Technology Mapping for Robust Asynchronous Threshold Networks

Cheoljoo Jeong Steven M. Nowick*
Department of Computer Science, Columbia University
New York, NY, 10027, USA
Email: {cjeong, nowick}@cs.columbia.edu

Abstract

As process, temperature and voltage variations become significant in deep submicron design, timing closure becomes a critical challenge in synchronous CAD flows. One attractive alternative is to use robust asynchronous circuits which gracefully accommodate timing discrepancies. However, there is currently little CAD support for such robust methodologies. In this paper, an algorithm for technology mapping of robust asynchronous threshold networks is presented. The proposed algorithm is the first to systematically optimize for either delay or area, without destroying the hazard-freedom properties of the initial unoptimized circuits. The algorithm was implemented and experiments were performed on a near-complete industrial DES circuit provided by Theseus Logic, using a particular asynchronous threshold circuit style called NCL (Null Convention Logic), which had been already optimized in an existing constrained asynchronous synthesis flow based on synchronous CAD tools. Average delay improvements up to 26.7% and area improvements up to 4.5% were obtained, when considering the largest subcircuits (with over 400 inputs and outputs). When only the single longest path delay of each subcircuit is considered, the algorithm obtained worst-case delay improvements up to 26.4%. Though the proposed method is applied in the NCL design flow, the contribution is general enough to be used for other robust asynchronous threshold circuit styles.

1 Introduction

As process, temperature and voltage variations become significant in deep submicron design, timing closure becomes a critical challenge in synchronous CAD flows [4]. One attractive alternative is to use robust asynchronous circuits which gracefully accommodate timing discrepancies. Asynchronous design has been the focus of renewed interest and research activity because of the potential benefits of low power consumption, low electromagnetic interference, robustness to parameter variations, and modularity of designs [31]: as an example, it is reported in [20] that an asynchronous re-design of Motorola CPU08 requires 40% less power and genereates 10dB less peak EMI noise than the synchronous version. However, there is currently little CAD support for such robust methodologies. In either synchronous or asynchronous synthesis flows, a technology mapping step is important since it is typically the first step when optimization is performed with realistic cost parameters of the target technology.

In this work, a technology mapping algorithm is introduced for a class of highly-robust asynchronous circuits, based on threshold gates and dual-rail encoded data. The algorithm is the first to systematically optimize for either delay or area, without destroying the hazard-freedom properties of the initial unoptimized circuits. The focus of the method is on a particular style of asynchronous threshold networks, called Null Convention Logic (NCL) [8, 18], which exhibits very low power and robustness to delay variation, and which has been used for a number of industrial designs by Theseus Logic. However, the contribution is general enough to be applied to other classes of asynchronous threshold circuit styles.

In particular, the original NCL unoptimized synthesis flow, introduced by Theseus Logic, makes constrained use of existing synchronous synthesis tools, to ensure a robust asynchronous circuit implementation. Their mapping is highly conservative, since using synchronous tools may lead to undesirable behaviors, which are carefully avoided. A straightforward template-based method (see Section 2.3) is used, where a (synchronous) Boolean netlist is first optimized, and then macro-expanded into corresponding asynchronous cells (and the clock is replaced by handshaking protocols), with limited local optimizations then applied to merge some cells. In contrast, our proposed technology mapping algorithm performs a systematic optimization of the asynchronous netlist, with a global notion of optimality, while still preserving the robustness of the asynchronous netlist.

The contributions of the paper are as follows. First, the proposed technology mapping algorithm is the first systematic approach for asynchronous threshold circuits that preserves the robustness property of the circuits. Unlike earlier asynchronous technology mapping algorithms, it can map across both datapath and control circuits, and can handle substantial implementations. Unlike typical synchronous technology mapping algorithms, it maps a netlist of sequential gates with hysteresis, targeting either area or delay (using a sophisticated library characterization). In addition, a simple finite basis is proposed for threshold networks, which is new as far as the authors are aware of, with two primitive functions; this basis is then extended to a more heterogeneous basis to ensure robustness during mapping algorithm. Finally, some theoretical results are provided on robust decomposition into subject graphs and robust covering of these subject graphs.

Second, the proposed algorithm was implemented in a prototype CAD tool and applied to an industrial DES circuit provided by Theseus Logic, which is near-complete except for the registration and acknowledgment logic. Three of the five subcircuits were substantial, with several hundred inputs and outputs, and over a thousand gates in the original circuit implementation, and all were already pre-optimized using some existing but limited safe cell-merger techniques. Using our new technology mapping algorithm, average delay improvements up to 26.7% and area improvements up to 4.5% were obtained, when considering the largest subcircuits (with over 400 inputs and outputs). When only the single longest path delay of each subcircuit is considered, the algorithm obtained worst-case delay improvements up to 26.4%. Though the proposed method is applied in the NCL design flow in this paper, the contribution is general enough to be adapted to other threshold-style asynchronous design flows.

Related Work. There have been ongoing research efforts in technology mapping of asynchronous control circuits since the early 1990's. For fundamental-mode control circuits, such as "burst-mode" circuits including those synthesized by the *Minimalist* [9] or the *3D* [33] tools, a first approach for hazard-free technology mapping was introduced by Siegel et al. [25]. Kudva et al. [16] subsequently developed a hazard-free method for implementing burst-mode circuits using customized complex CMOS gates. Beerel et al. [2] presented technology mapping algorithm which minimizes the average-case delay of burst-mode circuits, while James and Yun [12] extended this work to transistor-level optimization for generalized C-element implementations.

For quasi delay-insensitive (QDI) control circuits [19], most of the approaches focused on decomposing hazard-free complex gates into networks of standard gates or custom gates like generalized C-elements, while still preserving the QDI property. Beerel [1] introduced decomposition and technology mapping into the standard-C architecture, which was then extended by Siegel and De Micheli [24]. Burns [3] presented general conditions and algorithms for robust decomposition of sequential elements using arbitrary (including sequential) functions but the algorithm had no notion of optimality. Cortadella et al. [6, 5, 14] presented a substantial set of decomposition and technology mapping techniques for speed-independent control circuits synthesized using *Petrify* [6]. Recently, Ho et al. [11] introduced a technology mapping algorithm for QDI circuits into LUT-based FPGAs.

