A Novel Asynchronous First-In-First-Out Adapting to Multi-synchronous Network-on-Chips

Thi-Thuy Nguyen, Xuan-Tu Tran
SIS Laboratory, VNU University of Engineering and Technology, Vietnam National University, Hanoi
144 Xuan Thuy road, Cau Giay, Hanoi, Vietnam
{thuynt_54, tutx}@vnu.edu.vn

Abstract— **The integration of a variety of IP cores into a single chip to meet the high demand of new applications leads to many challenges in timing issues, especially the interface between different clock domains. Globally Asynchronous, Locally Synchronous (GALS) approach addresses these challenges by dividing a chip into several independent subsystems working with different clock signals. In multi-synchronous Network-on-Chip (NoC) based on GALS architecture, the network routers run with different frequencies, so the problem is how to transfer data safely and efficiently between them. In order to build a synchronization unit to tackle this problem, in this paper, we propose a novel efficient asynchronous First-In-First-Out architecture targeting to multi-synchronous NoCs. Token ring structure, register-based memory, and modified Asynchronous Assertion - Synchronous De-assertion techniques are applied to improve the performance of the proposed asynchronous FIFO. After simulating and verifying the design, we have implemented our asynchronous FIFO architecture with CMOS 180*nm* technology from AMS. Implementation results are analyzed and compared with previous works to show the strong points of our design.**

Keywords— Asynchronous FIFO; Multi-synchronous; Network-on-Chip; Globally Asynchronous, Locally Synchronous; Asynchronous Assertion, Synchronous De-assertion

I. Introduction

Network-on-Chip (NoC) paradigm has been known as the evolutionary methodology in solving the bandwidth bottleneck of traditional on-chip interconnections [1], [2]. Although having many potential advantages such as high throughput communication, high scalability and versatility, as well as good power management efficiency, the NoC paradigm must deal with the difficulty of distributing a synchronous clock signals through the entire large area chip. The Globally Asynchronous, Locally Asynchronous (GALS) approach attempts to address this issue by partition the chip into many multiple functional islands [3]. Since each island is clocked by a different clock signal, safely transmitting data between different clock domains becomes a big challenge for designing NoCs based on GALS, especially reconfigurable NoCs.

There are many ways to design a synchronization unit interfacing between two different clock domains, such as using handshake signaling technique [4], parallel synchronizer, shared flop synchronizer [5], pause or stretch the receiver's clock. However, using First-In-First-Out (FIFO) mechanism is one of the most efficient and widely used methods [6]. In addition, in multi-synchronous NoCs, normally each network router has five input/output ports, in which four input/output ports are connected to four neighboring routers. It means that about eight asynchronous FIFOs are needed to interface between different clock domains. Therefore, the implementation area of the proposed asynchronous FIFO plays an important role in the total area cost of the chip. In this work, we designed an asynchronous FIFO to be used for interfacing different clock domains in multi-synchronous NoCs. To make our asynchronous FIFO become robust with a small area overhead, we use token rings and register-based buffers to generate pointers and control reading and writing operations. Besides that, we also apply a modified Asynchronous Assertion, Synchronous De-assertion technique to empty/full detector circuits.

The rest of this paper is arranged as follows. In Section II, we will provide some existed dominant works in FIFO design within their features. The proposed asynchronous FIFO architecture will be described in detail in Section III. After that, the implementation results analysis and comparison with previous works will be provided at Section IV. Finally, conclusions and remarks will be given in Section V.

II. Related Works

Using FIFO is known as an efficient way to design a synchronization unit to communicate between independent subsystems; therefore, there are a large number of FIFO architectures within various features. The following subsections will give a brief overview about the previous works with their characteristic analysis.

A. Bi- synchronous FIFO of Miro Ivan Panades

One of the most efficient FIFO designed for adapting multi-synchronous NoCs is the Bi- synchronous FIFO proposed by Miro Ivan Panades in [6]. In Panades' Bi- synchronous FIFO, pointers are generated by using token-ring structure while data are stored in register-based memory. Despite of having a simple structure as well as being suited for multi-synchronous NoCs, the ways full/empty status is detected make this design have some drawbacks. Firstly, the Bi- synchronous FIFO uses many two-flip-flop synchronizers, leading to the increase in the total implementation area. Secondly, as consequence of full prediction mechanism, the full de-assertion might be delay two more writing clock periods – not included two writing clock periods delayed caused by two-flip-flop synchronizer. This make throughput of the Bi- synchronous FIFO decreased. As a result, the empty detector is made to be more complex to increase the throughput of FIFO.

