# Design of Asynchronous Circuits on Commercial FPGAs Using Placement Constraints

Tatsuki Otake
The University of Aizu, Japan
m5231143@u-aizu.ac.jp

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

**Abstract— In this paper, we propose a design method to implement asynchronous circuits with bundled-data implementation on commercial Field Programmable Gate Arrays (FPGAs) using placement constraints. Using the proposed method, we can obtain the asynchronous circuits whose performance is close to and the energy consumption is smaller (21.3% reduction on average) than the synchronous counterpart with a fewer delay adjustment.**

## I. Introduction

Recently, Field Programmable Gate Arrays (FPGAs) are well used in machine learning or Internet of Things (IoT) applications. Commercial FPGAs used in these fields suppose to implement synchronous circuits which use global clock signals to control circuit components. However, synchronous circuits face the following problems along with the improvement of the semiconductor miniaturization technology. The first problem is the increase in power consumption. The power consumption in the clock network will be increased when the clock signal is distributed to a wide area with high frequency. The second problem is electromagnetic interference. When more and more circuit components are triggered by the clock signal, the peak current is increased which results in the increase of the electromagnetic interference.

One of the solutions is the use of asynchronous circuits. Asynchronous circuits use local handshake signals to control circuit components instead of the global clock signal. Because of the absence of the global clock signal, asynchronous circuits are low power consumption and low electromagnetic interference.

[1], [2], [3], and [4] addressed a design method to implement asynchronous circuits on commercial FPGAs. These researches mainly focused on the implementation of circuit modules which are specific to asynchronous circuits. [5] proposed a design support tool set for Intel FPGAs in which automate constraint generation, timing verification, and delay adjustment which are specific to asynchronous circuits. Using the design support tool set with the design support environment provided by FPGA vendors, we may design asynchronous circuits on commercial FPGAs easily. However, none of these researches addressed the timing closure issue for the design of asynchronous circuits on commercial FPGAs. In fact, the placement of the logic in asynchronous circuits is changed when asynchronous circuits are re-synthesized by delay adjustment to satisfy timing constraints. This makes timing closure difficult with the expected performance.

In this paper, we propose a design method to implement asynchronous circuits with bundled-data implementation on commercial FPGAs using placement constraints. The purpose of the placement constraints is to improve the circuit performance while reducing the number of delay adjustments. We investigate the effect of the number of delay adjustments and the circuit performance by generating placement constraints which are available in the design environment provided by an FPGA vendor.

The organization of this paper is organized as follows. In section 2, we describe asynchronous circuits with bundled-data implementation and FPGAs. In section 3, we describe the motivation to use placement constraints. In section 4, we describe the generation of placement constraints. In section 5, we describe the experimental results In section 6, we describe the conclusion and future work.

## II. Background

### A. *Asynchronous Circuits with Bundled-data Implementation*

Bundled-data implementation is one of the data encoding methods in asynchronous circuits. Hereafter, we call bundled-data implementation as BD circuit. In BD circuits, N-bit data are represented by N+2 signals. Additional two signals correspond to a request signal ($req$) and an acknowledge signal ($ack$). The data-path circuit in BD circuits is the same as the one used in synchronous circuits. The completion of data operations is guaranteed by delay elements on the $req$ signals. Therefore, the circuit performance of BD circuits depends on the delay of the control circuit with delay elements.

Figure 1 shows the circuit model of BD circuits used in this work. The left side of Figure 1 is the control circuit and the right side is the data-path circuit. Control circuit consists of control modules ($ctrl_i$ $0 \leq i \leq n-1$). $ctrl_i$ controls the data operations in the state $i$ of BD circuits and one control module controls one state of circuit. $ctrl_i$ consists of D-latch ($dl_i$), C-element ($c_i$), and Q-module ($q_i$) [6]. The data-path circuit consists of registers ($reg_k$), multiplexers ($mux_l$), and functional units ($fu$). Delay elements in the control circuit ($sd_i$, $bd_i$, $id_i$, and $hdmux_{i,l}$) and the data-path circuit ($hd_k$) are inserted to satisfy the timing constraints of BD circuits.

