A Delay Adjustment Method for Asynchronous Circuits with Bundled-data Implementation Considering a Latency Constraint

Kazumasa Yoshimi
The University of Aizu, Japan
m5201119@u-aizu.ac.jp

Hiroshi Saito
The University of Aizu, Japan
hiroshis@u-aizu.ac.jp

Abstract— In this paper, we propose a delay adjustment method for asynchronous circuits with bundled-data implementation considering a latency constraint. In bundled-data implementation, delay adjustment to satisfy timing constraints affects circuit area, performance, and design time. Therefore, in this paper, we define places to insert delay elements so that the area of delay elements is minimized. Then, we modify delay adjustment for hold and idle constraints for an existing method. In the experiments, we evaluate the number of delay adjustments and the number of cells used for delay elements for the circuits obtained by the proposed method. We also evaluate circuit area, execution time, dynamic power consumption, and energy consumption. The experimental results show that the proposed method reduces energy consumption with the reduction of the number of delay adjustments and the reduction of the number of cells used for delay elements.

I. Introduction

Almost all digital VLSIs are synchronous circuits which are controlled by global clock signals. Power consumption by the clock tree and synchronization failures by the clock skew will become remarkable when the semiconductor sub-micron technology improves more and more.

Asynchronous circuits which control circuit components by local handshake signals with request and acknowledge signals do not have problems related to clock signals. However, the design of asynchronous circuits is more difficult than the design of synchronous circuits. It is necessary to choose an appropriate delay model and data-encoding method for the design of asynchronous circuits according to applications. The design method and timing requirements depend on the choice. Nevertheless, available design automation tools for asynchronous circuits are restricted.

Asynchronous circuits with bundled-data implementation represent N-bit signal by N+2 signals [1]. The additional 2 signals are request and acknowledge signals. Data-path circuits are the same as the ones used in synchronous circuits. Delay elements on request signals guarantee timing to write data into registers. Therefore, the circuit performance of bundled-data implementation depends on the delay of the control circuit which includes delay elements.

It is required to adjust delay elements to satisfy timing constraints in asynchronous circuits with bundled-data implementation. Delay adjustment affects circuit area, performance, and design time. Therefore, an efficient delay adjustment is required.

In this paper, we propose a delay adjustment method for asynchronous circuits with bundled-data implementation considering a latency constraint. The proposed method is based on [2] which was proposed by us previously. We developed a design support tool set for asynchronous circuits with bundled-data implementation to implement bundled-data implementations on FPGAs. The design support tool set has constraint generators, timing verifier, and delay adjustment tool. To reduce the area of delay elements and the number of delay adjustments, we modify delay adjustments 1) for hold constraints in which delay elements are added to all input data bits if a hold violation happens in [2] (register-level adjustment) to adjust only violated input data bits (bit-level adjustment) and 2) for idle constraints in which violated paths are modified independently in [2] to adjust violated paths from the first control module to the last one in the control path.

As a related work, Chakraborty and Dill proposed a min/max timing analysis for asynchronous finite state machines in [3]. However, they did not mention delay adjustment.

The rest of this paper is organized as follows. In section II, we describe a circuit model of asynchronous circuits with bundled-data implementation used in this paper. In section III, we describe the proposed method. In section IV, we describe a design flow which uses the proposed delay adjustment method. In section V, we describe experimental results using the proposed method. Finally, in section VI, we describe conclusions and future work.

II. Asynchronous circuits with bundled-data implementation

A. Circuit Model

Figure 1 represents a circuit model of asynchronous circuits with bundled-data implementation used in this paper. This circuit consists of a control circuit and a data-path circuit. The control circuit consists of control modules $ctrl_i$ $(0 \leq i \leq n-1)$. One control module controls one state s_i in the circuit. A $ctrl_i$ consists of a Q-module q_i [4], delay elements sd_i, bd_i, id_i, a glue logic to generate in_i signal for q_i such as C-element [1], a latch dl_i to store the result of the glue logic. The data-path circuit consists of registers (reg_k), multiplexers (mux_m), functional units (fu), delay elements (hd_k, $hd_{in_i mux_m}$), and glue logics. The glue logics in the data-path circuit generate register write signals from ack_i signals and multiplexer control signals from in_i signals.

