

To SAT or Not to SAT: Ashenhurst Decomposition in a Large Scale

Hsuan-Po Lin, Jie-Hong R. Jiang, and Ruei-Rung Lee

Department of Electrical Engineering/Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT

Functional decomposition is a fundamental operation in logic synthesis. Prior BDD-based approaches to functional decomposition suffer from the memory explosion problem and do not scale well to large Boolean functions. Variable partitioning has to be specified *a priori* and often restricted to a few bound-set variables. Moreover, non-disjoint decomposition requires substantial sophistication. This paper shows that, when Ashenhurst decomposition (the simplest and preferable functional decomposition) is considered, both single- and multiple-output decomposition can be formulated with satisfiability solving, Craig interpolation, and functional dependency. Variable partitioning can be automated and integrated into the decomposition process without the bound-set size restriction. The computation naturally extends to non-disjoint decomposition. Experimental results show that the proposed method can effectively decompose functions with up to 300 input variables.

1. INTRODUCTION

Functional decomposition [1, 5, 9] aims at decomposing a Boolean function into a network of smaller sub-functions. It is a fundamental operation in logic synthesis and has various applications to FPGA synthesis and to the minimization of circuit communication complexity. There has been intensive work on this subject since FPGAs became a viable design style and BDD packages were available. See, e.g., [14] for a survey.

Most prior work on functional decomposition used BDD as the underlying data structure. With proper variable ordering, BDD can be exploited for the computation of functional decomposition. Despite having been a powerful tool, BDD poses several limitations: Firstly, BDD suffers from the memory explosion problem. In representing a Boolean function, a BDD can be of large size (in the worst case, exponential in the number of variables). It is even more so when special variable ordering rules need to be imposed on BDDs for functional decomposition. Therefore it is typical that a function under decomposition can have just a few variables. Secondly, variable partitioning needs to be specified *a priori*, and cannot be automated as an integrated part of the decomposition process. In order to effectively enumerate different variable partitions and keep BDD sizes reasonably small, the set of bound set variables cannot be large. Thirdly, for BDD-based approaches, non-disjoint decomposition cannot be handled easily. In essence, decomposability needs to be analyzed by cases exponential in the number of joint (or common) variables. Finally, even though multiple-output decomposition [16] can be converted to single-output decomposition [8], BDD sizes may grow largely in this conversion.

The above limitations motivate the need for new data structures and computation methods for functional decom-

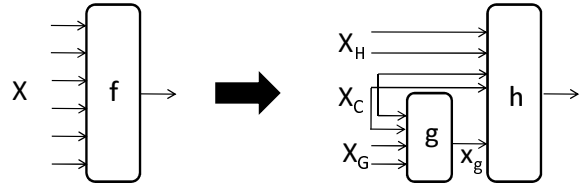


Figure 1: Ashenhurst decomposition

position. This paper shows that, when Ashenhurst decomposition [1] is considered, these limitations can be overcome through satisfiability (SAT) based formulation. Ashenhurst decomposition is a special case of functional decomposition, where, as illustrated in Figure 1, a function $f(X)$ is decomposed into two sub-functions $h(X_H, X_C, x_g)$ and $g(X_G, X_C)$ with $f(X) = h(X_H, X_C, g(X_G, X_C))$. For general functional decomposition, the function g can be a functional vector (g_1, \dots, g_k) instead. It is this simplicity that makes Ashenhurst decomposition particularly attractive in practical applications.

The enabling techniques of our method, in addition to SAT solving, include Craig interpolation [4] and functional dependency [10]. Specifically, the decomposability of function f is formulated as SAT solving, the derivation of function g is by Craig interpolation, and the derivation of function h is by functional dependency.

Compared with BDD-based methods, the proposed algorithm is advantageous in the following aspects. Firstly, it does not suffer from the memory explosion problem and is scalable to large functions. Experimental results show that Boolean functions with more than 300 input variables can be decomposed effectively. Secondly, variable partitioning needs not be specified *a priori*, and can be automated and derived on the fly during decomposition. Hence the size of the bound set variables X_G need not be small. Thirdly, it works for non-disjoint decomposition naturally. Finally, it is easily extendable to multiple-output decomposition. Nonetheless, a limitation of the method is its expensive generalization to functional decomposition beyond Ashenhurst's special case.

As interconnects become a dominating concern in modern nanometer IC designs, scalable decomposition methods play a pivotal role in circuit communication minimization. With the advantages of the proposed method, hierarchical logic decomposition could be made feasible in practice. In addition, our results may shed light on scalable Boolean matching for heterogeneous FPGAs as well as topologically constrained logic synthesis.

This paper is organized as follows. Section 2 introduces essential preliminaries. Our main algorithms are presented in Section 3, and evaluated with experimental results in Section 4. Section 5 compares some related prior work. Finally, Section 6 concludes the paper and outlines future work.

