Area Optimizations for Dual-Rail Circuits Using Relative-Timing Analysis

Tiberiu Chelcea, Girish Venkataramani, Seth C. Goldstein
School of Computer Science
Carnegie Mellon University
email: {tibi,girish,seth}@cs.cmu.edu

Abstract

Future deep sub-micron technologies will be characterized by large parametric variations, which could make asynchronous design an attractive solution for use on large scale. However, the investment in asynchronous CAD tools does not approach that in synchronous ones. Even when asynchronous tools leverage existing synchronous toolflows, they introduce large area and speed overheads. This paper proposes several heuristic and optimal algorithms, based on timing interval analysis, for improving existing asynchronous CAD solutions by optimizing area. The optimized circuits are 2.4 times smaller for an optimal algorithm and 1.8 times smaller for a heuristic one than the existing solutions. The optimized circuits are also shown to be resilient to large parametric variations, yielding better average-case latencies than their synchronous counterparts.

1 Introduction

The vast majority of existing EDA flows target synchronous circuits. However, with the emergence of deep-submicron technologies, synchronous circuits exhibit several problems. First, it is projected that the parametric variance of device delay may approach 35% by 2020 [2], which means that the clock period must be slowed for the worst case variation, resulting in performance losses. Second, clock distribution is becoming increasingly difficult. The ITRS'05 projects that chips will be more GALS-oriented and that, by 2020, up to 40% of the chip will communicate using asynchronous handshaking to avoid problems in clock distribution [2].

As a consequence, asynchronous logic may become a more attractive solution, since it can naturally deal with parametric variance. However, a widespread acceptance of asynchronous logic is predicated on the existence of quality CAD tools. A promising way to address the lack of CAD tools for asynchronous design is to re-use existing synchronous tools, and translate their results into asynchronous circuits. Several papers explore this approach ([4, 3, 12, 14, 11]): synchronous circuits are translated into quasi delay-insensitive (QDI) circuits. QDI circuits are synthesized assuming that gates and wires have fixed, but unbounded, delays. They are tolerant to operational variations (delays, voltage, temperature), but also have large area overheads. These overheads are introduced to avoid the *orphans problem* (stale data inside the circuit), which might result in data hazards and deadlock. Usually, this problem appears because dual-rail combinational logic may exhibit the *early propagation effect*: producing primary outputs even before all primary inputs have arrived.

The classical approach to solving the orphan problem is to introduce completion detection (CD), which makes the circuit resilient to the early propagation effect. However, as shown in Section 8, the fraction of area occupied by the CD circuitry can be as high as 89% of total circuit area. Even though modern chips can accommodate a large number of transistors, area overheads usually translate into speed overheads. To reduce these overheads, we propose using relative-timing analysis [15] to optimize these asynchronous implementations by reducing the size of the CD logic. One such system where relative-timing optimizations played a crucial role is the Intel Asynchronous Instruction-Length Decoder [13], which was several times faster than its synchronous counterpart.

This paper introduces several novel relative-timing algorithms that optimize asynchronous circuits for area. In addition, we also show that these circuits are tolerant to parametric variation and compare favorably with corresponding synchronous designs.

The contributions of this paper are:

- A synthesis flow that starts with synchronous designs, translates them into asynchronous circuits, and improves them by incorporating relative-timing analysis in the optimization process.
- Three new optimization methods which improve the area of dual-rail circuits. One is a direct application of

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

timing analysis of dual-rail circuits, while the remaining two (one heuristic and one optimal) further modify the implementation in order to reduce area. The optimal algorithm produces circuits that are, on average, 2.4x smaller than existing solutions, while the heuristic one produces circuits 1.8x smaller.

- An exploration of the effects of parametric variation in dual-rail circuits, which shows that asynchronous circuits are more resilient than synchronous ones when device delays are probabilistic.

The paper is organized as follows. Some relevant background in asynchronous design and existing toolflows is provided in Section 2. Section 3 describes the relative timing analysis methodoloy used for deducing delay intervals of the circuit. Using the results of this analysis, Sections 4, 5 and 6 propose three optimizations (two heuristic and one optimal) for improving circuit area. We discuss related work in Section 7, report experimental results in Section 8, and conclude in Section 9.

2 Background

2.1 Asynchronous design styles

Asynchronous circuits do not have a global clock signal, and, instead, implement synchronization between components using local signals. The typical solution is to communicate data on *channels*, which consist of data wires and one or more control signals. There are several choices for the communication protocol on these channels; the most widely used is *4-phase handshaking*, which consists of an active phase (in which data is sent to the receiver), and a return-to-zero phase (in which all channel signals reset to zero).

There are several possible delay insensitive encodings of data items on channels. The most widely used style is *dual-rail*, in which a data bit A is encoded on two-wires (A_1 and A_0), and each channel has only one control signal, an acknowledge. To send a value of '1', (A_0,A_1) becomes (0,1), and to send a value of '0', (A_0,A_1) becomes (1,0). This communication style is very robust to gate and wire delays, but has large area demands. In addition, the existing synthesis tools for synchronous circuits need to be modified to implement dual-rail modules. However, as we will show in this paper, we believe that the robustness of these circuits is the answer to large parametric variations in future deep sub-micron technologies.

Figure 1 shows the synchronous and dual rail implementations of a simple function (Z=(A·B)·(C+D)). It is clearly apparent from this figure that dual-rail implementations have large (roughly 2x) area overheads compared with synchronous implementations.

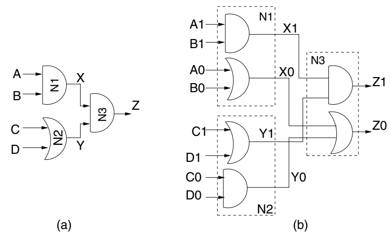

Figure 1. *Synchronous (a) and dual-rail (b) implementations of the function Z=(A·B)·(C+D).*

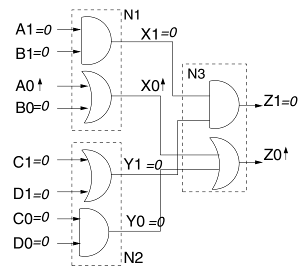

Figure 2. *A case of the orphan problem: upon the early arrival of A=0, Z becomes zero even before inputs B, C and D have arrived.*

2.2 Early Propagation and Orphans

The *early propagation effect* in a dual-rail circuit means that the circuit can output a value (a) before all primary inputs have arrived and/or (b) all internal gates have reached a final, stable value. It must be noted that this effect can manifest in both the active phase of the handshaking (when computing a result based on new data inputs), and on the reset to zero phase (when reseting the circuit in anticipation of new data items).

The *orphans* problem is closely related to the early propagation effect, but not limited to it. A dual-rail circuit is said to exhibit wire or gate orphans iff a signal transition on a wire or a gate is unobservable (or un-acknowledged).

