A Study on an Interface Circuit for Burst Transfers from Synchronous to Asynchronous Circuits Considering Cycle Times

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 an interface circuit for burst transfers from synchronous to asynchronous circuits. The proposed interface circuit realizes burst transfers in a single handshake cycle. To realize burst transfers, we decide the number of registers from the difference between cycle times of synchronous and asynchronous circuits and burst length. In the experiment, we compared the proposed interface circuit with a FIFO-based interface circuit in terms of energy consumption. The proposed interface circuit could reduce energy consumption by at least 9.7%.

I. Introduction

Most digital systems are designed based on System-on-a-Chip (SoC) composed of several circuits such as microprocessors, memories, specific circuits, etc. When these circuits are controlled by different clock signals, synchronizers are required to reduce the metastability problem between different clock domains.

To solve this problem, [1] proposed Globally Asynchronous Locally Synchronous (GALS) systems composed of several local synchronous circuits. In GALS systems, each local synchronous circuit is controlled by an independent clock signal and communicated with other circuits asynchronously. To guarantee asynchronous communication, interface circuits are required between different synchronous circuits.

To transfer data between synchronous and asynchronous circuits, handshake-based interface circuits were proposed in [2, 3, 4, 5]. Data is stored in the internal register during the handshake process between synchronous and asynchronous circuits. However, these interface circuits transfer only one data in a single handshake cycle. Therefore, these interface circuits are not suitable for burst transfers because handshake communication is performed for each data transfer.

On the other hand, FIFO-based interface circuits proposed in [6, 7, 8, 9, 10] can be used for burst transfers between synchronous and asynchronous circuits. In the FIFO-based interface circuits, data can be transferred by writing data to multi-stage storage in sequence. However, compared with handshake-based interface circuits, the structure of FIFO-based interface circuits tends to be complex because the generation of tokens and full/empty signals is required.

In this paper, we propose an interface circuit for burst transfers from synchronous to asynchronous circuits. The proposed interface circuit is based on the handshake-based interface circuit [5]. The proposed interface circuit realizes burst transfers in a single handshake cycle by deciding the number of registers from the difference between cycle times of synchronous and asynchronous circuits and burst length.

Compared with [5] where multiple handshake cycles are required for burst transfers, the proposed interface circuit realizes burst transfers in a single handshake circuit by deciding the number of registers from the difference between cycle times of synchronous and asynchronous circuits and burst length. As a result, the proposed interface circuit can reduce the data transfer time (i.e., latency). In addition, compared with FIFO-based interface circuits [6, 7, 8, 9, 10] where a control circuit for each stage in the FIFO is required for burst transfers, the proposed interface circuit realizes burst transfers using one control circuit for multiple registers. As a result, the proposed interface circuit can reduce the circuit area.

The rest of this paper is organized as follows. Section II describes asynchronous circuits with bundled-data implementation. Section III describes the handshake-based interface circuit described in [5]. Section IV describes the proposed interface circuit for burst transfers. Section V describes the experimental results. Finally, section VI describes the conclusion and future work.

II. Asynchronous Circuits with Bundled-data Implementation

Bundled-data implementation is one of the data encoding schemes in asynchronous circuits. In bundled-data implementation, N-bit signals are represented by $N+2$ wires including request req and acknowledgment ack signals. The timing for writing data to registers is guaranteed by delay elements on req and ack.

In bundled-data implementation, control schemes are divided into four-phase and two-phase handshake protocols. From here, we represent the rising and falling transitions of a signal as $signal+$ and $signal-$. In the four-phase handshake protocol, four signal transitions ($req+$, $ack+$, $req-$, and $ack+$) are used to transfer data. In the two-phase handshake protocol, two signal transitions ($req+/req-$ and $ack+/ack-$) are used to transfer data.

