Optimization of Robust Asynchronous Circuits by Local Input Completeness Relaxation

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 using synchronous CAD flows. One attractive alternative is to use robust asynchronous circuits which gracefully accommodate timing discrepancies. However, these asynchronous circuits typically suffer from high area and latency overhead. In this paper, an optimization algorithm is presented which reduces the area and delay of these circuits by relaxing their overly-restrictive style. The algorithm was implemented and experiments performed on a subset of MCNC circuits. On average, 49.2% of the gates could be implemented in a relaxed manner, 34.9% area improvement was achieved, and 16.1% delay improvement was achieved using a simple heuristic for targeting the critical path in the circuit. This is the first proposed approach that systematically optimizes asynchronous circuits based on the notion of local relaxation while still preserving the circuit's overall timing-robustness.**

I. Introduction

As process, temperature and voltage variations become significant in deep submicron design, timing closure becomes a critical challenge using synchronous CAD flows [3]. One attractive alternative is to use robust asynchronous circuits which gracefully accommodate timing discrepancies. Several classes of asynchronous circuits are highly-robust to delay variations, and only require small localized timing constraints to be satisfied. 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 [20]. As an example, Theseus Logic developed an asynchronous version of the Motorola CPU08 microcontroller as part of the MCORE project and reported 40% less power and 10dB less peak EMI noise than Motorola's synchronous version [13], [16]. Also, in recent work at Seiko/Epson, Karaki [9] demonstrated that asynchronous design is an effective approach for dealing with high variability of delays, when building flexible LTPS (Low Temperature Poly Silicon) TFT (Thin Film Transistor) displays. However, little CAD support and high area overhead in such design styles are among the major obstacles in adopting these robust methodologies.

In this paper, a class of highly-robust asynchronous circuits is targeted: dual-rail threshold networks. Basic design styles for this class of circuits have been proposed by several researchers: DIMS (Delay-Insensitive Minterm Synthesis) [15], [18], [4] and NCL (Null Convention Logic) [6]. The DIMS style is a simple approach to designing robust dual-rail asynchronous circuits from Boolean netlists in a template-based manner. NCL [6], [11] is a more recent approach to designing these circuits, and has been incorporated into a commercial CAD tool flow at Theseus Logic Inc. and applied to several industrial circuits. The NCL design style has been used to develop an asynchronous version of the Motorola MCORE processor [13]. Also, more than 18 other chips were designed and fabricated using the NCL flow with the largest having 660,000 transistors [11]. Currently, the NCL flow is used in a DARPA CLASS project, led by Boeing, which is a major new initiative to develop commercially viable asynchronous CAD flow.

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

Compared to other asynchronous circuit styles such as those based on single-rail data encoding [22], these asynchronous circuits require few timing requirements: arbitrary gate and wire delays are allowed as long as weak timing constraints are satisfied at wire fanout points (details are provided in Section 2). However, although both DIMS and NCL approaches result in highly-robust asynchronous circuits, they suffer from large area and latency overhead.

In this work, an optimization algorithm for this class of asynchronous circuits is presented. The algorithm can target both DIMS-style and NCL asynchronous circuits. The method optimizes area and delay of these circuits by relaxing their overly-restrictive style *without* sacrificing the robustness property of the circuits. In particular, given an original Boolean netlist, previous approaches replace each Boolean gate in the netlist by an equivalent robust dual-rail asynchronous block. However, these approaches are overly conservative in that *not every* Boolean gate needs to be replaced by a robust block to get a robust *circuit*. The goal of the paper is to identify a minimal set of Boolean gates, under different cost functions, which need to be replaced by robust asynchronous blocks to ensure the overall robustness of the entire circuit. Such an approach allows other Boolean gates to be replaced by more efficient, relaxed (i.e. non-robust) asynchronous blocks while still preserving the overall robustness of the circuit.

Problem formulation. The technique we focus on, in this paper, exploits the notion of "input completeness". A gate or a circuit is called *input complete* if its output changes *only after all* of its inputs have changed. Our goal is to selectively relax the input completeness requirement when mapping the original Boolean netlist into a dual-rail asynchronous circuit. In particular, the input completeness of a gate or circuit is said to be *relaxed* if its output may change *before* all inputs have changed. A relaxed gate or circuit is also said to perform *eager evaluation*. This optimization problem can be formulated as follows.

