World Academy of Science, Engineering and Technology
International Journal of Electronics and Communication Engineering
Vol:3, No:6, 2009

A ring segmented bus architecture for Globally Asynchronous Locally Synchronous System

Masafumi KONDO, Yoichiro SATO, Kazuyuki TASHIRO, Tomoyuki YOKOGAWA, and Michiyoshi HAYASE,

Abstract—**Recently, most digital systems are designed as GALS (Globally Asynchronous Locally Synchronous) systems. Several architectures have been proposed as bus architectures for a GALS system : shared bus, segmented bus, ring bus, and so on. In this study, we propose a ring segmented bus architecture which is a combination of segmented bus and ring bus architecture with the aim of throughput enhancement. In a segmented bus architecture, segments are connected in series. By connecting the segments at the end of the bus and constructing the ring bus, it becomes possible to allocate a channel of the bus bidirectionally. The bus channel is allocated to the shortest path between segments. We consider a metastable operation caused by asynchronous communication between segments and a burst transfer between segments. According to the result of simulation, it is shown that the GALS system designed by the proposed method has the desired operations.**

Index Terms—**GALS systems bus architecture, segmented bus, ring bus.**

I. Introduction

WITH the development of circuit integration technology, the scale and speed of circuit have been increasing. Digital systems are usually implemented as synchronous circuits. However, relative growth of wire delay to element delay makes it difficult to deliver clock pulses to all components in phase[1]. Therefore, we can not but decrease the clock frequency and a limit is occurring at the speed of synchronous circuits. To solve the problem, the method to design the system as an asynchronous circuit composed of subsystems which are designed as synchronous circuits (called *Clock Domain*, CD) gathers much attention. Such a system is called a *Globally Asynchronous Locally Synchronous* (GALS) system[2]. The performance of GALS systems depends on the efficiency of asynchronous data transfer between CDs. To improve the performance of GALS systems, it is important to provide the configuration of the bus assuming the data transfer between CDs.

The bus architecture of GALS systems is classified into four types: shared bus, switch network, ring bus and segmented bus[3][4]. A shared bus architecture is scalable and requires small area to configure the bus and has a simple mechanism to arbitrate a conflict of bus requests. In the case of a large scale system, however, the throughput degradation for a data transfer occurs because one CD occupies all part of the bus, and increasing delay associated with the growth of bus length enlarges the response time of bus control. A switch network architecture is sperior to shared bus at a point of the throughput because the bus is constructed by combining small matrix switches. The scalability of the switch network is however limited, and the response time and the required area are enlarged. In a ring bus architecture, transmission circuits of CDs are connected in the form of a ring and data are transferred based on a packet communication. Although this allows the throughput enhancement, it is inferior to a shared bus in the required area, the scalability, and the response time. In a segmented bus architecture a bus is dynamically divided into multiple buses which can be used concurrently. Although the required area is enlarged relative to a shared bus, this allows lower power consumption as well as high throughput enhancement.

As mentioned above, while the required area is enlarged, a segmented bus architecture is the most up-and-coming in aspects of throughput and response time. In the case of data transfer between the CDs (called *segments* in a segmented bus architecture) at both ends of the bus, however, other segments can not use the bus. This leads to throughput degradation. To solve the problem, we construct the bus in the form of a ring by connecting the segments at both ends of the bus. In this paper, we propose the bus architecture which is a combination of ring bus and segmented bus architecture. We call this form of bus architecture *ring-segmented bus*.