In this work, we use Click Element [11] to control asynchronous circuits. Click Element is one of the control templates for bundled-data implementation. Click Element is

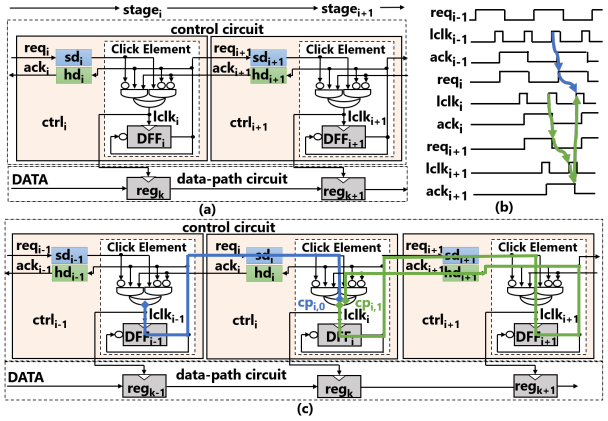

Fig. 1. Asynchronous circuits with bundled-data implementation: (a) circuit model, (b) timing diagram of $ctrl_i$, and (c) control-paths $cp_{i,p}$.

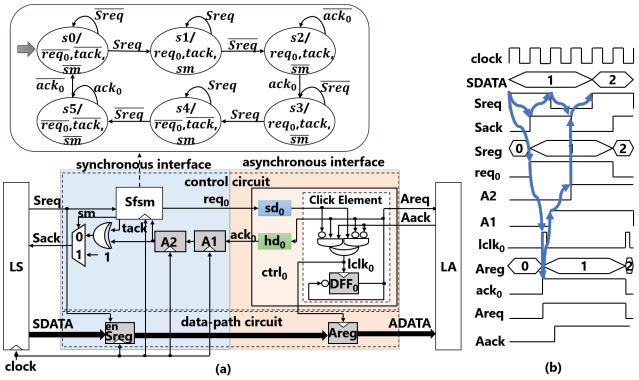

Fig. 2. *StoA* circuit described in [5]: (a) circuit model and (b) timing diagram.

implemented as the two-phase handshake protocol.

Figure 1(a) shows the circuit model of bundled-data implementation used in this work. This circuit model consists of data-path and control circuits. The data-path circuit is the same as the one used in synchronous circuits. The control circuit consists of control modules $ctrl_i$ ($0 \le i \le n\text{-}1$) corresponding to each pipeline stage $stage_i$.

$ctrl_i$ consists of a Click Element and delay elements (sd_i and hd_i). The Click Element consists of a D Flip-Flop (DFF) DFF_i and a logic for a local clock signal $lclk_i$. sd_i and hd_i are used to guarantee the setup and hold constraints of registers reg_k.

Figure 1(b) shows the timing diagram of $ctrl_i$. Blue and green arrows represent the generation of $lclk_i+$ from req_i+ and $ack_{i+1}-$. $ctrl_i$ generates $lclk_i+$ when req_i+ and $ack_{i+1}-$ arrive at the logic for $lclk_i$. $lclk_i+$ controls DFF_i and reg_k at the same time. Then, DFF_i generates ack_i+ to pass the control to $ctrl_{i+1}$. Finally, ack_i+ arrives at $ctrl_{i-1}$ to acknowledge that the operation of $ctrl_i$ is completed. The behavior of $ctrl_i$ for req_i- is the same as the behavior of $ctrl_i$ for req_i+.

As shown in the timing diagram of $ctrl_i$, there are two control-paths for $lclk_i$. From here, we introduce p ($0 \le p \le m-1$) which represents the identifier of paths. Figure 1(c) shows the two control-paths $cp_{i,p}$ for $lclk_i$. $cp_{i,0}$ (blue line) represents a forward path from $lclk_{i-1}$ to $lclk_i$ through sd_i. $cp_{i,1}$ (green line) represents a backward path from $lclk_i$ to $lclk_i$ through hd_{i+1}.

To evaluate the performance of bundled-data implementation, we introduce a local cycle time (LCT) and an asynchronous cycle time (ACT). LCT_i is the cycle time of $lclk_i$. ACT is the maximum value of LCT_i. We define the maximum delay of $cp_{i,p}$ as $t_{maxcp_{i,p}}$. LCT and ACT can be represented by the following equations.

$$LCT_i = max\{t_{maxcp_{i,0}}, t_{maxcp_{i,1}}\} \quad (1)$$

$$ACT = max\{LCT_0, \cdots, LCT_{n-1}\} \quad (2)$$

III. Interface Circuit from Synchronous to Asynchronous Circuits

In this paper, we realize burst transfers by extending the interface circuit described in [5]. Figure 2(a) shows the interface circuit. The interface circuit is called *StoA* circuit because the *StoA* circuit transfers data from a local synchronous circuit (LS) to a local synchronous circuit (LA) using request and acknowledgment signals. $Sreq$ ($Areq$) and $Sack$ ($Aack$) represent request and acknowledgment signals for LS (LA).