Problem 1 (Input Completeness Relaxation) *Given a Boolean logic network $G = (V, E)$, which is to be implemented as a dual-rail asynchronous circuit, find a set of gates in the original circuit such that 1) the relaxed dual-rail expansion of these gates still ensures that the resulting dual-rail circuit is timing-robust, and 2) the given cost function is minimized.*

In this paper, a theorem and a corollary that formalize a necessary and sufficient condition for legal relaxation are presented. This allows global optimization of circuits by a series of local relaxation of input completeness. A relaxation algorithm that targets three cost functions was proposed using the unate covering framework and the algorithm was implemented and experiments were performed with ten circuits randomly chosen from a set of larger MCNC benchmark circuits. The cost functions considered in the paper are as follows: a) the number of fully-expanded (i.e. robust) gates, b) area after dual-rail expansion, and c) the critical path delay. On average, 49.2% of the gates could be implemented in a relaxed manner and, as a result, 34.9% area improvement was achieved, and 16.1% delay improvement was achieved using a simple heuristic for targeting the critical path in the circuit. The average runtime of the algorithm was 6.13 seconds. This is the first proposed approach that systematically optimizes asynchronous circuits based on the notion of local relaxation while still preserving the overall circuit robustness.

Related work. To overcome area overhead of DIMS and NCL circuits, several optimization techniques have been proposed. One approach, which is the focus of this paper, is to allow *early evaluation* in gates in the circuit. Two strategies have been proposed: (i) optimizing every gate, but adding local completion detectors [1], [4], [10], and (ii) optimizing only some of the gates, with no added local completion detectors [16]. In the first strategy, all gates are relaxed to speed up computation, but local detectors are used to ensure robust completion. However, this approach may degrade system performance and significantly increases area. A second approach, closer to our work, was first proposed by Smith et al., who illustrated a few examples where input completeness can be relaxed on a *selected* set of nodes *without* using local completion detectors. However, they did not provide general conditions for legal relaxation or general algorithms with a notion of global optimality. In contrast, our approach provides a general relaxation algorithm without using local completion detectors. Also, their circuits were relaxed only in set phases while the circuits could further be relaxed also in reset phases (as is proposed in this paper).

A recent paper by Zhou et al. [23] developed a similar approach to ours, and was proposed concurrently. However, while their approach only targets area, ours targets three cost functions: number of relaxed nodes, area, and critical path delay. On the other hand, their approach introduces an extension to allow *partial* relaxation, where nodes can be relaxed with respect to a subset of inputs, while ours do not. Finally, while their algorithm is based on a SAT (Satisfiability) framework, ours is based on a unate covering framework.

Beyond 'eager evaluation', there have also been research efforts to optimize dual-rail circuits along the line of traditional logic synthesis. Theseus Logic developed a simple template-based optimization method for NCL circuits using localized cell merger [7], and recent approaches has also been developed for robust technology mapping [8]. However, these approaches work on already dual-rail expanded circuits and do not exploit the notion of relaxation.

II. Background

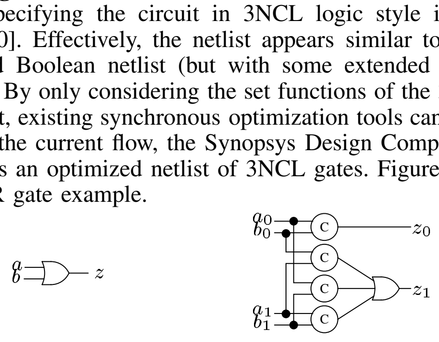
A. Boolean logic network

Let $B = \{0, 1\}$. A *Boolean function* f with n inputs and m outputs is defined as a mapping $f : B^n \rightarrow B^m$. 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 (see [5]). A *local 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.

B. NCL logic

NCL is a circuit implementation style for asynchronous threshold networks [6]. In this section, background on NCL design flow and details on its implementation style and robustness properties are presented, as well as a review of unate covering.

NCL design flow. The current NCL commercial design flow starts by specifying the circuit in 3NCL logic style in a VHDL program [10]. 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 initially applied. In the current flow, the Synopsys Design Compiler is used. The result is an optimized netlist of 3NCL gates. Figure 1(a) shows a 3NCL OR gate example.



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

Fig. 1. 2NCL logic example

Next, the optimized 3NCL logic is transformed into a 2NCL logic style, where each 3NCL gate is macro-expanded either into a DIMS-style dual-rail 2NCL block or into an NCL-style dual-rail 2NCL block. As an example, Figure 1(b) shows a DIMS-style 2NCL dual-rail equivalent circuit of the 3NCL OR gate in Figure 1(a) (see below for more details). [1] After dual-rail expansion, the logic is mapped to a pre-defined library of NCL threshold gates.

3NCL circuits. 3NCL is a three-valued logic with values $\{0, 1, N\}$. This representation allows a single bit of data to be captured with a single symbolic variable or wire. Of these 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). 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 a binary-valued Boolean circuit. NCL logic implements a single 3NCL gate using the DIMS-style dual-rail expansion [18], where each single variable (or bit) is mapped to a dual-rail Boolean equivalent. A DIMS-style logic implements a Boolean function as a network of complex robust minterms (C-elements [12]) feeding into OR-gates for 0 and 1 outputs (two OR-gates). The resulting circuit is timing-robust, as discussed later in this section.

Figure 1 shows an example of how a 3NCL gate is dual-rail expanded into a network of 2NCL gates in DIMS-style. In the example, a two-input 3NCL 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 wires a_1, b_1, z_1 represent the 1-rails of a, b, z. Four 2NCL "AND" gates, which are C-elements,[2] 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.

