Lightening Asynchronous Pipeline Controller Through Resynthesis and Optimization

Jeongwoo Heo
School of Electrical and Computer Engineering
Seoul National University, Seoul, Korea
e-mail: jwheo@snucad.snu.ac.kr

Taewhan Kim
School of Electrical and Computer Engineering
Seoul National University, Seoul, Korea
e-mail: tkim@snucad.snu.ac.kr

Abstract— A bundled-data asynchronous circuit is a promising alternative to a synchronous circuit for implementing high performance low power systems, but it requires to deploy special circuitry to support the asynchronous communication between every pair of consecutive pipeline stages. This work addresses the problem of reducing the size of asynchronous pipeline controller. Lightening the pipeline controller directly impacts two critical domains: (1) it mitigates the increase of controller area caused by high process-voltage-temperature variation on circuit; (2) it contributes to proportionally reducing the leakage power. (Note that a long delay in circuit between pipeline stages requires a long chain of delay elements in the controller.) Precisely, we analyze the setup timing paths on the conventional asynchronous pipeline controller, and (i) *resynthesize new setup timing paths, which allows to share some of the expensive delay elements* among the paths while assuring the communication correctness. Then, we (ii) *optimally solve the problem of minimizing the number of delay elements* by formulating it into a linear programming. For a set of test circuits with a 45nm standard cell library, it is shown that our synthesis and optimization method reduces the total area of delay elements and the leakage power of pipeline controller by 46.4% and 43.6% on average, respectively, while maintaining the same level of performance and dynamic power consumption.

I. Introduction

Over the past few decades, a synchronous circuit design style plays a key role in the proliferation of various VLSI systems due to its simplicity in implementation through synchronization using a global clock network. However, the reliability issue of a global clock network is increasingly important as the demand for low power and high performance design increases. In recent process technologies, it has become much harder to fine-tune a clock skew due to the noise and delay variability caused by process-voltage-temperature (PVT) variation. In addition, the reduction of dynamic power consumption in a global clock network comes up against its limitation with limited supply voltage (V_{dd}) scaling [1], [2]. For these reasons, many researchers are pessimistic about accepting synchronous style implementation for future high speed and low power electronic devices [3].

An asynchronous circuit is one of the alternatives to the conventional design style that is able to cope with the limitation. Contrary to the use of a global clock network in a synchronous circuit, an asynchronous circuit is a design style that exploits handshaking for the communication between circuit components, from which it is expected to mitigate the issues such as a large dynamic power consumption and a high peak current [4], [5]. However, one barrier for the use of asynchronous circuits is that a considerable design effort is demanded, especially for implementing the handshaking mechanism. One of the methods to lower the barrier is employing handshaking control templates [6], [7], [8], and among them, pipeline templates are widely used for high performance applications [9].

Pipeline templates are divided into two categories depending on the use of dynamic logic, which enhances the circuit performance in general. Sutherland and Fairbanks [10] made data transmission faster by adding their custom dynamic logic, and Singh and Nowick [11] devised a pipeline template with full capacity storage using dynamic logic. Additionally, Fant and Brandt [12], Martin and Nystrom [13], and Xia, Ishihara, Hariyama, and Kameyama [14] proposed the methods of generating dual outputs by exploiting dynamic logic. However, considerable care or experience is required to fit dynamic logic into the current design flow in the industry. Although there are a few CAD tools that target optimizing asynchronous circuits [15], [16], they assume a restricted cell library or a specific design style. On the other hand, by using purely static logic, it is possible to fully exploit the current EDA tool chains and standard cell libraries in implementing asynchronous circuits. Sutherland [17] used custom latch and C-element in their pipeline template, and Singh and Nowick [18] proposed MOUSETRAP, a high-performance pipeline template with a transparent latch. Recently, on the top of [18], Ho and Chang [19] significantly reduced glitch power consumption through data transmission by using a normally opaque latch.

Lightening the pipeline controller directly impacts *two domains*: (1) it mitigates the increase of *controller area* caused by large PVT variation; (2) it reduces the *leakage power* consumption. For example, for low V_{dd} operation, an IoT device with ARM cortex-M0+ [21] uses a long delay chain of ring oscillator to take into account the performance gap caused by process variation with V_{dd} scaling, in which the maximum and minimum frequency difference is more than 10X when the temperature changes from 0°C to 75°C for sub-threshold voltage regime. Likewise, maintaining a long delay between pipeline stages caused by PVT variation in an asynchronous

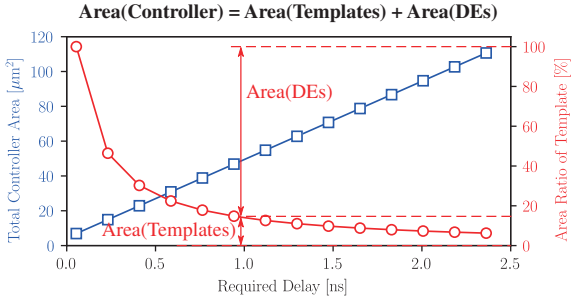

Fig. 1: Changes of total implementation area for a pipeline controller and the ratio of a pipeline template area to total area as the required delay for a single pipeline stage increases. (We measured the area and delay by referring to 45nm NanGate Open Cell Library [20].)

