# Boolean Mapping based on Testing Techniques†

*Ricardo Ferreira ‡, Anne-Marie Trullemans*
*UCL- Laboratoire de Microélectronique*
*Place du Levant, 3, B-1348 Louvain-la-Neuve, Belgium,*
*Phone: (+32) 10 47 25 65 - Fax: (+32) 10 47 25 98*
*e-mail : cacau@dice.ucl.ac.be, atrullemans@dice.ucl.ac.be*

*Ricardo Jacobi*
*Instituto de Informática, UFRGS, Brazil*
*e-mail: jacobi@inf.ufrgs.br*

† *Supported by the EC project KIT #144 LLSynth.* ‡ *Supported by Capes and UFV-Brazil*

**Abstract**

This paper presents a novel mapping technique, which uses a Boolean matching algorithm based on testing techniques. The matching is detected by checking the controllability and observability of signals in the cell structure against the subject functions of the network. The method was implemented inside the Sis (Touati 1990) synthesis environment. The comparison with the Sis structural mapping shows that the Boolean mapping achieves better results in similar or smaller computing time.

**Keywords**

Technology mapping, Boolean matching, Observability, Covering

## 1 INTRODUCTION

Technology mapping is the final and strategic step of the logic synthesis. In multilevel implementations, the mapping starts with a logic optimized network, modeled as a direct acyclic graph (Dag), where each node represents a logic function. The mapping step translates this logic system into a network of logic gates (the netlist), while minimizing the cost: silicon area, circuit delay and/or power dissipation. It depends on two interrelated processes: gate matching and network covering. The gate matching checks which gates of the library may implement each node function. The covering selects a list of gates to implement the whole system with a minimum cost.

Structural matching and Boolean matching are the two major approaches used to check a library cell. Structural matching verifies the equivalence between the cell and the node function structures. Both cell and node functions

VLSI: Integrated Systems on Silicon R. Reis & L. Claesen (Eds.)

must be cast to a common representation, usually a simple netlist of 2 input gates (Nor2 or And2). The cell matching then reduces to check for graph isomorphism. But one library cell may have several equivalent decompositions, and the matching has to consider all possible alternatives. To simplify the check, library cells are generally represented by trees, which excludes cells with internal fanout. Moreover, structural representation considers only completely specified functions.

A solution is to use Boolean matching, based on logic equivalence between Boolean functions. If $f(x_1, \ldots, x_n)$ is the function of a node in the Boolean network, and $g(y_1, \ldots, y_n)$ the function of a cell in the library, Boolean matching reduces to find an input variable permutation $\pi$ and an input phase assignment $\phi$ of $x$ for which cell $g$ produces function $f$ or its complement: $f(\mathbf{x}) = g(\pi\phi(\mathbf{x}))$ or $f(\mathbf{x}) = \overline{g}(\pi\phi(\mathbf{x}))$ ?

In practice, the function $f$ may be incompletely specified, but the function $g$ (the library cell) is always completely specified. The total number of variants to be considered is: $(n! \cdot 2^n \cdot 2)$. This number becomes quickly intractable for exhaustive checking.

Several heuristic approaches have been proposed for the covering: rule based methods, algorithmic methods and stochastic methods. Rule based systems like (Darringer *et al.* 1984), reduce the complexity by local transformations, in order to cover the network. But the method is too library dependent and allows only local optimizations. The algorithmic methods (Keutzer 1987, Savoj 1992, Mailhot *et al.* 1993) produce usually better results because they rely on global optimizations. Stochastic methods are global techniques based on space navigation, such as genetic algorithms. The covering adopted in Sis and in our approach are algorithmic based techniques.

In this paper we present a Boolean mapping algorithm based on the Boolean matching proposed by (Trullemans *et al.* 1996a). This matching derives from testing techniques that analyze a structural equivalent of the cell in terms of observability and controllability functions, and compare it to a binary decision diagram (BDD) representation of the node functions. An improvement of this matching technique is implemented in the Sis environment. We compare this new matching technique to the Sis approach, and show that it may deal efficiently with tree representations as well as non-tree cells such as Mux and Xor.

