Arabian Journal for Science and Engineering
https://doi.org/10.1007/s13369-021-05550-6

TECHNICAL NOTE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

A Bit Addressable Register with Variable Write/Read Data widths

Aiman El-Maleh[1] · **Saleh AlSaleh**[1] · **Muhammad E. S. Elrabaa**[1]

Received: 11 November 2020 / Accepted: 9 March 2021

Keywords Variable data width buffer · FIFO · FPGA

1 Introduction

Many applications nowadays offload their compute intensive kernels to FPGAs. Unlike CPUs, FPGAs can support fine grain parallelism at any data size (i.e. length). Hence, many applications such as compression [1,2] and deep learning networks [3] use/generate data with variable size. Such data has to be packed into standard lengths (e.g. 32-bit words) supported by the host CPU. For example, hardware compression implementations such as WaveSZ and GhostSZ [4,5] require their inputs and outputs to be packed/unpacked for compression/decompression. This requires the design of a queue or register that allows writing and reading of variable number of bits (hence it can be used to compact fragments of data into standard length words). FIFOs (first-in-first-out) are data storage buffers used to facilitate bursty communication between a data producer and a data consumer. It allows the producer and consumer to store/retrieve data in the same order. The producer does not need to wait for the previous data to be consumed unless the FIFO is full. Thus, the FIFO size is usually chosen to accommodate one or more bursts of data. Existing FIFO designs support equal, different (e.g. [6]), or multiple input/output data lengths (e.g. [7]). For all these designs, however, the data lengths are fixed at design time. They cannot support operation with variable length data. As will be shown later, one way to implement variable length data buffers would be to use serialized writing of data to a buffer which would require numerous cycles. Alternatively, one could use conventional SRAM-based FIFO buffer with meta data (indicating the length of each data word) stored alongside each data item. For example, the FIFO word size would be the maximum data Size + LOG (Size). Not only does this increases the SRAM's size, but it does not provide the required bit compaction.

In this paper, we propose a design of a bit addressable register (BAR) that allows writing and reading a variable number of bits (sizes are selectable at runtime), i.e. the size of each word pushed into the buffer could differ from one push to another and is specified at each push operation. Similarly, the size of each word being popped from the buffer could differ from one pop operation to the other. The proposed design is modeled as an RTL synthesizable module in Verilog. The register size (total number of bits) and input/output ports sizes (representing the maximum data length) are all parametrized.

Hence the contribution of this paper is as follows:

1. A new FIFO design that can support dynamic (i.e. at runtime) variable length data writing and reading. Not only are the lengths of written/read data can be independently specified at run time, the lengths of each written/read data can be specified at run time independently from previously written/read data, thus allowing fragments of variable length data to be compacted into words with dynamically selectable widths,
2. An optimized parametrized Verilog description is developed that can be synthesized for any technology (ASICs or FPGAs),
3. Comparison with a serialized buffer design in terms of speed and resource utilization was carried out.

2 Proposed BAR Design

The general structure of the proposed BAR follows a typical FIFO structure where the memory elements (FFs in this case instead of SRAM rows) are organized in such a way to yield a circular buffer with pointers to the tail/head bits/FFs of

✉ Aiman El-Maleh
aimane@kfupm.edu.sa

Saleh AlSaleh
salehs@kfupm.edu.sa

Muhammad E. S. Elrabaa
elrabaa@kfupm.edu.sa

[1] King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Published online: 10 April 2021

the data in the buffer. The tail pointer represents the FF in the BAR where new data can be written in the FFs below it. Similarly, the head pointer represents the FF where a number of data bits can be read from all the way to the tail. Figure 1(a) shows the general signal interface of the BAR. It should be noted that even though the physical interface (i.e. the ports' sizes) is fixed at design time, the lengths of the actual data being written/read during operation of the BAR can vary in length (but obviously cannot exceed the width of the interface ports' sizes).

The BAR has three design parameters; ***k***, the total size/capacity of the buffer in bits ((i.e. number of FFs in the BAR), ***n***, the width of the write port (represents the maximum word size that can be written/pushed into the BAR), and ***m***, the width of the read port (represents the maximum word size that can be read or popped from the BAR). Figure 1(b) shows the internal structure of a BAR design example (an 8-bit BAR with 4-bit data ports, i.e. k=8, n=m=4). Not shown in the figure are three counters that hold the number of bits in the BAR, and pointers to the head/tail bits. As this figure shows, the BAR is built upon a simple concept; have each flip-flop's input selectable from one of the n-bit *Wdata* port wires using *k* n-to-1 MUXs. Similarly, each one of the m-bit *Rdata* output wires is selectable from one of the *k* flip-flops outputs (i.e. the BAR has an m-bit wide, k-to-1 MUX). The values of the FFs enable signals and the selection controls for the MUXs are determined based on the values of Tail, Head, *wsz*, *rsz*, and *k*. During a write operation, if there are at least *wsz* empty FFs in the BAR, a combinational circuit enables FFs from Tail up to to Tail + *wsz* - 1 (modulo *k*, for looping back if Tail+*wsz* $\geq k$). This circuit basically converts the Tail counter value + *wsz* to a thermometer code (all FF enables are zeros except the *wsz* bits equal to and above the Tail counter value). For the FFs input MUXs, the selection lines are simply set to LOG(n)-1 at the Tail position down to 0 at Tail + n - 1 (modulo *k*) position, then repeated, however only the *wsz* enabled FFs will be written to. The selection lines for the *m* output lines are set in a similar fashion to select FFs' outputs from the Head counter value to Head + *rsz* - 1 (modulo *k*).

