Regular Paper

Verification and Violation Correction of Timing Constraints for Gate-Level Asynchronous Circuits

Metehan Özcan,† Masashi Imai,† Hiroshi Nakamura†
and Takashi Nanya†

Traditional asynchronous design methodologies basically create correct-by-design circuits with almost no assumptions on delay values in the circuit. However, this over-pessimism usually creates slow circuits. Recently, asynchronous design methodologies which utilize delay information and apply timing optimizations have been suggested. These timing optimizations bring new timing constraints to be observed especially after the layout phase. The aim of this work is to develop a timing verification methodology and an appropriate CAD framework for gate-level asynchronous circuits using well-known static timing analysis method, which will be a bridge between asynchronous logic synthesis tools and common layout tools. Verification of timing constraints and correction of violations (if exist any) after layout are two main objectives. First, basic concepts for verification methodology will be given. Then, an algorithm for verification and violation correction of timing constraints for general asynchronous circuits is proposed. Later, asynchronous data-path circuits are examined in more detail. Finally, current status of the developed CAD framework is explained along some experimental results.

1. Introduction

With advances in device technology interconnections are becoming responsible for most of the signal delay and distribution of clock signal becomes more difficult to attain. Due to its average-case behavior, timing reliability and nonexistence of clock skew, asynchronous systems have been gaining interest for future high-speed systems.

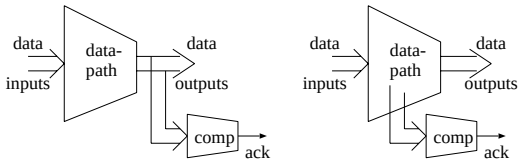
In the absence of a clock for synchronization, asynchronous modules communicate using request-acknowledgment protocol [1]. One side issues request signal and necessary data for an operation, and the other side processes data and acknowledges the end of operation using acknowledgment signal. Note that, acknowledgment signal should be generated after processing of data is finished.

For data-path circuits, since there is no clock, successive arrival of data should be distinguished with some extra timing signal(s). One way to achieve this to use one extra strobe signal for a bunch of bits. This kind of data-path circuits are called single-rail data-path circuits. Another possible way is to use M-out-of-N encoding, in which exactly M lines become 1 and (N-M) lines become 0 for a valid data. One example for this is dual-rail (1-out-of-2) encoding. In this encoding two wires are used for each data bit, one of them indicating arrival of data 0 and the other one arrival of data 1. (0,0) (no change in both lines) is called spacer and used between successive data arrivals. (1,1) is regarded as invalid data.

In asynchronous systems, delay model also has a great impact on the design and analysis of the circuit [2]. Delay-Insensitive (DI) model assumes gate delays and wire delays are unbounded. In Quasi-Delay-Insensitive (QDI) model, "isochronic forks" are added to DI model. Fork points for wires are assumed to have same delay, i.e., isochronic. Another variety is Speed Independent (SI) model, in which unbounded gate delays with zero wire delays are assumed. But these delay models may sometimes impose over-pessimistic delay assumptions on the circuit, resulting in slow circuits. Especially in local parts of the circuit, delay values can be estimated before layout and these estimated delay values could be utilized to enhance circuit performance.

Recently delay models and design methodologies, which utilize delay values and apply some timing optimizations, have been suggested for asynchronous circuits. Scalable-Delay-Insensitive (SDI) model is based on the idea that once a system is laid out and fabricated, elements in the design are affected almost similarly by the changes in the operating environment [3]. For dual-rail asynchronous

† Research Center for Advanced Science and Technology, The University of Tokyo



a) QDI model b) SDI model

(comp : completion generation circuit)

Fig. 1 Data-path circuits based on QDI and SDI model.

data-path circuits based on QDI model, acknowledgment signal is generated by checking that all the data bits have made the necessary transition (**Fig. 1**a). However, in this way delay of the completion generation circuit becomes large and degrades the circuit performance. In SDI model circuits, as far as completion signal is generated after all the data-path circuit completes its transition, circuit operates correctly. Therefore, generation of the completion signal could start on the way before data-path circuit completes its transition (Fig. 1b). Consequently, new timing constraints emerge which are to be observed also during layout. These kind of optimizations are utilized during the design of TITAC-2, an asynchronous microprocessor [4], however layout part was performed manually.

Another recently suggested methodology is Relative Timing (RT) circuits [5),6)]. Instead of using absolute delay values in the circuit, this method utilizes relation between two signals, i.e., one signal is earlier than the other one or two signals are equivalent in time, and applies timing optimizations based on the relative timing of two signals. For this purpose, they first synthesize circuit based on SI model and obtain initial timing values for signals. Later they generate RT assumptions between signals and utilize these relations for timing optimizations. Again, like SDI circuits, these constraints are to be observed during the layout phase.

Although optimizations are applied in synthesis level and some constraints are emerging as a result, there is not much work on how to perform layout of optimized circuits using existing layout tools. The aim of this work is to develop a verification methodology and a CAD framework to fill the gap between currently available (synchronous) layout tools and asynchronous synthesis methodologies which utilize timing optimizations. Furthermore, after the verification of the circuit, any violation of the timing constraints should be corrected without the expensive back-tracing to synthesis phase.

