A Note on Optimization Algorithms for FF/Latch-Based High-Level Synthesis

Keisuke Inoue

Department of Global Information and Management
International College of Technology, Kanazawa
2-270, Hisayasu, Kanazawa Shi, Ishikawa Ken, 921-8164, Japan
Email: k-inoue@neptune.kanazawa-it.ac.jp

***Abstract*—This paper presents a new design framework for register-transfer-level data-paths. The conventional D-flip-flop-based register (D-REG) is very practical, since the designers can concentrate only on the timing constraints between registers. However, with the development of deep sub-micron technology and the increase in the data length, the D-REG hardware cost is becoming relatively larger than the other hardware resources. Thus, latch-based design methods have been proposed as alternatives to D-REG-based design methods, since the latch-based register has smaller hardware cost than D-REG. A disadvantage of the conventional latch-based architecture is the increase in the hardware resources. As a result, the total register cost cannot be fully reduced. We propose a new design framework, a kind of level-triggered latch design, in which a D-REG is replaced by a pair of latch-based registers: a master latch-based register (M-REG) and a slave latch-based register (S-REG).**

I. Introduction

In modern very large scale integration (VLSI) design, the design flow becomes a highly sophisticated and complex procedure [1]. In past, most VLSI developers start design from the register-transfer level. However, it becomes difficult to treat the numerous number of VLSI resources (registers, functional units, multiplexers, wires, etc.) by hands. Thus, higher level design will be indispensable also in the future. The process of high-level synthesis (HLS) converts a behavioral description of an application into its register-transfer level. HLS has great potential to improve the quality of VLSIs [2].

HLS using flip-flop (FF)-based register has been widely studied in both academia and industry for many years. The main reason is the convenience of timing verification. The designers can concentrate only on the timing constraints between registers. However, with the development of deep sub-micron technology and the increase in the data length, the FF-based register cost is becoming relatively larger than the other hardware resources [3].

Several methods have been proposed to reduce register cost.

1) Bitwidth-aware high-level synthesis (BIT-H) [4]: changing the bit-width of FF-based register according to the data length during operation scheduling, resource allocation, and resource binding.
2) Register write inhibition by resource dedication (REWIRED) [5]: removing some intermediate FF-based registers during register allocation and binding.
3) Latch-based high-level synthesis framework (HLS-L) [6]: replacing all the FF-based registers with latch-based registers during operation scheduling, resource allocation, and resource binding.

Although most traditional HLS methods focus on uniform-width resources, BIT-H adds awareness of the bitwidth information into the HLS flow, and different bit-width FF-based registers are used to obtain good quality-of-result. After the precise bitwidth information is available, BIT-H binds data to FF-based registers so as to minimize the total bit-widths. However, this method requires accurate input data length analysis, and cannot treat the case that input data could be changed after fabrication. In HLS, data generated by FUs are basically assumed to be stored in registers. Consequently, intermediate registers are required in any design. REWIRED removes intermediate registers, and let generated data by FUs directly propagate to the next FUs. Although REWIRED contributes to reduce registers, it makes FU sharing conditions strict, and could require extra FUs. Level latch design methods were studied in logic-level design as alternatives to the conventional edge-triggered flip-flop design method [7][8]. HLS-L explores the solution space of latch design in HLS design flow, and derives the HLS latch design conditions. However, since HLS-L assumes that all registers are latch-based registers, each task of HLS has to consider the complicated timing condition of latch-based registers.

In this paper, we propose a register clustering method in latch-based HLS in which latch-based registers are grouped into several clusters. In conventional design, FF-based register is required mainly due to the hold timing constraint. We point out that some FF-based registers can be removed without timing violation if considering resource binding. Compared to the conventional full FF-based register design, our design has smaller hardware cost by removing registers. Our method does not require precise data length analysis, extra FUs, and complicated design conditions.

The main contributions of this paper are as follows.

- Proposing a new design framework in which an D-REG is replaced by a pair of latch-based registers: a master latch-based register (M-REG) and a slave latch-based register (S-REG) (Section III).
- A heuristic and an exact algorithms are proposed to decide resource binding with the minimum number of D-REGs (Section IV).
- Discussing M-REG sharing with several S-REG for further optimization (Section V).

