Coping with the variability of combinational logic delays

J. Cortadella[†], A. Kondratyev[‡], L. Lavagno[*‡] and C. Sotiriou[§]

[†]Univ. Politècnica de Catalunya, Barcelona, Spain
[‡]Cadence Berkeley Labs, Berkeley, USA
[*]Politecnico di Torino, Torino, Italy
[§]FORTH, Crete, Greece

Abstract— **This paper proposes a technique for creating a combinational logic network with an output that signals when all other outputs have stabilized. The method is based on dual-rail encoding, and guarantees low timing overhead and reasonable area and power overhead. This is achieved by adopting a set/reset behavior that is reminiscent of the operation of dynamic logic, but which is suitable for static logic as well. The transformation from single-rail to dual-rail can be applied both at the technology-independent and at the technology-dependent levels. We also discuss various scenarios in which completion detection can be used to measure the delay of a synchronous circuit at fabrication time or at run time, and of an asynchronous circuit at run time. We conclude by showing, on a large set of benchmarks, the effectiveness of the proposed technique.**

I. Introduction

Variability is a growing concern to digital circuit designers. Parameters determining the delay of gates and wires vary broadly both between chips and within chips. While statistical timing analysis techniques try to apply various empirically derived models to combine together all sources of variability along a critical path, designers would like to have a way to *measure* rather than just *predict* the actual delay of a circuit. The reason is that a large fraction of circuits work faster or much faster than worst-case analysis would predict. Moreover the range of variation of actual performance is very broad even between nominally identical chips produced by the same fabrication line.

Unfortunately delay fault testing is a costly proposition, because it requires the designer to start from a very expensive initial two-level representation, and then use only a subset of the logic optimization techniques [1]. Moreover, delay test patterns are more numerous than functional fault test patterns, and hence require more time on expensive testing machines. Microprocessors can be tested for operation speed, and binned accordingly, thanks to the very high levels of controllability and observability, and to the availability of functional patterns that test known worst-case paths.

The goal of this paper is to propose a technique, derived from the properties of dynamic logic circuits but adapted to work with *any static CMOS library*, that enables ASIC designers to easily measure exactly when a combinational circuit is done computing. In practical terms, we guarantee that *without almost any timing overhead* and with some area overhead, every combinational logic block has an additional completion detection output that rises a few gate delays after the last primary output has settled. This paper improves with respect to the state of the art in the following directions, by proposing:

- a technique for dual-rail network creation with much lower area, power and delay overhead than previously known techniques, and requiring only standard static CMOS gates.
- a novel method, using timing assumptions rather than completion detection, for fast reset to the spacer (inactive) state.
- an architecture for completion detection based on standard combinational gates, rather than Muller C elements, again exploiting timing assumptions in the reset phase.

We show in Section VI that circuits obtained using our logic synthesis flow have a *nominal* delay which is on average within 33% of the corresponding circuit optimized using traditional techniques, with a 100% area overhead. However, by using the completion detection output their *true* delay can be measured and used to reliably latch their output values, thus reducing the *margins* that must be taken at design time. Moreover, most of this overhead is due to the completion detection network, that currently is generated to check all outputs, not just critical ones, and thus can be further optimized.

Regarding power consumption, we can assume that it will increase approximately by a factor of 2, because our circuits do not have any glitches (which are known to consume about half of the active power in combinational logic), but have on average twice as much capacitance and twice as many transitions as traditional combinational logic. The increase in transitions due to the set/reset phase operation described below, approximately offsets the reduction due to absence of glitches. Leakage and dynamic power thus approximately doubles, due to the doubling of active area and wiring. [1]

Two key assumptions motivate the use of circuits with completion detection:

- The difference between worst case and true delays is in the range of 60-100% (see the detailed breakdown of worst case delay penalties in Section II), and will most likely increase over time.
- The fraction of "truly critical" register-to-register combinational logic blocks is relatively small (10-20% according to internal sources). Many such blocks, especially in ASIC designs, have significant timing slack.

Based on these two assumptions, we believe that, by using circuits with completion detection, one can achieve at least *25% of performance increase with 10% to 20% penalty in power and area* purely remaining in the synchronous domain. In this paper we are considering the following application scenarios for our design flow:

1) Fully synchronous design, when the circuit must be tested off-line for operation speed and binned accordingly. In this scenario, a few test patterns are applied (preferably in normal operation mode, but also possibly in scan mode) in order to sensitize the longest paths, and the completion detection output is sensed in order to determine the speed.
2) Almost synchronous design, when the circuit must be robust with respect to dynamic parametric variation, due e.g. to temperature and/or supply voltage. In this case, failures to raise the completion output in time can be used to slow down or stop the clock, or to trigger a system reset in the worst case.
3) Asynchronous circuit design, in which one takes advantage of both parametric variations and *data-dependent computation times* (e.g. it is well-known that the average case delay for a

[1]Results including power consumption and post-routing extracted capacitances will be included in the final version of the paper, if accepted.

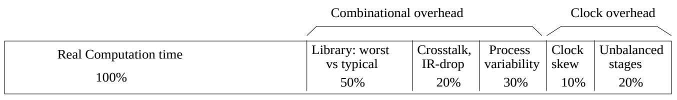


Fig. 1. Delay penalties in the modern synchronous methodology

ripple carry adder is logarithmic in the number of bits, assuming uniformly distributed input values).

4) Low-EMI design, in which the circuit must absorb current from its power supply with patterns that do not depend on the actual data values, in order to minimize emission and make security attacks based on Differential Power Analysis much harder.

The paper is organized as follows: Section II provides the motivation for our contribution. Section III describes previous work in the area. Section IV provides the theoretical background to justify the correctness of our approach and discusses application scenarios. Section V illustrates the two design flows that we have developed. Section VI describes experimental results showing their effectiveness.

