Synthesis of PCHB-WCHB Hybrid Quasi-Delay Insensitive Circuits

Chi-Chuan Chuang[1], Yi-Hsiang Lai[1], and Jie-Hong R. Jiang[1,2,3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, 10617, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei, 10617, Taiwan
[3]Computer Science Laboratory, Tomsk State University, Tomsk, 634050, Russia
{chichuan327@hotmail.com, r02943086@ntu.edu.tw, jhjiang@ntu.edu.tw}

ABSTRACT

The increasing cost paid in clocking integrated circuits and combating timing variations forces designers to rethink asynchronous approaches to system realization. Among various techniques, quasi-delay-insensitive (QDI) design is promising due to its very relaxed timing assumption. Its expensive logic overhead, however, often nullifies its promise of performance and power improvements, and remains a major obstacle against its adoption. To overcome this obstacle, this paper proposes an efficient static performance analysis procedure and a synthesis flow for precharged half buffer (PCHB) and weak-conditioned half buffer (WCHB) circuit optimization. Experimental results demonstrate efficient performance analysis and effective area reduction under pipeline cycle time constraints.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*automatic synthesis*

General Terms

Algorithms, Logic Synthesis, Verification

Keywords

Asynchronous Pipeline, Half Buffer, Quasi-Delay Insensitivity, Static Performance Analysis

1. INTRODUCTION

Asynchronous approaches to system construction gradually gain their relative practicality due to the increasing cost in synchronizing modern nanometer integrated circuits under various sources of timing uncertainty. Asynchronous circuits are well known for their potential advantages in terms of elimination of clock tree and its power consumption, flexibility in pipelining, reduction on electromagnetic interference (EMI) and IR drop, reusability, robustness against timing variability, and other benefits. Despite these benefits, asynchronous designs remain not popular largely because of its lack of design automation tools and substantial area overhead, among other issues [1, 17].

Depending on their delay models, asynchronous circuits may vary in their underlying timing assumptions. Well-known delay models, in the descending order of timing robustness, include delay insensitive (DI), quasi-delay insensitive (QDI), speed independent (SI), and burst mode circuits [6]. Among them, the QDI model is the most robust and yet practical one. Under the QDI model, asynchronous design can be made close to the standard synchronous design flow [6, 16]. Nevertheless, large area overheads arise due to the conservative timing assumption of QDI circuits.

There are different QDI design styles, e.g., delay insensitive minterm synthesis (DIMS) [13], pre-charged half buffers (PCHB) [7, 11], null-convention logic [4], etc. Among them, PCHB can be practical. In fact, there have been commercial asynchronous designs (e.g., Intel ethernet switch, Achronix FPGA, etc.) based on PCHB implementation. This paper focuses on the synthesis of PCHB and, its close relative, weak-conditioned half buffer (WCHB) circuits. In particular we consider performance constrained area minimization for hybrid PCHB and WCHB circuits.

The analysis and synthesis of asynchronous circuits can be challenging in distinct respects, compared to synchronous designs. Performance analysis of asynchronous pipelines may involve sophisticated computation on timed marked graphs, e.g., finding shortest paths [15], solving linear programming [8], plotting throughput versus token number graphs [7], analyzing time separation of events [12], simulating pipelines, etc. Analyzing complex pipelines involving different protocols may require heuristic methods [18]. Moreover, not all prior methods can provide criticality information for circuit optimization. On the other hand, performance optimization of asynchronous pipelines can be expensive, too. For example, slack matching of full buffer pipelines may require solving mixed integer linear programming (MILP) [1]. The computation may not well scale to large designs. On the other hand, slack matching for half buffer pipelines awaits accurate solutions [3].

In contrast to prior work, this paper focuses on *acyclic*[1] PCHB and WCHB pipelines. We show that, for *data independent* token flow, performance analysis (specifically cycle time computation) can be done by *linear time* traversal over the underlying delay graph. The computation is efficient, similar to static timing analysis (STA) in synchronous design. It also provides criticality information essential for circuit performance improvement, and thus can be powerful supporting incremental synthesis, which is crucial to enable large scale optimization.

For circuit optimization, on the other hand, we propose a synthesis flow transforming a gate-level logic netlist to a hybrid PCHB-WCHB circuit. Slack matching is applied to meet target cycle time (throughput) constraints, and WCHB replacement is applied for area recovery. Essentially the performance advantage of PCHB and the area advantage of WCHB can be leveraged to achieve improved pipeline design. In fact, for a design with complex pipeline structures, non-critical pipeline modules may exist to some extent and area recovery through WCHB replacement can be substantial. Experimental results show efficient timing analysis and effective area-performance optimization.

2. PRELIMINARIES

[1]Although only acyclic pipelines are considered in this paper, they may arise naturally in the design automation flow of converting synchronous to asynchronous design.

DAC'14, June 01 - 05, 2014, San Francisco, CA, USA.

