Embedded multiport data buffer for a solid-state drive controller

Hailuan Liu¹, Lingyan Fan¹, a), Jianjun Luo¹, and Guorui Feng²

Abstract Solid-state drives (SSDs) for mobile and embedded systems may not provide very high performance by today’s standards; however, they are small, low cost and consume little power. SSD controllers are thus designed as DRAM-less chips. SRAM cells created by the IC foundry as a standard module are embedded in the SSD controller as data buffer and can be single-port or two-port. In an SSD controller, more than two IPs simultaneously access the same SRAM, such as the CPU, data interface, and multiflash memory channel. It is thus complicated to exchange data between multiple ports. A new architecture of the multi-port data buffer (M-Buffer) is proposed in this study to solve this problem. M-Buffer is composed of wide SRAM, a smart arbitrator and several interface port logics. The M-Buffer can be designed as a reusable architecture and is small, low cost and consumes little power.

Keywords: SSD, SRAM, data buffer
Classification: Integrated circuits (memory, logic, analog, RF, sensor)

1. Introduction

The recent explosion of information has spawned semiconductor technologies that help to store large amounts of data with a high degree of reliability and high performance at a lower cost with ever smaller form factors. Due to several advantages, including markedly faster read/write speeds, lower power consumption, and no noise, solid-state drives (SSDs) have become the technology of choice to replace traditional hard disk drives (HDDs) [1, 2, 3, 4, 5, 6]. An SSD consists of a controller ASIC (Application Specific Integrated Circuit) and a group of flash memory chips. Usually, a data stream is stripped among these flash memory chips using multiple flash channels [7, 8, 9, 10, 11, 12, 13, 14, 15].

A data buffer is a necessary component of an SSD controller for data transmission and exchange. Most SSD solutions use an external dynamic RAM (DRAM) chip attached to the flash controller to obtain a large buffer capacity [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]. A controller that is attached to external DRAM supports a multidata interface (typically AXI) that is easy for other IP integration. Auxiliary DRAM achieves strong performance, particularly high speeds in random read or write throughput, which are important in some applications, such as disk arrays for servers. SSD solutions attached to auxiliary DRAM can provide the desired storage device’s performance for these applications. However, auxiliary DRAM has disadvantages: additional DRAM increases material cost, and data may be lost in the case of sudden power-off.

However, there are some applications, such as those in mobile or embedded systems, which demand only a basic SSD module with acceptable performance, small size, low cost, low power consumption, and high reliability. In this study, an SSD controller is designed without attached DRAM but with embedded SRAM. The embedded SRAM cell, which is proposed by the IC foundry as a standard module, is used as the data buffer for the SSD controller.

Controllers generally include only single-, two- or dual-port SRAM. However, in an SSD controller, more than two IPs, such as the CPU, data interface, and multi-flash memory channel, simultaneously access the same SRAM [12, 13]. Traditional SRAM is thus not suitable for SSD data buffers. In this study, we propose the embedded multiport data buffer SRAM (M-buffer) for an SSD controller.

In this paper, section 2 describes the typical SSD controller architecture and then the SSD controller architecture with M-buffer SRAM. Section 3 describes the design of the M-buffer SRAM. Section 4 presents and discusses the experiment results of this study, and section provides the conclusions of this study.

2. SSD controller

We first provide an overview of the system architecture of an SSD. An SSD consists of a group of NAND flash memory modules (chips) and a controller. A host computer communicates with the SSD through a high-speed host interface (e.g., SAS, SATA, PCIe bus), which connects to the SSD controller. The controller is then connected to each of the NAND flash chips via memory channels.

Fig. 1 shows the typical architecture of an SSD controller. The SSD controller manages the NAND flash memory and handles I/O requests received from the host [7, 13]. The flash memory is spread across multiple flash chips, where each chip contains one or more flash dies. Contemporary SSDs typically have 4-16 flash chips and have as many as 16 dies per chip. Each chip is connected to one or more physical memory channels, and these memory channels are not shared across chips.

Each flash memory channel has its own data and control connection to the SSD controller. The connection for each channel is typically an 8- or 16-bit wide bus between the controller and one of the flash memory chips. Both data and flash commands can be sent over the bus.

To perform these tasks, the controller runs firmware,
which is often referred to as the flash translation layer (FTL). FTL tasks are executed on one or more embedded processors inside the controller. The controller has access to DRAM, which can be used to store various controller data (e.g., how host memory addresses map to physical SSD addresses) and to cache relevant SSD pages. When the controller handles I/O requests, it performs a number of operations on the data, such as scrambling the data to improve raw bit error rates and performing ECC encoding/decoding.