II. Motivation

The main advantage of completion detection in combinational circuits is the ability to run the circuit using true rather than worst-case delays. Figure 1 shows the delay penalty stemming from two main factors: a) static timing analysis technology (the industry standard for doing timing sign-off) that estimates delays for the worst case scenario and b) clocking overhead in the conventional synchronous methodology.

These penalties are:

1) The difference in the library files between typical and worst case. This may account for up to a 50% penalty.
2) Conservative estimation of the slowdown caused by signal integrity violations: namely IR drop and crosstalk. Each of them could be responsible for a 10% penalty [2], [3].
3) Variations of delays due to the process variations. Here we assumed it to be about 30% for the latest technologies (see e.g. [4]), though more aggressive forecasts also exist.
4) Clock skew, which is about 10% of the clock cycle for the modern ASIC methodology [5].
5) Non-ideal balancing of synchronous stages, which increases the clock cycle with respect to the mean cycle time by up to 20% [5].

The first three sources come from the combinational logic itself, while the last two are coming from the clocking scheme.

Summing these penalties gives a quantitative estimation of the margin that separates true from worst-case delays. This margin shows the optimization room that is available when implementing circuits with completion detection. Part of this advantage is offset by the overheads due to the dual rail conversion, the two-phase operation and the completion detection logic itself. However, as shown in Section VI, these penalties are lower than the combinational logic penalty cited above. Clock penalties, as well as exploiting data-dependent delays, may provide additional performance improvements when the circuits with completion detection are used in an asynchronous environment, as discussed below.

III. Previous Work

Our work has been inspired by two active research areas: asynchronous design and synthesis of dynamic logic.

Asynchronous systems do not rely on timing assumptions to reason about settling of data at the circuit outputs. Instead they

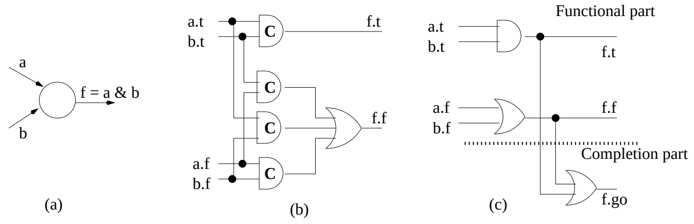


Fig. 2. Two ways of dual-rail implementation for AND node

coordinate behaviors in a causal way. As a result, logic synthesis for asynchronous circuits needs to take care not only of the functionality but also the proper ordering of the gates switching. This is achieved by introducing redundancy a) in the protocol with which the circuit input/output is handled (also called “mode of operation”) and b) in the circuit itself, in order to derive information about completion of the evaluation.

One of the most popular operation modes in asynchronous design alternates data communication between set and reset phases [6], with the two-fold goal of: a) providing a clean separation between two consecutive data sets and b) ensuring monotonic behavior at circuit outputs, avoiding spurious transitions known as hazards [7]. Circuit inputs and outputs change from a *spacer* in the *reset* phase, which carries no data information and serves only synchronization purposes, to a proper *data codeword* during the *set* phase, and then back to the spacer in the next reset phase.

The simplest scheme for such communication is given by dual-rail encoding, in which each signal a is represented by two wires $a.t$ and $a.f$: $a = 1$ is encoded as $a.t = 0, a.f = 1$, and $a = 0$ is encoded as $a.t = 1, a.f = 0$. The spacer for the reset phase is usually encoded as $a.t = a.f = 0$ (the complementary encoding, with spacer $a.t = a.f = 1$ and exactly one signal falling to indicate a valid codeword, can also be used). An attractive property of dual-rail encoding is the capability to determine that a proper codeword has settled by checking the codeword itself, without making timing assumptions. Indeed, if the circuit is hazard-free, then the fact that exactly one of the wires in each dual-rail pair goes high, tells that the output has settled and the set phase is over.

Two regular methods for implementing an arbitrary Boolean network with two-phase operation and dual-rail encoding were suggested in [8] and [9]. In both techniques every node of a network representing Boolean function f is implemented by two logic cones $f.t$ and $f.f$, where $f.t$ represents the original function f, while $f.f$ represents its inverse $\overline{f}$. In the DIMS method [8] $f.t$ and $f.f$ are implemented as sums of minterms (see Figure 2(b) e.g. for the implementation of AND function), with every minterm being realized by a C-element (a sequential gate with equation $c = ab + ac + bc$). When inputs make transitions from the spacer to a codeword, then one (and only one) output rises. Moreover, since only one of the minterms in the cone can rise during the set phase, the rising of the corresponding output rail happens as the last transition in the cone. In this way the information about completion is transmitted from inputs of the cone to its outputs, providing 100% observability for every circuit path. This advantage, however, comes at a significant cost, because none of the conventional logic optimizations are possible in DIMS circuits without losing completion detection properties.

This difficulty is partially addressed in the NCLX design flow [9] which is the closest approach to ours. In [9] a dual-rail network is created by 1) simply adding duals to every gate and 2) eliminating inversions by using corresponding complementary rails. Completion

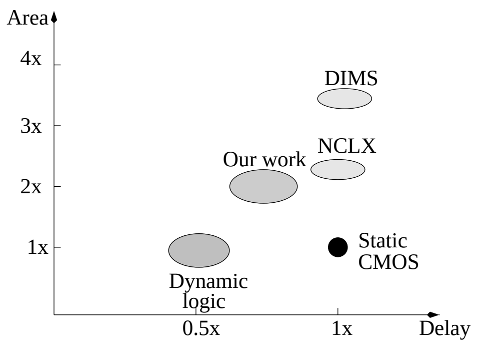


Fig. 3. Architecture-dependent area-delay trade-offs

detection is carried out separately from the data evaluation (see Figure 2(c), which uses an OR gate as a completion detector for $f.t$ and $f.f$).