Figure 2 illustrates both the early propagation effect and the orphans problem in dual-rail implementations. Suppose that a '0' is received on A ($A_0 = 1$ and $A_1 = 0$) long before the values on B, C, and D are received. Then, the circuit produces the correct final value Z=0 ($Z_0 = 1$ and $Z_1 = 0$). However, at this point, signals B_0 and B_1 become wire orphans, and gate "N2" becomes a gate orphan: their transitions do not change the final value of Z, and are thus un-acknowledged.

In dual-rail circuits, orphans may lead to incorrect behavior. For example, in the above case, the environment receives a value on Z and can acknowledge. Thus, the A value may be reset-to-zero even before B, C, or D are re-

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

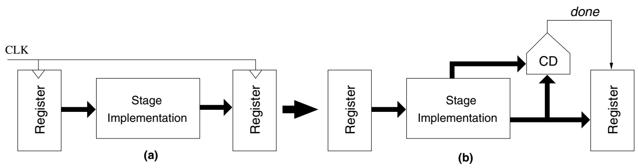

Figure 3. *NCL-X architecture: each synchronous stage (a) is transformed into an asynchronous implementation with explicit completion detection (b).*

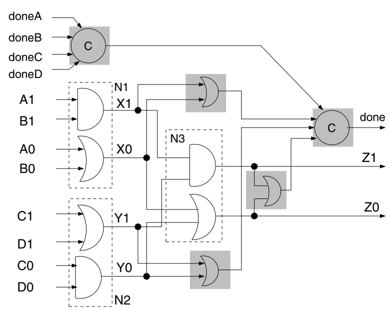

Figure 4. *NCL-X implementation of Z=(A·B)·(C+D).*

ceived, which may result in deadlocks or data hazards. The same scenario may occur during the reset-to-zero phase: if Z is seen as reset, a new wave of computation may be sent to the circuit. At this point, if there are orphan gates (i.e. not fully reset), data hazards occur since these gates may compute on stale data.

2.3 NCL-X Synthesis Flow

In [11], the authors have introduced an automatic toolflow which simultaneously converts synchronous circuits into asynchronous ones, and solves the problem of orphans in these circuits. Their flow starts from RTL specifications, which are synthesized with synchronous CAD tools. These implementations are then converted in a template-based fashion (i.e. gate-by-gate) into a Null-Convention Logic (NCL) [9] implementation. The final circuit is then augmented with a completion detection (CD) circuit which detects when all the gates in the circuit have reached a stable value, thus eliminating all orphans.

The target architecture of [11] is shown in Figure 3(b). The NCL-X implementation is separated into stage functions and state (registers that store computed values). The register implementation is fixed. The implementation of stage functions is a direct template-based translation of the synchronous implementation.

In more detail, the authors of [11] augment the asynchronous circuit with an extra "done" signal, which becomes '1' during the active phase of communication when all gates have computed their stable value, and '0' during the reset phase when all the gates have correctly reset. This is achieved by associating a CD to each gate: a simple OR gate whose inputs are the two wires of the dual-rail signal. Then all these individual CD signals are fed into a C element[1], which produces the "done" signal. This implementation is very robust, but introduces large area overheads.

In addition, the area problem is compounded by the lack of C elements in commercial synchronous standard gate libraries. Thus a 2-input C element is usually implemented with 4 NAND gates (18 transistors), whereas an optimal, transistor-level implementation costs only 8 transistors [16]. Furthermore, each C element with more than 2 inputs must be decomposed into trees of 2-input C elements. Thus, the goal of our optimizations is to minimize the number of C-elements required in the CD, without sacrificing robustness.

3 Relative-Timing Analysis of Dual-Rail Circuits

This section introduces the main ideas behind area optimizations proposed in this paper. We use relative timing analysis to deduce the interval timing properties of the combinational circuit, and thus reduce the size of the completion detection.

3.1 Timing Analysis

The basic assumption behind QDI circuits is that gates and wires have unbounded, unknown delays; the only timing assumption permitted is the *isochronic fork* – whenever a wire forks to two destinations, the delays on the forks are equal. In constrast, our approach assumes that the delay of gates and wires are bounded by given time intervals: $\delta_G = (t_G^m, t_G^M)$ for a gate, and $\delta_W = (t_W^m, t_W^M)$ for a wire. These intervals represent the lower- and upper-bound delays for propagating an input change to the output. These delays can either be obtained from standard-cell library characterizations or can represent the theoretical limits of parametric variation.

Given δ_G and δ_W for all gates and wires, we can easily compute the *propagation delays* ($\Delta_G = (T_G^m, T_G^M)$) for each gate in the circuit. The propagation delay of a gate in a circuit C is the time interval when the gate might output a value, once the circuit is applied a set of inputs. If a gate G has N inputs and the inputs arrival times are $\Delta_{I_k} = (T_{I_k}^m, T_{I_k}^M), k \in 0..N$, then gate G produces the output within $\Delta_G = (MIN(T_{I_k}^m) + t_G^m, MAX(T_{I_k}^M) + t_G^M)$. That is, the gate will produce an output at some point between the earliest possible input transition (minimum of

[1] A C element has N inputs and one output; its output becomes 1 when all inputs are 1, and 0 when all inputs are 0.

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

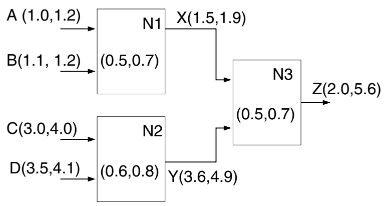

Figure 5. *Timing example for Z=(A·B)·(C+D).*

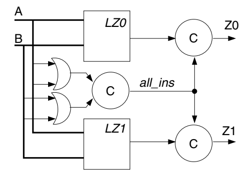

Figure 6. *Implementation of a 2-input strict dual-rail gate.*

lower bounds) and the latest possible input transition (maximum of upper bounds), plus the gate's delay. A similar formula is defined for the propagation delay on wires in the circuit. Thus, starting with gate and wire delays and the primary input arrival times, the timing characterization of the circuit can be efficiently performed.

In addition, for a given circuit C, we define a global measure, the *global propagation delay*, which is the time interval in which *all* the circuit's outputs (primary outputs – PO) are produced:

$$GlobalPD(C) = (\max(T_o^m), \max(T_o^M)), \ \ \forall_{o \in PO}$$

The formula simply says that sometime between the maximum of all outputs' lower bounds and the maximum of all outputs' upper bounds, all outputs are available to the environment. $GlobalPD$ acts as a propagation delay interval for the entire circuit.

These concepts are illustrated in Figure 5, which shows the same circuit from Figure 1 without the internals of each dual-rail gate. The gate propagation intervals are set for illustration purposes; for simplicity, the wire delays are assumed to be zero. For this particular example, since the circuit has a single output, $GlobalPD$ is the propagation delay of Z.

