A Study on the Optimization of Asynchronous Circuits During RTL Conversion from Synchronous Circuits

Shogo Semba
The University of Aizu, Japan
d8211108@u-aizu.ac.jp

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

Abstract— In this paper, we propose three optimization methods for asynchronous circuits during the Register Transfer Level (RTL) conversion from synchronous RTL models. The modularization of data-path resources and the restriction of the use of D flip-flops reduce the circuit area while fixing the control signal of the multiplexers reduces the dynamic power consumption. In the experiment, we evaluated the effect of the three optimization methods. The combination of the three optimization methods could reduce the energy consumption 24.6% in the case of a differential equation solver and 12.6% in the case of a tiny encryption algorithm compared to the ones without the proposed optimization methods.

I. Introduction

Asynchronous circuits are low power consumption and low electromagnetic interference compared to synchronous circuits. In asynchronous circuits, circuit components are controlled by local handshake signals instead of global clock signals.

However, the design of asynchronous circuits is more difficult than the design of synchronous circuits. According to the selection of a delay model, handshake protocol, and data encoding scheme, the design method and design constraints are different. In addition, hazard-free implementation is required. Nevertheless, available Electronic Design Automation (EDA) tools to support the design of asynchronous circuits are insufficient.

To make asynchronous circuit designs easy, design methods which convert synchronous Gate-level (GL) netlists to asynchronous GL netlists (i.e., GL conversion) were proposed in[1, 2, 3, 4, 5, 6, 7, 8]. In these methods, registers by D flip-flops (DFFs) are converted into latches and the latches are controlled by latch controllers based on local handshake signals. Indeed, the GL conversion methods make asynchronous circuit designs easier, because commercial EDA tools are used as much as possible. However, GL conversion methods cannot evaluate the quality of asynchronous circuits in early design stages, cannot utilize logic optimization for asynchronous circuits, and do not fit to the design of asynchronous circuits on commercial Field Programmable Design Arrays (FPGAs). In FPGA designs, the standard design entry is a Register Transfer Level (RTL) model.

Compared to the GL conversion methods, [9] addressed an RTL conversion method for asynchronous circuits to solve the problems of the GL conversion methods. In addition, [9] showed that the RTL conversion method can generate better asynchronous circuits compared to the GL conversion methods in terms of energy consumption. On the other hand, the RTL conversion is more difficult than the GL conversion, because there are various representation styles for RTL models. Although [9] used an intermediate representation for synchronous RTL models to deal with various representation styles, the quality of asynchronous circuits may depend on the representation style of the synchronous RTL models.

In this paper, we propose three optimization methods for the RTL conversion of asynchronous circuits from synchronous RTL models. One is the modularization of data-path resources to optimize the circuit area. Second is the use of DFFs without an enable signal to reduce the circuit area. Third is fixing the control signals of multiplexers by the insertion of latches to avoid unnecessary operations during the reset of the control signal to reduce the dynamic power consumption.

The rest of the paper is organized as follows. Section II describes asynchronous circuits with bundled-data implementation. Section III describes the overview of the RTL conversion method in [9]. Section IV describes three optimization methods. Section V describes the experimental results. Finally, section VI describes the conclusion and future work.

II. Asynchronous Circuits with Bundled-data Implementation

Bundled-data (BD) implementation is one of the data encoding schemes in asynchronous circuits. In the BD implementation, N-bit data are represented by $N+2$ signals. Additional two signals correspond to local handshake signals; the request signal *req* and the acknowledge signal *ack*. The operation timing is guaranteed by a delay element on *req* signals. Therefore, the performance of the BD implementation depends on the delay of the control circuit including delay elements.

Figure 1(a) represents the circuit model of the BD implementation used in this work. This model consists of a data-path circuit and a control circuit.