In a segmented bus architecture, a direction of data transfer channel is uniquely determined because segments are connected in series via a bridge. Thus when allocating a data transfer channel to a path, in the case that the other channel is already allocated there, it is not possible to establish the channel. Particularly, in the case of transfer between the segments at both ends of the bus, the throughput seriously degrades because all parts of the bus are occupied. Connecting segments at both ends of the bus, data transfer channel can be allocated to two types of path in clockwise and counter-clockwise rotation. Even if a data transfer channel is already allocated, there is a chance to establish another channel and it allows the throughput enhancement. Allocating a channel to the shortest path among the two type of path, a data transfer is performed efficiently. Creating an independent segment bus from the bus used to transfer data between segments, data

M. Kondo, Y. Sato, T. Yokogawa and M. Hayase are with Graduate School of Systems Engineering, Okayama Prefectural University, Kuboki 111, Soja-shi, Okayama, 719–1197 Japan (e-mail: kondo@radish3.cse.oka-pu.ac.jp, {sato, t-yokoga, hayase}@cse.oka-pu.ac.jp).

K. Tashiro is with FUJITSU TEN, Goshodoori 1-2-8, Hyougo-ku, Koube-shi, Hougo, 652–8510 Japan (e-mail: t-kazu@tm.ten.fujitsu.com).

Manuscript received February 26, 2009; revised March 31, 2009.

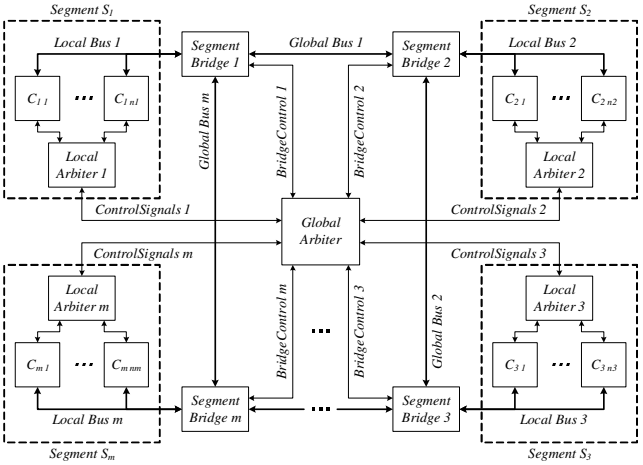

Fig. 1. Structure of a ring segmented bus architecture

transfer inside a segment can be performed during data transfer via the segment.

In section II, we show an outline of the ring segmented bus architecture and the data transfer method. In section III, we show a structure of a segment bridge connecting segments. In section IV, we show a structure of an arbiter. In section V, we show the result of design and simulation of the ring segmented bus architecture.

II. Ring segmented bus architecture

A. Components

Figure 1 shows the basic architecture of ring segmented bus. We show the function of each component as follows.

- *Segment* S_i $(1 \leq i \leq m)$
  is a set of components which is designed as a synchronous circuit. A segment includes a local bus connecting components and an arbiter arbitrating the local bus mentioned below.
- *Component* C_{ij} $(1 \leq j \leq n_i)$
  is a component included in the segment S_i.
- *Local Bus* LB_i
  is a bus transferring data in the segment S_j.
- *Local Arbiter* LA_i
  is an arbiter of the segment S_j. This arbiter arbitrates following three requests and grants the bus.
  1) request to transfer data between components (C_{ij}, C_{il}) in the segment S_i.
  2) request to transfer data to a component C_{jl} in different segment S_j $(i \neq j)$.
  3) request to transfer data from a component C_{jl} in different segment S_j $(i \neq j)$.

  These types of requests are specified the priority and request type 3) has top priority. This is to reduce the overhead doe to unsuccess of the channel establishment, when such a request occurs and the channel from C_{jl} to S_i is already established. When multiple components issue request type 3) or issue request type 1) and 2) simultaneously, the arbiter grants the bus LB_i based on the priorities of components.
- *Global Bus* GB_i
  is a bus transferring data between the segment S_i and S_{i+1} (when $i = m$, S_m and S_0).
- *Global Arbiter* GA
  is an arbiter controlling the global bus. A global arbiter arbitrates requests of data transfer between segments based on the priorities of segments. It issues a request to establish a data transfer channel in global bus and a request to use the destination local bus. The detailed structure of a global arbiter is described in chapter IV.
- *Segment Bridge* SB_i
  is a circuit connecting local bus LB_i and global buses GB_{i-1}, GB_i physically. A segment bridge connects GB_{i-1} and GB_i, and separate S_i from a global bus. In the case of data transfer from or to S_i, LB_i is connected to GB_i or GB_{i-1} according to a control signal of GA. From here we represent the connection from S_i to S_{i+1} as *Up-direction* and the connection to S_{i-1} as *Down-direction*. Note that the Up-directional segment of S_m is S_0 and the Down-directional segment of S_0 is S_m.
- *Bridge Control* BC_i
  is a control signal delivered from GA to SB_i and specify the connection between LB_i, GB_i and GB_{i-1}.
- *Control Signals* i
  is a control signal between GA and LA_i and used to establish the data transfer channel involving S_i.