978-1-7281-4123-7/20/$31.00

circuit requires a long chain of delay elements in the controller. The red curve in Fig. 1 indicates that delay elements (DEs) occupy a substantial portion of a pipeline controller as the required delay increases due to variation.

In this paper, we present a new asynchronous pipeline template for two-phase bundled-data protocol. We target lightening the conventional state-of-the-art pipeline template in [19] to impact the two factors in *Domain 1* and *Domain 2* while retaining all benefits (e.g., a considerable saving in glitch power) reaped from the template. Precisely, we *resynthesize new setup timing paths, which allows to share some of the expensive DEs*. (Besides saving DEs, if DEs are to be implemented with the adjustable delay buffers, e.g., [21], [22], etc., their control lines are much simplified as well.) We then *optimally solve the problem of minimizing the total amount of DEs to be inserted* while satisfying timing constraints by formulating it into a linear programming.

II. Preliminaries

A. Bundled-data Asynchronous Circuits

Bundled-data encoding is a coding style that transmits each data bit using exactly one signal wire, and for implementing this, a handshaking mechanism through request and acknowledgement signal wires between two components should be installed, as shown in Fig. 2. A bundled-data asynchronous circuit consists of a datapath, which is the same structure as a synchronous circuit, and a controller, and it is particularly attractive since it has relatively small area overhead compared to its dual-rail counterpart. However, all the timings of datapath operations should be controlled by the delays of handshaking signals (*req, ack*), thus sufficient timing margins should be considered to endure variation.

B. Two-phase Bundled-data Protocol

A two-phase bundled-data protocol operates as shown in Fig. 3(a). A transaction starts with issuing data and making a transition on a request signal by a sender. When this signal is accepted by the receiver, it starts to read the transmitted data and finished the transaction by making a transition on its acknowledgement signal.

C. Conventional State-of-the-Art Pipeline Template [19]

The pipeline template [19], shown in Fig. 3(b), is composed of two XOR gates, a NOR gate, and a resettable transparent latch (pink) at each pipeline stage, all of which are connected to support the communication protocol on that stage. The transactions are performed as follows: initially, let us assume all request and acknowledgement signals are set to 0. Then, the upper XOR is set to 1 while the lower XOR is set to 0, thus the NOR is set to 0, which makes the latch close. Then, as $\text{req}^{(i-1)}$ becomes 1 at the latch input, the upper XOR goes to 0, causing NOR to 1, which makes the latch transparent. Thereafter, the lower XOR becomes 1, causing NOR to 0, which makes the latch close again. This short time interval (i.e., the sum of the NOR and lower XOR delays) during which the latch is transparent, thus reducing glitches, is the biggest merit of this template.[1] After a relatively long period of time through the delay elements (DEs), the request signal req^i of logic value 1 gone to the template on stage $i+1$ comes back as the acknowledgement signal $\text{ack}^{(i+1)}$, which completes the transaction on stage i. The next transaction on stage i is then initiated with the opposite polarities, i.e., the request and acknowledgement signals of logic value 1.

[1]Note that our proposed template structure never enlarges this time interval.

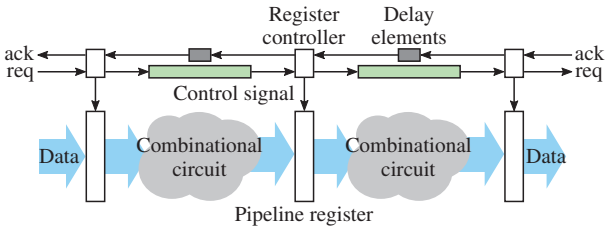

Fig. 2: Structure of a bundled-data asynchronous circuit. Delay elements should be inserted at the long green and short gray bars to build up the setup and hold timing paths, respectively.

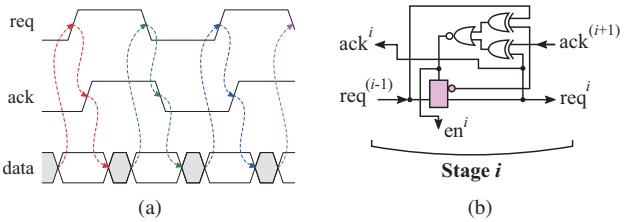

Fig. 3: (a) Two-phase handshaking protocol. (b) Pipeline template in [19].

III. Pipeline Template Resynthesis and Delay Element Optimization

A. Resynthesizing Setup Timing Paths

Fig. 4(a) shows a section of an asynchronous circuit including the conventional pipeline template [19], in which the red and blue paths indicate the setup timing paths for pipeline stages i and $i+1$, respectively. Note that the setup timing path on each stage should be long enough so that its delay can exceed the longest path delay of the combinational circuit on that stage, and the long setup timing path is realized by inserting delay elements (DEs), marked with green color in Fig. 4(a). We see that the two setup timing paths in stages i and $i+1$ are physically completely disjoint in Fig. 4(a), which means the total amount of DEs is exactly the sum of the DEs in all stages. On the other hand, Fig. 4(b) shows our modified setup timing paths, exhibiting *two distinct features*:

1. The red setup timing path is passing through both templates in stages i and $i+1$ while the blue setup timing path is passing through both templates in stages $i+1$ and $i+2$. Thus, the two setup timing paths partially overlap.
2. DEs, marked as yellow triangles, are inserted into a template spot between NOR and upper XOR. We call such DEs *sharing DEs* and the rest of DEs (i.e., green triangles) *non-sharing DEs*.