Organization of the paper is as follows. In Section 2 format of timing constraints, which will be the interface between synthesis and layout phases, and whether they can be indeed generated by synthesis tools are explained. Later basic principles of the verification methodology is given in Section 3. Section 4 gives challenges for constraint verification and violation correction for general asynchronous circuits and a new algorithm for this purpose. In Section 5, timing constraints for asynchronous data-path circuits and their verification are discussed in detail. Section 6 gives the current situation of developed CAD framework along some experimental results. Finally conclusion and further work are given in Section 7.

2. Constraint Format Determination

Absolute delay values are variant within system's lifetime, so they are relatively difficult to estimate in the early stages of the design. For example, both SDI and RT circuits utilize the relation between the delays of signals within the circuit. From this observation, we propose to express the timing relations and constraints within the circuit as **fast path–slow path** pairs. Like the relation between data-path signal and completion signal, there are order relations between signals in asynchronous circuits. A signal transition path starts from a circuit node or nodes and ends with a circuit node or nodes, possibly including some intermediate nodes between them. As definition, maximum fast path delay should be always smaller than minimum slow path delay. For performance figures, both paths are desired to be as fast as possible and slow paths are required to be slower than fast paths for some margin to guarantee correct operation. As long as this fast-slow relation is obeyed, slow paths could be made close to fast paths.

To designate a path in the circuit, we need start point(s) and end point(s). To designate a path more specifically, optionally some intermediate points and inhibit points (through which a path is not allowed to pass) can be used, which are useful for cutting cycles for non-combinational circuits or false path elimination.

Timing constraints are identified at logic design level and this information is conveyed for

the constraint verification. To calculate the delay value of a path, maximum delay analysis is performed for a fast path and minimum delay analysis is performed for a slow path using Static Timing Analysis (STA) method.

An important issue is whether timing constraints could be expressed as fast path–slow path pairs and could be indeed generated by logic synthesis tools.

First observation is that two signals, e.g., a and b, in a circuit can be related in one of the four ways; a occurs before b, b occurs before a or a and b occur simultaneously with respect to a minimum delay value δ or a different signal c. For the simultaneity, two signals may be used instead of each other for logic realization. In the last case, signals a and b regarded as the same signal from the timing point of view to realize signal c, unless there are special dependencies between signals other than timing issues. In each case another reference point in the circuit, signal s, should be designated in order to be able to have some delay value for a, b and c. Then, relation between two signals can be expressed as follows;

- a occurs before $b \rightarrow delay_a < delay_b$
- b occurs before $a \rightarrow delay_b < delay_a$
- a and b occur simultaneously w.r.t. $\delta \rightarrow delay_a < delay_b + \delta \quad \& \quad delay_b < delay_a + \delta$
- a and b occur simultaneously w.r.t. $c \rightarrow delay_a < delay_c \quad \& \quad delay_b < delay_c$

Note that now all four cases can be expressed as an inequality for delay values. Points in the circuit corresponding to earlier signal become end points for fast path, circuit points corresponding to later signal become end points for slow path and circuit points corresponding to common reference point become start points for the paths.

When a timing optimization is applied for an asynchronous circuit, relation between two signals is utilized. Since logic synthesis methods and corresponding tools use this relation, they can indeed produce this relation as a timing constraint in the format of fast path–slow path pairs with a very small effort. During static timing analysis cycles in the circuit are cut to prevent infinite loops during traversal. However, intermediate points are especially used for allowing cycles inherent to some asynchronous circuits. Inhibit points are optional and usually used for not analyzing unrelated parts of the circuit, possibly resulting in improved runtime values. In our methodology, it is assumed that constraints are produced at logic synthesis level in fast path-slow path format.

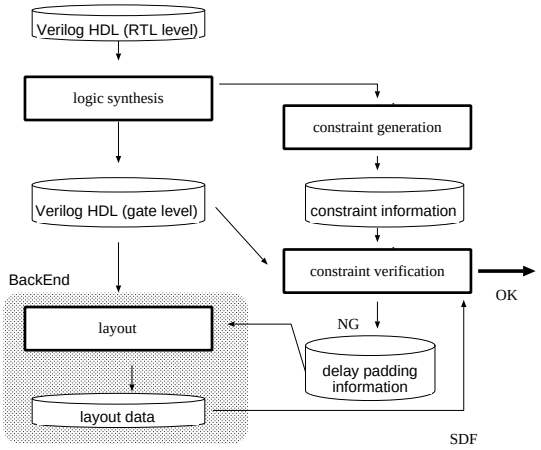


Fig. 2 Verification methodology.

3. Verification Methodology

When a timing optimization is applied using the relation of signals in the circuit, this relation is to be progressed in the design flow. As explained in the last section, this information is incorporated as timing constraints in the form of fast path-slow path pairs. This constraint information becomes the interface between logic synthesis and layout phases. Later timing constraints are verified using STA method using delay values before or after layout.

If any violation occurs for timing constraints, they are corrected by adding delay pads into the end points of the slow paths. However, this procedure should be carried out carefully since adding a delay pad into slow path may result in generating new constraint violations if paths are interleaved.

Figure 2 shows the design methodology of asynchronous circuits with our verification and correction concept. Verilog RTL level descriptions for asynchronous circuits are taken as input. Later logic synthesis is applied to obtain gate-level asynchronous circuits. During the logic synthesis, constraint are generated in the proposed format of fast path-slow path pairs. Depending on the synthesis methodology and the type of the basic building blocks for the circuit, different constraint information may to be generated. Thus, some extra information is to be added into the general design flow or a constraint generation function is to be added for each different logic synthesis method. Currently, the latter is employed in

the design methodology used in our research group. As different logic synthesis methodologies are added into the design methodology, this issue will be further examined.