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. 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 cell library. An NCL threshold gate with hysteresis [17] is a gate whose set and reset functions 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. As an example, a two-input C-element, with inputs x_1 and x_2, has a set function $x_1 + x_2$. The reset function is $R = \overline{x_1 + x_2}$, indicating that both inputs must be reset before the output can be reset.

C. Orphans

A key challenge in designing and optimizing asynchronous threshold circuits is to ensure hazard-free 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 [6]. Before presenting some examples, a few definitions are required.

Suppose that an NCL circuit is in a reset state where all the wires have 0's. Once all the inputs arrive 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

[1]Alternatively, In the NCL-style 2NCL expansion, a single (complex) threshold gate is used instead, for each rail (i.e. only two gates are used in the NCL-style dual-rail implementation of the 3NCL OR gate).

[2]A C-element copies its input value to its output only when both inputs change to the same value. Otherwise, the output does not change.

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 the circuit. A signal transition is *unacknowledged* if it is not acknowledged by any signal transition on a primary output.

Definition 1 (Orphans) A circuit is said to have an **orphan** if, for some input transition, there is a signal transition sequence which is not acknowledged by a signal transition on any primary output. The circuit has a **wire orphan** if a signal transition on a *wire* is not acknowledged, and the circuit has a **gate orphan** if a signal transition sequence on a *path* through one or more gates is not acknowledged, by a signal transition on any primary output.

Intuitively, a circuit has a wire orphan if a wire transition may be unobservable at the circuit outputs, and a circuit has a gate orphan if a transition on a path through one or more gates may be unobservable at the circuit outputs.

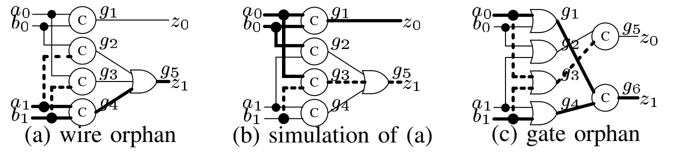


Fig. 2. Wire and gate orphan examples [19]

As a wire orphan example, consider the circuit in Figure 2(a). 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. 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 of g_3 is extremely slow, and the transition does not reach 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 can correctly settle to 0's regardless of this wire orphan. In the second set phase, let $a_0 = 1, a_1 = 0, b_0 = 1, b_1 = 0$ (Figure 2(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.

As a gate orphan example, consider a circuit in Figure 2(c), where orphans span over gates. Under an input transition, $a_0 = 1, a_1 = 0, b_0 = 0, b_1 = 1$, in a set phase, unobservable transitions can arise on a *path* through gate g_3, which starts at the input wire a_0 (or b_1) and ends at the output wire of g_3. In the figure, the thick lines indicate observable signal transitions and the dotted lines indicate *unobservable* signal transitions, which form a path through gate g_3. Therefore, the circuit in Figure 2(c) is said to have a gate orphan. For more examples on gate orphans, refer to [6].

Note that, when converting an irredundant 3NCL circuit to 2NCL, using DIMS-style, gate-orphan-freedom is guaranteed by construction. However, since this paper is concerned with optimization, a key goal is to ensure that no new gate orphans are introduced by the proposed optimization techniques. As illustrated in Figure 1, 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.

In the NCL synthesis flow, wire orphans are *not* considered serious and, in practice, timing constraints are easily enforced during physical design to ensure proper timing. 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. Thus, the timing requirements are on specialized non-isochronic (i.e. skewed) fork delays. The NCL commercial tool flow is aimed at eliminating problems due to wire orphans at the physical design level [11].

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, and will be *guaranteed* as an invariant by the proposed algorithm.

Note that a circuit with wire orphans can be considered correct according to the isochronic fork assumption [12], while a circuit with gate orphans can be considered correct under the extended isochronic fork assumption [21].

Input completeness. A circuit is said to be *input complete* if the circuit outputs change only after all inputs change. Similarly, a gate is *input complete* if the gate output changes only after all inputs change. As seen in Section II, 3NCL gates are input complete. When a 3NCL circuit is expanded into a dual-rail circuit, it is useful to enforce this input completeness property for the dual-rail block of each original 3NCL gate, since such a construction ensures that the final dual-rail circuit will be gate-orphan-free.

Definition 2 A dual-rail implementation of a *3NCL gate* is *input-complete with respect to its inputs* if an output makes a transition only after all the inputs have made transitions.

As an example, the dual-rail circuit Figure 1(b) is input complete with respect to its input signals, a and b.

The notion of input completeness of a dual-rail block can be defined *separately* for set and reset phases. In a dual-rail implementation of a 3NCL gate, if an output goes up only after all inputs go up, then the implementation is said to be *input complete in the set phase*. Also for reset phases, if an output goes down only after all inputs go down, then the implementation is said to be *input complete in the reset phase*. If an output of a dual-rail implementation can go up before all inputs go up, it is said to be *early evaluating*. For reset phase, if an output can go down before all inputs go up, the implementation is said to be *early resetting*.