The circuit model used in this work must satisfy four kinds of timing constraints, setup constraints, hold constraints, branch constraints, and idle constraints [5]. In

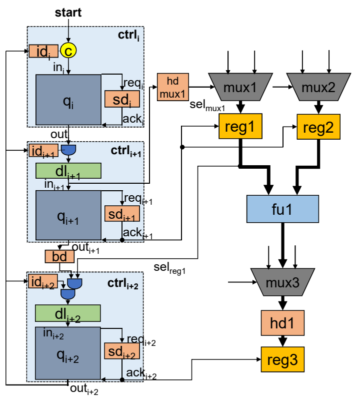


Fig. 1. The circuit model of BD circuits.

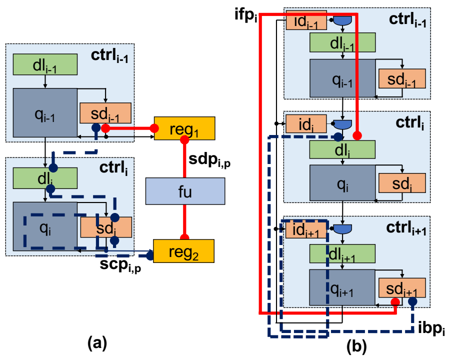


Fig. 2. Paths for timing constraints: (a) the paths for setup constraints and (b) the paths for idle constraints.

this work, we generate placement constraints to optimize the performance of BD circuits. Since the generation is related to setup constraints and idle constraints, we describe these constraints in the followings.

Input data written to registers must be stable before the register write timing. This constraint is called the setup constraint. Figure 2(a) shows the control-path and the data-path related to the setup constraints. The dashed line in Fig. 2(a) shows the control-path $scp_{i,p}$ from the output of $sd_{i-1}$ to the destination resister through $dl_i$ and $q_i$. The solid line shows the data-path $sdp_{i,p}$ from the output of $sd_{i-1}$ to the destination register through data-path resources. Suppose that we represent the minimum delay of $scp_{i,p}$ as $t_{minscp_{i,p}}$, the maximum delay of $sdp_{i,p}$ as $t_{maxsdp_{i,p}}$, the margin for $t_{maxsdp_{i,p}}$ as $t_{sdpm_{i,p}}$, and the setup time of the destination register as $t_{setup_{i,p}}$. The setup constraint can be represented by the following inequality.

$$t_{minscp_{i,p}} > t_{maxsdp_{i,p}} + t_{sdpm_{i,p}} + t_{setup_{i,p}} \qquad (1)$$

If the setup constraint is not satisfied, we need to adjust $sd_i$.

The control signals of $ctrl_i$ must be reset before the next control. To preserve this condition, the reset of the control signal from the previous control module must arrive at $ctrl_i$ before the reset of the control signal from the last control module. This constraint is called the idle constraint. Figure 2(b) shows the forward path and the backward path related to the idle constraints. The dashed line in Fig. 2(b) shows the backward path $ibp_i$ from the output of $sd_{i+1}$ in the last control module $ctrl_{i+1}$ to $dl_i$ in $ctrl_i$. The solid line shows the forward path $ifp_i$ from the output of $sd_{i+1}$ to $dl_i$ in $ctrl_i$ through the previous control module $ctrl_{i-1}$. Suppose that we represent the minimum delay of $ibp_i$ as $t_{minibp_i}$, the maximum delay of $ifp_i$ as $t_{maxifp_i}$, and the margin for $t_{maxifp_i}$ as $t_{ifpm_i}$. The idle constraint can be represented by the following inequality.

$$t_{minibp_i} > t_{maxifp_i} + t_{ifpm_i} \qquad (2)$$

If the idle constraint is not satisfied, we need to adjust $id_{i+1}$.