II. FF/Latch Mixed Design

FF/latch mixed design has been proposed, where the register type of each register is not restricted to the same one (only FF or latch). Since our proposed method is based on FF/latch mixed desgin, we show a brief summary here. Figure 1 is an example data-flow graph (DFG) where operations o_1 and o_2 are scheduled at 1^{st} and 2^{nd} clock-cycles (CCs), respectively. We focus on the execution of o_1 on datapath. Figure 2-left shows three types of datapaths, where R_1, R_2, R_3 are registers, F is a functional unit (FU), and M is a multiplexer. Symbol '■' represents master latch (M-REG), symbol '□' represents slave latch (S-REG). flip-flop-based register (D-REG) is a concatenation of M-REG and S-REG, and symbol '■□' represents D-REG.

In Fig. 2(a), the left figure shows a datapath, and the right figure shows the timing diagram of executing o_1 in conventional D-REG design. $d_{\max}$ and $d_{\min}$ are the maximum- and minimum-path delays between R_1 and R_3, respectively. Assume that F is shared with o_1 and o_2. To store **b** in R_3 at CC2, M-REG in R_3 must be in the transparent state during the negative clock period in CC1, and S-REG must be in the transparent state during the positive clock period in CC2. Since F is reused by o_2 in CC2, the fastest effect of this reusing violates the output of F (denoted as 'x') after $d_{\min}$. S-REG does not store x since M-REG is in the latched stated in CC2. This is the correct execution of o_1 in D-REG design.

Figure 2(b) shows the design where all the M-REGs are removed. This type of design is called full latch design which is a basic model of HLS-L [6]. Since M-REG in R_3 is removed, data x directly propagates to S-REG in R_3. As a result, wrong data x is stored in R_3. To prevent this timing violation, HLS-L requires additional design constraints that cause the increase in resources.

Finally, we show FF/latch-mixed design as shown in Fig. 2(c). We focus on the fact that timing problem in latch-based design is caused only by the structure of the output register, not by the input register. In this example, M-REGs in both R_1 and R_2 can be safely removed (M-REG in R_3 is still remained). Figure 2(c)-right shows the resulting timing chart which is the same with the correct timing in Fig. 2(a).

From the point of view of timing accuracy, conventional design (a) and FF/mixed design (c) are acceptable. Furthermore, from the point of view of hardware cost, (c) is better than (a) by removing two M-REGs. Our objective is to explore the solution space of the design in which some M-REGs are suitably removed to reduce the hardware cost without a risk of timing error. Note that each register would be an input register for an operation, and the output register for another operation, which is decided by resource binding.

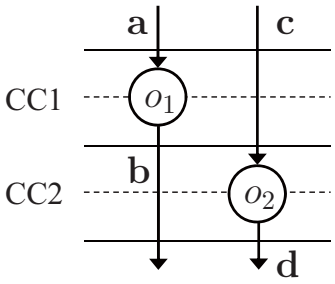
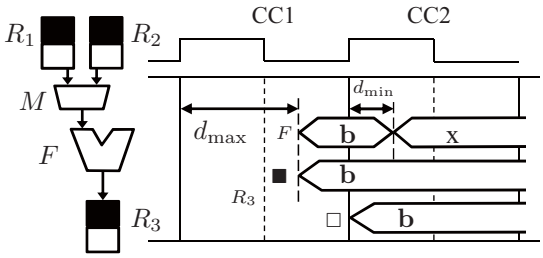
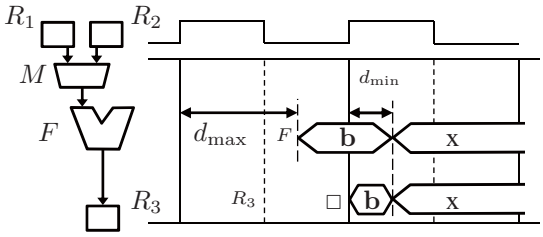


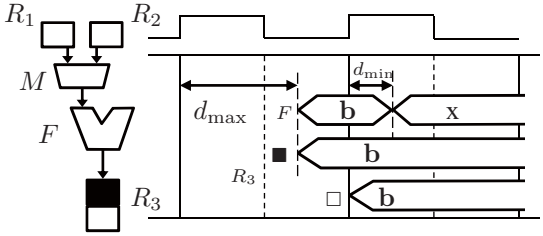
Fig. 1. Example data-flow graph with scheduling result.



(a) conventional D-REG design



(b) full latch design



(c) Proposed design

Fig. 2. Example timing charts in three designs.