2.1 Quasi-Delay Insensitive Model

The *delay insensitive* (DI) model makes no timing assumption on the gate and wire delays in a circuit, and is the most robust delay model in asynchronous design. It is, however, impractical since only very limited functions can be realized under this model. On the other hand, the *quasi-delay insensitive* (QDI) model is a robust and yet practical delay model in asynchronous design. It imposes no timing assumption, similar to the DI model, except for some designated wire forks (or fanouts), called *isochronic forks*, where the delays to the ends of a fork are assumed to be the same. This assumption can be easily satisfied as the isochronic forks are often localized within standard cell modules. Compared with circuits under other delay models, such as *speed-independent* (SI) circuits, where wires are assumed of ideal zero delay, and *self-timed* circuits, where certain timing conditions are assumed for correct operation, QDI circuits are easier to design without sophisticated timing verification and are robust against timing variability. This paper focuses on QDI circuit synthesis.

2.2 4-Phase Dual-Rail Protocol

QDI asynchronous pipelines are commonly implemented with a 4-phase dual-rail protocol. In a dual-rail encoding system, a 1-bit data d is encoded by a pair (d_0, d_1) of wires, whose valuation $(0, 0)$ represents a NULL (i.e. empty value) state, and $(1, 0)$ and $(0, 1)$ represent VALID 0 and VALID 1 states, respectively. The NULL and VALID valuations alternate to form a 4-phase protocol in a communication channel between a sender and a receiver. The protocol proceeds with the following 4-phase cycle: The sender sends a VALID (0 or 1) signal, the receiver acknowledges receipt of the signal, the sender clears the channel to NULL, and finally the receiver resets the acknowledgement.

2.3 Half Buffers

In a pipeline design, a module is called a *full buffer* if its input and output channels can hold different data tokens at the same time. On the other hand, if the input and output channels of the module can hold only one data token, it is called a *half buffer*. Although a full-buffer pipeline design can be more concurrent than its half-buffer counterpart, they are more complicated to realize.

This paper considers two common types of half buffer templates, namely, the weak-conditioned half buffer (WCHB) and pre-charged half buffer (PCHB).

2.3.1 Weak-Conditioned Half Buffer

For a WCHB, its output being in a NULL (respectively VALID) state implies all its inputs being in NULL (respectively VALID) states. As a result, it suffices to detect only the output state to know the input states. To satisfy the criterion that output validity implies input validity, the evaluation blocks can be realized in a minterm expansion form. On the other hand, the criterion that output nullity implies input nullity is satisfied by stacking the PMOS transistors controlled by the input wires on the pull-up network. These criteria make input completion detection circuitry unnecessary and mitigate area overhead in WCHB.

In the WCHB operation, a sender buffer starts to precharge to enter the NULL state only if all its inputs are in NULL states and its receiver buffer has finished evaluation (i.e. in a VALID state). On the other hand, a sender buffer starts to evaluate to enter the VALID state only if its receiver buffer has finished precharging (i.e. in a NULL state).

The operation of a three-stage WCHB pipeline can be analyzed with the *signal transition graph* (STG). There are two critical cycles that determine the *cycle time* τ_{WCHB} (the time between the generation of two successive tokens) of the pipeline [3]. That is,

$$\tau_{\text{WCHB}} = \begin{cases} 3t_{eval} + t_{prech} + 2t_{CD}, & \text{if } t_{eval} \geq t_{prech} \\ 2t_{eval} + 2t_{prech} + 2t_{CD}, & \text{if } t_{eval} < t_{prech} \end{cases}$$

where t_{eval} is the evaluation time, t_{prech} is the precharge time through the PMOS pull-up network, and t_{CD} is the output completion detection time.

2.3.2 Pre-Charged Half Buffer

Unlike WCHB, a PCHB module requires completion detection at its inputs (besides the output) to tell whether an input is ready in its NULL or VALID state. The logic circuit implemented in the evaluation blocks can be arbitrary without being in the minterm expansion form. Hence the output may finish evaluation (i.e. be in a valid state) even before all inputs are in valid states. On the other hand, a PCHB module may start to precharge once its inputs and output are in valid states and its *ack_in* signal is low (indicating the inputs and output of its next stage are in valid states as well). In contrast, a WCHB module may start to precharge only after its inputs are all precharged. Thus PCHB may start its precharging earlier than WCHB.

The operation of a three-stage PCHB pipeline can be analyzed with the STG. There is one critical cycle that determines the cycle time τ_{PCHB} of the pipeline [3]. That is,

$$\tau_{\text{PCHB}} = 3t_{eval} + t_{prech} + 2t_{CD} + 2t_C$$

where t_{prech} is the precharge time, t_{CD} the completion detection time, and t_C is the C-element delay time. Notice that the precharge time of PCHB and that of WCHB may differ to some extent. For modules with many inputs, WCHB may take longer time to precharge than PCHB since the WCHB module has a long cascade of PMOS transistors.

It is worth to note that the handshaking mechanisms of PCHB and WCHB are compatible. The compatibility can be easily seen from the STG of hybrid PCHB-WCHB pipeline. It is this compatibility that we exploit in QDI circuit synthesis. Essentially we take advantage of the high performance of PCHB and the area saving of WCHB for optimization.