The data-path circuit is almost the same as the one used

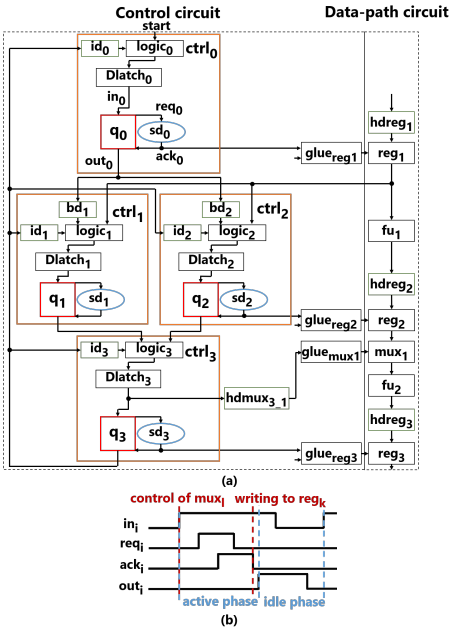

Fig. 1. Circuit model used in this work: (a) structure and (b) timing diagram of $ctrl_i$.

in synchronous circuits. The data-path circuit consists of functional units fu_h, multiplexers mux_l, and registers reg_k. If there are hold violations on the registers, a delay element $hdreg_k$ is inserted to the input of the registers to satisfy the hold constraints.

The control circuit consists of control modules $ctrl_i$ ($0 \le i \le n-1$), glue logics $glue_{reg_k}$ and $glue_{mux_l}$, and delay elements $hdmux_{i\_l}$. One $ctrl_i$ is used to control data-path resources used in a state i.$glue_{reg_k}$ and $glue_{mux_l}$ are used to generate the write signals for the registers and the control signals for the multiplexers. $hdmux_{i\_l}$ is used to guarantee the hold constraint for the registers caused by a transition of the control signals for the multiplexers.

$ctrl_i$ consists of a Q-module q_i [11], a D latch $Dlatch_i$, a glue logic $logic_i$, and delay elements sd_i, bd_i, and id_i. $logic_i$ generates a rising transition to trigger $ctrl_i$ when required input signals arrive at $ctrl_i$. The delay element sd_i is used to guarantee the setup constraints of the registers controlled by $ctrl_i$. The delay element bd_i is used to guarantee the correct timing of a control branch to enter $ctrl_i$. The delay element id_i is used to guarantee the initialization of $ctrl_i$.

Figure 1(b) represents the timing diagram of $ctrl_i$ where a rising transition of signals is represented by $signal+$ while a falling transition is represented by $signal-$. $ctrl_i$ starts the control of data-path resources in the state i when $out_{i-1}+$ generated from the previous control module $ctrl_{i-1}$ arrives at $ctrl_i$. The signal transition is changed to in_i+ through $logic_i$ and $Dlatch_i$. The rising transition in_i+ is used to control mux_l and propagated to q_i. q_i generates the request signal req_i+. req_i+ is changed to ack_i+ through sd_i. ack_i+ is returned to q_i. Then, q_i generates req_i- which is changed to ack_i- through sd_i. The falling transition ack_i- is used to write data into reg_k. Finally, q_i generates out_i+ to pass the control to the next control module $ctrl_{i+1}$. After out_i+, $ctrl_i$ waits in_i- and generates out_i- to initialize $ctrl_i$.

In the timing diagram, we call the time from in_i+ to out_i+ as active phase and the time from out_i- to the next in_i+ as idle phase. In the active phase, operations are performed in the data-path circuit while in the idle phase, no operations are performed. The control signals are initialized in the idle phase.

III. RTL Conversion

Our optimization methods for asynchronous circuits are performed during RTL conversion using [9]. In this section, we briefly describe the RTL conversion method described in [9].

The RTL conversion method consists of two parts. The first part called *Sync2XML* generates an eXtensible Markup Language (XML) file from a given synchronous RTL model. The XML file is called Model-XML. The second part called *XML2Async* generates the RTL model of a BD implementation. According to [9], the XML file is used to convert synchronous RTL models with various representation styles in Verilog Hardware Description Languages (HDLs). Therefore, the XML file has a format to represent resources, both control and data-paths, and control timing of data-path resources in a given synchronous RTL model.