B. Data transfer method

Figure 2 shows the detailed control signals involved in S_i. We show the data transfer method in the ring segmented bus architecture.

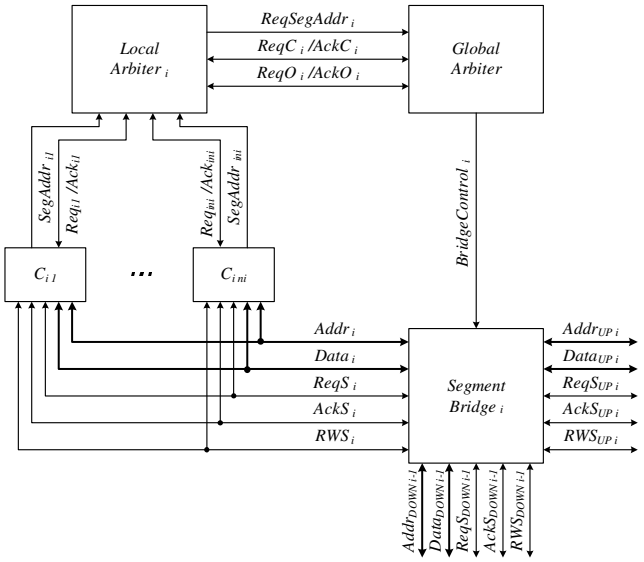

Fig. 2. Control signals involved in a segment

The component C_{ij} sends to LA_i a data transfer request signal Req_{ij} and destination segment address $SegAddr_{ij}$.

LA_i arbitrates Req_{ij} and determines whether it is a data transfer within a segment or not based on $SegAddr_{ij}$. When it is a data transfer within a segment, LA_i sends a data transfer acknowledgement signal Ack_{ij} to the component C_{ij} and grants the bus LB_i. When it is a data transfer between segments, LA_i sends to GA a data transfer request signal $ReqC_i$ and the destination segment address $ReqSegAddr_{ij}$, and requests GA to establish a data transfer channel. The subsequent procedure to establish the data transfer channel is as follows.

GA arbitrates $ReqC_i$ and calculate the channel length from the addresses of S_i and the destination segment S_k. GA attempts to allocate a data transfer channel to the shortest path first, and then to another path. When it is possible to establish the data transfer channel, GA sends a local bus usage request signal $ReqO_k$ and requests to grant the local bus LB_k in S_k. When the channel can not be allocated to the both paths, $ReqC_i$ is not accepted.

In case that LB_k is free, receiving $ReqO_k$, LA_k sends to GA a local bus usage acknowledgement signal $AckO_k$ and grants LB_k to GA. In case that LB_k is not free, LA_k sends $AckO_k$ to GA after completing the current data transfer.

Receiving $AckO_k$, GA sends the signals BC_i and BC_k to the bridges SB_i and SB_k respectively, and establishes a data transfer channel. Since each segment bridge generally connects global buses, it is not necessary to control the bridge of the overpassed segment. After establishing the data transfer channel, GA sends a data transfer acknowledgement signal $AckC_i$.

Receiving $AckC_i$, LA_i sends Ack_{ij} to the component C_{ij} and grant the segment bus.