Partitioning a circuit into loosely connected functional and completion detection parts allows a designer using the flow described in [9] to optimize and implement circuits using the full power of logic synthesis and EDA tools. This results in lowering the area penalty, which however cannot be reduced beyond 2x because of the dual-rail encoding.

Such area overhead may or may not be acceptable depending on other advantages (speed/power) that these circuits could provide. The main goal of detecting completion is to cover the increasing gap between "true" and worst case nominal delays. Unfortunately this is impossible in both approaches mentioned above, because the speed advantage, due to using the true delays instead of worst-case approximations, is offset by the need to use two operational phases (set and reset) in a single computational cycle. The nearly 2x delay penalty from the two-phase operational mode is difficult to recover through reducing delays from worst case to true.

Dual-rail networks with two-phase operation are also used in dynamic logic synthesis [10], [11]. In the following we discuss about domino logic, even though most considerations extend to other dynamic logic styles. A major limitation of domino logic is that it can only implement non-inverting logic (other styles require strict layering of, e.g. pre-charged and pre-discharged gates). Hence complements of internal signals need to be realized through separate cones of logic using dual functions and giving rise to partially or fully dual-rail circuits. Similar to asynchronous systems, dynamic circuits work in two phases, namely *pre-charge* and *evaluate*. This feature, however, allows dynamic logic to achieve significant speed improvements over static CMOS, because the reset (pre-charge) phase is very short. Unfortunately, the inherent noise sensitivity and charge-sharing problems associated with this design style limit its application to small timing-critical portions of systems, which are usually hand-designed.

Our paper makes an attempt to take the best of both worlds and develop a design flow that combines performance advantages inherent in true delay measurement, with the robustness and simplicity of static standard cell CMOS, all within a standard ASIC design flow. The objective of this this work in terms of area-delay trade-offs is shown in Figure 3. The figure already takes into account the fact that circuits with completion detection (our work, DIMS and NCLX) work with true delays, and hence faster than static CMOS.

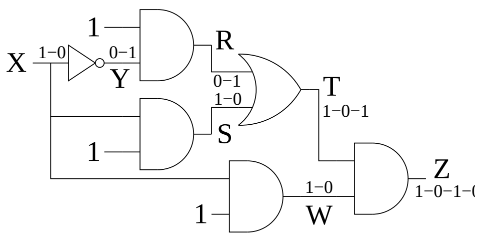


Fig. 4. Hazards in combinational circuits.

IV. Monotonic Boolean networks

A. Hazards

Glitches in combinational circuits are called *hazards* [7]. A hazard is a transient change on a signal caused by the gate propagation delays. Figure 4 shows an example of hazard. The change $1 \to 0$ at input X may produce a *static-1* hazard at signal T and a *dynamic* hazard at signal Z. A sequence of events that produce such behavior is

$$X\downarrow\ S\downarrow\ T\downarrow\ Y\uparrow\ Z\downarrow\ R\uparrow\ T\uparrow\ Z\uparrow\ W\downarrow\ Z\downarrow .$$

To characterize the dynamic behavior of a combinational circuit we need to define the allowed transitions at the primary inputs of the circuit. We will focus on a specific set of transitions that are relevant for the work in this paper.

Definition 4.1 (Monotonic Input Transition (MIT)): A monotonic input transition consists of the change in the value of a subset of primary inputs in the same direction, i.e. all changes are either $0 \to 1$ or $1 \to 0$.

Definition 4.2 (Monotonic operation mode (MOM)): The *monotonic operation mode* works by iteratively alternating two sub-phases, each of which is a Monotone Input Transition (MIT): (a) a subset of the inputs changes monotonically and (b) a subset of the outputs changes monotonically.
In order to use this mode of operation for completion detection, the set of changing outputs must be known in advance. In particular, with dual-rail encoded signals exactly one input and one output in every pair changes value in each phase. Dual-rail operation normally alternates (a) and (b) sub-phases in which half of the inputs and half of the outputs rise in the set phase, with (a) and (b) sub-phases in which all inputs and outputs return to zero in the reset phase. However, MOM and most of the definitions and theorems in this paper do not require dual-rail encoding and are equally suitable, e.g., for 1-of-4 encodings or for dual-rail encodings in which the spacer is encoded 11, rather than 00.

We now define a class of Boolean networks that are hazard-free under MOM.

B. Monotonic Boolean networks

We assume the reader to be familiar with the basic concepts on *Boolean functions* and *Boolean networks*. Each non-input node n_i of a Boolean network has an associated Boolean function f_i in terms of its local fanin.

Definition 4.3 (Unate functions): A function f is *increasing*[2] (*decreasing*) in a variable x_i if changes in x_i from 0 to 1 cannot cause a change in f from 1 to 0 (from 0 to 1). A function is *unate* in a variable if it is increasing or decreasing in that variable.

[2]Past work called this *local* characteristic of a gate "monotonicity". Here we call it unateness, while we use the term "monotonic" for a *global* property of the gate function, with respect to primary inputs.

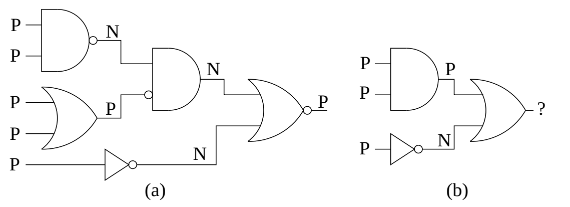


Fig. 5. (a) Monotonic network, (b) non-monotonic network.

Now we introduce the concept of monotonicity of the nodes of a Boolean network. Monotonicity is related to the fact that each node can be assigned a positive or negative phase in the network, which reflects the direction in which its output changes due to a change in value of primary inputs.

Definition 4.4 (Monotonicity): A node n_i with local function f_i is *positive* if for each input x_i in its local fanin it holds that

- x_i is positive and f_i is increasing in x_i, or
- x_i is negative and f_i is decreasing in x_i.