Later, these constraints are verified with the tool environment developed, which can be performed either before layout or after layout using layout data information in Standard Delay File (SDF) format. For verification static timing analysis with Depth First Search (DFS) traversal is used; maximum delay analysis from end points back to start points for fast paths is performed (and similarly minimum delay analysis for slow paths). During traversal, if any inhibit point or primary input of the circuit is reached, that path is discarded. If an end point is reached after visiting all intermediate points, that path is considered as a valid path. After verification, if any constraint violation exist, delay padding information is generated and layout is re-performed by adding delay pads into proper places to correct the violations. Finally, after re-layout constraints are once more verified, which are expected to be satisfied if there is no cyclic relation between constraints (refer to Section 4).

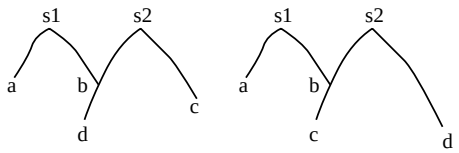
4. Constraint Verification and Violation Correction for Asynchronous Circuits

For general asynchronous circuits, constraints are to be generated by synthesis tools because they cannot be detected without the knowledge of synthesis phase. Furthermore, as the constraints may be interleaved, delay padding is a challenging problem since adding a delay pad may eventually generate new constraint violations.

The crucial observation is that, constraints should be somehow sorted and addition of delay pads should be done in ascending order to obtain optimum delay padding. To conceptualize this fact, let's consider the relation between a constraint $a < b$ and another constraint $c < d$, with start points as $s1$ and $s2$ respectively. Also note that if there is a violation in $a < b$, it is to be corrected by adding a delay pad into the end point of b.

Case one; if end point of b is not included in c or d, adding delay pad into b will not effect c or d. Therefore, this case is straightforward.

Case two; end point of b is included within d (**Fig. 3**a). Assume that delay values for a, b, c, and d are 15, 10, 30, and 20 respectively. If



a) Endpoint of b included in d b) Endpoint of b included in c

Fig. 3 Relation of two interleaved constraints.

$c < d$ is checked first, a delay pad of 11 will be added into d. Later when $a < b$ is checked, a delay pad of 6 will be added into b. However, this will increase the delay value of d into 37, making final delay values as 15, 16, 30, and 37 respectively. If the checking is done in the reverse order, final delay values will become 15, 16, 30, and 31.

Case three; end point of b is included within c (Fig. 3b). Again assume that delay values for a, b, c, and d are are 15, 10, 30, and 20 respectively. If $c < d$ is checked first, a delay pad of 11 will be added into d. Later when $a < b$ is checked, a delay pad of 6 will be added into b. However, this will increase the delay value of c into 36, making delay values as 15, 16, 36, and 31 respectively. To have correct delay values, $c < d$ should be checked again and a delay pad of 6 should be added into d, making final delay values as 15, 16, 36, and 37 respectively. If the checking is done in the reverse order, final delay values will again become 15, 16, 36, and 37, but only two checks will be enough in contrast to three checks in former case.

It can be seen from the above three cases that, naive delay padding without sorting will need more constraint checks when a delay padding location is included within another fast path, and will not produce optimum delay padding when a delay padding location is included within another slow path. Based on this observation, we propose the following algorithm to verify constraints and to correct constraint violations, which is to be run if any constraint violations exist in the circuit.

Algorithm for violation correction of interleaved timing constraints

Definition 1. A constraint $cons_1$ is said *smaller* than another constraint $cons_2$, if any end point of $cons_1$ is included within the fast path or the slow path of $cons_2$, and shown as $cons_1 < cons_2$. Similarly, $cons_2$ is said *larger* than constraint $cons_1$.

Definition 2. A constraint $cons_1$ is said *recursively smaller* than another constraint

$cons_2$, if $cons_1 < cons_2$ or for another constraint $const3$, $cons_1 < cons_3$ and $cons_3 < cons_2$.

Other definitions:

eq_i**:** constraint i expressed as $eq_i = fp_i < sp_i$
p_i**:** places to add delay pads (endpoints of sp_i)

v_i**:** list of constraints in conflict with const. i
Y**:** list of all constraints
S_i**:** set of constraints *smaller* than eq_i
SS_i**:** set of constraints *recursively smaller* than eq_i
L_i**:** set of constraints *larger* than eq_i
X**:** sorted list of constraint subsets

Algorithm:

```
1.  for each eq_i in Y do  → create S_i and L_i
2.    if p_i is included in some fp_j or sp_j
3.      add eq_j into L_i and eq_i into S_j
4.        if eq_i is already in L_j (conflict!)
5.          put eq_i into v_j and eq_j into v_i
6.  for each eq_i in Y do  → individual
7.    if S_i and L_i are empty
8.      if there is violation for eq_i
9.        determine delay pad for eq_i
10.       subtract eq_i from Y
11. for each eq_i in Y do        → create SS_i
12.   for each eq_j in S_i do
13.     if eq_j is not in v_i
14.       insert eq_j into SS_i
15.       for each eq_k in S_j do recursively
16.         if eq_k is in L_i
17.           put eq_i into v_k and eq_k into v_i
18.         else
19.           insert eq_k into SS_i
20. j = 1, X_0 = {}        → sort constraints
21. while (Y! = {}) do
22.   for each eq_i in Y do
23.     if SS_i composed of elmt.s from X_k
                                   (where k < j)
24.       put eq_i into X_j and subtract from Y
25.   j = j + 1
26. for i = 1 to length of X do  → fix viol.
27.   for each eq_i in X_i do
28.     if there is violation for eq_i
29.       determine delay pad for eq_i
30. if violation still remains
31.   ERROR!! report conflicting constraints
```