The component C_{ij} which is accepted the data transfer sends a handshake request signal $ReqS_i$ to the data transfer channel. It simultaneously sends the data destination address $Addr_i$ which has the addresses of S_k and the destination component, and a data transfer direction signal RWS_i. Components in the channel determine whether $ReqS_i$ is addressed to them or not by comparing $Addr_i$ and their addresses. Receiving $ReqS_i$, the component of S_k sends the handshake acknowledgement signal $AckS_k$ to the data transfer channel. Receiving $AckS_k$, the source component sends the data in the case of writing transfer (that is, $RWS_i =' 0'$), and receives the data in the case of reading transfer (that is, $RWS_i =' 1'$).

III. Structure of a Segment Bridge

Since each component is synchronized with the clock pulse of the segment which the component belongs to, signals between segments change asynchronously. Such an asynchronous signal change induces a metastable operation and causes malfunction of a system at worst. To reduce the frequency of such a malfunction, it is necessary to insert a synchronizer to a segment bridge and synchronize input signals[5]. Input and output signals of a segment bridge are classified into data signals ($Data_i$, $Data_{Up\_i}$, $Data_{Down\_i}$) and control signals. In order to improve data transfer efficiency, only control signals are synchronized. We divide a segment bridge into a part processing control signals and a part processing data signals, and show the structure of them separately. We omit the index i of each symbol after here, because all segment bridges have the same structure.

A. Processing control signals

Data signals ($Data$) and address signals ($Addr$) are composed of groups of signals. If a synchronizer is inserted into each group of signals, a gap of timing for synchronization can occur because of the dependency of a timing for synchronization on the arrival time at synchronizer. Thus the group of signals which are asserted simultaneously must not be synchronized individually.

To synchronize multiple groups of signals, we introduce a register which holds the values of the group of signals on the bus and synchronize only the handshake signals. The most simple synchronizer is constructed as series connection of flip flops (FFs). A reason for connecting FFs in series is to reduce the frequency of metastable operation[6]. We show the structure of a segment bridge with a synchronizer in Figure 3.

The handshake signals indicate the timing where the component in the receiver segment samples the signal group and the timing where the component in the sender segment updates the value of the register. Although the group of signal is not synchronized, the values are transferred from the register of the sender segment to the one of the receiver.

B. Processing data signals

In many systems, a certain amount of data is transferred between components continuously. Since it is not possible to perform such a burst transfer between segments by register allocated to each segment, it is necessary to use FIFO (First In First Out). FIFO outputs data in order of storing the data and can maintain the consistency of continuous data. Here, each component is synchronized with a clock of the segment which the component belongs to, the timings of writing and reading are different. Thus it is necessary to use asynchronous FIFO which has independent writing and reading clock.

We show the structure of a segment bridge with an asynchronous FIFO in Figure 4. Inserting MUX and DEMUX in the input and the output of an asynchronous FIFO, the data bus is connected with the asynchronous FIFO only in the case of data transfer between segments.

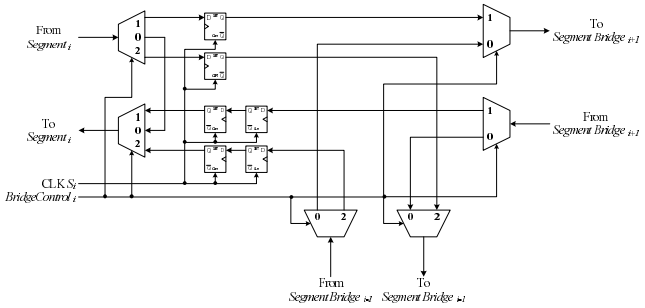

Fig. 3. Structure of a segment bridge with a synchronizer

If writing speed of the sender segment is fast relative to reading speed of the receiver, there is the possibility that the asynchronous FIFO overflows. If writing speed is slow relative to reading speed, underflow may occur, similarly. Therefore, $Full$ and $Empty$ flag signals which indicate its state of data are prepared in an asynchronous FIFO. Signals involved with an asynchronous FIFO and a clock pulse of each segment are sent and received between segments via each bridge.

