International Journal of Engineering Innovations in Advanced Technology
ISSN: 2582-1431 (Online), Volume-4 Issue-4, December 2022

Deign Of Multi Data Functional Based Fipflop For High-Speed Data Communication System

Vemuganti Laxmi Prasanna
MTech Student, Department Of ECE, VLSI System Design, Avanthi Institute of Engg. & Tech., India.
Email: prasannavemuganti@gmail.com

Dr.S. Kishore Reddy
Associate professor, HOD, Department Of ECE, VLSI System Design, Avanthi Institute of Engg. & Tech., India.
Email: kishorereddy416@gmail.com

D Suryaprakesh
Assistant professor, Department Of ECE, VLSI System Design, Avanthi Institute of Engg. & Tech., India.
Email: kishorereddy416@gmail.com

***Abstract*—To increase system composability and facilitate timing closure, fully synchronous clocking is replaced by more relaxed clocking schemes, such as mesochromous clocking. Under this regime, the modules at the two ends of a monochromous interface receive the same clock signal, thus operating under the same clock frequency, but the edges of the arriving clock signals may exhibit an unknown phase relationship. In such cases, clock synchronization is needed when sending data across modules. In this brief, we present a novel mesochromous dual-clock first-input– first-output (FIFO) buffer that can handle both clock synchronization and temporary data storage, by synchronizing data implicitly through the explicit synchronization of only the flow-control signals. The proposed design can operate correctly even when the transmitter and the receiver are separated by a long link whose delay cannot fit within the target operating frequency. In such scenarios, the proposed mesochromous FIFO can be extended to support multicycle link delays in a modular manner and with minimal modifications to the baseline architecture. When compared with the other state-of-the-art dual-clock mesochromous FIFO designs, the new architecture is demonstrated to yield a substantially lower cost implementation**

***Key Words*- synchronization, FIFO, clock signals, mesochromous**

Manuscript received Oct 10, 2022; Revised Oct 25, 2022; Accepted Nov 4, 2022

I. INTRODUCTION

Ever shrinking transistor sizes have enabled the integration of a greater number of components onto a single chip—thus making systems-on-a-chip (SoCs) with many complex modules a common design solution. Unfortunately, global interconnect scaling has not been able to maintain the same performance increases [1], causing the timing of high-speed global clock signals to become a major concern in system design. This has resulted in clock distribution circuits requiring increasing circuit resources and design time. Nearly all existing digital systems utilize synchronous design techniques which normally require an accurate and highly synchronized global clock reference to be supplied to all areas of the circuit. One solution for coping with the clock distribution problem is to utilize self-timed or asynchronous circuits, which do not have a global timing reference signal. However, the lack of mature design tools and the reluctance of industry to incur the cost and risk of moving away from successful synchronous design flows have limited the acceptance of these design styles [2]. An alternative approach is to create systems that mix asynchronous and synchronous design techniques using a globally asynchronous locally synchronous (GALS) [3] design approach. In this paradigm, blocks are built using traditional synchronous design techniques, but these synchronous blocks do not share global timing

information and are asynchronous with respect to each other. While it is often convenient to divide a system into multiple subcomponents, it is unlikely that these components will operate autonomously. Accordingly, data transfer is required between local synchronous blocks. Accomplishing this task reliably and efficiently are key challenges in GALS designs

