Identifying High-Level Components in Combinational Circuits

Travis Doom, Jennifer White, Anthony Wojcik

Department of Computer Science
Michigan State University
East Lansing MI

Greg Chisholm

Decision and Information Sciences Division
Argonne National Laboratory
Argonne IL

Abstract

The problem of finding meaningful subcircuits in a logic layout appears in many contexts in computer-aided design. Existing techniques rely upon finding exact matchings of subcircuit structure within the layout. These syntactic techniques fail to identify functionally equivalent subcircuits which are differently implemented, optimized, or otherwise obfuscated. We present a mechanism for identifying functionally equivalent subcircuits which is capable of overcoming many of these limitations. Such semantic matching is particularly useful in the field of design recovery.

1 Introduction

The identification of meaningful subcircuits within a larger design is of interest in many CAD applications. Of particular interest is the identification of a cluster of connected low-level devices which form a high-level component. Previous approaches to this problem have relied upon the discovery of subgraph isomorphisms to identify subcircuits [1–3]. While useful in applications such as converting a transistor netlist into a gate netlist, techniques which rely upon exact structural matching (syntactic algorithms) have limited application to higher levels of design since high-level components have many valid implementations.

We present a solution to the problem of identifying meaningful subcircuits which is structure independent. By using a *semantic* technique, we are capable of identifying subcircuits which are equivalent to a high-level component in many situations for which syntactic techniques fail [4]. The structural changes imposed by new implementations, design optimizations for area and power, and many other complicating factors cause purely syntactic techniques to fail, but are amenable to our semantic technique. Although semantic techniques are not limited to any particular level of circuit description or application, this paper will consider only the identification of high-level components from gate-level netlists.

The results presented in this paper are restricted to identifying the functionality of synchronous combinational components with no loops or other timing issues. Since combinational circuits are the basis of various logic circuits, the transformation of combinational netlists to a higher level of design (a netlist of high-level components and glue-logic) will provide a future basis for understanding sequential circuit functionality.

2 The equivalence problem

Consider some subcircuit (or *cluster*) of a combinational circuit. Such a subcircuit has $|\vec{i}|$ inputs, $\vec{i} = \langle i_1, \ldots, i_{|\vec{i}|} \rangle$, $|\vec{o}|$ outputs, $\vec{o} = \langle o_1, \ldots, o_{|\vec{o}|} \rangle$, and a vector of Boolean functions (the *cluster function*) which determines the relationships among them:

$$\vec{F}(\vec{i}) = \langle f_1(\vec{i}), \ldots, f_{|\vec{o}|}(\vec{i}) \rangle \quad (1)$$

Likewise, for any high level component with inputs $\vec{x}$ and outputs $\vec{y}$, there exists a vector of Boolean functions (the *pattern function*) which describes its behavior:

$$\vec{G}(\vec{x}) = \langle g_1(\vec{x}), \ldots, g_{|\vec{y}|}(\vec{x}) \rangle \quad (2)$$

Let us also define two bijections, π_I, the *input permutation function*, and π_O, the *output permutation function*:

$$\pi_I : \{i_1, \ldots, i_{|\vec{i}|}\} \rightarrow \{x_1, \ldots, x_{|\vec{x}|}\} \quad (3)$$

$$\pi_O : \{f_1, \ldots, f_{|\vec{o}|}\} \rightarrow \{g_1, \ldots, g_{|\vec{y}|}\} \quad (4)$$

0-8186-8409-7/98 $10.00

Definition 1 *Two vectors of Boolean functions $\vec{F}$ and $\vec{G}$ are input-permutation, output-permutation equivalent ($\mathcal{PP}$-equivalent) if bijections exist such that:*

$$\forall_k, 1 \leq k \leq |\vec{o}|, f_k(\vec{i}) = \pi_O(f_k)(\pi_I(\vec{i})) \qquad (5)$$

We can now define semantic equivalence for combinational designs.

Definition 2 *Two combinational designs D_1 and D_2 with corresponding vectors of Boolean functions $\vec{F}$ and $\vec{G}$ are* semantically equivalent *iff $\vec{F}$ and $\vec{G}$ are $\mathcal{PP}$-equivalent. The input bijection π_I and the output bijection π_O under which $\vec{F}$ and $\vec{G}$ are $\mathcal{PP}$-equivalent describe the* semantic matching *between D_1 and D_2.*

3 Other approaches

We now describe some existing algorithms which have been used to solve some instances of the equivalence problem.