B. *Asynchronous FIFO of Clifford E.Cummings*

The asynchronous FIFO proposed by Cumming in [7] is very robust FIFO which uses pointers generated from Gray-code counters to control the writing/reading data into/from a dual Random Access Memory (dual-RAM). In particular, two pointers are compared asynchronously to detect FIFO's full/empty status, and then full/empty flags are asserted immediately and de-asserted safely by using Asynchronous Assertion, Synchronous De-assertion (AASD) technique. However, the AASD technique lengthens the critical paths, so the operating frequency that FIFO achieves has been reduced. Moreover, this design is not suitable for multi-synchronous NoCs where the depth of FIFO is normally less than 10 words.

C. *Other FIFO architectures*

Another design style is building a FIFO based on a ring of stages, where each stage includes a data storage cell, a *'put'* interface and a *'get'* interface. This model is represented firstly by T. Chelcea and S. M. Nowick [8], and then improved by Tark Ono and Mark Greenstreet [9] which only uses typical standard cells instead of requiring some custom ones (pre-charged cells in the original design). The later design has flexible communication protocols. However, for multi-synchronous NoCs with determined communication protocols, this feature is not really necessary. Recently, reconfigurable FIFO structures are also presented for adapting multi- voltage/frequency domains [10].

D. *What changes in our design*

In our design, token-ring and register-based memory are used to build a simple FIFO architecture which can implemented without using any macro block. In addition, AASD is flexible applied in full/empty detection components to improve the FIFO's performance. On one hand, thanks for characteristic of the AASD technique, each pointer contains only one token and does not need to be synchronized. This reduces the number of two-flip-flop-synchronizer and some logic gates used in bubble-encoding algorithm as in Panades' design. On the other hand, the AASD technique lengthens the critical path. Therefore, we modified this technique to reduce the length of these paths. More detail about our design will be described in the flowing section.

III. Proposed Asyncronous FIFO Architecture

Our proposed asynchronous FIFO architecture has five main components as represented in Fig. 1: buffer, write pointer, read pointer, full detector, and empty detector. Write pointer and full detector operate in writing clock domain while read pointer and empty detector operate in reading clock domain. Buffer which is the only storage element of the FIFO can be accessed by both clock domains.

Before going to describe five mentioned components of the proposed FIFO, two special techniques will be introduced. These techniques are used to simplify the mechanism of the FIFO as well as to improve its performance. The first technique is token ring and the second one is the AASD.

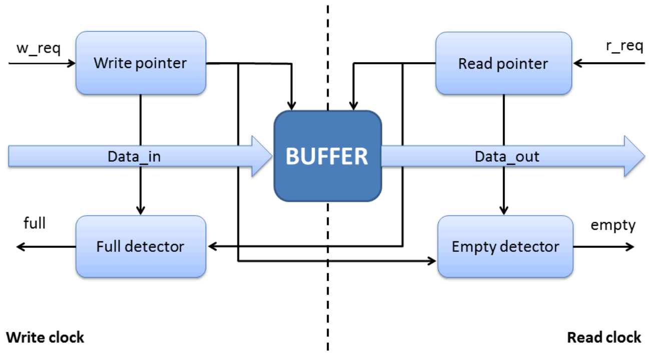

Fig. 1. Block diagram.

A. *Token-ring structure*

Token ring is a succession of registers connected in circular manner like a cyclic register with tokens. A token can be represented by a logic state 1 of one register. A token ring can be used as the state machine in which the states are defined by the position of a token or some adjacent tokens or even other ideas due to target applications. This structure is used to generate pointers in the proposed FIFO architecture.

For example, Fig. 2 demonstrates a token ring which includes only one token. When '*enable*' signal is active, the value of the i^{th} cell is shifted to the $(i+1)^{th}$ cell of the shift register, especially the value of the last cell is shifted to the first cell.

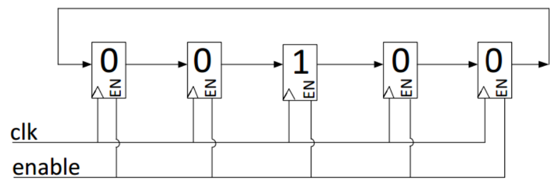

Fig. 2. An example of token-ring structure.