It is important to notice that, in performing this analysis, the logic function of each gate is not considered: instead, we assume that any input change (either rising or falling) at any time can change the output of a gate, thus modeling the early propagation effect.

There are two main reasons for not considering the logical function of gates. First, the C element is the only gate that waits for all inputs to transition from 0 to 1 *and* from 1 to 0 (i.e. corresponding to the active and passive phases of operation), before generating a transition on the output; this gate does not have a combinational synchronous counterpart, and it will thus not occur in our circuits. Second, it is well known that an exact min-max analysis (taking into account the logic function as well as the delays of each gate) is intractable [10], and only approximate bounds can be realistically computed [7]. Since our method does not consider the logic functions of each gate, the time complexity of analysis is linear in the number of gates.

Furthermore, by conflating both the rising and falling transition delays into a single measure (gate propagation delay), a single optimization step eliminates orphans in both the active and the return-to-zero phases of circuit operation.

3.2 Strict Dual-Rail Gates

So far, we have considered only regular dual-rail gates, that can produce early outputs. An alternative implementation is to use strict gates, which eliminate the early propagation effect at its output, and thus reduces the number of orphans. A strict gate waits for all its inputs to arrive before producing an output. Strict gates are also referred to as input-complete in the literature [17]. Effectively, a strict gate acts as an implicit completion detection for all the input fan-in, making them extremely expensive area-wise. However, the propagation delay interval at the output of a strict gate is narrower, and thus they can potentially reduce the size of the CD needed at the circuit output.

Figure 6 shows the implementation of a generic strict gate with two inputs. The LZ0 and LZ1 boxes correspond to the gates producing Z_0 and Z_1 in the regular implementation. The strict gate first detects whether the inputs have arrived (one OR gate for each gate input), and then synchronizes all these signals through a C element; the arrival of all inputs is indicated on *all_ins*. Finally, for each of Z_0 and Z_1, the strict gate synchronizes *all_ins* with the result of the computation.

An N-input strict gate has N extra OR gates and N+1 extra C elements compared with the corresponding early implementation, which is a significant overhead. The gate delay of a strict gate δ_S is determined by a race condition between computing the result in LZ0/LZ1 and determining whether all inputs have arrived, plus the propagation interval of a C element.

The propagation delay of a strict gate is $\Delta_S = (MAX(T_{I_k}^m), MAX(T_{I_k}^M)) + \delta_S$ (i.e. the interval between the *latest* minimum arrival time of all the inputs and the *latest* maximum arrival time of all the inputs, plus the gate delay of S). The output interval tends to be narrower since Δ_S considers only the *latest* minimum arrival time of its inputs, whereas a Δ_R for its regular counterpart considers the *earliest* minimum arrival time of its inputs.

Returning to our example, suppose that gate N3 is made strict; the propagation interval of N3 is now (1.4,1.9) (i.e.

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

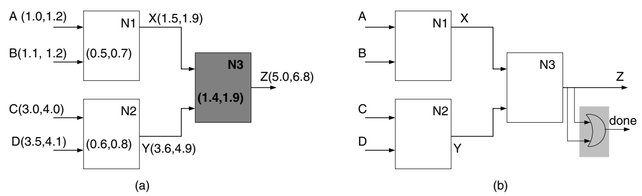

Figure 7. *The implementation of Z=(A·B)·(C+D) with strict gates.*

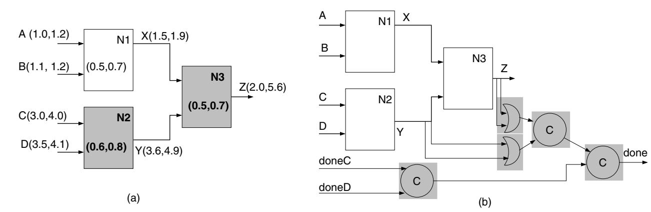

Figure 8. *Optimized CD for Z=(A·B)·(C+D): Direct Method*

the delay overhead of strictness is (0.9–1.2)). The new latencies through the circuit are shown in Figure 7a (strict gates, such as N3, are shown in darker shades). Notice that now, $GlobalPD$ has shrunk to 1.8 (down from 3.6) and the input arrival intervals of C and D do not overlap with the output interval (and thus do not need CD). In addition, since gate N2 produces the inputs to the strict gate N3, it does not need CD – it is implicitly acknowledged by the outputs of N3. The final implementation of the circuit is shown in Figure 7b.

The entire CD for the circuit is reduced to a single OR gate, but this particular implementation has larger area than the one in Figure 8 (strict gates are large). This is the basic area trade-off that our optimizations focus on. As we show in Section 8, since 89% of the circuit area of an NCL-x circuit is occupied by CD, strict gates can be instrumental in reducing this area overhead.

However, over the next three sections, several algorithms will use strict gates to reduce area.

4 Direct Optimization

The first of the proposed optimization algorithms, called "direct", reduces the area of CD using interval analysis, but does not attempt to modify the implementation of the circuit. Given the propagation delay of a gate Δ_G and the global propagation delay $GlobalPD$ of a circuit, our key observation is that a gate G is not stable when all the primary outputs are produced iff $T_G^M \geq GlobalPD^m$. The formula simply indicates that, if the upper bound of a gate's propagation delay is larger than the earliest time when all the primary outputs are produced, the gate may still have not produced an output, and needs CD.

Thus, the "direct" method for building CD starts by computing $GlobalPD$ as shown above. Then, for each gate, $g \in 0..NGates$, the method sets $NeedsCD$ as follows:

$$NeedsCD_g = \begin{cases} 1 \text{ if } T_g^M \geq GlobalPD^m \\ 0 \text{ if } T_g^M < GlobalPD^m \end{cases}$$

Figure 8a shows the result of such analysis for the running example. The shaded boxes represent the gates for which $NeedsCD = 1$. Note that the upper bound for the propagation delay of N1 (1.9) is less than $GlobalPD^m$ (2.0). In addition, the arrival times for the A and B inputs are also less than the global propagation delay, and thus need not be used in producing the global "done" signal. The resulting implementation is shown in Figure 8b. Compared with Figure 4, this implementation saves three C elements and one OR gate (remember that C elements are implemented as trees).

5 Greedy Optimization

The "direct" method presented in 4 performs a simple timing analysis of the circuit, and then determines whether the dual-rail gates need completion detection or not. While this is useful, more sophisticated optimization is to weigh the trade-offs between the use of regular and strict gates, thereby finding an implementation with reduced area.

Strict gates usually have narrower propagation delays than their regular counterparts; they also provide implicit acknowledgment to their input gates, which no longer need completion detection. However, they are much larger than regular gates. Thus, an optimization algorithm needs to balance the tension between all these characteristics.