In *Sync2XML*, resource information, path information, and timing information are extracted from a given synchronous RTL model through a parser. Then, the information is represented in the Model-XML. For example, Fig.2(a) represents the RTL model of a synchronous circuit and Fig.2(b) represents the Model-XML of the synchronous circuit.

In *XML2Async*, data-path resources and $ctrl_i$ are mapped based on the resource information, the connections among resources are established based on the path information, and the glue logics for registers and multiplexers are generated based on the timing information. For example, Fig.2(c) represents the RTL model of a BD implementation from the Model-XML in Fig.2(b).

IV. Proposed Method

In this paper, we propose three optimization methods during RTL conversion from the RTL models of synchronous circuits to the RTL models of BD implementation. The first optimization method is an area optimization of combinational circuits by modularizing data-path resources. The second optimization method is also an area optimization method by restricting to use DFFs without enable signals. The last optimization method is a dynamic power optimization method of combinational circuits by

Fig. 2. Example of RTL conversions: (a) RTL model of a synchronous circuit, (b) Model-XML, and (c) RTL model of a BD implementation.

fixing control signals for multiplexers. We describe each method in the followings.

A. *Modularization for Data-path Resources*

The purpose of modularizing data-path resources is to assign the modularized resources to "through points" of the maximum delay constraints in order to optimize circuit area. In BD implementations, we are required to

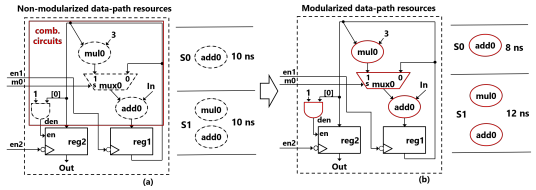

Fig. 3. Modilarization of data-path resources: (a) without modularization and (b) with modularization.

assign the maximum delay constraints for paths to satisfy a given latency or cycle time constraint.

By assigning "through points", we can reduce the area of data-path resources. Let us explain this using an example shown in Fig.3. Fig.3(a) describes a converted RTL model without modularization and a data flow graph for the RTL model. Each state in the graph has the same delay. In the state $s0$ and $s1$, we have the data-paths whose source and destination registers are equivalent but used functional units are different ($add0$ in $s0$ and $add0$, $mul0$ in $s1$). This may result in the difference of the path delays, e.g., 8 ns for the path in $s0$ and 10 ns for the path in $s1$. Under a given latency constraint, we can change the delay of $s1$ to 12 ns (2 ns for $s0$ is borrowed in $s1$). This is possible for BD implementations, because each state can have the different delay value due to the use of the local handshake signals. As a result, we may obtain a data-path resource with smaller size if a loose value can be assigned to the maximum delay constraint.

To realize the area optimization, we need to assign "through point" for the maximum delay constraint. In the above example, we assign the multiplier $mul0$ and the adder $add0$ as "through points". Fig.3(b) describes a converted RTL model with modularization and a data flow graph for the RTL model.

Modularization can be performed easily during RTL conversion. This is because the Model-XML has the resource information and the path information for the RTL model of a given synchronous circuit.

B. *Restricting the Use of DFFs*

The purpose to restrict the use of DFFs is to optimize circuit area for registers. In BD implementations, registers are triggered by the ack_i signals from $ctrl_i$. If no need to write data into registers in state i, there is no connection from ack_i to the registers. This allows registers to use DFFs without an enable signal. Since in general a DFF without an enable signal is smaller size than a DFF with an enable signal, it results in the reduction of circuit area.

On the other hand, enable signals may be generated by data-path resources. This is particularly remarkable in the RTL models of synchronous circuits which are synthesized by using a high-level synthesis tool with a clock gating option. In such a case, we cannot remove the enable signals.