A typical SSD controller with DRAM achieves strong performance, particularly high speeds in random read or write throughput but also has some disadvantages. The DRAM interface has more than twenty pins between the DRAM and the controller, and the DRAM interface on the controller chip also occupies the die area, thus increasing die size and package cost. An onboard DRAM chip requires an additional PCB (printed circuit board) area, which may be unfeasible for small form-factor applications.

In this study, an SSD controller is designed without attached DRAM, as shown in Fig. 2. In the proposed architecture, multiple data buffer SRAM is embedded in an SSD controller; thus, DRAM is not required. This design is small, low cost, consumes little power, and is highly reliable; however, random data access speeds are slightly lower than other designs. In general, an SSD controller with DRAM can only be used as the CPU’s code memory, stack and variable memory. In some cases, single-port SRAM can also be used in some control logic designs as a ping-pong data buffer. Two- or dual-port SRAM is typically designed as FIFO (first-in-first-out) for data synchronization or data transmission. However, for an N port SOC system, such as an SSD controller, N ports must access the SRAM simultaneously, including the CPU, multilash memory channel and high-speed host interface. For example, if N/2 ports write and N/2 ports read simultaneously, in general, N single-port SRAM is required. However, it is complex for the SSD controller to achieve such a multi-segmented data transfer with one- to N-port data striping.

In this study, an M-buffer SAM architecture is designed for on-chip multiple-port buffer access. Fig. 3 shows the proposed M-buffer SAM structure, which includes one large single-port SRAM with a wide data bus, a highly efficient arbiter and an N interface logic module.

To achieve multiple port access, the M-Buffer bandwidth must match the sum of each port bandwidth. For an N-port M-Buffer

\[
BW(SRAM) \geq \sum_{i=0}^{N} BW(port_i) \quad (1)
\]

where the operator BW is the bandwidth. Assuming that there are N ports, and that each port has the same bus as \(W_{port}\) and the same frequency \(f_{port}\), the total port bandwidth is:

\[
BW(SRAM) = N \times W_{port} \times f_{port} \quad (2)
\]

Time-division multiplexing is commonly applied in this situation. When the SRAM frequency is the same as the port frequency, a single SRAM port has a bus width of \(W_{SRAM} = N \times W_{port}\). The arbitrator then decides which port can connect to the SRAM to access relevant data. The arbitrator uses round-robin mode, and the port is connected with an interface logic module to M-Buffer. The interface N logic module with a small FIFO balances the different data widths between the port and SRAM.

Fig. 4 shows the M-Buffer running time. From cycle 0 to N – 1 with a round-robin arbitrator, each port accesses the SRAM in one cycle. If port0 is the read operation, the interface 0 logic module obtains \(W_{SRAM}\) bit Data0 from SRAM on Cycle0. Then, port0 sends data on Cycle1 and obtains \(W_{port}\) bit data on each clock. Until Cycle n, interface 0 obtains the next \(W_{SRAM}\) bit Data1. Port0 sends the last part of Data0 and then continues to transfer the first part of Data1 during the next cycle. The same process occurs with interface 1, 2, etc. On the port side, each clock obtains
W_{port} bit data and determines the bandwidth requirements for W_{port} × f_{port}.

Compared to SRAM’s time consumption, the write time of a M-Buffer port can be designed exactly as with SRAM. However, the port read time is different because M-Buffer obtains data from the SRAM depending on the arbitrator. The read latency ranges from 0 to N – 1 cycle.

### 3.2 Arbitrator optimization

The round-robin arbitrator is not the most effective system when all ports are not active concurrently. Even though only one port is active during data transfer, the system still has the maximum read latency N – 1 cycle. For some N port systems, if M(M ≤ N) ports access the M-Buffer at peak, the SRAM bandwidth can decrease to \( \sum_{i=0}^{M} BW(\text{port}_i) \), which can achieve the proposed design easily and reduces the logic size.

In the conventional method, a port only can have one of only two statuses: active or inactive. In general, a port is active if there are input data. The status can be defined as:

\[
P(i) = \begin{cases} 
1, & \text{ith port is active} \\
0, & \text{ith port is not active}
\end{cases} (3)
\]

When port \( i \) is chosen, the controller processes the port request if port \( i \) is set to active status.

When we search all ports, input ports can be calculated by \( \sum_{i=0}^{M} [P(i) = 1] \), where \( \{X\} \) is the Iverson bracket and equals 1 when statement \( X \) is true, and 0 otherwise. Although this process cannot skip ports that have access to data, it can respond to many low-cost requests.