The *StoA* circuit is composed of synchronous and asynchronous interfaces. The synchronous interface is controlled by the four-phase handshake protocol. In contrast, the asynchronous interface is controlled by the two-phase handshake protocol.

The synchronous interface is composed of a finite state machine $Sfsm$, a register $Sreg$, a two-flop synchronizer ($A1$ and $A2$), an XOR gate, and a multiplexer. $Sfsm$ is used to generate req_0 for the asynchronous interface. $Sreg$ is used to receive data ($SDATA$) from LS. The two-flop synchronizer is used to synchronize ack_0 from the asynchronous interface. The XOR gate and multiplexer are used to send $Sack$ to LS.

The asynchronous interface is composed of a register $Areg$ and $ctrl_0$. $Areg$ is used to send the received data ($ADATA$) to LA at the appropriate timing in which the asynchronous interface receives req_0 and $Aack$. $ctrl_0$ is used to control $Areg$ at the appropriate timing. Any synchronizer is not used to synchronize req_0 because the timing for writing data to $Areg$ is guaranteed by sd_0 and hd_0.

Figure 2(b) shows the timing diagram of the *StoA* circuit. Blue arrows represent the behavior of the *StoA* circuit. The *StoA* circuit starts its operation when $Sreq+$ from LS arrives at $Sfsm$. The synchronous interface writes $SDATA$ to $Sreg$ using $Sreq+$. To acknowledge that $SDATA$ is written to $Sreg$, $Sfsm$ controls the multiplexer to generate $Sack+$. $Sfsm$ also generates req_0+ to transfer data from $Sreg$ to $Areg$. The asynchronous interface generates $lclk_i+$ using req_0+ and $Aack-$. $lclk_i+$ controls DFF_0 and $Areg$. Then, DFF_0 generates ack_0+ and $Areq+$ to pass the control to LA. ack_0+ arrives at

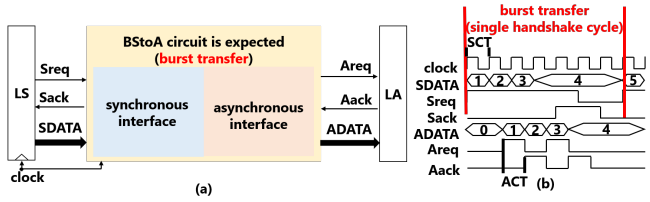

Fig. 3. *BStoA* circuit: (a) block diagram and (b) timing diagram.

$A1$ and $A2$ through hd_i to acknowledge that the operation of the asynchronous interface is completed. Finally, $Sfsm$ controls the multiplexer to generate $Sack-$ using the output of $A2$.

IV. Proposed Interface Circuit from Synchronous to Asynchornous Circuits

In this paper, we propose an interface circuit for burst transfers from synchronous to asynchronous circuits. The proposed interface circuit is based on the *StoA* circuit described in Sect. III. To realize burst transfers in a single handshake cycle, we decide the number of internal registers from the difference between cycle times of synchronous and asynchronous circuits and burst length. We assume that a synchronous cycle time SCT and an asynchronous cycle time ACT are predetermined and the burst length is represented by 2^n. The proposed interface circuit for burst transfers is called *BStoA* circuit.

The *BStoA* circuit realizes burst transfers in a single handshake cycle. Figure 3 shows the image of the *BStoA* circuit. The *BStoA* circuit receives 2^n data with $Sreq+$ from LS in continuous cycles. After receiving 2^n data, the *BStoA* circuit sends $Sack+$ to LS. The *BStoA* circuit also sends the received data with $Areq+$ or $Areq-$ to LA. After sending 2^n data to LA, the *BStoA* circuit sends $Sack-$ to LS to acknowledge that the burst transfer is completed.

A. The Number of Required Internal Registers

For burst transfers, the *BStoA* circuit sends data to LA while receiving 2^n data with $Sreq+$ from LS in continuous cycles. However, there is a case that data from $Sreg$ cannot be written to $Areg$ correctly when SCT and ACT are different. This is because the data of $Sreg$ is updated during the waiting time for writing the data to $Areg$. Figure 4 shows the predicted timing diagrams for burst transfers.