This algorithm first creates *smaller* and *larger* sets for constraints in lines 1 through 5. If any conflicting constraint exists, e.g. $a < b < a$, then these constraints are marked specially. Then constraints which are not in relation with other constraints are first verified and corrected in lines 6 through 10. Later using *smaller* and *larger* sets constructed in first part, *recursively smaller* sets are constructed in lines 11 through 19. Again if any constraints are in conflict, e.g. $a < b < c < a$, two of them from the cyclic list added into conflict list of each other to prevent infinite recursion for corrections. After *recursively smaller* sets are constructed, final sorted list of constraints, i.e., X, is constructed in lines 20 through 25, of which elements are verified and corrected in ascending order (lines 26 through 29). If any violation remains, list of conflicting constraints for the constraints with violation is output to the user.

Let's see how the algorithm works for a sample set of constraints;

$eq_1 = d < g$, $eq_2 = k < l$, $eq_3 = b < c$, $eq_4 = c < d$, $eq_5 = a < b$, $eq_6 = l < m$, $eq_7 = b < f$, $eq_8 = f < d$

For easier representation inequalities are given letters to visualize relation between constraints. In addition to above constraints, also assume that m is included within the slow path of d. At the first stage, S_i's and L_i's are constructed as follows;

$S_1 = \{c < d, f < d\}$, $S_2 = \{\}$, $S_3 = \{a < b\}$, $S_4 = \{b < c, l < m\}$, $S_5 = \{\}$, $S_6 = \{k < l\}$, $S_7 = \{a < b\}$, $S_8 = \{b < f, l < m\}$

$L_1 = \{\}$, $L_2 = \{l < m\}$, $L_3 = \{c < d\}$, $L_4 = \{d < g\}$, $L_5 = \{b < c, b < f\}$, $L_6 = \{c < d, f < d\}$, $L_7 = \{f < d\}$, $L_8 = \{d < g\}$

In the next step, SS_i's are constructed as follows;

$SS_1 = \{c < d, b < c, a < b, l < m, k < l, f < d, b < f\}$, $SS_2 = \{\}$, $SS_3 = \{a < b\}$, $SS_4 = \{b < c, a < b, l < m, k < l\}$, $SS_5 = \{\}$, $SS_6 = \{k < l\}$, $SS_7 = \{a < b\}$, $SS_8 = \{b < f, a < b, l < m, k < l\}$

Finally, X_i's are constructed as follows;

$X_1 = \{k < l, a < b\}$, $X_2 = \{b < c, l < m, b < f\}$, $X_3 = \{c < d, f < d\}$, $X_4 = \{d < g\}$,

Note that, searching whether places to add delay pads are included in some other fast paths or slow path is the most computationally expensive part of the algorithm (since DFS is used, this search is linear to the number of constraints and the number of nodes in the circuit). However, this search is inevitable if constraints are interleaved. In this algorithm, this search is performed only once for each constraint. Sorting of constraints are performed with operations on constraint set. This is computationally

less expensive since the number of constraints are expected to be far less than the number of nodes in the circuit. Similarly, checking of constraints are also performed only twice for each constraint (which is also linear to the number of nodes in the circuit). Finally, calculated delay pads have the minimum values.

Let's see how the algorithm works when conflicting set of constraints exist, i.e. there is a cyclic relation between constraints;

$eq_1 = a < b$, $eq_2 = b < c$, $eq_3 = c < d$, $eq_4 = d < a$.

S_i's and L_i's are constructed as follows;

$S_1 = \{d < a\}$, $S_2 = \{a < b\}$, $S_3 = \{b < c\}$, $S_4 = \{c < d\}$.

$L_1 = \{b < c\}$, $L_2 = \{c < d\}$, $L_3 = \{d < a\}$, $L_4 = \{a < b\}$

In the next step, SS_i's and v_i's are constructed as follows;

$SS_1 = \{d < a, c < d\}$, $SS_2 = \{\}$, $SS_3 = \{b < c\}$, $SS_4 = \{c < d, b < c\}$.

$v_1 = \{b < c\}$, $v_2 = \{a < b\}$, $v_3 = \{\}$, $v_4 = \{\}$

Finally, X_i's are constructed as follows;

$X_1 = \{b < c\}$, $X_2 = \{c < d\}$, $X_3 = \{d < a\}$, $X_4 = \{a < b\}$,

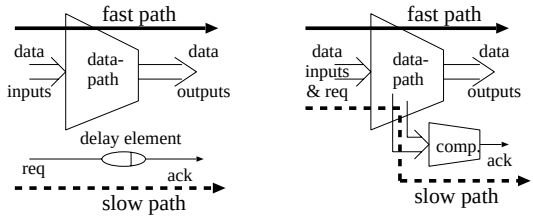
In this way, a circular list of inequalities will be broken at one point; here in this example at the equation $b < c$.