3.1 Factorial permutation

Although testing the equivalence of two single-output functions represented as reduced, ordered Binary Decision Diagrams (BDDs) can be achieved in constant time [5], such a test requires that the correspondences between the input variables be clearly identified. Because input and output variable correspondences are not generally available, the straightforward method for determining if two multiple-output functions are $\mathcal{PP}$-equivalent is to test for equivalence over the set of $|\vec{i}|! \cdot |\vec{o}|!$ possible pairs of bijection functions (i.e. over all input and output permutations). For numbers of inputs greater than seven to nine, the straightforward permutation technique is computationally intractable.

3.2 Logic verification

In logic verification, a *specification* describing some functional behavior is compared to a circuit *implementation* of that function to prove equivalence. Verification techniques exist which are capable of dealing with problems involving large numbers of inputs, sequential behavior, and with significant numbers of intermediate gates. Verification techniques, however, require that correspondences between the implementation and specification be known [6]. Since we cannot assume knowledge of such correspondences when attempting to identify high-level components in a flat netlist, verification techniques are generally not applicable.

3.3 Boolean matching

Technology mapping (also known as *cell-library binding*) is part of the synthesis process whereby logic representations must be transformed into interconnections of a set of implementation dependent cells. Technology mapping is used to create cost-optimized implementations for some logic function or Boolean network in a particular implementation style in terms of some library of building blocks (cells). Detection of equivalence of these Boolean functions to cells, referred to as *Boolean matching*, is a well studied problem [7].

In many ways, the problem of determining equivalence between a combinational circuit and a high-level entity library is similar to the problem of Boolean matching. Boolean matching algorithms are designed to efficiently match small (fewer than six inputs) single-output clusters with a component of their cell libraries which implements the function at the least cost.

A general solution to the equivalence problem, on the other hand, must be capable of efficiently matching functions with any number of inputs and outputs, but need only concern itself with a single (although possibly multiple-output) *pattern function* rather than an entire library of such functions. The goal of semantic matching is not to find the "best" implementation of a function from a set of possible implementations, but to identify equivalence and variable correspondences between a particular subcircuit and a particular high-level component. It appears that no suitable solution has been proposed in the literature for this problem.

4 Semantic matching

We now describe an algorithm for determining if a semantic match exists between a subcircuit and a high-level component. The semantic matching problem is concerned with the identification of high-level components more complex then those dealt with in Boolean matching, but lacking the input/output correspondences between the logic design and the library components which verification techniques require. Since the functionality of the high-level component may be represented in any number of structural forms, it is necessary to identify the subcircuit by proving semantic equivalence.

4.1 Boolean signatures

A *signature* of a Boolean function is a unique characteristic representation of some property of the function. Although it is possible for two otherwise unrelated functions to have the same signature, having equal signatures is a necessary condition for an equivalence matching. Boolean signatures have been used successfully to increase the efficiency of Boolean matching algorithms [8].

A *signature function* is a function which takes a generic function as an input and returns a characteristic signature for that input function. The value of a signature function must be determined only by the behavior of the generic function; variable order, variable labels, and random elements may not be used as part of the signature determination.

Since sharing a signature is a necessary condition for equivalence, signature functions can be used to eliminate functions from equivalence consideration. The primary limit to the effectiveness of such filtering is the complexity cost of the signature function.

4.2 The vector input signature

We introduce a new signature function which has proven to be an adequate initial filter for many problems. This signature takes advantage of the fact that the vector functions under consideration consist of multiple functions, each corresponding to a single output.

Definition 3 *A positive (negative) Boolean* unit vector *is defined as a vector in which exactly one element has the value 1 (0) and in which all other elements have the value 0 (1).*

Definition 4 *For any vector of Boolean functions* $\vec{F}(\vec{i}) = \vec{o}$ *we define* i_j*'s positive unit* vector input signature *to be the sum of the function outputs (i.e. the cardinality of the on-set) when the positive unit vector with input* i_j *equal to 1 is applied.*

$$F_{+vec}(i_j) = \sum_{n=1}^{|\vec{i}|} f_n(\vec{u}), \text{ where } u_k = 1 \text{ iff } k = j \quad (6)$$

The negative unit vector input signature is defined similarly.