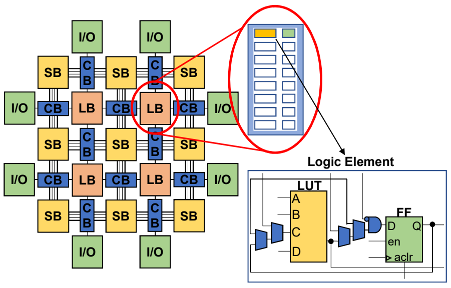


Fig. 3. An example of FPGAs.

## B. FPGA

FPGA is a reconfigurable device where designers can change circuit structure at any time. This flexibility allows designers to change the functional specification freely. In addition, the design cost of FPGAs is low compared to application specific integrated circuits. Therefore, FPGAs are getting to be widely used in IoT and machine learning applications.

Figure 3 shows the general structure of FPGAs. The FPGAs consist of input-output blocks (I/Os), logic blocks (LBs) which construct the logic, switch blocks (SBs) which connect the logic block and connection blocks (CBs). An LB consists of multiple Logic Elements (LEs). The LEs consist of Look Up Tables (LUTs) to implement a combinational logic and Flip Flops (FFs) to store data. In addition to these basic components, recent FPGAs include multipliers, block memories, and hard macros such as an ARM processor.

## III. Motivation

We face the following two problems when we design BD circuits on commercial FPGAs.

The first problem is that it takes a long time to satisfy all timing constraints. This is because the placement is changed when we re-synthesize BD circuits after the delay adjustment to solve timing violations. Figure 4 (a) and (b) show the placement before and after a re-synthesis.

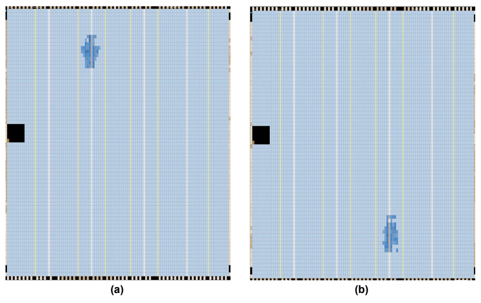


Fig. 4. Change of the placement : (a) before re-synthesis and (b) after re-synthesis.

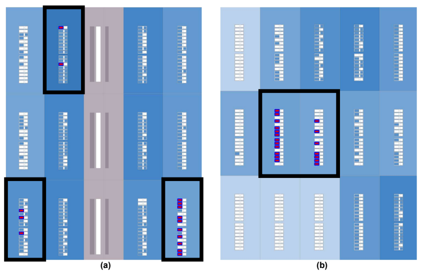


Fig. 5. Placement in $ctrl_i$ : (a) without a placement constraint, (b) with a placement constraint.

Because the delays in BD circuits are also changed by the change of the placement, new timing violations may happen even though some violations are fixed by the delay adjustment.

The second problem is the degradation of the circuit performance of BD circuits. The cause of this problem is the placement of the logics in the same resource to the different locations. For example, in Fig. 5 (a), the logics in $ctrl_i$ are placed in different logic blocks. As a result, the circuit performance is decreased because the wire delay between the logics become long.

Above two problems may be mitigated to apply placement constraints for each logic in BD circuits. For example, if the logics which are included in $ctrl_i$ are placed in the adjacent logic block like Fig. 5 (b), the wire delay between logics will be reduced. This is the motivation of the proposed method which generates placement constraints to implement BD circuits on commercial FPGAs while optimizing the performance with the reduction of the number of delay adjustments.

## IV. Proposed Method

In this work, we assume to implement BD circuits on Intel FPGAs. Therefore, first, we generate placement constraints for the BD circuits. Then, we propose a design flow to design the BD circuits on the Intel FPGAs.

