Boolean Decomposition Revisited

Alan Mishchenko **Robert Brayton** **Alessandro Tempia Calvino** **Giovanni De Micheli**

Department of EECS, UC Berkeley
{alanmi, brayton}@berkeley.edu

Ecole Polytechnique Federale de Lausanne (EPFL)
{ alessandro.tempiacalvino, giovanni.demicheli}@epfl.ch

Abstract

Ashenhurst-Curtis decomposition (ACD) is well-known and widely used in logic synthesis for logic restructuring to save area and reduce delay, and in technology dependent optimization to overcome structural bias when mapping into LUTs and LUT structures. However, available implementations of ACD suffer from excessive complexity and slow runtime. The paper offers several simplifications that allow for a fast and flexible implementation of ACD using truth tables for functions up to 16 inputs. A practical extension allows for an efficient use of ACD in delay-driven mapping, which can enhance state-of-the-art LUT mappers.

1 Introduction

Ashenhurst-Curtis decomposition (ACD) [1][6] was introduced more than 60 years ago and has found applications in logic synthesis and technology mapping, for example, in functional decomposition [10], decomposing multi-valued relations [17], and encoding of multi-valued networks [8].

More recently, ACD is used in ABC [2] for mapping into lookup tables (LUT) structures [18] (command *if* –S <*NN*>) which tends to mitigate structural bias and improve the quality of standard LUT mapping [13], even if dedicated hardware to implement LUT structures is not used. ACD is also used in post-mapping resynthesis [14] (command *lutpack*), when logic cones composed of several LUTs are collapsed into single-output Boolean functions and re-expressed using fewer LUTs by applying ACD. Another practical application of ACD is mapping into a LUT cascade structure [12] (command *cascade*), which realizes multi-output combinational logic using memory blocks.

These applications rely on the traditional formulation of ACD [1][6] breaking the input variables into two groups: the bound set (BS) and the free set (FS). Figure 1 shows an ACD of an 8-variable function with a 6-variable BS and a 2-variable FS, resulting in four 6-input LUTs (LUT6s).

Several known approaches to ACD [10][20][22] allow for the shared set (SS) when one or more LUTs in terms of the BS variables are single-variable functions (buffers), as shown in Figure 2 for nodes L3 and L4 from Figure 1. The larger the SS size, the fewer non-trivial LUTs are required. Maximizing the SS in [10] is implemented using binary decision diagrams (BDDs) [4], making it not applicable when truth tables are used.

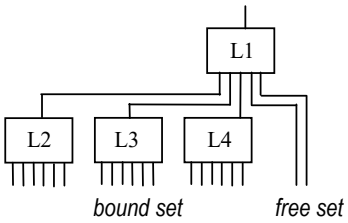

Figure 1: Decomposition with a bound set and a free set.

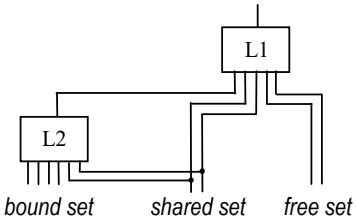

Figure 2: Decomposition with a shared set.

In this paper, we revisit the known formulation of ACD with the SS [12] aiming at making it computationally efficient in LUT mappers and post-mapping resynthesis engines, which manipulate local functions using truth tables.

First, we discuss computing high-quality (minimal-LUT) decomposition with a SS. Our procedure enumerates BS's of a function and computes maximal feasible SS's for each BS, resulting in minimal-size LUT implementations of a function. This computation decomposes 10K 8-input functions optimally in 1 sec, which is 3.5 times faster than existing truth-table-based implementations in [14] and [18].

Second, we show how to compute ACD with support-limited BS functions, that is, BS functions whose support is smaller than the BS size. For example, a 10-variable function decomposed using two FS variables and three 8-variable BS functions may be realizable with three 6-variable BS functions after support minimization. This decomposition is better since it requires four LUT6s spanning two levels (Figure 1), unlike the naïve ACD with three 8-variable BS functions, requiring at least seven LUT6s spanning three levels (Figure 3).

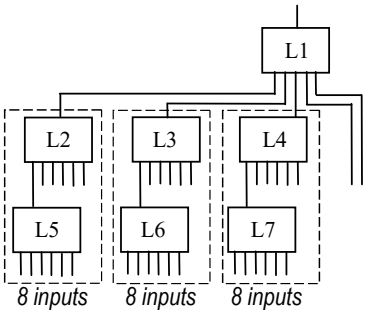

Figure 3: Decomposition with an 8-variable bound set resulting in a network composed of 7 LUT6s.