A node n_i with local function f_i is *negative* if for each input x_i in its local fanin it holds that

- x_i is negative and f_i is increasing in x_i, or
- x_i is positive and f_i is decreasing in x_i.

A node is *monotonic* if it is either positive or negative.
While this definition works for any network which admits a consistent labeling of each node (including primary inputs and primary outputs) as positive or negative, for the sake of simplicity in the following we assume that all primary inputs are dual-rail encoded with spacer 00, and hence are defined to be *positive*.

Definition 4.5 (Monotonic and positive Boolean networks): A Boolean network is monotonic (MBN) if all its nodes are monotonic. An MBN is positive (PBN) if all its nodes are positive (e.g. a domino logic network is positive).

Theorem 4.1 (Hazard-free behavior of an MBN): An MBN is hazard-free under a monotonic input transition.

Proof: Let us consider the case in which a subset of inputs changes from 0 to 1. We will prove that the propagated changes can produce at most one transition in each node, which can only be increasing in positive nodes and decreasing in negative nodes. The proof is done by induction on the depth of the nodes.

For level 0 (primary inputs), all nodes are positive and at most one change $0 \to 1$ is produce at each input, by the definition of MIT. Assume that the monotonicity of changes holds up to level $i-1$ and let us analyze the case for a positive node n at level i (the proof is similar for a negative node). By the induction hypothesis, monotonicity holds for any fanin signal of n. If a fanin signal x comes from a positive node, then the function is increasing in x. If a fanin signal y comes from a negative node, then it is decreasing in y (otherwise the network would not be a MBN). The only possible changes in x and y are $0 \to 1$ and $1 \to 0$, respectively. No matter in which order they occur, the only possible propagated change to n is $0 \to 1$, thus preserving the monotonicity up to level i.

The proof for the change $1 \to 0$ at the inputs is similar. ■

Figure 5 depicts a monotonic and a non-monotonic network. The labels P and N indicate the positive and negative nodes, respectively. The network in Fig. 5(b) is not monotonic since it is not possible to assign a phase to the output node. Note that the input transition $000 \to 111$ applied to all inputs may produce a glitch at the output, if the inverter and the OR gate switch before the AND gate.

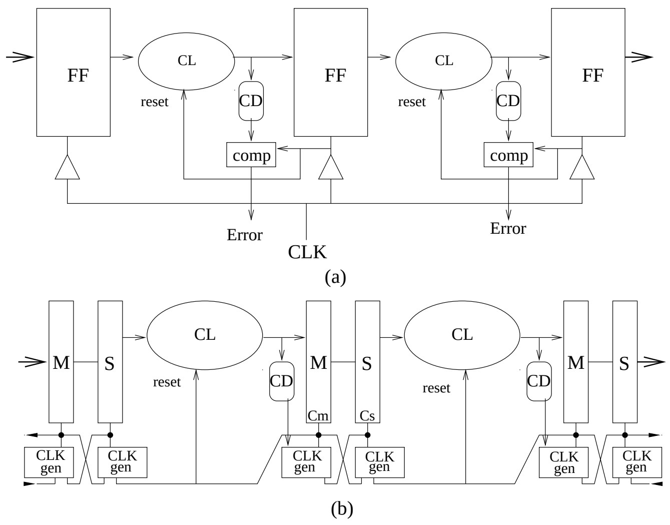


Fig. 6. Architectures with completion detection

C. Use cases

Combinational logic with completion detection can be used in two different environments: synchronous and asynchronous (see Figure 6(a) and (b)).

In the synchronous environment, the output of the completion detector (denoted by CD in Figure 6(a)) is used to check whether a circuit can work at a given clock frequency. If a change at the output of a completion detector happens *before* the clock edge, then data inputs of receiving flip-flops have settled before the clock rises and no synchronization fault occurs. If, however, the completion detector makes a transition *after* the clock edge, then there are chances that erroneous values have been stored in flip-flops, and the *error* signal must be raised.

The *error* signal may be used:

- during production test to bin chips according to their performance
- to provide an on-line testing capability, assuming that the system may roll back and repeat the computation cycle or is capable of stretching the clock.

In order to give a qualitative estimation of the potential performance benefits from using circuits with completion detection, let us compare their cycle time with the cycle time of conventional synchronous designs.

The cycle time in a synchronous circuit must satisfy the following timing constraints:

$CL_{max} + T_{skew} + T_{setup} + T_{CQ\_max} \leq T_{cycle}$ *setup constraint*
$CL_{min} + T_{CQ\_min} \leq T_{hold} + T_{skew}$ *hold constraint*

In these inequalities CL_{max} and CL_{min} stand for worst and best case propagation delays through combinational logic, while T_{CQ} is the clock-to-output delay of a flip-flop.

The cycle time in a circuit with a completion detection also needs to satisfy hold and setup constraints which are:

$CL_{true} + T_{skew} + T_{setup} + T_{CQ\_max} + T_{reset\_max} + T_{CD} \leq T_{cycle}$
$CL_{min} + T_{CQ\_min} \leq T_{hold} + T_{skew}$

where CL_{true} is the true delay of combinational logic, while T_{CD} and T_{reset_max} are the delay overheads of the architecture with completion detection (delays of the completion detector and reset phase respectively).

A circuit with completion detection must satisfy an additional timing assumption in order to function correctly in the reset phase:

$T_{reset\_min} \leq T_{hold}$

where T_{reset_min} is the minimum delay for propagating spacer values to the outputs of the combinational logic. This constraint is similar to the hold constraint and is needed to ensure that spacer does not overwrite data values before they are stored in registers.

The reset signal can be derived directly from the clock, assuming that it has an (asymmetric) duty cycle of T_{reset_max}/T_{cycle}. In this case the reset phase is triggered by the rising edge of the clock and its duration is defined by the width of a clock pulse.