III. Proposed High-Level Synthesis

A. Operation Scheduling

Operation scheduling is a HLS task to assign each operation in DFG to the start CC, and the latency is defined as the number of CCs needed for the whole computation. In our architecture, operation scheduling can be done in the same way with the conventional D-REG-based HLS. That is, the execution CCs of each operation o_i can be computed as $\lceil d_i/T_c \rceil$, where d_i is the execution delay of o_i and T_c is the clock period. Each operation is scheduled in a CC with keeping the resource constraint. For each pair of operations, if there is a data-dependency between them. the preceding operation must be scheduled in the earlier CC than the other operation.

B. Functional Unit Binding

The lifetime of an operation is the time duration between the birth and death CCs. It is possible to obtain FU binding with no hold violation with a simple modification in the lifetimes. Each operation should not be changed until one more CC after the death CC of the operation: $t'_o(o_i) = t_o(o_i) + 1$, where $t_o(o_i)$ is the death CC of the lifetime of o_i, and $t'_o(o_i)$ is the modified death CC. It means that the adjacent operations in operation scheduling cannot share the same FU. However, as shown in Sect. II, adjacent operations can share the same FU if the output register is D-REG. In the proposed architecture, the lifetime of o_i, denoted as $T_o(o_i)$ is defined as follows:

(C1) For each operation o_i, if its output is stored in D-REG, $T_o(o_i) = t_o(o_i)$; otherwise, $T_o(o_i) = t_o(o_i) + 1$.

C. Register Binding

The lifetime of a data is the time duration between the CC at which the data is generated and the CC at which the data is finally referenced. In similar to FU binding, register binding could also cause hold violation. For example in Fig. 2, if R_1 is overwritten by another data at CC2, the fastest effect violates $\mathbf{b}$. Each data $\mathbf{a}$ should not be overwritten until one more CC after the death CC of the lifetime: $t'_d(\mathbf{a}) = t_d(\mathbf{a}) + 1$, where $t_d(\mathbf{a})$ is the death CC of the lifetime of $\mathbf{a}$, and $t'_d(\mathbf{a})$ is the modified death CC. If all the outputs of the operations referencing $\mathbf{a}$ lastly are stored in D-REG, the lifetime of $\mathbf{a}$ does not need to be enlarged. In the proposed architecture, the lifetime of $\mathbf{a}$, denoted as $T_d(\mathbf{a})$ is defined as follows:

(C2) For each data $\mathbf{a}$, if all the output data of operations o_i that reference $\mathbf{a}$ lastly (i.e., $t_o(o_i) = t_d(\mathbf{a})$) are bound to D-REGs, $T_d(\mathbf{a}) = t_d(\mathbf{a})$; otherwise, $T_d(\mathbf{a}) = t_d(\mathbf{a}) + 1$.

D. Example

Figure 3(a) shows another example DFG to explain our design. Figure 3(b) is the FU binding solution and register binding solution in the full latch design (i.e., full S-REG design), where a rectangle represents the lifetime of operation/data. In this case, each lifetime must be enlarged by one CC according to (C1) and (C2), and two FUs and two registers are required. Let us assume that R_1 is S-REG, and R_2 is D-REG. The lifetimes of o_1 and $\mathbf{a}$ are shorten by one CC based on the design rules (C1) and (C2). As a result, we can save one FU (Fig. 3(c)). If all the registers are D-REGs (i.e., conventional D-REG design), we can obtain a synthesis result with the same number of resources with our design. However, the hardware cost of our design is better than conventional design by changing R_1 from D-REG to S-REG (i.e., removing M-REG from R_1).

IV. D-REG Minimization Problem and Algorithms

Using D-REG requires not only register costs, but also additional control cost. It is important to minimize the number of D-REGs. As shown in the example in Fig. 3, not every register needs to be D-REG. In practical design, the number of available resources would be given. This fact motivates us to minimize the number of D-REGs (i.e, removing M-REG as much as possible) under the resource constraint.

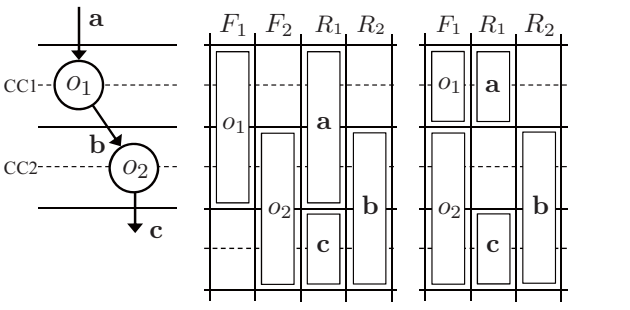
Our target synthesis problem can be written as follows.