When SCT is longer than ACT as shown in Fig. 4(a), the data of $Sreg$ is updated after the data is written to $Areg$ using req_0. The data of $Sreg$ is transferred to $Areg$ correctly. The latency L for transferring 2^n data from LS to LA can be represented by the following equation.

$$L = 2^n \cdot SCT + ACT \quad (SCT \geq ACT) \tag{3}$$

The latency represents the delay until the *BStoA* circuit sends $Areq$ to LA after receiving $Sreq$. In Fig 4, the between purple lines represents the latency.

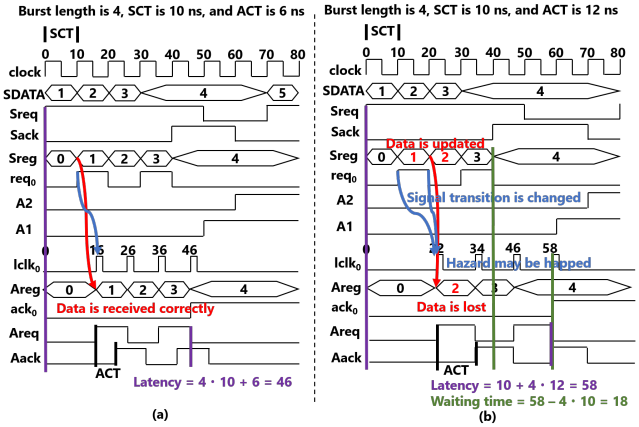

Fig. 4. Predicted timing diagrams for burst transfers: (a) $SCT \geq ACT$ and (b) $SCT < ACT$.

When SCT is shorter than ACT as shown in Fig. 4(b), the data of $Sreg$ can not be written to $Areg$ because the waiting time for writing data to $Areg$ is longer than SCT. As a result, the data of $Sreg$ is updated before the data is written to $Areg$. In other words, the transferred data is lost. Similarly, the next transition of req_0 will arrive at the logic of $lclk_i$ before/after $lclk_i+$ is generated. As a result, $lclk_i+$ will be not generated or $lclk_i+$ will change to $lclk_i-$ during the control of $Areg$. This unexpected signal transition is a hazard for $lclk_i$. The latency L for transferring 2^n data from LS to LA and the waiting time WT for writing data to $Areg$ can be represented by the following equations.

$$L = SCT + 2^n \cdot ACT \quad (SCT < ACT) \tag{4}$$
$$WT = L - 2^n \cdot SCT \quad (SCT < ACT) \tag{5}$$

The waiting time represents the delay until 2^n data are written to $Areg$ after writing 2^n data to $Sreg$. In Fig 4, the between green lines represents the waiting time.

To transfer data correctly, the *BStoA* circuit must hold the data of $Sreg$ until the data is written to $Areg$. To satisfy this condition, we prepare several registers $Sreg_k$ like a memory. The *BStoA* circuit can hold the data of $Sreg_{k-1}$ by writing the next data to $Sreg_k$.

The number of $Sreg_k$ depends on WT because WT affects the number of times that data of $Sreg_k$ is updated before data is written to $Areg$. We define the number of $Sreg_k$ as num_r. num_r can be represented by the following equations.

$$num\_r = \begin{cases} 1 & (SCT \geq ACT) \\ \lceil \frac{WT}{ACT} \rceil & (SCT < ACT) \end{cases} \tag{6}$$

No matter how much the difference between SCT and ACT is large, num_r is at most 2^n because the burst length is 2^n.