In this subsection, we introduce a greedy heuristic algorithm, which attempts to narrow the $GlobalPD$ propagation delay of the circuit. The hope is that a narrower $GlobalPD$ will result in less overlaps with propagation delays of each gate, thus yielding a smaller global completion detection. The pseudo-code for the greedy algorithm is presented in Figure 9.

The algorithm starts by performing the "direct" method (Section 3); the output interval becomes the current best solution. Each iteration of the optimization loop consists of looping through all the gates in the circuit, and converting each one to 'strict', applying the "direct" method, computing $GlobalPD$ for the new circuit, and reverting the gate to 'regular'. The best solution (i.e. the narrowest $GlobalPD$ interval) is saved. At the end of each step, the current best solution is compared with the previous best; if the previous solution was larger, the algorithm continues; otherwise, the algorithm exits the optimization loop and computes the completion detection (as in Subsection 4).

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

```
greedy (circuit C) {
    C = OPT_direct(C); // perform the "direct" analysis
    CrtBest = GlobalPD(C); // set the initial best solution
    forever {
        PrevBest = CrtBest; // save the current best solution
        for i ∈ 0..N {
            if (G_i is strict) continue;
            make gate G_i strict;
            OPT_direct(C); // perform the "direct" analysis
            CrtSol = GlobalPD(C);
            if (CrtSol < CrtBest) {
                G_best = G_i; // save current best candidate
                CrtBest=CrtSol; // save current best solution
            }
        }
        if (PrevBest narrower than CrtBest) break;
        make G_best strict;
    }
    compute CD for C using CrtBest interval
}
```

Figure 9. The greedy optimization algorithm

The time complexity of this algorithm is $O(N^2)$, where N is the number of gates in the circuit. To improve the running time, the circuit can be topologically sorted. Then, when flipping gate i, the algorithm can perform timing analysis only for gates $i+1 \ldots N$, since those are the only ones that may be affected by the current change in gate i. At the end of each iteration, once a gate has been switched permanently to strict, a full application of the "direct" method is performed, to set up the initial values for the next iteration.

It is important to notice that the proposed "greedy" algorithm indirectly minimizes area by narrowing the $GlobalPD$ interval. Thus, the algorithm cannot guarantee that it will not introduce area overheads. In practice, however, due to the large area of the CD circuitry, the price paid for making some gates strict is more than compensated by the reduction in CD area, as we show in Section 8.

6 Optimal mILP Algorithm

This section introduces an optimal formulation of this optimization problem, which finds the minimum number of gates that must be strict such that total circuit area is minimized. We create a mixed Integer Linear Programming (mILP) formulation of the optimization problem. The first subsection describes the variables, constraints, and objective function of the base formulation. Since some of the constraints in the formulation are logical, the next subsection describes how they are linearized. Finally, the last subsection presents certain application-specific techniques to speed up the branch-and-bound phase of solving the mILP problem.

6.1 Base mILP Formulation

Given a circuit graph, we first insert pseudo-nodes to represent the primary inputs (PI) and primary outputs (PO) of the circuit. The inputs to the formulation are:

- `PI`, the primary inputs.
- `PO`, the primary outputs.
- `GT`, the gates in the circuit.
- `C=(V=PI ∪ PO ∪ GT, E)`, the circuit graph.
- $\forall i \in PI, \mathtt{PDI} = (\mathtt{PDI}_i^m, \mathtt{PDI}_v^M)$: the propagation delays (arrival times) of the primary inputs.
- $\forall v \in V, \mathtt{ND} = (\mathtt{ND}_v^m, \mathtt{ND}_v^M)$: the gate delay of the regular version.
- $\forall v \in V, \mathtt{SD} = (\mathtt{SD}_v^m, \mathtt{SD}_v^M)$: the gate delay of the strict version.
- $\forall (i,j) \in E, \mathtt{WD} = (\mathtt{WD}_{i,j}^m, \mathtt{WD}_{i,j}^M)$: the delays on the wires in the circuit
- $\forall v \in V, \mathtt{NArea}_v$: the area of the regular gate.
- $\forall v \in V, \mathtt{SArea}_v$: the area of the strict gate.
- `CArea`, `OR2Area`: the area of a 2-input C-element and of a 2-input OR gate, respectively.

Given these inputs, the optimization problem finds an assignment to a set of binary variables, $GS_v, \forall v \in V$, such that the area of the circuit is minimized. When $GS_v = 1$, it implies that node v is to be strict, otherwise it is regular. In equations 1–6, we will describe all the constraints in the ILP formulation.

$$\begin{aligned} GDm_{v\in V} &= GS_v \times \mathtt{SD}_v^m + (1-GS_v) \times \mathtt{ND}_v^m \\ GDM_{v\in V} &= GS_v \times \mathtt{SD}_v^M + (1-GS_v) \times \mathtt{ND}_v^M \end{aligned} \tag{1}$$

The two variables GDm_v and GDM_v represent the delay interval of gate $v \in V$. They depend on the state of the gate: if the gate is "strict" the delays are the minimum (maxi-

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

mum) gate delays for the strict gate, otherwise they are the minimum (maximum) gate delays for the regular gate.

$$
\begin{aligned}
LBm_{v\in V} &= \begin{cases} \texttt{PDI}^m, & \text{if } v \in PI, \\ \text{Min}_{\forall(u,v)\in E}(\Delta_u^m + \texttt{WD}_{u,v}^m), & \text{otherwise.} \end{cases} \\
LBM_{v\in V} &= \begin{cases} \texttt{PDI}^m, & \text{if } v \in PI, \\ \text{Max}_{\forall(u,v)\in E}(\Delta_u^m + \texttt{WD}_{u,v}^m), & \text{otherwise.} \end{cases} \\
UBM_{v\in V} &= \begin{cases} \texttt{PDI}^M, & \text{if } v \in PI, \\ \text{Max}_{\forall(u,v)\in E}(\Delta_u^M + \texttt{WD}_{u,v}^M), & \text{otherwise.} \end{cases}
\end{aligned} \tag{2}
$$

These three variables compute the propagation delay properties for a given gate, $v \in V$. Specifically, they compute the earliest lower bound (LBm_v), the latest lower bound (LBM_v), and the latest upper bound (UBM_v) of the propagation delay from the primary inputs of the circuit to the inputs of gate v. While Eq. 2 computes the propagation delay at the input of the gates, the following constraints determine the propagation delay interval, at the outputs of the gates.

$$
\begin{aligned}
\Delta_{v\in V}^m &= \begin{cases} LBm_v + GDm_v, \text{if } GS_v = 0 \\ LBM_v + GDm_v, \text{if } GS_v = 1 \end{cases} \\
\Delta_{v\in V}^M &= UBM_v + GDM_v
\end{aligned} \tag{3}
$$

The lower bound of the propagation delay is either the earliest arrival time of the inputs (if gate is regular) or the latest arrival time of the inputs (if gate is strict), plus the minimum gate delay (see Eq. 1). The upper bound is always the latest arrival time of the inputs, plus the the maximum gate delay.