II. LITERATURE SURVEY

Dally and Poulton [4] and Balch [5] present high-level views of dual-clock FIFO structures, but details of dual-clock FIFO designs are lacking in the literature. Fully asynchronous FIFOs often appear in the literature [6], [7], but these designs do not utilize clocks, and therefore, are difficult to apply in cases of synchronizing da ta between clock domains. Table I lists several dual-clock FIFO designs. In the work presented by Greenstreet [8], the clocks are derived from the same base frequency, but may have an arbitrary phase difference—which is slightly more general than strict mesochronous The FIFO designed by Chakraborty et al. requires time to develop a frequency difference estimate before transferring data, as well as usage of different circuits depending on which clock domain has the higher rate [9]. Siezovic [10] presents a linear FIFO architecture for data synchronization, which has the limitations presented in Section II-A. An alternative FIFO architecture for use in some dual-clock applications is presented by Chelcea and Nowick [11]. The design uses independent registers as storage elements, and each register has its own and signals. This scheme reduces the latency when the FIFO size is small, but is less suitable when the FIFO size is large. This work uses a dual-port SRAM as the storage element which increases memory density and improves FIFO size scalability [13]. Compared with the most similar previous work [12], this design includes configurable logic to make it suitable for many environments, and also enables complete oscillator halting during idle times to achieve high energy efficiency. The proposed FIFO design has been fabricated in what we believe is the first VLSI implementation of a GALS array processor.

III.EXISTING SYSTEM

To best address dual-clock FIFO issues, we first consider the case of a single-clock synchronous FIFO. This section covers these fundamental FIFO principles.

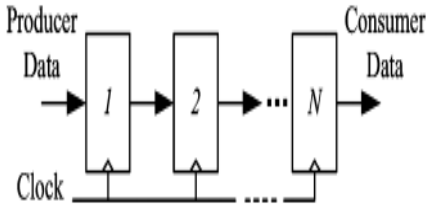

Figure 1. Linear shift-register FIFO block diagram.

A. *Linear FIFOs*

The simplest FIFO structure consists of a linear chain of latches or flip-flops connected serially as a shift register. Data is shifted into one end of the chain and propagates through every memory element until it reaches the end as shown in Fig. 1. This FIFO is synchronous since all movement of data requires a common clock. Alternatively, a linear elastic FIFO uses control signal handshakes to propagate data from location to location. Unlike the synchronous case, a datum can propagate through the FIFO without any new items entering. This results in the FIFO being at various degrees of fullness, hence, the name elastic. FIFOs of this nature work well with asynchronous designs and many examples of these can be found in the literature [15], [16]. A simple example of this type of FIFO is shown in Fig. 2.

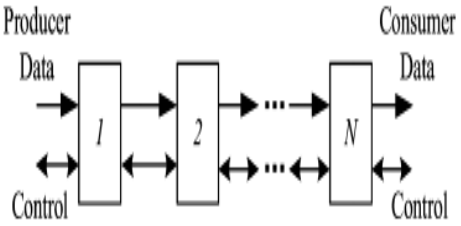

Figure 2. Linear elastic FIFO block diagram.

IV.PROPOSED SYSTEM

Modern systems-on-chip (SoC) implemented in deeply scaled technologies faces slow wires and process/voltage/temperature (PVT) variations. These challenges make the synchronous abstraction increasingly untenable over large chip areas, thereby requiring immense design effort to achieve timing closure [1], [2]. Partitioning the SoC into globally asynchronous, locally synchronous domains [3], [4] partially alleviate the problem, since synchronous operation and its associated timing constraints are confined inside each domain. However, in this case,

when crossing clock domains, the signals must be synchronized to the receiving clock domain, in order to avoid metastability [5], [6].

In addition to delivering synchronized signals across the clock domain interface, it is also important to ensure that any synchronized data that cannot be immediately consumed by the receiving domain are safely stored until it can be serviced. Since data must both be synchronized and (temporarily) stored, it is imperative that these two elemental and intertwined operations—synchronization and buffering—are combined in a cost-effective way that minimizes any latency and area overhead. In this brief, we focus on mesochromous clock domains, where clocks operate under the same frequency, but with a fixed, arbitrary phase difference. In such cases, using a generic asynchronous dual-clock first-input–first-output (FIFO) [7] for mesochromous clock domain, crossing is possible, but incurs unnecessary latency overhead [8]. Currently, there are two major approaches for efficient synchronization and buffering across mesochronic interfaces: 1) in a loosely coupled implementation [8], synchronization and buffering occur separately while 2) in a tightly coupled implementation [9]–[11], they are combined and fused into a single structure.