2. PRELIMINARIES

As conventional notation, sets are denoted by upper-case letters, e.g., S ; set elements are in lower-case letters, e.g., $e \in S$. The cardinality of S is denoted by $|S|$. A partition of a set S into $S_i \subseteq S$ for $i = 1, \dots, k$ (with $S_i \cap S_j = \emptyset, i \neq j$, and $\bigcup_i S_i = S$) is denoted by $\{S_1|S_2|\dots|S_k\}$. For a set X of Boolean variables, its set of valuations (or truth assignments) is denoted by $\llbracket X \rrbracket$, e.g., $\llbracket X \rrbracket = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ for $X = \{x_1, x_2\}$.

2.1 Functional Decomposition

DEFINITION 1. *Given a completely specified Boolean function f , variable x is a support variable of f if $f_x \neq f_{\neg x}$, where f_x and $f_{\neg x}$ are the positive and negative cofactors of f on x , respectively.*

DEFINITION 2. *A set $\{f_1(X), \dots, f_m(X)\}$ of completely specified Boolean functions is (jointly) decomposable with respect to some variable partition $X = \{X_H|X_G|X_C\}$ if every function $f_i, i = 1, \dots, m$, can be written as*

$$f_i(X) = h_i(X_H, X_C, g_1(X_G, X_C), \dots, g_k(X_G, X_C))$$

for some functions h_i, g_1, \dots, g_k with $k < |X_G|$. The decomposition is called *disjoint* if $X_C = \emptyset$, and *non-disjoint* otherwise.

It is known as *single-output decomposition* for $m = 1$, and *multiple-output decomposition* for $m > 1$. Note that, in multiple-output decomposition, functions h_1, \dots, h_m share the same functions g_1, \dots, g_k . For $k = 1$, the decomposition is known as the so-called *Ashenhurst decomposition* [1].

Note that, for $|X_G| = 1$, there is no successful decomposition because of the violation of the criterion $k < |X_G|$. On the other hand, the decomposition trivially holds if $X_C \cup X_G$ or $X_C \cup X_H$ equals X . The corresponding variable partition is called *trivial*. This paper is concerned about decomposition under non-trivial variable partition. Moreover, we focus on Ashenhurst decomposition.

The decomposability of a set $\{f_1, \dots, f_m\}$ of functions under the variable partition $X = \{X_H|X_G|X_C\}$ can be analyzed through the so-called *decomposition chart*, consisting of a set of matrices, one for each member of $\llbracket X_C \rrbracket$. The rows and columns of a matrix are indexed by $\{1, \dots, m\} \times \llbracket X_H \rrbracket$ and $\llbracket X_G \rrbracket$, respectively. For $i \in \{1, \dots, m\}$, $a \in \llbracket X_H \rrbracket$, $b \in \llbracket X_G \rrbracket$, and $c \in \llbracket X_C \rrbracket$, the entry with row index (i, a) and column index b of the matrix of c is of value $f_i(X_H = a, X_G = b, X_C = c)$.

PROPOSITION 1. [1, 5, 9] *A set $\{f_1, \dots, f_m\}$ of Boolean functions is decomposable as*

$$f_i(X) = h_i(X_H, X_C, g_1(X_G, X_C), \dots, g_k(X_G, X_C))$$

for $i = 1, \dots, m$ under variable partition $X = \{X_H|X_G|X_C\}$ if and only if, for every $c \in \llbracket X_C \rrbracket$, the corresponding matrix of c has at most 2^k column patterns (i.e., at most 2^k different kinds of column vectors).

2.2 Functional Dependency

DEFINITION 3. *Given a Boolean function $f : \mathbb{B}^m \rightarrow \mathbb{B}$ and a vector of Boolean functions $G = (g_1(X), \dots, g_n(X))$ with $g_i : \mathbb{B}^m \rightarrow \mathbb{B}$ for $i = 1, \dots, n$, over the same set of variable vector $X = (x_1, \dots, x_m)$, we say that f functionally depends on G if there exists a Boolean function $h : \mathbb{B}^n \rightarrow \mathbb{B}$, called the dependency function, such that $f(X) = h(g_1(X), \dots, g_n(X))$. We call functions f, G , and h the target function, base functions, and dependency function, respectively.*

Note that functions f and G are over the same domain in the definition; h needs not depend on all of the functions in G .

The necessary and sufficient condition of the existence of the dependency function h was given in [7]. Moreover a SAT-based computation of functional dependency was presented in [10]. It forms an important ingredient in part of our formulation.

2.3 Satisfiability and Interpolation

A brief introduction to SAT solving and circuit-to-CNF conversion, essential to our development, can be found in [10]. To introduce terminology and notation for later use, we mention the following theorem.

THEOREM 1 (CRAIG INTERPOLATION THEOREM). [4] *Given two Boolean formulas φ_A and φ_B , with $\varphi_A \wedge \varphi_B$ unsatisfiable, then there exists a Boolean formula ψ_A referring only to the common variables of φ_A and φ_B such that $\varphi_A \Rightarrow \psi_A$ and $\psi_A \wedge \varphi_B$ is unsatisfiable.*

The Boolean formula ψ_A is referred to as the *interpolant* of φ_A with respect to φ_B . Some modern SAT solvers, e.g., MiniSat [6], are capable of constructing an interpolant from an unsatisfiable SAT instance [13].

