A Study on the Design of Interface Circuits Between Synchronous-Asynchronous Modules Using Click Elements

Shogo Semba
The University of Aizu
shogo-s@u-aizu.ac.jp

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

Abstract— In this paper, we propose interface circuits between synchronous-asynchronous modules using Click Elements. Click Elements are used to control the asynchronous parts in the proposed interface circuits. In the experiment, compared with the interface circuit based on the two-flop synchronizer, the proposed interface circuits could reduce the latency and handshake overhead by up to 4.9 cycles and 17.0 cycles.

I. Introduction

Recently, most of the digital systems are based on the concept of System-on-a-Chip (SoC). SoCs are composed of microprocessors, memories, specific circuits, and so on. These circuits are often controlled by different clock signals. Therefore, synchronizers are required to reduce the metastability problem.

To solve the metastability problem, Globally Asynchronous Locally Synchronous (GALS) was proposed in [1]. GALS systems are composed of several local synchronous modules. Each module is controlled by an independent clock signal and communicated with other modules asynchronously. To satisfy the asynchronous communication, interface circuits are required between different synchronous modules. As the interface circuits for different synchronous modules, a two-flop synchronizer [2], an interface circuit using a pausible clock [3], an interface circuit using a clock gating [4], and a FIFO [5] were proposed.

Asynchronous circuits may be used to reduce the power consumption of synchronous modules. Asynchronous circuits are potentially low power consumption because circuit components are controlled by local handshake signals instead of global clock signals. Similar to the communication of different synchronous modules, interface circuits are required between synchronous-asynchronous modules when asynchronous circuits are communicated with synchronous circuits.

To transfer data between synchronous-asynchronous modules, various interface circuits were proposed in [6, 7, 8, 9, 10, 11, 12]. In [6, 7, 8, 9, 10], FIFOs were used to transfer data between synchronous-asynchronous modules to achieve high throughput. Usually, control circuits of the FIFOs are complex because the decision of memory addresses and the generation of read/write signals are required. In [11], communication registers based on handshake signals were used to support the coherent communication of multi-bit data between synchronous-asynchronous modules. However, the timing for writing data to the registers is not guaranteed because there is only a read/write clock signal as an input signal from synchronous modules. In [12], an interface circuit based on the two-flop synchronizer was used to transfer data between synchronous-asynchronous modules. However, data could not be transferred through only the interface circuit because protocol converters are required when start and end signals are used for asynchronous controllers.

In this paper, we propose interface circuits between synchronous-asynchronous modules using Click Elements [13]. The proposed interface circuits are based on handshake protocols. The proposed interface circuits use a four-phase handshake protocol to control the synchronous part. In contrast, the proposed interface circuits use a two-phase handshake protocol to control the asynchronous part. The control of the synchronous parts is based on a two-flop synchronizer [2] while the control of the asynchronous parts is based on a Click Element.

The rest of this paper is organized as follows. Section II describes Click Elements. Section III describes the proposed interface circuits. Section IV describes the experimental results. Finally, section V describes the conclusion and future work.

II. Click Elements

Click Element [13] is one of the control templates used in the design of asynchronous circuits with bundled-data implementation. In bundled-data implementation, N bit signals are represented by $N + 2$ signals. Additional two signals correspond to local handshake signals; the request signal *req* and the acknowledgment signal *ack*. The timing for writing data to registers is guaranteed by delay elements on *req*.

Click Elements are implemented as a data-driven two-phase handshake protocol. In the two-phase handshake protocol, only two signal transitions (rising transitions of *req* and *ack* or falling transitions of *req* and *ack*) are used to transfer data. In contrast, in the four-phase handshake protocol, four signal transitions (rising transition of *req*, rising transition of *ack*, falling transition of *req*, and falling transition of *ack*) are used to transfer data.