## 2 DEFINITIONS

Let $x_1, x_2, \ldots, x_n$ be the variables of the space $B^n$, where $B = \{0, 1\}$. We use $\mathbf{x}$ to represent a vertex or a vector of variables in $B^n$. Let $f : B^n \rightarrow B$ be a Boolean function, and $x_i$ an input variable of $f$. The cofactor of $f(x_1, \ldots, x_i, \ldots, x_n)$ with respect to variable $x_i$ is: $f_{x_i=1} = f(x_1, \ldots, 1, \ldots, x_n)$. The cofactor of $f(x_1, \ldots, x_i, \ldots, x_n)$ with respect to variable $\overline{x}_i$ is: $f_{x_i=0} = f(x_1, \ldots, 0, \ldots, x_n)$. The *Boolean difference* of a function $f$ with respect to

an input variable $x_i$, is the function $\Delta f_{x_i} = f_{x_i=1} \oplus f_{x_i=0}$, where $\oplus$ is the exclusive-or operator. This function is also called the *observation function* of the input variable $x_i$. It indicates the condition for which any change at the signal $x_i$ can be observed at the output of $f$. An input $p$ of a logic gate is said to have a *controlling value* if its value prevents other inputs from affecting the output of the gate. The controlling value for And-type gates (And, Nand) is 0, and the controlling value function is $Control(p) = \bar{p}$. The controlling value for Or-type gates (Or, Nor) is **1**, and the controlling value function is $Control(p) = p$. No controlling value exists for inverters and Xor gates.

## 3 BOOLEAN MATCHING

A recent and comprehensive survey of Boolean matching methods was presented in (Benini *et al.* 1995). For incompletely specified functions, the first algorithm for detecting a match under input variable permutation and/or input and output phase assignment was proposed by Mailhot (Mailhot *et al.* 1993). This algorithm uses compatible graphs, but the size of these graphs is exponential in the number of input variables, and their application is thus limited to functions with a maximum of four inputs. Savoj (Savoj 1992) uses the tautology check, but this doesn't solve the problem of finding the variable assignment. He introduced a class of filters that are valid even for incompletely specified functions.

In (Trullemans *et al.* 1996a) a new Boolean matching method was presented: the controlling value matching, which allows to consider only a subset of input variables, and prune the permutation tree as soon as these variables are rejected. Moreover, when a correct input variable permutation - even partial - is found, the corresponding input phase assignment can be directly deduced : a total of $2^n$ possible input phase assignments is saved. But this method may fail to recognize valid cells, when the cell structure is not a tree. We modify here the algorithm to validate it for don't cares and general Dag structures.

### 3.1 Controlling value Boolean matching

The method is based on the equivalence of observation functions for equivalent functions (observability equivalence). These observation functions are applied on a structural* equivalent of the proposed cell, and this structure is scanned from the external inputs to the output. Inside the structure, every elementary gate is checked using the controlling value paradigm (controlling value check). Along the scan inside the structure, the observation functions of the internal signal lines are to be computed: this is called *Observation Function Deduction.* If there are multiple fanouts in the circuit, no exact deduction is possible, and

*composed only with And-, Or- and inverter-type gates

only heuristics (Trullemans *et al.* 1996a, Trullemans *et al.* 1996b) could be applied.