There has been little previous research on technology mapping for threshold networks [8, 18]. This problem is fundamentally different from the mapping problems for burst-mode or speed-independent circuits, which have typically focused on individual (and usually small)

*This work was partially supported by a subcontract to the DARPA CLASS program (under contract to Boeing) and NSF ITR Award No. NSF-CCR-0086036.

controllers: a threshold netlist typically consists of sequential threshold gates with hysteresis, where datapath is encoded with delay-insensitive codes, and both datapath and control are treated uniformly. In addition, the robustness property which must be preserved — 'gate-orphan-freedom' — is different than the usual QDI property, since it only requires that, at fanout points, path delays are always larger than wire delays (see Section 2.4). Smith et al. [26] introduced various techniques for optimizing asynchronous threshold circuits, which are mostly local (peephole) optimization techniques.

Organization. The remainder of this paper is organized as follows. Section 2 presents some background material and Section 3 presents examples to give a motivation of the ideas used in the algorithm. Section 4 provides theorems that are used in the algorithm. Our algorithm is described in Section 5 and we present our experimental results in Section 6. Finally, Section 7 presents conclusions and future work.

2 Background

2.1 Boolean logic network

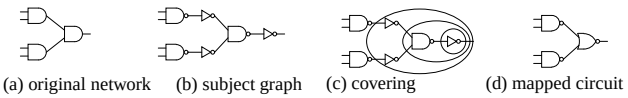
Let $\mathfrak{B} = \{0, 1\}$. A *Boolean function* f with n inputs and m outputs is defined as a mapping $f : \mathfrak{B}^n \rightarrow \mathfrak{B}^m$. A *logic network* is a directed acyclic graph, $G = (V, E)$, with V partitioned into three subsets called *primary inputs*, *primary outputs*, and *internal vertices*. Such a network is a common model used by logic synthesis and mapping algorithms. A Boolean function is associated with each internal vertex in the logic network, and there is a set of assignments of primary outputs to internal vertices denoting which variables are directly observable from outside of the network (see [7]).

2.2 Technology mapping

Technology mapping is the task of transforming an unbound (i.e. technology-independent) logic network into a bound network, i.e. into an interconnection of components that are instances of elements of a given technology library. Traditional tree-based technology mapping algorithms consists of three steps: *decomposition*, *partitioning*, and *covering* [13, 7].

In the decomposition step, the given logic network is decomposed into an equivalent network, where each node now corresponds to a *base function.* Base functions are used to facilitate the matching and covering step and ensures that at least one legal cover can be found. The original logic network given in Figure 1(a) is decomposed into a subject graph as given in Figure 1(b) using two-input NANDs and inverters as base functions.

In partitioning, the decomposed logic network is partitioned into subnetworks, called *subject graphs*. Each subject graph will be mapped separately. Typically, multi-fanout points are used as partition boundaries [13, 7]. In the example of Figure 1, there are no fanout points and the circuit in Figure 1(b) itself is considered to be a subject graph, and the result will be a so-called "leaf-DAG", which is almost a tree, but where fanouts occur only at the primary inputs [7]. More advanced approaches also allow systematic mapping across fanout points, using DAG-based covering [17]. While partitioning is a heuristic, it is widely-used in existing synchronous technology mapping algorithms, to help reduce the complexity of the mapping problem. In this paper, partitioning is performed at multi-fanout points, since DAG-based covering may be unsafe when mapping NCL circuits (see Section 3).



(a) original network (b) subject graph (c) covering (d) mapped circuit

Figure 1. A technology mapping example

Finally, in the covering step, an optimal cover for each subject graph is found by choosing a set of library cells preserving functional correctness. The covering of a small region of the subject graph by a single library cell is typically treated as a subroutine of the covering step, and is called the *match* step. For an efficient implementation of the covering step, each library cell is typically decomposed into a set of pattern graphs using the same base functions which were used in the decomposition step for the subject graph. An optimal cover can be computed by dynamic programming by traversing the subject graph in a bottom-up fashion. In a single pass, the results of the mapped subject graph already traversed are optimal, and these optimal partial results are directly used to compute the next optimal result in the traversal. At each vertex of a subject graph, we match library cells whose pattern graphs are functionally equivalent to the subtree locally rooted at the given vertex, and pick the library cell which gives the minimum cost over all possible matchings [7].

As an example, in Figure 1(c), the rightmost inverter of the subject graph is matched by pattern graphs of three library cells: inverter, two-input AND, and two-input NOR. Given that two-input NOR was chosen to be the best match, the final mapped circuit is shown in Figure 1(d).

A *trivial mapping* of a subject graph consists of the direct mapping of each of its vertices to library cells, where each single vertex is matched separately to its corresponding cell.

In this paper, unless otherwise stated, we refer to tree-based technology mapping algorithms based structural matching by technology mapping algorithms, following the basic flow presented in [13, 7].

2.3 NCL logic

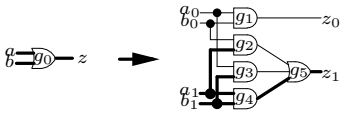
NCL is a circuit implementation style for asynchronous threshold networks [8, 18]. It is based on a delay-insensitive encoding of the datapath, and assumes a two-phase discipline in which data communication alternates between *set* and *reset* phases. Data changes from spacer (called NULL) to proper codeword (called DATA) in the set phase and then back to NULL in the reset phase [21, 23].