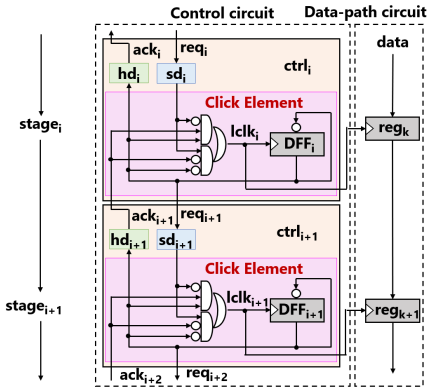

Fig. 1. Asynchronous circuits with bundled-data implementation using Click Elements.

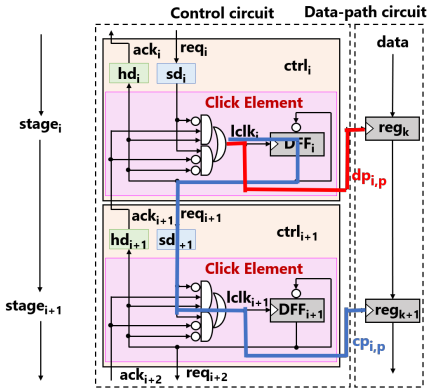

Fig. 2. Paths related to a local cycle time.

The two-phase handshake protocol is generally low latency compared with the four-phase handshake protocol. In addition, Click Elements can transfer control and data at the same time.

Figure 1 represents the circuit model of asynchronous circuits with bundled-data implementation using Click Elements. This circuit model consists of a data-path circuit and a control circuit. The data-path circuit is almost the same as the one used in synchronous circuits. The control circuit consists of control modules $ctrl_i$ $(0{\leq}i{\leq}n-1)$ assigned for each pipeline stage $stage_i$.

$ctrl_i$ consists of a Click Element, a delay element sd_i to satisfy the setup constraint of registers, and a delay element hd_i to satisfy the hold constraint of registers. The Click Element consists of a D Flip-Flop (DFF) DFF_i and a logic to generate a local clock signal $lclk_i$.

$ctrl_i$ starts its operation when a rising or falling transition of req_i arrives at $ctrl_i$. The rising transition of req_i generates a rising transition of $lclk_i$ through sd_i. Then, $lclk_i$ controls DFF_i in $ctrl_i$ and registers reg_k in the data-path circuit. Finally, DFF_i generates a rising transition of ack_i to pass the control to $ctrl_{i+1}$. In addition, the rising transition of ack_i arrives at $ctrl_{i-1}$ through hd_i to acknowledge that the operation of $ctrl_i$ is completed. Note that the behavior of $ctrl_i$ for the falling transition of req_i is the same as the behavior of $ctrl_i$ for the rising transition of req_i.

Global Cycle Time. In this paper, we define a local cycle time (lct) and a global cycle time (gct) to evaluate the performance of asynchronous circuits. lct_i is the maximum delay of control-paths for $stage_i$ while gct is the maximum value of lct_i.

Figure 2 represents paths related to lct_i. lct_i and gct can be obtained by the following equations.

$$lct_i = max(t_{maxcp_{i,0}} - t_{maxdp_{i,0}}, \cdots, t_{maxcp_{i,m-1}} - t_{maxdp_{i,m-1}}) \quad (1)$$

$$gct = max(lst_0, \cdots, lst_{n-1}) \quad (2)$$

p $(0{\leq}p{\leq}m-1)$ represents the identifier of paths. $t_{maxcp_{i,p}}$ represents the maximum delay of a path $cp_{i,p}$ from $lclk_{i-1}$ to the destination register. $t_{maxdp_{i,p}}$ represents the maximum delay of a path $dp_{i,p}$ from $lclk_{i-1}$ to the source register. lct_i is the largest value of $t_{maxcp_{i,p}}$ minus $t_{maxdp_{i,p}}$. gct is the maximum value of lct_i.

III. Proposed Interface Circuits

In this paper, we propose interface circuits between synchronous-asynchronous modules using Click Elements. We assume that synchronous modules are controlled by clock signals while asynchronous modules are controlled by local handshake signals.

We propose two interface circuits based on handshake protocols. One is the *StoA* circuit to transfer data from the synchronous module to the asynchronous module. Another is the *AtoS* circuit to transfer data from the asynchronous module to the synchronous module.