5. Constraints for Asynchronous Data-Path Circuits and Their Verification

In case asynchronous circuits can be divided into control and data-path circuits, verification and correction of data-path circuit can be performed rather easier by focusing on request-acknowledgment signals. This section gives in detail constraints for asynchronous data-path circuits. Again note that after constraints are specified, any constraint violation can be corrected again by adding delay pads into endpoints of slow paths.

Asynchronous data-path circuits communicate with control circuits using request-acknowledgment signals. Therefore, during the timing analysis emphasis should be given to data input-output signals and req-ack signals.

In general, data-path circuits can be divided into two main groups; data-path circuits without idle phase, i.e. bundled data-path circuits in which successive arrival of data is expressed using a strobe signal and data-path circuits with idle phase, i.e. M-out-of-N signaling circuits. Here we use dual-rail circuits as an example for M-out-of-N signaling circuits, but same principles apply for other kinds as well.



a) Single-rail b) Dual-rail

Fig. 4 Timing constraints for single- and dual-rail data-path circuits.

5.1 Single-rail Data-path Circuits

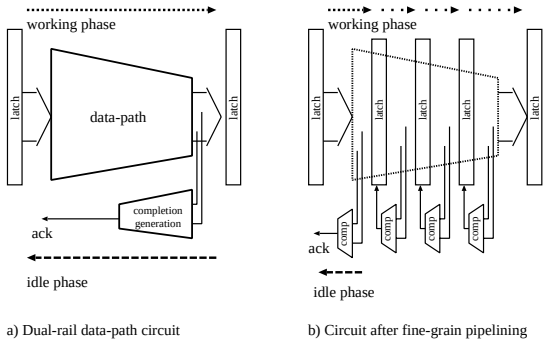
For this kind of circuits, a strobe signal is used to designate successive arrival of data. This strobe signal is directly used as the request signal for the data-path and completion of the operation is designated by ack signal which is generated using an appropriate delay element. The delay value of the delay element should be larger than the largest delay in the data-path. Therefore, start and end points can be decided as shown in **Fig. 4**a. Any violation in the circuit can be corrected by increasing delay element appropriately.

5.2 Dual-rail Data-path Circuits

Dual-rail asynchronous circuits also use request-acknowledgment scheme for communication. However, between successive arrivals of data, spacer (all zeros in data inputs) is inserted into the circuit. This also sets outputs of the circuit to all zeros. In this way data path circuit alternate between working phase (from arrival of data inputs until generation of outputs and ack signal) and idle phase (from inserting spacer into data input until outputs and ack return to zero). As shown in Fig. 1b, if dual-rail data-path circuits are designed according to the SDI model and completion signal is generated from some midpoint in the circuit (not like QDI model where completion is detected after all data-path circuit stabilizes), a new timing constraint arises. Start and end points of slow and fast paths can be decided as shown in Fig. 4b. Again any violation in the circuit can be corrected by adding an appropriate delay pad at the end of completion generation circuit.

5.3 Dual-rail Data-path Circuits with Fine-grain Pipelining

Fine-grain pipelining is a method used for concealing the overhead of idle phase (and even overhead of control logic sometimes) for dual-



a) Dual-rail data-path circuit b) Circuit after fine-grain pipelining

Fig. 5 Fine-grain pipelining of dual-rail data-path circuits.

rail asynchronous data-path circuits [7),8)]. **Figure 5**a shows dual-rail data-path circuits without fine-grain pipelining. Circuit can only start processing of next data after the whole circuit finishes its working phase and then its idle phase.

The basic idea of fine-grain pipelining is to introduce latches into the circuit to divide data-path into substages and let the idle phase of each substage start immediately after data reaches to the next latch without waiting the data to reach to the rightmost latch (Fig. 5b). In this way, idle phase of a substage can be overlapped with the working phases of the succeeding substages; practically concealing the overhead of idle phase. Note that, although structurally similar, this method is different from increasing the number of pipeline stages and data input ratio so that there exist data in each pipeline stage. In order fine-grain pipelining to effectively conceal the idle phase, some of the pipeline stages should be empty during the progress of data [13)].

One concern is the latches inserted into the data-path circuit which cause increase in the latency of the circuit. Differential Domino Logic (DDL) gates [9)] are a special logic family which behave like transparent latches when their one special 'precharge' input is level one. If precharge input is level zero, they can be preset to all zero values quickly. For this reason they are often used for implementing dual-rail encoded asynchronous data-path circuits.

Furthermore, timing optimizations like in the Section 5.2 can be applied within each substage, like early completion generation or checking only a few bits for completion generation instead of whole bits. In this way different high-throughput fine-grain pipelining methods can be obtained [10)].

For fine-grain pipelined circuits, data in each substage should alternate between working and idle phase similar to original whole data-path circuit. Basically there are two sets of timing constraints which are to be guaranteed even after layout. First one (constraint 1) corresponds to the situation that presetting data in current stage should be done only after data is processed and reaches to the next stage, otherwise data disappears at that stage. Second one (constraint 2) corresponds to the situation that after processing the data, reading the next one should be done only after spacer is ready at inputs, otherwise same data would be read twice as a malfunction. If these two constraints are satisfied, data will alternate between working and idle phase exactly once and correct progress of data will be guaranteed.

