# An FIFO Memory Design for 8-to-32 Data Exchange Bus [§]

Chua-Chin Wang, Yih-Long Tseng, and Yi-Wei Chen

Dept. of Electrical Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan 80424
email : ccwang@ee.nsysu.edu.tw

## Abstract

An FIFO memory architecture is proposed to be utilized in data exchange between processing units which possess non-homogeneous bus widths. Neither arbiter logics nor modules are required in such a design to determine input sequences or output sequences. Hence, the delay is drastically shortened. Two pointers, which are read pointer (RP) and write pointer (WP), respectively, point to the head and the tail of the valid data queue in the FIFO. The simulation results of the proposed design which is implemented by Verilog HDL (hardware description language) reveal that the design is capable of processing the data under a 200 MHz clock rate using TSMC 0.35 1P4M CMOS technology.

**Key Words:** FIFO, data exchange, nonhomogeneous bus width, arbiter

## 1. Introduction

The demand of high-speed and reliable data transfer between two devices is very critical in multi-processor systems and communication systems, [2], [3], [6]. FIFOs, thus, become a non-redundant module in such data transfer scenarios. Reliable FIFO operations guarantee the removal of the bottleneck, and the resilience to error scenarios [8]. Many efforts has been thrown on the performance improvement of either synchronous or asynchronous FIFOs [4], [5]. A long ignored question is what if the data widths are different on both sides of the FIFOs. That is, the bus width is non-homogeneous. An immediate solution to such a problem is to add encoder-decoder-like (codec-like) modules at two sides of the FIFO. The codecs performs either a serial-to-parallel format transformation for a short-data stream to a long-data stream, or a parallel-to-serial format transformation the other way around. For instance, one device uses a byte-wide I/O port, while another device uses a double-word-wide (DW-wide) I/O port. The FIFO between these two devices must possess a certain arbiter module to determine and control the bidirectional data flow [1], [4], [6], [7]. Unavoidably, the arbiter module itself becomes

§This research was partially supported by National Science Council under grant NSC 89-2215-E- 110-014 and 89-2215-E-110-015.

†the contact author

the bottleneck of the entire operation. In this paper, we present a FIFO design which is capable of processing the data exchange between a byte-wide device and a DW-wide device. There is no arbiter in the proposed design such that the delay of the data format transformation is eliminated. Two pointers, read pointer and write pointer, are employed to cope with the determination of the data I/O sequencing.

## 2. Non-homogeneous Bus Width FIFO

Referring to Fig. 1 [3], a typical FIFO design is illustrated. A slow arbiter is required to determine the read and write sequences. Notably, the data widths of the two sides of the FIFO is identical.

### 2.1. Architecture of the proposed FIFO

It is obvious that the drawbacks of prior FIFO works are lack of flexibility of data widths, and slow due to the arbiter operation. We propose a design in Fig. 2 to resolve these difficulties. A two-port 256×8 RAM module is the core of the FIFO storage. Each address is associated with one “VALID” bit to indicate whether the corresponding byte is good for any future read operations. The two-port RAM cell is capable of simultaneous reading and writing given certain conditions. Referring to Fig. 3, the *i*th RAM cell is allowed to be written provided that FULL = 0, and $WP_i$ is activated. Note that $WP_i$ is the output *i* of the WR_decoder given WP[7:0] as the input.  By contrast, the data storage of the *i*th RAM cell is readout when $RP_i$ is activated, and $VALID_i = 1$. Similarly, $RP_i$ is the output *i* of the RD_decoder given RP[7:0] as the input. It should be noted that the data of the RAM cells are kept being readout without any triggering actions of a clock signal. Not only will it drastically reduce the delay, the hold time and setup time violations are also avoided. The details of the operations of the proposed design are given as follows.

**Write Operation :** Notably, the WP[7:0] contains the current address of the FIFO for the buffered-in byte . It is called the write pointer (WP). The flow of the write operation is shown as Fig. 4.