3. MAIN ALGORITHMS

We show that Ashenhurst decomposition of a set of Boolean functions $\{f_1, \dots, f_m\}$ can be achieved by SAT solving, Craig interpolation, and functional dependency. Whenever a non-trivial decomposition exists, we derive functions h_i and g automatically for $f_i(X) = h_i(X_H, X_C, g(X_G, X_C))$ along with the corresponding variable partition $X = \{X_H|X_G|X_C\}$.

3.1 Single-Output Ashenhurst Decomposition

We first consider Ashenhurst decomposition for a single function $f(X) = h(X_H, X_C, g(X_G, X_C))$.

3.1.1 Decomposition with known variable partition

Proposition 1 in the context of Ashenhurst decomposition of a single function can be formulated as satisfiability solving as follows.

PROPOSITION 2. *A completely specified Boolean function $f(X)$ can be expressed as $h(X_H, X_C, g(X_G, X_C))$ for some functions g and h if and only if the Boolean formula*

$$\begin{aligned} & (f(X_H^1, X_G^1, X_C) \neq f(X_H^1, X_G^2, X_C)) \wedge \\ & (f(X_H^2, X_G^2, X_C) \neq f(X_H^2, X_G^3, X_C)) \wedge \\ & (f(X_H^3, X_G^3, X_C) \neq f(X_H^3, X_G^1, X_C)) \end{aligned} \quad (1)$$

is unsatisfiable, where a superscript i in Y^i denotes the i^{th} copy of the instantiation of variables Y .

Observe that Formula (1) is satisfiable if and only if there exists more than two distinct column patterns in some matrix of the decomposition chart. Hence its unsatisfiability is exactly the condition of Ashenhurst decomposition.

Note that, unlike BDD-based counterparts, the above SAT-based formulation of Ashenhurst decomposition naturally extends to non-disjoint decomposition. It is because the unsatisfiability checking of Formula (1) essentially tries to assert that under every valuation of variables X_C the corresponding matrix of the decomposition chart has at most two column patterns. In contrast, BDD-based methods have to check the decomposability under every valuation of X_C separately.

Whereas the decomposability of function f can be checked through SAT solving of Formula (1), the derivations of functions g and h can be realized through Craig interpolation and functional dependency, respectively, as shown below.

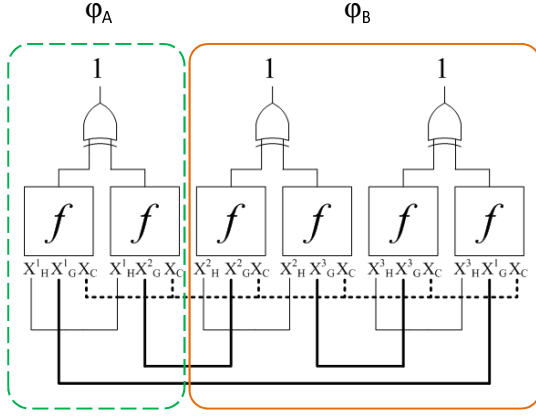


Figure 2: Circuit representing the conjunction condition of Formulas (2) and (3)

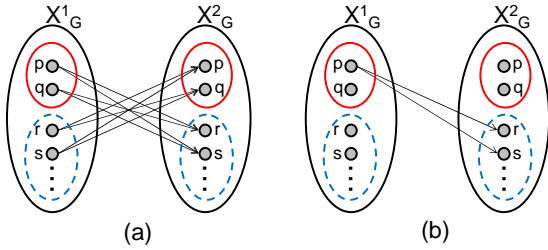


Figure 3: (a) Relation characterized by $\psi_A(X_G^1, X_G^2, c)$ for some $c \in [X_C]$; (b) relation after cofactoring $\psi_A(X_G^1 = p, X_G^2, c)$ with respect to some $p \in [X_G^1]$

To derive function g , we partition Formula (1) into two sub-formulas

$$\varphi_A = f(X_H^1, X_G^1, X_C) \neq f(X_H^1, X_G^2, X_C), \text{ and} \quad (2)$$

$$\varphi_B = (f(X_H^2, X_G^2, X_C) \neq f(X_H^2, X_G^3, X_C)) \wedge (f(X_H^3, X_G^3, X_C) \neq f(X_H^3, X_G^1, X_C)). \quad (3)$$

Figure 2 shows the corresponding circuit representation of Formulas (2) and (3). The circuit representation can be converted into a CNF formula in linear time [15], and thus can be checked for satisfiability.