We show in Sec. III-B that although the new setup timing paths are physically conflicted by *Feature 1*, the logical behavior of all setup timing paths fulfill the original mission. *Feature 2* then provides an opportunity to reduce the total amount of DE resources by allocating as many sharing DEs as possible while satisfying all necessary timing constraints. (The timing constraints for correct operations on asynchronous circuits and the detailed formulation of DE minimization are described in Sec. III-C and Sec. III-D, respectively.) It should be noted that some DEs are also required at the location of the gray rectangle in Fig. 2 to build up the hold timing paths for guaranteeing reliable data sampling. However, since the local enable networks in asynchronous circuits are balanced

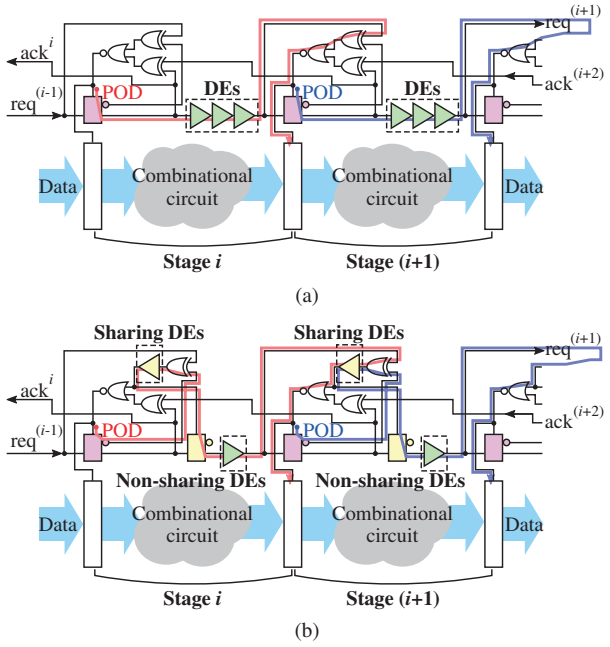

Fig. 4: The asynchronous pipeline circuits using (a) the conventional pipeline template [19] and (b) our proposed pipeline template. The resynthesized setup timing paths are highlighted with red and blue colors, and newly added logic gates are marked with yellow color.

in general and the number of DEs required is very small in comparison with that of setup timing paths, we focus on minimizing DEs on setup timing paths.

B. Validating Logical Correctness of Template Behavior

Fig. 5 shows the logic structure of our newly devised pipeline template, on which the two setup timing paths physically share the upper XOR and sharing DEs. We also allocate one latch (yellow rectangle) at the front of non-sharing DEs in each stage. The role of this latch is to prevent the request signal req^i starting from the ordinary template latch (pink rectangle) from passing through the non-sharing DEs until the latch enable signal $\text{en}^{i\rightarrow(i+1)}$ coming from the path $\text{XOR}^i_{\text{upper}}$ $\rightarrow$ sharing DEs is on. Consequently, the delay of the setup timing path can be maintained by controlling the amounts of sharing DEs as well as non-sharing DEs. Note that en^i changes to low when either of the outputs of two XORs is set to high. In other words, regardless of the delay from sharing DEs, it is achievable to reduce the glitch power consumption through data transmission like the conventional template since the delay of the path $\text{XOR}^i_{\text{lower}} \rightarrow$ NOR is short enough.

Fig. 6 shows a part of the waveforms produced by SPICE simulation. The state change on $\text{req}^{(i-1)}$ (crimson wave) enables signal en^i to make its latch (pink box in Fig. 5) transparent (green wave), which triggers $\text{XOR}^i_{\text{upper}}$ (pink wave). Then,

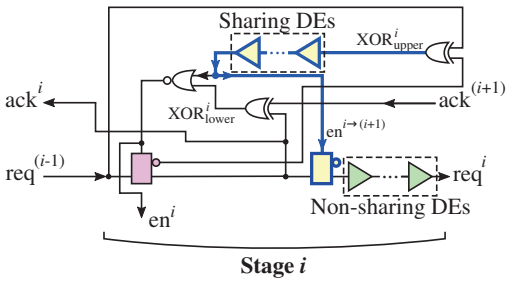

Fig. 5: Logic structure of our pipeline template with DEs.

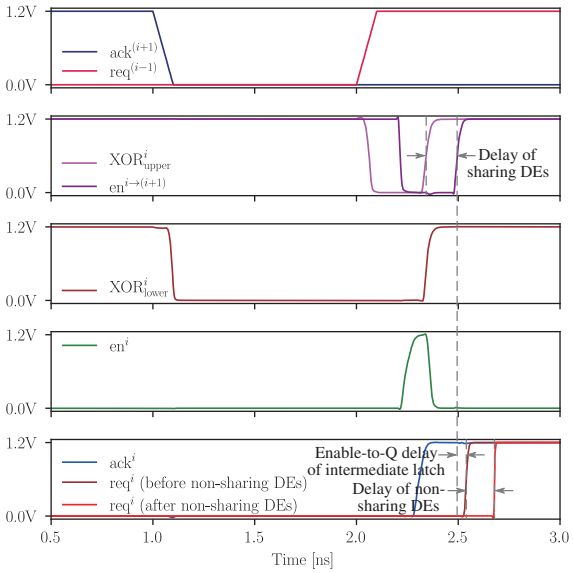

Fig. 6: Simulation waveforms of our pipeline template in Fig. 5.