(W1). As soon as the $\overline{\text{Write\_request}}$ is raised, the contents of WP is buffered into the Full Detector to determine whether RP = (WP + 2) mod 256. At the same time, the WP is also fed into the WR_Decoder to pre-decode the address to be written.

(W2). If RP = (WP + 2) mod 256 is true, the FULL flag is set and then an error message is delivered to the sending device to indicate an overflow error. Note that the write operation of the data byte keeps running without any hesitation. However, its corresponding valid bit is marked reset, which means invalid.

(W3). If RP = (WP + 2) mod 256 is false, the data is written into the address which WP indicates. In the meantime, its corresponding valid bit is set.

The safe margin between RP and WP is

chosen to 2 instead of 1 in the above flow is due to the fact that the data placed on the bus won't be processed until the next working clock edge, regardless of positive edge or negative edge. If “1” is used, a read operation and a write operation might be invoked to one single cell.

**Read Operation :** It is noted that the RP[7:0] contains the starting address of the FIFO for the buffered-out double word . It is called the read pointer (RP). The flow of the read operation is shown as Fig. 5.

(R1). RP[7:0] is concatenated by the output of the Read Pointer module, which is 6-bit long of RP[7:2], and two padding ``0''s at RP[1:0]. The reason is that the format of the output is double word, i.e., 32 bits. Hence, the contents of RP[7:2] is the aligned 4-bit boundary in the RAM module.

(R2). The Output Buffer always contains the data bytes in (RP), (RP+1), (RP+2), (RP+3). They, however, will not be placed on the output bus until all of their corresponding valid bits are verified to be set by the Valid Decider module.

(R3). As soon as the $\overline{\text{Read\_request}}$ is sensed, the contents of the RP is sent to the Valid Decider along with two padding zeros at the two LSB bits. Meanwhile, it is fed into the RD_Decoder to pre-decode the bytes to be read (or deleted from the FIFO). The selected 4 bytes are fetched to the Output Buffer waiting for the permission from the Valid Decider.

(R4). The Valid Decider checks all of the valid bits of the consecutive 4 bytes starting from where RP points to. If there are all valid, the double word in the Output Buffer is placed on the output bus.

## 2.2. Performance analysis

The features of the proposed design compared to prior FIFO designs are summarized as follows.

a). Prior FIFOs are not designed for non-homogeneous data width. Hence, the long data must be divided into small partitions which then are either written or readout. The complexity of such a design will be *O(n)*, where *n* is the ratio of the long data to the short data. On the contrary, the proposed design is independent of the data length, which implies that complexity is *O(1)*.

b). No slow arbiter is required in our design. The arbiter in prior FIFO works is basically used to prevent the write and read operations occurring at the same time. It intrinsically contains a clock-triggered FSM (finite state machine), which generally is a slow circuit.

## 3. Performance Simulations

In order to verify the performance of the proposed FIFO, the entire design is carried by

Verilog RTL codes, and then synthesized by SYNOPSYS using TSMC 0.35 1P4M cell library. Fig. 6 is the normal read and write operations of the proposed FIFO. Detailed descriptions of the notations are given as follows.

(N1). Byte 1, 2, 3, and 4 are validated at the end of clock 5. Hence, a valid double word is placed on the output bus.

(N2). During clock period 7, though the $\overline{\text{Read\_request}}$ is pulled low, the double word is not delivered owing to the valid bit of the highest byte is not set.

(N3). At the end of clock period “a”, byte 5, 6, 7, and 8 are all validated to be sent out.

(N4). Byte 9, a, b, and c are similarly manipulated.

(N5). The $\overline{\text{Write\_request}}$ is kept low to continuously feed data bytes to the FIFO.

By contrast, Fig. 7 is the scenario that the FULL flag is set which is highlighted as “F1”. Notably, any further write operation is prohibited unless there is a readout operation, which is “F2” in Fig. 7. The maximal I/O clock rate is 208 MHz (≈ 200MHz), which implies that the throughput is 6.4 Gbps = 32 bits × 200 MHz. The critical delay of the proposed design is 4.36 ns. The overall synthesized gate count of the proposed FIFO is 29494.0. Fig. 8 reveals the dual part of the proposed design.