Fine-grain pipelined data-path circuits are not pure combinational circuits, therefore require special handling to cut cycles in the circuit to able to perform timing analysis. In our previous work [11)], we identified start, end and appropriate intermediate and inhibit points for two mainly used fine-grain pipelining methods separately. **Figure 6** shows the two above constraints for **general** fine-grain pipelining circuits. For DDL gates, inputs are shown at left-hand side, outputs at right-hand side and precharge (pcb) input at the bottom side. For each substage local completion circuits are built and fed to DDL gates for precharging. This input to precharge pins becomes the cause for cycles in the circuit. Therefore, they should be carefully utilized to cut the loops in the circuit for STA analysis and to correctly designate fast and slow paths.

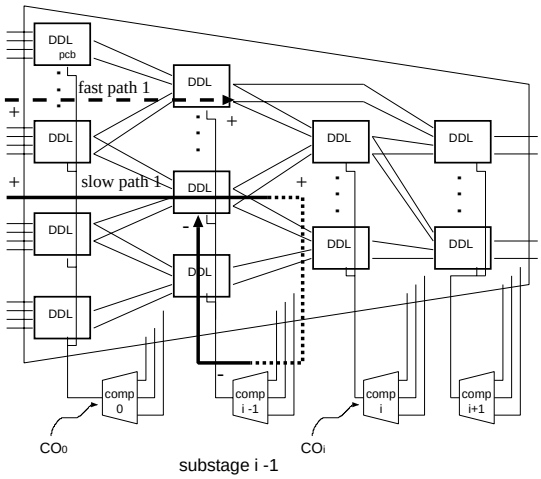
For constraint 1, fast and slow paths can be designated for substage i as follows;

Fast path: ends with rising transition
- **start** inputs of DDL gates except pcb in substage 0; DDL_0^{in}
- **end** outputs of DDL gates in substage i; DDL_i^{out}

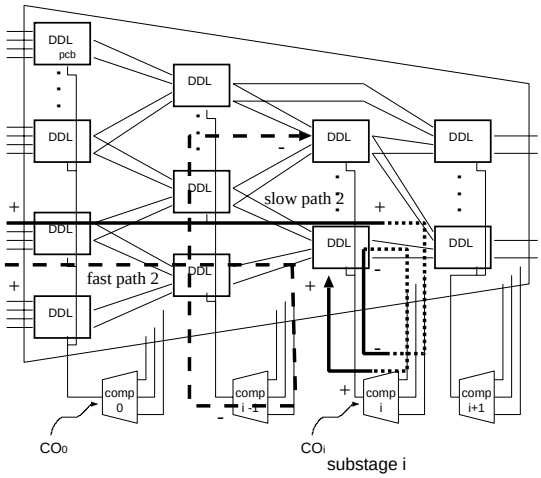
Slow path: ends with falling transition
- **start** DDL_0^{in}
- **end** pcb inputs of DDL gates in substage i; DDL_i^{pcb}
- **intermediate** Output pin of completion circuit i; CO_i

Similarly, for constraint 2;

Fast path: ends with falling transition



(a) Constraint for reading once (Constraint 1)



(b) Constraint for not reading twice (Constraint 2)

Fig. 6 Constraints for fine-grain pipelined circuits.

start DDL_0^{in}
end DDL_i^{in}
intermediate CO_{i-1}

Slow path: ends with rising transition
start DDL_0^{in}
end DDL_i^{pcb}
intermediate CO_i

Outputs of the local completion circuits other than the ones designated as the intermediate points, may be utilized as inhibit points to prevent analysis of the unrelated parts of the circuit.

Let's follow the progress of data in the circuit to affirm these two constraints. Initially all inputs to the circuit are zero and completion circuit outputs are 1 making DDL gates ready to accept data. When data is fed into the circuit it advances through substages. As an example when we consider substage i-1, precharging of data to zero in that substage should be done only after data passes through this stage (constraint 1). After precharging occurs in substage i-1, spacer will be fed into the inputs of substage i. Spacer should be ready at the inputs of substage i before precharge inputs at substage i change into 1 (constraint 2), otherwise previous data will be read twice in substage i. Correct operation will return substage i into original state and make it ready to accept next wave of data.

If any violation occurs in the circuit, delay pads can be added into the end points of slow paths. Note that end point polarities of slow paths are different for constraint 1 and 2. Although fast and slow paths are interleaved, by designating delay pad insertion points as output pins of completion circuit and correctly adjusting delay pads according to polarities, correct delay pad values can be obtained. Also note that, local completion circuits may be in any format, e.g., using completion sensing or delay elements, but this will not effect constraint designation and violation correction basically.

Although constraints for different fine-grain pipelining schemes are shown separately in literature, this is the first time constraints are designated for general case to the authors' knowledge. With the special naming of precharge input pin, constraints for fine-grain pipelined circuits can be generated automatically, even avoiding the need to produce them by synthesis tools (this part is also implemented in our CAD framework).

6. Experimental Results

The general design methodology for asynchronous circuits is shown in Section 3. In our research group, a CAD environment is developed for logic synthesis of asynchronous circuits with dual-rail encoded fine-grain pipelined data-paths, which also includes verification methodology suggested in this paper.