StoA and *AtoS* circuits consist of a synchronous part and an asynchronous part. *StoA* and *AtoS* circuits use the two-flop synchronizer to control the synchronous part because the two-flop synchronizer can reduce the metastability problem with a simple circuit structure when data are transferred between different timing modules. In contrast, *StoA* and *AtoS* circuits use the Click Element to control the asynchronous part because the Click Element is also a simple structure that can transfer control and data at the same time.

Figure 3 represents the overview of *StoA* and *AtoS* circuits. *LS* and *LA* represent a local synchronous module and a local asynchronous module, respectively. For *LS* (*LA*), *Sreq* (*Areq*) and *Sack* (*Aack*) represent a request signal and an acknowledgment signal, respectively. *StoA* and *AtoS* circuits transfer data between different timing modules using these signals.

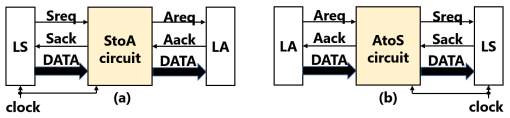

Fig. 3. Overview of the proposed interface circuits: (a) *StoA* circuit to transfer data from a synchronous module to an asynchronous module and (b) *AtoS* circuit to transfer data from an asynchronous module to a synchronous module.

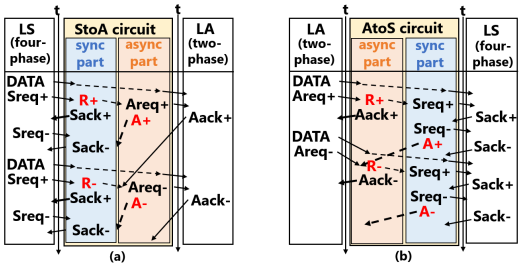

Fig. 4. Handshake protocols: (a) *StoA* circuit and (b) *AtoS* circuit.

A. Handshake Protocols of the Proposed Interface Circuits

StoA and *AtoS* circuits use the four-phase handshake protocol to control the synchronous part. In contrast, *StoA* and *AtoS* circuits use the two-phase handshake protocol to control the asynchronous part. From here, we represent a rising transition of a signal as *signal*+ and a falling transition of a signal as *signal*−.

StoA and *AtoS* circuits transfer data between synchronous-asynchronous modules like a pipeline circuit. However, to guarantee the timing for writing data to registers, the writing data to the registers must be waited until the acknowledgment signal is returned to the sender.

Figure 4(a) represents the handshake protocol for *StoA* circuit. First, *LS* sends *DATA* with *Sreq*+ to *StoA* circuit. Next, the synchronous part of *StoA* circuit sends *DATA* with a request signal *R*+ to the asynchronous part of *StoA* circuit. Then, the asynchronous part of *StoA* circuit sends *DATA* with *Areq*+ to *LA*. In addition, the asynchronous part of *StoA* circuit sends *A*+ to the synchronous part of *StoA* circuit. Finally, the synchronous part of *StoA* circuit sends *Sack*−. After receiving *Sack*−, *LS* can send the next *DATA* with *Sreq*+ to *StoA* circuit. Note that the behavior of *StoA* circuit for *R*− is the same as the behavior of *StoA* circuit for *R*+.

Figure 4(b) represents the handshake protocol for *AtoS* circuit. The handshake protocol is similar to the handshake protocol of *StoA* circuit. On the other hand, to guarantee the timing for writing data to registers, the synchronous part of *AtoS* circuit sends *A*+ to the asynchronous part of *AtoS* circuit after the synchronous part of *AtoS* circuit is received *Sack*+ from *LS*.