Now, the lower bound of the propagation delay for the entire circuit (see Subsection 3.1) can be computed:

$$
GlobalPD^m = MAX_{\forall v\in PO}(\Delta_v^m) \tag{4}
$$

As defined in 3.1, $GlobalPD^m$ is the earliest time when all the primary outputs could be produced. This helps us determine which gates need to participate in completion detection. For each gate, we define this decision using a binary variable, $NeedsCD$, as described below:

$$
\forall NeedsCD_{v\in V} = \begin{cases} 0, \text{if } GlobalPD^m > \Delta_v^M \bigvee \\ \quad \exists\, u \mid GS_u = 1 \wedge (v,u) \in E \\ 1, \text{otherwise.} \end{cases} \tag{5}
$$

A gate v does not need completion detection ($NeedsCD_v = 0$) if the upper bound of its propagation delay is smaller than $GlobalPD^m$ (i.e. the gate is stable before the earliest time when all primary outputs are produced), or if one of gate v's successors is a strict gate, which provides implicit completion detection. $NeedsCD$ is 1 otherwise.

Based on these constraints, we can now infer the total area of the circuit, which we want to minimize:

$$
\begin{aligned}
&GA_{v\in V} = GS_v \times \texttt{SArea}_v + (1 - GS_v) \times \texttt{NArea}_v \\
&GateArea = \textstyle\sum_{v\in V} GA_v \\
&SCD = \textstyle\sum_{v\in V} NeedsCD_v \\
&CDArea = SCD \times \texttt{OR2Area} + (SCD - 1) \times \texttt{CArea} \\
&TotalArea = GateArea + CDArea
\end{aligned} \tag{6}
$$

The decision variables, GS_v and $NeedsCD_v$ tell us if gate v should be strict and if v needs completion detection at its output. These decisions in turn determine the size of the gates, and thus overall circuit area, which is the sum of: (1) total gate area, $GateArea$, determined by decisions made by strict variables, GS_v for each v, and (2) the CD circuit area. For the latter, we assume that the CD is constructed as a tree of 2-input C-elements, with a two-input OrGate at the leaves of the tree, whose inputs are the dual-rail signals (see Figure 8b). SCD computes the number of gates which need completion detection, and represents the number of leaves in the CD tree. $CDArea$ uses SCD to compute the size the of the CD tree.

This completes our mILP formulation. The objective function is to minimize $TotalArea$ in Eq. 6. However, some of the constraints in this formulation are logical, e.g., Eq. 2, Eq. 3, Eq. 4, Eq. 5, implying that they are not linear. If all the variables are bounded (as they are in our formulation), such logibal constraints can be transformed into intenger linear constraints.

6.2 Linearizing Logical Constraints

In general, a pair of logical OR constraints can be linearized by introducing a decision variable, as long as the variables are bounded. For example, consider the equality constraints that assign MIN/MAX terms to variables, such as Eq. 2 and Eq. 4. A MAX function, $z = MAX(u,v)$ can be written as a pair of logical OR constraints as $\{z \geq u, z \geq v, u \geq z\} \vee \{z \geq u, z \geq v, v \geq z\}$. A standard technique to linearize these constraints is to introduce a decision variable that adds a large number, say M, to the smaller side of an inequality constraint, thereby making them inactive. For example, using the decision variable, α, the MAX function can be written as the constraints set, $\{z \geq u, z \geq v, u + M \times \alpha \geq z, v + M \times (1-\alpha) \geq z\}$. Thus, we have conditionally (based on value of α) added the large number M to the smaller side of each inequality constraint. Using similar techniques, we can linearize the logical constraints in Eq. 3, yielding:

$$
\begin{aligned}
&\Delta_v^m + GS_v \times M \geq LBm_v + GDm_v \\
&\Delta_v^m - GS_v \times M \leq LBm_v + GDm_v \\
&\Delta_v^m + (1 - GS_v) \times M \geq LBM_v + GDMin_v \\
&\Delta_v^m - (1 - GS_v) \times M \leq LBM_v + GDMin_v
\end{aligned} \tag{7}
$$

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

While such a technique linearizes the logical constraints, it also introduces additional decision variables and constraints to the formulation. This could be avoided through problem-specific knowledge. For example, in Eq. 5, we know that the variable $NeedsCD_v$ will be minimized, since it always leads to an increase in the objective function. Using this knowledge, Eq. 5 becomes:

$$GSDst_v = \sum_{(v,u)\in E}(GS_u)$$
$$\Delta_v^M - (NeedsCD_v + GSDst_v) \times M \leq GlobalPD^m \quad (8)$$

With these new constraints, the problem is specified as a pure mILP formulation, which is then provided as input to an ILP solver such as `cplex` [1].

The M quantity used in linearizing the formulation is technology and circuit dependent, and it has to be larger than the largest propagation delay through a circuit in a given technology. In our experiments, M is set to be the largest delay of any of the gates in the circuit under consideration, multiplied by the number of gate levels in the circuit.

6.3 Improving the ILP Model

The ILP model presented above is convex and thus it has a global minima representing the optimal solution. However, ILP formulations are unstable by their very nature, and can have very long run times, since they are exponential in the number of variables. Our initial experiments indicated that even circuits with 20-30 gates may take hours to optimize. However, we can leverage our knowledge of the problem to improve its running time by introducing several techniques which either reduce the size of the problem (by setting some variables to their final, optimal values), or provide hints to the ILP solver as to where to look first for an optimal solution. With these changes, circuits with several hundred gates can be optimized in a matter of seconds.

Single Input Gates. Gates with only one input (inverters, buffers) cannot be strict in the optimal solution. For such a gate, an output transition is implicitly an acknowledgement of a transition on the single input, which is exactly the behavior of a 'strict' version. Since the area of a strict version is much larger, there is no benefit in turning these gates into 'strict', and the ILP formulation is augmented with constraints that set GS to zero.

Level zero gates. In our proposed implementation, the primary inputs to the circuit are collected in a C-element, which indicates that all the inputs have arrived (Figure 4). Thus, all the inputs to the gates whose inputs are only primary inputs, (i.e. on level zero) are implicitly acknowledged, so turning them into strict gates does not help. For all these gates, we set GS to zero in the ILP formulation.

$NeedsCD$ **Elimination.** In some cases, it is possible to guarantee that some gates will not have completion detection in the optimal solution. This is the case for every gate that meets all of the following conditions: (a) has as predecessors only gates which will not be 'strict' in the final solution, (b) does not need completion detection after the "direct" analysis (Section 4), and (c) even if the gate is made strict, the gate still does not need completion detection. Basically, condition (a) guarantees that input arrival times do not change in the optimal solution, condition (b) checks whether the gate needs CD now, and condition (c) checks whether, under the most adversarial condition (gate becoming strict), the gate needs CD. Thus, by fulfilling these conditions, a gate will not need completion detection in the optimal solution, and $NeedsCD$ can be set to zero for these gates in the ILP formulation.