The next theorem asserts the equivalence of two functions from their observation functions, and is the basis of the method:

**Theorem 1** *Let $f : B^n \rightarrow B$ and $g : B^n \rightarrow B$ be two Boolean functions. The function $g$ is an equivalent to the function $f$, or to its complement, if and only if $\Delta g_{x_i} = \Delta f_{x_i}$ for any input variable $x_i \in \mathbf{x}$.*

```
Match(function f, cell g,permutation Π = initial permutation)
(1) {   if ( scanline(f,g,Π) == Found ) return match_found;
(2)     return Match(f,g,Π = next permutation;);      }

scanline(function f, cell g,permutation Π)
(4) {   For each input x_i      /* input observation functions */
(5)       Δg_{Π x_i} = Δf_{x_i};
(6)     For each gate k_i in the cell g      /* scan line the cell g */
(7)     {  Let p,q the gate inputs and let r the gate output;
           /* Controlling value check for p */
(8)        If ( fanout(q) > 1 ) Δg_p = Multi_fanout_observability(q);
(9)        If ( Δg_q · Control(p) ≠ 0 )
(10)         If ( p is an input with unknown phase )
(11)           If ( Δg_q · Control(p)' ≠ 0 )      return Wrong_permutation;
(12)           Else phase_invert(p);
(13)         Else return Wrong_permutation;
(14)       Else
(15)         /* Controlling check value for q */ ...
(16)       Δg_r = Δg_p + Δg_q   }   /* Deduction */
(17)    return BDD_check(f,g,Π); }
```

**Figure 1** Algorithm for controlling value matching

The *match* algorithm (lines 1-2) is shown in figure 1. The initial permutation is derived from the BDD order. The permutation $\Pi$ assigns the input variables $\mathbf{x}$ of $f$ to the input variables $\mathbf{y}$ of $g$. For each permutation, the *scanline* algorithm is called. The library cell function $g$, represented by the circuit $C$ , is proposed as an implementation of the function $f$. To locate the design errors within the circuit $C$, its primary inputs are initialized with the observation functions derived from $f$ (lines 4-5). The only acceptable errors here are missing inverters (wrong phase assignments) or wire exchanges at primary inputs (input variable permutations). For any other detected error, the library cell $g$ is not a match for the given function $f$ and the partial permutation is pruned.

The structure of the cell $g$ is analyzed from the primary inputs to the primary output (lines 6-16), following a 'scan line' (figure 2a). The analysis consists of two basic phases: controlling value analysis (lines 8-15) and observation function deduction (line 16).

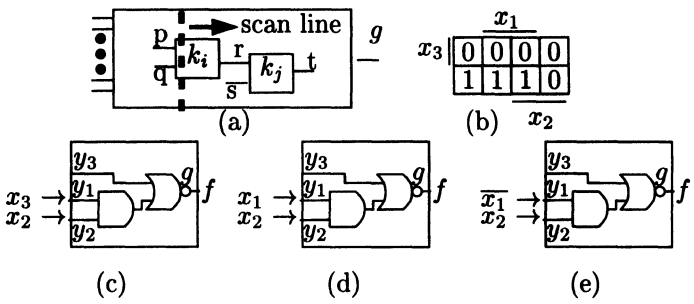


**Figure 2** (a) Scan Line (b) Function $f$ (c) Rejected Permutation (d) Accepted permutation (e) Correct Phase

## (a) Controlling value analysis

The controlling value analysis is used to check the correctness of each logic gate in this netlist. Let us consider a logic gate $k_i$ in the cell $g$ with two inputs $p$,$q$ and one output $r$ (figure 2a). We assume that the gate $k_i$ is an And-type gate or an Or-type gate, and that the observation functions of the gate inputs $p$ and $q$ with respect to the required $f$ are known. If the gate input $p$ is a primary input, its observation function is initialized with the one computed directly from $f$, otherwise it has to be deduced from the observation function of its fan-ins (see Section b).