A. Proposed Mesochromous Fifo

The proposed mesochromous FIFO architecture combines the benefits of the loosely coupled [8] and tightly coupled approaches [9], [10], while avoiding their weaknesses. The new design couples synchronization and buffering in a cost-efficient implementation that fully supports multicycle link delays. A completely different operating approach is adopted, whereby the data are synchronized implicitly through the explicit synchronization of flow-control signals.

B. Architecture and Operation

Fig.1 shows the proposed mesochromous FIFO. Data that need to be synchronized are stored in a memory placed in the transmitter domain. Two monotonically increasing counters index the memory positions, where data are stored and accessed. The transmitter-synchronous tail pointer points to the write position in memory where a new data word is stored, while the receiver-synchronous head pointer points to the position from where a word will be read out. A pair of opposite-direction single-bit 4-flop synchronizers are used to sync enqueue (write) and dequeue (consume) events between the two sides. The 4-flop synchronizers are used to account for the worst case scenario that can occur with the asynchronous reset of the read and write pointers, as described in [8]. When the transmitter sends a data word that needs to be synchronized, it writes the data into the memory position pointed by the tail pointer. At the same time, the tail pointer is increased, and the push signal that is fed to the forward "tx2rx" mesochromous synchronizer [see Fig. 4(a)] is asserted. When the enqueue event is synchronized across the interface, the receiver can safely read out the data from the memory position pointed by the head pointer. This operation is shown in the short example of Fig. 4(b), which depicts the transfer of three data words ("A," "B," and "C") from TX to RX. Once the receiver actually consumes the data (e.g., reads it and sends it downstream), the pop signal is asserted, and the head pointer is incremented to point to the position where the next data word is found. Note that the receiver does not try to read data from the updated head pointer position, unless a new push event has been synchronized, indicating that new data exist and are safe to be read out.

In order to not lose track of multiple enqueue events, the receiver employs a "status" counter that counts the number of synchronized data items currently in the queue. Whenever a new data word is received, as indicated by an incoming enqueue signal from the "tx2rx" synchronizer in Fig. , the counter is increased; when a data word is consumed, the counter is decreased to reflect the change in the queue's state. In this way, two key objectives are achieved: 1) data are implicitly synchronized through the explicit synchronization of the enqueue events and 2) the FIFO order is preserved in the buffer. The next step is to synchronize the queue's state to the transmitter domain and guarantee that the queue does not overflow. To achieve this, the transmitter also uses a status counter, as shown in Fig. 4(a), to keep its own version of the number of items currently stored in the queue. The counter is incremented, or decremented, whenever an item is enqueued, or dequeued, from the queue, respectively. Since dequeue (pop) events are receiver-synchronous, they have to be synchronized to the transmitter domain through a separate backward synchronizer. On a dequeue, the receiver asserts the pop signal of the "rx2tx" synchronizer. Once the signal is synchronized, the transmitter decreases its status counter, effectively remaining in sync with the downstream buffer's state.