IV. Structure of an Arbiter

The ring segmented bus architecture includes a local arbiter (LA) and a global arbiter (GA). In this paper, we show only the structure of GA which has complex structure relative to LA.

Since we design GA by HDL at the functional level, we show the method to process conflict. As mentioned above, there exist two channels to transfer data in the ring segmented bus architecture. We also show the method to determine the channel.

A. Global arbiter

GA arbitrates $ReqC$ which is sent from LA of each segment. Since GA receives $ReqC$ asynchronously, GA need to be designed as an asynchronous arbiter.

A state register which indicates the global bus usage is prepared in GA. The size of the state register corresponds to the number of segments and each bit of the register corresponds to each segment state. When the ith bit of the state register is $'1'$, the global bus GB_i and GB_{i-1} is in use, that is, SB_i is used to the data transfer between segments.

We show the arbitrating procedure of GA.

Step1. Detecting a leading of $ReqC$, go to Step 2. Detecting a trailing of $ReqC$ which is already accepted, GA initializes the corresponding state register and quits.

Step2. GA calculates a data transfer channel from the source and destination addresses. GA compares the calculated data transfer channel with the corresponding value of the state register. When the channel can be established, $ReqC$ is set to a valid data transfer request signal $ValidReq$ and go to Step 3. Otherwise, go back to Step 1.

Step3. GA arbitrates $ValidReq$ and updates the value of the state register corresponding to the data transfer channel and go to Step 4.

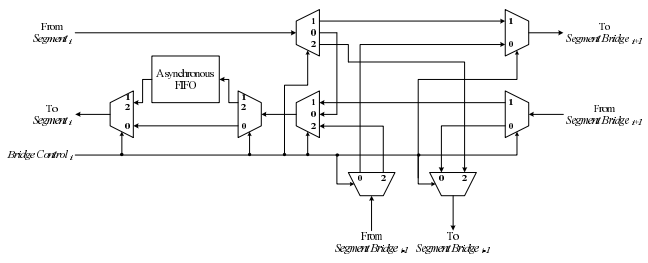

Fig. 4. Structure of a segment bridge with an asynchronous FIFO

Step4. GA sends $ReqO$ to LA of the destination segment and requests to use LB of the destination segment and go to Step 5.

Step5. When GA receives $AckO$, that is, LB of the destination segment is accepted to use, go to Step 6.

Step6. GA sends a bridge control signal to the source and destination segments and establishes a data transfer channel. When establishing the Up-directional channel, the data transfer channel is established by connecting the bridge of source Up-directionally and the one of destination Down-directionally. After establishing the channel, go to Step 7.

Step7. GA sends $AckC$ to LA of the segment sending the accepted $ValidReq$.

B. Calculating the direction of the shortest path

The Up-directional and Down-directional data transfer channels are stored to each channel register based on the addresses of source and destination segments. Comparing the channel register with the state register, the channel can be established if no conflict occurs. Since the channel prefers to be allocated to the shortest path, it is necessary to determine whether the direction of the shortest path is Up-direction or Down-direction.

In order to determine the direction of the shortest path, the positional relation between the source and destination segments needs to be clarified. When the destination segment is behind the diagonal of the source, the direction of shortest path is Up-direction. Otherwise, the direction of shortest path is Down-direction. The positional relation between segments is determined from the distance between segments. The distance is determined from the addresses of the source and destination segments. The distance unit is not the physical distance but the number of segments in the data transfer channel. When the source segment is x, the destination segment is y and the distance of the segments is d, then $d = y - x$. Thus the direction of the shortest path is determined from the following formulas.

$$\begin{aligned} \text{Up}-\text{direction} &: -m < d \leq -\frac{m}{2},\ 0 < d \leq \frac{m}{2}. \\ \text{Down}-\text{direction} &: -\frac{m}{2} < d < 0,\ \frac{m}{2} < d < m. \end{aligned}$$

V. Design of Ring Segmented Bus Architecture

A. Burst transfer

In order to transfer data between segments continuously, the data need to be sent and received with appropriate timing based on the flag signal of the asynchronous FIFO.