The controlling value check is computed by $\Delta g_q \cdot Control(p)$ (*scanline* algorithm, line 9). The observation function ($\Delta g_q$) of a signal line $q$ indicates the condition for which any change at the signal line can be observed at circuit output. The controlling value function ($Control(p)$) prevents other input $q$ from affecting the gate output and the circuit output. Thus the check must be satisfied (= 0) if $p$ and $q$ are the correct inputs of gate $k_i$, and the input $p$ has a correct phase. If the controlling value is false, and $p$ is not a primary input, then the partial permutation is not correct (line 13). If $p$ is a primary input, the negative controlling value check will be computed by $\Delta q.\overline{Control(p)}$ (line 10-13). If this check is true (= 0), we need to insert an inverter, and the input phase is found . Otherwise, the current input variable permutation is not correct.

**Example 1** *Let us consider the function $f$ and library cell $g$ of figure 2b and 2c. The observation functions of $f$ are:*

$$\Delta f_{x_1} = x_2\overline{x}_3, \Delta f_{x_2} = \overline{x}_1\overline{x}_3, \Delta f_{x_3} = x_1 + \overline{x}_2$$

*Let us first check the partial permutation $x_2 \rightarrow y_2$ and $x_3 \rightarrow y_1$ (figure 2c). At gate 1, $\Delta g_{x_3} \cdot Control(x_2) = \Delta g_{x_3}\overline{x_2} \neq 0$ and $\Delta g_{x_3} \cdot \overline{Control(x_2)} \neq 0$: the input $x_3$ could be observed for any logic value of the input $x_2$. This is not possible because gate 1 is an AND gate, and this permutation must be rejected.*

*Note that the input $y_3$ of cell g has not yet been assigned: we consider only a subset of the input variables.*

*Now, we will try the partial input variable permutation $x_1 \rightarrow y_1$ and $x_2 \rightarrow y_2$ (figure 2d), which gives $\Delta g_{x_2} \cdot \overline{Control(x_1)} = \Delta g_{x_2} x_1 = 0$. The negative controlling value check for $x_1$ is validated : an inverter is to be added at input $x_1$. For $x_2$, $\Delta g_{x_1} \cdot Control(x_2) = 0$: the controlling value is validated, and the phase is correct. The phase assignment (figure 2e) is thus directly determined from this analysis: $\phi(y_1, y_2) = (1, 0)$ .*

## (b) Observation Function Deduction

To continue the scanning along the structure, we have to determine the observation functions of the internal signal lines in the netlist equivalent of the cell. These will be deduced from the initial observation functions set at the primary inputs (*scanline* algorithm, line 16).

Let us consider a tree circuit, and some particular gates $k_i$ and $k_j$ (figure 2a). If $p$ and $q$ are the correct inputs of gate $k_i$, we want to deduce the observation function of its output $r$. We propose to approximate the observation function by $\Delta g_r \approx \mathbf{\Delta g_r} = \Delta g_p + \Delta g_q$.

The next theorem shows that the controlling value check of the signal line $r$ may be conserved with this approximation, and gives the exact solution for tree circuits:

**Theorem 2** *In a tree circuit, consider a gate $k_i$, with inputs p and q, and output r. If r and s are inputs of another gate $k_j$, which output is t, we have*

$$\mathbf{\Delta g_r} \cdot Control(s) = 0 \textbf{ iff } \Delta g_r \cdot Control(s) = 0$$

## (c) Matching with *don't cares* on tree structures

Consider now a function $f$ which is incompletely specified. We have to define the incompletely specified observation functions.

**Definition 1** *Let $\Delta f_{x_i}^{on} = (f_{x_i=1} \oplus f_{x_i=0})(\overline{f_{DC}}_{x_i=1}\ \overline{f_{DC}}_{x_i=0})$ and $\Delta f_{x_i}^{dc} = f_{DC x_i=1} + f_{DC x_i=0}$ be respectively the on-set and the don't care set of the observation functions with respect to an input variable $x_i$, of an incompletely specified function $f$, with don't care set $f_{DC}$.*

Only the on-sets of these observation functions are important. The matching algorithm will take into account *don't cares* when the observation functions of $g$ are initialized by $\Delta f_{x_i}^{on}$ instead of $\Delta f_{x_i}$. For tree structures, if the controlling value check is false, there cannot be a match. Doing so, we will perhaps accept bad cells - rejected later by the BDD check - but never reject acceptable solutions.