Currently, verification tool is able to check constraints for data-path circuits and suggest delay pad values for any constraint violation. In addition, automatic generation of constraints for fine-grain pipelined circuits is incorporated into the tool environment (*constraint generation* part in Fig. 2). Basically, the special naming of the 'pcb' inputs of DDL gates is utilized to find the output pin of each local completion

Table 1 Verification results for sample circuits.

		delay element		comp. det.
		add	shift	add
S E T 1	Substage 0;	**4.03**	**2.03**	**6.49**
	slow/fast (ps)	1277/317	1246/613	2056/317
	Substage 1;	**2.34**	**1.44**	**3.40**
	slow/fast (ps)	1437/613	1438/996	2087/613
	Substage 2;	**1.94**	**1.32**	**2.63**
	slow/fast (ps)	1657/853	1657/1260	2240/853
	Substage 3;	**1.69**	**1.26**	**2.24**
	slow/fast (ps)	1884/1114	1924/1525	2500/1114
	Substage 4;	**1.53**	**1.19**	**1.97**
	slow/fast (ps)	2172/1417	2109/1765	2792/1417
	Substage 5;	**1.41**	**-**	**1.45**
	slow/fast (ps)	2338/1662	-	2304/1662
S E T 2	Substage 0;	**-**	**-**	**-**
	slow/fast (ps)	-	-	-
	Substage 1;	**1.50**	**2.08**	**2.15**
	slow/fast (ps)	4285/2852	4286/2057	7876/3668
	Substage 2;	**1.83**	**2.15**	**2.38**
	slow/fast (ps)	4503/2464	4503/2094	7917/3324
	Substage 3;	**1.67**	**1.95**	**2.26**
	slow/fast (ps)	4732/2827	4771/2444	8537/3771
	Substage 4;	**1.62**	**1.80**	**2.11**
	slow/fast (ps)	5047/3115	4959/2750	8829/4179
	Substage 5;	**1.51**	**-**	**1.54**
	slow/fast (ps)	5237/3471	-	6894/4479
run time (sec.)		**≈ 1**	**≈ 1**	**≈ 1**
area (mm^2)		**0.104**	**0.050**	**0.104**

circuit in the circuit structure. In this way, the start, end, intermediate and inhibit points can be generated due to Section 5.3 and this information is utilized by verification tool to calculate the delay values for each constraint. Layout data can also be back-annotated in SDF format. The tool applies DFS traversal of the circuit with polarity after start, end, intermediate and inhibit points are designated.

Some example circuits are designed applying fine-grain pipelining, where completion part is for each substage is implemented with delay elements or with completion detection method (they correspond to $LP_{SR}2/2$ and $LP2/2$ in[10] respectively). In fine-grain pipelining with delay elements, the delay value of each stage is matched with a delay element and completion signal is generated thereafter. Whereas in fine-grain pipelining with completion detection, arrival of data to the next stage is sensed by a completion detection circuit and completion signal is generated thereafter. To verify constraints for fine-grain pipelining, 32 bit dual-rail *adder* and *shifter* circuits are designed, placed, routed and checked for constraints. Layout is done by using $0.25\,\mu m$, five layer CB-C10 technology [12]. Automatic placement and routing is performed by Cadence tools and constraint verification is performed using the tool developed.

Table 1 gives verification results for some sample designs on a 2 GHz Pentium 4 machine with 1 GB of memory. The results are grouped into two sets referring to the constraints 1 and 2 in Fig. 6. For each box, lower line gives minimum delay for slow paths/maximum delay for fast paths, whereas upper line gives their ratio which indicates how much allowance exist for these two racing paths. Final two rows in the table give the run times for the tool and areas of corresponding designs. For the sample circuits, verification process takes around one second and constructing the data structure takes around 2 to 5 seconds.

Note that if allowance is set to 2, violations arise for *adder* circuit with completion detection in constraint set 1 at substages 4 & 5 and in constraint set 2 at substage 5 (slow/fast ratio is smaller than 2 for these substages). For these violations, calculated delay pads by the tool are 43, 921 and 1,192 ps respectively, eventually correcting violations.

When start, end and optional intermediate and inhibit points are given along the end-point polarity, the tool can also verify any timing constraint. In addition, the algorithm proposed in Section 4 is implemented and tested on some basic examples.

Currently, CAD environment is being developed to synthesize single-rail data-path circuits besides dual-rail counterparts. Verification part can already handle both kind of dapa-paths.

7. Conclusion

A methodology for verifying timing constraints in asynchronous circuits and a CAD tool framework for verification of constraints and correction of constraint violations for asynchronous circuits have been presented. Violation correction after layout is important since it avoids an expensive back-tracing to logic synthesis phase.

Then a new algorithm is proposed for general asynchronous circuits for constraint verification and violation correction. The algorithm applies a special sorting on timing constraints and enables to determine optimal delay padding to correct violations for interleaved timing constraints.

Concealing the idle phase overhead and generation of fast completion signals are main challenges in asynchronous data-path circuit design. In this work, timing constraints are examined in detail for asynchronous data-path circuits and

guidelines are extracted to practically cover all types of data-path circuits. Fine-grain pipelining is an effective method for concealing idle phase for dual-rail data-path circuits, but also bring new constraints to be met and makes timing analysis more challenging with cycles due to local completion circuits. The presented CAD environment also includes automatic generation of timing constraints for fine-grain pipelined circuits, their verification and correction of constraint violations.

Next step in our work will be the utilization of the new algorithm and CAD environment for more sample circuits.

Acknowledgments This work was supported in part by Information-technology Promotion Agency (IPA) Japan and Semiconductor Technology Academic Research Center (STARC) Japan. The authors would also like to thank members of the project group, especially Yoshiyuki Miyazawa and Yuka Nakagoshi from NEC Informatec Systems, Ltd. and Wataru Takahashi and Kazutoshi Wakabayashi from Multimedia Research Lab., NEC Corporation.