LEMMA 1. *For function $f(X)$ decomposable under Ashenhurst decomposition with variable partition $X = \{X_H|X_G|X_C\}$, the interpolant ψ_A with respect to φ_A of Formula (2) and φ_B of Formula (3) corresponds to a characteristic function such that,*

- (i) for φ_A satisfiable under some $c \in [X_C]$, $\psi_A(b_1, b_2, c) = 1$ with $b_1 \in [X_G^1]$ and $b_2 \in [X_G^2]$ if and only if the column vectors indexed by b_1 and b_2 in the matrix of c of the decomposition chart of f are different;
- (ii) for φ_A unsatisfiable under some $c \in [X_C]$, there is only one column pattern in the matrix of c of the decomposition chart of f ;
- (iii) for unsatisfiable φ_A , variables X_G are not the support variables of f and thus $\{X_H|X_G|X_C\}$ is a trivial variable partition for f .

Figure 3(a) illustrates the relation characterized by interpolant $\psi_A(X_G^1, X_G^2, c)$ for some $c \in [X_C]$. The left and right

sets of gray dots denote the elements of $[X_G^1]$ and $[X_G^2]$, respectively. For function f to be decomposable, there are at most two equivalence classes for the elements of $[X_G^i]$ for $i = 1, 2$. In the figure, the two clusters of elements in $[X_G^i]$ signify two equivalence classes of column patterns indexed by $[X_G^i]$. An edge (b_1, b_2) between $b_1 \in [X_G^1]$ and $b_2 \in [X_G^2]$ denotes that b_1 is not in the same equivalence class as b_2 , i.e., $\psi_A(b_1, b_2, c) = 1$. For example, p and r in the figure are in different equivalence classes and $\psi_A(p, r, c) = 1$, whereas p and q are in the same equivalence class and $\psi_A(p, q, c) = 0$. Essentially the set of such edges are characterized by the equivalence relation $\psi_A(X_G^1, X_G^2, c)$. So every element in one equivalence class of $[X_G^1]$ is connected to every element in the other equivalence class of $[X_G^2]$, and vice versa, in Figure 3(a).

We next show how to extract function g from the interpolant ψ_A .

LEMMA 2. *For an arbitrary $a \in [X_G^1]$, the cofactored interpolant $\psi_A(X_G^1 = a, X_G^2, X_C)$ is a legal implementation of function $g(X_G^2, X_C)$.*

After renaming X_G^2 to X_G , we get the desired $g(X_G, X_C)$.

Consider Figure 3. After cofactoring $\psi_A(X_G^1, X_G^2, c)$ with respect to $p \in [X_G^1]$, all the edges in Figure 3(a) will disappear except for the ones connecting p with the elements in the other equivalence class of $[X_G^2]$ as shown in Figure 3(b). Hence $\psi_A(p, X_G^2, c)$ can be used as an implementation of g function.

So far we have successfully obtained function g by interpolation. Next we need to compute function h . The problem can be formulated as computing functional dependency as follows. Let $f(X)$ be our target function; let function $g(X_G, X_C)$ and identity functions $v_x(x) = x$, one for every variable $x \in X_H \cup X_C$, be our base functions. So the computed dependency function corresponds to our desired h . Since functional dependency can be formulated using SAT solving and interpolation [10], it well fits in our computation framework.

REMARK 1. *For disjoint decomposition, i.e., $X_C = \emptyset$, we can simplify the derivation of function h , without using functional dependency.*

Given two functions $f(X)$ and $g(X_G)$ with variable partition $X = \{X_H|X_G\}$, we aim to find a function $h(X_H, x_g)$ such that $f(X) = h(X_H, g(X_G))$, where x_g is the output variable of function $g(X_G)$. Let $a, b \in [X_G]$ with $g(a) = 0$ and $g(b) = 1$. Then by Shannon expansion

$$h(X_H, x_g) = (\neg x_g \wedge h_{\neg x_g}(X_H)) \vee (x_g \wedge h_{x_g}(X_H)),$$

where $h_{\neg x_g}(X_H) = f(X_H, X_G = a)$ and $h_{x_g}(X_H) = f(X_H, X_G = b)$. The derivation of the offset and onset minterms is easy because we can pick an arbitrary minterm c in $[X_G]$ and see if $g(c)$ equals 0 or 1. We then perform SAT solving on either $g(X_G)$ or $\neg g(X_G)$ depending on the value $g(c)$ to derive another necessary minterm.

The above derivation of function h , however, does not scale well for decomposition with large $|X_C|$ because we may need to compute $h(X_H, X_C = c, x_g)$, one for every valuation $c \in [X_C]$. There are $2^{|X_C|}$ cases to analyze. Consequently when common variables exist, functional dependency may be a better approach to computing h .

The correctness of the so-derived Ashenhurst decomposition follows from Lemma 2 and Proposition 1, as the following theorem states.

THEOREM 2. *Given a function f decomposable under Ashenhurst decomposition with variable partition $X = \{X_H|X_G|X_C\}$,*

then $f(X) = h(X_H, X_C, g(X_G, X_C))$ for functions g and h obtained by the above derivation.