Definition 5 *For any vector of Boolean functions* $\vec{F}(\vec{i}) = \vec{o}$ *we define the function's* vector signature *to be an ordered set of* $|\vec{i}|$ (x, y) *pairs where each such pair corresponds to an input* i_j *of* $\vec{F}$ *and where* x *(*y*) represents the positive (negative) unit vector input signature.*

Table 1 shows the results of applying the vector signature to the vector function of a 4-bit ALU. The resulting vector signature is: $\{2 \times (1,7), 1 \times (2,2), 1 \times (2,5), 6 \times (2,7), 3 \times (3,5), 1 \times (6,5)\}$.

Input Names	Vector Input Signature	
	Positive	Negative
sel1, Cn'	1	7
sel3	2	2
a0	2	5
sel0, sel2, b0, b1, b2, b3	2	7
a1, a2, a3	3	5
m	6	5

Table 1: **Vector Input Signature for the TI 54181 4-bit ALU.** The positive and negative unit vector input signatures are shown for a 4-bit ALU with selection inputs sel0-3, mode input m, carry input Cn', and data inputs a0-3 and b0-3. The vector signature partitions the function inputs into 5 signature classes.

Vector signatures are an effective signature for multiple-output functions in which the number of inputs is not significantly larger than the number of outputs. This is not surprising when we consider that the number of outputs determines the size of the range of the signature function (the range of the function is $|\vec{o}|^2$).

When any vector input signature uniquely defines a single input (that is, no other input shares its (x,y) signature value) then a correspondence is clearly identified. The vector signature for the 4-bit ALU shown in Table 1 has three signature classes with only a single member (the signature classes for `sel3`, `a0`, and `m`). Thus, any correspondence between the pattern function representing the ALU and any cluster function would have to identify the cluster inputs which correspond to `sel3` and `m`. Using any one such identified correspondence allow us to describe an additional $2(n-1)$ vectors in which both the input under test and the input for which correspondence is known have the opposite value from the rest of the inputs. These vectors can be applied to create more precise signature classes and possibly uniquely identify more correspondences. This process can be continued until all unique correspondences have been exploited.

4.3 The algorithm

Our approach to the semantic matching problem makes use of signature information to reduce the number of input correspondences which must be consid-

ered. This is accomplished though the use of *suspect sets*, defined below.

Definition 6 *The signature values for any input signature function can be used to partition the function inputs into classes corresponding to their signature. We refer to such a list of inputs as a* signature class.

The following result is clear.

Theorem 1 *Input correspondences between the pattern and cluster function can only take place between members of their respective signature classes with equal signature value.*

Let $\vec{F}(\vec{i}) = \vec{o}$ be the vector of Boolean functions for some subcircuit. Let $\vec{G}(\vec{x}) = \vec{y}$ be the vector of Boolean functions for a high-level component. Semantic equivalence and input/output correspondences between the subcircuit and the high-level component can be determined as follows:

Semantic Matching Algorithm

- **Step 1:** *Create Binary Decision Diagrams.* Create BDDs (under some reasonable variable ordering) for the outputs of each vector of Boolean functions.
- **Step 2:** *Determine Signature Classes.* Determine the vector signatures for $\vec{F}$ and $\vec{G}$ and partition each function's input variables into signature classes. If the signature classes and partition sizes are not equivalent, then the functions cannot be equivalent (this check is the "traditional" way in which signatures are used in Boolean matching).
- **Step 3:** *Determine Suspect Sets.* For each of the cluster function $\vec{F}$'s inputs, $i_1, \ldots, i_{|\vec{i}|}$, create a *suspect set.* The suspect set is defined to be the subset of pattern function $\vec{G}$'s inputs, $x_1, \ldots, x_{|\vec{x}|}$, which have the same signature as the input i_j to which the set corresponds. Initialy, each cluster input i_j's suspect set will contain the partitioned set of pattern function inputs which have the same vector signature class as the input. Additional filtering of the suspect set must be accomplished by applying other appropriate input signature functions. This process can be repeated until all suspect sets are below some threshold size (see Section 4.3.1).
- **Step 4:** *Iterate though Legal Input Correspondences.* Theorem 1 allows us to eliminate from consideration all matchings which include a correspondence between a cluster function input i_j and any pattern function input x_k which is not present in i_j's suspect set. Therefore, we exhaustively search the pruned matching space by examining every legal correspondence.

