, we divide all functions in the Boolean network by the candidate and its complement by using both algebraic and Boolean division. Obviously, the weight of a candidate is the sum of the best weights got for each divided function..

```
commonCube(f, g, cube, lit)
{   if (f=baseSet OR g=baseSet) AND nbLit(cube)>1
    then updateCubeList(cube);
    while (lit:=nextInput(f, lit))
    {   zbddLit := getZBDDVar(lit);
        if nbLit(g)=1 then
            if g=zbddLit
            then zbddVarG:=baseSet
            else zbddVarG:=emptySet
        else zbddVarG:=subSetWhenVarIs1(g,zbddLit);
        if zbddVarG=emptySet then continue;
        if nbLit(f)=1 then
            if f=zbddLit
            then zbddVarF:=baseSet
            else zbddVarF:=emptySet
        else zbddVarF:=subSetWhenVarIs1(F,zbddLit);
        commonCube(zbddVarF, zbddVarG,
                   zbddMultiply(cube,zbddLit), lit);
    }
}
commonCubesExtract()
{   for all funci and funcj
        commonCube(funci,funcj, baseSet, 0);
}
```

Figure 11: common cube extraction procedure

Afterwards, we have to select the best candidate and divide the Boolean network by it. In fact, in order to have a powerful decomposition in terms of quality and cpu time, we select a set

of candidates instead of using only one candidate at each iteration (figure 8). To select a set of candidates by keeping the right knowledge on the literal saving, we introduce the term of *compatibility* between candidates. Then, we say that 2 candidates c1 and c2 are compatible if and only if the entire Boolean network can be divided in the same way by {c1, c2} and {c2,c1}. In other work, c1 and c2 are compatible if the parts of Boolean functions affected are disjoint which means that the weight of c2 (c1) computed before and after dividing the Boolean network by c1 (c2) is the same. Then, after computing all compatibilities we build a compatibility graph and use an algorithm to find the best weighted clique which represents a set of compatible candidates.

3.5. Boolean division

The Boolean division problem is a complex problem to solve because it requires the computation of Boolean operations which may have a high complexity under some data structures. It is mainly because classical methods such MIS (Brayton 87) solve this problem by expressing the divisor as the *don't care* of its expression by its variable. Moreover, this method has the disadvantage to compute the complement of a divisor which is of an exponential complexity with sum of product structures instead of being immediate with BDDs. Therefore, methods using SOP should include a bunch of heuristics to handle large designs which badly impact the quality of the results. Additionally, most industrial tools make only use of algebraic techniques since they consider Boolean ones too costly. Even with the BDD structure this problem is not easy to solve since Boolean division creates a large number of nodes in the BDD structure which may easily blows up the system if no memory control is done.

In our method, we consider the Boolean division with a novel approach (figure 12). This method uses the principle of the implicit minimizer and computes the best bounds of the remainder and the dividend. First the smallest remainder is computed with smallest lower bound (Rmin) and largest upper bounds (Rmax) to increase the Boolean space. Afterwards, bounds for the dividend are computed by taking into account the result of the remainder. The figure 12 shows the principle of our Boolean division and the figure 13 shows an example of division.

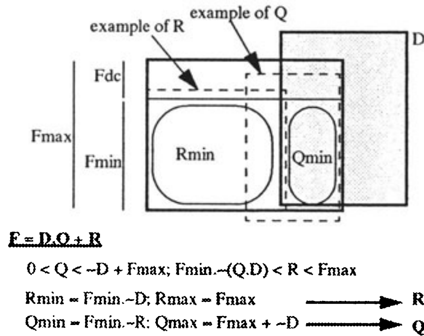

Figure 12: Boolean Decomposition

Note that during execution of these operations, dynamic ordering and garbage collector control the memory used in order to avoid over size due to the creation of internal BDD nodes.

F = a + b.c
D = a + c

Rmin = Fmin.~D	= (a + b.c).~a.~c = 0	R = 0
Rmax = Fmax	= a + b.c	
Qmin = Qmin.~R	= a + b.c	Q = a + b
Qmax = Fmax + ~D	= a + b.c + ~a.~c = a + b + ~c	

F = (a + c).(a + b) + 0

Figure 13: Example

4. UNIFIED APPROACH WITH VERIFICATION

Since time to market is of prime interest for EDA companies, it is mandatory to detect as soon as possible any errors in the final implementation. For that purpose, formal verification techniques has been widely developed and adopted (Madre 90, Touati 91).