## 4. Conclusion

We present a novel FIFO design which is capable of processing the data transfer between two devices with different data lengths without using any arbiter. The readout operation and the write operation are executed simultaneously. A minimum number of flags is required. The simulation results turn out to be very appealing.

## Reference

[1] A. Bystrov, and A. Yakovlev, Ordered arbiters, “*Electronics Letters*,” vol. 35, no. 11, May 1999.

[2] S. K. Das, M. C. Pinotti, and F. Sarkar, “Optimal and load balanced mapping of parallel priority queues in hypercubes,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 6, June 1996.

[3] M. Hashimoto, M. Nomura, K. Sasaki, K. Komatsizaki, H. Fujiwara, T. Honzawa, K. Abe, T. Tachibana, N. Kitagawa, “A 20-ns 256K×4 FIFO memory,” *IEEE J. of Solid-State Circuits*, vol. 23, no. 2, pp. 490-499, Apr. 1988.

[4] J. Jex, P. Nag, T. Burton, and R. Mooney, “Split FIFO phase synchronization for high speed interconnect,” *1995 IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, pp 66-69, 1995.

[5] M. Muegge, and D. Chenoweth, “36 bits

wide FIFO for deep, bus oriented applications," Southcon/93, pp. 615-619, 1993.

[6] D. Picker, and R. D. Fellman, "A VLSI priority packet queue eith inheritance and overwrite," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 2, pp. 245-252, June 1995.

[7] G. V. Russo, and M. Russo, "A novel class of sorting networks," *IEEE Trans. on Cricuits and Systems - 1 : Fundamental Theory and Applications*, vol. 43, no. 7, pp. 544-552, July 1996.

[8] D. Wyland, "New features in synchronous FIFOs," *Northcon/93*, pp. 224-232, 1993.