One can see that a synchronous architecture with completion detection provides a performance advantage if

$CL_{true} + T_{reset\_max} + T_{CD} < CL_{max}$

Experimental results give a quantitative estimation of T_{reset_max} and T_{CD}, showing that the conditions of the above inequality are indeed met very often.

In order to use circuits with completion detection in an asynchronous environment, one can exploit standard micropipeline-based architectures [12]. For example, they are suitable for desynchronized circuits [13], which are derived from synchronous synthesizable specifications. The only difference is that the request signals triggering controllers are derived from completion detectors rather than from matched delays, as shown in Figure 6(b).

V. Transformations to obtain and preserve a Monotonic Boolean Network

The questions that we want to address in this section are the following:

1) Given an arbitrary Boolean network, how do we transform it into an MBN with no delay increase and minimum area increase?
2) Given an MBN, which transformations and optimizations can we apply, such that the result is another MBN?

A. Conversion to Monotonic Boolean Network

We can generate an MBN from a generic Boolean network by using a special encoding of inputs and outputs, the dual-rail code, using two different procedures. The first one operates on a technology-independent Boolean network, generates a PBN, and requires technology mapping afterward. The second one operates on a technology-mapped Boolean network and generates an MBN by only using De Morgan's laws. The result of both can also be optimized, by using the techniques listed in Section V-B, which preserve MBN.

Technology-independent (TI) conversion

1) From each primary input x, two primary inputs x^t and x^f are created in the DR circuit to represent the *true* and *false* evaluations of x.
2) From each node implementing the function $y_i = f_i(x_1, \ldots, x_n)$, two nodes are created with the functions

$$y_i^t = \mathsf{DR}\left(f_i(x_1, \ldots, x_n)\right) \qquad y_i^f = \mathsf{DR}\left(\overline{f_i}(x_1, \ldots, x_n)\right)$$

where DR denotes the transformation of the function into positive unate, changing also its input signals from x_i to x_i^t and x_i^f as appropriate. Formally, the transformation DR can be recursively defined as follows, using Shannon's expansion:

$$\mathsf{DR}(0) = 0 \qquad \mathsf{DR}(1) = 1$$
$$\mathsf{DR}(x \cdot f_x + \overline{x} \cdot f_{\overline{x}}) = x^t \cdot \mathsf{DR}(f_x) + x^f \cdot \mathsf{DR}(f_{\overline{x}})$$

As an example, the function

$$y = a\overline{b} + b(c + \overline{d})$$

would be converted into

$$y^t = \mathsf{DR}\left(a\overline{b} + b(c + \overline{d})\right) = a^t b^f + b^t(c^t + d^f)$$
$$y^f = \mathsf{DR}\left(\overline{a\overline{b} + b(c + \overline{d})}\right) = (a^f + b^t)(b^f + c^f d^t)$$

As a particular case, the inverters ($y = \overline{x}$) are eliminated and converted into wires ($y^t = x^f$ and $y^f = x^t$).

Figure 7 depicts a complete example of how a single-rail circuit (a) is converted into a dual-rail circuit (b). After technology mapping, the circuit in Fig. 7(c) can be obtained.

We are now interested in proving that the behavior of a dual-rail circuit is equivalent to that of the single-rail. The following theorem indicates that the dual-rail equivalence of the inputs implies the dual-rail equivalence of the outputs.

Theorem 5.1: Given a function $y = f(x_1, \ldots, x^n)$ and the assumption that $x_i = x_i^t = \overline{x}_i^f$, for any input x_i, then it holds that $y = y^t = \overline{y}^f$.

Proof: By induction on the number of variables in the support of f. It clearly holds for 0 variables, since $\mathsf{DR}(0) = 0$ and $\mathsf{DR}(1) = 1$.

Let us assume that it holds for functions up to $n - 1$ variables and that the support of f has n variables. By choosing one splitting variable x, we have

$$y^t = x^t \mathsf{DR}(f_x) + x^f \mathsf{DR}(f_{\overline{x}})$$

By induction we know that $\mathsf{DR}(f_x) = f_x$ and $\mathsf{DR}(f_{\overline{x}}) = f_{\overline{x}}$, since the size of support of f_x and $f_{\overline{x}}$ is smaller than n. Therefore,

$$y^t = x f_x + \overline{x} f_{\overline{x}} = y.$$

The proof is similar for $\overline{y}^f$. ■

Technology-mapped (TM) conversion

This procedure assumes that each gate used in the technology mapped netlist has a *dual* gate, based on De Morgan's law. This assumption is satisfied by most modern gate libraries.

1) For each gate in the circuit, producing signal y_i^t from signals $y_j^t, \ldots, y_k^t$, add a dual gate using De Morgan's law, producing signal y_i^f from signals $y_j^f, \ldots, y_k^f$. Inverters $y_i = \overline{y_j}$ are simply replaced by wires connecting $y_i^t = y_j^f$ and $y_i^f = y_j^t$.
2) Label each node as positive or negative, starting from primary outputs. Labeling is done as in Section IV-B, only in the opposite direction, in order to perform phase correction at inputs if needed. In case multiple paths of different length reconverge at a node, we give priority to the *longest path*, to which we do not want to add inverters in the following step.
3) For each gate input or primary input which is inconsistently labelled, insert an inverter connecting it to the dual signal. E.g. an input of gate y_i^t connected to signal y_j^t and requiring the opposite phase is connected to $\overline{y_j^f}$ via an inverter which does not change functionality but corrects the phase.

Figure 8 depicts a complete example of how a technology-mapped circuit is converted into a dual-rail circuit. Steps 1 and 2 are shown in Figure 8(b). Levels are labelled from the outputs to the inputs, where P indicates a positive net, and N indicates a negative net. Step 3 is shown in Figure 8(c), where phase correction at the inputs takes place and wires are swapped and inverters added.

Both procedures are conceptually equivalent. However they are different from a practical point of view. The TI-conversion is suitable for early stages of the synthesis flow, in which limited logic synthesis can still be executed on the converted circuits. The TM-conversion is suitable for circuits that have already been mapped and analyzed, in which the designer prefers to introduce as few changes as possible on the core data-path.