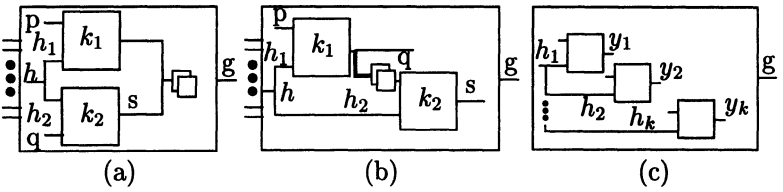


**Figure 3** Reconvergence (a) after $g_2$ (b) before $g_2$ (c) Multiple fanout

## (d) Analysis of a Multiple Internal fanout Cell

If there are multiple internal fanouts in the library cell, the observation function deduction is more complex. If we know the observation functions for all the fanout branches, we can deduce the observation function of a fanout stem (De Micheli 1994). The deduction is computed by a simple network traversal, from the output to the inputs. In the network of figures 3a and 3b, the observation function of the signal line $h$ can be computed as:

$$\Delta g_h = \Delta g_{h_2\, h=\overline{h}} \oplus \Delta g_{h_1} \tag{1}$$

In the matching process, we want to deduce the observation function of the fanout branches from that of the fanout stem, i.e. from the inputs to the outputs. Normally, no exact deduction is possible, and only heuristics could be applied. We will propose a new one, which can be proven as an approximation of the exact expression. Starting from the equation 1, we can write:

$$\begin{aligned} \Delta g_{h_1} &= \Delta g_h \oplus \Delta g_{h_2\, h=\overline{h}} \\ &= \Delta g_h \oplus (\Delta s_{h_2} \cdot \Delta g_s)_{h=\overline{h}} \\ &= \Delta g_h \overline{\Delta s_{h_2}}_{h=\overline{h}} + \Delta g_h \overline{\Delta g_s}_{h=\overline{h}} + \overline{\Delta g_h} \Delta s_{h_2\, h=\overline{h}} \Delta g_{s\, h=\overline{h}} \end{aligned} \tag{2}$$

In this expression, $\Delta g_{h_1}$ and $\Delta s_{h_2}$ are known, but not $\Delta g_s$. We propose to take only the first term as an approximation of $\Delta g_{h_1}$:

$$\Delta g_{h_1} \approx \Delta g_h \overline{\Delta s_{h_2}}_{h=\overline{h}}$$

This will always give results included in the complete solution, even if there are redundancies in the net. If there are more than two fanout branches, the approximation may be generalized, for $n$ fanout branches as:

$$\Delta g_{h_n} = \Delta g_h \bigoplus_{i=1}^{n-1} (\Delta g_{h_i\, h_{i+1}=\ldots=h_n=\overline{h}}) \quad \text{(if } n \text{ is even)}$$

$$\Delta g_{h_n} = \overline{\Delta g_h} \bigoplus_{i=1}^{n-1} (\Delta g_{h_i h_{i+1}=\ldots=h_n=\overline{h}}) \quad \text{(if } n \text{ is odd)}$$

Each term $\Delta g_{h_i}$ may be replaced by $(\Delta y_{ih_i} \cdot \Delta g_{y_i})$. Using the definition of the Xor function, the two expressions may be developed. Only the first term is kept, because the others depend on $\Delta g_{y_i}$ (where $i = 2$ to $n$), which are not known:

$$\Delta g_{h_n} \approx \Delta g_h \cdot \prod_{i=1}^{n} (\overline{\Delta y_{ih_i h_{i+1}=\ldots=h_n=\overline{h}}})$$

## 3.2 Library organization

Before the technology mapping, a setup phase is used to process gates in the library and generate particular data structures called Boolean Primitive Classes (BPC). All the gates in a BPC are equivalent to a BPC representative in the sense that the function of each gate can be obtained by inverting inputs and/or output of the BPC representative. The BPC structure reduces the number of calls to the Boolean matching algorithm.