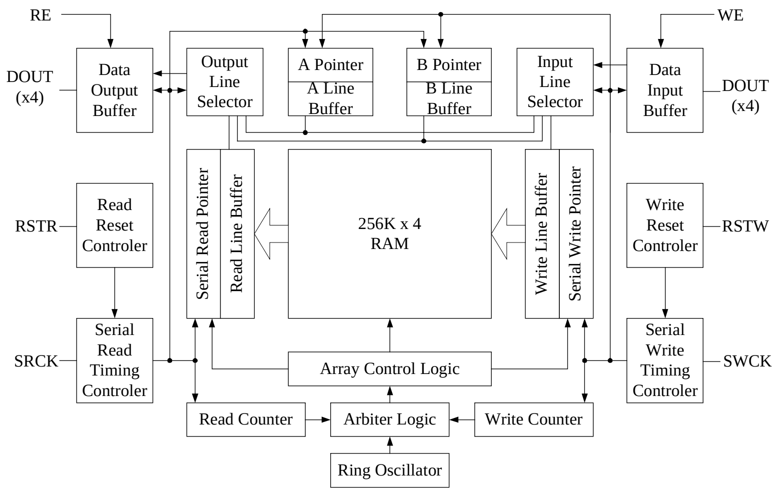

Figure 1: Coventional FIFO configuration

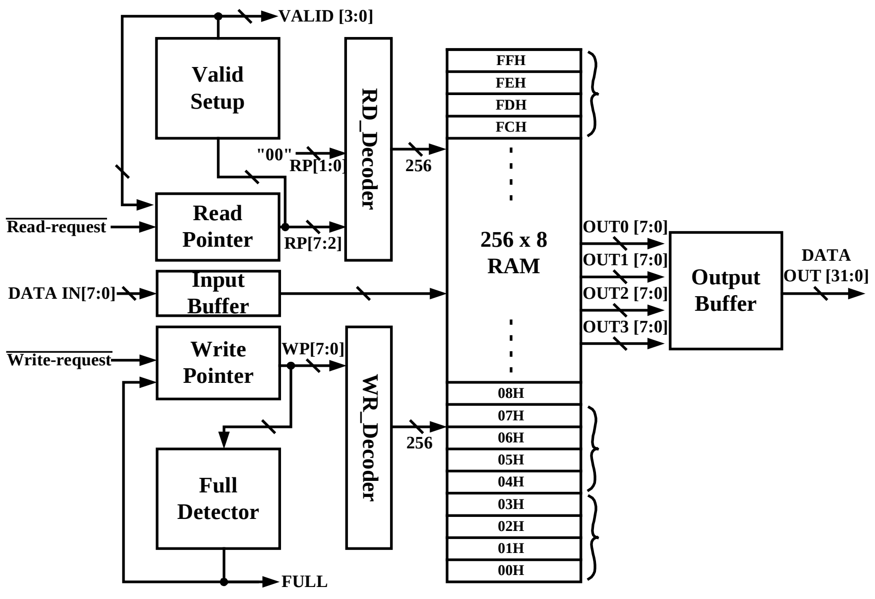

Figure 2: Proposed FIFO architecture

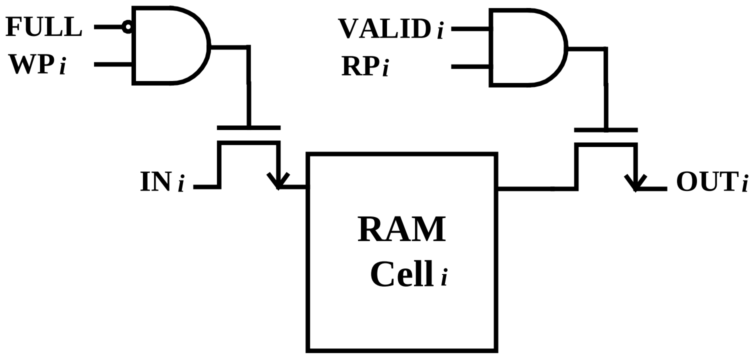

Figure 3: Two-port memory cell and control signals