3. PERFORMANCE ANALYSIS OF ASYNCHRONOUS PIPELINES

We focus on *data-independent* performance analysis of *acyclic* PCHB-WCHB pipeline circuits. A linear-time static analysis procedure is proposed to compute the minimum cycle time for a given pipeline circuit. Moreover, it provides useful criticality information for cycle time reduction and thus throughput enhancement. Essentially the performance analysis algorithm will serve as a core engine for slack matching and WCHB area recovery in our synthesis flow.

3.1 Delay Model

Given an acyclic PCHB-WCHB pipeline circuit, we model it as a delay graph as follows.

DEFINITION 1. *A* delay graph *is a directed acyclic graph $G(V, E)$, where V and $E \subseteq V \times V$ are the vertex and edge sets, respectively. The set V of nodes is partitioned into input nodes V_{I}, output nodes V_{O}, and other internal nodes V_{INT}. Every node $v \in V$ (representing some input, output, or PCHB/WCHB module) is associated with three delay attributes: the evaluation delay (denoted $v.t_{eval}$), the precharge delay (denoted $v.t_{prech}$), and the combined delay of completion detector t_{CD} and, for the PCHB case, C-element t_C (denoted $v.t_c$). That is, $v.t_c = t_{CD} + t_C$ for PCHB and $v.t_c = t_{CD}$ for WCHB.*

For $v \in V_{\text{I}}$, $v.t_{eval} = v.t_{prech} = v.t_c = 0$; for $v \in V_{\text{O}}$, $v.t_{prech} = v.t_c = 0$ and $v.t_{eval} \geq 0$. That is, we assume the environment can produce tokens to the pipeline inputs whenever available and consume tokens from the pipeline outputs with a certain consumption delay.

Given a delay graph $G(V, E)$, for each node $v \in V$ there are four essential values to compute.

- $v.eval\_s$: evaluation start time (all foot NMOS transistors are turned on and all inputs are in VALID states),

- $v.eval\_f$: evaluation finish time (the output is in a VALID state),
- $v.prech\_s$: precharge start time (all precharge PMOS transistors are turned on), and
- $v.prech\_f$: precharge finish time (the output is in a NULL state).

3.2 PCHB Cycle Time

Given a delay graph $G(V, E)$ constructed from a PCHB pipeline circuit, for $v \in V_{\text{INT}}$ the aforementioned four values can be computed as follows.

$$\begin{aligned} v.eval\_s^{(i)} &= \max\{\max_{u \in FO(v)}\{u.prech\_f^{(i-1)} + u.t_c\}, \\ &\quad v.prech\_f^{(i-1)} + v.t_c, \max_{u \in FI(v)}\{u.eval\_f^{(i)}\}\} \end{aligned}$$

where $FI(v)$ and $FO(v)$ denote the fanins and fanouts, respectively, of node v, and the superscript (i) signifies the the i^{th} token being processed.

$$\begin{aligned} v.eval\_f^{(i)} &= v.eval\_s^{(i)} + v.t_{eval} \\ v.prech\_s^{(i)} &= \max\{\max_{u \in FI(v)}\{u.eval\_f^{(i)}\} + v.t_c, \\ &\quad \max_{u \in FO(v)}\{u.eval\_f^{(i)} + u.t_c\}, \\ &\quad v.eval\_f^{(i)} + v.t_c\} \\ v.prech\_f^{(i)} &= v.prech\_s^{(i)} + v.t_{prech} \end{aligned}$$

Initially, $v.prech\_f^{(0)} + v.t_c = 0$ for every $v \in V$.

THEOREM 1. *Given a delay graph $G(V, E)$, for $v \in V$ let*

$$\begin{aligned} \tau_v &= \max\{\max_{u \in FO(v)}\{u.prech\_f^{(1)} + u.t_c\}, \\ &\quad v.prech\_f^{(1)} + v.t_c\} - \max_{u \in FI(v)}\{u.eval\_f^{(1)}\}. \end{aligned}$$

Then any cycle time greater than $\tau = \max_v \tau_v$ is a valid cycle time of G.

THEOREM 2. *Given a delay graph $G(V, E)$, the computed cycle time τ of Theorem 1 is the minimum, assuming that all input tokens are simultaneously inserted into the graph and no tokens are blocked in any part of the graph.*

3.3 WCHB Cycle Time

Given a delay graph $G(V, E)$ constructed from a WCHB pipeline circuit, for $v \in V_{\text{INT}}$ we have the following four equations.

$$\begin{aligned} v.eval\_s^{(i)} &= \max\{\max_{u \in FO(v)}\{u.prech\_f^{(i-1)} + u.t_c\}, \\ &\quad \max_{u \in FI(v)}\{u.eval\_f^{(i)}\}\} \\ v.eval\_f^{(i)} &= v.eval\_s^{(i)} + v.t_{eval} \\ v.prech\_s^{(i)} &= \max\{\max_{u \in FI(v)}\{u.prech\_f^{(i)}\}, \\ &\quad \max_{u \in FO(v)}\{u.eval\_f^{(i)} + u.t_c\}\} \\ v.prech\_f^{(i)} &= v.prech\_s^{(i)} + v.t_{prech} \end{aligned}$$

Initially, $v.prech\_f^{(0)} + v.t_c = 0$ for every $v \in V$.