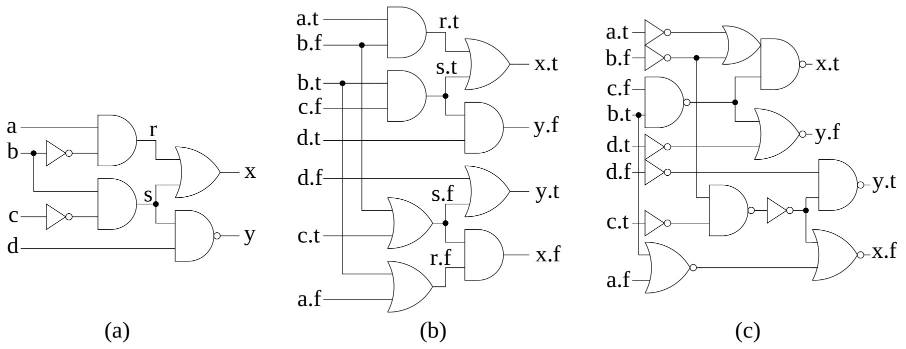


Fig. 7. From single-rail (a) to dual-rail (b), and technology mapping (c).

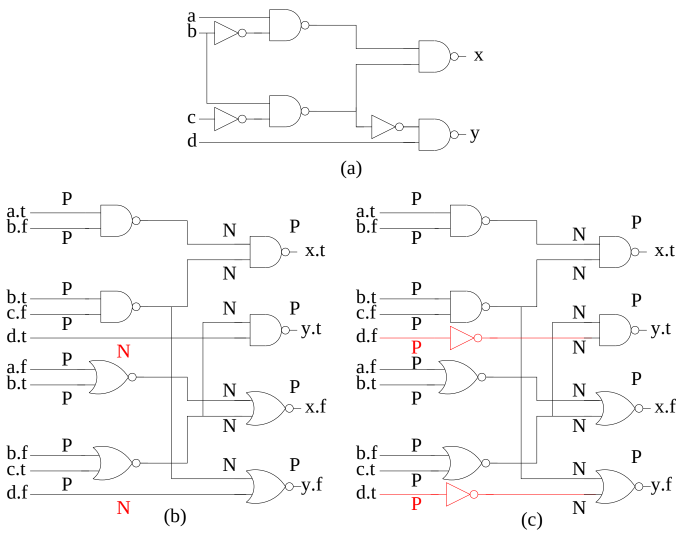


Fig. 8. Steps for technology-mapped circuit conversion: (a) original technology mapped circuit, (b) dual-rail circuit with level labeling and (c) phase correction by introduction of inverters.

B. Hazard-non-increasing transformations

In [14], a set of transformations that do not introduce new hazards in Boolean networks was presented. They extend the set originally given in [7] and include De Morgan's laws, dual global flow, global flow, tree decomposition, gate replication, collapsing, kernel-factoring and cube-factoring. These transformations cover, among others, the conventional algebraic optimizations performed during technology-independent logic synthesis [15].

During technology mapping, the set of transformations applied to the network also usually falls into the previous categories. In particular, technology mappers perform the following transformations:

1) Tree decomposition into 2-input gates.
2) Gate replication of multiple-fanout nodes (e.g. [16]).
3) Collapsing several nodes in order to map them to a library gate.

Additionally, pairs of inverters are inserted in the wires to increase the chances of matching better gates in the library. This insertion also maintains the levelization, and hence the monotonicity, of the network. Some advanced technology mappers also incorporate algebraic transformations [17].

In summary, logic synthesis and technology mapping can be performed on MBNs as long as the set of transformations fall into the category of hazard-non-increasing.

C. Fast reset and completion detection

A circuit obtained using one of the two procedures above would operate at half the speed of its original counterpart, due to the need for resetting all primary inputs before another monotonic phase can begin. We can speed up this reset phase by inserting signals that force gate outputs to their "quiescent" value (*i.e.* to the value that they assume when all dual-rail inputs are at 0, in the spacer condition). Of course there is a trade-off here between the number of gates which are reset in this manner (the more numerous, the faster becomes the reset phase) and the large capacitance connected to the reset signal (which may end up slowing down the circuit too much). We are in the process of evaluating what is the optimal fraction of the total number of levels where one should insert the reset signal, and what optimization algorithms can be used here. Gates with fewer inputs are obviously better candidates, since the delay of a gate increases quadratically with the number of transistors in series.

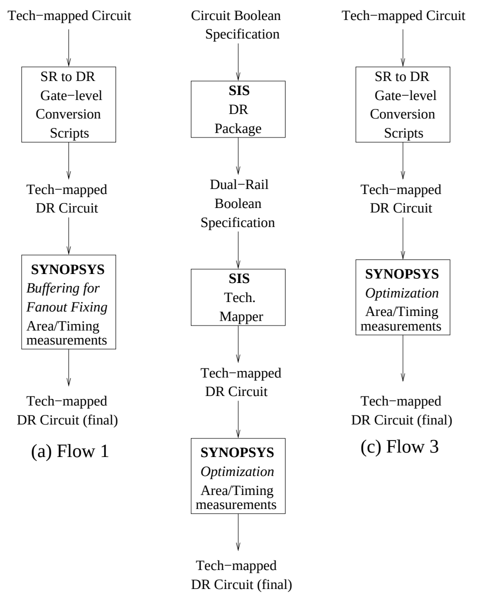
Furthermore, the network that detects when all outputs have stabilized can be built simply by using an `or` gate for each dual-rail pair, whose output rises when one of the signals rises, and a tree of `and` gates.

VI. Results

Both procedures described in Section V have been automated and can be applied to any circuit. The first procedure, which operates at the level of Boolean networks, has been implemented as an add-on package to the SIS synthesis tool [18]. The second procedure, which operates using transformations on a technology-mapped netlist, has been implemented as a sequence of netlist conversion scripts.