Figure 5 shows the timing diagram for burst transfers when the number of $Sreg_k$ is 2 ($num\_r = \lceil \frac{18}{12} \rceil = \lceil 1.5 \rceil$). Note that we use the num_r-input's multiplexer between

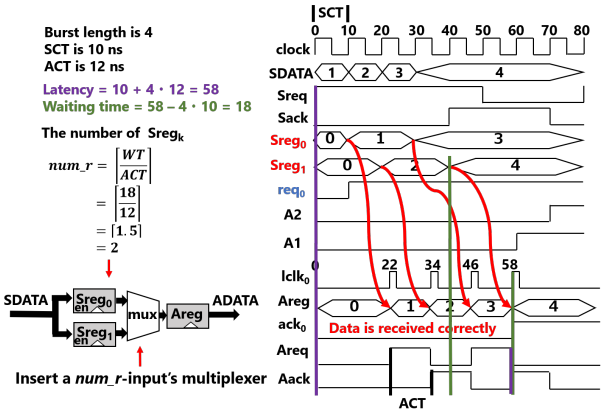

Fig. 5. Tming diagram for burst transfers when $Sreg_k$ are used.

$Sreg_k$ and $Areg$ because data from several $Sreg_k$ must be written to $Areg$ alternately. In addition, we need to fix the signal transition of req_0 during burst transfers to prevent the hazard for $lclk_i$.

B. Circuit Models

Figure 6(a) shows the proposed $BStoA$ circuit ($SCT \geq ACT$). To send $Sack+$ to LS after receiving 2^n data, we add the states of $Sfsm$. The added states loop until 2^n data are received. To check that 2^n data are received, we insert a counter $Scount$. Similarly, to send ack_0+ to the synchronous interface after receiving 2^n data, we insert a counter $Acount$ and DFF in the asynchronous interface. The bit-widths of $Scount$ and $Acount$ increase by $\log_2 n$ depending on the burst length.

Figure 6(b) shows the proposed $BStoA$ circuit ($SCT < ACT$). By referring to num_r obtained from equation (6), we insert $Sreg_k$ in the synchronous interface and insert a num_r-input's multiplexer in the asynchronous interface. To prevent the hazard for $lclk_0$, we fix the signal transition of req_0 from $Sfsm$. The signal transition of req_0 changes only once when $Sreq+$ arrives at $Sfsm$. As a result, the signal transition of $lclk_0$ changes only once. For writing 2^n data to $Areg$, $ctrl_0$ must generate $lclk_0+$ 2^n times. To generate $lclk_0+$ 2^n times, we create a new request signal $nreq_0$ by inserting DFF in the asynchronous interface. By inserting an XOR gate with req_0 and $nreq_0$ as inputs, $ctrl_0$ generates $lclk_0+$ 2^n times.

C. Timing Constraints

To guarantee the operation of the $BStoA$ circuit, it is necessary to satisfy setup and hold constraints for the internal registers in the $BStoA$ circuit. The setup and hold constraints for $Sreg_k$ can be verified by Static Timing Analysis (STA) for LS. In this paper, we define the setup and hold constraints for $Areg$.

Figure 7 shows the timing paths for the $BStoA$ circuit. We check the data-paths $dp_{i,p}$ and control-paths $cp_{i,p}$ for $Areg$. $dp_{i,p}$ (red line) is a path from the clock signal to

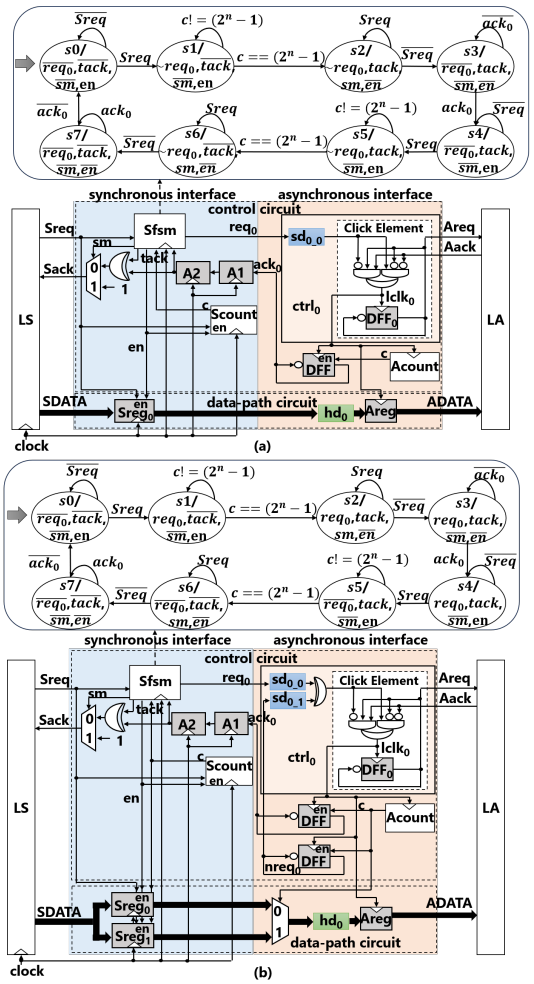

Fig. 6. $BStoA$ circuit: (a) $SCT \geq ACT$ and (b) $SCT < ACT$.