We first show the writing transfer. Figure V-A shows the timing chart of burst transfer, where four data (D_1 D_2 D_3 D_4) are transferred from the component C_{ij} of S_i to the component C_{kl} of S_k.

$ReqS$ from C_{ij} arrives at C_{kl} one and two clock cycles later by the synchronizer in the bridge of S_i and S_k, respectively. $AckS$ from C_{kl} arrives at C_{ij} clock cycles later, similarly. When there is empty area in the asynchronous FIFO in the bridge of S_k, that is, $Full$ flag does not assert, C_{ij} sends data

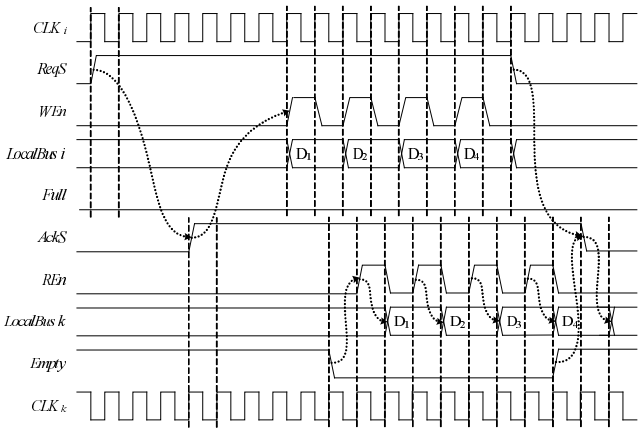

Fig. 5. Timing chart of burst transfer

with a write enable signal WEn. On the other hand, there is data in the asynchronous FIFO, that is, $Empty$ flag does not assert, C_{kl} sends a read enable signal REn to FIFO and receives data from FIFO. C_{ij} negates $ReqC$ after sending the whole transferred data. When $ReqS$ is negated and there is not data in FIFO, C_{kl} finishes to receive data and deasserts $AckS$.

We next show the reading transfer. The handshake by $ReqC$ and $AckS$ is similar to the writing transfer. When there is empty area in the asynchronous FIFO in the bridge of S_i, that is, $Full$ flag does not assert, C_{kl} sends data with a write enable signal WEn. On the other hand, there is data in the asynchronous FIFO, that is, $Empty$ flag does not assert, C_{ij} sends a read enable signal REn to FIFO and receives data from FIFO. C_{ij} negates $ReqC$ after receiving the whole transferred data. When $ReqS$ is negated, C_{kl} finishes to receive data and deasserts $AckS$.

As stated above, it is necessary to detect each flag to perform the burst transfer. Thus two clock cycles are required to transfer one data.

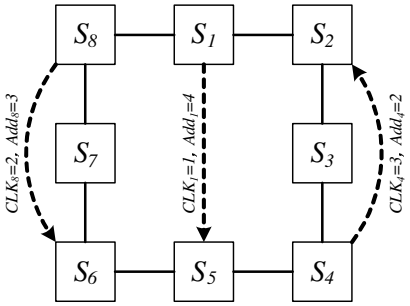

Fig. 6. Structure of the designed GALS system

B. Simulation

We design a GALS system with the ring segmented bus and implement it on FPGA. Since the design of an asynchronous FIFO includes a complex pointer circuit, we construct the asynchronous FIFO using IP cores. We construct each component as a circuit which adds to the contents of RAM the specified number Add_i. We run a simulation, where the number of segments is 8 and the number of components of each segment is 4. Figure 6 shows the structure of the system. Figure 7 shows the result of simulation of the data transfer described by the broken lines in Figure 6. CLK_i in Figure 6 represents the dividing ratio of the segment S_i to the global clock.

