Chapter 3
Sequential Logic Synthesis Using Symbolic Bi-decomposition

Victor N. Kravets and Alan Mishchenko

Abstract In this chapter we use under-approximation of unreachable states of a design to derive incomplete specification of combinational logic. The resulting incompletely specified functions are decomposed to enhance the quality of technology-dependent synthesis. The decomposition choices are computed implicitly using novel formulation of symbolic bi-decomposition that is applied recursively to decompose logic in terms of simple primitives. The ability of binary decision diagrams to represent compactly certain exponentially large combinatorial sets helps us to implicitly enumerate and explore variety of decomposition choices improving quality of synthesized circuits. Benefits of the symbolic technique are demonstrated in sequential synthesis of publicly available benchmarks as well as on the realistic industrial designs.

3.1 Introduction and Motivation

Due to recent advances in verification technology [2] circuit synthesis of semiconductor designs no longer has to be limited to logic optimization of combinational blocks. Nowadays logic transformations may involve memory elements which change design’s state encodings or its reachable state space and still be verified against its original description. In this chapter we focus on a more conservative synthesis approach that changes sequential behavior of a design only in unreachable states, leaving its intended “reachable” behavior unchanged. Unreachable states are used to extract incomplete specification of combinational blocks and are applied as don’t cares during functional decomposition to improve circuit quality.

To implement combinational logic of a design we rely on a very simple, yet complete, form of functional decomposition commonly referred to as bi-decomposition.

V.N. Kravets (✉)
IBM TJ Watson Research Center, Yorktown, NY
e-mail: kravets@us.ibm.com

In general, for a given completely specified Boolean function its bi-decomposition has form

\[ f(\mathbf{x}) = h(g_1(\mathbf{x}_1), g_2(\mathbf{x}_2)) \]

where \( h \) is an arbitrary 2-input Boolean function. This decomposition is not unique and its quality varies depending on selected subsets \( \mathbf{x}_1 \) and \( \mathbf{x}_2 \) that form possibly overlapping (i.e., non-disjoint) partition of \( \mathbf{x} \). The problem of finding good bi-decomposition has been studied in [1, 10, 18, 19, 21]. The main contribution of the material in this chapter is symbolic formulation of bi-decomposition for incompletely specified functions. The bi-decomposition is used as main computational step in the prototype sequential synthesis tool and is applied recursively to implement logic of combinational blocks whose incomplete specification is extracted from unreachable states of a design. Our symbolic formulation of bi-decomposition finds all feasible solutions and picks the best ones, without explicit enumeration.

Computation of variable partitions in our symbolic formulation of bi-decomposition favors implicit enumeration of decomposition subsets. They are represented compactly with a binary decision diagram (BDD) [4] and are selected based on optimization objective. Unlike previous approaches (e.g., [1, 23]) that rely on BDDs, the decomposition is not checked explicitly for a variable partition and is solved implicitly for all feasible partitions simultaneously utilizing fundamental property of BDDs to share partial computations across subproblems. Thus, no costly enumeration that requires separate and independent decomposability checks is needed. The technique was also used to tune greedy bi-decomposition when handling larger functions.

To overcome limitations of explicit techniques authors in [14] proposed solution that uses a satisfiability solver [11]. Their approach is based on proving that a problem instance is unsatisfied. The unsatisfiable core is then used to greedily select partition of variables that induces bi-decomposition. Authors demonstrate the approach to be efficient in runtime, when determining existence of non-trivial decomposition. The experimental results on a selected benchmark set, however, are primarily focused on the existence of decomposition and do not offer a qualitative synthesis data.

The problem of using unreachable states of a design to improve synthesis and verification quality has been studied before in various contexts. In general, these algorithms either avoid explicit computation of unreachable states or first compute them in pre-optimization stage. Approaches that do not explicitly compute unreachable states are mostly limited to incremental structural changes of a circuit and rely on ATPG environment or induction [5, 8, 12] to justify a change. In contrast, approaches that pre-compute subsets of unreachable states treat them as external don’t cares [20] for re-synthesis of combinational logic blocks [6, 15]. In this chapter we adopt the later approach as it offers more flexibility in logic re-implementation through functional decomposition.

This chapter has the following structure. After brief introduction and motivation preliminary constructs are given in Section 3.2. Section 3.3 describes