THEOREM 3. *Given a delay graph $G(V, E)$, for $v \in V$ let*

$$\begin{aligned} \tau_v &= \max\{\max_{u \in FO(v)}\{u.prech\_f^{(1)} + u.t_c\}, \\ &\quad v.prech\_f^{(1)} + v.t_c\} - \max_{u \in FI(v)}\{u.eval\_f^{(1)}\}. \end{aligned}$$

Then any cycle time greater than $\tau = \max_v \tau_v$ is a valid cycle time of G.

THEOREM 4. *Given a delay graph $G(V, E)$, the computed cycle time τ of Theorem 3 is the minimum, assuming that all input tokens are simultaneously inserted into the graph and no tokens are blocked in any part of the graph.*

```
PCHB-WCHB_CycleTimeComputation
  input: an acyclic PCHB-WCHB pipeline circuit C
  output: cycle time τ
  begin
  01   construct delay graph G(V, E) from C;
  02   foreach v ∈ V in a topological order
  03        compute v.eval_s^(1) and v.eval_f^(1);
  04   foreach v ∈ V in a topological order
  05        compute v.prech_s^(1) and v.prech_f^(1);
  06   τ := 0;
  07   foreach v ∈ V
  08        compute τ_v;
  09        if τ < τ_v
  10             τ := τ_v;
  11   return τ;
  end
```

Figure 1: Algorithm: Cycle Time Computation

3.4 Cycle Time Computation

Given a PCHB-WCHB pipeline circuit, the procedure of Figure 1 computes the tight lower bound of its cycle time. Since it traverses the delay graph three times for delay calculation, the overall complexity is linear $O(n)$ in the graph size n.

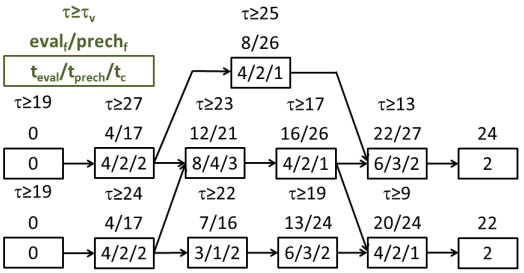

Figure 2: Pipeline under performance analysis

EXAMPLE 1. *Consider the delay graph of Figure 2, where each node v contains three attributes $(v.t_{eval}, v.t_{prech}, v.t_c)$. Assume PCHB is the underlying pipeline structure. Then $v.eval\_f$, $v.prech\_f$, and τ_v can be computed as shown above the node. Moreover, the overall minimum cycle time is 27.*

Notice that the procedure of Figure 1 computes for every node v the cycle time τ_v, which provides useful criticality information for pipeline optimization as to be discussed in Sections 4.5 and 4.6.

It is worth mentioning that, although the computed cycle time τ is a tight lower bound, not the entire circuit has to be limited by τ. In fact, a sub-circuit of a pipeline design may operate at a lower cycle time, provided that the sub-circuit is not in the transitive fanin and fanout cones of the most critical nodes, whose $\tau_v = \tau$. This observation suggests a refined performance analysis procedure to identify cycle times for sub-circuits with different levels of criticality. The procedure may proceed by first identifying the transitive fanin and fanout cones of nodes with $\tau_v = \tau$. So the performance of the identified sub-circuit is limited by τ. It then identifies the largest cycle time, say τ', among the nodes in the remaining sub-circuit. The transitive fanin and fanout cones of the remaining nodes with $\tau_v = \tau'$ are then identified, and the performance of the identified sub-circuit is limited by τ'. This process may iterate until no sub-circuit left.

4. QDI CIRCUIT SYNTHESIS

Our synthesis procedure aims at transforming a Boolean expression into a QDI implementation in a hybrid PCHB and WCHB design style optimized with respect to area and performance constraints.

4.1 Synthesis Flow Overview

The proposed synthesis flow is shown in Figure 3. The procedure takes as input a (single-rail encoded) gate-level logic

QDI_CircuitSynthesis
input: a logic netlist $\mathcal{C}$, target cycle time τ, and
a PCHB-WCHB library L
output: a hybrid PCHB-WCHB implementation of $\mathcal{C}$
begin
01 perform technology independent logic synthesis on $\mathcal{C}$;
02 perform cut-based technology mapping on $\mathcal{C}$ w.r.t. L;
03 **if** τ is not achieved in $\mathcal{C}$
04 perform slack matching on $\mathcal{C}$ w.r.t. τ;
05 perform area recovery on $\mathcal{C}$;
06 **return** $\mathcal{C}$;
end

Figure 3: Algorithm: QDI Circuit Synthesis

netlist (e.g., obtained through the standard synchronous design flow). We assume the logic netlist is combinational and acyclic. It transforms the netlist to an optimized hybrid PCHB-WCHB pipeline circuit. The steps are elaborated in detail as follows.