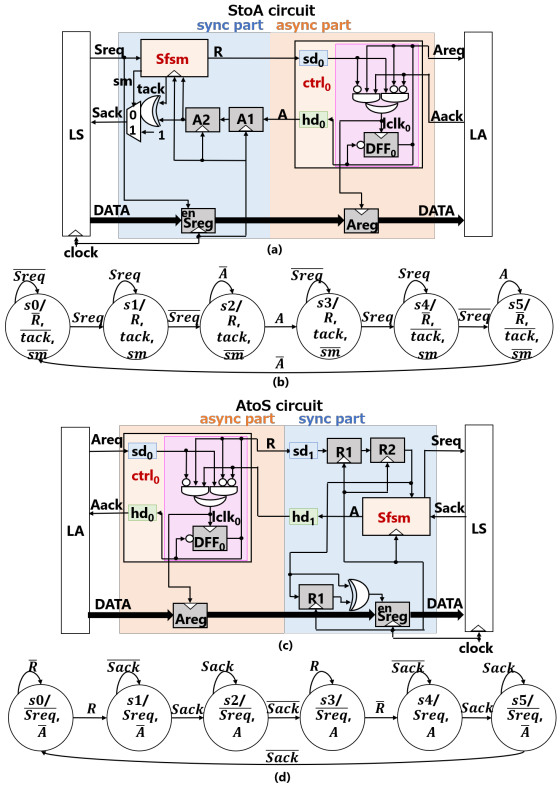

Fig. 5. Proposed interface circuits: (a) circut model of *StoA* circuit, (b) FSM of *StoA* circuit, (c) circut model of *AtoS* circuit, and (d) FSM of *AtoS* circuit.

B. Circuit Models

StoA and *AtoS* circuits use the two-flop synchronizer for the synchronous part. In contrast, *StoA* and *AtoS* circuits use the Click Element for the asynchronous part.

Figure 5(a) represents the circuit model of *StoA* circuit. The synchronous part of *StoA* circuit consists of a finite state machine (FSM) $Sfsm$, a register *Sreg*, a two-flop synchronizer ($A1$ and $A2$), an XOR gate, and a multiplexer. Figure 5(b) represents the state transition graph of $Sfsm$. $Sfsm$ generates $R+$ ($R-$) after receiving *Sreq*+ (*Sreq*−) from *LS*. We use the two-flop synchronizer to receive $A+$ ($A-$) which is an asynchronous input. In addition, we use the XOR gate and multiplexer to distinguish the generation of *Sack*+ from *Sreq*+ and the generation of *Sack*− from $A+$ ($A-$). On the other hand, the asynchronous part of *StoA* circuit consists of $ctrl_i$ based on the Click Element and a register *Areg*. In the asynchronous part of *StoA* circuit, any synchronizer is not required for the request signal *R* because the timing for writing data to *Areg* is guaranteed by sd_i.

Figure 5(c) represents the circuit model of *AtoS* circuit. The synchronous part of *AtoS* circuit is almost the same as the one used in *StoA* circuit. In the state transition graph of $Sfsm$ as shown in Fig. 5(d), $Sfsm$ generates $A+$ ($A-$) from *Sack*+ to guarantee the timing for writing

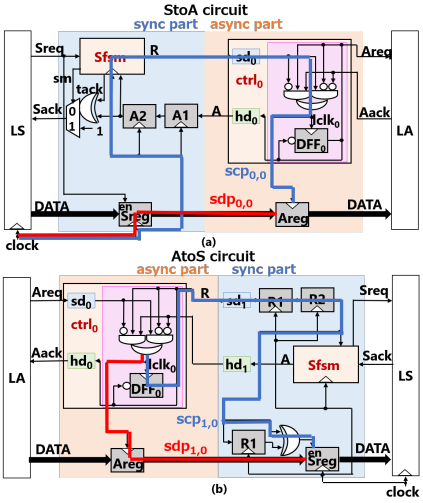

Fig. 6. Paths related to setup constraints for registers: (a) in the asynchronous part of *StoA* circuit and (b) in the synchronous part of *AtoS* circuit.