Delay elements sd_i, bd_i, and id_i are used to satisfy setup constraints, branch constraints, and idle constraints described

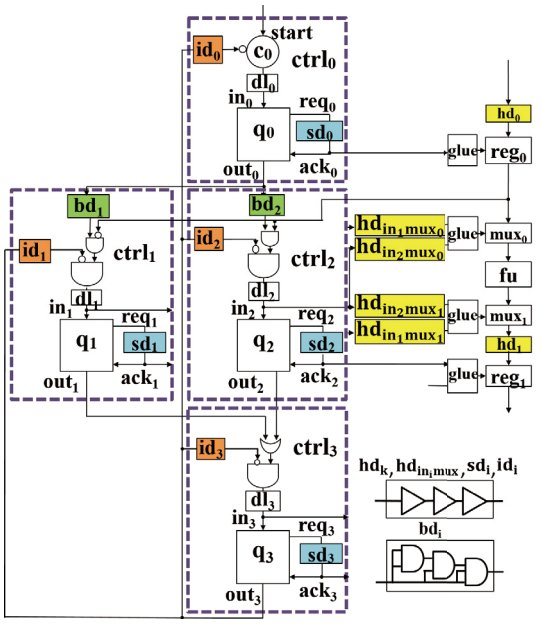

Fig. 1.: Circuit model of asynchronous circuits with bundled-data implementation.

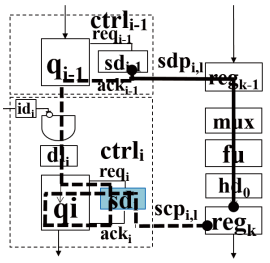

Fig. 2.: Data-path $sdp_{i,l}$ and control path $scp_{i,l}$ for a setup constraint.

in the next sub-section. Delay elements hd_k and $hd_{in_i mux_m}$ are used to satisfy hold constraints. sd_i, id_i, hd_k, and $hd_{in_i mux_m}$ consist of a buffer chain while bd_i consists of a two-input AND gate chain as described in Fig.1.

We describe the behaviors of the circuit model. Hereafter, we represent a rising transition and a falling transition of a signal *sig* as *sig*+ and *sig*−. The circuit starts operations by the arrival of *start*+ from outside to the control circuit. Control module $ctrl_i$ starts the control of data-path resources in state s_i by the arrival of out_{i-1}+. The output of latch dl_i controls multiplexers through glue logics and triggers q_i as in_i. q_i generates req_i+ when in_i+ arrives at q_i. req_i+ is changed to ack_i+ via sd_i and ack_i+ is returned to q_i. q_i generates req_i−. req_i− is changed to ack_i− via sd_i and ack_i− is returned to q_i. ack_i− is used to control registers to write data through glue logics. After ack_i− is returned to q_i, q_i generates out_i+ and transfers the control to the next control module. The last control module $ctrl_{n-1}$ generates out_{n-1}− and initializes all control modules by triggering in_i− and out_i−. We call the initialization of control modules as idle phase *idle*.

B. *Timing Constraints*

Asynchronous circuits with bundled-data implementation in this paper must satisfy the following four types of timing constraints[2].

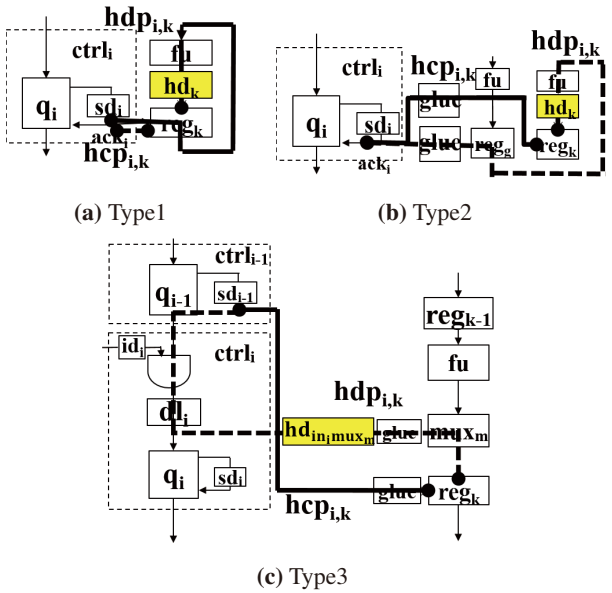

Fig. 3.: Data-path $hdp_{i,k}$ and control path $hcp_{i,k}$ for hold constraints. (a) In Type 1, $hdp_{i,k}$ constitutes a self-loop. (b) In Type 2, $hdp_{i,k}$ is a path to reg_k through a different register reg_g. (c) In Type 3, $hdp_{i,k}$ is a path to dl_i to reg_k through mux_m.