### A. *Generation of Placement Constraints*

The purpose of placement constraints is to improve the circuit performance of the BD circuits while reducing the number of delay adjustments. In the Intel FPGAs, designers can use Design Partition (DP), Logic Lock (LL), and Location Assignment (LA) as placement constraints. In this work, we focus on DP and LL only. We use DP to reduce the number of delay adjustments and LL to improve the circuit performance.

In general, a floorplan method is used to generate placement constraints. However, the quality of the floorplan method depends on the accuracy of the delay estimation. On the other hand, since the data-path circuit of BD circuits are almost the same as synchronous circuits, we can rely on the placement by the commercial FPGA design support tool. From this observation, we only consider a floorplan method of the control circuit based on the connections of control modules.

#### A.1. Design Partition

Design Partition (DP) [7] is one of the functions supported in the Intel Quartus Prime. Using the DP, we can assign the keeping level of placement and routing before and after re-synthesis. Therefore, we can expect that the placement of the circuit components in the BD circuits is preserved before and after re-synthesis. This results in the reduction of the number of delay adjustments.

In this work, we generate a DP for each top-level module, $ctrl_i$, and delay element. By generating the DP separately, we can restrict logic optimization such as gate sharing. A DP for each $ctrl_i$ is to avoid the logic optimization for the logics in $ctrl_i$ except delay elements.

On the other hand, we do not generate a DP for each data-path resource. This is because to allow the logic optimization for data-path resources as much as possible. Instead, we generate a DP for the top-level module to control the keeping level of placement and routing before and after re-synthesis.

In the proposed method, we generate two TCL scripts related to the DP. The first one is used for the initial synthesis. In the first TCL script, all DPs are set to “Post-Synthesis” which keeps the logic structure before and after re-synthesis. The second one is used with adjusted delay elements. In the second TCL script, DPs except for delay elements are set to “Post-Fit” which keeps the placement and routing before and after re-synthesis. We do not set “Post-Fit” to the delay elements because the number of cells used in the delay elements is changed after a delay adjustment. It takes a long time for synthesis if the DPs for the delay elements are also set to “Post-Fit”, because the location for the addition of new cells in the delay elements is strictly restricted.

Figure 6 shows an example of the DP commands. Figure 7 shows the placement of before and after re-synthesis when we use DP. The location of the circuit components except delay elements is unchanged.

#### A.2. Logic Lock

Logic Lock (LL) is one of the placement constraints supported in the Quartus Prime. Using LLs, designers can assign the coordinate of the region for the placement of resources.

```
1. set_global_assignment -name PARTITION_NETLIST_TYPE POST_SYNTH or POST_FIT
   -section_id "instance name"
2. set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
   PLACEMENT_AND_ROUTING -section_id "instance name"
```

Fig. 6. Example of DP commands.

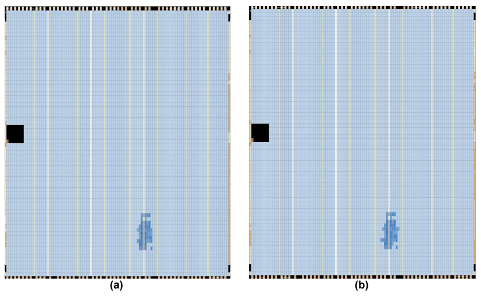


Fig. 7. Placement with DPs: (a) before re-synthesis and (b) after re-synthesis.

In this work, we generate an LL for each top-level module, $ctrl_i$, and register. By generating the LLs, we avoid placing the logics of these resources to different locations. Especially, since the performance of BD circuits depends on the delay of the control circuit, the generation of the LL for each $ctrl_i$ results in the prevention of long wires inside $ctrl_i$. Moreover, different from synchronous circuits where minimization of delays between registers is the main concern for optimization, not only the paths between registers but also the paths between $sd_i$ and registers also affect the performance of the BD circuits. Therefore, by generating the LL for each register, we reduce the delay variation between $sd_i$ and registers. Note that $hd_k$ which is a delay element for the hold violation of $reg_k$ is included to the LL of $reg_k$ to avoid long wires between $hd_k$ and $reg_k$.