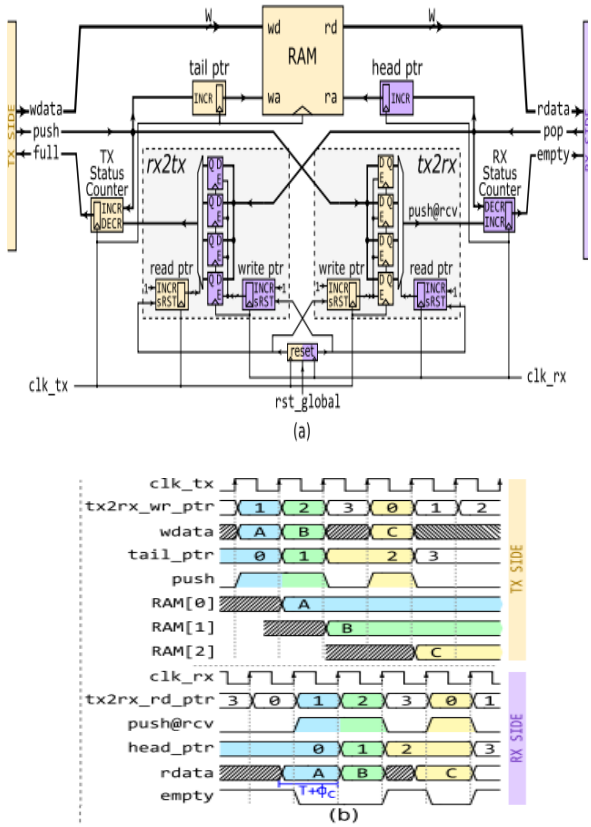

Figure 3. (a) Proposed mesochromous FIFO, which implicitly synchronizes data through the explicit synchronization of the flow-control push/pop signals. (b) Short cycle-by-cycle example of the operation of the proposed design.

Similar to [8], the forward latency of synchronizing and enqueuing a new data item is between one and three cycles, depending on he spread between the read and write pointers after reset. For safe operation under any phase difference, the initial spread between the read and write pointers at each synchronizer is two. When the reset of the read pointer of the push synchronizer is delayed (due to metastability), the forward latency is increased by one cycle. On the contrary, if the reset of the write pointer is delayed, the forward latency is decreased. In the case that the reset signal is not delayed, or delayed on both sides, the forward latency remains unchanged. The backward latency, i.e., the number of cycles needed to synchronize the pop events, is also between one and three cycles. However, worst case forward and backward latencies cannot occur simultaneously. The read pointer of the push synchronizer is driven by the same reset signal (the rx-side-synchronized version of the asynchronous reset) with the write pointer of the pop synchronizer. The same happens with the write and read pointers that are driven by the reset signal synchronized to the tx side. Therefore, once one side experiences a latency of three cycles (delayed reset of the read pointer), the other side will experience a latency of one cycle (delayed reset of the write pointer). Overall, the sum of the forward and backward latencies is constant at four cycles.

V.CONCLUSION

Irrespective of the physical proximity of the sender and the receiver in a mesochromous clock interface, the proposed low-cost dual-clock FIFO combines mesochromous clock synchronization and buffering in a scalable manner. Data are safely transferred on the receiver side of a mesochromous interface without being explicitly synchronized. Synchronization involves only the single-bit push/pop flow-control signals. This implicit synchronization of data saves considerable amount of area/power, especially in the case of multicycle links, without introducing additional latency, or reducing through.

References

[1]. Burchardt, A., Hekstra-Nowacka, E., and Chauhan, A.,“A Real-time Streaming Memory Controller”, Design, Automation and Test in Europe 2005 Proceedings, 2005, vol.3, pp. 20-25.

[2]. Heithecker, S. and Ernst, R., “Traffic Shaping for An FPGA Based SDRAM Controller with Complex QoSRequirements”, Design Automation Conference 2005, June 2005, pp. 575-578.

[3]. ANALOG DEVICES, AD9480 datasheet, Rev. A, 2004, 7.

[4]. ALTERA, DDR SDRAM Controller White Paper, Ver1.1, 2002, 8.

[5]. Guo Li, Zhang Ying, Li Ning, and Guo Yang, “The Feature of DDR SDRAM and the Implementation of DDR SDRAM Controllers via VHDL”, The Journal of China Universities of Posts and Telecommunications, 2002, vol.9,no. 1, pp. 61-65.

[6]. Digital Systems Design Using VHDL by Charles H. Roth, Jr

[7]. CMOS Designing by Kanzg

[8]. Basic Digital Design by Morris Mano

[9].http://www.latticesemi.com/products/intellectualproperty/referencedesigns/ddrsdram controller.cfm