signal $\text{en}^{i\rightarrow(i+1)}$ is enabled to make its latch (yellow box) transparent (purple wave) after a certain period of time, which is exactly the delay of the sharing DEs (yellow triangles). As $\text{req}^{(i-1)}$ passes through the non-sharing DEs (green triangles), it becomes req^i (red wave). Fig. 7 describes all feasible scenarios of logic behavior of our pipeline template. Two branches from each of logic states A and B take into account the race between the request (from left) and the acknowledgement (from right) signals. The behavior corresponding to the black and blue dashed boxes reveals that our setup timing path is working correctly for sending 1-state and 0-state of request signal, respectively.

C. Reformulating Timing Constraints

Timing correctness of a bundled-data asynchronous circuit is composed of two groups of constraints: template-level constraints and protocol-level constraints [23]. The former assures the quasi-delay insensitive assumption of a pipeline template and the latter assures successful data sampling and transmission. In the following, we formulate the two groups of timing constraints. Table I defines the list of notations used in the formulation.

• **Template-level constraints**: Like the conventional pipeline template [19], our template with DEs should also be hazard-free, for which it has to satisfy the following two constraints.

TABLE I: Definition of notations.

Terms	Definition
D^i_R	Propagation latency of the request signal on stage i
D^i_A	Propagation latency of the acknowledgement signal on stage i
T^i_R	Request-to-enable delay at the controller on stage i
T^i_A	Acknowledgement-to-enable delay at the controller on stage i
D^i_L	Propagation latency of enable signal to the sink on stage i
T^i_C	Enable-to-Q delay of the pipeline register on stage i
D^i_P	Delay of the datapath on stage i
T^i_s	Setup time for the pipeline register on stage i
T^i_h	Hold time for the pipeline register on stage i
$d^i_{\text{DE-S}}$	Delay corresponding to the sharing DEs on stage i
$d^i_{\text{DE-NS}}$	Delay corresponding to the non-sharing DEs on stage i
$d^{i\rightarrow(i+1)}_{\text{latch}}$	Enable-to-Q delay of newly inserted latch on stage i
$Latency^i$	Latency of stage i on the conventional implementation
$Cycle^i$	Cycle time of stage i on the conventional implementation

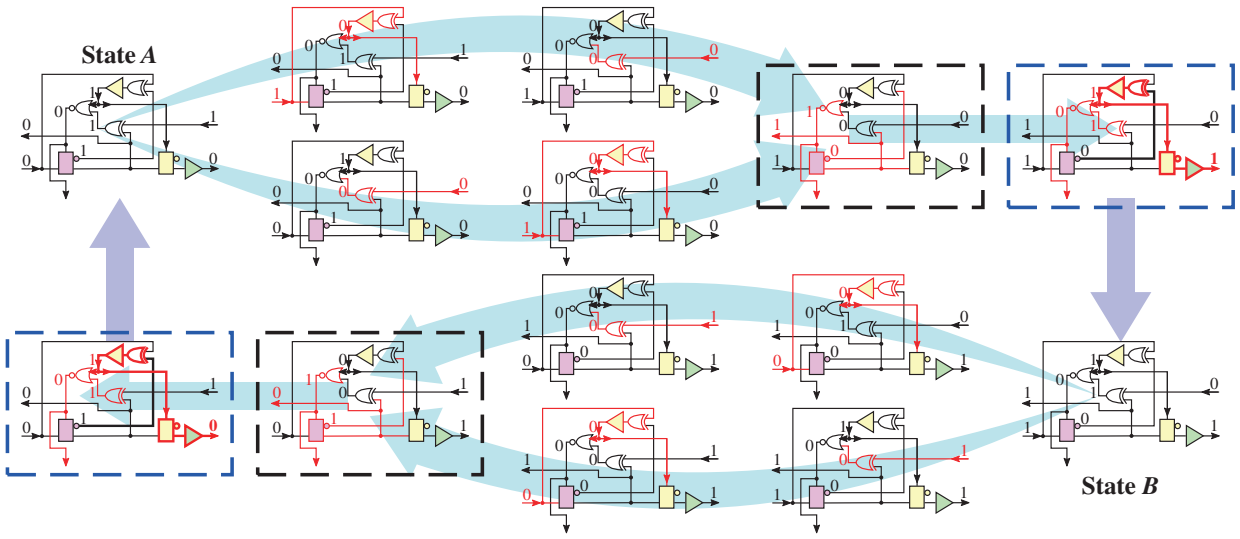

Fig. 7: Logic behavior of our pipeline template in Fig. 5. The parts in red color indicate the change of logic values.