B. *Asynchronous Assertion, Synchronous De-assertion*

This technique is referred as a special case of two-flip-flop-synchronizers (two or more flip-flops). It is used to transfer data from a combinational logic (not clocked) to a clocked domain safely and effectively. If the changing of data is asynchronous to the destination clock domain, the circuit of this technique will behave like an original two-flip-flop-synchronizer to tolerate meta-stability. In this case, data path is the 'dash' line in Fig. 3. On the contrary, if data change synchronously to the destination clock domain, data flow will be the 'dash-dot' line because it does not need to be synchronized anymore.

C. *Buffer & pointers*

The circuit of buffer & pointer components is represented in Fig. 4.

As mentioned above, pointers are created by using token ring in which the token is a shift register containing the logic state '1'. Position of the token is also the position of pointers. Token rings used to implement write pointer and read pointers are clocked by '*w_clk'* and '*r_clk'*, respectively.

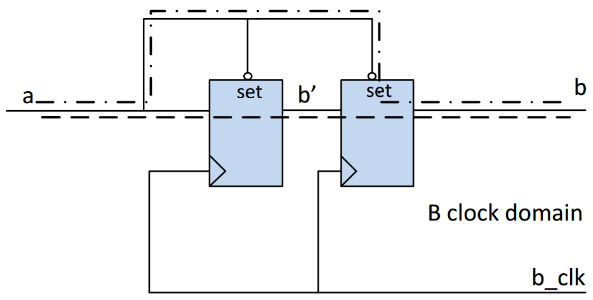

Fig. 3. Asynchronous Assertion, Synchronous De-assertion.

Read pointer is always points to the position of current memory location where data to be read from. Read pointer is shifted to the right when the FIFO is not empty ('*empty*' signal is inactive) and reading operation is required ('*r_req*' signal is active) at the rising edge of '*r_clk*'.

Write pointer always points to the position (the next memory location) where data is being written in. Similarly to read pointer, when the FIFO is not full ('*full*' signal is inactive) and writing operation is required ('*w_req*' signal is active) at the rising edge of '*w_clk*', the write pointer is shifted to the right.

Buffer is the only storage component of the proposed asynchronous FIFO and it can be accessed by both two clock domains as shown in Fig. 4. Because the FIFO depth requirement for multi-synchronous NoCs is not high, buffer is implemented by a sequence of registers being clocked by '*w_clk*' signal. Once writing operation occurs, the input data will be written in one register when its enable signal value is true. On the other hand, the read pointer controls data reading process by using tri-state buffers. The output data are the output of tri-state buffers.

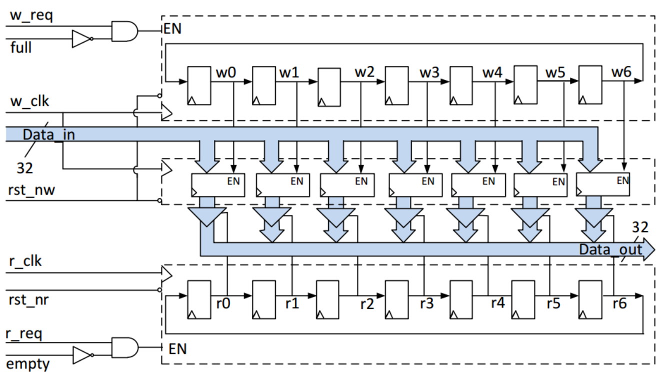

Fig. 4. 7×32 FIFO buffer & pointer components.

The number of registers defines the depth of the FIFO. In our design, the FIFO contains maximum seven 32-bit words which is the minimum depth of FIFO to get 100% throughput.

D. *Full and Empty detector*

The '*full*' or '*empty*' flags indicate that no data should be written in full condition or no data should be read in empty condition as it can lead to loss of data or generation of irrelevant data.

To assert full/empty effectively and safely, a modified asynchronous assertion, synchronous de-assertion technique is proposed. Similarly to AASD technique, if the changing of data is asynchronous to the destination clock domain, the circuit of modified AASD technique will behave like an original two-flip-flop-synchronizer to tolerate meta-stability. However, in the case when data change synchronously to the destination clock domain, data will flows as 'dash-dot' line in Fig. 5. This means the data is delayed one b_clk periods in comparison with AASD technique. Thanks to this modified technique, critical paths which pass through full/empty detector circuit will be shortened. Nevertheless, as the modified AASD technique is applied to full/empty detectors, full and empty flag might be asserted when FIFO just going to be full/empty. This has no danger to FIFO functional, but this will add more delay to full/empty de-assertion.