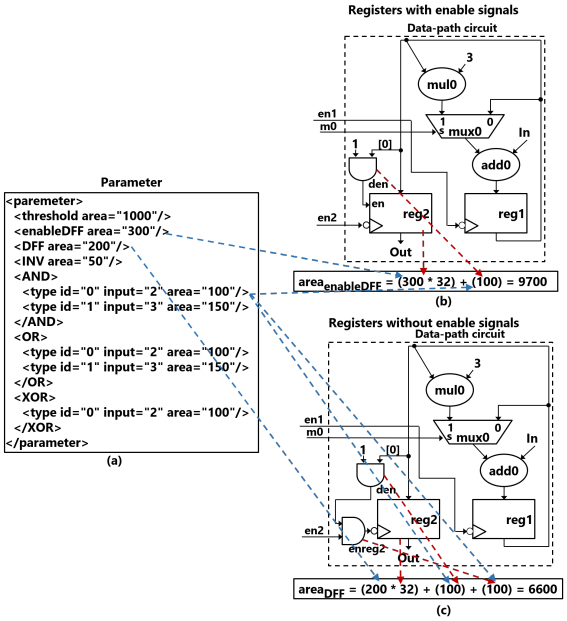

Fig. 4. Area estimation for $reg2$: (a) parameters, (b) registers with enable signals, and (c) registers without enable signals

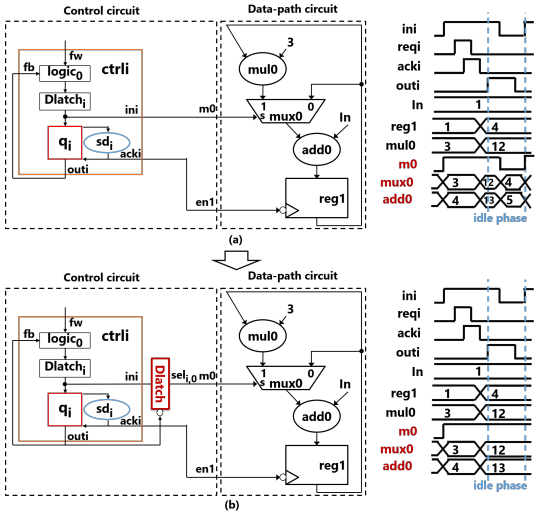

Fig. 5. Fixing the control signal for multiplexers: (a) RTL model and the waveform during the idle phase before fixing and (b) RTL structure model and the waveform during the idle phase after fixing.

Instead, we restrict the use of DFFs by moving the assignments for the enable signals to outside of the register representations when the following inequality is satisfied.

$$area_{enableDFF} - area_{DFF} > threshold \qquad (1)$$

where $area_{enableDFF}$, $area_{DFF}$, and $threshold$ represent the circuit area when DFFs with an enable signal are used, the circuit area when DFFs without an enable signal are used, and a threshold value. Note that the combinational logics to generate the enable signals are also included in $area_{enableDFF}$ and $area_{DFF}$. Since we do not know the exact area for the DFFs and the combinational logics, we estimate $area_{enableDFF}$ and $area_{DFF}$ from the reference of the used technology library and the number of literals and operations in the assignments for the enable signals.

This optimization is also performed during RTL conversion easily, because the information for registers is represented in the resource information in the Model-XML. The information includes whether enable signal is used or not, the bit-width of the registers, and the number of operations and literals in the assignments of the enable signals. In addition, we need to prepare a parameter XML file which represents the circuit area of basic logic operations, the circuit area of DFFs, and the threshold value as shown in Fig.4(a).

Figure 4 shows an example of this optimization. From the parameter in Fig.4(a) and the calculations of $area_{enableDFF}$ in Fig.4(b) and $area_{DFF}$ in Fig.4(c), the DFFs without an enable signal is used because the difference is more than the threshold value.