Problem1 : Given a scheduled DFG and the resource constraints of FUs and registers, find FU and register binding solutions, so that (C1) and (C2) are met for all the data and operations, resource constraints are met, and the number of D-REGs is minimized. □

A. Heuristic Algorithm

In this section, we propose a heuristic algorithm to obtain near-optimal solutions in polynomial time if DFG is a directed acyclic graph. We use a path-based resource binding approach to perform resource binding in fast time.

Definition1 : The weighted and ordered compatibility graph (WOCG), G_{cg}, is a directed graph. The vertex set is composed of vertices each of which represents a data. For the sake of convenience, we treat a data in DFG and the corresponding vertex in G_{cg} without distinction. Arcs represent the compatibility (i.e., no lifetime overlapping) between data. For each vertex-pair, the arc (a_{i1}, a_{i2}) is added to G_{cg} if a_{i1} and a_{i2} are compatible, and a_{i1} is generated earlier than a_{i2}. $w_v(a_{i1})$ represents the weight on a_{i1}. There is one source node s and one sink node t. □



(a) DFG (b) full latch (c) proposed design

Fig. 3. Example synthesis results.

The following are the definition of each notation for each data $\mathbf{a}$. Suppose that operation $o_{\mathbf{a}}$ outputs $\mathbf{a}$.

$\mathcal{M}_{\mathbf{a}}$: Set of operations o_j, such that o_j is the same operations type with $o_{\mathbf{a}}$, register binding for the output data of o_j is undecided, and $t_o(o_{\mathbf{a}}) = t_o(o_j)$.

$m_{\mathbf{a}}$: The minimum number such that the lifetimes of $m_{\mathbf{a}}$ operations in $\mathcal{M}_{\mathbf{a}}$ cannot be extended due to the current resource constraint.

$\mathcal{N}_{\mathbf{a}}$: Set of data a_j, such that the lifetime of a_j is undecided, and $t_o(o_{\mathbf{a}}) = t_d(a_j)$.

$n_{\mathbf{a}}$: The minimum number such that the lifetimes of $n_{\mathbf{a}}$ data in $\mathcal{N}_{\mathbf{a}}$ cannot be extended due to the current resource constraint of registers.

$\mathcal{Q}_{\mathbf{a}}$: Subset of data in $\mathcal{N}_{\mathbf{a}}$, that are referenced by $o_{\mathbf{a}}$.

$s_{\mathbf{a}}$: The number of operations that reference data $\mathbf{a}$ lastly, and FU binding for these operations is undecided.

$C_{\mathbf{a}}$: The cost for an FU of the same type with $o_{\mathbf{a}}$.

C_R: The cost for a register.

For each data $\mathbf{a}$, the lifetimes of at least $m_{\mathbf{a}}$ operations in $\mathcal{M}_{\mathbf{a}}$ cannot be extended due to the resource constraint. This is equal to choose at least m_a operations from $\mathcal{M}_{\mathbf{a}}$, and bind their output data to D-REG. We provide higher priority to the data who has higher priority to be bound to a D-REG. It can be represented as the following.

$$\frac{m_{\mathbf{a}}}{|\mathcal{M}_{\mathbf{a}}|} \tag{1}$$

With regard to FU binding, this is equal to choose at least $n_{\mathbf{a}}$ data from $\mathcal{N}_{\mathbf{a}}$, and bind the output data of all the operations referencing these data lastly, to D-REGs. The basic weight on $\mathbf{a}$ is defined as $n_{\mathbf{a}}/|\mathcal{N}_{\mathbf{a}}|$ by the same motivation with (1).