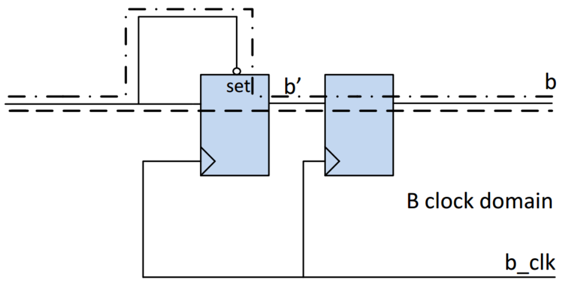

Fig. 5. The modified AASD technique.

When the write pointer and read pointer point to the same position, FIFO is either full or empty. Since the empty detector is correlated to the FIFO throughput, it must be optimized first. Therefore, to distinguish between full and empty states of FIFO without increasing the complexity of circuit, full flag will be asserted when FIFO contains (N-1) elements (N is the depth of FIFO) and empty flag will be asserted when two pointers point to the same position. Position of read and write pointers in full case are shown in Fig. 6.

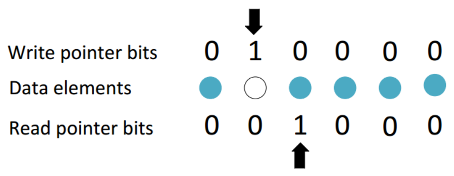

Fig. 6. Pointers' position when FIFO is full.

Fig. 7 represents the full detector circuit. When FIFO is going to be almost full, the write pointer is going to catch the read pointer. This lead to '*full_pre'* signal can only be asserted when write pointer incremented (write pointer shifted right), so the assertion of *'full_pre'* signal is synchronous to the write clock domain. However, the de-assertion of *'full_pre'* takes place when read pointer incremented, which is asynchronous to the write clock domain. As results, in order to improve the efficient of the full assertion flag circuit and optimize the effect to the critical paths, the modified AASD is applied.

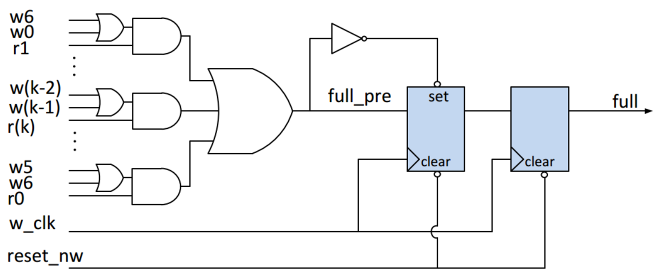

Fig. 7. Full detector circuit.

FIFO is empty when read and write pointers point to the same memory location as presented in Fig. 8.

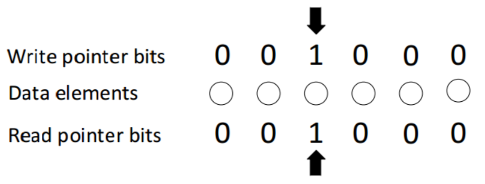

Fig. 8. Pointers' position when FIFO is empty.

The design of Empty detector is presented in Fig. 9. Contrast to the full detector, *'empty_n'* signal can only be asserted when *'r_ptr'* incremented (*'r_ptr'* shifted right), so the assertion of *'empty_n'* signal is synchronous to read clock domain. However, the de-assertion of *'empty_pp0'* takes place when *'w_ptr'* incremented, which is asynchronous to the read clock domain. Similarly to full detector circuit, the modified AASD is also used in empty detector circuit.

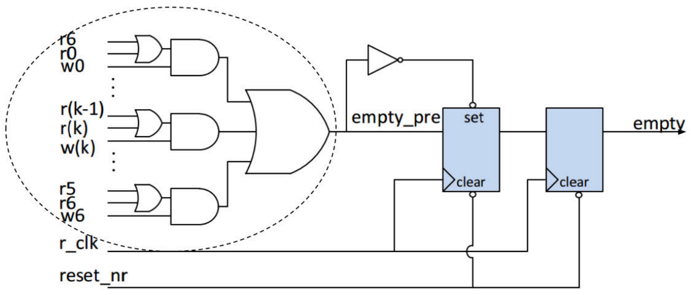

Fig. 9. Empty detector circuit.