B.1. Setup Constraints

Input data to registers must be stable before the setup time of registers. Figure 2 describes paths related to a setup constraint. $sdp_{i,l}$ represents a data-path from sd_{i-1} to destination register reg_k controlled by $ctrl_i$ through source register reg_{k-1}. $scp_{i,l}$ represents a control path from sd_{i-1} to destination register reg_k through $ctrl_i$. $t_{maxsdp_{i,l}}$ represents the maximum delay of $sdp_{i,l}$. $t_{minscp_{i,l}}$ represents the minimum delay of $scp_{i,l}$. $t_{setup_{i,l}}$ represents the setup time of the destination register. $sm_{i,l}$ ($sm_{i,l} > 0$) represents the margin for $t_{maxsdp_{i,l}}$. The setup constraint can be represented by the following inequality.

$$t_{minscp_{i,l}} > t_{maxsdp_{i,l}} + t_{setup_{i,l}} + sm_{i,l} \quad (1)$$

If the inequality is not satisfied, we need to adjust delay element sd_i.

B.2. Hold Constraints

Input data to registers must be stable during the hold time after writing to registers. Figure 3 describes paths related to hold constraints. $hdp_{i,k}$ represents a data-path from the output of sd_i to the input of reg_k . $hdp_{i,k}$ can be classified into three types. Type 1 (Fig.3(a)) constitutes a self-loop, Type 2 (Fig.3(b)) is a data-path to reg_k through a different register reg_g, and Type 3 (Fig.3(c)) is a data-path from dl_i to reg_k. $t_{minhdp_{i,k}}$ represents the minimum delay of $hdp_{i,k}$. $t_{maxhcp_{i,k}}$ represents the maximum delay of $hcp_{i,k}$. $t_{hold_{i,k}}$ represents the hold time of reg_k. $hm_{i,k}$ ($hm_{i,k} > 0$) represents the margin for $t_{maxhcp_{i,k}}$. The hold constraint can be represented by the following inequality.

$$t_{minhdp_{i,k}} > t_{maxhcp_{i,k}} + hm_{i,k} + t_{hold_{i,k}} \quad (2)$$

If the inequality is not satisfied for Type 1 and Type 2, we adjust delay element hd_k before register reg_k. On the other

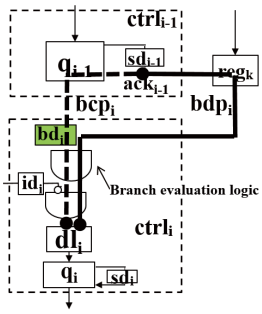

Fig. 4.: Data-path bdp_i and control path bcp_i for a branch constraint.

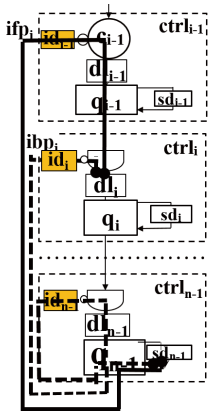

Fig. 5.: Backward path ibp_i and forward path ifp_i for an idle constraint.

hand, if the inequality is not satisfied for Type 3, we adjust delay element $hd_{in_i mux_m}$ between dl_i to mux_m.

B.3. Branch Constraints

A conditional signal from a register for a control branch must arrive at a branch evaluation logic in $ctrl_i$ before the control signal from the previous control module arrives at the branch evaluation logic. Figure 4 describes paths related to a branch constraint. bcp_i represents a control path from sd_{i-1} to dl_i in $ctrl_i$. bdp_i represents a data-path from sd_{i-1} to dl_i through reg_k. t_{minbcp_i} represents the minimum delay of bcp_i. t_{maxbdp_i} represents the maximum delay of bdp_i. bm_i ($bm_i > 0$) represents the margin for t_{maxbdp_i}. The branch constraint can be represented by the following inequality.

$$t_{minbcp_i} > t_{maxbdp_i} + bm_i \tag{3}$$

If the branch constraint is not satisfied, we need to adjust delay element bd_i.

B.4. Idle Constraints

Control modules must be returned to the initial state correctly before the next input data arrives. Figure 5 describes paths related to an idle constraint. ibp_i represents a backward path from sd_{n-1} of the last control module of a control path, $ctrl_{n-1}$, to dl_i in $ctrl_i$ through dl_{n-1} of $ctrl_{n-1}$. ifp_i represents a forward path from sd_{n-1} of $ctrl_{n-1}$ to dl_i of $ctrl_i$ through $ctrl_{i-1}$. t_{maxifp_i} represents the maximum delay of ifp_i. t_{minidp_i} represents the minimum delay of idp_i. im_i ($im_i > 0$) represents the margin for t_{maxifp_i}. The idle constraint can be represented by the following inequality.

$$t_{minibp_i} > t_{maxifp_i} + im_i \tag{4}$$

If the idle constraint is not satisfied, we need to adjust delay element id_i.