The BPC structure contains some informations to improve the performance of the Boolean matching. These informations concern the characteristic signatures and symmetric variables that are used to reduce the total number of possible input variable permutations and the total number of possible input phase assignments, which may dramatically speedup the matching process.

# 4 IMPROVING SIS WITH BOOLEAN MATCHING

Covering and matching are two strong interrelated processes. Sis uses a tree structural check for matching, which leads to a pattern based algorithm for the covering step. The logic circuit to be mapped is first decomposed into a network of 2-input Nand (or Nor) gates and inverters. The covering algorithm splits the network into a forest of trees, which are *covered* by patterns that represent the cells in the library. In addition, Sis implements some *ad hoc* techniques that handles the particular case of cells with internal fanout, like Xor and Mux. The tree covering adopted by Sis traverses the network from the inputs to the outputs and, for each node, all the patterns in the library are checked for matching: the time complexity directly depends on the size of the library.

In our approach, the matching algorithm is a Boolean one, which can find matches that are not detected by structural methods, and may exploit the degrees of freedom provided by *don't care* conditions. Moreover, Boolean covering and matching can handle the input/output phase of the library cells. This reduces the initial subject graph by avoiding the inclusion of inverters

pairs. The base function becomes a $NPN^*$ And function, and no additional inverters are needed. Xor, Mux and other cells could also be used as base functions, by adapting the decomposition algorithm: Thakur (Thakur *et al.* 1996) adds 2-1 multiplexers, which increases the granularity of the base functions. His results show a gain in area, or a gain in delay without area increase compared to the usual Huffman based technology decomposition implemented in Sis.

## 5 BOOLEAN COVERING

In the Boolean mapping that we use, the nodes are grouped in *clusters* from the outputs to the inputs. A *cluster* is a connected sub-graph of the subject graph, having only one vertex with zero out-degree (a single output). It is characterized by its *depth* ( longest path from the root to a leaf) and number of inputs. A *cluster function* is associated to each cluster, and is matched against a subset of the gates, which are filtered out of the library through signature and symmetry analysis. In consequence, the time complexity depends on the number of clusters generated at each node, and not on the size of the library.

The covering algorithm implemented in our package is based on a clustering process enhanced to deal with reconvergent fanout. The algorithm assumes that the network has been partitioned into subject graphs and decomposed into base functions beforehand. Its pseudo-code is shown in figure 4.

```
Cover(node top, set nodes, set inputs)
(1)  if ( size(inputs) > max_input_cluster ) return; /* Prune Cluster Generation */
(2)  forall(x,inputs) {
(3)      if ( x is not mapped )
(4)          Cover(x, make_set(x), get_inputs(x));
(5)      if ( x is internal node And fanouts(x) reconverge at top )
(6)          Cover(top, nodes, new_inputs); }
(7)  if ( size(inputs) <= max_input_match)
(8)      library.match_boolean_class(top, inputs);
```

**Figure 4** Algorithm for network Boolean covering

The parameter **top** is the root node for the covering process. According to the library cells available, it may be interesting to allow the generation of clusters with a larger number of inputs. This allows the detection of reconvergent fanout cells that have a relatively small number of inputs and a larger number of internal signals. The cluster generation process will stop (line 1) when the number of cluster inputs reaches the **max_input_cluster** value. This controls the matching space exploration. The recursive algorithm computes all clusters for each node (lines 2-6) in the network rooted at top, and

*If there is a permutation operator $P$ and complementation operators $N_i, N_o$, such that $f(\mathbf{x}) = N_o g(PN_i\mathbf{x})$ is a tautology, then $f$ and $g$ belongs to the same $NPN$ class.

try to find the best set of matches that reduces the cost function. For each generated cluster, the number of its inputs is used as a filter before calling the match algorithm (lines 7-8). The parameter **max_input_match** indicates the maximum number of inputs of a library cell, and controls the size of the clusters that may be matched.