In this paper, if a dual-rail implementation of a 3NCL gate is input complete with respect to a signal s (in set and reset phases), the implementation will be said to *cover* signal s, or *provide robustness* to signal s.

D. Unate covering

The *unate covering problem* (UCP) occurs in many contexts including the two-level logic minimization problem [14]. In this paper, the proposed problems will be solved using the framework of the unate covering problem.

Problem 2 (Unate covering problem) *Given a finite set of elements U and a collection C of subsets of U, find a minimum-cardinality subset $C' \subseteq C$ which covers U, i.e. $\bigcup_{C_i \in C} C_i = U$.*

Unate covering problem can be extended with a weight function $w : C \to \mathcal{R}^+$ which assigns a weight to each subset of U in C. For the weighted unate covering problem, the goal is to find a subset $C' \subseteq C$ such that $\sum_{C_i \in C'} w(C_i)$ is the minimum. In this paper, a UCP instance will be denoted by (U, C) and a weighted UCP instance will be denoted by (U, C, w).

III. Motivational Examples

In this section, examples are presented to show key points of the proposed relaxation approach. The first small example shows in detail how a 3NCL circuit can be translated to a 2NCL circuit in a more relaxed manner without affecting the overall robustness of the circuit. The second example is larger and shows how there are different choices of which 3NCL gates to relax.

Relaxed dual-rail circuit example. In Figure 3(a), a 3NCL circuit that computes exclusive-or (XOR) of two inputs a and b is given. A straightforward dual-rail implementation of the given 3NCL circuit, consisting of three dual-rail blocks, X, Y, and Z, is shown in Figure 4. The dual-rail blocks X, Y, and Z correspond to the gates x, y, and z of Figure 3(a).

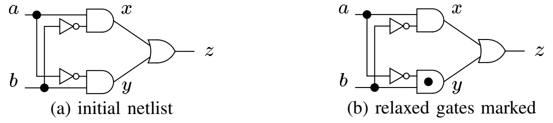


Fig. 3. 3NCL circuit example

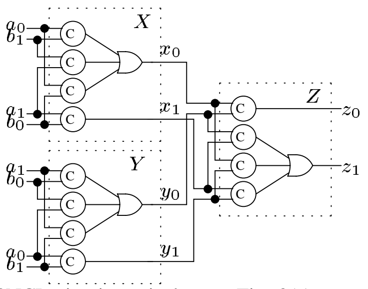


Fig. 4. Robust 2NCL circuit equivalent to Fig. 3(a)

Suppose that, initially, the dual-rail circuit is in a reset state, having all wires initialized to 0. The two dual-rail blocks, X and Y, ensure that the gate output signals x and y[3] make transitions only after signals a and b have arrived. The dual-rail block Z on the right ensures that primary output signal z makes transition only after internal signals x and y makes transitions. Also in the reset phase which starts after the primary outputs are settled to completion, the same property holds due to the hysteresis property of 2NCL gates.

However, this 2NCL implementation is overly restrictive. Consider again the 3NCL circuit in Figure 3(a). Signals a and b are each acknowledged on two distinct paths: through gate x and through gate y. Similarly, in Figure 4, the dual-rail inputs a and b are each acknowledged through two input complete blocks X and Y. If the bottom AND gate in the 3NCL circuit is flagged as relaxed as shown in Figure 3(b), the dual-rail block Y can be re-implemented using an input *incomplete* block Y' as shown in Figure 5. The resulting dual-rail implementation will still be robust (i.e. gate-orphan-free).

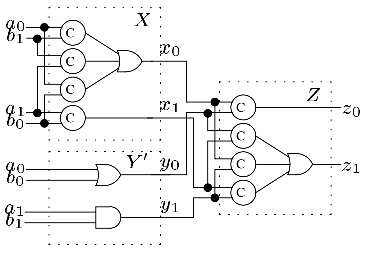


Fig. 5. Relaxed 2NCL circuit equivalent to Fig. 3(b)

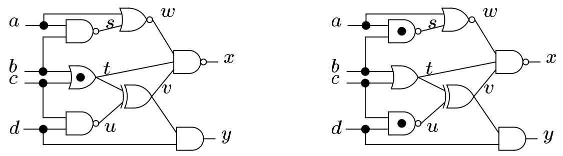
Note that dual-rail block Y' is *not* input complete since the signal y may make transition before all input signals arrive. The block both early evaluates in a set phase and early resets in a reset phase. However, for 3NCL signals a and b, block Y ensures robustness and the primary output z can change after both input signals a and b change. For a single primary input or a gate output signal, its input completeness need to be ensured only once in an entire dual-rail circuit.