Constraint 1: Once the latch on each pipeline has begun to accept the request signal from its previous stage, *the acknowledgement signal to this stage should not change its state until the template on that stage becomes a suspended (steady) state*. This constraint can be expressed as

$$\begin{aligned}&\min\left\{d^i_{\text{XOR}}+d^i_{\text{NOR}}, d^i_{\text{XOR}}+d^i_{\text{DE-S}}+d^i_{\text{NOR}}\right\}\\&\le d^i_{\text{XOR}}+d^i_{\text{DE-S}}+d^{i\to(i+1)}_{\text{latch}}+d^i_{\text{DE-NS}}+d^{(i+1)}_{\text{XOR}}\\&\quad+d^{(i+1)}_{\text{DE-S}}+d^{(i+1)}_{\text{NOR}}+d^{(i+1)}_{\text{latch}}.\end{aligned}\tag{1}$$

If the delays of the gates of the same type are all identical, the inequality in Eq. (1) can be simplified as

$$0 \le d_{\text{XOR}} + 2\times d_{\text{latch}} + d^i_{\text{DE-S}} + d^i_{\text{DE-NS}} + d^{(i+1)}_{\text{DE-S}}.$$

Constraint 2: Once the latch on each pipeline stage has begun to accept the request signal from its previous stage, *the request signal should not change its state until the template on that stage becomes a suspended (steady) state*. This constraint can be expressed as

$$\begin{aligned}&\min\left\{d^{(i+1)}_{\text{XOR}}+d^{(i+1)}_{\text{NOR}}, d^{(i+1)}_{\text{XOR}}+d^{(i+1)}_{\text{DE-S}}+d^{(i+1)}_{\text{NOR}}\right\}\\&\le d^i_{\text{XOR}}+d^i_{\text{NOR}}+d^i_{\text{latch}}+d^i_{\text{XOR}}+d^i_{\text{DE-S}}+d^{i\to(i+1)}_{\text{latch}}+d^i_{\text{DE-NS}}.\end{aligned}\tag{2}$$

If the delays of the gates of the same type are all identical, the inequality in Eq. (2) can be simplified as

$$0 \le d_{\text{XOR}} + 2\times d_{\text{latch}} + d^i_{\text{DE-S}} + d^i_{\text{DE-NS}}.$$

Note that we should make sure that there is no hazard on the net shared by the setup timing paths, which trivially holds when *Constraint 2* is met. In addition, the enable signal to the new latch we inserted should arrive no earlier than the data signal, which is trivial as well.

• **Protocol-level timing constraints**: For a bundled-data asynchronous pipeline circuit depicted in Fig. 8, its protocol-level timing constraints can be expressed as

$$\begin{aligned}&D^i_R + T^{(i+1)}_R + D^{(i+1)}_L \ge D^i_L + T^i_C + D^i_P + T^{(i+1)}_s,\\&D^i_A + T^i_A + D^i_L + T^i_C + D^i_P \ge D^{(i+1)}_L + T^{(i+1)}_h.\end{aligned}\tag{3}$$

The first and second inequalities in Eq. (3) respectively indicate the setup timing and hold timing constraints for reliable data transmission at the pipeline registers. In the following we reformulate the setup and hold timing constraints, based on the two inequalities in Eq. (3), which should be met for every asynchronous circuit that employs our proposed pipeline template.

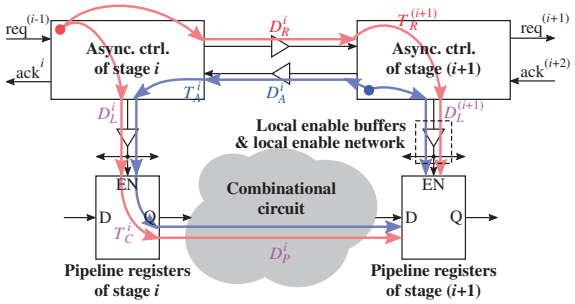

Fig. 8: The view of timing paths on a bundled-data asynchronous circuit. The red and the blue lines respectively denote the launching and the capturing paths of the setup and hold timing paths between stages i and $i+1$.

(1) The point-of-divergence (POD) of setup timing path on each stage is the location (red and blue dots in Fig. 4(b)) at which the enable signal feeds latch. The launching path starts from the POD and terminates at the data pin to the next stage pipeline registers of an asynchronous circuit, passing through the local enable buffers driving the enable signals to the pipeline registers. On the other hand, the capturing path passes through the request signal line and the local enable buffers, finally terminating at the enable pins to the pipeline registers. Thus, the setup timing constraint in Eq. (3) is updated as

$$\begin{aligned}&d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{DE-S}} + d^{i\to(i+1)}_{\text{latch}} + d^i_{\text{DE-NS}} + d^{(i+1)}_{\text{XOR}}\\&+ d^{(i+1)}_{\text{DE-S}} + d^{(i+1)}_{\text{NOR}} + D^{(i+1)}_L \ge D^i_L + T^i_C + D^i_P + T^{(i+1)}_s.\end{aligned}\tag{4}$$

(2) Likewise, the POD of hold timing path on each stage is the location where an enable feeds the next pipeline latch. The launching path starts from the POD, passes through the acknowledgement signal line, the local enable buffers, and the datapath in the current stage finally terminates at the data pin to the pipeline registers on the next stage. On the other hand, the capturing path is quite short, which starts from the same POD, goes through the local enable buffers and terminates at the enable pin to the pipeline registers on the next stage. Hence, the hold timing constraint in Eq. (3) is updated as

$$d^{(i+1)}_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{NOR}} + D^i_L + T^i_C + D^i_P \ge D^{(i+1)}_L + T^{(i+1)}_h.\tag{5}$$