Based on these two procedures we implemented a total of three design flows for dual-rail conversion, as shown in Figure 9. All three flows are compatible with commercial EDA tools. In fact the last step in all three flows is to feed the circuit into Synopsys Design Compiler^{TM}. This tool is used in order to: a) appropriately buffer nets in order to drive the fanout, b) re-optimize the circuit after technology mapping and c) perform timing analysis and area estimation.

The three flows were applied to the largest SIS combinational benchmark circuits and to a set of larger Verilog RTL circuits, *i.e.* the pipeline stages of a DLX CPU and a pipelined DES stage. All circuits were mapped to the UMC 0.18μm technology library. Due to the relatively small size of the circuits considered (fewer than 100,000 gates) we are ignoring the effects of layout on the performance. We are aware that in principle, for large flat designs, the area increase will result also in a delay increase. However, we are expecting that dual-rail conversion for delay measurement will be used only on relatively small, critical portions of the overall circuit. The final version of the


Tech−mapped Circuit
SR to DR Gate−level Conversion Scripts
Tech−mapped DR Circuit
SYNOPSYS Buffering for Fanout Fixing Area/Timing measurements
Tech−mapped DR Circuit (final)
Circuit Boolean Specification
SIS DR Package
Dual−Rail Boolean Specification
SIS Tech. Mapper
Tech−mapped DR Circuit
SYNOPSYS Optimization Area/Timing measurements
Tech−mapped DR Circuit (final)
Tech−mapped Circuit
SR to DR Gate−level Conversion Scripts
Tech−mapped DR Circuit
SYNOPSYS Optimization Area/Timing measurements
Tech−mapped DR Circuit (final)

(a) Flow 1

(b) Flow 2

(c) Flow 3

Fig. 9. Flow Implementations: (a) TM-conversion with Fanout Fixing after Dual-Rail transformation, (b) TI-conversion with Optimization after technology-mapping, and (c) TM-conversion with Optimization after Dual-Rail transformation

paper, if accepted, will contain post-layout delay and power figures for some representative benchmarks.

Flow 1 implements the TM-conversion using netlist conversion scripts. At the end of the conversion process of the circuit to dual-rail, the circuit requires fanout fixing, as inverters which were originally used for buffering were removed during the conversion process. Thus, optimizing the fanout drive of the gates of the dual-rail circuit is necessary. Fixing fanout, while preserving the circuit structure was implemented in Synopsys DC by setting all circuit gates as "don't touch" and then compiling the circuit for minimum input to output delay. In this way, DC is only allowed to insert buffers to fix maximum input to output delay without touching the original netlist. This approach is much more effective than imposing fanout constraints on the design. Flow 1 results are shown in Table VI. The shorthands SR, DR, DR_R and DR_CD in the column labels correspond to Single-Rail, Dual-Rail, Dual-Rail with Fast Reset, and Dual-Rail with Fast Reset and Completion Detection respectively. Percentage increase is computed as $100 \times (DR - SR)/SR$ for both area and delay.

Flow 2 implements the TI-conversion using the add-on package written for the SIS synthesis tool. In Flow 2, fanout fixing is not necessary, because the conversion is performed at the level of Boolean network, thus SIS will fix fanout during technology mapping. After SIS technology mapping, we move the circuit into Synopsys DC and let it re-optimize and re-map the circuit. In that way, DC is given full power to apply its own optimization transformations on the mapped dual-rail netlist and then re-map it to optimize the timing of the circuit. Although this process is potentially hazardous, as it cannot be guaranteed that the transformations and optimizations imposed on the dual-rail netlist are hazard-non-increasing, we verified that in practice for the circuits we tried, monotonicity was preserved after optimization for timing by Synopsys DC.

Flow 3 is identical to Flow 1 except for the last step. Synopsys DC is allowed to re-optimize the circuit and technology map it as in Flow 2, instead of only fixing fanout as in Flow 1.

The summary about the penalty in design flows 1, 2 and 3 with respect to conventional single-rail implementation is shown in Table II.

As can be seen from Table II, the delay penalty for the *data phase* of the dual-rail circuits, indicated by columns DR and DR_R, is relatively small. The penalty due to completion detection, which is currently larger, can be reduced by considering it only for critical outputs. The area penalty is about two-fold. Figure 10 shows the delay and the area of the benchmark circuits, relative to the optimized and mapped single-rail original, *i.e.* in the same form as Figure 3. The dots are spread around the 2x area, 1x delay point with respect to traditional combinational logic. However, one should note that the delay results are *worst-case*, while the delay of our circuits can be measured exactly, and thus they will work closer to the *average case*.

As dual-rail circuits operate using the two-phase discipline, the *reset phase* delay must also be taken into account. For the circuits presented here, with reset employed every 6 circuit levels, the reset time is between 0.16ns and 0.22ns.

VII. Conclusions

This paper proposes a novel technique to create circuits with completion detection, based on a dual-rail encoding. We describe both theory ensuring correct hazard-free operation, and implementations working both at the technology-independent and at the technology-dependent levels. The experimental results show that a 100% area and power increase and a 30% delay increase are sufficient to obtain a circuit that is fully able to signal when its outputs have stabilized.

Bearing in mind the margin of 60-100% between worst case and true delays, one can conclude that circuits with completion detection could operate at 25-65% higher clock frequencies than traditional ones.

This enables for the first time a trade-off between applying completion detection to the whole design, and achieving the 25-65% speedup with 100% area and power penalty, versus applying it only to the most critical stages of logic, and gaining less speed with significantly smaller penalty in area and power. Exploring this trade-off is left to future work.

References

[1] S. Devadas and K. Keutzer, "Synthesis of robust delay-fault testable circuits: Theory," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 87–101, Jan. 1992.