3NCL circuits 3NCL is a three-valued logic with $\{0, 1, N\}$. This representation allows a single bit of data to be captured with a single symbolic variable or wire. Of the three values, 0 and 1 represent valid DATA and N represents NULL. A 3NCL gate alternates between two phases. Initially, the input wires and the output wire of a 3NCL gate are initialized to N. When all the inputs have valid DATA value (0 or 1), the output finally changes monotonically to a correct DATA value. For example , the output of a 3NCL OR gate changes to a DATA value only after *all* the inputs have changed to DATA value (0 or 1). This property is called "input completeness", because it guarantees the robust acknowledgment of the input data arrival. Next, in the reset phase, the output maintains the DATA value until all the inputs are reset to N. When all the inputs change to N, the output changes to N, completing the robust reset phase.

2NCL circuits A 3NCL circuit built using 3NCL gates is theoretically delay-insensitive, but eventually this circuit should be implemented using binary-valued Boolean circuits. NCL logic implements a single 3NCL gate using the DIMS-style dual-rail expansion [28], where each single variable (or bit) is mapped to a dual-rail Boolean equivalent. The resulting circuit is robust, as discussed later in this section.

Figure 2 shows an example of how a 3NCL gate is dual-rail expanded into a network of 2NCL gates, which will be called a *dual-rail block*. In the example, a 3NCL two-input OR gate, with inputs a and b, and one output z, is transformed into a network with four inputs, a_0, a_1, b_0, b_1, and two outputs, z_0, z_1. Here, the wires a_0, b_0, z_0 represent the 0-rails of a, b, z and the wries a_1, b_1, z_1 represent the 1-rails of a, b, z. Four 2NCL AND gates, which are C-elements, are used to distinguish each of the four unique input combinations of a and b, and one or zero 2NCL OR gate is used for each of the output rails.

In the Figure, the thick lines indicate the signal transitions under the input combination $a = 1, b = 1$ in 3NCL logic and, correspondingly, $a_0 = 0, a_1 = 1, b_0 = 0, b_1 = 1$ in 2NCL logic.



(a) a 3NCL OR gate (b) 2NCL dual-rail expansion

Figure 2. 2NCL logic example

To transform 3NCL inverters into 2NCL logic, connecting input 1-rail and output 0-rail and connecting input 0-rail and output 1-rail in 2NCL expansion achieves inversion. More details of NCL gates will be discussed below. As a result, 2NCL circuits are inherently monotonic and do not have any inversion, ensuring hazard-freedom in each set phase. Similarly, and symmetrically, since C-elements are used to implement the 2NCL AND functions, the reset is also monotonic and hazard-free.

To obtain a 2NCL circuit from a 3NCL circuit, each gate of the 3NCL circuit is visited in topological order in the circuit, from primary inputs to primary outputs, and is in turn expanded to a corresponding network of 2NCL gates.

NCL threshold gates with hysteresis A 2NCL circuit is eventually mapped using *NCL threshold gates with hysteresis*, which are defined in the NCL technology library. An NCL threshold gate with hysteresis [27] is a gate whose set function and reset function are not combinational, but rather are sequential. Once the gate is set, the output does not change until the reset condition occurs, and once it is reset, the output does not change until the set condition occurs. More formally, the set function of a NCL threshold gate with n inputs, $x_1, \ldots, x_n$, implements a threshold function S and the reset function is always $R = \overline{x_1 + \ldots + x_n}$. A threshold function $S(x_1, \ldots, x_n; w_1, \ldots, w_n; T)$ is characterized by a weight vector $\vec{w} = (w_1, \ldots, w_n)$ and a threshold value $T \in \mathcal{R}$, where $S(x_1, \ldots, x_n; \vec{w}; T) = 1$ iff $\sum_{1 \le i \le n} w_i \cdot x_i \ge T$. In NCL threshold gates, threshold values and weight functions are restricted to be positive integers. Positive integral threshold functions are Boolean functions which are positive unate in all of its variables. NCL threshold gates can be considered as a form of generalized C-elements [3].

As an example, a two-input C-element, with inputs x_1 and x_2, has a set function $S(x_1, x_2; 1, 1; 2)$, indicating that each input has weight 1, and the threshold is 2. For this example, the reset function is $R = \overline{x_1 + x_2}$, indicating that both inputs must be reset before the output can be reset. The current NCL library defines 25 different types of threshold gates as well as a few unate (but non-threshold) gates, with up to four gate inputs.

Even though 2NCL gates have a hysteresis property, in many cases, it is often possible for optimization algorithms to only consider the set functions explicitly, and reset functions will be correctly implemented by default by the mapped gates. The reset functionality, or hysteresis behavior, can be restored later since all the NCL gates have identical reset function [18]. Therefore, this paper will mostly refer to 2NCL gates based solely on their set functionality unless otherwise stated. For example, a 2NCL "AND" gate will refer to an NCL sequential threshold gate whose set function is $S(x_1, x_2, 1, 1; 2) = x_1 \cdot x_2$ and whose reset function is $R = \overline{x_1 + x_2}$. Thus, 2NCL AND gates are essentially the same as C-elements. Similarly, a 2NCL "OR" gate means an NCL threshold gate whose set function is $S(x_1, x_2, 1, 1; 1) = x_1 + x_2$ and whose reset function is $R = \overline{x_1 + x_2}$.

NCL design flow The algorithm proposed in the paper is general for asynchronous threshold circuits; however, they have been implemented to fit into Theseus Logic's existing NCL synthesis flow. Hence, this subsection briefly reviews their current tool flow.

The NCL design flow starts by specifying the circuit in a 3NCL logic style in a VHDL program. Effectively, the netlist appears similar to a standard unoptimized Boolean netlist (but with some extended enumerated data types). By only considering the set functions of the 3NCL gates in the netlist, existing synchronous optimization tools can be applied. In the current flow, Synopsys Design Compiler is used. The result is an optimized netlist of 3NCL gates.

Next, the optimized 3NCL logic is conceptually transformed into 2NCL logic style, where each 3NCL gate is expanded into a dual-rail block. (In practice, this step is merged with the following step.)

After dual-rail expansion, the logic is mapped to a pre-defined library of NCL threshold gates. In practice, a single dual-rail block of the 2NCL circuit is usually mapped in a direct, template-based manner to two 2NCL threshold gates, one for each of the 0-rail and 1-rail logic, but DIMS-style mappings are also possible.