Branching Order. The ILP solver implements a branch-and-bound algorithm, which can be controlled by several parameters. There are several knobs the user can control to improve the solver runtimes in the branch and bound phase. One such knob is variable order when branching. In our case, we have noticed that in the optimal circuits, the strict gates tend to be concentrated in the upper levels, closer to the primary outputs. Therefore, in our ILP model, we set the GS priorities (i.e. order of branching) proportional to the level of the gate. This way, the solver will first inspect the gates near the primary outputs.

Branching Direction. We also define a priority when exploring the branching direction, i.e., for the GS variables, should the solver first branch on a value of 1 or 0. Following the observation that strict gates are most effective near the inputs of the circuit, we set the branching direction to 1 for gates in the top 25% levels, and to 0 for rest of the gates.

Initial Solution. The user can provide the ILP solver with an initial integer solution, which usually speeds up the search for optimal. In our case, we provide the solver the result of the "greedy" method, since this one results in smaller area than "direct", and thus closer to optimal.

7 Related Work

Several approaches have been proposed for translating synchronous circuits into asynchronous ones [4, 3, 12, 14, 11, 5]. Our toolflow is similar to these, but the main thrust of our approach is different: we provide relative-timing optimization algorithms to reduce the area overheads typically introduced by such toolflows.

Kondratyev et al. [11] propose a method to deal with orphans. Their solution is discussed in detail in Section 2, and improved on by our proposed methods.

In [14], Sokolov investigates the circuits of [11] in the context of secure applications and proposes two relative-timing optimizations. In the first one, only the gates in alternate levels have completion detection. The method implicitly assumes that the gates in a level have roughly equal

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

latencies, which is not the case in real circuits, and thus does not solve the orphans problem. The second method introduces completion detection only for the gates on the longest path through the circuit; however, finding this path when considering the logical functionality of the gates is intractable [10], and this optimization is applicable only for extremely small circuits.

An alternative solution is presented in [5]. Their circuits allow for early propagation, and compute the completion detection in the background. This scheme allows their circuits to be used with a special register implementation style (anti-token), which may take better advantage of early outputs than [11, 14]; however, this solution suffers from even larger area overheads. Our relative-timing solutions can be applied to optimizing these circuits too.

Another area optimization of dual-rail circuits is presented in [8], also in the context of translating synchronous circuits into asynchronous ones. Their translation method considers the logica function of a circuit, from which it derives the equations for each dual-rail output using Shannon's decomposition, and then technology maps the result using only hazard non-increasing transformations. By allowing some timing constraints, the completion detection logic can be improved to just a tree of AND gates. In our approach, we start with a technology mapped bundled data circuit, and translate each gate into a dual-rail gate in a template-based fashion. The optimization algorithms are applied to the dual-rail circuit; however, these algorithms do not make any assumptions on how the dual-rail circuit was derived, and can thus be used to optimize the circuits obtained using [8].

Finally, the authors of [17] also use strict and regular dual-rail gates to optimize the area of the circuit. They cast the optimization as a binate covering problem. The crux of their optimization algorithm is to determine an assignment of gates that are strict, such that all gates are acknowledged and area is minimized. The circuits they synthesize are still QDI (the only timing constraint is the isochronic fork). The mILP formulation proposed in this paper subsumes the formulation in [17] by solving a much harder problem. However, the circuits we synthesize are no longer QDI, but tolerate a parameterizable bounded delay interval.

8 Results

8.1 Methodology

In order to verify our proposed methods, we have implemented the toolflow to automatically synthesize optimized dual-rail circuits from their single-rail equivalents. First, a Verilog description of a synchronous (single-rail) circuit is synthesized with Synopsys Design Compiler (SDC). The result is a technology mapped implementation in the ST Micro 0.18μ technology. This implementation is processed by several custom Perl scripts, which replace each STMicro gate by a corresponding dual-rail implementation. The resulting dual-rail implementation is read back into SDC, which computes and writes the gate delays file (SDF). The dual-rail Verilog implementation, and the corresponding SDF file are then read into our custom timing tool, which then performs the analysis. This timing tool also receives as a parameter the device delay variation, which is used to modify the delays from the SDF files. The "nclx", "direct", and "greedy" algorithms are custom implemented, while the "ilp" algorithm is implemented with the aid of a commercial ILP software, the `cplex` callable library [1].

All experiments are pre-layout. It is known that circuit modifications may change the gate and wire timing after layout. However, by using a hierarchical place and route tool (such as Cadence First Encounter), the range of these changes can be bounded, and hence quantified in the model. Therefore, we can account for layout effects in our analysis by incorporating these delay ranges directly in our analysis, without any modifications to the algorithms.

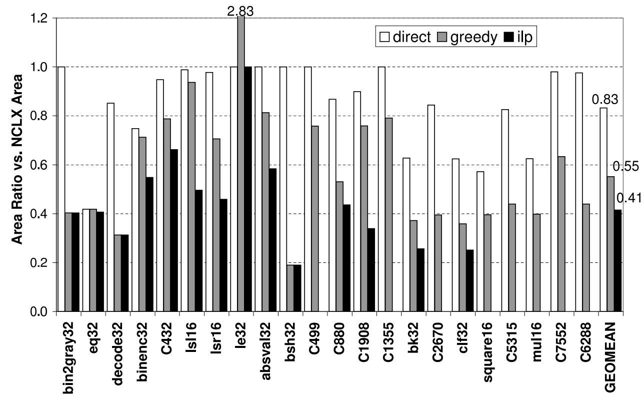

Figure 10. *Ratio of area in our proposed methods vs. the NCL-X method.*

8.2 Area

Using the methodology described above, we have synthesized and simulated several typical arithmetic functions (two types of 32-bit adders, 32-bit comparators, equality, and absolute value circuits, left- and right-shifters, a 16 bit multiplier and squarer, several binary and gray encoding circuits), as well as a few ISCAS'89 [6] benchmarks (C432, C499, C880, C1908, C2670, C5315, C6288, and C7552). For a variation of 0% (i.e. the device delays are the same as in the SDF files), we have applied all of the four methods described in this paper (NCL-X, direct, greedy, mILP), and the results are presented below.