It is better to choose a data to bind to a D-REG, such that this binding makes the lifetimes of other data shorten as many as possible. This heuristic depends on the number of operations referencing the data lastly. For example, we should choose a data such that only one operation references it lastly, rather than a data such that two operations references it lastly. The former case requires two D-REGs. The latter case, on the other hand, requires one. It is represented as follows.

$$\frac{n_{\mathbf{a}}}{|\mathcal{N}_{\mathbf{a}}|} * \sum_{a_j \in \mathcal{Q}_{\mathbf{a}}} \frac{1}{s_j} \tag{2}$$

Based on the above observations, we define the vertex weight of $\mathbf{a}$ in G_{cg} as the weighted sum of the metrics for reducing FUs and registers:

$$w_v(\mathbf{a}) = \alpha * C_{\mathbf{a}} * eq(1) + (1 - \alpha) * C_R * eq(2) \tag{3}$$

where α is a real value ($0 \leq \alpha \leq 1$) that represents the importance of each product term, and W^+ is a some positive value in order to include as much operations as possible in a maximum weighted path.

Example : We show the structure of WOCG for an example scheduled graph as shown in Fig. 4(left). Figure 4(right) shows the resulting WOCG, where the vertices are data. The source and sink s and t are connected to all the other data vertices. We add an edge from $\mathbf{a}$ to $\mathbf{c}$ because there is no lifetime overlapping between them. However, we do not add an edge from $\mathbf{a}$ to $\mathbf{b}$ because there is lifetime overlapping between them.

B. Integer Linear Programming (ILP) Formulation

To solve our problem exactly, and to obtain a better knowledge of our heuristic algorithm, we formulate our problem as an ILP. Our ILP uses the four types of 0-1 variables. The following list shows the variables and the necessary-and-sufficient condition each variable takes 1.

- $x_{i,j}$ takes 1 if and only if data a_i is bound to j^{th} register R_j.
- $y_{i,j}$ takes 1 if and only if operation o_i is bound to j^{th} FU of the same type.
- z_i takes 1 if and only if a_i is bound to a D-REG.
- w_j takes 1 if and only if R_j is a D-REG.

Each data a_i must be bound to a register:

$$x_{i,j} = 1 \quad \forall R_j \in \mathcal{R}, \tag{4}$$

where $\mathcal{R}$ is the set of the available registers.

Each pair of data a_{i1} and a_{i2} with lifetime overlapping cannot be bound to the same register R_j:

$$x_{i1,j} + x_{i2,j} \leq 1 \quad \forall R_j \in \mathcal{R}. \tag{5}$$

Each operation o_i must be bound to an FU of the same type:

$$y_{i,j} = 1 \quad \forall f_j \in \mathcal{F}_i, \tag{6}$$

where $\mathcal{F}_i$ is the set of available FUs of the same type with o_i, and f_j is j^{th} FU in $\mathcal{F}_i$.

Each pair of operations o_{i1} and o_{i2} with lifetime overlapping cannot be bound to the same FU f_j of the same type:

$$y_{i1,j} + y_{i2,j} \leq 1 \quad \forall f_j \in \mathcal{F}_i. \tag{7}$$

The design constraint in (C2) in Sect. III can be translated into the following constraint. Let a_{i1} and a_{i2} be a data-pair, such that the birth CC of a_{i2} is the death CC of a_{i1} plus one CC. If these data are bound to the same register, the output data

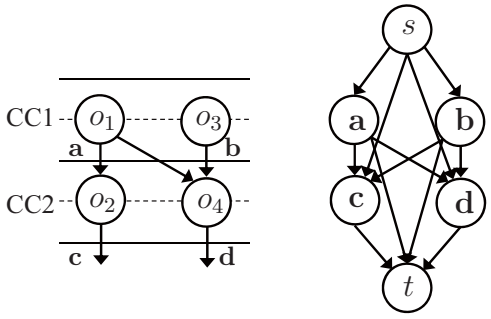


Fig. 4. Example DFG (left) and the corresponding graph for our heuristic algorithm (right).

of all the operations referencing a_{i1} lastly (such an operation set is denoted as $\mathcal{O}_{i1}$) must be bound to a D-REG.

$$x_{i1,j} + x_{i2,j} \leq z_{i3} + 1 \quad \forall o_{i3} \in \mathcal{O}_{i1}. \tag{8}$$

The design constraint in (C1) in Sect. III can be translated into the following constraint. Let o_{i1} and o_{i2} be an operation-pair of the same operation type, such that o_{i1} ando_{i2} are executed in consecutive CCs, and o_{i1} is executed earlier than o_{i2}. If o_{i1} and o_{i2} are bound to the same FU, the output data of o_{i1} must be bound to a D-REG.

$$y_{i1,j} + y_{i2,j} \leq z_{i1} + 1, \tag{9}$$

where a_{i1} is the output data of operation o_{i1}.