4.2 Technology Independent Logic Optimization

Before being mapped to a PCHB-WCHB circuit, a given logic netlist is optimized by standard technology independent logic synthesis to simplify its Boolean expression. In particular, we represent the circuit as an AND-INVERTER graph (AIG), where every node represents a 2-input AND gate and the two inputs of a gate can be optionally complemented by inverters (represented with a single bit flip). Due to its compact data structure, the AIG is a scalable representation for large industrial designs. We optimize an AIG with conventional rewriting techniques [9] for circuit size and logic level minimization. The optimized AIG is to be further processed by technology mapping for PCHB-WCHB realization.

4.3 Library Construction

For the purpose of technology mapping, a library of PCHB and WCHB modules is constructed. Since long stacking of transistors in the pull-up and pull-down networks of a module is not desirable, we restrict the number of inputs to a module no greater than 4. PCHB and WCHB modules for all Boolean functions up to 4 inputs are built. Notice that, for a dual-rail encoded logic netlist, complementing a variable/function x corresponds to swapping its two encoding bits x_0 and x_1, and thus is at no hardware cost. Accordingly, for the 65536 (i.e. 2^{2^4}) Boolean functions up to 4 inputs, we only need to construct their 402 representatives under the equivalence of input negation and input permutation (the so-called NP-equivalence).

To satisfy the weak condition of WCHB, the function evaluation block of a WCHB module is realized in a minterm expansion form with possible logic sharing. In contrast, no such requirement is needed for PCHB.

4.4 Technology Mapping

Given an AIG optimized by technology independent logic synthesis, it is to be realized with respect to our constructed PCHB-WCHB library. Since the PCHB and WCHB modules can realize any Boolean function with up to 4 inputs, we may perform technology mapping by enumerating 4-feasible cuts, similar to the technology mapping of lookup-table based field programmable gate array (FPGA) [10].

When a cut is generated, we may derive its corresponding Boolean function as well as a set of *satisfiability don't cares* (SDCs) [5] from the underlying AIG. In essence, the SDCs are the set of truth assignments to the inputs of the cut that cannot appear due to the fact that the functions to these inputs may not be surjective. Because of the underlying dual-rail encoding, any such don't care can be optionally assigned to *the onset function, offset function, both of the onset and offset functions, or none of the onset and offset functions.* Notice the fundamental difference that, for a single-rail logic circuit, an SDC can only be assigned to either the onset or the offset. We exploit SDCs for circuit optimization as follows.

Given a cut function f and its SDCs, we look for different combinations of the SDC assignments and choose the best realizations of the onset function f_1 and offset function f_0 separately from the library. Essentially we retrieve a PCHB/WCHB module from the library by Boolean function modulo the equivalence under input negation and input permutation (the so-called NP-equivalence). Notice that NP-equivalence has to be applied for library look-up since both onset and offset functions have to be realized.

4.5 Slack Matching

We explore an iterative heuristic approach to improve PCHB pipeline throughput via buffer insertion. For practical implementation, a simple strategy is applied. We first perform buffer insertion to balance pipeline stages. The inserted buffers are sorted according to their criticality in the pipeline, and are iteratively removed provided that their removal incurs no cycle time violation with respect to a given target cycle time τ.

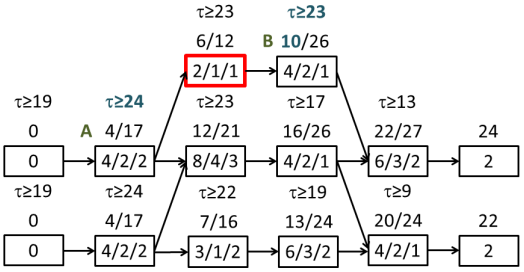

Figure 4: Pipeline under slack matching

EXAMPLE 2. *Given the delay graph of Figure 2, its cycle time can be improved by slack matching as shown in Figure 4, where the highlighted buffer node is inserted. Since the value of the most critical τ_A is determined by $B.prech\_f$, inserting a buffer between nodes A and B reduces the precharge finish time of A's new fanout node. It effectively improves the cycle time from 27 to 24.*

4.6 Area Recovery

We exploit the compatibility between PCHB and WCHB, and the area advantage of WCHB for PCHB pipeline area recovery. Specifically, replacing 1-input, 2-input, 3-input, and 4-input PCHB modules with their corresponding WCHB modules may potentially save up to 12, 16, 15, and 19 transistors, respectively. For implementation, we sort the nodes in the delay graph according to their timing criticality and potential area reduction due to WCHB replacement. WCHB replacement is performed iteratively according to the order, and only those replacements that incur no cycle time violation with respect to a given target cycle time τ are accepted. The proposed performance analysis procedure is applied to assess the performance impact due to the replacement; the timing and criticality information of the delay graph is incrementally updated after a replacement. The process terminates when no improvements can be achieved.