Third, we consider the decomposition of multi-output Boolean functions modified to derive BS functions used for several outputs, as shown in Figure 4. Our implementation enumerates feasible BS functions of each output and checks the possibility of sharing them with other outputs using truth tables, unlike previous work on multi-output ACD for LUT-mapping [22], which uses BDDs.

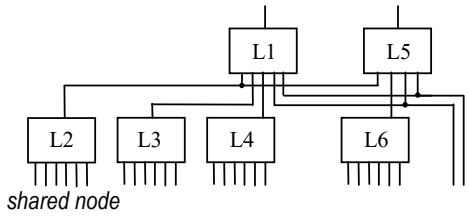

Figure 4: Decomposition with a shared node.

Finally, we show that ACD can be used for delay-driven restructuring of the LUT network. The idea is to keep a subset of early arriving variables in the BS while putting the remaining variables, including the late arriving ones, into the FS. The past work [14][18] does not emphasize delay optimization because the approach used is not versatile.

The special case of ACD, *disjoint-support decomposition* [1][3], is not considered in this paper focusing on a more general non-disjoint decomposition. Another related type of ACD, called *support-reducing decomposition* [9], has been extensively researched in the context of standard-cell mapping, while the present paper focuses on LUT mapping, also known as mapping for FPGAs.

The rest of the paper is organized as follows. Section 2 introduces the background. Section 3 and Section 4 discuss ACD with a maximal SS and with minimization of the support of BS functions, respectively. Section 5 presents ACD for multi-output functions. Section 6 discusses the use of ACD for timing optimization. Section 7 gives experimental results. Section 8 draws conclusions and outlines future work.

2 Definitions

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. We use the terms Boolean network, logic network and circuit, interchangeably. A *K-LUT network* is a Boolean network composed of *K*-input lookup tables (*K*-LUTs). A *K*-LUT is a node capable of realizing any *K*-input Boolean function.

A node *n* has zero or more *fanins,* i.e. nodes that are driving *n*, and zero or more *fanouts*, i.e. nodes driven by *n*. The *primary inputs* (PIs) are nodes of the network without fanins. The *primary outputs* (POs) are a specified subset of nodes of the network, which deliver functions implemented in the network to exterior circuitry.

A completely-specified Boolean function *F essentially depends* on a variable if there exists an input combination such that the value of the function changes when the variable is toggled. The *support* of *F* is the set of all variables on which function *F* essentially depends. The supports of two functions are *disjoint* if they do not contain common variables. A set of functions is *disjoint* if their supports are pair-wise disjoint.

A *decomposition* of a completely specified Boolean function is a Boolean network with one primary output that is functionally equivalent to the given function.

Ashenhurst-Curtis decomposition (ACD) of *s* single-output Boolean function *F* can be expressed as follows:

$$F(x_{bs}, x_{ss}, x_{fs}) = G(\, H(x_{bs}, x_{ss}), x_{ss}, x_{fs}\,),$$

where the bound set (x_{bs}), the free set (x_{ss}), and the free set (x_{fs}) are disjoints variable subsets, which together form the support of *F*. The BS function *H* may be multi-output with the number of outputs less than the BS size, while the support of composition function *G* is typically chosen to fit into one *K*-input LUT.

3 Maximizing the shared set

This section discusses, a truth-table-based implementation of ACD using a shared set, for a single-output function.

The computation is iterated over bound sets of a given size. The size is determined based on the support size of the target function and the given LUT size. For example, for an *N*-input function using *K*-input LUTs, it is convenient to consider a *K*-variable BS and a (*N*-*K*)-variable FS. When *N*=8 and *K*=6, there are 8*7/2 = 28 different 6-variable BS's. When support minimization of the BS function is done, as shown in Section 4, it is helpful to consider (*K*+*A*)-variable BS and (*N*-*K*-*A*)-variable FS where $A \in \{1, 2, 3\}$.

For each BS, the truth table is transformed to have the BS variables to be more significant ones, compared to the FS variables. In this case, if the BDD of the function is available, the BS variables are ordered first (i.e. above the FS variables) after the transformation. When truth tables are

used, the variable reordering is performed using a dedicated procedure, which swaps two variables. Note that the first BS composed of K most significant variables in the support of the function does not need variable swapping, because the original truth table already reflects this order. Every consecutive BS can be derived from a previous BS by one two-variable swap.

Each assignment of the BS variables selects one (N-K)-input function in terms of the FS variables. In the transformed truth table, these functions are listed next to each other. The decomposition procedure accesses them one at a time by extracting 2^{N-K} bits at a given offset in the truth table, determined by the values of the BS variables.