III. Proposed Method

In asynchronous circuits with bundled-data implementation, timing constraints must be guaranteed by delay elements. Insertion of delay elements affects not only the quality of synthesized circuits such as area and performance but also the time of design. Therefore, it is required to minimize the number of cells used for delay elements and the number of delay adjustments. In this section, first, we define places to insert delay elements to minimize the area of delay elements. Then, we describe a method for delay adjustments.

A. Insertion Places of Delay Elements

Insertion places of delay elements except hold constraints are based on [2]. sd_i for a setup constraint is inserted on req_i signal as shown in Fig.2. bd_i for a branch constraint is inserted on the signal between q_{i-1} and dl_i as shown in Fig.4 before the evaluation of a control branch using a conditional signal. id_i for an idle constraint is inserted on the feedback signal from the last control module $ctrl_n$ to dl_i in $ctrl_i$. As all of these signals are 1-bit signals, we can reduce the area of these delay elements.

In cases of hold constraints, delay elements hd_k and $hd_{in_i mux_m}$ are inserted to different places according to the types of $hdp_{i,k}$. In cases of Type 1 and Type 2 of $hdp_{i,k}$, we insert delay elements hd_k before the target register reg_k as shown in Fig.3(a) and (b). As the input of reg_k is usually multiple bits, we insert hd_k at bit-level to reduce the area of hd_k. It means that different size of hd_k will be inserted to input data bits where hold constraints are violated. In case of Type 3 of $hdp_{i,k}$, we insert a delay element $hd_{in_i mux_m}$ between in_i and mux_m as shown in Fig.3(c). As the signal between in_i and mux_m is 1-bit signal, we can reduce the area of $hd_{in_i mux_m}$. Compared to [2] where hd_k is inserted before reg_k with not bit-level but register-level, we can reduce the area of hd_k and $hd_{in_i mux_m}$ in the proposed method.

B. Approach for Delay Adjustment

Delay adjustment for each timing constraint consists of addition or removal of cells from the corresponding delay element. The addition of cells is required to satisfy timing constraints. On the other hand, removal of cells is required to keep performance. We explain delay adjustments using the case of setup constraints. Delay adjustments for other timing constraints can be carried out in the similar way.

Delay adjustments for setup constraints are based on the following inequality.

$$\begin{aligned} t_{maxsdp_{i,l}} + t_{setup_{i,l}} + sm_{i,l} < t_{minscp_{i,l}} \\ < t_{maxsdp_{i,l}} + t_{setup_{i,l}} + sm_{i,l} + margin_{scp} \end{aligned} \tag{5}$$

The first condition by the first and second terms of the inequality represents a setup constraint described in (1). If the subtraction of the first term from the second terms is a negative value, it means a setup violation. In such a case, we add cells to sd_i so that the value of the second term becomes larger than the value of the first term. However, we restrict the addition of cells so that the second condition by the second and third terms is satisfied. This is because the addition of more cells results in performance degradation. We restrict the addition of cells by a margin for scp, $margin_{scp}$.

$t_{minscp_{i,l}}$ may over the third term of the inequality if more cells are added to sd_i. In such a case, we remove cells from sd_i so that the second condition of the inequality is satisfied.

The decision of $margin_{scp}$ affects both delay adjustment and circuit performance, a smaller value may result in performance improvement. However, it may require more delay adjustments. Therefore, it is important to decide an appropriate value for $margin_{scp}$. Currently, we decide the value of $margin_{scp}$ based on the delay of cells used for delay elements. Moreover, in all setup constraints, we set the same value to $margin_{scp}$. For hold, branch, and idle constraints, we use $margin_{hdp}$, $margin_{bcp}$, and $margin_{ibp}$ for paths $hdp_{i,k}$, bcp_i, and ibp_i. We assign different values for $margin_{scp}$, $margin_{hdp}$, $margin_{bcp}$, and $margin_{ibp}$. Assigning different values for $margin_{scp}$, $margin_{hdp}$, $margin_{bcp}$, and $margin_{ibp}$ allows us to adjust delay elements for each timing constraint considering the trade-off between the number of delay adjustments and circuit performance.

Different from [2], we adjust id_i from the first control module to the last control module in a control path to reduce the number of delay adjustments. This is because the insertion of id_{i-1} which is located to the previous control module $ctrl_{i-1}$ affects to ifp_i of the idle constraint for $ctrl_i$. By using t_{maxifp_i} with added or removed cells in id_{i-1}, we can adjust id_i for $ctrl_i$ more precisely.