These errors can be either introduced by the system itself or by the designer who may have modified the schematic to respect a specific constraint. So, both aspect should be considered to provide an convenient system which guarantees the correctness of the design without too much CPU overhead.

The novelty of our system is to use the same data structure with its canonicity property to optimize the network and verify if the final implementation is correct regarding the initial functional description. So, there is no overhead in the computation time.

In case of errors, a report clearly tells which signal is incorrect and the designer can easily backannotate the functional description. Also, our system can be used as a prover. In that case, the designer specifies properties (for instance a traffic light is not green and red at the same time). These properties are converted in our data representation, and compared with in the specification. In that case, it provides the designer a powerful way of checking that his HDL description respects his specification.

5. EXPERIMENTAL RESULTS

This new Boolean optimization algorithm was implemented as part of the ASIC Synthesizer of Compass Design Automation. This program was written on Mainsail (MAINSAIL) on SPARC 1030. It was tested on standard MCNC benchmarks (MCNC 89) and compared with the factoring results of MIS (Brayton 87) from Berkeley, and CATAMOUNT from the University of Washington (Stanion 94). The first one uses classical sum of product representations, and the latest the new Boolean techniques applied on BDDs as previously described. As we are preliminary concerned by the number of literals, we use this criteria to compare all methods. It has also the interest of being independent of the mapping process and the target library, and so fair comparisons can be drawn.

Results are shown in table 1. The first column gives the name of the benchmarks provided by MCNC. The 2 others give information on the I/O connection numbers. The MIS column provides results given in (Brayton 87), and the CAT (for CATAMOUNT) were given in (Stanion 94). COM gives the results for Compass. The ratio for MIS and CAT is computed as the difference of literals with COM divided by MIS and CAT respectively. It shows an average improvement of 16.7% in terms of literals compared with MIS, and 10.5% compared with CAT. Run

times to proceed these benchmarks are in a range of 10% compared to Catamount which is considered as similar for a user point of view.

Benchmark	MIS	CAT	COM	COM vs MIS (%)	COM vs CAT (%)
rd53	70	68	60	-14.3	-11.8
misex1	86	88	75	-12.8	-14.8
misex2	164	163	117	-28.7	-28.2
f51m	168	174	167	-0.6	-4.0
5xp1	164	159	148	-9.8	-6.9
z4ml	69	67	56	-18.8	-16.4
sao2	183	174	201	9.8	15.5
9sym	258	116	139	-46.1	19.8
vg2	246	254	133	-45.9	-47.6
rd73	192	173	178	-7.3	2.9
bw	299	306	223	-25.4	-27.1
9symml	277	278	127	-54.2	-54.3
alupla	205	182	176	-14.1	-3.3
duke2	772	[illegible]	528	-31.6	-28.3
misex3	1053	1010	1342	27.4	32.9
rd84	381	319	149	-60.9	-53.3
Sum average	4587	4267	3819	-16.7	-10.5

Table 1: Results

We have also evaluated this method on industrial designs, which was in fact the main evaluation criteria of this approach for a CAD company. These designs have complexity ranging between 1,000 to 50,000 gates which cover in fact the standard design complexity that actual Logic Synthesis tools should handle. It appears that 10% of these designs could not be handle with standard SOP techniques when BDD completed in few minutes. Others have been drastically speed-up (up to 100 times), and the results were ranging from -6% upto 17%. In addition, this technique allows to easily compare the complexity of the function towards its complemented form, and thus keep the better one, still allowing further improvements.

6. CONCLUSION AND FUTURE WORK

We have proposed new multilevel synthesis techniques based on Binary Decision Diagrams and implicit cube manipulation. These techniques have been successfully applied on industrial designs due to the efficiency of our complete dynamic approach to build and apply our BDD core algorithm. This BDD core always controls the memory usage to prevent any explosion and also to speed up the complete process which is a novel and powerful approach allowing to synthesize and optimize larger designs. We have also discussed how this BDD core is linked to each synthesis process.

Both cpu time and memory usage can still be improved by using several BDDs with different input orderings instead of using only one. Then, in the future we will improve our candidate extraction and division algorithms in order to take advantages of partioning approaches. Even the very encouraging results we got with our BDD optimizer we think that we can still greatly improve it.

Next development will focus on the extraction of Boolean divisors directly from the topology of the BDD. Also, even if literals are good estimator, they cannot reflect well the complexity of all library cells provided by CAD vendors. For instance, the integration of Low Power

issues privilege the development of new AOIs (and or inverter) cells for which the complexity is not related to the number of literals. So, we plan to interface our Optimizer with a Boolean Matching System to allow a more precise weighting and selection of candidate divisors. We believe that this development would considerably improve the quality of the designs by considering earlier the design goals (area, speed, or power).