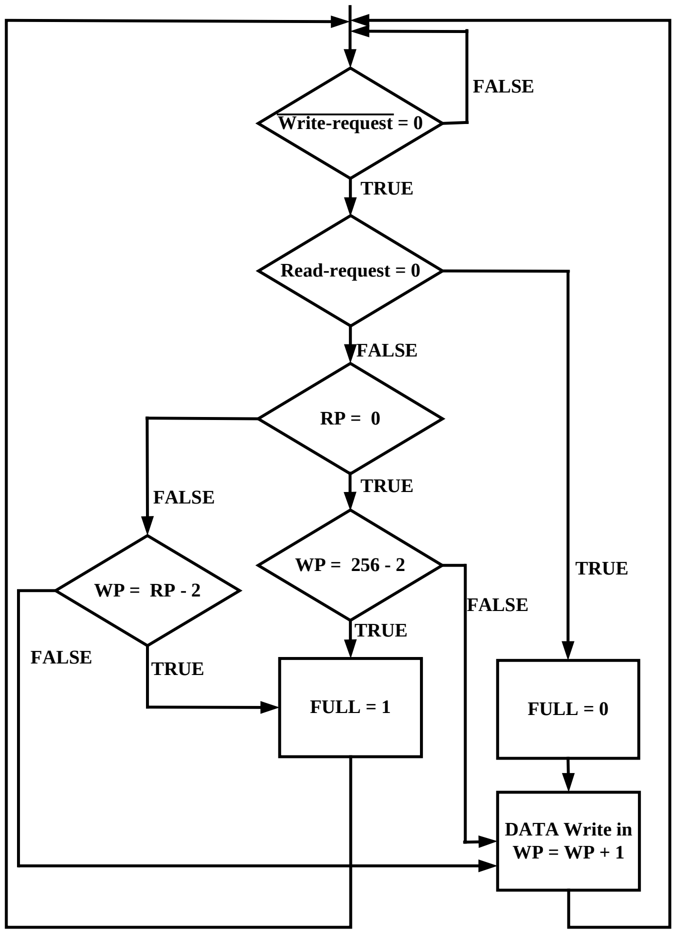

Figure 4: Write operation data flow

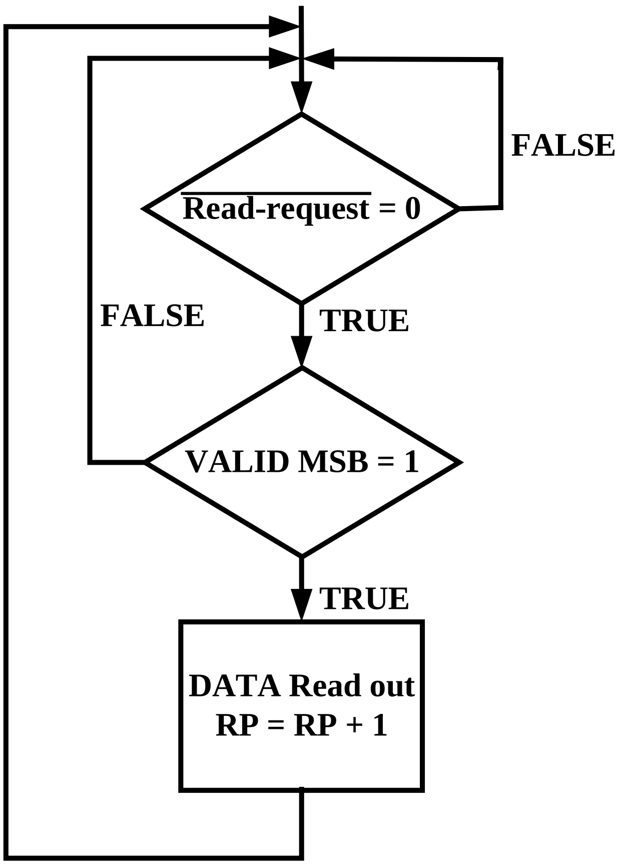

Figure 5: Read operation data flow

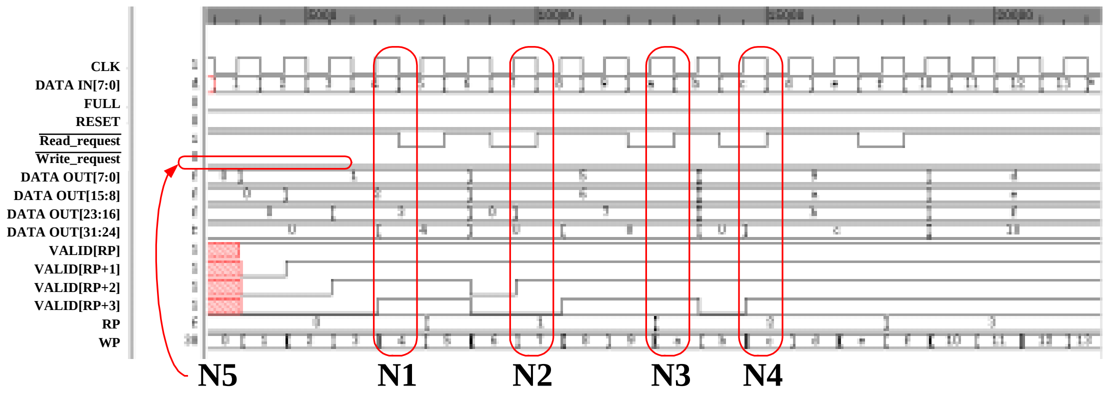