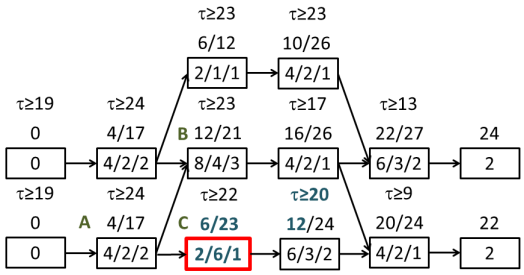

Figure 5: Pipeline under WCHB replacement

EXAMPLE 3. *Given the delay graph of Figure 4, suppose that all the nodes correspond to PCHB modules. Observe that a WCHB module has larger precharge time than its PCHB counterpart. Among the fanouts of a node v, suppose the precharge finish time of $u \in FO(v)$ is much smaller than other $w \in FO(v)$. Then performing WCHB replacement on u may incur less cycle time penalty on v. In the example of Figure 5, for node A, its fanout C is thus more preferred for replacement than B. By replacing a non-critical node with a functionally equivalent and smaller WCHB module, the area may be reduced while the cycle time may remain intact.*

5. EXPERIMENTAL RESULTS

The proposed performance analysis and logic synthesis procedures were implemented in the C++ language under the ABC [2] synthesis and verification environment. All experiments were conducted on a Linux machine with two 6-core Xeon 2.3GHz CPU and 32GB RAM.

A library of PCHB and WCHB modules were constructed under NP-equivalence for any Boolean function with up to 4 inputs. We used the Predictive Technology Model (PTM) 180 nm technology file [14] to generate delay information for our library cells. While we ignored wire delays, it should not be difficult to incorporate them into the delay graph.

ISCAS and ITC benchmark circuits were selected for experiments. The circuits were first synthesized using ABC script `strash`, `balance`, `dch`, `dch`, `dch` for technology independent logic transformation. On the other hand, for technology mapping, ABC command `if` was modified for cut generation and technology mapping under our settings, including new area and delay costs, SDC conditions, etc. By technology mapping, circuits purely consisting of PCHB modules were generated. They were subject to the subsequent performance constrained area minimization. We conducted the performance constrained area minimization under two settings: First, let the target cycle time τ be the cycle time of the original pure PCHB circuit. WCHB replacement was performed for area recovery with respect to the given τ. Second, let τ be 80% of the cycle time of the original pure PCHB circuit. Slack matching by buffer insertion was performed to meet τ, and then WCHB replacement was performed for area recovery.

For the first experiment, the results are shown in Table 1, where Columns 2, 3 and 4 show the transistor count of function evaluation blocks, the transistor count of the rest parts, and total transistor count, respectively, in the original pure PCHB circuit; Column 5 shows the cycle time of the original pure PCHB circuit; Column 6 shows the total CPU time for synthesis; Columns 7 and 8 show the number of substituted modules with WCHB and the total module number, respectively, in the hybrid PCHB-WCHB circuit; Columns 9, 10, and 11 show the transistor count of function evaluation blocks, the transistor count of the rest parts, and total transistor count, respectively, in the PCHB-WCHB hybrid circuit; Column 12 shows the cycle time after WCHB substitution; Column 13 shows the total CPU time for PCHB-WCHB hybrid circuit synthesis; Columns 14 and 15 show the area ratio, cycle time ratio of PCHB-WCHB to PCHB. As can be seen, after WCHB substitution, the area can be reduced by an average of 11.6%, without cycle time increase. For circuit `c6288`, which is a 16×16 multiplier, since it has long datapath, the WCHB substitution cannot significantly reduce the circuit area without increasing the cycle time.

Table 2, with a similar arrangement as Table 1, compares circuits implemented by pure PCHB modules and by pure WCHB modules. Essentially, this table gives the lower bounds for circuit area recovery. Although the area can be (maximally) reduced by an average of 18.4%, the cycle time can be on average 2.3 times longer than the original PCHB circuits. For example, circuit `c6288`, although the area can be reduced to 81.2% of the original size, the cycle time can be 2.9 times longer than the original PCHB circuit. Comparing Tables 1 and 2, we see the effectiveness of WCHB substitution in reducing area (close to pure WCHB implementation) while maintaining the original cycle time.

For the second experiment, the results are shown in Table 3, where Column 2 shows the total area of the original circuit in transistor count; Column 3 shows the cycle time of the original circuit; Columns 4, 5 and 6 show the number of inserted buffers, area ratio, and cycle time ratio, respectively, after buffer insertion to balance pipeline stages; Columns 7, 8 and 9 show the number of remaining buffers, area ratio, and cycle time ratio, respectively, after removing noneffective buffers (to keep new cycle time within 80% of the original one); Column 10 shows the number of modules substituted with WCHB; Column 11 shows the total number of modules after slack matching; Columns 12 and 13 show the area ratio and cycle ratio after slack matching and WCHB substitution; Column 14 shows the total CPU time for synthesis. In slack matching, we inserted buffers between nodes by their level difference, so the area overheads are large with an average of about 454%. However, the cycle time can be improved by 43% on average. Based on our 80% cycle time constraint, we removed a considerable amount of buffers, and reduced the area overhead to 25% on average. For circuit `s35932`, the cycle time after buffer insertion is greater than the target cycle time constraint, and no buffer can be removed. Finally, we performed WCHB substitution to further reduce the area overhead to about 6.4%. Moreover, some circuits, e.g., `b22`, are even smaller in area than the original ones.