D. Optimally Allocating Delay Elements

Unlike the conventional pipeline templates, our template flexibly distributes DEs to multiple locations of every setup timing path while sharing some DEs among setup timing paths. Consequently, it is possible to minimize the total number of DEs to be inserted while satisfying the template-level and the protocol-level timing constraints reformulated in Sec. III-C. Let $\mathcal{S} = \{0, 1, 2, \cdots\}$ denotes the set of all pipeline stage indexes, then we formulate the DE minimization problem into a linear programming (LP) problem and solve it optimally as

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in \mathcal{S}} \left(d^i_{\text{DE-S}} + d^i_{\text{DE-NS}} \right) \\
\text{subject to} \quad & \text{Eq. } (1), (2), (4), (5), && \forall i \in \mathcal{S}, \\
& d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{DE-S}} + d^{i \to (i+1)}_{\text{latch}} \\
& \quad + d^i_{\text{DE-NS}} + d^{(i+1)}_{\text{XOR}} + d^{(i+1)}_{\text{DE-S}} \\
& \quad + d^{(i+1)}_{\text{NOR}} \leq Latency^i + \epsilon, && \forall i \in \mathcal{S}, \\
& d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{DE-S}} + d^{i \to (i+1)}_{\text{latch}} \\
& \quad + d^i_{\text{DE-NS}} + d^{(i+1)}_{\text{XOR}} + d^{(i+1)}_{\text{DE-S}} \\
& \quad + d^{(i+1)}_{\text{NOR}} + d^{(i+1)}_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{NOR}} \\
& \quad \leq Cycle^i + \epsilon, && \forall i \in \mathcal{S},
\end{aligned}$$

where parameter ϵ is set to a very small value (e.g., 10ps) which is used to avoid the infeasibility of the formulation caused by the discrepancy of gate delay models. The first four ensure the satisfaction of template-level and protocol-level timing constraints, and the last two inequalities prevent the performance degradation caused by inappropriate distribution of DEs. For example, when one additional DE is required to meet the setup timing constraint on stage i, it could be distributed to either $d^i_{\text{DE-S}}$, $d^i_{\text{DE-NS}}$, or $d^{(i+1)}_{\text{DE-S}}$. If all timing constraints of stages $i-1$ and $i+1$ have already been satisfied, the allocation of the DE to $d^i_{\text{DE-S}}$ or $d^{(i+1)}_{\text{DE-S}}$ will make the cycle time or latency unnecessarily longer and deteriorate the overall circuit performance. However, with the consideration of the last two inequalities, LP optimizer is able to select $d^i_{\text{DE-NS}}$ for the DE distribution in such a situation, thereby avoiding the unnecessary degradation of the circuit performance.

IV. Experimental Results

To validate our proposed asynchronous pipeline template, we first produced a set of circuits of pipelined architecture to be used in our experiments by serially linking the multiples (=1, 2, 3, 4) of 5 copies of an ISCAS'85 combinational circuit with 6 (=5+1), 11 (=10+1), 16 (=15+1), and 21 (=20+1) stages. We chose 5 ISCAS'85 circuits C2670, C3540, C5315, C6288, and C7552 for generating a total of 20 asynchronous pipelined circuits. For example, the circuit labeled C2670X5 has 5 copies of combinational circuit C2670, placed each one in between two consecutive pipeline stages. We implemented the asynchronous circuits using 45nm NanGate Open Cell Library [20] with Synopsys Design Compiler and IC Compiler. Then, we extracted the maximum and the minimum datapath delays between the pipeline stages using Synopsys PrimeTime for one specific corner for simplicity.[2] Note that the delay extraction can be easily extended to the case of multi-corner multi-mode scenarios by repeating the whole process. We used IBM CPLEX optimizer [24] through PICOS interface [25] for solving our LP formulation described in Sec. III-D with ϵ=10ps. We also observed the timing behavior and measured the power consumption of controllers (i.e., template with DEs) using Synopsys FineSim.

Table II summarizes the numbers of DEs, total areas of DE implementation, cycle times, latencies, and leakage/dynamic power consumptions of the state-of-the-art pipeline templates with DEs in [19] and our proposed templates with optimized DEs. The first two columns show the comparison of the number of DEs and total area of DEs, i.e., delay buffers and additional latches, from which we see that our controller uses 43.1%∼51.2% less number of delay buffers at the expense of allocating one latch per each pipeline stage. As a result, it reduces the area increase by 46.4% on average. We observed that it took less than one second to solve the LP formulation for all test cases. In addition, DEs accounted for the majority of the total area of pipeline controller in our experiments, as estimated in Fig. 1, but the ratio of pipeline controller area to datapath totally depends on a target design. For example, in our experiments, since the ratio varies from 6.21% to 17.6% when we using the pipeline controller with DE insertion, we believe a significant improvement is achievable for asynchronous circuits with controllers of relatively large size. Fig. 9 depicts the implementation results of the pipeline circuit consisting of circuits C5315, C6288, and C7552 in between pipeline stages, in which our implementation uses 44.9% less number of delay buffers (from 691 to 386) at the expense of just 4 more latches.