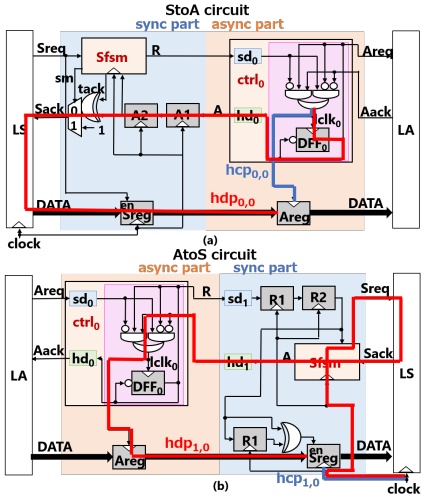

Fig. 7. Paths related to hold constraints for registers: (a) in the asynchronous part of *StoA* circuit and (b) in the synchronous part of *AtoS* circuit.

data to *Sreg*. In addition, we use a DFF and an XOR gate to generate an enable signal of *Sreg* because R is a two-phase handshake protocol. In contrast, the asynchronous part of *AtoS* circuit is the same as the one used in *StoA* circuit.

C. Timing Constraints

In *StoA* and *AtoS* circuits, it is necessary to satisfy setup and hold constraints for the internal registers of *StoA* and *AtoS* circuits to operate the circuits correctly. Note that we assume that the clock cycle time (CT) of the synchronous part and *gct* of the asynchronous part are fixed.

There are setup and hold constraints for each register in *StoA* and *AtoS* circuits. The setup and hold constraints for *Sreg* in the synchronous part of *StoA* circuit can be guaranteed by satisfying the traditional setup and hold constraints used in the design of synchronous circuits. Similarly, the setup and hold constraints for *Areg* in the asynchronous part of *AtoS* circuit is guaranteed by satisfying the setup and hold constraints used in the design of asynchronous circuits in [14]. In this paper, we define the setup and hold constraints for *Areg* (*Sreg*) in the asynchronous (synchronous) part of *StoA* (*AtoS*) circuit.

Setup Constraints. The input data for *Areg* (*Sreg*) in the asynchronous (synchronous) part of *StoA* (*AtoS*) circuit must be stable before the setup time to write the input data to the registers. This is called the setup constraint for *Areg* (*Sreg*).

Figure 6(a) represents paths related to the setup constraint for *Areg* in the asynchronous part of *StoA* circuit. $sdp_{i,p}$ ($sdp_{0,0}$) represents a data-path from the clock signal to the destination register *Areg* through the source register *Sreg*. In contrast, $scp_{i,p}$ ($scp_{0,0}$) represents a control-path from the clock signal to *Areg* through $ctrl_0$. We define the maximum delay of $sdp_{i,p}$ as $t_{maxsdp_{i,p}}$, the minimum delay of $scp_{i,p}$ as $t_{minscp_{i,p}}$, the margin for $t_{maxsdp_{i,p}}$ as $t_{sdpm_{i,p}}$, and the setup time of *Areg* as $t_{setup_{i,p}}$. The setup constraint can be represented by the following inequality.

$$t_{minscp_{i,p}} > t_{maxsdp_{i,p}} + t_{sdpm_{i,p}} + t_{setup_{i,p}} \quad (3)$$

If the setup constraint is violated, we must adjust the number of cells for sd_i (sd_0).

Figure 6(b) represents paths related to the setup constraint for *Sreg* in the synchronous part of *AtoS* circuit. This setup constraint is similar to the inequality (3). However, we consider the number of cycles (num) of the path from $R1$ to *Sreg* because the path delay from $R1$ to *Sreg* depends on CT. Therefore, the minimum delay of $scp_{i,p}$ is the minimum delay of $scp_{i,p}$ except for the path from $R1$ to *Sreg*. The setup constraint can be represented by the following inequality.

$$t_{minscp_{i,p}} + CT \times num > t_{maxsdp_{i,p}} + t_{sdpm_{i,p}} + t_{setup_{i,p}} \quad (4)$$

If the setup constraint is violated, we must adjust the number of cells for sd_i (sd_1).

Hold Constraints. The data must be stable for the hold time after the next input data are written to *Areg* (*Sreg*) in the asynchronous (synchronous) part of *StoA* (*AtoS*) circuit. This is called the hold constraint for *Areg* (*Sreg*).