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

Benchmark	#Inps	#Outs	#Gates	NCLX Run Time	Direct Run Time	Direct Area	Greedy Run Time	Greedy Area	mILP Run Time	mILP Area	mILP #Vars	mILP #Constr.
bin2gray32	32	32	32	0.13	0.08	1.000	0.06	0.403	0.15	0.403	672	1128
eq32	64	1	37	0.17	0.12	0.418	0.12	0.418	0.23	0.407	731	1158
decode32	5	32	49	0.10	0.09	0.851	0.11	0.313	12.12	0.313	1239	2068
binenc32	32	6	58	0.14	0.11	0.748	0.17	0.713	0.26	0.548	1239	1903
C432	36	7	80	0.24	0.21	0.948	0.29	0.788	1666.29	0.662	2391	4223
lsl16	32	16	81	0.10	0.18	0.988	0.88	0.937	624.98	0.496	1819	3534
lsr16	32	16	81	0.16	0.50	0.977	0.16	0.705	1155.23	0.459	2315	4080
le32	64	1	91	0.21	0.23	1.000	0.10	2.830	0.61	1.000	1571	2852
absval32	32	32	92	0.18	0.15	1.000	0.50	0.813	367.67	0.583	2420	4149
bsh32	37	32	96	0.31	0.27	1.000	0.42	0.190	7.54	0.190	2880	5181
C499	41	32	162	0.53	0.38	1.000	0.51	0.758	X		4065	7430
C880	60	26	168	0.40	0.37	0.868	1.24	0.531	2365.29	0.436	4385	7724
C1908	33	25	190	0.54	0.53	0.899	1.90	0.759	1223.17	0.339	3263	5300
C1355	41	32	220	0.34	0.40	1.000	1.57	0.791	X		4530	8393
bk32	64	32	285	0.24	0.57	0.627	1.22	0.372	79.32	0.256	4923	8293
C2670	157	64	290	0.76	0.47	0.844	10.16	0.395	X		8409	14239
clf32	64	32	309	0.69	0.45	0.624	0.91	0.358	71.30	0.251	5195	8737
square16	17	32	529	1.20	0.67	0.572	19.54	0.396	X		7923	15436
C5315	178	123	536	1.09	1.43	0.825	59.22	0.440	X		15239	27443
mul16	32	32	715	2.76	2.75	0.625	32.82	0.398	X		9977	19540
C7552	207	108	832	2.86	2.11	0.980	103.25	0.439	X		18223	33305
C6288	32	32	1200	3.66	3.70	0.976	205.60	0.552	X		28271	55511
Geometric Mean:						**0.833**		**0.552**		**0.415**		

Table 1. *Benchmark circuits optimized with the methods proposed in the paper. The table shows the problem size, running times, and the area ratio between the proposed methods and NCL-X. For ILP, the number of variables and constraints for the ILP formulation are also shown.*

Figure 10 shows the area ratio of the circuits optimized with our proposed methods vs. the NCL-X circuits. These designs have between 32 (bin2gray32) and 1200 (C6288) dual-rail gates (Table 1), with an average of 278 gates. The running time for the optimization software (see Table 1) on a machine with a 2.4GHz Pentium 4 processor and 512MB RAM is a matter of seconds for the "nclx" and "direct" methods, while the "greedy" algorithm runs in at most 3 minutes for the largest design (though most commonly less than 20 seconds). For the ILP formulation, we have set a cut-off time of 2 hours. The benchmarks marked with "X" did not produce an optimal value in the alloted time, while the others require 1 second to 39 minutes. Table 1 also includes the number of variables and the number of constraints in the ILP model for the respective benchmark.

On average, the direct method improves area by a factor of 1.2, greedy by 1.8, and mILP by a factor of 2.4. Notice however that "greedy" can sometimes increase the area overhead (for "le32", the circuit becomes 2.83 times larger than NCL-X!), since the method optimizes for the narrowest propagation interval, and not directly for area.

Compared with synchronous designs, dual-rail circuits are much larger. However, on average, the optimal method (mILP) produces circuits that are only 3.45 times larger than synchronous ones. In contrast, the average for NCL-X is

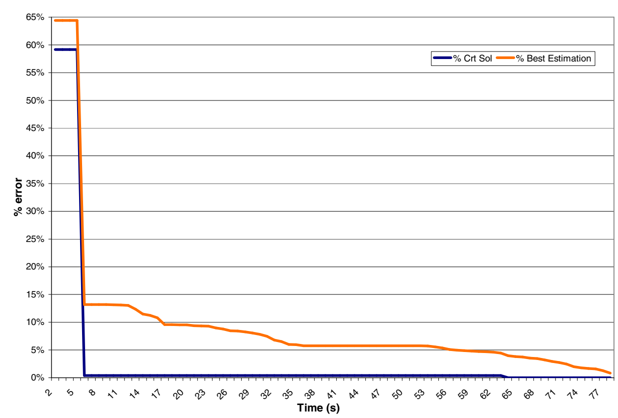

Figure 11. *ILP optimization profile for a 32-bit Brent-Kung adder*

8.3x. It should be noted that the smallest theoretical ratio achievable by dual-rail circuits is around 2x.

Figure 11 shows the behavior of the ILP solver when optimizing the 32 bit Brent-Kung adder; this behavior is typical for all the problems that have been solved. The solver maintains two values, which are shown here: the current best integer solution, and the largest provable interval where an optimal solution might exist, as a displacement from the

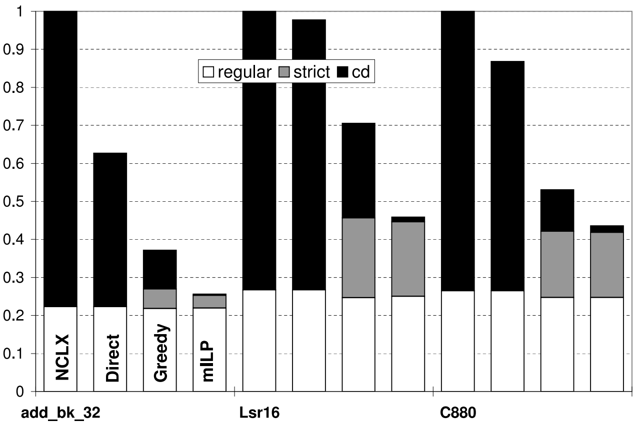

Figure 12. *Area breakdown into regular dual-rail gates, strict dual-rail gates, and completion detection for each method. The Y axis is normalized to the area of NCL-X.*

current value (the "Best Estimation" shape). To fit the two quantities on the same graph, we have plotted the current integer solution as a percentage displacement from the optimal solution for the problem ("Crt Sol"). The optimization ends when "Best Estimation" becomes 0% – the solver knows that there are no better solutions besides the current best. However, as seen in Figure 11, a near optimal (0.40% off) solution is found immediately, but the solver spends an extra minute slightly improving the solution and proving its optimality. For all practical purposes, this is wasted time, and in the future we would like to improve our ILP formulation with cuts to stop the search.