Example. Consider the 5-variable function of the Booth partial product whose 32-bit truth table represented in hexadecimal is 0xF335ACC0. Assume the BS is the three most significant variables and the FS is the two least significant variables. The functions in terms of FS variables have truth tables with $2^{N-K} = 2^{5-3} = 2^2 = 4$ bits. There are $2^K = 2^3 = 8$ of them, corresponding to hexadecimal digits in the truth table of the original function (0xF, 0x3, 0x3, etc).

The target function can be realized using M BS functions if the number of different FS functions for the given BS, known as *column multiplicity* (μ), does not exceed 2^M. Much of the previous work on ACD tries to minimize M by selecting $M = \lceil \log2(\mu) \rceil$. However, M does not have to be minimized because, for practical functions, some of the M functions can be buffers, leading to a shared set (Figure 2).

Example. Continuing the above example, there are 8 FS functions but only 6 of them are different (because functions with truth tables 0x3 and 0xC are duplicated). Thus, the column multiplicity is 6 and we need at least $\lceil \log2(6) \rceil = 3$ BS functions to implement the decomposition. This decomposition is not support-reducing (using the terminology of [9]) because the top-most block depends on the same number of variables (five) as the original function, but it may still be useful if the top-most function is simpler.

Ideally, all BS functions are buffers, resulting in only one non-trivial K-input BS function and M-1 variables in the SS.

To check whether a decomposition with L ($0 < L < M$) single-variable functions (or buffers) and M-L non-buffer BS functions exists, the proposed method enumerates subsets of L out of K variables in each BS. For each subset, the method checks if the number of unique FS functions in each cofactor w.r.t. the L variables does not exceed 2^{M-L}. If this is the case, a decomposition with L buffers exists.

This check is implemented for all ($S = K!/L!/(K-L)!$) subsets of L out of K variables using the truth table transformed to have the BS variables as the most significant ones. For this, we extract one FS function at a time and add it to S dynamic arrays accumulating unique cofactors for each subset S of BS variables. If, for any subset, the number of unique cofactors exceeds 2^{M-L}, we skip this subset; if the number exceeds 2^{M-L} for all subsets, we skip this BS.

This computation enumerates all BS's and all subsets of each BS efficiently, resulting in a decomposition with the largest possible *SS* size. If timing information is present and the function to be decomposed is located on the critical path of the design, the enumeration of BS's can be limited to include only early arriving variables, as shown below in Section 5 on delay-driven mapping.

4 Support minimization of BS functions

Section 3 presented ACD for a single-output function with minimization of the SS size. If all but one BS functions depend on a single BS variable, the SS size is maximized. In this case, there is only one non-trivial BS function depending on all BS variables. However, if there are two or more non-trivial BS functions, they often can be derived in such a way that they do not depend on all BS variables.

The support minimization of the BS functions is particularly useful when the BS size is larger than the LUT size. One such example was given in the Introduction: instead of three 8-input BS functions (Figure 3), we may be able to use three 6-input BS functions (Figure 1), if the target function allows for this. This leads to a reduction in the size and depth of the LUT network after mapping.

The proposed algorithm for support minimization is similar to solving a constrained encoding problem [11][21].

To formulate a condition when the BS functions are independent of some of the BS variables, we recall that the FS functions represent the cofactors of the target function w.r.t. the BS variables. Thus, if each BS function does not depend on some BS variables, other BS functions should be able to distinguish the corresponding FS functions if they are different from each other.

Deriving the BS functions satisfying this condition can be done by solving a covering problem, in which columns are subsets of BS variables and rows are pairs of different FS functions to be distinguished. A *one* in the covering table indicates that a BS function does not have to depend on the given variable in order to distinguish the FS function pairs. A solution to the covering problem is a set of variable subsets removable, each of which can be removed from a BS function. Knowing the number of non-trivial BS functions and their target support size, we synthesize them to distinguish all different FS function pairs.

5 Logic sharing across multiple outputs

Logic sharing across multiple outputs can be extracted using a method, similar to the one used in Section 3, to check for the existence of single-variable BS functions.

For this, truth tables of those outputs whose logic sharing is being extracted, are transformed to have variables, belonging to the given BS, as the most significant ones.

Next, one or more decompositions of each of the outputs are derived and checked if the resulting BS function(s) can be used to decompose other outputs. This is done by a method similar to that for maximizing the SS.

Indeed, an SS variable represents a single-variable BS function. To prove that it can be used, we check if it partitions the cofactor FS functions into sets having no more than a certain fixed number of unique cofactors. Similarly, a given BS function, derived to decompose one output, can be used as a candidate BS function for other outputs. If a limit on the unique cofactor count of an output is not exceeded, the function can be used to decompose the output.