Finally, a simple rule-based cell merger algorithm developed by Theseus Logic is applied to the mapped circuit [10]. This algorithm is based on a restricted set of safe cell merging rules and the cell merger algorithm does not fully explore the design space. In this paper, a better method to address the cell merger problem is proposed.

2.4 Orphans

A key challenge in designing and optimizing asynchronous threshold circuits is to ensure robust implementations. An *orphan* can arise, when a signal transition on either a wire or a gate in the circuit is unobservable, and may cause a circuit malfunction if the transition is too slow [8, 15]. Before presenting some examples, a few definitions are required.

Suppose an NCL circuit is in a reset state where all the wires have 0 values. Once all the input data arrives and all the circuit outputs are computed, there must be at least one path from primary input to primary output where all the signal transitions are $0 \rightarrow 1$. The events on each such path are said to form a *signal transition sequence*.

A signal transition s_2 is said to *acknowledge* a signal transition s_1 if s_1 always precedes s_2 in any possible signal transition sequence in a set phase of NCL circuit. A signal transition is *unacknowledged* if it is not acknowledged by any signal transition on a primary output. An *orphan* is a signal transition sequence which is not acknowledged by any primary output of the circuit.

Orphans are classified into two classes: *wire orphans* and *gate orphans*. A wire orphan is an unacknowledged signal transition sequence of length 1 and a gate orphan is an unacknowledged signal transition sequence with length > 1.

(a) a wire orphan (b) computation interference

Figure 3. A wire orphan example [30]

As a wire orphan example, consider the circuit in Figure 3(a), which is taken from Figure 2(b). When $a_0 = 0, a_1 = 1, b_0 = 0, b_1 = 1$ in a set phase, the gate g_4 and g_5 fires. The thick lines indicate the wires where signal transition takes place. Of these, the dotted ones represent wire orphans whose signals do not further propagate through the gates, which are therefore unacknowledged.

Now, suppose that the lower wire orphan on the input wire of g_3 is extremely slow, and the transition does not reach gate g_3 by the time the next set phase begins. Note that in the intervening reset phase both output rails z_0 and z_1 will correctly settle to 0s regardless of this wire orphan.

In the second set phase, let $a_0 = 1, a_1 = 0, b_0 = 1$, and $b_1 = 0$ (Figure 3(b)). The thick solid lines indicate signal transitions in the second set phase. Because of the wire orphan, now a spurious signal transition may appear at g_3 firing g_5. Now, both output rails z_0 and z_1 fire, which obviously is illegal in delay-insensitive encoding.

Orphans can also span over gates, as illustrated in Figure 4. When $a_0 = 1, a_1 = 0, b_0 = 0, b_1 = 1$, a gate orphan is observed to propagate through gate g_3 until reaching its dead end. For more examples on gate orphans, refer to [8].

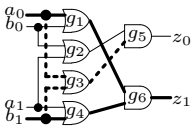


Figure 4. A gate orphan example [30]

Note that, when converting an irredundant 3NCL circuit to 2NCL, using DIMS-style, gate-orphan-freedom is guaranteed by construction. As illustrated in Figure 2, a DIMS-style 2NCL network equivalent for any 3NCL gate has the property that, during the set phase, *exactly one* of the C-elements (i.e. left column of gates) will be activated for each DATA input combination, which then feeds exactly one OR-gate, to assert one of the two dual-rail outputs. The result is that only one gate path will be activated, and no other gates will change value. A similar property holds during the reset phase. Hence, the mapping from 3NCL to 2NCL networks always preserves robustness, and therefore serves as a useful starting point for the proposed optimizations. This property also ensures that very few paths will be exercised during the set phase, contributing to the power efficiency of NCL circuits.

In NCL logic, wire orphans are not considered serious and can be engineered so they would not be a problem in real circuits. Effectively, they occur at fanout points, where an unobservable wire fanout delay (i.e. wire orphan) must always be faster than a significant observable *path delay*. The NCL tool flow is aimed at eliminating problems due to wire orphans at the physical design level by guaranteeing that differences between the fork delays are smaller than the gate delays.

However, gate orphans are more serious since they involve series of gates, and can more easily cause trouble with the circuit functioning. Therefore, in this paper, the problem of ensuring freedom from gate orphans is addressed. Note that, though orphans are discussed in the context of set functions, it was shown that if an NCL circuit is free of gate orphans in the set phase, then it is also free of gate orphans in the

reset phase [15]. So, this paper will focus only on the set functions of the cells when considering gate orphans.

Note that a circuit with wire orphans can be considered correct according to the isochronic fork assumption [19], while a circuit with gate orphans can be considered correct under the extended isochronic fork assumption [32].

3 Motivational Example

In this section, several examples are presented to show key points of the proposed algorithms. Examples show how gate orphans can be introduced by traditional technology mapping algorithms and an example that illustrates how the proposed cell merger algorithm works.

Example 1: Arbitrary decomposition can be dangerous. In the first example (Figure 5), suppose that the three-input AND gate g_1 is given in a gate-orphan-free NCL circuit, where g_1 never fires unless it is acknowledged by a primary output. Given that the two-input AND function is a base function, a traditional technology mapping algorithm may decompose a three-input AND gate into a network of two two-input AND gates (e.g. g_2 and g_3 in Figure 5). After the covering step, assume that these two-input AND functions are directly mapped to two-input AND cells themselves.

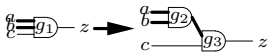
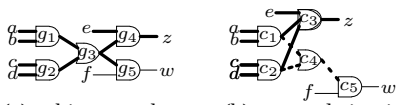


Figure 5. Example 1: Arbitrary decomposition [3]

Then, when $a = 1, b = 1, c = 0$ in a set phase of the circuit operation, while g_1 of the original circuit does not fire, but, in the mapped circuit on the right, g_2 fires even though g_3 does not fire. In the mapped circuit, a gate orphan propagates through g_2 since this signal transition is never acknowledged by a primary output.