We manually decide the coordinate of the LLs for $ctrl_i$. This is because the delay of the control circuit decides the performance of BD circuits. We decide the coordinate of the LLs as follows. Suppose that the region size of the LLs is already decided from the estimated circuit area. For all other LLs, we just generate a region (i.e., rectangle) from the estimated circuit area and ask the Quartus Prime to decide the coordinate automatically.

- Make a control finite state machine (FSM) for the target BD circuit
- Add additional directed arcs to the FSM in which represent the signal transitions from the last state to other states for the initialization of the control signals (this corresponds to idle constraints)
- Select state $i$ whose incoming and outgoing arcs is the maximum and allocate the base coordinate (e.g., center coordinate) to the LL of $ctrl_i$
- Repeat until the coordinate of the LLs for all $ctrl_i$ is determined
  - Select state $i$ which has the most incoming and outgoing arcs for the already allocated states
  - Allocate the coordinate of the LL for $ctrl_i$ such that the sum of the hamming distance for the coordinate of the already allocated states becomes the smallest

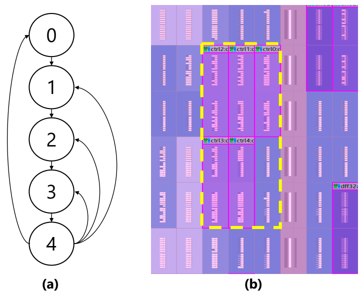
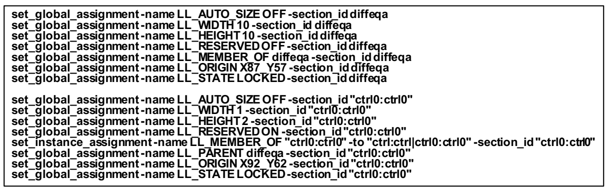


Fig. 8. The decision of the coordinate for $ctrl_i$: (a) a control FSM and (b) the placement result.



```
set_global_assignment -name LL_AUTO_SIZE OFF -section_id diffeqa
set_global_assignment -name LL_WIDTH 10 -section_id diffeqa
set_global_assignment -name LL_HEIGHT 10 -section_id diffeqa
set_global_assignment -name LL_RESERVED OFF -section_id diffeqa
set_global_assignment -name LL_MEMBER_OF diffeqa -section_id diffeqa
set_global_assignment -name LL_ORIGIN X87_Y57 -section_id diffeqa
set_global_assignment -name LL_STATE LOCKED -section_id diffeqa

set_global_assignment -name LL_AUTO_SIZE OFF -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_WIDTH 1 -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_HEIGHT 2 -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_RESERVED ON -section_id "ctrl0:ctrl0"
set_instance_assignment -name LL_MEMBER_OF "ctrl0:ctrl0" -to "ctrl:ctrl|ctrl0:ctrl0" -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_PARENT diffeqa -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_ORIGIN X92_Y62 -section_id "ctrl0:ctrl0"
set_global_assignment -name LL_STATE LOCKED -section_id "ctrl0:ctrl0"
```

Fig. 9. A TCL script which includes the commands for LLs.

The coordinate of $ctrl_i$ is decided by the above procedure such that $ctrl_i$ which has a connection is placed to the adjacent location. We assume that the base coordinate is the center of the LL for the top-level module. This results in that the wire delay between $ctrl_i$ and register is balanced in all connections.

Figure 8(a) shows a control FSM for a BD circuit. Figure 8(b) shows the placement result with the LLs whose coordinate is decided by the above procedure. Figure 9 shows a part of a TCL script which includes the commands for the LLs of the $ctrl_0$. $LL\_WIDTH$ and $LL\_HEIGHT$ show the width size and height size of the LL region. $LL\_RESERVED$ means that other resources can be placed to the LL (OFF) or not (ON). $LL\_origin$ means that the coodinate of region.