E. Reset synchronization

To reset the proposed FIFO, we use an asynchronous reset signal with reset synchronizers which use AASD technique. This type of reset is named Asynchronous reset, synchronous de-reset and is designed to take advantage of both asynchronous reset type and synchronous one.

The *'rst_n'* signal is synchronized to write and read clock domains and is turned into *'rst_nw' and 'rst_nr',* respectively as shown in Fig. 10.

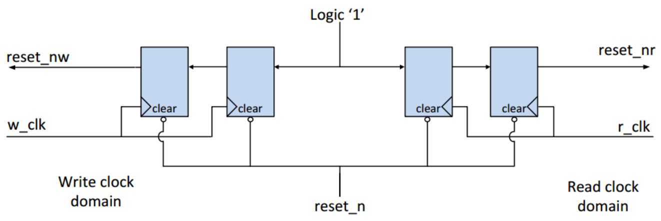

Fig. 10. Reset synchronization.

When *'rst_n'* signal is low-active, *'rst_nw'* and *'Rrst_nr'* is reset to the '0' logic immediately. This causes that write pointer, read pointer, full flag and register-based memory are set to the default status. When *'rst_n'* is de-acitve (has high voltage), it allows the input of the first synchronizer flip-flop which is tied high is clocked through the flip-flop-synchronizer. Therefore, the removal of reset signals in each clock domain will be synchronous to that domain. This eliminates the drawbacks of asynchronous reset from the circuit.

IV. Implementation Result Analysis

After being simulated and verified, the proposed asynchronous FIFO is synthesized on CMOS 180*nm* technology from ASM using Synopsys Design Compiler in worst case condition.

To get 100% throughput as well as minimize the area of design, we implemented an asynchronous 7×32 FIFO. The graph in Fig. 11 gives information about total area and power consumption of this architecture in term of frequency.

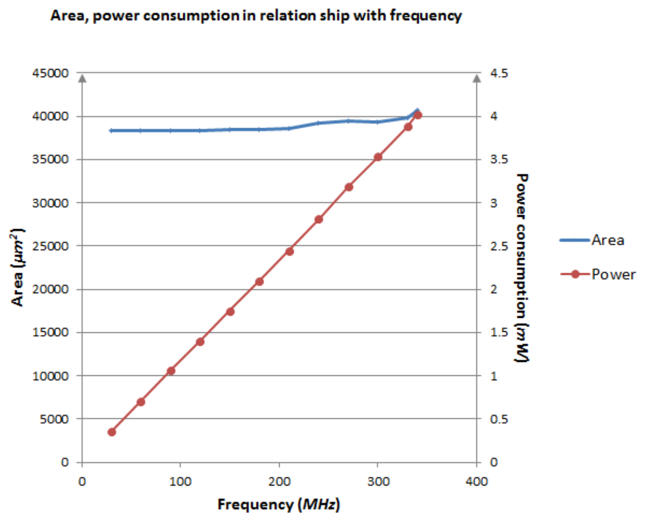

Fig. 11. Area, Power consumption in relationship with frequency.

It is clear from the graph, from 30*MHz* to 340*MHz,* the higher the operating frequency of the design gets, the more power consumption increase. While, as the operating frequency raises, the area slightly fluctuates and the trend is upward.

By doing the gate-level simulation we ensure the final implementation function as intended and determine that the maximum throughput our design can achieve is 10.88*Gbps* at operating frequency of 340*MHz*. At that frequency, the implement area and power consumption it takes are about 40 655.61μm^2 (3601 gates) and 4.025*mW*, respectively.

A. Throughput

From the overview point, throughput of our asynchronous FIFO – is analyzed as function of FIFO's depth. TABLE I shows the comparison between our design and previous works in term of Minimum of FIFO's depth to get the required throughput.

TABLE I. MINIMUM OF FIFO DEPTH IN FUNCTION OF REQUIREMENT THROUGHPUT

Design	**50% throughput (Minimum depth of FIFO)**	**100% throughput (Minimum depth of FIFO)**
Our design	5	7
Panades *et al.* [6]	5	6
Ono *et al.* [9]	3	6
Cummings [7]	3	6

Especially, the firstly condition for asynchronous FIFO get the 100% throughput is that the write and read frequency must be equal.