Example 2: DAG-based covering can be dangerous. DAG-based technology mapping [17] which does not partition the unmapped logic network into trees can be dangerous since it can introduce new gate orphans. DAG-based algorithms are characterized by the fact that a single vertex of a subject graph can be covered by more than one cell in the mapped circuit.



(a) subject graph (b) mapped circuit

Figure 6. Example 2: DAG-based covering

A subject graph is given In Figure 6(a). Suppose that the given subject graph is gate-orphan-free and the vertex g_3 of the subject graph is covered by the two cells, c_3 and c_4, during the covering step. Figure 6(b) shows the mapped circuit where the other vertices g_1, g_2 and g_5 are trivially mapped to c_1, c_2, and c_5, respectively, Under the input combination $a = b = c = d = e = 1, f = 0$, while the subject graph is free of gate orphans, the mapped circuit has a gate orphan which propagates through c_4, which is drawn as dotted lines in Figure 6.

4 Theorems on Gate-Orphan-Freedom

In this section, a theorem and a corollary used in the technology mapping algorithm are presented.

Theorem 1 *Given a gate-orphan-free NCL circuit G, merging cells into a single functionally equivalent cell does not introduce gate orphans.*

From the above theorem, it can be also concluded that a sequence of cell mergings does not introduce gate orphans.

The following corollary suggests how gate-orphan-freedom can be preserved in technology mapping. A *gate-orphan-free decomposition* is defined as a decomposition of the original logic network into a gate-orphan-free subject graph, where a *gate-orphan-free subject graph*, in turn, is defined to be a subject graph whose trivial mapping is free of gate orphans. Given that the trivial mapping of the subject graph of G is free of gate orphans, covering of the subject graph by library cells is essentially the process of merging vertices of the subject graph.

Corollary 1 *Given a gate-orphan-free NCL circuit G, tree-based technology mapping with gate-orphan-free decomposition does not introduce new gate-orphans.*

5 Technology Mapping for Asynchronous Threshold Networks

In this section, a technology mapping algorithm for NCL circuits which preserves gate-orphan-freedom is presented. A brief overview of the approach is presented, followed by details on each step.

5.1 Overview of the approach

The algorithm takes a mapped NCL circuit which consists of 2NCL threshold cells, which can obtained either from dual-rail expansion from a 3NCL circuit or after applying rule-based cell merger algorithm to a dual-rail expanded 2NCL circuit. The algorithm loosely follows the framework of traditional tree-based technology mapping algorithms with structural matching, but with some important modifications to ensure the robustness of the mapping.

The algorithm works in three steps. First, the given robust (i.e. gate-orphan-free) 2NCL logic network is decomposed using a modified set of base functions. A new finite basis is proposed for general threshold networks, consisting of two primitive threshold cells. The functionality of any threshold logic network can be decomposed into a network consisting only of these nodes. However, such an arbitrary decomposition may introduce gate orphans. Hence, an extended basis is proposed, and a more limited decomposition is proposed, which exploits the simple finite basis wherever possible, but also includes some irreducible complex nodes, thus forming a heterogeneous decomposed logic network. The resulting network remains gate-orphan-free.

Next, the decomposed network is partitioned into distinct subject graphs, each of which will be mapped separately. A tree-based decomposition is used, to ensure that no gate orphans may be introduced. Pattern graphs are then generated for each cell in the given library, to prepare for the mapping step. In addition to pattern graphs built out of the simple base functions, some special graphs are added to ensure safe mapping of irreducible nodes.

Finally, the subject graph is optimally covered by the pattern graphs of library cells using a structural matching approach. Dynamic programming is used, and both area and delay costs are targeted. The result is an optimally-mapped, but still robust (i.e. gate-orphan-free) logic network, whose functionality is equivalent to the original netlist.

5.2 Decomposition: Choice of base functions

A simple finite basis is now proposed for arbitrary threshold logic networks using positive monotonic threshold gates with integer weights. As an example, all the 2NCL cells implement positive integral threshold functions. The proposed basis consists of two threshold cells: (i) a two-input threshold OR function and (ii) a two-input threshold AND function. The former can be combined, using associative law, to build up arbitrary 1-of-N threshold gates (i.e. OR-gates). The latter can likewise be combined, using associative law, to build up arbitrary N-of-N threshold gates (i.e. C-elements). These two functions indeed form a finite basis for any positive integral threshold functions, which follows from the fact that any Boolean function which is positive unate in all its input variables have a monotonic cover.

While this simple basis can capture all such threshold functions, it does not necessarily guarantee robust logic, i.e. gate-orphan-freedom. Therefore, for the technology mapping algorithm (see below), the basis will be augmented with limited complex nodes, as needed, to ensure a safe decomposition.

5.3 Gate-orphan-free decomposition

Decomposition of a logic network into a gate-orphan-free subject graph is crucial in ensuring technology mapping algorithm to preserve gate-orphan-freedom. In the NCL design flow, there are two ways to obtain gate-orphan-free subject graphs. First, right after expanding the 3NCL circuit into a DIMS-based dual-rail 2NCL circuit, the circuit is free of gate orphans by construction. If this circuit is taken as the starting point, traditional matching and covering algorithms can be used without introducing any gate orphans by Corollary 1.

However, in practice, the fully-expanded 2NCL circuit may not be available from the synthesis tool, and only robust post-optimized 2NCL circuits may be available, such as those obtained after application of the rule-based Theseus cell merger. For the latter case, an alternative method is proposed for undoing the existing optimizations, and recovering a more fully-expanded (i.e. pre-optimized) netlist, to allow for full reign for the new optimal algorithm.

Theorem 2 *In a gate-orphan-free logic network G, an n-input OR gate always can be decomposed into a tree of two-input OR gate without introducing gate orphans.*

Figure 7 shows an outline of the gate-orphan-free decomposition algorithm. The algorithm takes a post-optimized 2NCL circuit and returns a subject graph and a modified set of base functions.