$Areg$ through $Sreg_k$. There are three types of paths for $cp_{i,p}$. $cp_{i,0}$ (blue line) is a path from the clock signal to $Areg$ through $ctrl_i$. $cp_{i,1}$ (green line) is a path from $lclk_i$ to $lclk_i$ through LA. $cp_{i,2}$ (purple line) is a path from $lclk_i$ to $lclk_i$ through sd_{0_1}.

Before the explanation for the timing constraints, we define variables. We represent the maximum and minimum delays of $dp_{i,p}$ as $t_{maxdp_{i,p}}$ and $t_{mindp_{i,p}}$. We represent the maximum and minimum delays of $cp_{i,p}$ as $t_{maxcp_{i,p}}$ and $t_{mincp_{i,p}}$. We represent the setup and hold times of $Areg$ as $t_{setup_{i,p}}$ and $t_{hold_{i,p}}$. We represent the margins for $t_{maxdp_{i,p}}$ and $t_{maxcp_{i,p}}$ as $t_{dpm_{i,p}}$ and $t_{cpm_{i,p}}$.

Setup Constraint. The input data for $Areg$ must be stable before the setup time to write the input data to $Areg$. This is called the setup constraint for $Areg$. The setup constraint can be represented by the following inequality.

$$t_{mincp_{i,0}} > t_{maxdp_{i,p}} + t_{dpm_{i,p}} + t_{setup_{i,p}} \quad (7)$$

If this setup constraint is violated, we must adjust the number of cells in sd_{0_0}.

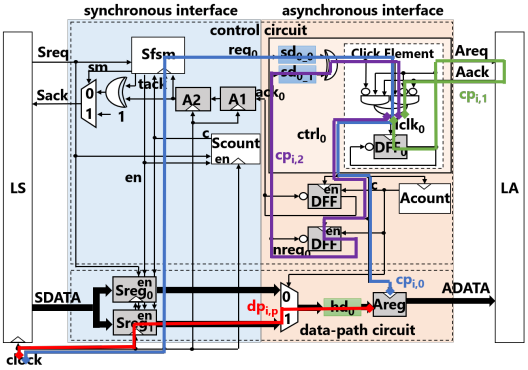

Fig. 7. Data-path $dp_{i,p}$ and control-path $cp_{i,p}$ for *BStoA* circuit.

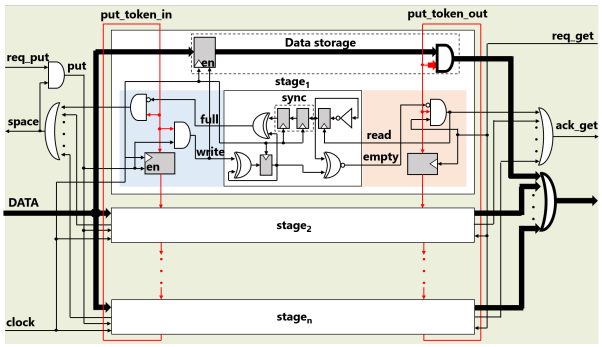

Fig. 8. FIFO-based interface circuit in [10].

When SCT is shorter than ACT, $lclk+$ is generated by req_0 or $nreq_0$. We also consider the delay of $nreq_0$ which is used for writing data from the second time onwards to $Areg$. Since the data of $Sreg_k$ is changed at each SCT, the delay of $nreq_0$ must be longer than SCT. The delay of $nreq_0$ must be satisfied by the following inequality.

$$SCT \leq t_{mincp_{i,2}} < ACT \quad (8)$$

If this constraint is violated, we adjust the number of cells in sd_{0_1}. Since the generation timing of $lclk+$ depends on $t_{maxcp_{i,p}}$, we do not need to adjust the number of cells in sd_{0_1} if $t_{mincp_{i,1}}$ is longer than SCT.