Table I shows how 3NCL gates can be transformed into corresponding 2NCL (i.e. dual-rail) blocks. The rows of the table includes 3NCL gates and their possible transformations. The columns indicate two choices of transformations: traditional robust transformation is shown in the middle column and our new relaxed transformation is shown in the right column.

Choices in relaxation. Next, a larger example that illustrates how there are different choices of which 3NCL gates to relax is presented. There may exist many possible choices in picking which gates to relax (or which gates to fully-expand), which will result in different area and delay.

Figure 6 shows two 3NCL circuits with different choices of gates to relax, where the relaxed gates are marked with bullets. In the circuit on the left, one gate, t, is relaxed. In the circuit on the right, two gates, s and u, are relaxed. This difference stems from which gates are used to ensure robustness to the input signals b and c. In the left circuit, gates s and u are fully-expanded and ensure robustness to signals b and c. In the right circuit, the gate t is fully-expanded and covers signals b and c. Therefore, the overall cost in the final dual-rail implementation depends on which gates are chosen to relax (or to fully-expand).

[3]Based on the context, a 3NCL signal name may refer to its dual-rail encoded signals.



(a) relaxing one gate (b) relaxing two gates

Fig. 6. Choices in relaxation

IV. Theorems on Input Completeness

A theorem and a corollary are now presented, which formalize the precise conditions for "legal relaxation". The theorem describes a necessary and sufficient condition for a gate-orphan-freedom of a netlist in terms of *observability* of each internal signal in the netlist. For this purpose, a few definitions are presented first.

Definition 3 A 3NCL circuit is **gate-orphan-free** when, for any primary input and gate output signals, there exists at least one acknowledgment path to a primary output over all possible input transitions.

Definition 4 Given a 3NCL gate, a 3NCL-to-2NCL transformation of the gate is **legal** when (a) it has equivalent functionality to the 3NCL gate,[4] and (b) it is *locally gate-orphan-free* for all possible input transitions in the set and reset phases.

Theorem 1 (Input Completeness Relaxation) *Let a 3NCL circuit G be given. Then a 2NCL implementation of G is gate-orphan-free if and only if (a) G is free of gate orphans and (b) the 3NCL-to-2NCL transformation of each 3NCL gate is legal.*

Proof: First, the IF part is proved. Since the given 3NCL circuit is gate-orphan-free, for each gate output signal, there exists at least one acknowledgment path to a primary output. Then, after 2NCL instantiation of 3NCL gates in G, each 2NCL block's output signal for a 3NCL gate also has an acknowledgment path to a primary output. Since each 2NCL block is a legal 2NCL instantiation of 3NCL gates in G, for each input to a block, each internal gate of a 2NCL block is acknowledged by the block output. Therefore, combining the above two arguments, it can be concluded that each gate in 2NCL circuit is acknowledged and therefore is gate-orphan-free.

The proof of the ONLY IF part is by contradiction and is trivial. Suppose that either (a) or (b) does not hold. If (a) does not hold, there exists a signal s (gate output or primary input) that is not acknowledged in the 3NCL netlist. Then, the dual-rail outputs of s in the corresponding 2NCL block will also not be acknowledged. Alternatively, if (b) does not hold for s, the 2NCL block is either functionality-inequivalent to the 3NCL gate or contains gate orphans by definition of legality. In each case, the 2NCL circuit has gate orphans. ■

[4]More precisely, the functionality is equivalent after considering dual-rail encoding of the original gate.

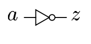
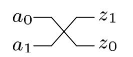
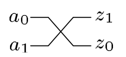
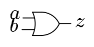
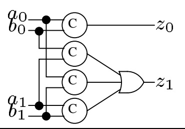
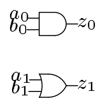
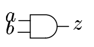
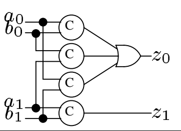
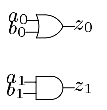
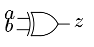
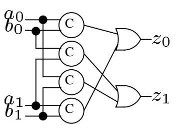
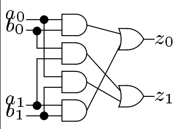
3NCL gate	full expansion (no relaxation)	relaxation
		
		
		
		

TABLE I. 3NCL-TO-2NCL TRANSFORMATION TABLE

From the above theorem, a key corollary can be derived which states a *local* condition which needs to be satisfied for each gate output signal to make a dual-rail implementation free of gate orphans.