First the component of S_1 which has the highest frequency requests a data transfer between segments. Then the data transfer channel is established Up-directionally as the shortest path and the transfer starts. S_8 starts the transfer Down-directionally as the shortest path. Although S_4 subsequently requests the transfer, all channels are occupied by the data transfer of S_1 and S_8. S_4 thus starts the transfer Down-directionally as the shortest path after completing the transfer

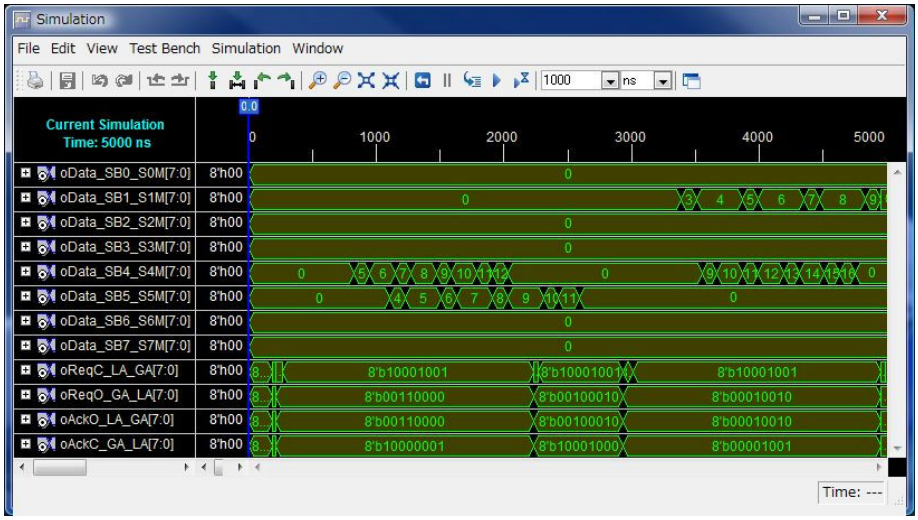

Fig. 7. Result of simulation

of S_1. Here, S_1 requests the transfer again after completing the transfer of S_7. Although the channel can not be allocated to the shortest path similar to the previous transfer because of the transfer of S_4, the channel is allocated to the roundabout path Down-directionally.

We implement the designed ring segmented bus architecture on FPGA (Spartan-3 from Xilinx). The implemented GALS system has the desired operations. Because of the capacity of FPGA devices, the implemented system has four segments and two components in each segment.

VI. Conclusion

In this paper, we proposed a ring segmented bus architecture for a GALS system. We designed the architecture and implemented it on FPGA. According to the simulation, it was shown that the GALS system designed by the proposed method had the desired operations. We considered the reduction of metastable opearation caused by an asynchronous communication and the burst transfer using an asynchronous FIFO. Future works are the detailed design of an asynchronous arbiter and optimum distribution of ring segments.

References

[1] T. Nanya, "A perspective on asynchronous microprocessor," *Journal of Information Processing Society of Japan*, vol. 39, no. 3, pp. 181–186, 1998, (in Japanese).

[2] I. Sutherland and S. Fairbanks, "Gasp: a minimal fifo control," *Seventh International Symposium on Asynchronous Circuits and Systems (ASYNC 2001)*, pp. 46–53, 2001.

[3] T. Villiger, H. Käslin, F. K. Gürkaynak, S. Oetiker, and W. Fichtner, "Self-timed ring for globally-asynchronous locally-synchronous systems," in *ASYNC '03: Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 141.

[4] J. Plosila, T. Seceleanu, and P. Liljeberg, "Implementation of a self-timed segmented bus," *IEEE Design and Test of Computers*, vol. 20, no. 6, pp. 44–50, 2003.

[5] M. Stein, "Crossing the abyss: asynchronous signals in a synchronous world," *EDN Magazine*, pp. 59–69, 7 2003.

[6] Y. Yamasoto, Y. Sato, H. Kagotani, and T. Okamoto, "A formula for performance evaluation of synchronizers constructed with cascaded cmos d flip-flops," *Transactions of the Institute of Electronics, Information and Communication Engineers*, vol. J84, no. D-1, pp. 1484–1492, 1999, (in Japanese).