There are two key techniques used to robustly decompose the original netlist. First, existing cell merger optimizations (if any are used by the tool flow) are undone, through a reverse lookup table, to expand the netlist to an initial decomposed form. Since only safe mergers were used by the tool, if they are correctly identified, the reverse-engineered circuit remains robust. Only unambiguous (i.e. functional) reverse mappings are used, to ensure that the optimizations are correctly reversed. Second, some of the remaining nodes (such as multi-input OR-gates) can also be further decomposed into the proposed simple base functions, using associative law, without introducing gate orphans. No further decompositions are employed, to guarantee that the resulting netlist remains robust. As an example, n-input AND cells ($n > 2$) are note decomposed.

The algorithm traverses the circuit in the topological order from the primary inputs to the primary outputs. In each step, both of the above decomposition steps are performed concurrently. When the algorithm visits a cell of the circuit, four cases can happen. First, when the cell is either a two-input OR cell or a two-input AND cell, we do not decompose the cell since the cell already implements a base function. Second, if the cell is an OR gate with more than two inputs, the OR gate is decompose into a tree of two-input OR cells. This decomposition is always safe by Theorem 2.

```
GOFDecompose(G = (V, E))
1  for (each v ∈ V)
2  do if (v is either two-input OR or two-input AND)
3        then do not decompose v
4        else if (v is an OR cell with n > 2 inputs)
5                then decompose v into a tree of 2-input OR cells
6        else if (v is a result of cell merger)
7                then roll back v by reversing the merger rule
8        else do not decompose v and register v as a new base function
```

Figure 7. Gate-orphan-free decomposition

When, the cell is resulting from the rule-based cell merger algorithm, we roll back the cell into the original network of cells through reverse engineering. There are 12 template-based cell merging rules used in the Theseus cell merger algorithm, where each cell merging rule consists of a a 'pattern template' and a cell. The Theseus cell merger algorithm iteratively searches for a pattern and replaces the pattern with the associated cell. The rules are mostly functional and inverse rules can be defined. In the gate-orphan-free decomposition algorithm, inverse rules are applied when a cell can be guaranteed to be resulting from a cell merger rule.

Finally, when it is impossible to decompose the cell with a guarantee of gate-orphan-freedom, the cell is not decomposed and the cell function is registered as a new base function.

After the gate-orphan-free decomposition, the resulting subject graph consists of original base functions (two-input OR and two-input AND) and new heterogeneous base functions added by the decomposition algorithm (line 8 in Figure 7).

5.4 Matching and covering

As a result of gate-orphan-free decomposition, there may exist more than two base functions and base functions can have more than two input variables. To handle this, matching and covering algorithms are modified to deal with more general classes of subject graphs and pattern graphs.

To generate pattern graphs for each cell in the library, the two original base functions (two-input threshold OR and two-input threshold AND) are used. In the first step, each threshold cell in the library is decomposed into a set of pattern graphs, each with a distinct structural pattern, to prepare for structural matching. This approach is similar to the traditional approach of [13, 7], however there are two key differences: (a) the actual cells include sequential threshold gates, and (b) the finite basis itself is different, involving two binary threshold cells (i.e. threshold OR and threshold AND).

Note that pattern graphs of some cells are represented as leaf-DAGs. For example, THAND cell of the NCL library implements a Boolean function $f = ac + ad + bc$ which cannot be represented as a tree. Representation of pattern graphs using leaf-DAGs is a partial solution to the problem of capturing variable sharing; this is the same problem facing synchronous technology mapping algorithms and a similar solution is used [22, 7].

In the second step of pattern graph generation, special pattern graphs are added, as needed, to facilitate the mapping of irreducible nodes in the subject graph. As indicated above, these nodes correspond to complex threshold gates which cannot be further decomposed safely, without potentially introducing gate orphans. The currently implemented approach is to simply map each of these irreducible nodes to the corresponding library complex gate. To this end, for each such irreducible node in the given subject graph, a single special complex pattern graph is added for the corresponding library cell, which is a complex root (i.e. complex base function) whose children are all variables. The subsequent matching algorithm, below, will ensure that such irreducible nodes will be mapped only to the equivalent complex library cell.

Note that the above approach to pattern generation, while fairly general, still excludes some potential matches. For example, it disallows the legal merger of a complex irreducible node with a simple base function. These cases are somewhat rare, given the fanin restrictions of NCL library. However, this limitation can be overcome if pattern graphs for each cell are generated starting with simple functions and complex irreducible functions. All possible trees constructed from these base functions can be enumerated and checked if the given tree implements the same function as the cell function. The space of generated trees can be bounded by the condition that a pattern tree must have exactly the same number of leaves as the number of input variables of the cell function. Leaf-DAGs for complex pattern graphs can also be generated.

```
Match(P, S)
1  // P is a pattern graph, S is a subject graph
2  if (P is a leaf)
3    then return true
4    else if (S is a leaf)
5           then return false
6         if (cell functions of P and S are different)
7           then return false
8           else // let P_1 ... P_k be P's fanins
9                // and S_1 ... S_k be S's fanins
10               for (all permutation Π = (σ_1, ..., σ_k) over [1, k])
11               do if (Match(P_σ1, S_σ1) · ... · Match(P_σk, S_σk))
12                     then return true
13               return false
```

Figure 8. Outline of matching algorithm

Figure 8 shows an outline of the matching algorithm used in the proposed technology mapping algorithm. The matching algorithm works similar to the traditional matching algorithm [7], when the root of the input subject graph is either two-input AND or two-input OR. When the root of the input subject graph is a complex function which was not decomposed (and added as a new base function) in the gate-orphan-free decomposition step, then the only possible pattern that can match the root of S is the complex function itself which was added as a base function. Hence, the algorithm effectively follows the traditional flow for matching algorithms.