Figure 7(a) represents paths related to the hold constraint for *Areg* in the asynchronous part of *StoA* circuit. $hdp_{i,p}$ ($hdp_{0,0}$) represents a data-path from $lclk_i$ to the destination register *Areg* through the source register *Sreg*. In $hdp_{i,p}$, we consider the number of cycles (num) of the path from $A1$ to *Sreg*. In cotrast, $hcp_{i,p}$ ($hcp_{0,0}$) represents a control-path from $lclk_i$ to *Areg*. We define

the minimum delay of $hdp_{i,p}$ except for the path delay from $A1$ to $Sreg$ as $t_{minhdp_{i,p}}$, the maximum delay of $hcp_{i,p}$ as $t_{maxhcp_{i,p}}$, the margin for $t_{maxhcp_{i,p}}$ as $t_{hcpm_{i,p}}$, the hold time of $Areg$ as $t_{hold_{i,p}}$. The hold constraint can be represented by the following inequality.

$$t_{minhdp_{i,p}} + CT \times num > t_{maxhcp_{i,p}} + t_{hcpm_{i,p}} + t_{hold_{i,p}} \quad (5)$$

If the hold constraint is violated, we must adjust the number of cells for hd_i (hd_0).

Figure 7(b) represents paths related to the hold constraint for $Sreg$ in the synchronous part of $AtoS$ circuit. This hold constraint is the same as the inequality (5).

IV. Experimental Results

In the experiment, we verify the functional correctness of the proposed interface circuits. In addition, we evaluate the latency and handshake overhead of the proposed interface circuits. The latency represents the number of cycles until the receiver receives the data after the sender sends the data to the receiver. The handshake overhead represents the number of cycles until the sender can send the next data to the receiver.

First, by referring to the design flow of asynchronous circuits for commercial Field Programmable Gate Arrays (FPGAs) in [14], we synthesized the proposed interface circuits using Quartus Prime 21.1. The target device was EP4CE115F29C7 (Cyclone IV E). For the synchronous part, we used clock constraints to satisfy the target CT. For the asynchronous part, we used maximum delay constraints for control-paths and local clock constraints for $lclk_i$ to satisfy the target gct. CT and gct were 10 ns, 20 ns, and 30 ns. In addition, we used Design Partitions and LogicLocks for asynchronous control modules and delay elements to reduce the number of delay adjustments by fixing the placement of them.

As a reference, we prepared the extended interface circuit ($StoS$ circuit) based on the two-flop synchronizer [2] as shown in Fig. 8. The interface circuit is based on the four-phase handshake protocol. The sender sends $DATA$ with R to the receiver. Then, the receiver sends A to the sender.

To verify the functional correctness of the proposed interface circuits, we performed a gate-level (GL) simulation using Questa 2021.1 for the synthesized interface circuits. In the GL simulation, the voltage and the temperature were set to 1.2 V and 85°. We think that the GL simulation is comprehensive enough for the evaluation because we synthesized the interface circuits to satisfy the timing constraints by a delay adjustment based on Static Timing Analysis (STA) results. We prepared test patterns for the simulation by giving arbitrary values. After the simulation, we confirmed that all output data of the proposed interface circuits were the same as the input data from the sender.

Figure 9(a), (b), and (c) represent the waveforms of $StoS$ circuit, $StoA$ circuit, and $AtoS$ circuit, respectively. CT was 10 ns and gct was 20 ns. Red arrows represent the behavior of the latency of the interface circuits. Blue arrows represent the behavior of the handshake overhead of the interface circuits. In all circuits, we confirmed that 32-bit data were sent with $Sreq1$ ($Sreq$ or $Areq$) and 32-bit data were received with $Sreq2$ ($Areq$ or $Sreq$) correctly.