The cost function for a match (line 8) is given by the cost of the selected cell plus the cost of matching the cluster inputs, which have already been matched. In case of area optimization, the cell cost is only the cell surface. For each cluster input the best phase is selected. To handle reconvergent fanout (lines 5-6), if all fanouts of the cluster nodes reconverge at the node top, the new cluster with internal fanout will be generated and taken into account. There are additional parameters that heuristically control the performance of the mapping. For example, the covering may be set to not collapse reconvergent fanouts. This is useful when the library contains only tree-like cells because it provides faster results.

One more advantage of the Boolean covering is the possibility to extend the matching candidates by using the network don't cares. Two main approaches for the Boolean covering problem with don't cares were developed in the literature. In Ceres, Mailhot (Mailhot *et al.* 1993) generates all clusters up to 6 inputs. Don't care handling in Ceres is restricted to 4 input variables. An alternative method was proposed by Savoj (Savoj 1992). It is based on the tautology check by BDD computation but does not restrict the cluster size of incompletely defined functions. However, the use of the full don't care set for matching may slow down the mapping by two orders of magnitude. We don't address this point here.

## 6 RESULTS

We ran a set of benchmarks from the Mcnc suite to evaluate our Boolean mapping approach compared to the Sis structural one, on an Ultra Sparc I workstation. The version of Sis, modified to include our Boolean mapping, is referred here as *Land*. All benchmarks, except $c432, cm150a, x2, ttt2$, were optimized with the rugged script followed by a call to ***full simplify***. The Boolean matching was processed without don't cares.

Table 1 shows a comparison between Land and Sis for the library syncho.genlib, which is distributed with Sis. It is a standard cell library which includes Xor2, 2-1 Mux and tree like cells up to 8 inputs. Columns G (gates) shows the cell count used by the mapped circuit, and columns T (time) shows the Cpu time in seconds. Column $A_s$ for Sis presents the total area of the mapped circuit, and Columns $A_l/A_s$ for Land shows the normalized total area (ratio Land/Sis). For a depth of 2, Land generates *all* clusters with that logic depth. This means that we may have gates up to 4 inputs. Land reported an average area gain of 7% and was in average 3 times faster than Sis. For

| Bench. | | *Sis* | | Land, Depth=2 | | | Land, Depth=3 | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | G | $A_s$ | T | G | $\frac{A_l}{A_s}$ | T | G | $\frac{A_l}{A_s}$ | T |
| cm152a | 17 | 528 | 0.7 | 15 | .92 | 0.2 | 16 | .95 | 2.3 |
| z4ml | 31 | 856 | 0.9 | 23 | .78 | 0.3 | 23 | .78 | 0.6 |
| x2* | 40 | 1176 | 1.6 | 40 | .97 | 0.5 | 36 | .92 | 1.8 |
| cm150a* | 41 | 1328 | 1.3 | 34 | .78 | 0.8 | 35 | .79 | 0.8 |
| sao2 | 95 | 2728 | 3.4 | 93 | .95 | 1.0 | 86 | .91 | 7.7 |
| C432* | 155 | 4336 | 5.2 | 121 | .85 | 1.4 | 111 | .83 | 6.2 |
| 9symml | 156 | 4828 | 8.4 | 147 | .95 | 2.0 | 133 | .91 | 19.4 |
| C1355 | 244 | 7392 | 6.6 | 226 | .94 | 2.2 | 226 | .94 | 2.9 |
| C880 | 255 | 7768 | 7.8 | 250 | .92 | 2.6 | 261 | .88 | 13.0 |
| C1908 | 264 | 8016 | 7.9 | 261 | .93 | 2.4 | 267 | .95 | 4.1 |
| ttt2* | 344 | 10864 | 19.6 | 305 | .91 | 4.7 | 302 | .91 | 37 |
| apex6 | 476 | 13376 | 14.7 | 474 | .97 | 4.3 | 441 | .95 | 10.8 |
| **Aver.** | 100% | 100% | 100% | 94% | 93% | 29% | 91% | 91% | 136% |

*: not optimized.