C. *Fixing Control Signals for Multiplexers*

Fixing the control signals for multiplexers reduces the dynamic power consumption of the data-path resources. It is motivated from the unnecessary operations of the data-path resourced during the idle phase of the control circuit.

Figure 5(a) represents the circuit model and the waveform during the idle phase. In the idle phase of the control circuit, the data-path resource $add0$ performs an addition due to a change of the control signal for the multiplexer $mux0$. During the idle phase of the control signal, in_i and out_i of all $ctrl_i$ are changed to zero as the reset of the control signals before the next control. A signal transition of in_i from one to zero is propagated to $mux0$ in which results in the unnecessary operation of $add0$.

To prevent the signal transition of in_i to $mux0$, we insert a D latch between $Dlatch_i$ and $mux0$ as shown in Fig.5(b). Since the signal transition is not propagated to $mux0$ by the inserted D latch, we can avoid the unnecessary operation of $add0$. This results in the reduction of the dynamic power consumption of $add0$.

During RTL conversion, this optimization can be performed easily by referring to the timing information of the Model-XML where the control of multiplexers and registers is described for each state (see Fig.2(b)). We insert D latches to the control signals of multiplexers when the values of the control signals in the initial state and the last state (before the idle phase) are different.

Fixing the control signal for multiplexers mul_l by the insertion of D latches requires an additional timing constraint. The inserted D latches must be opened after in_i+ arrives the inserted D latches. If the timing constraint is violated, the control signal is changed during the idle phase. Figure 6(a), (b), and (c) represent an example

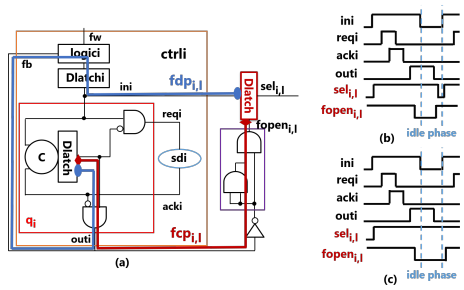

Fig. 6. Timing constraint for fixing the control signals for multiplexers: (a) the paths related to the additional timing constraint, (b) the waveform when the timing constraint is violated, and (c) the waveform when the timing constraint is satisfied.

of the timing constraint, a waveform where the timing constraint is violated, and a waveform where the timing constraint is satisfied. We formulate the timing constraint as follows. $t_{minfcp_{i,l}}$ represents the minimum delay of the path $fcp_{i,l}$ (the red line in Fig.6(a)) from out_i to the clock pin of the inserted D latch. $t_{maxfdp_{i,l}}$ represents the maximum delay of the path $fdp_{i,l}$ (the blue line in Fig.6(b)) from out_i to the data pin of the inserted D latch. The timing constraint is represented by the following inequality

$$t_{minfcp_{i,l}} > t_{maxfdp_{i,l}} \tag{2}$$

If this inequality in violated, we need to insert a delay element $fd_{i,l}$ by AND gates to satisfy this inequality.

V. Experimental Results

To evaluate the proposed optimization methods, we converted three synchronous RTL models of synchronous circuits, DIFFerential EQuation solver (DIFFEQ), Elliptic Wave Filter (EWF), and Tiny Encryption Algorithm (TEA), to the RTL models of the BD implementations using the proposed method. Then, we evaluated the designed circuits after logic synthesis in term of circuit area, execution time, dynamic power consumption, and energy consumption. We extended the RTL conversion tool in [9] to implement the proposed optimization methods.

Initially, we prepared the RTL models of the synchronous circuits by synthesizing the SystemC models of DIFFEQ, EWF, and TEA using Cadence Stratus 18.1. The used technology library was an eShuttle 65 nm process technology. We explored the synchronous circuits with the shortest clock cycle time. The clock cycle times of DIFFEQ, EWF, and TEA were $1,600$ ps in the all cases.

To evaluate each optimization method, we prepared five RTL models of the BD implementations.