3.1.2 Decomposition with unknown variable partition

The previous construction assumes that a variable partition $X = \{X_H|X_G|X_C\}$ is given. We show how to automate the variable partition within the decomposition process of function f . A similar approach was used in [11] for bi-decomposition of Boolean functions.

For each variable $x_i \in X$ we introduce two control variables α_{x_i} and β_{x_i} . In addition we instantiate variable X into six copies X^1, X^2, X^3, X^4, X^5 , and X^6 . Let

$$\varphi_A = (f(X^1) \neq f(X^2)) \wedge \bigwedge_i ((x_i^1 \equiv x_i^2) \vee \beta_{x_i}) \text{ and} \quad (4)$$

$$\begin{aligned} \varphi_B = & (f(X^3) \neq f(X^4)) \wedge (f(X^5) \neq f(X^6)) \wedge \\ & \bigwedge_i (((x_i^2 \equiv x_i^3) \wedge (x_i^4 \equiv x_i^5) \wedge (x_i^6 \equiv x_i^1)) \vee \alpha_{x_i}) \wedge \\ & \bigwedge_i (((x_i^3 \equiv x_i^4) \wedge (x_i^5 \equiv x_i^6)) \vee \beta_{x_i}), \end{aligned} \quad (5)$$

where $x_i^j \in X^j$ for $j = 1, \dots, 6$ are the instantiated versions of $x_i \in X$. Observe that $(\alpha_{x_i}, \beta_{x_i}) = (0, 0), (0, 1), (1, 0)$, and $(1, 1)$ indicate that $x_i \in X_C, x_i \in X_G, x_i \in X_H$, and x_i can be in either of X_G and X_H , respectively.

In SAT solving the conjunction of Formulas (4) and (5), we make *unit assumptions* [6] on the control variables. Similar to [11] but with a subtle difference, we introduce the following *seed variable partition* to avoid trivial variable partition and to avoid $|X_G| = 1$. For the unit assumption, initially we specify three distinct variables with one, say, x_j , in X_H and two, say, x_k, x_l , in X_G , and specify all other variables in X_C . That is, we have $(\alpha_{x_j}, \beta_{x_j}) = (1, 0), (\alpha_{x_k}, \beta_{x_k}) = (0, 1), (\alpha_{x_l}, \beta_{x_l}) = (0, 1)$, and $(\alpha_{x_i}, \beta_{x_i}) = (0, 0)$ for $i \neq j, k, l$.

LEMMA 3. *For an unsatisfiable conjunction of Formulas (4) and (5) under a seed variable partition, the final conflict clause consists of only the control variables, which indicates a valid non-trivial variable partition.*

If the conjunction of Formulas (4) and (5) is unsatisfiable under a seed variable partition, then the corresponding decomposition (indicated by the final conflict clause) is successful. Otherwise, we should try another seed variable partition. For a given function $f(X)$ with $|X| = n$, the existence of non-trivial Ashenhurst decomposition can be checked with at most $3 \cdot C_3^n$ different seed partitions.

Rather than just looking for a valid variable partition, we may further target one that is more balanced (i.e., $|X_H|$ and $|X_G|$ are of similar sizes) and closer to disjoint (i.e., $|X_C|$ is small) by enumerating different seed variable partitions. As SAT solvers usually refer to a small unsatisfiable core, the returned variable partition is desirable because $|X_C|$ tends to be small. Even if a returned unsatisfiable core is unnecessarily large, the corresponding variable partition can be further refined by modifying the unit assumption to reduce the unsatisfiable core and reduce $|X_C|$ as well. The process can be iterated until the unsatisfiable core is minimal.

After automatic variable partition, functions g and h can be derived through a construction similar to the foregoing one. The correctness of the overall construction can be asserted.

THEOREM 3. *For a function f decomposable under Ashenhurst decomposition, we have $f(X) = h(X_H, X_C, g(X_G, X_C))$ for functions g and h , and a non-trivial variable partition $X = \{X_H|X_G|X_C\}$ derived from the above construction.*

3.2 Multiple-Output Ashenhurst Decomposition

So far we considered single-output Ashenhurst decomposition for a single function f . We show that the algorithm is extendable to multiple-output Ashenhurst decomposition for a set $\{f_1, \dots, f_m\}$ of functions.

Proposition 1 in the context of Ashenhurst decomposition of a set of functions can be formulated as satisfiability solving as follows.

PROPOSITION 3. *A set $\{f_1(X), \dots, f_m(X)\}$ of completely specified Boolean functions can be expressed as*

$$f_i(X) = h_i(X_H, X_C, g(X_G, X_C))$$

for some functions h_i and g with $i = 1, \dots, m$ if and only if the Boolean formula

$$\begin{aligned} & \bigvee_i (f_i(X_H^1, X_G^1, X_C) \neq f_i(X_H^1, X_G^2, X_C)) \wedge \\ & \bigvee_i (f_i(X_H^2, X_G^2, X_C) \neq f_i(X_H^2, X_G^3, X_C)) \wedge \\ & \bigvee_i (f_i(X_H^3, X_G^3, X_C) \neq f_i(X_H^3, X_G^1, X_C)) \end{aligned} \quad (6)$$

is unsatisfiable.