  For each match, we substitute the decision variables of the BDD(s) representing the cluster function inputs with the variables in the BDD(s) representing their matched pattern function inputs. By using this technique we avoid having to "re-canonicalize" completely because the order for the matching variables will be identical under the same BDD manager. Reordering the variables of a BDD in this way can be performed using the standard BDD library substitution function in time proportional to the size of the BDD.
- **Step 5:** *Determine Legal Output Correspondences.* Compare each substituted BDD representing a cluster function output o_j with each BDD representing a pattern function output. Since each BDD is now represented in terms of the same decision variables, each such equivalence check is an $O(1)$ [5] operation. We define the output suspect set for each cluster function output to be the set of pattern function outputs whose BDDs are equivalent. If all suspect sets contain a unique match, then matching under consideration is a legal correspondence and the functions are equivalent.

4.3.1 Complexity

The technique presented in Section 3.1 requires $|\vec{i}|!|\vec{o}|!$ comparisons. Our algorithm achieves significant improvement.

Let n represent the cardinality of the largest input suspect set determined in Step 3. An upper bound on the number of legal input correspondences is $n!^{|\vec{i}|}$. As long as n is constrained to a reasonably small size (less than seven to nine), it can be treated as a constant value c and the input correspondence selection is exponential in complexity: $O(c^{|\vec{i}|})$. Reasonably small values of n can be achieved through pruning suspect set sizes by applying multiple signature values until all suspect set sizes fall below some threshold.

Such pruning is effective in most components save those with large numbers of symmetric inputs (which are indistinguishable to Boolean signatures). In such cases, however, *any* input matching will succeed for the symmetric inputs, which actually simplifies the

process of proving semantic equivalence as a correspondence will be identified very early in the execution of the algorithm.

Although BDDs are an efficient mechanism for representing the functionality of most components, their size may become intractably large for certain functions under some (or all) variable orderings [5]. Since we can indicate a "good" variable ordering for our pattern function library, we can eliminate most BDD based concerns. If the BDD for any cluster function output exceeds the size of the largest BDD representing a pattern function output, we can immediately discard that input matching and discontinue BDD generation, since no legal correspondence can exist between functions which have BDDs of different sizes under the same variable ordering. Pathological functions (such as multipliers) which have no efficient BDD representation remain an open issue.

Since each cluster output BDD is tested against each pattern output BDD exactly once in Step 5, the complexity of determining legal output correspondence is only $O(|\vec{o}|^2)$.

Therefore, the overall complexity of this approach is $O(c^{|\vec{i}|}|\vec{o}|^2) = O(c^{|\vec{i}|})$. This exponential algorithm is a significant improvement over factorial methods and makes semantic matching feasible for most components of reasonable size.

5 Experimental results

Our algorithm for semantic matching was implemented in C using the University of Colorado's decision diagram library [9] and executed on a Sun SPARCstation Ultra.

Table 2 provides a comparison of our procedure with the factorial approach. For each component, we report the size of the subcircuit, the size for the BDD representation of the component's pattern function (under some reasonable variable ordering), the number of input matchings and the total number of BDD equivalence checks made during the program's runtime. The runtime shown is the worst-case runtime (a complete search of the correspondence space). For non-symmetric circuits, the time to determine a single correspondence can be considered roughly 50% of the overall run time. For circuits containing symmetries, the entire time is necessary to identify all legal correspondences, but only a fraction of the time is necessary to determine a single correspondence.

The `z4ml` circuit (a 3-bit adder) shows a case in which the inputs are indistinguishable from their vector signature, and thus the number of input matchings is 7!. Note that due to the algorithm's automatic pruning of the output search space, the number of comparisons is only 20304, an order of magnitude less then the number of comparisons necessary in a 120160 (7!4!) non-pruned search.

The `alu4` circuit (a 4-bit ALU) is complex enough to have fairly well distributed vector signatures and thus is able to take advantage of vector signature information to recognize that only 8640 of the greater than 87 billion possible input matchings can possibly produce a legal correspondence. The use of vector signatures has made this intractable comparison feasible. Furthermore, note that only 69411 comparisons are necessary out of the 3.5 million billion total correspondences (14!8!) possible.

Obviously, circuits exist for which a single vector signature does not adequately prune the matching space. A single vector signature is capable of reducing the number of input matchings for the 173 input LGSynth'93 `pair` circuit from 173! to approximately 73!. While certainly a significant reduction of search space, additional signatures need to be applied to permit semantic matching within a reasonable execution time.