The two columns in the middle in Table II show the comparison of the differences between (averaged) cycle times and latencies. We verified that all the maximum differences of cycle times and latencies are less than the sum of ϵ and the delay of one delay buffer, which means we did not harm the original performance. The last two columns in Table II compare the leakage and dynamic power consumptions in pipeline controllers. In short, the leakage power is decreased by 43.6% on average due to reduced DE allocation. Meanwhile, since the dynamic power consumption depends on the amount of internal and external signal transitions regardless of the total cell count, the power consumption remains almost the same level. The little increases of dynamic power consumption is

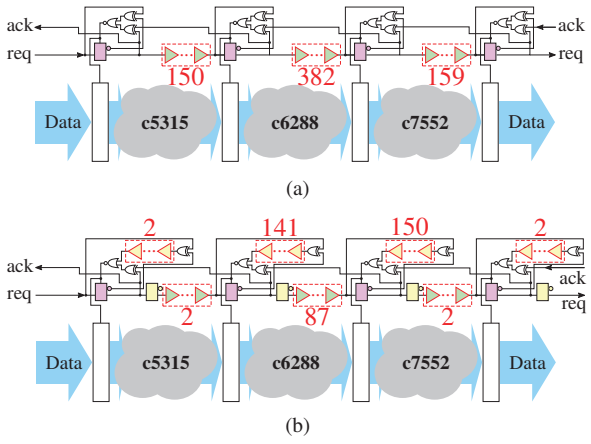

Fig. 9: The asynchronous circuits with (a) the templates and DEs in [19] and (b) ours with optimized insertion, for the pipeline consisting of C5315, C6288, and C7552. The red number denotes the number of delay buffers inserted.

[2]In experiments, temperature is set to 25°C. However, as the temperature goes up to 70 ∼ 80°C while using deep sub-micron technology beyond 14nm, the leakage power consumption could significantly increase.

TABLE II: Comparison of the conventional method and our approach in terms of the number of DEs and the area increase for DE implementation, cycle time, latency, and leakage and dynamic power consumption of pipeline controllers. The number of DEs means the number of delay buffers for [19] while the total number of delay buffers plus the additional latches for ours.

Ckt.	#Buffer+#Latch		Area [μm^2]			ΔCycleTime [ns]		ΔLatency [ns]		Leakage power [μW]			Dynamic power [μW]		
	[19]	Ours	[19]	Ours	Ratio	avg	max	avg	max	[19]	Ours	Ratio	[19]	Ours	Ratio
c2670x5	888+0	496+6	708.624	413.364	0.583	-0.010	0.005	-0.013	0.009	11.639	7.189	0.618	348.731	355.105	1.018
c2670x10	1572+0	777+11	1254.456	652.232	0.520	-0.010	-0.003	-0.013	-0.007	20.642	11.547	0.559	598.008	614.606	1.028
c2670x15	2274+0	1168+16	1814.652	978.880	0.539	-0.013	0.005	-0.017	0.009	29.865	17.254	0.578	848.409	863.748	1.018
c2670x20	2916+0	1424+21	2326.968	1197.798	0.515	-0.008	0.015	-0.012	0.011	38.353	21.306	0.556	1097.281	1129.952	1.030
c3540x5	1404+0	794+6	1120.392	651.168	0.581	-0.013	0.005	-0.016	0.009	17.966	10.843	0.604	344.869	349.081	1.012
c3540x10	2462+0	1220+11	1964.676	1005.746	0.512	-0.010	-0.003	-0.013	-0.007	31.555	16.980	0.538	594.063	601.330	1.012
c3540x15	3563+0	1856+16	2843.274	1527.904	0.537	-0.017	0.005	-0.021	0.009	45.671	25.691	0.563	825.725	831.650	1.007
c3540x20	4642+0	2293+21	3704.316	1891.260	0.511	-0.013	-0.003	-0.017	-0.007	59.518	31.962	0.537	1050.112	1056.003	1.006
c5315x5	1084+0	609+6	865.032	503.538	0.582	-0.003	0.015	-0.006	0.011	14.043	8.575	0.611	346.192	351.122	1.014
c5315x10	1915+0	944+11	1528.170	785.498	0.514	-0.003	0.005	-0.006	0.009	24.848	13.595	0.547	589.807	596.571	1.011
c5315x15	2767+0	1438+16	2208.066	1194.340	0.541	-0.007	0.015	-0.010	0.011	35.910	20.566	0.573	834.526	845.607	1.013
c5315x20	3630+0	1789+21	2896.740	1489.068	0.514	-0.013	-0.003	-0.017	-0.007	47.108	25.781	0.547	1083.376	1110.517	1.025
c6288x5	2694+0	1532+6	2149.812	1240.092	0.577	-0.010	-0.003	-0.013	-0.007	33.785	19.893	0.589	341.892	344.458	1.008
c6288x10	4731+0	2361+11	3775.338	1916.264	0.508	-0.006	0.015	-0.009	0.011	59.379	30.971	0.522	580.736	580.445	0.999
c6288x15	6623+0	3474+16	5285.154	2819.068	0.533	-0.006	0.015	-0.009	0.011	83.195	45.532	0.547	831.866	836.741	1.006
c6288x20	8697+0	4317+21	6940.206	3506.412	0.505	-0.013	0.015	-0.016	0.011	109.242	56.781	0.520	1073.727	1081.467	1.007
c7552x5	1119+0	630+6	892.962	520.296	0.583	-0.023	-0.003	-0.026	-0.007	14.472	8.832	0.610	346.684	351.485	1.014
c7552x10	1952+0	969+11	1557.696	805.448	0.517	-0.004	0.015	-0.008	0.011	25.301	13.902	0.549	595.372	609.790	1.024
c7552x15	2942+0	1526+16	2347.716	1264.564	0.539	-0.009	-0.003	-0.012	-0.007	38.056	21.644	0.569	843.958	858.654	1.017
c7552x20	3796+0	1867+21	3029.208	1551.312	0.512	-0.007	0.015	-0.010	0.011	49.144	26.738	0.544	1088.786	1104.171	1.014
Avgerage	-	-	-	-	0.536	-	-	-	-	-	-	0.564	-	-	1.014