Hold Constraint. The data must be stable for the hold time after the input data are written to $Areg$. This is called the hold constraint for $Areg$. The hold constraint can be represented by the following inequality.

$$t_{mindp_{i,p}} + SCT \cdot num\_r > t_{maxcp_{i,0}} + t_{hcpm_{i,p}} + t_{hold_{i,p}} \quad (9)$$

If this hold constraint is violated, we must adjust the number of cells in hd_0. Note that the hold violations rarely occur because we decide the number of $Sreg_k$ to write data to $Areg$ correctly when SCT is shorter than ACT.

V. Experimental Results

In the experiment, we compare the proposed *BStoA* circuit with a FIFO-based interface circuit represented in [10] in terms of latency, circuit area, dynamic power consumption, and energy consumption. We selected [10] for the comparison because the FIFO-based interface circuit in [10] is synthesizable using standard libraries without specific libraries for asynchronous circuits. In addition, [10] addressed many interface circuits as related work. Note that the latency represents the delay until the interface circuits send the request signal to the receiver after receiving the request signal from the sender.

First, we prepared the Verilog HDL of the *BStoA* circuit. To check the quality of the *BStoA* circuit when the cycle time and burst length change, we set the burst lengths to 8, 16, and 32, SCT to 15 ns, and ACT to 13 ns and 17 ns. By referring to the burst length, SCT, and ACT, we calculated the number of $Sreg_k$ using equation (6).

Then, we synthesized the *BStoA* circuit using Quartus Prime 20.1 by referring to the design flow of interface circuits for commercial Field Programmable Gate Arrays (FPGAs) in [12]. The target device was EP4CE115F29C7 (Cyclone IV E). For the synchronous interface, we used the clock constraints to satisfy SCT. For the asynchronous interface, we used the maximum delay constraints for control-paths and local clock constraints for $lclk_i$ to satisfy ACT. In addition, we used Design Partitions and LogicLocks for the interfaces to reduce the number of delay adjustments by fixing the placement of them.

To verify the functional correctness of the *BStoA* circuit, we performed a gate-level (GL) simulation using ModelSim-Intel FPGA Edition 2020.1. We prepared test patterns for the simulation by giving arbitrary values. After the simulation, we confirmed that all output data of the *BStoA* circuit were the same as the input data. In addition, we confirmed that the *BStoA* circuit realized burst transfers in a single handshake cycle.

We designed the FIFO-based interface circuit in [10]. Figure 8 shows the FIFO-based interface circuit. The FIFO-based interface circuit receives DATA with req_put+. If the FIFO storage is full, $space-$ is generated. When the data of the FIFO-based interface circuit are read by req_get+, ack_get+ is generated. To make a fair comparison between the *BStoA* circuit and FIFO-based interface circuit, we decided the number of FIFO stages by referring to num_r. To realize burst transfers, we decided that the number of the stages in the FIFO was $num\_r + 2$ because the signal indicating whether the FIFO is full or not was delayed two cycles by the two-flop synchronizer.

Figure 9(a) shows the latency of the *BStoA* circuit. The latency was obtained from the GL simulation using ModelSim. Compared with the FIFO-based interface circuit, the *BStoA* circuit did not have a significant impact on the latency because the latencies of the *BStoA* circuit and the FIFO-based interface circuit depend on the slower value of SCT and ACT.

Figure 9(b) shows the number of Logic Elements (LEs) of the *BStoA* circuit. The number of LEs was ob-