6 Conclusion

To summarize, we have met our goal of achieving a method for determining a semantic matching between a subcircuit and a high-level component in a tractable number of comparisons. We have presented the underlying equivalence problem and provided a algorithm based on the concept of suspect sets capable of solving problems of a reasonable size.

Preliminary experiments demonstrate the effectiveness of the technique using a single vector signature filter. Future goals include the introduction of additional filters to decrease the runtime and increase the capabilities of the program.

In the long term, we will use this technique as a reengineering tool. Semantic matching techniques allow us to achieve a functional specification of many digital designs by identifying clusters of logic which correspond to higher-level functional components. By identifying high-level components such as ALUs, adders, multiplexers, decoders, encoders, and other common functional entities within the circuit, we reduce the complexity of producing functional descriptions as well as of identifying data lines, control lines, and other "additional knowledge" [10] which may be of use in further specifying the design. Such an approach requires the implementation of efficient

Circuit Name	Num. Inputs	Num. Outputs	BDD Size	Input Matchings		Correspondences Checked		Run Time (sec)
				Method 1	Method 2	Method 1	Method 2	
alu2	10	6	231	3.6e+06	2.0e+00	2.9e+10	32	0.2
alu4	14	8	1452	8.7e+10	8.6e+03	3.5e+15	6.9e+04	232.4
cc	21	20	57	5.1e+19	1.4e+07	1.2e+38	1.5e+09	37675.5
f51m	8	8	73	4.0e+04	4.8e+01	1.6e+09	4.3e+02	0.1
pm1	16	13	42	2.1e+13	2.0e+05	1.3e+23	2.8e+06	273.4
sct	19	15	102	1.2e+17	4.0e+07	1.6e+29	6.0e+08	75647.4
t481	16	1	202	2.1e+13	2.3e+07	2.1e+13	2.3e+07	88354.5
z4ml	7	4	47	5.0e+03	5.0e+03	1.2e+05	2.0e+04	4.55

Table 2: **Experimental Results.** The circuits included in this table are a subset of the LGSynth'93 benchmark suite. The results listed for Method 1 are calculated for the Factorial Permutation approach (Section 3.1). The results presented for Method 2 are experimental results for a single vector signature implementation of the algorithm presented in Section 4.3.

partitioning techniques [11] as well as the identification and incorporation of *don't care* conditions into the semantic matching algorithm.

Acknowledgements

The authors gratefully acknowledge the contributions of Steve Eckmann and Ken Dritz of Argonne National Laboratory. Their comments and insight have been invaluable in this research.

References

[1] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 31–37, June 1993.

[2] M. Bochner, "LOGEX – an automatic logic extractor from transistor to gate level for CMOS technology," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 517 – 522, June 1988.

[3] F. Luellau, T. Iloepken, and E. Barke, "A technology independent block extraction algorithm," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 610 – 615, June 1984.

[4] S. Eckmann and G. H. Chisholm, "Assigning functional meaning to digital circuits," Technical Report ANL/DIS/TM-43, Argonne National Laboratory, July 1997.

[5] R. E. Bryant, "Symbolic manipulation of boolean functions using a graphical representation," in *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, pp. 688 – 694, June 1985.

[6] Y. Lai, S. Sastry, and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 452–458, Oct. 1992.

[7] L. Benini and G. D. Micheli, "A survey of boolean matching techniques for library binding," in *Proceedings of the ACM TODAES*, July 1997.

[8] F. Mailhot and G. D. Micheli, "Algorithms for technology mapping based on Binary Decision Diagrams and on boolean operations," *IEEE Transactions on CAD/ICAS*, vol. 12, pp. 599–620, May 1993.

[9] F. Somenzi, "CUDD: CU decision diagram package." Available on the World Wide Web as URL: www.bessie.colorado.edu/~fabio/CUDD, 1997. Release 2.1.2.

[10] M. Ohmura, H. Yasuura, and K. Tamaru, "Extraction of functional information from combinational circuits," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 176 – 179, Nov. 1990.

[11] T. Doom, J. White, and G. Chisholm, "The identification of functional components in combinational circuits," Tech. Rep. Submitted for Publication, Division of Information Science, Argonne National Laboratory, November 1997.