- *async* - no optimization
- *async_m* - with modularization of data-path resources

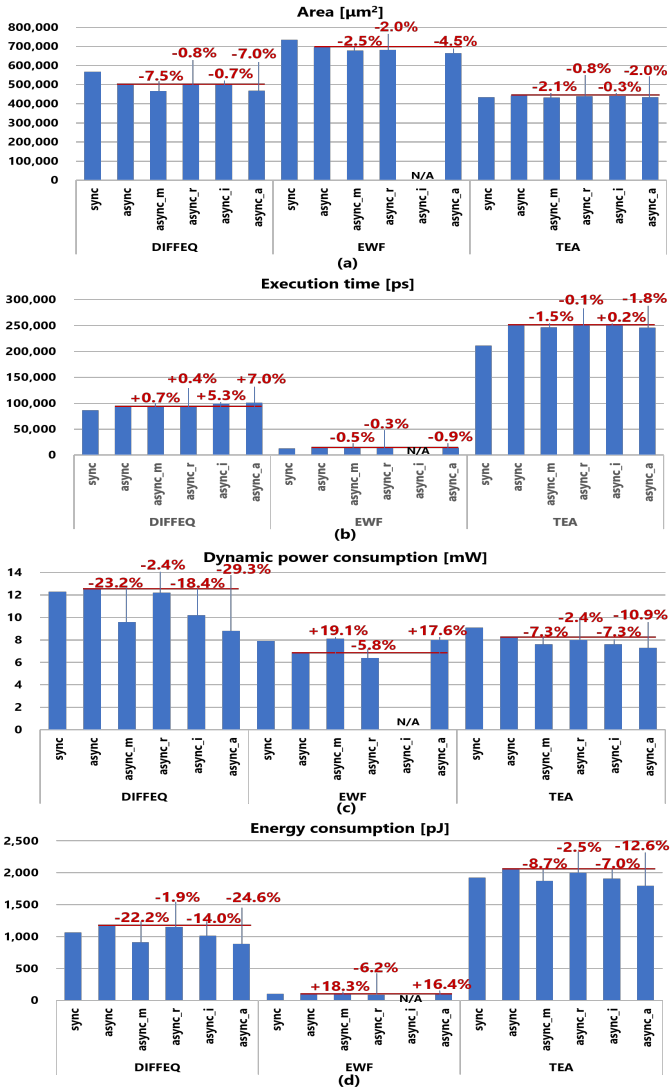

Fig. 7. Evaluation results: (a) circuit area, (b) execution time, (c) dynamic power consumption, and (d) energy consumption.

- *async_r* - with restricting the use of DFFs
- *async_i* - with fixing control signals for multiplexers
- *async_a* - with all optimization methods

Following to the design method in [10], we designed the BD implementations from the RTL models. To check the effect of the proposed optimization methods more precisely, we used the same maximum delay constraints and the local clock constraints for all designs. For the design of the BD implementations, we used Cadence Genus 18.1 for logic synthesis, Synopsys PrimeTime M-2017.06-SP3-1 for static timing analysis (STA) and power analysis, and Synopsys VCS M-2017.03-SP2 for logic simulation. The technology library was the eShuttle 65 nm.

Figure 7 (a) to (d) represents the circuit area, the execution time, the dynamic power consumption, and the energy consumption of the designed BD implementations

after logic synthesis. The circuit area was obtained from the report file generated by Genus. The execution time was obtained by simulating the designed circuits with arbitrary test patterns using VCS. The dynamic power consumption was obtained by PrimeTime with the switching activity interchange format file generated by VCS. The energy consumption was the product of the execution time and the dynamic power consumption. Note that $async_i$ of EWF is represented by N/A, because no latch is inserted to any control signal for multiplexers. Since the initial state and the last state have the same signal value, no unnecessary operations in the data-path resources exist in the original BD implementation (i.e., *async*) during the idle phase. Note that hereafter, we consider *async* as the baseline.