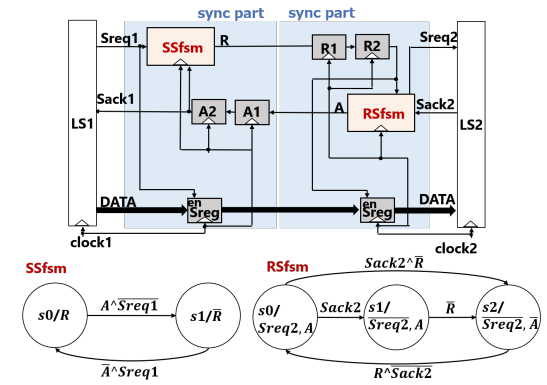

Fig. 8. Extended interface circuit based on [2].

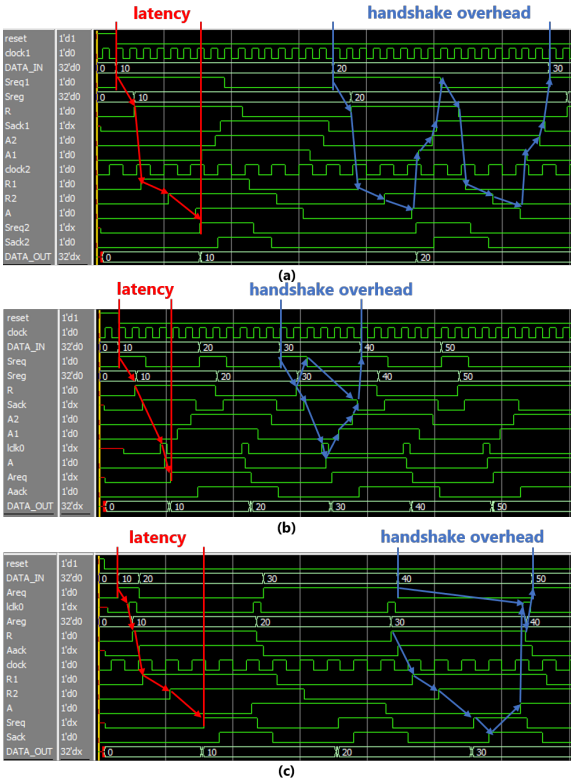

Fig. 9. Waveforms: (a) $StoS$ circuit, (b) $StoA$ circuit, and (c) $AtoS$ circuit.

Table I represents the latencies and handshake overheads of the proposed interface circuits. SCT ($Sgct$), RCT ($Rgct$), $Name$, $Latency$, and $Overhead$ represent CT (gct) of the sender, CT (gct) of the receiver, interface circuits, latencies, and handshake overheads, respectively. The latency and handshake overhead were obtained from

TABLE I
LATENCIES AND HANDSHAKE OVERHEADS OF THE INTERFACE CIRCUITS.

SCT (*Sgct*)	*RCT* (*Rgct*)	*Name*	*Latency*	*Overhead*
10.0	10.0	*StoS*	5.0	14.0
		StoA	3.1	5.0
		AtoS	4.0	5.1
	20.0	*StoS*	7.5	16.0
		StoA	5.2	6.0
		AtoS	7.5	10.1
	30.0	*StoS*	12.0	24.0
		StoA	7.1	7.0
		AtoS	12.0	15.1
20.0	10.0	*StoS*	2.8	10.0
		StoA	2.1	4.0
		AtoS	2.8	2.5
	20.0	*StoS*	5.0	14.0
		StoA	3.1	5.0
		AtoS	5.0	5.1
	30.0	*StoS*	6.8	15.0
		StoA	4.1	5.0
		AtoS	6.8	7.6
30.0	10.0	*StoS*	2.4	10.0
		StoA	1.8	4.0
		AtoS	2.4	1.7
	20.0	*StoS*	3.5	10.0
		StoA	2.4	4.0
		AtoS	3.5	3.4
	30.0	*StoS*	5.0	14.0
		StoA	3.1	5.0
		AtoS	5.0	5.0

the GL simulation.