Corollary 1 (Local Relaxation) *Let a 3NCL circuit G be given. Then a 2NCL implementation of G is gate-orphan-free if and only if, (a) at least one of the fanout gates of the signal ensures input completeness (i.e. is fully-expanded) for* each *primary input and gate output signal of G, and (b) 3NCL-to-2NCL transformation of each 3NCL gate is legal.*

Proof: The Corollary follows directly from Theorem 1, since the condition (a) of the Theorem holds if and only if at least one fanout gate of each signal in G ensures input completeness (condition (a) of the Corollary). ■

V. Relaxation Algorithm

A. An input completeness relaxation algorithm

A relaxation algorithm is now proposed, which finds a dual-rail implementation of the given 3NCL circuit without fully expanding *all* 3NCL gates. The algorithm can target three different cost functions. First, the algorithm minimizes the number of fully-expanded gates exactly and hence maximizes the number of relaxed 3NCL gates. The second cost function is to minimize the area the relaxed 2NCL (i.e. dual-rail) implementation. Finally, the algorithm heuristically targets the single *worst-case* critical path delay of the dual-rail implementation. In each case, the algorithm first flags which 3NCL gates must be mapped robustly and which can be mapped in a relaxed manner. Finally, the corresponding mapping is performed, from the initial 3NCL circuit to obtain the optimized 2NCL circuit.

Sketch of the relaxation algorithm. The proposed algorithm uses the unate covering framework and can target different cost functions through different weight assignments. Base on the corollary presented in the previous section, a uniform framework for solving the relaxation problem is presented.

The Local Relaxation Corollary suggests that the relaxation problem is essentially a *unate covering problem*, where each gate output and primary input signal *must be "covered"* (i.e. robustly acknowledged) by *at least one* of its fanout gates. That is, for each signal s in the 3NCL netlist, at least one of its fanout gates must be fully-expanded, i.e. its 2NCL expansion cannot be relaxed.

More formally, the relaxation problem is reduced to the unate covering problem using the following transformation. A set U of covered elements is defined to be the set of all 3NCL primary input and gate output signals. Also, a 3NCL gate $v \in V$ covers a signal u in U exactly when the given signal is an input to v. Therefore, the collection C of covering objects is formed as: $C = \{C_v : v \in V$ and $C_v = \{u \in U : v \text{ covers } u\}\}$, where a gate v is defined to *cover* u exactly when the gate output of u is fed as an input to gate v.

Figure 7 shows a general outline of the relaxation algorithm. The algorithm takes an initial 3NCL netlist $G = (V, E)$ as an input and returns a subset $V' \subseteq V$ of 3NCL gates which need to be fully-expanded to guarantee gate-orphan-freedom. The unate covering problem solver, UCPSolver, gives a solution that minimizes the weighted sum of gates in V'. In the algorithm, by changing the weight assignment scheme in Line 7, different cost functions can be targeted. After the set V' of gates is produced by Algorithm Relax, the final relaxed dual-rail implementation can be obtained by expanding each 3NCL gate into a dual-rail network.

```
Relax($G = (V, E)$)
1  // create a UCP instance; initially, $U = \emptyset$
2  for (each $v \in V$)
3  do $C_v \leftarrow$ the set of signals fed into $v$
4     $U \leftarrow U \cup C_v$
5  // assign weights to gates
6  for (each $v \in V$)
7  do $w(v) \leftarrow$ weight
8  // solve UCP instance $(U, C, w)$ to get a solution
9  $V' \leftarrow$ UCPSolver($U, \{C_v : v \in V\}, w$)
```

Fig. 7. Outline of the relaxation algorithm

B. Maximization of number of relaxed gates

First, the relaxation algorithm is configured to maximize the number of relaxed gates. Though this cost function does not directly address optimal area or delay after dual-rail expansion, it is a good first-cut cost measure since, in most cases, more relaxed gates means less area and shorter delay. For this, the weight function is defined as the constant value 1 for each 3NCL gate v. Since every 3NCL gate has the same weight, the unate covering problem is aimed at minimizing the number of picked (i.e. fully-expanded) 3NCL gates, or maximizing the number of unpicked (i.e. relaxed) gates.

C. Area optimization

Even if area and delay of dual-rail circuits can be optimized by maximizing the number of relaxed gates, it is not necessarily area-optimal since this cost function regards the expansion costs of each 3NCL gate to be identical. For example, while the area required for expansion of a 2-input OR 3NCL gate and the area for expansion of a 3-input 3NCL OR gate can differ significantly, the previous cost function cannot distinguish this difference. More realistically, a relaxation that targets minimization of the area *after* dual-rail expansion is more desirable. If the area of each 2NCL gate used in dual-rail implementations is known in advance, a weight function can be devised so that the area of the resulting relaxed dual-rail implementation is minimized.

To solve this optimization problem, a weight function can be defined for each 3NCL gate such that the weight of a 3NCL gate conveys information on area that the gate will require after it is fully expanded. The weight function used in the algorithm is: $\mathit{weight}(v) = \mathit{full\_area}(v) - \mathit{relaxed\_area}(v)$, where $\mathit{full\_area}(v)$ is the area of the dual-rail implementation of the 3NCL gate v *without* relaxation and $\mathit{relaxed\_area}(v)$ is the area of the relaxed dual-rail implementation. Intuitively, the weight of a gate is the area penalty that should be paid by making its dual-rail implementation robust. The relaxation algorithm with the given weight function is provably *optimum* in terms of area after dual-rail expansion.