We define \( N \) interfaces as a loop; therefore, there are \( NT \) interfaces in \( T \) loops. The \( i \)th accessed interfaces at the \( j \) loop are named \( I_j(i) \), which contains \( D_j(i) \) bytes of data. If the data are input and subjected to uniform distribution, the total input data are \( \frac{N \times D_j(i)}{2} \) during a loop.

To use the arbitrator efficiently, we improve the arbitrator optimization and define the active probability of interface \( i \) as \( P_j(i) \) on the \( j \)th loop. First, we calculate current busy status using accessed data in the past loops:

\[
P_j(i) = \sum_{l=1}^{k} D_{j-l}(i) \quad (4)
\]

The active probability of interface \( i \) as \( P_j(i) \) can be obtained by:

\[
P_j(i) = \frac{R_j(i)}{N} \sum_{i=1}^{N} R_j(i) \quad (5)
\]

To ensure that all data is transferred properly, two rules are used to allocate the number of interfaces:

1) At least an interface is assigned if \( P_j(i) > 0 \).
2) The assigned number \( s \) is greater than or equal to one of \( t \) if \( P_j(i) > P_j(t) \).

A simple access implementation is applied, followed by a threshold. To solve the problem of allocation without a loss of generality, we select \( n \) probabilities: \( \{P_j(1), \cdots P_j(M)\} \), where \( P_j(i) > 0, M \leq N \).

Furthermore, the corresponding numbers of allocations are \( n_1, \cdots n_M \). The condition of \( n_1 + \cdots + n_M = N \) is thus satisfied. Following the previous two limited rules, the task of finding a proper allocation method is set to:

\[
\min \sum_{i=1}^{M} |P_j(i) - \frac{N_j}{N}| \quad (6)
\]

The solution can be determine by following the following three steps. First, we allocate every port to an interface. This operation can guarantee that Rule 1 is strictly performed. Second, we allocate \( N – M \) interfaces for the remaining \( M – 1 \) ports:

\[
P_{\min}(i') = \min\{P_j(1), \cdots , P_j(M)\} \quad (7)
\]

\[
P_{\max}(i') = \max\{P_j(1), \cdots , P_j(M)\} \quad (8)
\]

The residual of probabilities can be obtained by \( P_j(i) = P_j(i) - P_j(i = 1, \cdots , M) \). Then, we calculate the step length:

\[
\Delta = \frac{P_j(i)}{N - M - 1} \quad (9)
\]

Last, we select port \( i' \), which has the maximum residual probability, and allocate it to one interface. The corresponding probability is decreased by \( P_{\max}(i') – \Delta \). To repeat Steps 2 and 3, we allocate all the remaining \( N – M \) interfaces. To improve the efficiency of the arbitrator and avoid long-term missed nodes, the proposed method uses cross-cycle evaluation to access the port by \( P(i,j) > P(i + n, j - 1) \), \( n = 1, 2, \cdots \).

During a loop, the mean accessed data can be calculated by:

\[
E[\text{cycle}(j)] = \frac{\sum_{i} [P(i,j-1) > T]D(i,j-1)}{N} \quad (10)
\]
where \( E[\text{cycle}(j)] > \frac{NTD(i)}{2} \) if \( T \) is greater \( T(i)/2 \), which indicates that the proposed method has data that is more frequently accessed than the method in Eq. (1) during a loop. Because \( D(i, j) \geq D(i, j - 1) \) and \( P(i, j) \geq P(i, j - 1) \), true accessed data are greater than \( E[\text{cycle}(j)] \). To mitigate missed access, we also set \( T \) to be a small value.

An intelligent round-robin arbitrator with a smart skip function will improve performance by only scheduling an interface that has operation requests. The difference between a normal round-robin arbitrator and a round-robin arbitrator with smart skip is shown in Fig. 5. If Interface1 and Interface3 have no access requests, then the later arbitrator will skip them.

### 4. Experiment results

To evaluate the real system performance and reliability of the proposed algorithm, a real SSD controller was designed to obtain real results. The designed SATA SSD controller has no external DRAM interface, and each flash channel can drive one 8- or 16-bit bus flash memory interface. SSDs with eight flash channels and one on-chip MCU are used, and there are a total of 10 buffer access ports, including one SATA port, eight flash ports and one CPU port, that access data simultaneously. The SSD architecture can support up to eight flash memory channels with a SATA 6.0 Gb/s data transfer rate.