caused by the additional allocation of a single latch on each stage, the switching power of which is a little higher than that of a delay buffer of the same delay.

V. Conclusion

This work addressed a new resource sharing problem on an asynchronous pipeline controller for two-phase bundled-data protocol. Lightening the pipeline controller directly impacted mitigating the increase of controller area caused by high PVT variation and reducing the leakage power. Precisely, we analyzed the setup timing paths on the conventional asynchronous pipeline controller, and *resynthesize new setup timing paths, which allows to share some of the expensive delay elements* among the paths. Then, we *optimally solved the problem of minimizing the number of delay elements* by formulating it into a linear programming. For a set of test circuits with a 45nm standard cell library, it was confirmed that our technique was able to reduce the total area of delay elements and the leakage power of pipeline controllers by 46.4% and 43.6% on average, respectively.

Acknowledgements

This work was supported by the BK21 Plus Project in 2019, the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1E1A1A03070465), Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1703-00, and Samsung Electronics.

References

[1] D. Flynn *et al.*, *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media, 2007.
[2] J. Rabaey, *Low power design essentials*. Springer Science & Business Media, 2009.
[3] G. Frantz, "Giving technology 2020 vision," TI Presentation, 2008.
[4] P. A. Beerel *et al.*, *A designer's guide to asynchronous VLSI*. Cambridge University Press, 2010.
[5] J. Sparsø and S. Furber, *Principles of asynchronous circuit design: A system perspective*. Springer Science & Business Media, 2001.
[6] M. Ferretti and P. A. Beerel, "High performance asynchrnous design using single-track full-buffer standard cells," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 6, pp. 1444–1454, 2006.
[7] D. Hand *et al.*, "Blade–a timing violation resilient asynchronous template," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2015, pp. 21–28.
[8] J. Simatic *et al.*, "A practical framework for specification, verification, and design of self-timed pipelines," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2017, pp. 65–72.
[9] S. M. Nowick and M. Singh, "High-performance asynchronous pipelines: An overview," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 8–22, 2011.
[10] I. Sutherland and S. Fairbanks, "Gasp: A minimal fifo control," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2011, pp. 46–53.
[11] M. Singh and S. M. Nowick, "The design of high-performance dynamic asynchronous pipelines: High-capacity style," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 11, pp. 1270–1283, 2007.
[12] K. M. Fant and S. A. Brandt, "Null convention logic™: A complete and consistent logic for asynchronous digital circuit synthesis," in *IEEE International Conference on Application Specific Systems, Architectures and Processors*, 1996, pp. 261–273.
[13] A. J. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.
[14] Z. Xia *et al.*, "Dual-rail/single-rail hybrid logic design for high-performance asynchronous circuit," in *IEEE International Symposium on Circuits and Systems*, 2012, pp. 3017–3020.
[15] A. Bardsley, *Balsa: An asynchronous circuit synthesis system*. University of Manchester, 1998.
[16] R. B. Reese *et al.*, "Uncle-an rtl approach to asynchronous design," in *IEEE International Symposium on Asynchronous Cricuits and Systems*, 2012, pp. 65–72.
[17] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.
[18] M. Singh and S. M. Nowick, "Mousetrap: High-speed transition-signaling asynchronous pipelines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 684–698, 2017.
[19] K.-H. Ho and Y.-W. Chang, "A new asynchronous pipeline template for power and performance optimization," in *IEEE/ACM Design Automation Conference*, 2014, pp. 1–6.
[20] "Nangate 45nm open cell library," http://www.nangate.com, 2011.
[21] J. Myers *et al.*, "A 12.4pj/cycle sub-threshold, 16pj/cycle near-threshold arm cortex-m0+ mcu with autonomous srpg/dvfs and temperature tracking clocks," in *IEEE Symposium on VLSI Circuits*, 2017.
[22] A. Kapoor *et al.*, "A novel clock distribution and dynamic de-skewing methodology," in *IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 626–631.
[23] G. Gimenez *et al.*, "Static timing analysis of asynchronous bundled-data circuits," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2018, pp. 110–118.
[24] "Ibm ilog cplex optimizer 12.8.0," https://www.ibm.com/analytics/cplex-optimizer, 2017.
[25] G. Sagnol, "Picos, a python interface to conic optimization solvers," Technical Report 12-48, ZIB. http://picos.zib.de, Tech. Rep., 2012.