Since the derivation of functions g and h_i , and automatic variable partitioning are essentially the same as the single-output case, we omit the detailed exposition.

3.3 Beyond Ashenhurst Decomposition

Is the above algorithm extendable to general functional decomposition, namely,

$$f(X) = h(X_H, X_C, g_1(X_G, X_C), \dots, g_k(X_G, X_C))$$

for $k > 1$? The answer is yes, but with prohibitive cost. Taking $k = 2$ for example, we need 20 copies of f to assert the non-existence of five different column patterns for every matrix of a decomposition chart, in contrast to the six for Ashenhurst decomposition shown in Figure 2. This number grows in $2^k(2^k + 1)$. Aside from this duplication issue, the derivation of functions g_1, \dots, g_k , and h may involve several iterations of finding satisfying assignments and performing cofactoring. The number of iterations varies depending on how the interpolation is computed and can be exponential in k . Therefore we focus mostly on Ashenhurst decomposition.

4. EXPERIMENTAL RESULTS

The proposed approach to Ashenhurst decomposition was implemented in C++ within the ABC package [2] and used MiniSAT [6] as the underlying solver. All the experiments were conducted on a Linux machine with Xeon 3.4GHz CPU and 6Gb RAM.

Large ISCAS, MCNC and ITC benchmark circuits were chosen to evaluate the proposed method. Only large transition and output functions (with no less than 50 inputs in the transitive fanin cone) were considered. We evaluated both single-output and two-output Ashenhurst decompositions. For the latter, we decomposed simultaneously a pair of functions with similar input variables. For a circuit, we heuristically performed pairwise matching among its transition and output functions for decomposition. Only function pairs with joint input variables no less than 50 were decomposed. Note that the experiments target the study of scalability, rather than comprehensiveness as a synthesis methodology.

Tables 1 and 2 show the decomposition statistics of single-output and two-output decompositions, respectively. In these

Table 1: Single-output Ashenhurst decomposition

circuit	#func	#var	#fail	#SAT_TO	#succ	#var_succ	#VP_avg	rate_valid-VP	time_avg (sec)	mem (Mb)
b14	153	50-218	0	108	45	50-101	1701	0.615	144.22	90.01
b15	370	143-306	0	51	319	143-306	1519	0.917	96.62	107.20
b17	1009	76-308	0	148	861	76-308	1645	0.904	87.12	125.84
C2670	6	78-122	0	1	5	78-122	1066	0.835	83.80	58.91
C5315	20	54-67	0	4	16	54-67	3041	0.914	50.90	51.34
C7552	36	50-194	0	2	34	50-194	1350	0.455	64.38	36.65
s938	1	66-66	0	0	1	66-66	3051	0.726	19.03	24.90
s1423	17	51-59	0	0	17	51-59	3092	0.723	13.66	25.34
s3330	1	87-87	0	0	1	87-87	3336	0.599	58.30	27.75
s9234	13	54-83	0	0	13	54-83	3482	0.857	37.86	35.33
s13207	3	212-212	0	0	3	212-212	569	0.908	70.26	50.62
s38417	256	53-99	6	72	178	53-99	1090	0.523	103.33	136.04
s38584	7	50-147	0	0	7	50-147	1120	0.924	47.13	51.56

Table 2: Two-output Ashenhurst decomposition

circuit	#pair	#var	#fail	#SAT_TO	#succ	#var_succ	#VP_avg	rate_valid-VP	time_avg (sec)	mem (Mb)
b14	123	50-223	18	65	40	50-125	1832	0.568	96.86	226.70
b15	201	145-306	0	31	170	145-269	1176	0.845	113.86	224.07
b17	583	79-310	0	88	495	79-308	676	0.824	103.12	419.35
C2670	5	78-123	0	1	4	78-123	254	0.724	66.95	55.71
C5315	11	56-69	0	2	9	56-69	370	0.594	59.20	60.05
C7552	21	56-195	0	2	19	56-141	188	0.465	89.57	78.67
s938	1	66-66	0	0	1	66-66	3345	0.720	61.24	34.77
s1423	14	50-67	0	0	14	50-67	3539	0.591	55.34	45.66
s3330	1	87-87	0	0	1	87-87	1278	0.423	66.83	47.43
s9234	12	54-83	0	0	12	54-83	2193	0.708	48.11	55.15
s13207	3	212-228	0	0	3	212-228	585	0.700	93.36	118.03
s38417	218	53-116	13	30	175	53-116	689	0.498	109.06	319.48
s38584	9	50-151	0	0	9	50-151	1656	0.713	46.17	207.78