D. Critical path delay optimization

The relaxation algorithm can also target delay optimization using a simple heuristic. This scheme focuses on the single worst-case critical path delay, and weights are assigned such that relaxation of gates is biased towards the gates in this critical path. While this approach is somewhat limited, since exactly one path is targeted, generalizations to handle more global delay reduction on multiple critical paths are expected to be straightforward.

For this cost function, the algorithm starts by finding the critical path in the *fully-expanded* (i.e. non-relaxed) dual-rail implementation of 3NCL netlist. When the critical path of the dual-rail netlist is found, the gates of original 3NCL netlist which *corresponds* to the critical path nodes of dual-rail netlist are back-annotated. Higher weights are assigned to critical 3NCL gates in the hope that non-critical gates will be more likely to be picked for full expansion.

VI. Experimental Results

The proposed relaxation algorithm was implemented and experiments were performed to evaluate its effectiveness. A CAD tool was written in C++ and experiments were conducted on a 800Mhz Celeron machine with 256MB RAM running Redhat Linux 7.3. The tool takes a logic network in the structural VHDL format and a technology library in the Synopsys Liberty format and outputs the optimized dual-rail circuit in the VHDL format. Unate covering problems were solved using the MINCOV program in ESPRESSO [14]. For computation of weight functions in the relaxation algorithm, an NCL cell library was used which is fully characterized for area and delay.

The CAD tool was applied to ten MCNC benchmark circuits, which were randomly chosen from a set of larger circuits. The circuits were preprocessed as follows. First, multi-level optimization was performed using `script_rugged` and then the resulting circuit was mapped using `map` into a Boolean netlist in SIS, which is then considered as a 3NCL netlist. During mapping, the library only contained gates with up to three inputs, since it is prohibitively expensive to dual-rail expand gates with more than 3 inputs.

original circuit		DIMS expansion			minimize # full blocks			minimize area			optimize delay		
name	#i/#o/#g	# full	area	delay	# full	area	delay	# full	area	delay	# full	area	delay
C1908	33/25/462	343	94352	30.0	180	59822	26.6	181	58618	27.9	184	60196	25.9
C3540	50/22/1147	911	281918	46.0	476	190470	39.3	477	189612	39.7	477	190996	38.7
C5315	178/123/1659	1259	335801	32.7	727	237273	29.8	730	235391	29.9	728	237453	28.5
C6288	32/32/3201	2385	567010	133.6	1246	361644	107.4	1247	361478	107.4	1246	361990	106.1
C7552	207/108/2155	1677	427101	44.8	1042	306573	43.4	1044	305203	43.4	1045	307113	43.4
dalu	75/16/756	633	201912	20.0	346	147830	14.6	359	144288	15.6	346	147830	14.8
des	256/245/2762	2329	712145	23.2	1157	466635	19.9	1159	462165	19.5	1162	469175	19.5
k2	45/43/684	597	222326	18.9	289	139898	14.4	300	131498	15.7	294	141490	14.0
t481	16/1/510	476	154466	20.8	211	101922	17.5	213	99514	18.1	211	101576	17.5
vda	17/39/383	309	121947	17.7	137	74033	15.0	143	69231	15.7	140	75957	15.7
average percentage					(51.8%)	(66.6%)	(84.0%)	(52.5%)	(65.1%)	(86.9%)	(52.7%)	(65.8%)	(83.9%)

TABLE II. EXPERIMENTAL RESULTS: COMPARISON WITH DIMS EXPANSION

original circuit		NCL expansion			minimize # full blocks			minimize area			optimize delay		
name	#i/#o/#g	# full	area	delay	# full	area	delay	# full	area	delay	# full	area	delay
C1908	33/25/462	343	55490	33.3	180	39756	29.2	181	37917	30.8	182	39389	28.3
C3540	50/22/1147	911	189970	51.0	476	148941	43.4	477	147575	43.8	480	148680	42.8
C5315	178/123/1659	1259	189370	36.4	727	157390	32.9	730	154238	33.1	727	156917	31.0
C6288	32/32/3201	2385	264750	151.1	1246	203910	119.5	1247	203490	119.5	1247	203963	123.0
C7552	207/108/2155	1677	224790	48.8	1042	182621	47.2	1044	180362	47.2	1042	182621	46.9
dalu	75/16/756	633	140190	21.7	346	121668	15.3	361	113949	16.3	348	121774	15.5
des	256/245/2762	2329	364812	24.8	1157	364812	21.4	1161	358692	20.9	1160	366472	20.9
k2	45/43/684	597	175590	20.2	289	122372	14.9	300	108765	16.4	301	119697	14.8
t481	16/1/510	476	109000	22.1	211	88333	17.8	213	84655	18.9	216	86706	17.7
vda	17/39/383	309	100230	19.0	137	67937	15.8	143	60214	16.6	138	68463	15.7
average percentage					(51.8%)	(77.4%)	(83.8%)	(52.5%)	(74.1%)	(85.9%)	(52.4%)	(77.0%)	(82.3%)