All of the proposed optimization methods could reduce the circuit area. Especially, the average reduction ratio of $async_m$ was the largest (4.0%). This is because a loose value was assigned to the maximum delay constraints for some of the data-paths with a through point.

For the performance, $async_i$ resulted in the performance degradation a little because the insertion of D latches slightly changed the critical path delay. $async_m$ and $async_r$ did not have the significant impact for the performance.

All of the proposed optimization methods also reduced the dynamic power consumption except $async_m$ of EWF. Especially, the average reduction ratio of $async_i$ was the largest (12.9%). On the other hand, the dynamic power consumption of $async_m$ of EWF was increased. The reason came from the increase of the number of toggles in the data-path resources. Because the toggles largely depend on circuit structure and delay, we think that it is not the direct effect of the proposed optimization methods during RTL conversion.

Similarly, all of the proposed optimization methods reduced the energy consumption except $async_m$ of EWF. This is because the energy consumption depends on the execution time and the dynamic power consumption.

The experimental result showed that all of the proposed optimization methods contribute to the reduction of the circuit area and the dynamic power consumption. By the combination of them (i.e., $async_a$), we could reduce the energy consumption 24.6% in the case of DIFFEQ and 12.6% in the case of TEA.

VI. Conclusion

In this paper, we proposed three optimization methods for the RTL conversion of asynchronous circuits from synchronous RTL models. We confirmed the effect of each optimization method in the experiment. In addition, the combination of the three optimization methods largely reduced the energy consumption of DIFFEQ and TEA compared to the ones without the optimization methods.

In our future work, we consider a dynamic power optimization method during RTL conversion in which latches are inserted to the operands of functional units to reduce the toggles of the functional units.

Acknowledgements

The authors would like to thank Mr. Tatsuoka, Mr. Fujimura, and Mr. Nakae in Socionext Inc. who gave useful advice in this work. This work is partially supported by Grant-in-Aid for Scientific Research from Japan Society for the promotion of science (#18K11221).

References

[1] J. Cortadella et al., "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications", IEEE TCAD, vol. 25, pp. 1904–1921, 2006.

[2] N. Andrikos et al., "A Fully-Automated Desynchronization Flow for Synchronous Circuits", Proc. DAC, pp. 982–985, 2007.

[3] P. A. Beerel et al., "Proteus: An ASIC Flow for GHz Asynchronous Designs", IEEE Design & Test of Computers, vol. 28, pp. 36–51, 2011.

[4] A. Kondratyev and K. Lwin, "Design of Asynchronous Circuits by Synchronous CAD Tools", Proc. DAC, pp.411–414, 2002.

[5] R. Zhou et al., "Quasi-Delay-Insensitive Compiler: Automatic Synthesis of Asynchronous Circuits from Verilog Specifications", Proc. NWSCAS, pp. 1–4, 2011.

[6] A. Branover, et al., "Asynchronous Design By Conversion: Converting Synchronous Circuits into Asynchronous Ones", Proc. Design, Automation and Test in Europe Conference and Exhibition, vol. 2, pp. 870–875, 2004.

[7] J. Oberg et al., "Automatic Synthesis of Asynchronous Circuits from Synchronous RTL Description", Proc. NORCHIP, pp. 200–205, 2005.

[8] Y. Zhang et al., "Challenges in Building An Open-source Flow from RTL to Bundled-Data Design", Proc. ASYNC, pp. 26–27, 2018

[9] S. Semba and H. Saito, "Conversion from Synchronous RTL Models to Asynchronous RTL Models", IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, vol. E102-A, No. 7, pp. 904–913, 2019.

[10] S. Semba and H. Saito, "Comparison of RTL Conversion and GL Conversion from Synchronous Circuits to Asynchronous Circuits", Proc. ISCAS, pp. 1–4, 2019.

[11] E. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T. P. Fang, "Q-Modules: Internally Clocked Delay Insensitive Modules", IEEE TC, vol. C-37, no. 9, pp. 1005–1018, 1998.