tables, circuits to be decomposed are listed in Column 1. Columns 2 and 3 list the numbers of instances (i.e., functions for single-output decomposition and function pairs for two-output decomposition) with no less than 50 inputs and the ranges of the input sizes of these instances, respectively. Column 4 lists the numbers of instances that we cannot find any successful variable partition within 60 seconds or within 1500 seed variable partitions. Column 5 lists the numbers of instances that are decomposable but spending over 30 seconds in SAT solving for the derivation of function g or h . Columns 6 and 7 list the numbers of successfully decomposed instances and the ranges of the input sizes of these instances, respectively. Columns 8 and 9 list the average numbers of tried seed partitions in 60 seconds and the average rates hitting valid seed partitions. Column 10 shows the average CPU times spending on decomposing an instance. Finally, Column 11 shows the memory consumption. As can be seen, our method can effectively decompose functions or function pairs with up to 300 input variables.

We measure the quality of a variable partition in terms of disjointness, indicated by $|X_C|/|X|$, and balancedness, indicated by $||X_G| - |X_H||/|X|$. The smaller the values are, the better a variable partition is. Figures 4 and 5 depict, for each decomposition instance, the quality of best variable partition found within 60 seconds¹ in terms of the above two metrics, with emphasis on disjointness. A spot on these two figures corresponds to a variable partition for some decomposition instance. Figure 4 and Figure 5 show the variable partition data *without* and *with* further minimal unsatisfiable (UNSAT) core refinement², respectively. Since a final conflict clause returned by a SAT solver may not reflect a minimal UNSAT core, very likely we can further refine the corresponding variable partition. Suppose the variable par-

¹The search for a best variable partition may quit before 60 seconds if both disjointness and balancedness cannot be improved in consecutive 1500 trials.

²For every decomposition instance, the UNSAT core refinement is applied only once to the best found variable partition. The CPU times listed in Tables 1 and 2 include those spent on such refinements.

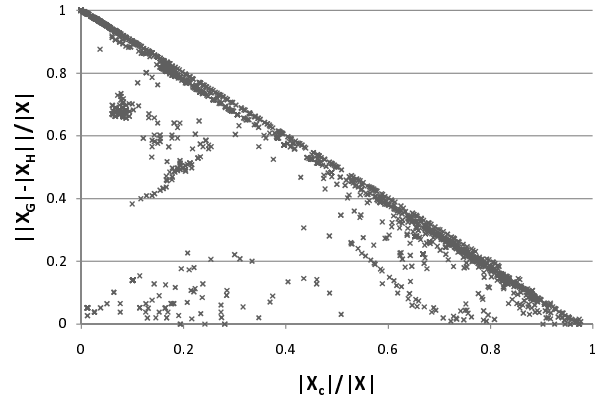


Figure 4: Best variable partition found in 60 seconds – without minimal UNSAT core refinement

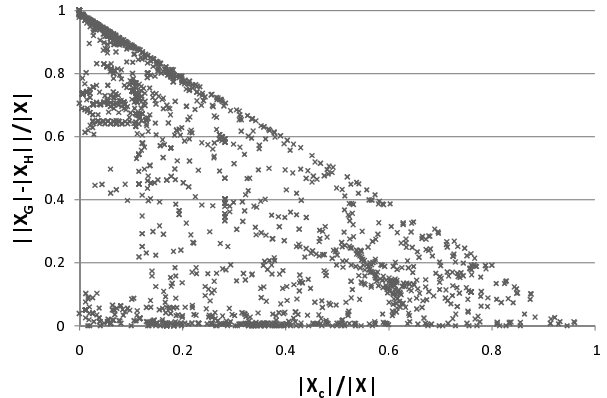


Figure 5: Best variable partition found in 60 seconds – with minimal UNSAT core refinement

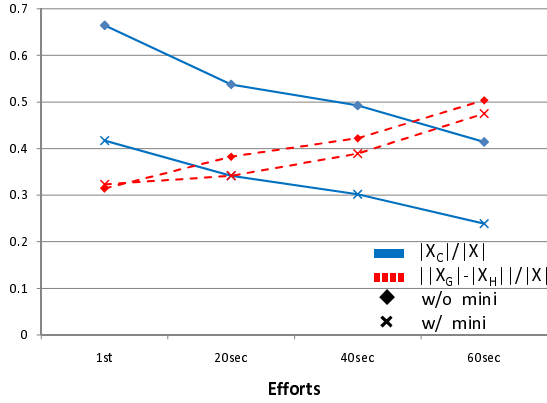


Figure 6: Variable partition qualities under four different efforts

tion is $X = \{X_H|X_G|X_C\}$ before the refinement. We iteratively and greedily try to move a common variable of X_C into X_G or X_H , if available, making the new partition more balanced as well. The iteration continues until no such movement is possible. On the other hand, for a variable x with control variables $(\alpha_x, \beta_x) = (1, 1)$, indicating x can be placed in either of X_H and X_G , we put it in the one such that the final partition is more balanced. Comparing Figures 4 and 5, we see that minimal UNSAT core refinement indeed can substantially improve the variable partition quality. Specifically, the improvement is 42.37% for disjointness and 5.74% for balancedness.