As indicated in TABLE I, the proposed asynchronous FIFO requires at least 7 memory cells to get 100% throughput while all other considered FIFO styles need just 6 memory cells. However, in order to get 50% throughput, the minimum depth of our FIFO is 4 which bigger than the ones of Ono & Mark Greenstreet and Cummings, but less than the one of Panades.

B. Area analysis

Because of the difference between the technology libraries we used (180*nm*) and the others used to implement (90nm) the considered worked, we compare the area of designs in term of Gate count. Data of our design in four cases are gotten from implementation results at the maximum frequencies design can achieve. In detail, the maximum frequencies 4×16 FIFO, 4×32 FIFO, 8×16 FIFO, and 8×32 FIFO can operate are 390*MHz,* 390*MHz,* 340*MHz* and 340*MHz,* respectively. TABLE II shows that in all case the gate number of our design is the smallest. The other designs are bigger from 0.01% to 44.86% than ours. In both Panades *et al.* design and Cummings' design row, the maximum difference between our design and their ones are in 8×16 columns. In comparison with Ono *et al.* or Cummings' design, the difference of these two designs and our design gets smaller when either FIFO's depth increases or FIFO's width increase.

C. Laten analysis

To make latency analysis be easier, we assume that the delay of data is independent from the reading request and writing request ('*w_req*' and '*r_req*' are always in active status).

The delay of data read from FIFO just after the empty flag is de-asserted (called "First Data") is denoted by T_{d1}. It can be calculated by the following equation.

$$T_{d1} = T_w + 2*T_r + \Delta T \quad (1)$$

Where,

- T_w is the writing period; T_r is the reading period.
- ΔT is the period from the rising edge of writing clock (where a writing operation occurs and makes '*empty_pre*' de-active) to the rising edge of reading clock (where the first flip-flop of two-flip-flop synchronizer gets '*empty_pre*' de-active).

TABLE II. IMPLEMENTATION AREA COMPARISON BETWEEN DIFFERENT DESIGNS

		4×16		**4×32**		**8×16**		**8×32**	
Design	***Technology library***	***Total area (μm^2)***	***Gate count (gates)***	***Total area (μm^2)***	***Gate count (gates)***	***Total area (μm^2)***	***Gate count (gates)***	***Total area (μm^2)***	***Gate count (gates)***
Proposed design	180*nm*	13296	1178 *100%*	24008	2126 *100%*	25385	2248 *100%*	48404	4287 *100%*
Panades *et al.* [6]	90*nm*	3600	1440 *122%*	6511	2604 *122%*	7117	2847 *127%*	12939	5176 *121%*
Ono *et al.* [9]	90*nm*	4266	1706 *145%*	6866	2746 *129%*	7004	2802 *125%*	10906	4288 *100%*
Cummings [7]	90*nm*	3226	1290.4 *110%*	5779	2311.6 *109%*	6237	2494.8 *111%*	11298	4519.2 *105%*

In (1), T_w is the time delay caused by the mechanism of empty detector. Our empty detector architecture is designed to be able to predict the empty status of the FIFO, and it adds one T_w to the value of T_{d1}. Besides, $2*T_r$ delay two-flip-flop synchronizer. ΔT depends on the propagation time of cycled sub-circuit in Fig. 9. Fig. 12 describes an example of T_{d1} delay.

$$0 < \Delta T < T_w + T_r \qquad (2)$$

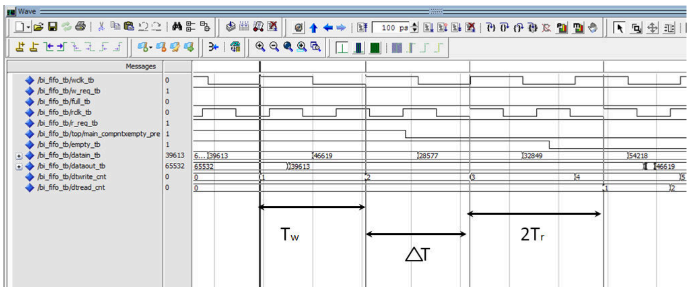

Fig. 12. An example of T_{d1} delay.

The delay of the n-th data, denoted by T_{dn}, is calculated relying on the delay of the "First Data" and is demonstrated in Fig. 13.