## B. Design Flow

Figure 10 shows the proposed design flow. The proposed design flow is based on [5]. It is extended to synthesize BD circuits using DPs and LLs. This proposed design flow assumes that designers synthesize the BD circuits more than 3 times. Note that CONSTGEN, MAXCONSTGEN, DELAYGEN, REPTIMING, TIMINGCHECKER, and DELAYADJUST are design support tools proposed by [5]. These tools automate delay constraint generation, timing verification, and delay adjustment specific to the BD circuits.

In the proposed design flow, designers need to prepare the RTL model of BD circuits using Verilog Hardware Description Language (HDL). In addition to the RTL model

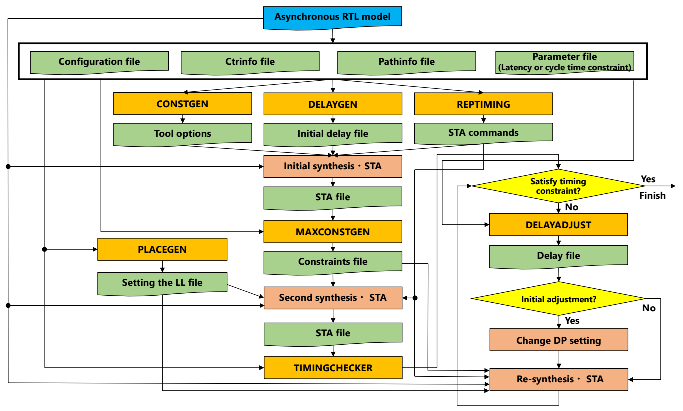


Fig. 10. The proposed design flow.

TABLE I
The designed BD circuits.

| name | DP | LL | Coordinate |
|---|---|---|---|
| *no_no* | No | No | - |
| *dp_no* | Yes | No | - |
| $dp\_ll_{t,ca,r}$ | Yes | top module, $ctrl_i$, reg | Auto |
| $dp\_ll_{t,cf,r}$ | Yes | top module, $ctrl_i$, reg | Fix top module and $ctrl_i$ |

TABLE II
The number of delay adjustments.

| name | DIFFEQ | EWF | IDCT |
|---|---|---|---|
| *no_no* | n/a | 8 | n/a |
| *dp_no* | 2 | 3 | 1 |
| $dp\_ll_{t,ca,r}$ | 3 | 3 | 6 |
| $dp\_ll_{t,cf,r}$ | 3 | 5 | 6 |

of the BD circuits, designers need to prepare four files, configuration file, ctrlinfo file, pathinfo file, and parameter file. They represent the circuit information such as control finite state machine, paths related to timing constraints, etc and the parameters such as the delay of a cell used for delay elements. Through CONSTGEN, DELAYGEN, and REPTIMING, we generate a set of synthesis options, a set of the initial delay elements, and a set of Static Timing Analysis (STA) commands. Moreover, we prepare a DP for each top-level module, $ctrl_i$, and delay element with "Post-Synthesis".

The purpose of the initial synthesis is to obtain the parameters such as circuit area (i.e., the number of LEs) and delay to generate constraints. Using MAXCONSTGEN with a given latency constraint and the obtained delay parameters, we generate the maximum delay constraints (*set_max_delay*) for each path and the clock constraints (*create_clock*) for each $ack_i$ signal. In addition, we generate an LL for each top-level module, $ctrl_i$, and register with the obtained circuit area parameters. PLACEGEN in Figure10 represents the generation of LLs.

The second synthesis is performed using the prepared constraints. In the second synthesis, the logics of the resources with the LLs are placed to adjacent LABs.

After the second synthesis, we verify timing constraints such as setup constraints using TIMINGCHECKER and generate delay elements in Verilog HDL using DELAYADJUST. In addition, we set the DPs to "Post-Fit" except delay elements to preserve the placement and routing in the second synthesis. Re-synthesis, timing verification, and delay adjustment are repeatedly performed until all timing constraints are satisfied.