References

1) Seitz, C.L.: System Timing, *Introduction to VLSI Systems*, Mead, C.A. and Conway, L.A.(Eds.), chapter 7, Addison-Wesley (1980).
2) Hauck, S.: Asynchronous Design Methodologies: An Overview, Technical Report TR 93-05-07, Dept. of Computer Science and Engineering, University Washington, Seattle (1993).
3) Nanya, T., Takamura, A., Kuwako, M., Imai, M., Ozawa, M., Ozcan, M., Morizawa, R. and Nakamura, H.: Scalable-Delay-Insensitive Design: A high-performance approach to dependable asynchronous systems, *Proc. Int. Symp. on Future of Intellectual Integrated Electronics*, pp.531–540, Sendai, Japan (Mar. 1999).
4) Takamura, A., Kuwako, M., Imai, M., Fujii, T., Ozawa, M., Fukasaku, I., Ueno, Y. and Nanya, T.: TITAC-2: A 32-bit asynchronous microprocessor based on scalable-delay-insensitive model, *Proc. ICCD 1997*, pp.288–294 (Oct. 1997).
5) Cortadella, J., Kishinevsky, M. and Stevens, K.S.: Synthesis of asynchronous control circuits with automatically generated timing assumptions, *ICCAD 1999*, pp.324–331 (1999).
6) Cortadella, J., Kishinevsky, M., Burns, S.M., Kondratyev, A., Lavagno, L., Stevens, K.S., Taubin, A. and Yakovlev, A.: Lazy transition systems and asynchronous circuit synthesis with relative timing assumptions, *IEEE Trans. on CAD of IC and Systems*, Vol.21, No.2 (Feb. 2002).
7) Furber, S.B. and Liu, J.: Dynamic logic in four-phase micropipelines, *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp.11–16, IEEE Computer Society Press (Mar. 1996).
8) Martin, A.J., Lines, A., Manohar, R., Nystroem, M., Penzes, P., Southworth, R. and Cummings, U.: The design of an asynchronous MIPS R3000 microprocessor, *Proc. Advanced Research in VLSI*, pp.164–181 (Sep. 1997).
9) Griffin, W.R. and Heller, L.G.: Cascade voltage switch logic: a different cmos logic family, *Proc. IEEE ISSCC*, pp.16–17 (Feb. 1984).
10) Singh, M. and Nowick, S.M.: High-throughput asynchronous pipelines for fine-grain dynamic datapaths, *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp.198–209, IEEE Computer Society Press (Apr. 2000).
11) Özcan, M., Imai, M. and Nanya, T.: Generation and verification of timing constraints for fine-grain pipelined asynchronous data-path circuits, *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp.109–114, IEEE Computer Society Press (Apr. 2002).
12) CB-C10 family 0.25 μ CMOS cell-based IC (2.5 V) block library, NEC Corporation (Jul. 1999).
13) Imai, M. and Nanya, T.: Performance comparison of finely pipelined datapaths using differential domino logic, Technical report of IEICE, CPSY99-93, pp.73–80 (Nov. 1999) (in Japanese).

(Received October 16, 2002)
(Accepted March 4, 2003)

Metehan Özcan received the B.S. degree in Computer Engineering and Information Sciences from Bilkent University, Turkey in 1993 and M.E. degree in Electrical and Electronic Engineering from Tokyo Institute of Technology, Japan in 1996. Currently, he is working as a researcher at Research Center for Advanced Science and Technology, the University of Tokyo and pursuing Ph.D. degree at the same institution. His research interests include asynchronous CAD and VLSI design.

Masashi Imai received the B.S. and M.E. degrees in Electrical and Electronic Engineering from Tokyo Institute of Technology, Japan in 1995 and 1997, respectively. Currently, he is working as a research associate at Research Center for Advanced Science and Technology, the University of Tokyo and pursuing Ph.D. degree at the same institution. His research interests include asynchronous CAD and VLSI design.

Hiroshi Nakamura received the B.E. degree in Electronic Eng., M.E. degree in Electrical Eng. and Ph.D. degree in Electronic Eng. from the University of Tokyo, Japan, in 1985, 1987, and 1990 respectively. From 1990 to 1996, he was research associate and then assistant professor in the Institute of Information Sciences and Electronics, University of Tsukuba. From March 1996 to January he has been in the University of California, Irvine as a visiting associate professor. In 1997, he joined the University of Tokyo where he is an associate professor at the Research Center for Advanced Science and Technology. His research interests include high performance computer architecture and high-level design assistance of digital systems.

Takashi Nanya received the B.E. and M.E. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1969 and 1971, respectively, and the Ph.D. degree in electrical engineering from the Tokyo Institute of Technology, Tokyo, Japan in 1978. He was with the NEC Central Research Laboratories from 1971 to 1981, and on the faculty of Tokyo Institute of Technology from 1981 to 1996. In 1996, he joined the University of Tokyo where he is a professor, senator, and the director of the Research Center for Advanced Science and Technology. His research interests include dependable computing, VLSI design and asynchronous computing. He received the IEICE Best Paper Award in 1987, the Okawa Publication Award in 1994, and ASP-DAC Best Paper Award in 1998. He is a vice-chair of IFIP-TC10 and IFIP WG10.4, and a fellow of IEEE and IEICE.