The proposed method gives possible BS functions for each output, and tests them for decomposing other outputs. In the process, multiple function sharing opportunities may be detected. The one that maximizes the number of shared BS functions across all outputs is chosen. For example, if a function can be shared across three outputs, it is preferred to another that can be shared across two outputs.

6 Delay-driven LUT mapping

Consider a network of *K*-LUTs and a target node with delay *D* that is on a critical path.

This section answers the following question: Is it possible to replace the target node by a new node whose delay is *D*-1 or less (assuming the unit-delay model). We also show how to use ACD to derive the new node and its fanin LUTs so that the resulting LUT subnetwork can replace the target node and produce the desirable delay improvement.

First, a set of *K*-LUTs composed of the target node and its timing-critical fanins are found, resulting in a subnetwork of *K*-LUTs. By construction, the arrival times of any input of the subnetwork is less than *D*-1. Next, an ACD of the Boolean function of the subnetwork is attempted with the FS composed of inputs whose arrival times are less than *D*-1 with the BS composed of the remaining inputs.

If, on the other hand, we are successful in finding a target function using two levels of *K*-LUTs, the delay-optimization problem is solved. In this case, the output node of the new LUT structure derived by ACD has an arrival time less the arrival times of the FS variables, which in this case is *D*-1. When the target node is replaced by the new node, the delay on that critical path is reduced.

The above discussion shows how to perform delay optimization in post-mapping resynthesis of LUT networks. The same method can be applied when evaluating delay-oriented matches in the inner loop of a LUT mapper.

It can be shown that this approach is more general than selective cofactoring w.r.t the late arriving variables [15] (command *speedup* in ABC), because, not only cofactoring w.r.t. these variables, but a more general decomposition of the target function is performed.

7 Experimental results

The proposed improvements to ACD are implemented in ABC and compared against the existing commands. In this section, the results of four experiments are reported.

7.1 Runtime of decomposition with maximal SS

First, we compare the proposed ACD that maximizes SS, against the implementation of the LUT structure mapping [18] in command *if –S <NN> -K <M>* in ABC, where *N* is the LUT size and $N \le M \le 2*N-1$. When mapping benchmarks using 8-input cuts into LUT structures "66", composed of two LUT6, the command is *if –S* 66 *–K* 8.

We collected 10K typical 8-input functions considered by the LUT mapper *if* in this case and decomposed using the proposed implementation. The runtime of the proposed implementation is about 3.5x faster.

7.2 Support-minimization of the BS functions

For the 10K typical 8-input function harvested in the above experiment, we checked the feasibility of minimizing the support of the BS functions. We found that about 80% of these practical functions decomposed with the maximal SS size have BS functions depending on fewer than 6 inputs. Since we use 6-input LUTs, these decompositions do not reduce the LUT count but does reduce the total LUT fanin count, which is beneficial for placement and routing.

We also collected 10K typical 10-input functions and processed them by ACD with 3-variable FS and 7-variable BS without SS. It turned out that 70% of the resulting 7-variable BS functions could be implemented using LUTs with 6 or fewer inputs. Using such decompositions in a LUT mapper is beneficial for both area and delay, because generally each 7-input function takes at least two LUT6's. Thus, in general, the decomposed network for a 10-variable target function spans three levels of LUT6, instead of two levels when the support minimization has been successful.

7.3 Logic sharing across multiple outputs

In a separate experiment, we applied ACD with BS functions shared across the different outputs, to the 8-input 8-output function of the S-box [24], a design primitive extensively used in cryptography. By decomposing each output functions separately using brute-force cofactoring (command *lutmin* [12] in ABC), the S-box takes 40 LUT6. Indeed, each 8-input function can be cofactored w.r.t. any two variables, resulting in four 6-input cofactors and one 4:1 MUX, which is also a 6-input function.

The S-box functions do not have shared cofactors w.r.t. any two inputs. However, the method from Section 5 computed a number of bound sets, for which BS functions could be shared across the output functions, resulting in an S-box implementation composed of 36 LUT6's, which a 10% reduction, compared to the brute-force cofactoring.

The same is true for the 8-input 8-output function of the inverse S-box: it can be decomposed using 36 LUT6's, instead of the 40 LUT6's as in the brute-force cofactoring.

7.4 Delay-driven LUT mapping

As part of future work, we intend to integrate delay-driven optimization described in Section 4 into the LUT mapper *if* in ABC. This will allow us to compare the logic level and the LUT count after mapping with the proposed delay optimization, against other delay-driven mapping options: (1) the use of structural choices [5], (2) high-effort delay optimization of AIGs using "lazy man's synthesis" [23], and (3) mapping into LUT structures "66" and "666" [18].