## V. Experiments

We designed three asynchronous circuits, DIFFerential EQuation solver (DIFFEQ), Ellipse Wave Filter (EWF), and Inverse Discrete Cosine Transform (IDCT) using the proposed method. Then, we evaluated the number of delay adjustments and the quality of the designed circuits.

In this experiment, we used the Intel Cyclone IV FPGA (EP4CE115F29C7). We also used the Intel Quartus Prime Standard Edition 18.1 for synthesis and static timing analysis, and the ModelSim Intel Starter Edition 10.5b for simulation. We implemented the generation of DPs, the generation of LLs (PLACEGEN), and the design support tool set in [5] using Python3.

As a reference, we initially designed the synchronous circuits (*Sync*) with the shortest clock cycle. The clock cycle times of DIFFEQ, EWF, and IDCT were 11 ns, 13 ns, and 13 ns. We generated the maximum delay constraints for the paths and the local clock constraints for $ack_i$ in the BD circuits based on the latency constraint which was the product of the clock cycle time and the number of states in the synchronous circuits.

To clarify the effect of DPs and LLs, we designed four BD circuits for DIFFEQ, EWF, and IDCT. The circuit names with their conditions are described in Table I. $dp\_ll_{t,ca,r}$ means that LLs with a region are generated for each top-level module, $ctrl_i$, and register. The coordinate of the region was decided by Quartus Prime automatically. $dp\_ll_{t,cf,r}$, on the other hand, means that the coordinate of the top-level module and $ctrl_i$ was decided by the procedure described in section IV.A.2. All other conditions were the same as $dp\_ll_{t,ca,r}$.

Table II shows the number of delay adjustments. "n/a" means that the BD circuits without timing violations were not obtained although we adjusted 10 times. Using DPs, we could obtain the BD circuits which satisfy all timing constraints within 6 delay adjustments. This means that the use of DPs was useful to reduce the number of delay adjustments because DPs preserve the logic, placement, and/or routing of the previous synthesis.

Figure 11 to Figure 14 show the circuit area, execution time, dynamic power consumption, and energy consumption in the designed circuits. The circuit area shows the number of LEs reported by Quartus Prime. The execution time was obtained by simulating the designed circuits with an arbitrary test sequence using ModelSim. The dynamic power consumption was obtained by PowerPlay Power Analyzer by assigning a Value Change Dump (VCD) file generated by the simulation. The energy consumption was the product of the execution time and dynamic power consumption.

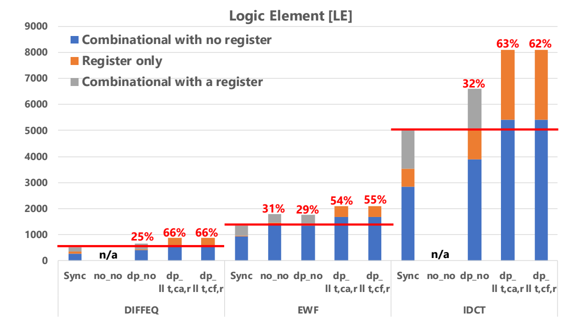


Fig. 11. Circuit area.

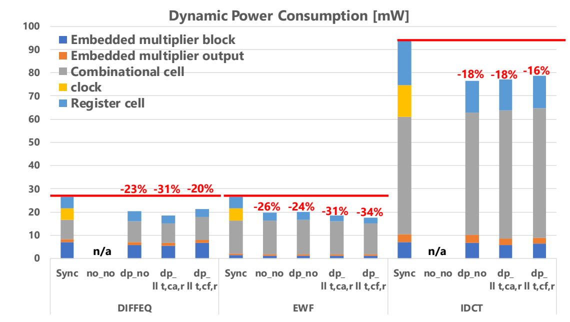


Fig. 13. Dynamic power consumption.