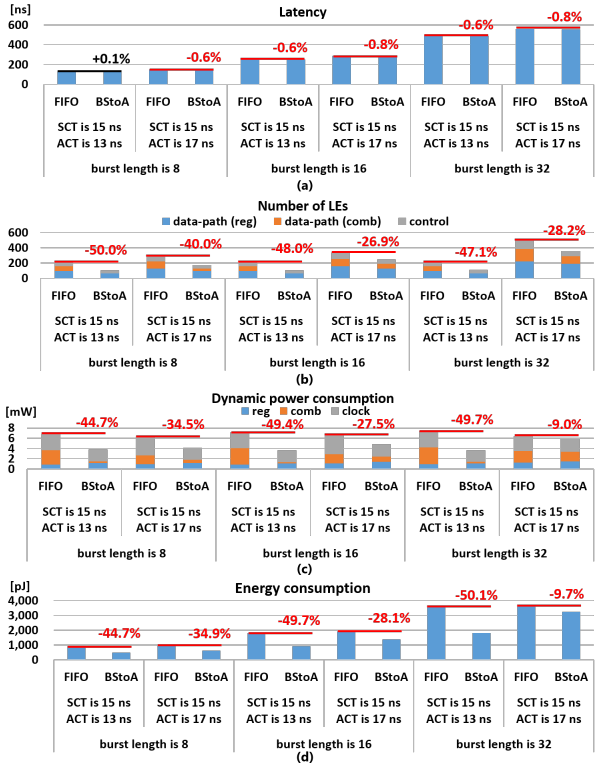

Fig. 9. Evaluation resualts: (a) latency, (b) number of LEs, (c) dynamic power consumption, and (d) energy consumpion.

tained from the synthesis report generated by Quartus Prime. Compared with the FIFO-based interface circuit, the number of LEs in the *BStoA* circuit was reduced in all cases. This is because the number of $Sreg_k$ was smaller than the number of stages in the FIFO. In addition, one control circuit was required for several $Sreg_k$, but control circuits were required for each stage in the FIFO.

Figure 9(c) shows the dynamic power consumption of the *BStoA* circuit. The dynamic power consumption was obtained by PowerPlay Power Analyzer with the value change dump file generated by ModelSim during GL simulation. Compared with the FIFO-based interface circuit, the dynamic power consumption of the *BStoA* circuit was reduced in all cases because of the reduction of the number of LEs.

Figure 9(d) shows the energy consumption of the *BStoA* circuit. The energy consumption was the product of the latency and the dynamic power consumption. Compared with the FIFO-based interface circuit, the energy consumption of the *BStoA* circuit was reduced in all cases because of the reduction of the dynamic power consumption.

Compared with the FIFO-based interface circuit, the proposed interface circuits could reduce energy consumption by at least 9.7% because of the reduction of the circuit area and dynamic power consumption. The number of $Sreg_k$ is smaller than the number of stages in the FIFO. Moreover, the proposed *BStoA* circuit uses one control circuit for $Sreg_k$ while the FIFO-based interface circuit uses a control circuit for each stage in the FIFO.

VI. Conclusion

In this paper, we proposed an interface circuit for burst transfers from synchronous to asynchronous circuits. The proposed interface circuit realized burst transfers in a single handshake cycle by deciding the number of registers from the difference between cycle times of synchronous and asynchronous circuits and burst length. In the experiment, the proposed interface circuit could reduce energy consumption by at least 9.7% compared with a FIFO-based interface circuit. In our future work, we are going to design an interface circuit for burst transfers from asynchronous to synchronous circuits. In addition, we are going to extend the interface circuit to deal with standard interfaces such as AXI.

Acknowledgements

This work is partially supported by Grant-in-Aid for Scientific Research from Japan Society for the promotion of science (#21K11812 and #23K16860) .

References

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

[2] A. E. Sjogren and C. J. Myers, "Interfacing Synchronous and Asynchronous Modules Within a High-Speed Pipeline," Proc. 7th Conference on Advanced Research in VLSI, pp. 47–61, 1997.

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

[4] D. L. Oliveira, et al, "A Synchronous Wrapper for High-Speed Heterogeneous Systems on FPGAs," 2016 IEEE ANDESCON, pp. 1–4, 2016.

[5] S. Semba and H. Saito, "A Study on the Design of Interface Circuits Between Synchronous-Asynchronous Modules Using Click Elements," Proc. SASIMI 2022, pp. 139–144, October, 2022.

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

[10] M. S. Abdelhadi, "Synthesizable Synchronization FIFOs Utilizing the Asynchronous Pulse-Based Handshake Protocol," IEEE NorCAS, pp. 1–7, 2020.

[11] A. Peeters, et al, "Click Elements: An Implementation Style for Data-Driven Compilation," Proc. ASYNC, pp. 3–14, 2010.

[12] S. Semba and H. Saito, "A Design Support Tool Set for Interface Circuits Between Synchronous and Asynchronous Modules," in IEEE Access, vol. 11, pp. 13408–13420, 2023.