Finally, Figure 12 shows the area breakdown into three main components: regular dual-rail gates, strict dual-rail gates, and the completion detection, for three benchmarks. For the NCL-X style, the CD occupies almost 80% of the total circuit area (across all benchmarks, the high is 89%). This is essentially wasted area, whose only role is to detect whether the circuit has achieved a stable state. For the direct method, this overhead reduces to about 40% (about 50% across all benchmarks), while for the greedy and mILP methods it could be as low as 1%. Also, notice that the area occupied by strict gates can become quite significant. For example, in "C880" (mILP), there are only 8 strict gates out of 168, but their area is roughly 40% of the total area; however, the total area for this method is 56% better than NCL-X, and 13% better than greedy.

8.3 Speed

In addition to characterizing area results, we have also performed several experiments on the speed of the circuits optimized by the proposed methods. These experiments allow us to quantify the effect of parametric variation on our circuits.

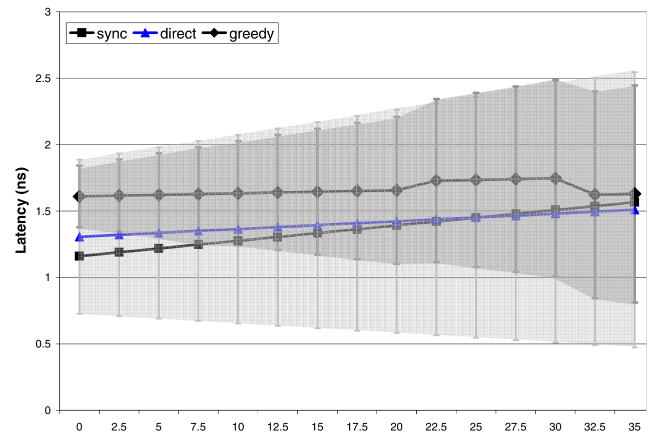

Figure 13. *Latency of a Brent-Kung adder in the presence of parametric variation.*

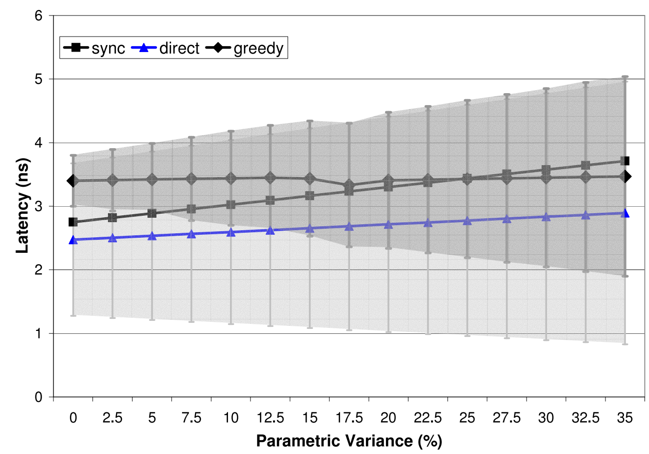

Figure 14. *Latency of a 16-bit multiplier in the presence of parametric variation.*

We have selected two representative circuits, a 32-bit Brent-Kung adder, and a 16 bit multiplier. For each of these circuits, the parametric variation was incremented from 0% to 35% (the projected ITRS variation [2]), in steps of 2.5%.

Figure 13 and Figure 14 show the expected delays for these two benchmark circuits. We are showing only two methods since the delays for NCL-X circuits are very close to those for the "direct" method, and the mILP method does not finish for the multiplier in the alloted time of 2 hours. The synchronous delays increase linearly with the parametric variation, since these delays have to account for the worst-case behavior. In contrast, the delays for the asynchronous implementations are not fixed, but are expressed

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE

as an interval for producing the global "done" signal (lighter shade for "direct" and darker shade for "strict"). In addition, for easy visualization, we have also plotted the middle of the output interval. Since these circuits are not governed by a clock, they can naturally take advantage of variability in device delays.

Figure 13 and Figure 14 show another important characteristic of dual-rail implementations. The slope of the midpoint output interval is sub-linear for both "direct" and "greedy" optimizations when parametric variation increases. In contrast, the delay of the synchronous implementation increases linearly with parametric variation. This indicates that, in addition to guaranteed correctness, using asynchronous circuits for deep-submicron technologies with parametric variation may also result in speed advantages. However, this result needs to be further refined by more detailed simulations of the circuit using statistical methods (e.g. Monte-Carlo) to detect the true average-case delay of the circuit, and not just the midpoint of the output interval.

9 Conclusions

With the emergence of deep submicron technologies, synchronous design becomes increasingly difficult, as parametric variance in future circuits require large pessimistic bound in clock cycles and clock distribution. In fact, ITRS predicts that asynchronous design is going to become more commonplace in future technologies. However, to gain acceptance, asynchronous design needs better CAD tools.

In this paper, we are building on a promising direction in asynchronous CAD tools: adapting synchronous tools for asynchronous design. Previous solutions exhibited large area and delay overheads. The methods presented in this paper alleviate these overheads by using timing information. In addition, in the presence of parametric variation, these optimized circuits are more resilient and may be faster than their synchronous counterparts.

References

[1] Ilog cplex, http://www.ilog.com/products/cplex/.

[2] International technology roadmap for semiconductors, 2005.

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

[4] I. Blunno and L. Lavagno. Automated synthesis of micropipelines from behavioral verilog HDL. In *ASYNC*, pages 84–92, 2000.

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

[6] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Intl. Symp. on Circuits and Systems*, pages 1929–1934, Portland, OR, May 1989. IEEE Computer Society Press.

[7] S. Chakraborty, D. L. Dill, and K. Y. Yun. Min-max timing analysis and an application to asynchronous circuits. *Proceedings of the IEEE*, 87(2):332–346, Feb. 1999.

[8] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. Coping with the variability of combinational logic delays. In *Proceedings of ICCD'04*, 2004.

[9] K. Fant and S. Brandt. Null convention logic: A complete and consistent logic for asynchronous digital circuit synthesis. In *Proceedings of the International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 96)*, pages 261–273, 1996.

[10] N. Ishiura. *Studies on Logic Simulation and Hardware Description Languages*. PhD thesis, Kyoto University, 1990.

[11] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous cad tools. *IEEE Transactions on Design and Test of Computers*, pages 2–12, 2002.

[12] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *ASYNC*, pages 114–123, 2000.

[13] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID: An asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 60–70, Apr. 1999.

[14] D. Sokolov. *Automated Synthesis of Asynchronous Circuits using Direct Mapping for Control and Data Paths*. PhD thesis, University of Newcastle-upon-Tyne, 2006.

[15] K. Stevens, S. Rotem, S. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken. Cad directions for high performance asynchronous circuits. In *Proceedings of the Design and Automation Conference*, pages 116–121, 1999.

[16] I. Sutherland. Micropipelines: Turing award lecture. *Commun. ACM*, 32 (6):720–738, June 1989.

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

13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07)
0-7695-2771-X/07 $20.00 © 2007 IEEE