IV. Design Flow

A. Generation of Maximum Delay Constraints

To satisfy a given latency constraint L, we assign path delay constraints. This is because asynchronous circuits do not have a global clock signal. Different from [2] where the maximum delay constraints are assigned to $scp_{i,l}$ and $sdp_{i,l}$ only, we assign the maximum delay constraints for bdp_i, ifp_i, and ibp_i. As $hcp_{i,k}$, $hdp_{i,k}$, and bcp_i coincide some of $scp_{i,l}$ or $sdp_{i,l}$ (or a sub-path of $scp_{i,l}$ or $sdp_{i,l}$), we ignore to generate the maximum delay constraints for $hcp_{i,k}$, $hdp_{i,k}$, and bcp_i.

Path delay constraints are calculated by the delay ratio r_i of a state s_i and the delay ratio r_{idle} of the idle phase *idle* for a given latency constraint L. r_i and r_{idle} are calculated as follows.

$$r_i = \frac{t_{maxsdp_i}}{\Sigma_{i=0}^{n-1} t_{maxsdp_i} + max(t_{maxifp_i}, \ldots, t_{maxifp_{n-1}})} \quad (6)$$

$$r_{idle} = \frac{max(t_{maxifp_i}, \ldots, t_{maxifp_{n-1}})}{\Sigma_{i=0}^{n-1} t_{maxsdp_i} + max(t_{maxifp_i}, \ldots, t_{maxifp_{n-1}})} \quad (7)$$

t_{maxsdp_i} is the maximum one of $t_{maxsdp_{i,l}}$ for data-paths $sdp_{i,l}$ controlled by $ctrl_i$. Currently, we assume that t_{maxsdp_i} and t_{maxifp_i} are obtained by static timing analysis (STA) for a synthesized circuit without path delay constraints. Instead of STA, we may use t_{maxsdp_i} for the data-paths $sdp_{i,l}$ of the synchronous counterpart if we would like to obtain a bundled-data implementation from the synchronous counterpart with the same latency.

The maximum delay constraint for $sdp_{i,l}$, $cmax_{sdp_{i,l}}$, and the maximum delay constraints for $scp_{i,l}$, $cmax_{scp_{i,l}}$, are calculated as follows.

$$c_{maxsdp_{i,l}} = L * DR_{max} * r_i \quad (8)$$

$$c_{maxscp_{i,l}} = L * CR_{max} * r_i \quad (9)$$

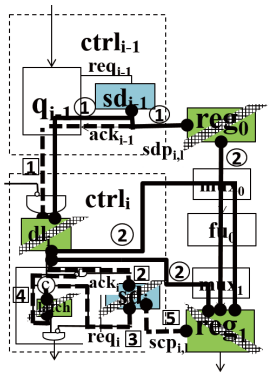

Fig. 6.: Data-path $sdp_{i,l}$ for a setup constraint consists of two sub paths. Control path $scp_{i,l}$ consists of five sub paths.

where DR_{max} and CR_{max} represent the ratio of the maximum data-path delay, and the ratio of the maximum control delay for latency constraint L. DR_{max} and CR_{max} must satisfy $DR_{max} < CR_{max}$. These values must be assigned as inputs with the latency constraint L. Note that if an $sdp_{i,l}$ is a multi-cycle operation, the sum of $c_{maxsdp_{i,l}}$ from the beginning state to the last state of $sdp_{i,l}$ is used.

We decompose $sdp_{i,l}$ into two sub data-paths and $scp_{i,l}$ into five sub control-paths like Fig.6. This is because in commercial STA tools, the start and end points of paths to be analyzed are recommended to set to a primary input, primary output, or register (flip-flop or latch). As an $sdp_{i,l}$ includes two registers (source and destination registers or dl_i and destination register), we decompose $sdp_{i,l}$ into sd_{i-1} to source register and source register to destination register or sd_{i-1} to dl_i and dl_i to destination register (① and ② in Fig.6). We represent the ratio of delay from sdp_{i-1} to source register or sdp_{i-1} to dl_i as $r_{sd_{i-1}2src}$ and the ratio of delay from source register to destination register or dl_i to destination register as $r_{src_{i-1}2dst}$. The maximum delay constraints for each sub data-path are $c_{maxsdp_{i,l}} * r_{sd2src}$ and $c_{maxsdp_{i,l}} * r_{src2dst}$. Similarly, $scp_{i,l}$ is decomposed into sd_{i-1} to dl_i, dl_i to sd_i, sd_i to q_i, q_i to sd_i, and sd_i to destination register (1 to 5 in Fig.6). We represent the ratio of delay from sd_{i-1} to dl_i, the ratio of delay from dl_i to sd_i, the ratio of delay from sd_i to q_i, the ratio of delay from q_i to sd_i, and the ratio of delay from sd_i to destination register as $r_{sd_{i-1}2dl_i}$, $r_{dl_i2sd_i}$, $r_{sd_i2q_i}$, $r_{q_i2sd_i}$, and r_{sd_i2dst}. The values of the maximum delay constraints for five sub control-paths are $c_{maxscp_{i,l}} * r_{sd_{i-1}2dl_i}$, $c_{maxscp_{i,l}} * r_{dl_i2sd_i}$, $c_{maxscp_{i,l}} * r_{sd_i2q_i}$, $c_{maxscp_{i,l}} * r_{q_i2sd_i}$, and $c_{maxscp_{i,l}} * r_{sd_i2dst}$.