For a delay model, the algorithm uses a nonlinear delay model based on table-lookup (more detail on the used delay model will be discussed in Section 6). The used delay model fits well with the dynamic programming framework of technology mapping algorithms. But since the delay of a cell partially depends on the output capacitance of the cell, the load binning technique [22] is used to find a solution. Load binning is a technique that extends the covering algorithm of technology mapping to solve simultaneously and optimally for multiple prospective fanout loads. During dynamic programming, as fanout loads are resolved on mapped gates, the actual output loads are then used to select the optimal solution so far that sees this load. Hence, the final result produces a single result for all the output loads in the circuit (and environment). The output capacitance of the cell is determined as the sum of the input capacitances of the input pins driven by this cell and the proposed algorithm extracts 20 representative input capacitance values from the library and uses these values for load binning. For primary outputs, a default output load value is used. Note that, while using load-dependent delay models results in sub-optimality due to the granularity of load binning and the partitioning, accurate load-independent delay models such as gain-based models [29] can be used to find delay-optimal mappings.

6 Experimental Results

The proposed technology mapping algorithm was implemented and experiments were performed to evaluate its effectiveness.The developed program implements the described technology mapping algorithm except that pattern graphs used in matching only consists of simple

base functions (i.e. complex irreducible base functions are not used). Programs were written in C++ and experiments were conducted on a 800Mhz Celeron machine with 256MB RAM running Redhat Linux 7.3. The developed programs take the logic network in the Berkeley BLIF format and the technology library in the Synopsys Liberty format, and outputs the mapped circuit in the BLIF format.

Delay model The delay model used in the library file was a nonlinear delay model with lookup-table.The NCL library characterizes each cell in terms of three parameters: input capacitance, cell delay, and output slew rate. The delay of an input-to-output path in a cell is specified using a two-dimensional array indexed by total output capacitance and input slew rate, where the output capacitance is determined as a sum of all input capacitances of the pins driven by the output and the input slew rate is contributed by the fanin gate. The array has 49 entries: for 7 output capacitances and 7 input slew rates. Additionally, each input pin of the library cell has its own input capacitance and one two-dimensional array is defined for each input-to-output path for computing the slew rate which will be used to determine input slew rates of the gates driven by the output pin of the cell. The delay model was used both for optimizing the circuits and for evaluating the circuits after optimization.

Experimental results Experimentation was conducted on a near-complete DES circuit provided by Theseus Logic and the results were compared with the original netlist which were pre-optimized using Theseus cell merger. The DES circuit, which had been pre-optimized with Theseus cell merger, consisted of five combinational clouds of logic wrapped with acknowledgment logic, registration, and a small amount of additional logic. Five combinational clouds of logic were extracted from the netlist and some of the combinational logic was quite substantial having up to 590 inputs, 306 outputs and 2196 internal gates. All the generated circuits from the technology mapper programs were verified for functional correctness against the original circuits using a program based on a BDD package.

Table 1 shows experimental results for the technology mapping program. In the table, the columns 2–4 show information on the original netlists which were pre-optimized using Theseus cell merger. Two runs of the algorithm were performed: The columns 5–7 show the results of area-optimized circuits and the columns 8–9 show the results of delay-optimized circuits. The parenthesized numbers in columns 6 and 9 contain the area percentage with respect to the original netlist, and the numbers in columns 7 and 10 are delay percentages averaged over all primary outputs. The last row of the table contains the area improvements averaged over all circuits and the delay improvements averaged over all primary outputs (each average is weighted by the number of primary outputs for the corresponding benchmarks).

For the delay-minimization run, two metrics were measured for each benchmark: *worst-case output delay* and *worst-case circuit delay*. For an individual benchmark circuit, the "worst-case output delay" is defined for an individual primary output as the worst-case path delay for this output, while the "worst-case circuit delay" is the worst-case path delay for the entire (multi-output) circuit. In the last column of the table, the *average worst-case output delay* for each benchmark is reported, i.e., the average, over *all* of the primary outputs of the circuit, of the individual "worst-case output delays".

The "average worst-case output delay" improvements, for the three largest circuits, were 26.7% (*des-r01*), 20.3% (*des-r04*), and 20.0% (*des-r05*). Better results were obtained for the very small *des-r03* and *des-r07* circuits.

Next, considering the "worst-case circuit delay", i.e. the single longest path of each circuit, the improvements were higher: 26.4% (*des-r01*), 26.3% (*des-r04*), and 26.0% (*des-r05*). The longest path delay is important in NCL circuits since acknowledgment signals are generated only after a signal transition along the longest path has been completed. Thus, this result gives a static worst-case bound on the possible operating speed of the circuit. Note that, while the delay-minimization run incurs area overhead, there exist application domains where area is not a major concern. Furthermore, modifying the algorithm run to combine area as a secondary cost metric is expected to reduce this overhead.

Finally, for the area-minimization run, the area improvements for the large circuits were 4.5% (*des-r01*), 2.2% (*des-r04*), and 2.2% (*des-r05*). Better result was obtained for the very small *des-r07* circuit. Thus, the area-minimization run limited effectiveness in minimizing area in post-optimized circuits and it is believed that area optimization capabilities were somewhat shadowed by the effectiveness of previously-applied Theseus optimizer. While local area reduction guarantees global area reduction, locally minimizing delay does not always result in global reduction in delay: this is believed to account for the effectiveness of Theseus optimizer in area optimization.

7 Conclusion

In this paper, an algorithm for technology mapping for asynchronous threshold networks was presented. The proposed algorithm is the first to systematically optimize for either delay or area without introducing new gate orphans, which is crucial in preserving the robustness of the circuit. The algorithm was implemented and experimented on NCL circuits which had been already optimized using Theseus tools and have shown further delay improvement of of 20.9% and area improvement of 2.7%, on average. Though the proposed method is applied in NCL design flow in this paper, contribution is general enough to be used in many other contexts.

Future work There are several possible research directions. First, usually DAG-based technology mapping algorithms provide more opportunity for optimizing circuits with respect to delay. Even if standard DAG-based algorithms are dangerous as discussed already, a careful extension of DAG covering can be devised to preserve gate-orphan-freedom. Second, the problem of gate-orphan-free decomposition was solved in the context of NCL design flow but a more generalized gate-orphan-free decomposition method can be devised for a wider application. Finally, a technology mapping algorithm that can target multiple cost functions at the same time, e.g. area and delay, can be devised to deal with the area overhead incurred when delay minimization is performed in the technology mapping algorithm.