Table I represents that the latency of the proposed interface circuits depends on the cycle time of the receiver. Compared with *StoS* circuit, *StoA* circuit could reduce the latency by up to 4.9 cycles because the asynchronous part of *StoA* circuit did not use the synchronizers. In contrast, compared with *StoS* circuit, *AtoS* circuit did not have a significant impact on the latency because the synchronous part of *AtoS* circuit used the two-flop synchronizer.

In addition, the proposed interface circuits could reduce the handshake overhead by up to 17.0 cycles compared with *StoS* circuit. This reduction comes from two factors. One is that the asynchronous part used only two signal transitions for each data transfer because it uses the two-phase handshake protocol. Another is that the asynchronous part reduced two cycles because it did not use the two-flop synchronizer.

On the other hand, *AtoS* circuit could reduce the handshake overhead compared with *StoA* circuit if the cycle time of the receiver was smaller than the cycle time of the sender. This is because the synchronous part of *AtoS* circuit completes the signal transitions for the four-phase handshake protocol in a short cycle time.

V. CONCLUSION

In this paper, we proposed interface circuits between synchronous-asynchronous modules using Click Elements. In the experiment, the proposed interface circuits could reduce the latency and handshake overhead by up to 4.9 cycles and 17.0 cycles compared with the interface circuit based on the two-flop synchronizer. As our future work, we are going to evaluate the proposed interface circuits for the connection of processors and accelerators. In addition, we are going to compare the proposed interface circuits and other interface circuits between synchronous-asynchronous modules.

ACKNOWLEDGEMENTS

This work is partially supported by Grant-in-Aid for Scientific Research from Japan Society for the promotion of science (#21K11812).

REFERENCES

[1] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Dept. of Computer Science, Stanford Univ., 1984.

[2] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer," Proc. ASYNC, pp. 89-96, 2003.

[3] D. L. Oliveira, T. Curtinhas, L. A. Faria and L. Romano, "A novel Asynchronous Interface with Pausible Clock for Partitioned Synchronous Modules," 6th Latin American Symposium on Circuits & Systems, pp. 1-4, 2015.

[4] E. Amini, M. Najibi and H. Pedram, "Globally Asynchronous Locally Synchronous Wrapper Circuit based on Clock Gating," IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, pp. 193–199, 2006.

[5] B. Keller, M. Fojtik and B. Khailany, "A Pausible Bisynchronous FIFO for GALS Systems," Proc. ASYNC, pp. 1-8, 2015.

[6] E. Beigne and P. Vivet, "Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture," Proc. ASYNC, pp. 172–183, 2006.

[7] T. Chelcea and S. M. Nowick, "Robust Interfaces for Mixed-Timing Systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 8, pp. 857–873, 2004.

[8] T. Ono and M. Greenstreet, "A Modular Synchronizing FIFO for NoCs," 3rd ACM/IEEE International Symposium on Networks-on-Chip, pp. 224–233, 2009.

[9] M. S. Abdelhadi, "Synthesizable Synchronization FIFOs Utilizing the Asynchronous Pulse-Based Handshake Protocol," IEEE Nordic Circuits and Systems Conference (NorCAS), pp. 1–7, 2020.

[10] F. Huemer and A. Steininger, "Timing Domain Crossing using Muller Pipelines," Proc. ASYNC, pp. 44–53, 2020.

[11] J. Kessels, "Register-Communication between Mutually Asynchronous Domains," Proc. ASYNC, pp. 66–75, 2005.

[12] D. L. Oliveira, K. Garcia, L. A. Faria, J. L. V. Oliveira and L. Romano, "A Synchronous Wrapper for High-Speed Heterogeneous Systems on FPGAs," 2016 IEEE ANDESCON, pp. 1–4, 2016.

[13] A. Peeters, F. te Beest, M. de Wit and W. Mallon, "Click Elements: An Implementation Style for Data-Driven Compilation," Proc. ASYNC, pp. 3-14, 2010.

[14] T. Otake and H Saito, "A Design Method for Designing Asynchronous Circuits on Commercial FPGAs Using Placement Constraints", IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, vol. E103-A, No. 12, pp. 1427–1436, 2020.