6. CONCLUSIONS AND FUTURE WORK

This paper has presented a linear-time algorithm for (data independent) cycle time analysis of acyclic PCHB-WCHB hybrid pipeline circuits. Built upon this analysis tool, a synthesis flow for PCHB-WCHB circuit optimization (specifically, performance constrained area minimization) has been proposed. Due to its computational efficiency, the analysis tool can serve as a core engine providing essential criticality information to guide incremental optimization, which is crucial for scalable synthesis. Experimental results have demonstrated the feasibility and effectiveness in circuit synthesis. For future work, performance analysis for cyclic pipelines awaits further development. Also our slack matching and area recovery methods can be further refined.

Acknowledgments

This work was supported in part by the National Science Council under grants NSC 101-2923-E-002-015-MY2, 102-2221-E-002-232, and 102-2622-E-002-014.

7. REFERENCES

[1] P. Beerel, A. Lines, M. Davies, and N.-H. Kim. Slack matching asynchronous designs. In *Proc. Int'l Symp. on Asynchronous Circuits and Systems*, pp. 184-194, 2006.

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* `http://www.eecs.berkeley.edu/~alanmi/abc/`

[3] P. Beerel, R. Ozdag, and M. Ferretti. *A Designer's Guide to Asynchronous VLSI.* Cambridge University Press, 2010.

[4] K. Fant and S. Brandt. Null Convention Logic: A complete and consistent logic for asynchronous digital circuit synthesis. In *Proc. Int'l Conf. on Application-Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.

[5] J.-H. R. Jiang and S. Devadas. Logic synthesis in a nutshell. In *Electronic Deisng Automation: Synthesis, Verification, and Test.* L.-T. Wang, K.-T. Cheng, and Y.-W. Chang (Editors), Morgan Kaufmann Publishers, pp. 299-404, 2009.

[6] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous CAD tools. *IEEE Design & Test of Computers*, 19(4): 107-117, 2002.

[7] A. Lines. Pipelined asynchronous circuits. M.S. thesis, California Institute of Technology, 1995.

[8] J. Magott. Performance evaluation of concurrent systems using Petri nets. *Information Processing Letters*, 18: 7-13, 1984.

Table 1: PCHB vs. Hybrid PCHB-WCHB

circuit	PCHB					PCHB-WCHB							area ratio	τ ratio
	transistor count			τ	time	#modules		transistor count			τ	time		
	func.	rest	total	(ns)	(s)	sub.	total	func.	rest	total	(ns)	(s)		
c5315	4739	22745	27484	1.622	0.29	240	437	6463	18555	25018	1.622	0.38	0.910	1.000
c6288	8395	29934	38329	3.628	1.05	131	524	8945	27614	36559	3.628	1.19	0.954	1.000
c7552	5214	23027	28241	1.740	0.32	305	449	6758	17908	24666	1.740	0.44	0.873	1.000
s5378	3913	19904	23817	1.386	0.22	202	383	4885	16526	21411	1.383	0.34	0.899	0.998
s9234.1	5009	26582	31591	1.614	0.31	296	511	6749	21621	28370	1.611	0.53	0.898	0.998
s13207	7827	41836	49663	1.657	0.48	566	801	11073	32336	43409	1.651	0.85	0.874	0.996
s15850	10032	54950	64982	2.094	0.8	777	1079	14643	42017	56660	2.094	1.4	0.872	1.000
s35932	29842	159950	189792	1.032	1.78	2710	3394	37324	117449	154773	1.032	5.57	0.815	1.000
s38417	29276	156924	186200	1.740	2.24	1839	3028	42012	126809	168821	1.740	8.27	0.907	1.000
s38584	36595	195284	231879	1.622	2.38	2564	3821	52996	152505	205501	1.622	10.83	0.886	1.000
b12	3860	22210	26070	1.496	0.28	255	442	5376	18089	23465	1.496	0.37	0.900	1.000
b14	16622	83802	100424	3.392	2.01	1165	1642	23665	64088	87753	3.392	3.91	0.874	1.000
b15	30670	158548	189218	3.266	6.42	2122	3136	44352	122127	166479	3.264	14.32	0.880	0.999
b17	91811	477161	568972	4.430	18.24	7736	9507	142857	343761	486618	4.421	70.45	0.855	0.998
b20	35527	177115	212642	3.392	5.86	2339	3497	48952	137503	186455	3.392	15.82	0.877	1.000
b21	35686	177885	213571	3.483	5.57	2415	3505	49478	137026	186504	3.483	15.26	0.873	1.000
b22	53162	262913	316075	3.494	7.95	3522	5199	73347	203561	276908	3.494	31.03	0.876	1.000
average													0.884	0.999