**Table 1** Library *synch.genlib* on depth cluster

a depth of 3, the maximum number of inputs is 8 (tree-like cluster). In this case, Land reported an average gain of 9% in a similar computing time.

## 7 CONCLUSION

This paper presents an efficient algorithm for Boolean mapping based on a fast Boolean matching approach. The Boolean matching is based on testing techniques which prune the space search during the matching step by early detection of unsuccessful matches. Applied to the area minimization problem, the benchmarks have shown both an area and performance gain with respect to the structural mapping of Sis.

Interesting improvements will be considered in future work. The main cost in the Boolean mapping is related to the relative large amount of clusters that may be generated and evaluated. While libraries exhibit gates with a large number of inputs, those gates are not frequently used and account for a performance degradation. Specific filtering techniques will in that case increase the mapping speed.

Our future research will focus on Boolean mapping with don't care processing, mainly for use in low power applications.

## REFERENCES

Benini, L. De Micheli, G. (1995) A survey of boolean matching techniques for library binding, in *International Workshop on Logic and Architecture Synthesis*, 11–37.

Darringer, J. A., *et al.* (1984) Lss: A system for production logic synthesis. *IBM J. Res. Develop*, **28**(5):537–545.

Iman, S. and Pedram, M. (1996) Pose: Power optimization and synthesis environment, in *Design Automation Conference.*

Keutzer, K. (1987) DAGON: Technology Binding and Local Optimization by Dag Matching, In *Design Automation Conference*, 341–347.

De Micheli, G. (1994) *Synthesis and Optimization of Digital Circuits.* McGraw Hill.

Mailhot, F. and De Micheli, G. (1993) Algorithms for technology mapping based on binary decision diagrams and on boolean operations, in *Trans. on CAD*, **12**, no 5.

Savoj, H. (1992) *Don't Care in Multi-level Network Optimization.* PhD thesis, University of California, Berkeley, California.

Touati, H. (1990) *Performance Oriented Technology Mapping.* PhD thesis, University of California, Berkeley.

Thakur, S., *et al.* (1995) Delay minimal decomposition of multiplexers in technology mapping, in *Design Automation Conference.*

Trullemans, A.-M. and Zhang, Q. (1996a) Rapid gate matching with don't care. in *European Design & Test Conf*, 407–411, Paris.

Trullemans, A.-M. and Ferreira, R. (1996b) Extending the controlling value boolean matching to general structures, in *International Workshop on Logic and Architecture Synthesis*, 311–318, Grenoble.

## 8 BIOGRAPHY

**Ricardo Ferreira** received the M.Sc. degree in Computer Science at UFMG, Brazil, in 1994. He is an assistant at the University of Vicosa (UFV), Brazil, since 1993. In 1995, he joined the Microelectronics Laboratory of Université Catholique de Louvain, Belgium. He is now pursuing the Ph.D degree there.

**Anne-Marie Trullemans** was born in Brussels, Belgium, in 1944. She received the engineering degree and the Doctorat en Sciences Appliquées from the Université Catholique de Louvain (UCL), Belgium, in 1967 and 1974, respectively. She is now Associated Professor at UCL. Her interest is in the development of CAD tools for simulation and design of VLSI. Actually, she concentrates her research on digital circuits synthesis tools from HDL.

**Ricardo Jacobi** received his M.S. degree in Computer Science at the Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 1986. He received his Doctoral degree from UCL, Belgium, in Dec. 93. He is an adjoin professor of the Informatics Institute at the UFRGS since Dec. 89.