TABLE III. EXPERIMENTAL RESULTS: COMPARISON WITH NCL EXPANSION

Two experiments were performed. In the first experiment (Table II), the relaxed dual-rail circuits were compared with DIMS-style dual-rail circuits. In the second experiment (Table III), the relaxed circuits were compared with NCL-style dual-rail circuits which already use a simple Theseus-specific optimization (called 'cell merger' [7]). In Table II and Table III, the first two columns show the name of the MCNC circuit and the number of inputs, outputs, and gates. The tables include four column categories for straightforward dual-rail circuits, relaxed circuits with the minimum number of fully-expanded blocks, area-optimized relaxed circuits, and relaxed circuits with optimized critical path delay. For each of the categories, three sub-columns are included: the number of fully-expanded blocks, the area after dual-rail expansion, and the critical path delay after dual-rail expansion. For Table III, NCL expansion includes NCL-specific optimization where, for a given 3NCL gate, logic for each rail is implemented using a single complex cell rather than using multiple cells, as discussed in Section II.

Table II shows experimental results which compares the relaxed circuits with DIMS-style asynchronous circuits. The algorithm could relax 49.2% (for "minimize # full blocks") of the 3NCL gates on average. Also, on average, they achieved 34.9% improvement (for "minimize area") in area and 16.1% improvement (for "optimize delay") in critical path delay.

Table III shows experimental results which compare the relaxed circuits with NCL-style asynchronous circuits. The proposed algorithm could relax 49.2% (for "minimize # full blocks") of the 3NCL gates and it achieved 25.9% improvement (for "minimize area") in area with 17.7% improvement (for "optimize delay") in critical path delay, on average. The area results were not as good as the results obtained from the DIMS comparison. The reason is that the benefit of relaxation was overshadowed by the NCL-specific area optimization technique which is discussed in Section II-B.

The average runtime of the algorithm was only 6.13 seconds per each benchmark. The algorithm ran in a few seconds for most of the circuits, with the worst case of 61.54 seconds (*des*). Though the unate covering problem is NP-hard, due to the local nature of relaxation, the covering problem instances consisted of many small independent sub-problems and, in practice, were solved efficiently.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, an optimization algorithm for a class of highly-robust asynchronous circuits is presented. Though the circuits are robust to timing variations, they suffer from high area and latency overhead inherent in their style. The proposed algorithm optimizes area and delay of these circuits by relaxing their overly-restrictive style *without* sacrificing the overall robustness property of the circuits. The proposed algorithm was implemented and experiments were performed on MCNC circuits. On average, 49.2% of the gates could be implemented in a relaxed manner and, as a result, 34.9% area improvement and 16.1% critical path delay improvement were achieved.

As future work, we plan to extend the approach to relax the set and reset phases independently. Also, we plan to develop a more sophisticated scheme for critical path delay improvement, to target reduction of multiple critical paths with appropriate weights. Finally, the proposed techniques, with modification, should be applicable to other classes of less robust asynchronous circuits.

REFERENCES

[1] C. F. Brej. *Early Output Logic and Anti-Tokens*. PhD thesis, University of Manchester, 2005.

[2] S. M. Burns. General conditions for the decomposition of state holding elements. In *Proc. ASYNC'96*, 1996.

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

[4] I. David, R. Ginosar, and M. Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Trans. Computers*, 41(1), 1992.

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

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

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

[8] C. Jeong and S. M. Nowick. Optimal technology mapping and cell merger for asynchronous threshold circuits. In *Proc. ASYNC'06*, 2006.

[9] N. Karaki. Asynchronous design: An enabler for flexible microelectronics. ASYNC'06 Invited Talk, 2006.

[10] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous cad tools. *IEEE Design & Test of Computers*, 19(4), 2002.

[11] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proc. ASYNC'00*, 2000.

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

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

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

[15] N. P. Singh. A design methodology for self-timed systems. Technical Report MIT/LCS/TR-258, Laboratory for Computer Science, MIT, 1981.

[16] 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, 2004.

[17] G. E. Sobelman and K. Fant. CMOS circuit design of threshold gate with hysteresis. In *Proc. ISCAS'98*, 1998.

[18] J. Sparsϕ, J. Staunstrup, and M. Dantzer-Sϕrenson. Design of delay insensitive circuits using multi-ring structures. In *Proc. EuroDAC'92*, 1992.

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

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

[21] K. van Berkel, F. Huberts, and A. M. G. Peeters. Stretching quasi delay insensitivity by means of extended isochronic forks. In *Proc. ASYNC'95*, 1995.

[22] K. van Berkel and M. Rem. VLSI programming of asynchronous circuits for low power. In *Asynchronous Digital Circuit Design*, Springer Verlag, 1995.

[23] Y. Zhou, D. Sokolov, and A. Yakovlev. Cost-aware synthesis of asynchronous circuits based on partial acknowledgement. In *Proc. ICCAD'06*, 2006.