Table 2: PCHB vs. WCHB

circuit	PCHB				WCHB				area ratio	τ ratio
	transistor count			τ	transistor count			τ		
	func.	rest	total	(ns)	func.	rest	total	(ns)		
c5315	4739	22745	27484	1.622	7845	15105	22950	3.39	0.835	2.089
c6288	8395	29934	38329	3.628	10914	20204	31118	10.55	0.812	2.908
c7552	5214	23027	28241	1.740	7680	15388	23068	3.82	0.817	2.195
s5378	3913	19904	23817	1.386	6002	13345	19347	2.69	0.812	1.943
s9234.1	5009	26582	31591	1.614	8159	17804	25963	3.40	0.822	2.107
s13207	7827	41836	49663	1.657	12396	28112	40508	3.63	0.816	2.188
s15850	10032	54950	64982	2.094	16214	36756	52970	4.73	0.815	2.256
s35932	29842	159950	189792	1.032	42016	104828	146844	1.53	0.774	1.484
s38417	29276	156924	186200	1.740	49904	105974	155878	4.02	0.837	2.312
s38584	36595	195284	231879	1.622	61890	129744	191634	3.54	0.826	2.183
b12	3860	22210	26070	1.496	6370	14861	21231	3.13	0.814	2.093
b14	16622	83802	100424	3.392	26630	55640	82270	7.71	0.819	2.274
b15	30670	158548	189218	3.266	49848	104094	153942	8.46	0.814	2.591
b17	91811	477161	568972	4.430	149514	312961	462475	11.67	0.813	2.635
b20	35527	177115	212642	3.392	56403	117330	173733	8.84	0.817	2.606
b21	35686	177885	213571	3.483	56552	117911	174463	9.05	0.817	2.599
b22	53162	262913	316075	3.494	83630	174273	257903	9.15	0.816	2.620
average									0.816	2.299

Table 3: Results of Slack Matching with WCHB Substitution

circuit	original circuit		inserted buffers			non-deleted buffers			WCHB sub.				time
	area	τ (ns)	#buf	area ratio	τ ratio	#buf	area ratio	τ ratio	#modules sub.	#modules total	area ratio	τ ratio	(s)
c5315	27484	1.622	1613	2.819	0.667	427	1.482	0.800	608	864	1.228	0.800	1.56
c6288	38329	3.628	7006	6.666	0.415	72	1.058	0.795	184	596	0.996	0.790	8.87
c7552	28241	1.740	1103	2.211	0.622	103	1.113	0.799	356	552	0.966	0.799	0.9
s5378	23817	1.386	1184	2.541	0.737	429	1.558	0.799	566	812	1.271	0.797	1.27
s9234.1	31591	1.614	2481	3.435	0.665	428	1.420	0.799	617	939	1.186	0.799	2.8
s13207	49663	1.657	2409	2.504	0.621	117	1.073	0.800	576	918	0.943	0.800	2.87
s15850	64982	2.094	6215	3.965	0.560	576	1.275	0.800	1113	1655	1.072	0.800	13.92
s35932	189792	1.032	1370	1.224	0.903	1370	1.224	0.903	4126	4764	0.949	0.894	8.11
s38417	186200	1.740	13633	3.270	0.625	2024	1.337	0.799	3308	5052	1.139	0.799	147.02
s38584	231879	1.622	17405	3.327	0.667	2123	1.284	0.800	3222	5944	1.119	0.800	235.18
b12	26070	1.496	1503	2.787	0.698	332	1.395	0.799	509	774	1.174	0.799	1.46
b14	100424	3.392	18179	6.612	0.443	463	1.143	0.800	1415	2105	0.983	0.800	161.69
b15	189218	3.266	40173	7.582	0.457	3006	1.492	0.800	4671	6142	1.206	0.800	1027.62
b17	568972	4.430	137729	8.504	0.401	2212	1.121	0.800	8820	11719	0.950	0.800	8769.49
b20	212642	3.392	37952	6.533	0.426	789	1.115	0.800	2611	4286	0.972	0.800	953.47
b21	213571	3.483	38661	6.612	0.441	805	1.117	0.800	2750	4310	0.967	0.800	462.54
b22	316075	3.494	58139	6.702	0.428	1129	1.111	0.800	4071	6328	0.961	0.800	1613.96
average				4.547	0.575		1.254	0.805			1.064	0.804	

[9] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proc. Design Automation Conference*, pp. 532-535, 2006.

[10] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 354-361, 2007.

[11] A. Martin and M. Nyström. Asynchronous techniques for system-on-chip design. *Proc. of the IEEE*, 94(6): 1089-1120, 2006.

[12] P. McGee and S. Nowick. An efficient algorithm for time separation of events in concurrent systems. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 180-187, 2007.

[13] D. E. Muller. Asynrhconous logics and application to information processing. In *Proc. Symp. Application of Switching Theory in Space Technology*, pp. 289-297, 1963.

[14] Nanoscale Integration and Modeling Group. *Predictive Technology Model*. `http://ptm.asu.edu/`

[15] C. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Trans. Software Eng.*, SE-6(5): 440-449, 1980.

[16] R. Reese, S. Smith, and M. Thornton. UNCLE — An RTL approach to asynchronous design. In *Proc. Int'l Symp. on Asynchronous Circuits and Systems*, pp. 65-72, 2012.

[17] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design*. Kluwer Academic Publishers, 2001.

[18] A. Smirnov and A. Taubin. Heuristic based throughput analysis and optimization of asynchronous pipelines. In *Proc. Int'l Symp. on Asynchronous Circuits and Systems*, pp. 162-172, 2009.