[2] "User guide," in *Celtic User Manual, Cadence Design Systems, Inc.*, 2004.

[3] "Datasheet," in *Physical Studio Datasheet, Sequence Design, Inc.*, 2003.

[4] S. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. of Asia and South Pacifi c Design Automation Conference*, May 2001.

[5] D. Chinnery and K. Keutzer, "Reducing the timing overhead," in *Closing the Gap between ASIC and Custom: Tools and Techniques for High-Performance ASIC design*. Kluwer Academic Publishers, 2002, ch. 3.

[6] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Addison-Wesley, 1980, ch. 7.

[7] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, John Wiley & Sons, Inc., 1969.

[8] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 313–340, Oct. 1993.

[9] A. Kondratyev and K. Lwin, "Design of asynchronous circuits by synchronous CAD tools," in *Proc. ACM/IEEE Design Automation Conference*, June 2002.

[10] M. Prasad, D. Kirkpatrick, R.Brayton, and A. sangiovanni Vincentelli, "Domino logic synthesis and technology mapping," in *Proc. International Workshop on Logic Synthesis*, vol. 1, 1997.

[11] R. Puri, A. Bjorksten, and T. Rosser, "Logic optimization by output phase assignment in dynamic logic synthesis," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1996, pp. 2–8.

Flow 1 DC Buffering only	SR delay (ns)	SR area (μm^2)	DR delay increase	DR area increase	DR_R delay increase	DR_R area increase	DR_CD delay increase	DR_CD area increase
SIS Benchmarks								
C1908	2.07	9709	-4%	79%	-0%	91%	12%	99%
C2670	1.53	13592	3%	79%	9%	93%	34%	121%
C3540	2.59	16665	7%	79%	14%	90%	23%	94%
C5315	2.08	33737	6%	77%	40%	90%	58%	103%
C7552	2.16	35294	26%	81%	32%	100%	50%	111%
alu4	1.85	9079	0%	80%	4%	90%	13%	93%
apex6	0.89	11006	11%	91%	15%	107%	57%	139%
dalu	1.34	10998	-2%	84%	2%	94%	18%	99%
des	1.46	51910	34%	73%	35%	99%	63%	108%
frg2	0.94	10965	3%	67%	5%	82%	45%	116%
i10	2.34	32489	5%	80%	41%	96%	59%	113%
i2	0.73	3062	9%	94%	16%	102%	21%	105%
i7	0.71	8754	-40%	62%	-39%	72%	8%	96%
i8	1.05	13149	-11%	86%	-7%	95%	25%	115%
i9	1.09	9612	-5%	77%	-0%	85%	29%	109%
k2	1.18	14942	0%	87%	4%	102%	30%	112%
pair	1.49	21544	6%	86%	12%	98%	37%	116%
rot	1.27	9132	9%	79%	13%	95%	42%	138%
vda	0.96	9059	0%	90%	6%	99%	35%	108%
x3	0.72	10669	-2%	91%	4%	203%	55%	236%
x4	0.61	5749	-3%	72%	3%	86%	59%	116%
RTL Circuits								
DLX_IF	0.82	18340	-4%	55%	-1%	69%	47%	89%
DLX_ID	4.01	158709	7%	68%	11%	86%	24%	103%
DLX_EX	3.46	121404	-2%	88%	5%	103%	16%	105%
DLX_MEM	0.50	3294	-18%	64%	-16%	94%	58%	184%
DES_stage	1.61	27704	-12%	74%	-8%	89%	18%	109%
average	1.52	25791	3%	77%	11%	95%	33%	109%

TABLE I

FLOW 1: TM-CONVERSION AND FANOUT FIXED BY SYNOPSYS DC.

Flow Comparison	SR delay	SR area	DR delay	DR area	DR_R delay	DR_R area	DR_CD delay	DR_CD area
Flow 1	1.52	25791	3%	77%	11%	95%	33%	109%
Flow 2	0.91	32008	8%	126%	9%	162%	46%	173%
Flow 3	0.91	32008	-6%	77%	-5%	85%	31%	97%

TABLE II

SUMMARY OF FLOWS 1,2 AND 3.

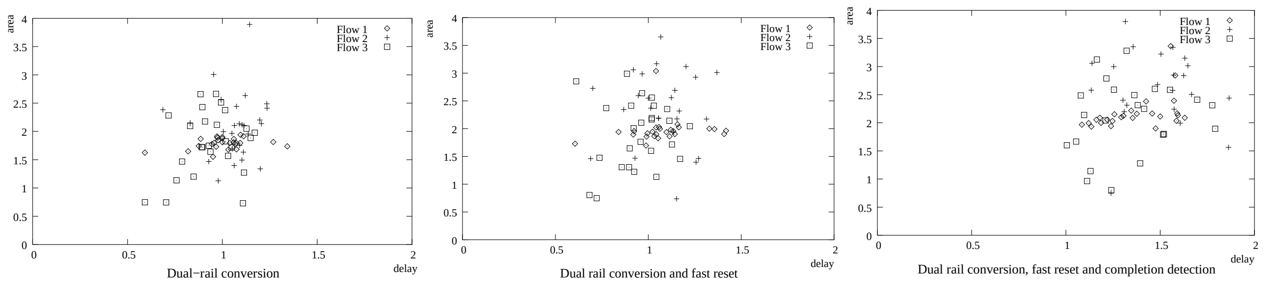


Fig. 10. Delay and area relative to single-rail CMOS circuits

[12] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[13] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Apr. 2004, pp. 149–158.

[14] D. S. Kung, "Hazard-non-increasing gate-level optimization algorithms," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1992, pp. 631–634.

[15] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[16] Y. Kukimoto, R. Brayton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Design Automation Conference*, 1998, pp. 348–351.

[17] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 8, pp. 813–834, Aug. 1997.

[18] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," U.C. Berkeley, Tech. Rep., May 1992.