We expect that the logic level produced by the proposed approach will be better at the cost of some area increase.

8 Conclusions

This paper revisits the well-known and widely used Ashenhurst-Curtis decomposition and offers an up-to-date solution to several related problems: finding the largest set of shared variables, computing support-limited functions, and sharing decomposition functions across outputs.

The paper does not add new theory, compared to the earlier work in this area, but proposes an efficient truth-table based implementation discussioned in Section 3-5. Another benefit of this work, is that it documents the source code developed to solve the above problems in ABC, an open-source logic synthesis system [2].

The methods proposed in the paper can be extended to work with incompletely-specified functions. In this case, the FS functions selected based on the values of the BS variables, are incompletely specified. They can be matched and counted in the presence of don't-cares.

Acknowledgements

This research was supported in part by SRC Contract 3173.001 "Standardizing Boolean transforms to improve quality and runtime of CAD tools", the NSA grant "Novel methods for synthesis and verification in cryptanalytic applications" and donations from AMD, Siemens, and Synopsys.

References

[1] R. L. Ashenhurst. "The decomposition of switching functions". *Technical Report,* Bell Laboratories, 1952, BL-1(11), pp. 541-602.

[2] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. https://github.com/berkeley-abc/abc

[3] V. Bertacco and M. Damiani. "Disjunctive decomposition of logic functions". *Proc. ICCAD '97*, pp. 78-82.

[4] R. E. Bryant. "Graph-based algorithms for Boolean function manipulation". IEEE TC, C-35(8), Aug. 1986, pp. 677-691.

[5] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping", *IEEE Trans. CAD*, Vol. 25(12), December 2006, pp. 2894-2903.

[6] H. A. Curtis. "A generalized tree circuit". *Journal of the ACM*, 1961, Volume 8 (4), pp. 484-496.

[7] C. Files. *A new functional decomposition method as applied to machine learning and VLSI layout.* Ph.D. Thesis. Portland State University, June 2000.

[8] J.-H. Jiang, Y. Jiang, and R. K. Brayton. "An implicit method for multi-valued network encoding". *Proc. IWLS'01*, pp.127-131.

[9] V. Kravets and K. Sakallah. "Constructive library-aware synthesis using symmetries". *Proc. DATE'00*, pp. 208-213.

[10] Ch. Legl, B. Wurth, and K. Eckl. "Computing support-minimal subfunctions during functional decomposition". *IEEE Trans. VLSI*, 6(3), pp. 354-363, Sept. 1998.

[11] G. D. Micheli, R. K. Brayton, and A. L Sangiovanni-Vincentelli. "Optimal state assignment for finite state machines". *IEEE Trans. CAD*, 4-3, pp. 269-285, July 1985.

[12] A. Mishchenko and T. Sasao, "Encoding of Boolean functions and its application to LUT cascade synthesis". *Proc. IWLS '02*, 115-120.

[13] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts". *Proc. ICCAD '07*, pp. 354-361.

[14] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks". *Proc. ICCAD '08*, pp. 38-44.

[15] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices", *Proc. FPGA'10*, pp. 181-184.

[16] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. "Optimum functional decomposition using encoding". *Proc. DAC '94*, pp. 408-414.

[17] M. Perkowski et al. "Decomposition of multiple-valued relations". *Proc. ISMVL '97*, Halifax, Canada, pp.13-18.

[18] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen, "Mapping into LUT structures", Proc. DATE'12, pp. 1579-1584.

[19] T. Sasao, M. Matsuura, and Y. Iguchi. "A cascade realization of multiple-output function for reconfigurable hardware". *Proc. IWLS '01*, pp. 225-230.

[20] T. Sasao. "A new expansion of symmetric functions and their application to non-disjoint functional decompositions for LUT-type FPGAs". *Proc. IWLS'00*, pp. 105-11.

[21] T. Villa et al. "NOVA: state assignment of finite state machines for optimal two-level implementations". *Proc. ITCAD '90*, pp. 905-924.

[22] B. Wurth, U. Schlichtmann, K. Eckl, and K. J. Antreich, "Functional multiple-output decomposition with application to technology mapping for lookup table-based FPGAs", *ACM TODAES*, Vol. 4(3), pp. 313-350.

[23] W. Yang, L. Wang, and A. Mishchenko, "Lazy man's logic synthesis", *Proc. ICCAD'12*, pp. 597-604.

[24] https://en.wikipedia.org/wiki/Rijndael_S-box