Figure 6: Simulations of the proposed FIFO in a normal status

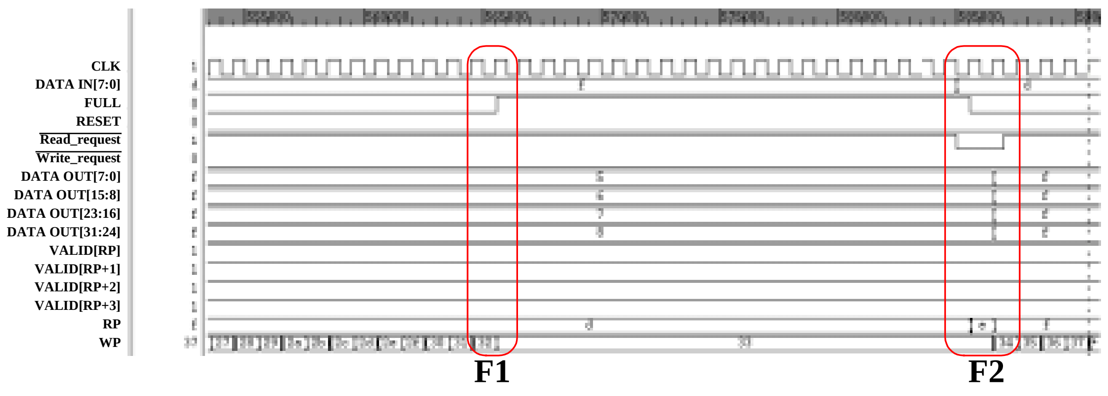

Figure 7: Simulations of the proposed FIFO with a FULL error

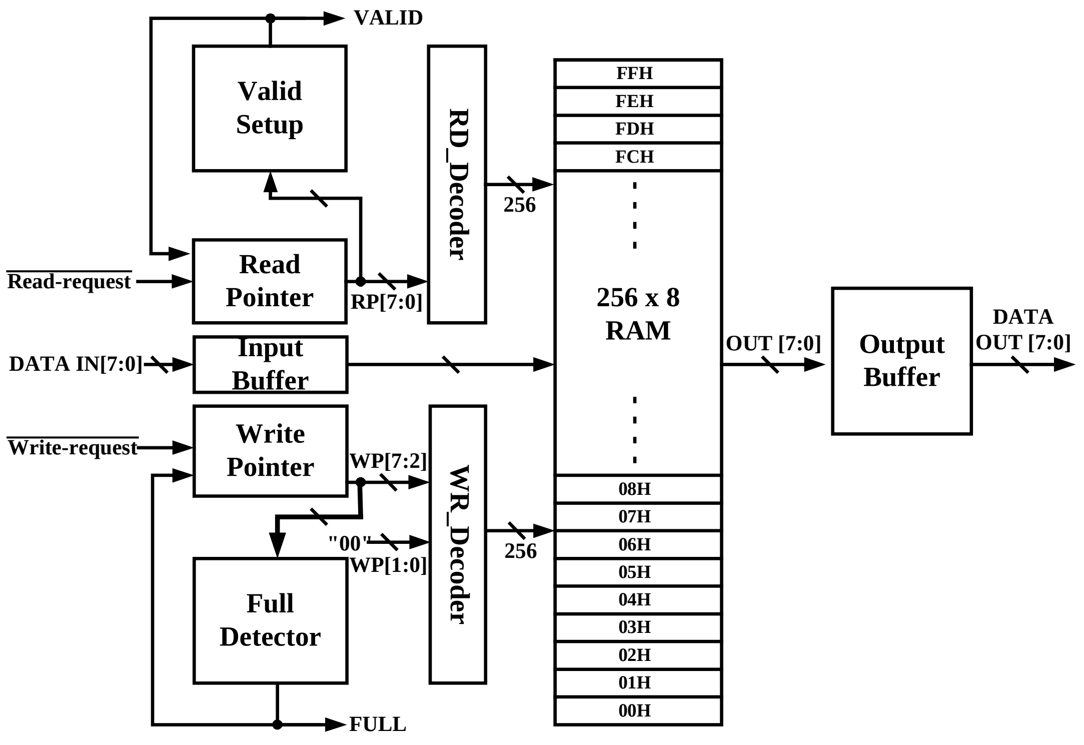

Figure 8: Proposed 32-to-8 FIFO design