Figure 6 compares the qualities of variable partitioning under four different efforts. In the figure, “1st” denotes the first-found valid partition and “tsec” denotes the best-found valid partition in t seconds. The averaged values of $|X_C|/|X|$ and $||X_G| - |X_H||/|X|$ with and without minimal UNSAT core refinement are plotted. In our experiments, improving disjointness is preferable to improving balancedness. These two objectives, as can be seen, are usually mutually exclusive. Disjointness can be improved at the expense of sacrificing balancedness, and vice versa. The figure reveals as well the effectiveness of the minimal UNSAT core refinement in improving disjointness. It is interesting to note that, on average, 1337 seed partitions are tried in 60 seconds, in contrast to 3 seed partitions tried to identify the first valid one.

Practical experience suggests that the AIG sizes and levels of the composition functions g and h are typically much larger than those of the original function f by an order of magnitude, despite the reduction of support variables. How to minimize interpolants effectively becomes an important subject for our method to directly benefit logic synthesis.

5. PRIOR WORK

Aside from BDD-based functional decomposition, we compare some related work using SAT. In bi-decomposition [11], a function f is written as $f(X) = h(g_1(X_A, X_C), g_2(X_B, X_C))$ under variable partition $X = \{X_A|X_B|X_C\}$, where function h is known *a priori* and is of special function types (namely, two-input OR, AND, and XOR gates) while functions g_1 and g_2 are the unknown to be computed. In contrast, the complication of Ashenhurst decomposition $f(X) = h(X_H, X_C, g(X_G, X_C))$ comes from the fact that both functions h and g are unknown. The problem needs to be formulated and solved differently while the basic technique used is similar to that in [11].

FPGA Boolean matching, see, e.g., [3], is a subject closely related to functional decomposition. In [12], Boolean match-

ing was achieved with SAT solving, where quantified Boolean formulas were converted into CNF formulas. The intrinsic exponential explosion in formula sizes limits the scalability of the approach. Our method may provide a partial solution to this problem, at least for some special PLB configurations.

6. CONCLUSIONS AND FUTURE WORK

A new formulation of Ashenhurst decomposition was proposed based on SAT solving, Craig interpolation, and functional dependency. Traditionally difficult non-disjoint and multiple-output decompositions can be handled naturally. Moreover variable partition needs not be specified *a priori* and can be embedded into the decomposition process. It allows effective enumeration over a wide range of partition choices, which is not possible before. Although Ashenhurst decomposition is a special case of functional decomposition, its simplicity is particularly attractive and preferable.

Because of its scalability to large designs as justified by experimental results, our approach can be applied at a top level of hierarchical decomposition in logic synthesis, which may provide a global view on optimization. It can be a step forward towards topologically constrained logic synthesis.

For future work, how to perform general functional decomposition and how to minimize interpolants await future investigation. Also the application of our approach to FPGA Boolean matching can be an interesting subject to explore.

Acknowledgment

This work was supported in part by NSC grants 95-2218-E-002-064-MY3 and 96-2221-E-002-278-MY3.

REFERENCES

- [1] R. L. Ashenhurst. The decomposition of switching functions. *Computation Lab*, Harvard University, Vol. 29, pp.74-116, 1959.
- [2] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification*. <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [3] J. Cong and Y.-Y. Hwang. Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 20(9):1077-1090, 2001.
- [4] W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, 22(3):250-268, 1957.
- [5] A. Curtis. *New Approach to the Design of Switching Circuits*. Van Nostrand, Princeton, NJ, 1962.
- [6] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, pp.502-518, 2003.
- [7] J.-H. R. Jiang and R. K. Brayton. Functional dependency for verification reduction. In *Proc. CAV*, pp.268-280, 2004.
- [8] J.-H. R. Jiang, J.-Y. Jou, and J.-D. Huang. Compatible class encoding in hyper-function decomposition for FPGA synthesis. In *Proc. DAC*, pp.712-717, 1998.
- [9] R. M. Karp. Functional decomposition and switching circuit design. *J. Soc. Ind. Appl. Math.* 11(2):291V335, 1963.
- [10] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In *Proc. ICCAD*, pp.227-233, 2007.
- [11] R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In *Proc. DAC*, 2008.
- [12] A. Ling, D. Singh, and S. Brown. FPGA technology mapping: A study of optimality. In *Proc. DAC*, pp.427-432, 2005.
- [13] K. L. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, pp.1-13, 2003.
- [14] C. Scholl. *Functional Decomposition with Applications to FPGA Synthesis*. Kluwer Academic Publishers, 2001.
- [15] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pp.466-483, 1970.
- [16] B. Wurth, U. Schlichtmann, K. Eckl, and K. Antreich. Functional multiple-output decomposition with application to technology mapping for lookup table-based FPGAs. *ACM Trans. on Design Automation of Electronic Systems*, 4(3):313-350, 1999.