To push data into the BAR, the data is applied to the *Wdata* port, its size (length) to *wsz* port, and the *Write* signal is asserted. Once data is written successfully, the write acknowledge signal *Wack* is asserted. Similarly, to pop data from the BAR, the required size is applied to the *rsz* port and the *Read* signal is asserted. The BAR would place the data on the *Rdata* output port and asserts the read acknowledgement signal *Rack*. BAR has also two output signals *e* and *f* that indicate when BAR is empty or full, respectively. The read/write acknowledgement signals are crucial for the operation of the BAR since a push operation may not be completed even though the BAR is not full if the number of bits to be written exceeds the available number of bits in the BAR. Similarly, a pop operation may not be successful even if BAR is not empty if the desired number of bits to be read exceeds the available number of stored bits.

Table 1 Resource Utilization and Maximum Frequency of BARs for different combinations of buffer, read port, and write port sizes

k	n	m	FFs	Area (ALMs)	Freq (MHz)
64	16	16	101	335	451.26
64	16	32	117	362	436.68
128	16	16	168	691	381.68
128	32	32	184	898	377.07
128	32	64	216	967	369.41
256	32	32	315	1524	337.95
256	32	64	347	1715	319.39

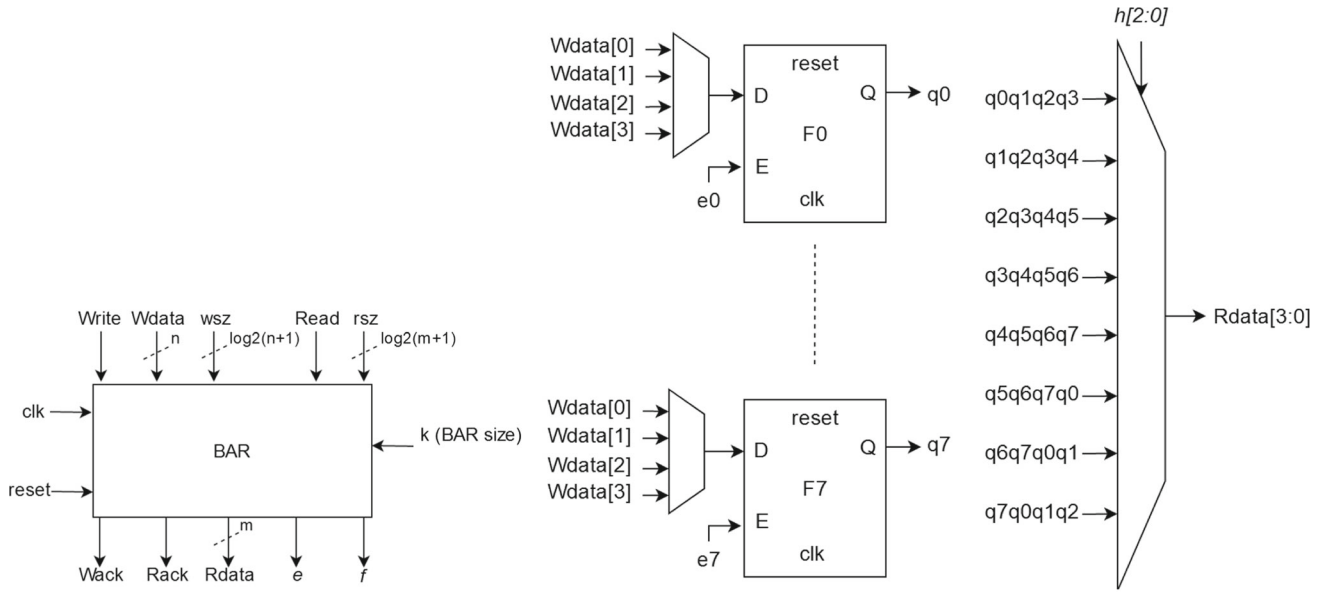
A simple Python code was developed to generate a structural Verilog code for the desired BAR given k, n and m as input parameters. This guarantees consistent synthesis using any logic synthesis tool.