7. REFERENCES

Akers (1978), "Functional Testing with Binary Decision Diagrams", *Proc. of the IEEE Conf. on Fault-Tolerant Comput., pp 75-82.*

Besson, Bouzouzou, Crastes, Floricica and Saucier (1992), "Synthesis on Multiplexer-based F.P.G.A. using Binary Decision Diagrams", *Proc. of ICCD 92.*

Brayton and al. (1984), "Logic Minimization Algorithms for VLSI Synthesis", *Kluwer Academic Publishers*, Boston.

Brayton and al. (Nov. 1987), "MIS: a Multi-Level LogicOptimization System", *IEEE Transaction on CAD*, pp.1062-1081.

Bryant (1986), "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, Vol C-35, no.8, pp. 677-691.

Coudert and Madre (1992), Implicit and Incremental Computation of Prime and Essential Prime Implicants of Boolean Functions", *29th ACM/IEEE Design Automation Conference.*

Coudert and Madre(1993a), "Toward a Symbolic Logic Minimization Algorithm", *Proc. VLSI Design'93*, Bombay, India.

Coudert, Madre and Fraisse (1993b), "A New Viewpoint on Two-Level Minimization", *30th ACM/IEEE Design Automation* Conference.

jeong et al (1991)., "Variable Ordering For FSM Traversal", *Proc. of the International Workshop on Logic Synthesis*, MCNC.

Madre and Billon (1988), "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour", *25th ACM/IEEE Design Automation Conference* , pp. 205-210.

Madre (1990), "PRIAM : un outil de verification formelle de circuits integres digitaux", *These de l'ENS des* Telecommunications, Paris.

Murgai, Brayton, Sangiovanni-Vincentelli (1992), "An Improved Synthesis Algorithm for Multiplexor-based PGAs", *29th ACM/IEEE Design Automation Conference* , Anaheim, CA, June 1992, pp. 380-386.

MCNC (1989), Introduction to Synthesis Benchmarks, *second International Workshop on Logic Synthesis*, North Carolina USA, May 1989

MAINSAIL Language manual, XIDAK inc.

Minato (1992), "Fast Generation of Irredundant Sum-of-Products Forms from Binary Decision Diagrams", *Proc of SASIMI '92*, pp. 64-73.

Minato (1993a), "Zero-Suppressed BDDs for Set Manipulation in Combinational Problems", *30th ACM/IEEE Design Automation* Conference.

Minato (1993b), "Fast weak division method for implicit cube representation", *proceeding SASIMI"93.*

Morreale (1970), "Recursive Operators for Prime Implicant and Irredundant Normal Form Determination", *IEEE Trans. on Computers,* 1970.

Murgai, Brayton , and Sangiovanni-Vincentelli (1992), "An Improved Synthesis Algorithm for Multiplexor-based PGAs", *29th* ACM/IEEE Design Automation Conference, Anaheim, CA, pp. 380-386.

Rudell (1993), "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", *Proc. of International Workshop in Logic Synthesis,* CA.

Stanion and Sechen (1994), "Boolean division and factorization using BDDs", *IEEE Transactions on CAD* vol 13, N° 9, sept. 94, pp. 1179-1184.

Touati, Brayton and Kurshan (1991), "Testing Language Containment for w-automata using BDDs", *Formal Methods in VLSI* Design, Miami, ACM, New-York.

8. BIOGRAPHY

Franck Poirot has got his PhD thesis in 1990 from the INPG in France under the responsability of Prof. Saucier. Then, he joined VLSI Technology as Software Developer and Compass Design Automation as Logic Synthesis project leader. Afterwards, he was appointed Program Manager to follow up the technical management of European projects. He has published more than 20 papers in high rated conferences such as DAC, ICCAD, EDAC or Euro-ASIC, and held two patents issued by the Patent and Trade-Mark Office in USA, and two others are still in the review process.
His main interests focus in several CAD aspects such as Logic Synthesis, Test, Formal Verification, Simulation, and Low Power designs.

Gerard Tarroux received the Ph.D. degree in micro-electronic from the University of Montpellier (France) in 1991. Then, he joined Compass Design Automation as Software Developer in the Logic Synthesis team. His reasearch interests include many aspects of CAD for VLSI with special emphasis on logic synthesis, FSM synthesis, formal verification, testing and low power designs.