References

[1] P. A. Beerel. *CAD Tools for the Synthesis, Verification, and Testability of Robust Asynchronous Circuits*. Phd thesis, Stanford University, 1994.

[2] P. A. Beerel, K. Y. Yun, and W. C. Chou. Optimizing average-case delay in technology mapping of burst-mode circuits. In *Proceedings of the 2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 244–259, 1996.

[3] S. M. Burns. General conditions for the decomposition of state holding elements. In *Proceedings of the 2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 48–57, 1996.

[4] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–99, 2003.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, E. Pastor, and A. Yakovlev. Decomposition and technology mapping of speed-independent circuits using Boolean relations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1221–1236, Sept. 1999.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D:315–325, Mar. 1997.

[7] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[8] K. M. Fant. *Logically Determined Design*. John Wiley & Sons, 2005.

[9] R. M. Fuhrer and S. M. Nowick. *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools*. Kluwer Academic Publishers, 2001.

[10] M. Hagedorn. private communication, 2005.

[11] Q. T. Ho, J. Rigaud, L. Fesquet, M. Renaudin, and R. Rolland. Implementing asynchronous circuits on LUT based FPGAs. In *Proceedings of FPL'02*, 2002.

[12] K. W. James and K. Yun. Average-case optimized transistor-level technology mapping of extended burst-mode circuits. In *Proceedings of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 70–79, 1998.

[13] K. Keutzer. DAGON: Technology binding and local optimization by DAG matching. In *Proceedings of the 24th Design Automation Conference*, pages 341–347, 1987.

[14] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Logic decomposition of speed-independent circuits. *Proceedings of the IEEE*, 87:347–362, 1999.

[15] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant. Checking delay-insensitivity: 10^4 gates and beyond. In *Proceedings of the 8th International Symposium on Asynchronous Circuits and Systems*, pages 149–157, 2002.

[16] P. Kudva, G. Gopalakrishnan, H. Jacobson, and S. M. Nowick. Synthesis of hazard-free customized CMOS complex-gate networks under multiple-input changes. In *Proceedings of the 33rd Annual ACM/IEEE Design Automation Conference*, pages 77–82, 1996.

[17] Y. Kukimoto, R. K. Brayton, and P. Sawkar. Delay-optimal technology mapping by DAG covering. In *Proceedings of the 35th Annual ACM/IEEE Design Automation Conference*, pages 348–351, 1998.

[18] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125, 2000.

[19] A. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4), 1986.

[20] J. McCardle and D. Chester. Measuring an asynchronous processor's power and noise. In *Proceedings of the Synopsys User Group Conference*, 2001.

[21] D. E. Muller. Asynchronous logics and applications to information processing. In H. Aiken and W. F. Main, editors, *Switching Theory in Space Technology*, pages 289–297. Stanford University Press, 1963.

		original circuit		area-optimized circuit			delay-optimized circuit		
name	# inputs/# outputs	# nodes	area	# nodes	**area**	delay	# nodes	area	**delay**
des-r01	352/64	1079	61378.4	1024	**58613.3 (95.5%)**	100.2%	1332	67927.8 (110.7%)	**73.3%**
des-r03	11/4	10	535.7	8	**512.6 (95.7%)**	95.3%	8	835.2 (155.9%)	**71.0%**
des-r04	590/298	2186	129941.0	2126	**127148.0 (97.8%)**	100.3%	2983	159374.0 (122.6%)	**79.7%**
des-r05	590/306	2196	130655.0	2136	**127863.0 (97.8%)**	100.3%	2987	159852.0 (122.3%)	**80.0%**
des-r07	3/2	4	138.2	2	**103.7 (75.0%)**	80.5%	2	115.2 (83.3%)	**57.5%**
average improvement					**(97.3%*)**	100.2%**		(120.4%*)	**79.1%****

* Overall average area improvement is an weighted average of area percentages, where larger circuits have larger weights.
** Overall average delay improvement is an weighted average of delay percentages, where circuits with more outputs have larger weights.

Table 1. Experimental results for technology mapping

[22] R. L. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, UCB/ERL M89/49. Electronics Research Laboratory, University of California at Berkeley, 1989.

[23] C. L. Seitz. System timing. In C. A. Mead and L. A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[24] P. Siegel and G. De Micheli. Decomposition methods for library binding of speed-independent asynchronous designs. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design*, pages 558–565, 1994.

[25] P. Siegel, G. De Micheli, and D. L. Dill. Automatic technology mapping for generalized fundamental-mode asynchronous designs. In *Proceedings of the 30th Annual ACM/IEEE Design Automation Conference*, pages 61–67, 1993.

[26] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb. Optimization of NULL convention self-timed circuits. *Integration, the VLSI Journal*, 37:135–165, 2004.

[27] G. E. Sobelman and K. Fant. CMOS circuit design of threshold gate with hysteresis. In *Proceedings of 1998 International Symposium on Circuits and Systems*, pages 61–64, 1998.

[28] J. Sparsϕ, J. Staunstrup, and M. Dantzer-Sϕrenson. Design of delay insensitive circuits using multi-ring structures. In *Proceedings of 1992 European Design Automation Conference*, pages 15–20, 1992.

[29] L. Stok and V. Tiwari. Technology mapping. In S. Hassoun and T. Sasao, editors, *Logic Synthesis and Verification*, pages 115–140. Kluwer Academic Publishers, 2001.

[30] Theseus Logic. *Introduction to NCL Logic: Training material*, 2002.

[31] C. H. van Berkel, M. B. Josephs, and S. M. Nowick. Scanning the technology: Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, 1999.

[32] K. van Berkel, F. Huberts, and A. M. G. Peeters. Stretching quasi delay insensitivity by means of extended isochronic forks. In *Proceedings of the 1st International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 99–106, 1995.

[33] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous state machines. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design*, pages 576–580, 1992.