3 Performance Evaluation

The area and speed of various synthesized BARs with varying sizes and different read and write port sizes have been evaluated using Intel's Quartus Prime Pro® targeting an Intel Aria® 10 GX device. Table 1 shows the synthesis results including number of flip flops and number of Adaptive Logic Modules (ALM) used (including those used as registers only). It also shows the maximum frequency for each synthesized BAR. For a given size, increasing the read and write port sizes has a smaller impact on speed than the total size of the BAR. Doubling the BAR size for the same read and write port widths reduces the speed by around 10% to 15%.

4 Comparison with An Alternative Design

The BAR design allows simultaneous push/pop of variable size data in one cycle at the cost of using wide MUXs at the FFs' inputs and the BAR's outputs. An alternative sequential bit-compacting buffer design could be obtained by first loading the variable-length input data into an *n*-bit parallel-load register (in one cycle), and then shift it serially into an *m*-bit output shift register in *wsz* cycles. Once this *m*-bit register is full, data can be read out as compacted m-bit words. This eliminates any MUXing at the buffer's FFs inputs/outputs, thus reducing the logic resources and increasing the operating frequency. With *n*=32 and *m*=32 synthesis results targeting the same Intel Aria ® 10 GX device showed that the sequential buffer requires only **69** ALMs and can run at a frequency of **875** MHz. With n=64 and m=64, the resulting buffer requires **98** ALMs and can run at a frequency of **805** MHz. However, unlike the BAR which can compact *wsz*-bit input data frag-



(a) Block Diagram (Interface) **(b)** Architecture

Fig. 1 Bit Addressable Register (BAR); **a** interface, and **b** architecture

ments into a longer, *rsz*-bit output words in (rsz/wsz)+1 cycles, the alternative register would need at least (*m/wsz*) +*m* + 1 cycles. Furthermore, for the alternative buffer, when the output shift register is partially filled and a new input word cannot entirely fit in the remaining bits of this register, this new word would have to be partially shifted into the output register till it gets filled. No new input can be written until the output register is read and the remaining portion of the previous input is shifted into it.

Based on the maximum frequency obtained from the synthesis results, the sequential buffer would take a minimum of **38.3** ns and **82** ns to compact 32-bit and 64-bit output data, respectively, assuming that the inputs' sizes were also 32 and 64-bits, respectively. These would be the absolute minimum times. The closest BAR configurations that were synthesized (as shown in Table 1) had k=128, n=32, and m=32 and 64 can run at **377** MHz and **369.4** MHz, respectively. They would take only **5.3** ns and **8.1** ns to compact 32-bit and 64-bit data, respectively. That is an order of magnitude reduction of latency. On top of this, these BAR configurations allow the storage of up to 128-bits (i.e. multiple 32-bit words) and the reading of variable size data.

5 Conclusion

A new FIFO design that allows writing/reading variable number of bits was proposed. It is fully synthesizable by any synthesis tool and is automatically generated from a developed template. We have demonstrated the efficiency of the proposed BAR design by synthesizing BARs with varying sizes using Intel® Arria® 10 GX FPGA. Compared to a simpler buffer design based on serially shifting the variable size data into a fixed size register, the BAR can achieve an order of magnitude lower compaction latency.

Acknowledgements This work is supported by King Fahd University of Petroleum & Minerals under Project No. DF191015.

References

1. Tao, D.; Di, S.; Chen, Z.; Cappello, F.: "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization, ' 'in. *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* **2017**, 1129–1139, 2017
2. Lee, S.M., Jang, J.H., Oh, J.H., Kim, J.K., Lee, S.E.: "Design of hardware accelerator for lempel-ziv 4 (lz4) compression," *IEICE Electronics Express*, vol. advpub, 2017.
3. Shawahna, A.; Sait, S.M.; El-Maleh, A.: Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access* **7**, 7823–7859, 2019
4. Tian, J., Di, S., Zhang, C., Liang, X., Jin, S., Cheng, D., Tao, D., Cappello, F.: "Wavesz: A hardware-algorithm co-design of efficient lossy compression for scientific data," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 74–88. [Online]. Available: https://doi.org/10.1145/3332466.3374525
5. Xiong, Q., Patel, R., Yang, C., Geng, T., Skjellum, A. Herbordt, M.C.: "Ghostsz: A transparent fpga-accelerated lossy compression framework," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 258–266.
6. Elrabaa, M.: A new fifo for transferring data between two unrelated clock domains. *Int. J. Electron.* **99**, 1–12, 2012
7. Smolansky, L., Kowal, S., Goren, A., Galanti, D.: "Adjustable depth/width fifo buffer for variable width data transfers," *US Patent US5673396A*, 1995.