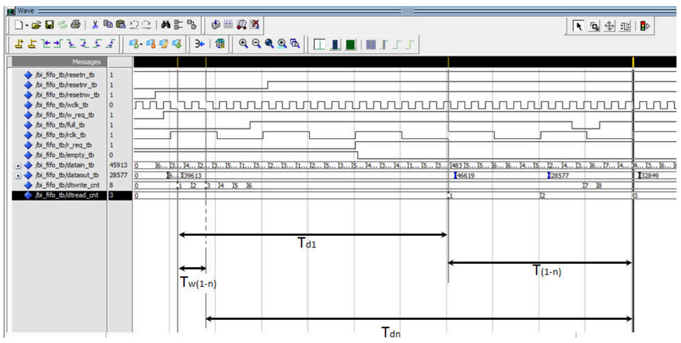

Fig. 13. General data delay.

$$T_{dn} = T_{d1} + T_{r(1\text{-}n)} - T_{w(1\text{-}n)} \qquad (3)$$

- $T_{r(1\text{-}n)} = (n-1)*T_r$ is the period to read (n-1) words.
- $T_{w(1\text{-}n)} = (n-1)*T_w$ is the period to write (n-1) words.
- $n < N$-1, N is the depth of FIFO.

As results, we the general delay form of the data which is read in n-th from FIFO after empty flag de-asserted:

$$T_{dn} = T_{d1} + (n\text{-}1)*(T_r - T_w) \qquad (4)$$

V. Conclusion

In this paper, we presented the design and implementation of an efficient asynchronous FIFO architecture well suited for synchronization unit in multi-synchronous NoCs. Thanks to a set of techniques applied such as: token ring structure, register-based memory, especially the modified AASD, the hardware implementation area of the proposed asynchronous FIFO can be reduced without any significant effect on communication throughput. We have also built simulation environment to check the functionalities of the proposed design. The asynchronous FIFO (7×32) was then synthesized on CMOS 180nm technology from AMS by using Synopsys Design Compiler to find the maximum frequency which it can be work and other information about the implementation. The practice shows that the design can achieve a maximum throughput of 10.88$Gbps$ at the operating frequency of 340MHz in the worst case condition. These obtained simulation results are analyzed and compared with the previous works to demonstrate the efficiency of our design. In next steps, the asynchronous FIFO will be integrated synchronization unit in multi-synchronous NoC platform to be validated in real-time conditions.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2013.17 "Reconfiguration Solution in Designing Network-on-Chip Architectures (ReSoNoC)".

References

[1] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in Proceedings of the 2001 Design Automation Conference, 2001, pp. 684–689.

[2] Tung Nguyen, Duy-Hieu Bui, Hai-Phong Phan, Trong-Trinh Dang, Xuan-Tu Tran, "High-Performance Adaption of ARM Processor into Network-on-Chip Architectures," in Proceedings of the 26th IEEE System-on-Chip Conference (SOCC), pp. 222-227, September 2013.

[3] A. Sheibanyrad, I. Micro Panades, A. Greiner, "Multisynchronous and fully Asynchronous NoCs for GALS architecture," in Design & Test of Computers, IEEE, 2008, pp. 572-580.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[4] Ran Ginosar, "Fourteen Ways to Fool Your Synchronizer," in Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems, 2003, pp. 89-97.

[5] Suk-Jin Kim, Jeong-Gun Lee, Kiseon Kim, "A Parallel Flop Synchronizer for Bridging Asynchronous Clock Domains," in Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits 2004, vol., no., pp.184,187, 4-5 Aug. 2004.

[6] I. Miro Panades, A. Greiner, "Bi-synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-On-Chip in GALS Architecture," in Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip, Princeton, 2007, pp. 83-92.

[7] Clifford E.Cummings, Peter Alfke, "Simulation and Synthesis Technique for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," in Proceedings of the SNUG_2002, 2002, pp. 1-18.

[8] Tiberiu Chelcea, Steven M. Nowick, "Robust Interface for Mixed-Timing Systems," in IEEE Transactions on VLSI Systems, 2004, pp. 857-873.

[9] T. Ono, M. Greenstreet, "A Modular Synchronizing FIFO for NoCs," in Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp. 224-233.

[10] Amir-Mohammad Rahmani, Pasi Liljeberg, Juha Plosila, Hannu Tenhunen, "Design and Implementation of Reconfigurable FIFOs for Voltage/Frequeny Island-based Networks-on-Chip", Microprocessors and Microsystems, Vol. 37, Issues 4-5, June-July 2013.