In the similar way, we calculate the maximum path delay constraints for bdp_i, ifp_i, and ibp_i.

B. Overview

Figure 7 describes a design flow used in this paper. The flow is based on [2]. The generation of path delay constraints and the delay adjustment are different from [2]. We use the method described in Sec. III. The design flow is partitioned into two processes, initial synthesis, and incremental synthesis.

The inputs of the design flow are 1) a Verilog RTL model of a bundled-data implementation, 2) technology libraries, and 3) a latency constraint (L) with related parameters. The related parameters are DR_{max}, CR_{max}, $sm_{i,l}$, $hm_{i,k}$, bm_i, im_i, $margin_{scp}$, $margin_{hdp}$, $margin_{bcp}$, and $margin_{ibp}$. Initially, we model delay element sd_i for each control module with a buffer. For other

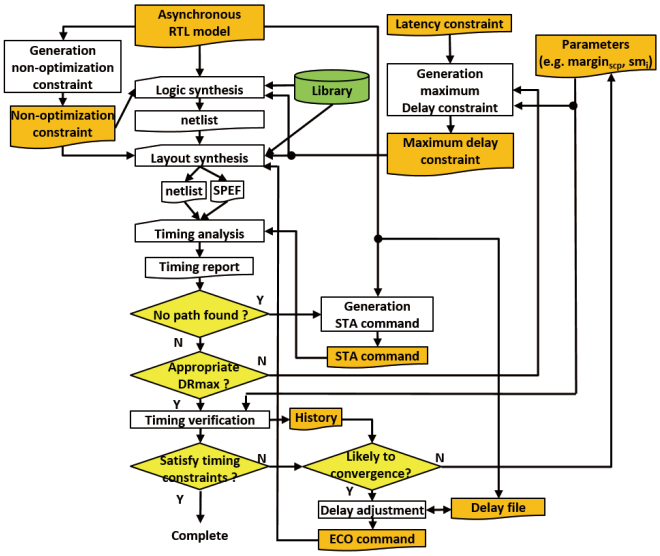

Fig. 7.: Design flow.

delay elements, we insert dummy modules which have a wire from the input port to the output port.

In the initial synthesis, we start from the generation of non-optimization constraints for q-modules q_i and delay elements. This is because optimization for q-modules and delay elements results in timing violations or hazards. We also generate path delay analysis commands (i.e., STA commands) for all paths related to timing constraints. Then, logic synthesis, layout synthesis, and STA are carried out using commercial EDA tools such as Synopsys Design Compiler. After STA, we modify STA commands if the start, end, and through points of paths are renamed during synthesis. Referring to the STA results for the initial synthesis, we decide all of the parameters such as DR_{max}. Note that we assign the values to DR_{max} and CR_{max} to satisfy $DR_{max} < CR_{max}$. In addition, CR_{max} is decided by DR_{max}, $sm_{i,l}$, and $margin_{scp}$.

In the incremental synthesis, we confirm whether DR_{max} is appropriate or not. By comparing with the values of path delay constraints and STA results for corresponding paths, we may re-generate the maximum path delay constraints changing the value of DR_{max} so that the difference between path delay constraints and STA results becomes small. Then, timing verification is carried out for all timing constraints described in Sec. II.B. If all timing constraints are satisfied, we finish the design. Otherwise, we check whether timing constraints tend to be satisfied by referring a history file for timing verification. The history file includes timing verification results for all timing constraints in all synthesis until current timing verification. If timing constraints tend to be satisfied, delay adjustment is carried out based on the proposed method. We generate a script for Engineering Change Order (ECO) commands to add or remove cells to delay elements. Otherwise, we increase the value of $margin_{scp}$ (this is the case when setup constraints do not tend to be satisfied). To increase $margin_{scp}$ restricts removal of cells from delay elements. This results in to increase the possibility to finish delay adjustment although the performance of the synthesized circuit is sacrificed.