Each register R_j becomes a D-REG (i.e., $w_j = 1$) if at least one data a_i with $z_i = 1$ are bound to R_j:

$$x_{i,j} + z_i \leq w_j + 1. \tag{10}$$

The objective is to minimize the number of D-REGs (i.e., removing M-REG as much as possible):

$$\text{Minimize} \quad \sum_{R_j \in \mathcal{R}} w_j. \tag{11}$$

V. M-REG Sharing Theoretic Discussion

So far, we focus on whether each register is D-REG (M-REG + S-REG) or S-REG. To pursue the further optimization, this section discusses the M-REG sharing issue. Let us consider two D-REGs R_x and R_y. These registers have own M-REGs. If R_x and R_y store data at different timing, they can share one M-REG. As a result, one M-REG can be removed for these two registers without risk of timing violation. This fact motivates us to divide registers into some group such that the registers in one group share one M-REG. This new design requires additional condition which can be written as follows.

(C3) For each D-REG pair R_1 and R_2, if they store data at different timing, they can share the same M-REG. □

One possible optimization problem is to minimize the number of D-REGs by M-REG sharing, which can be defined as follows.

Problem 2 : Given a scheduled DFG and the resource binding solution of FUs and registers, find register groups such that each group shares one M-REG, so that (C1) , (C2), and (C3) are met for all the data and operations, resource constraints are met, and the number of M-REGs is minimized. □

We have proven that the following theorem using the polynomial reduction from the graph vertex coloring problem.

Theorem 1 : the computational complexity of Problem 2 is NP-hard. □

VI. Concluding Remarks

In this paper, we proposed a new design framework, a kind of level-triggered latch design, in which an D-flip-flop-based register (D-REG) is replaced by a pair of latch-based registers: a master latch-based register (M-REG) and a slave latch-based register (S-REG). We pointed out that the number of M-REGs can be reduced if resource binding is taken into consideration. Therefore, we considered a resource binding problem to minimize the number of D-REGs under resource constraints. To tackle the problem, we proposed an efficient heuristic-based algorithm which is based on graph theory. Next, to obtain a better knowledge of our heuristic algorithm, and to solve general case input computations, we also formulated our problem as an integer linear programming (ILP). To implement these algorithms with software tool and apply to several benchmark designs is left as a future work.

References

[1] http://community.cadence.com/cadence_blogs_8/b/ii/archive/2014/06/23/dac-2014-high-level-synthesis-hls-users-share-advantages-challenges

[2] Y. Chen, G. Sun, Q. Zou, and Y. Xie, "3DHLS: Incorporating high-level synthesis in physical planning of three-dimensional (3D) ICs," Proc. Design, Automation & Test in Europe (DATE), pp. 1185–1190, 2012.

[3] F. Balasa, H. Zhu, and I.I. Luican, "Computation of storage requirements for multi-dimensional signal processing applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 15, Issue 4, pp. 447–460, April 2007.

[4] J. Cong, Y. Fan, G. Han, Y. Lin, J. Xu, Z. Zhang, and X. Cheng, "Bitwidth-aware scheduling and binding in high-level synthesis," Proc. IEEE Asia and South-Pacific Design Automation Conference (ASP-DAC), pp. 856–861, January 2005.

[5] P Tripathi, R. Jai, S. Kurra, and P.R. Panda, "REWIRED - Register Write Inhibition by Resource Dedication," Proc. IEEE Asia and South-Pacific Design Automation Conference (ASP-DAC), pp. 28–31, March 2008.

[6] S. Paik, I. Shin, T. Kim, and Y. Shin, "HLS-l: A high-level synthesisframework for latch-based architectures," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 29, no. 5, pp. 657–670, May 2010.

[7] C.-L. Chang, I.H.-R. Jiang, Y.-M. Yang, E.Y.-W. Tsai, and A.S.-H. Chen, "Novel pulsed-latch replacement based on time borrowing and spiral clustering," Proc. ACM international symposium on International Symposium on Physical Design (ISPD), 121–128, 2012.

[8] X. Yuan and J. Wang, "Statistical timing verification for transparently latched circuits through structural graph traversal," Proc. IEEE Asia and South-Pacific Design Automation Conference (ASP-DAC), pp. 663–668, Jan. 2010.