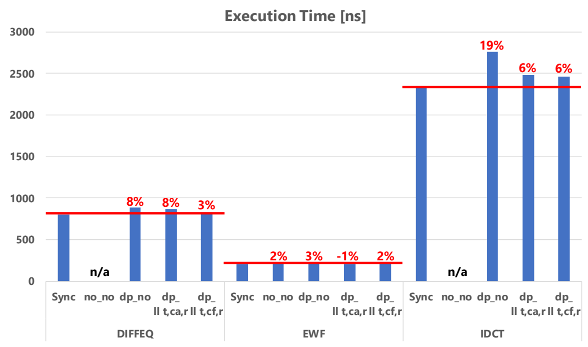


Fig. 12. Execution time.

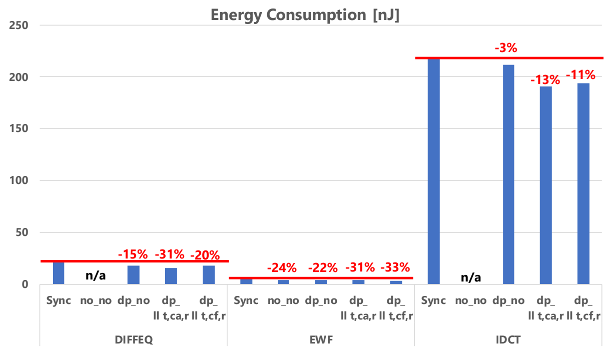


Fig. 14. Energy consumption.

The experimental result showed the efficiency of the use of LLs. On average, for the synchronous circuits (i.e., baseline), the energy consumption was reduced 25.0% in the cases of $dp\_ll_{t,ca,r}$ and 21.3% in the cases of $dp\_ll_{t,cf,r}$. $dp\_ll_{t,ca,r}$ was the best for the dynamic power consumption while $dp\_ll_{t,cf,r}$ was the closest performance to the synchronous circuits. In fact, fixing the coordinate of the regions for $ctrl_i$ could reduce the wire delay between $ack_i$ and registers and the time for the idle phase. On the other hand, the use of DPs and LLs required more LEs because logic optimization such as resource sharing was restricted.

## VI. Conclusion

In this paper, we proposed a design method to implement asynchronous circuits on commercial FPGAs using placement constraints. The use of the proposed method resulted in the reduction of the number of delay adjustments and the better circuit quality in terms of energy consumption and performance.

In our future work, we are going to consider a more effective floorplan method for the control circuit to improve performance. In addition, we extend the proposed method to deal with pipelined asynchronous circuits.

## Acknowledgements

This work is partially supported by Grant-in-Aid for Scientific Research from Japan Society for the promotion of science (#18K11221).

## References

[1] Q. T. Ho et al., "Implementing Asynchronous Circuits on LUT Based on FPGAs", *Proc. FDL*, pp. 36-46, 2002.

[2] M. Tranchero, and L. M. Reyneri, "Exploiting synchronous placement for asynchronous circuits onto commercial FPGAs", *Proc. Field Programmable Logic and Applications*, pp.622-625, 2009.

[3] R. Mocho et al., "Asynchronous Circuit Design on Reconfigurable Devices", *Proc. SBCCI*, pp. 20-25, 2006.

[4] L. Reyneri, and M. Tranchero, "Implementation of Self-Timed Circuits onto FPGAs Using Commercial Tools", *Proc. EUROMICRO*, pp.373-380, 2008.

[5] K. Takizawa et al., "A Design Support Tool Set for Asynchronous Circuits with Bundled-data Implementation on FPGAs", *Proc. FPL*, pp.232-235, 2014.

[6] F. U. Rosenberger et al., "Q-modules: internally clocked delay-insensitive modules", *IEEE Transactions on Computers*, vol.37, no.9, pp.1005-1018, 1988.

[7] Intel, "Quartus Prime Standard Edition Handbook Volume1: Design and Synthesis", 2017.