V. Experiments

We synthesize a differential equation (DIFFEQ) and an Elliptic Wave Filter (EWF) using the proposed method with a latency constraint. First, we evaluate the number of delay adjustments for idle constraints, the number of re-synthesis, the number of cells used for delay elements hd_k and $hd_{in_i mux_m}$. Then, we evaluate the area, execution time, dynamic power consumption, and energy consumption of the synthesized circuits. For comparison, we synthesize synchronous circuits and bundled-data implementations based on [2]. We modify commands for synthesis and STA generated from [2] where FPGAs are a target to the tools for ASICs. We partially automate timing verification and delay adjustment using Perl and Excel. We represent synchronous circuits, bundled-data implementations based on [2], and bundled-data implementations by the proposed method as "sync", "[2]", and "proposed".

For synthesis, we use Synopsys Design Compiler (I-2013.12-SP2), Cadence EDI (14.2), Synopsys VCS (I-2014.03), and Synopsys PrimeTime (H-2013.06-SP3-5) as logic synthesis, layout synthesis, logic simulation, and STA. We also use eShuttle 65nm cell libraries.

In order to decide latency constraint, we explore synchronous circuits with the minimum clock cycle time which satisfies STA and design rule check. The clock cycle time of both DIFFEQ and EWF is 2,000 ps. We setup latency constraints for DIFFEQ and EWF to 8,000 ps (2,000 * 4 states) and 30,000 ps (2,000 * 15 states).

Next, we decide parameters such as DR_{max}. We start from the decision of DR_{max}. This is because in bundled-data implementation delay elements except id_i are adjusted by data-path delays. We generate the maximum delay constraints for data-paths by reducing DR_{max} 0.05 by 0.05 from 1. After STA for layout synthesis, we select DR_{max} where more than 80 % of the maximum delay constraints between registers is satisfied. DR_{max} of DIFFEQ and that of EWF are 0.80 and 0.85. We assign 200 ps to $sm_{i,l}$, $hm_{i,k}$, bm_i, and im_i. We assign 100 ps to $margin_{scp}$, $margin_{hdp}$, $margin_{bcp}$, and $margin_{ibp}$. We assign CR_{max} of DIFFEQ to 1.00 and of EWF to 1.05. These values are decided from DR_{max}, $sm_{i,l}$, and $margin_{scp}$.

Table I represents the number of incremental synthesis, the number of delay adjustments for id_i, and the number of cells used for hd_k and $hd_{in_i mux_m}$. The second column ("# of syn") represents the number of incremental synthesis to satisfy all timing constraints. The left value of the third column (adj) represents the number of delay adjustments for id_i. The right value of the third column ("all for id_i") represents all possibilities of delay adjustments for id_i obtained by the product of # of syn and the number of control modules (5 control modules for DIFFEQ and 15 control modules for EWF). The left value of the last column ("hd_k") represents the number of cells used for hd_k and $hd_{in_i mux_m}$. The right value of the last column ("total cells") represents the number of cells used for all delay elements.

Compared to [2], the number of delay adjustments for id_i and the number of cells for hd_k and $hd_{in_i mux_m}$ are reduced. The former comes from that delay adjustment for id_i is carried out by the order of the first control module to the last control module. The latter comes from that delay adjustment for hd_k is carried out not register-level but bit-level. They also result in the reduction of the number of incremental synthesis. Delay

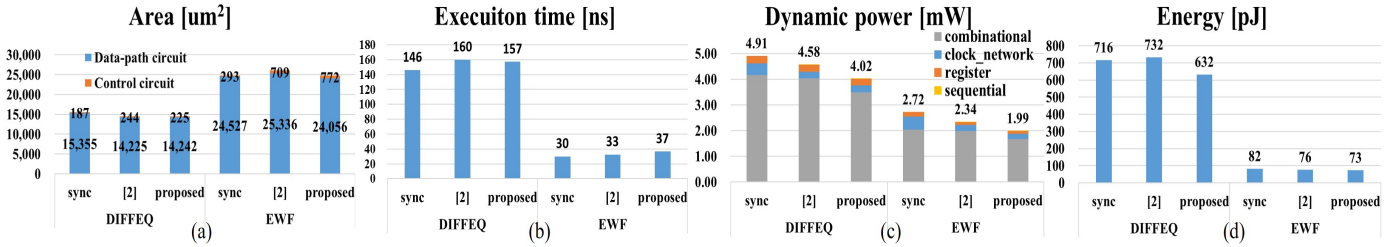

Fig. 8.: Results: (a)area, (b)execution time, (c)dynamic power consumption, (d)energy consumption.

TABLE I: The number of incremental synthesis, the number of delay adjustments for id_i, and the number of cells used for hd_k and $hd_{in_imux_m}$.

	# of syn	adj/all for id_i	hd_k/total cells
DIFFEQ			
[2]	11	17/55	352/411
proposed	3	1/15	15/80
EWF			
[2]	7	34/105	576/814
proposed	3	1/45	69/348

adjustment for some of constraints affects to other constraints (e.g., insertion of hd_k affects setup constraints). Therefore, the reduction of the number of delay adjustments for id_i and the number of cells for hd_k and $hd_{in_imux_m}$ results in the reduction of incremental synthesis.

Figure 8(a), 8(b), 8(c), and 8(d) represent the circuit area, execution time, dynamic power consumption, and energy consumption of the synthesized circuits after all timing constraints are satisfied. The area, execution time, dynamic power consumption, and energy consumption come from a report from EDI, a simulation for a test-bench, a report from PrimeTime, and the product of the execution time and the dynamic power consumption. Note that we assign the density of floorplanning to 70 % in DIFFEQ and 50 % in EWF which are decided by whether the cell location violations happen or not when we start from 70 %.

In the area, compared to "sync", 7.0 % is reduced in DIFFEQ. Compared to "[2]", 4.7 % is reduced in EWF. Control circuit in "[2]" and "proposed" represents the area of the control circuit with control modules and delay elements sd_i, bd_i, and id_i. Delay elements hd_k and $hd_{in_imux_m}$ are included in Datapath circuit.

In execution time, compared to "sync", 7.5 % is increased in DIFFEQ and 23.3 % is increased in EWF. Compared to "[2], 1.9 % is reduced in DIFFEQ and 12.1 % is increased in EWF. The reason why the execution time by "proposed" is the worst in EWF is that CR_{max} is set to 1.05 because of margins for $sm_{i,l}$ and $margin_{scp}$. CR_{max} larger than 1 means that the maximum delay constraints over a given latency constraint L.

In dynamic power consumption, compared to "sync", 18.2 % is reduced in DIFFEQ and 26.9 % is reduced in EWF. The reduction mainly comes from the reduction of clock_network and register due to asynchronous circuits. Compared to "[2]", 12.3 % is reduced in DIFFEQ and 15.0 % is reduced in EWF. Dynamic power consumption from combinational is reduced. This is because operation delays of "proposed" become longer than those of "[2]".

Finally, in energy consumption, compared to "sync", 11.8 % is reduced in DIFFEQ and 11.0 % is reduced in EWF. Compared to "[2]", 13.7 % is reduced in DIFFEQ and 4.0 % is reduced in EWF. The experimental results show that "proposed" reduces energy consumption compared to "[2]" with the reduction of delay adjustments and the reduction of cells used for delay elements.

VI. Conclusions

In this paper, we proposed a delay adjustment method for asynchronous circuits with bundled-data implementation. The proposed method is based on an existing method. However, insertion of delay elements to satisfy hold constraints and delay adjustment for idle constraints are modified. In the experiments, we confirmed that the proposed method reduces energy consumption compared to the existing method with the reduction of delay adjustments and the reduction of cells used for delay elements.

In our future work, we are going to extend the proposed method considering dependencies among timing constraints to reduce the number of delay adjustments more. In addition, we are going to extend the proposed method to deal with pipelined circuits.

Acknowledgements

This work is supported by VLSI Design and Education Center(VDEC), The University of Tokyo with the collaboration with Synopsys Corporation and eShuttle. This work is also supported by JSPS Kakashi 15K00080.

References

[1] J. Sparso and S. Furber, "Principles of asynchronous circuit design: a systems perspective", Springer, 2001.

[2] K. Takizawa, S. Hosaka, H. Saito, "A Design Support Tool Set for Asynchronous Circuits with Bundled-data Implementation on FPGAs", Proc. FPL, pp. 1–4, 2014.

[3] S. Chakraborty, D. I. Dill, K. Y. Yun, "Min-max timing analysis and an application to asynchronous circuits", Proceedings of the IEEE, Vol:87, Issue:2, pp.332–346, 1999.

[4] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T. P. Fang, "QModules: Internally Clocked Delay Insensitive Modules", IEEE Transaction of Computer, vol. C-37, no. 9, pp. 1005–1018, 1988.