#### 4.1 Physical area of the M-buffer SRAM

The flash controller requires memory for both the mapping buffer and data cache. Without a DRAM system, the total SRAM size is typically 1 to 2 MB to meet the SSD performance requirement.

If we use the typical ping-pong FIFO SRAM mode, the system consists of \( N \) SRAM cells with a row bit of 32. Each SRAM cell size can be 4/8/16/32 K depending on the flash algorithm. If we use the M-Buffer SRAM method, the row bit of M-Buffer SRAM cells is 128.

After developing a single-port SRAM 65 nm process, we found that the area was not linear with density. Table I shows the SRAM memory size and area. The column (real area) is the area implemented on place and route in each SRAM.

| SRAM Cell type | Columns Words | Row bits (bit) | SRAM total byte (Kbyte) | Area (um2) | Area real (um2) |
|----------------|---------------|---------------|-------------------------|------------|-----------------|
| 1              | 1024          | 32            | 4                       | 27855.17   | 34939.17        |
| 2              | 2048          | 32            | 8                       | 55078.4    | 65444.4         |
| 3              | 4096          | 32            | 16                      | 118743.04  | 132935.04       |
| 4              | 8192          | 32            | 32                      | 196485.12  | 215349.12       |
| 5              | 8192          | 128           | 128                     | 725528.16  | 762044.16       |

We know that the area of M-buffer SRAM is the smallest.

#### 4.2 SSD performance with M-buffer SRAM

A controller with the architecture shown in Fig. 6 was designed without auxiliary DRAM and constructed as a real silicon chip. Wafers were manufactured by a 65-nanometer semiconductor process, and the layout of this chip is shown in Fig. 7.

The chip was 3908 \( \times \) 3746 \( \mu \text{m} \) and was packaged with a 207-ball BGA. The chip supports up to eight flash channels running in parallel and was contained 1 MB of embedded M-Buffer SRAM. The chip’s packaging cost was lower than comparable chips because there are no DRAM pads/pins. In a real system, a DRAM chip is no longer required and does not occupy space on the PCB. Table II shows that the proposed method is cost-effective compared with the DRAM-based method.
faces is proposed. The proposed architecture can be helpful in future research of embedded SSDs. A designer can also standardize the M-Buffer; for different applications, using an M-buffer adds or reduces the number of interfaces to meet the whole chip’s requirements. The port protocol can also be designed as a popular bus protocol, such as AHB/AXI/Wishbone. Using an M-Buffer reduces chip size, keeps costs low and reduces power consumption. The proposed system can also use more memory, and the more memory the system uses, the better the advantage of M-Buffer. This architecture was also demonstrated proven in real silicon on an SSD controller chip in this study.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (2018YFB2202900), National Basic Research Plan of China (JCKY2018415C001) and the National Natural Science Foundation of China (Grant No. U1709220).

References

[1] L.-P. Chang and T.-W. Kuo: “An adaptive striping architecture for flash memory storage systems of embedded systems,” IEEE Proceedings Real-Time and Embedded Technology and Applications Symposium (2002) 187 (DOI: 10.1109/RTTAS.2002.1137393).
[2] K. Takeuchi: “Novel co-design of NAND flash memory and NAND flash controller circuits for sub-30 nm low-power high-speed solid-state drives (SSD),” IEEE J. Solid-State Circuits 44 (2009) 1227 (DOI: 10.1109/JSSC.2009.2014027).
[3] K. Smith: “LSI corporation, understanding SSD over-provisioning,” EDN Network (2013).
[4] J. Luo, et al.: “Solid-state drive controller with embedded RAID functions,” IEICE Electron. Express 11 (2014) 20140419 (DOI: 10.1587/elex.11.20140419).
[5] L. Zuolo, et al.: “Solid-state drives memory driven design methodologies for optimal performance,” Proc. IEEE 105 (2017) 1589 (DOI: 10.1007/978-981-13-0599-3-7).
[6] K. Takeuchi: “Novel co-design of NAND flash memory and NAND flash controller circuits for sub-30 nm low-power high-speed solid-state drives (SSD).” 2008 IEEE Symposium on VLSI Circuits (2008) 124 (DOI: 10.1109/VLSIC.2008.4585977).
[7] T. Hatanaka and K. Takeuchi: “NAND controller system with channel number detection and feedback for power-efficient high-speed 3D-SSD,” IEEE J. Solid-State Circuits 47 (2012) 1460 (DOI: 10.1109/JSSC.2012.2190187).
[8] J. Kim, et al.: “Chip-level RAID with flexible stripe size and parity placement for enhanced SSD reliability,” IEEE Trans. Comput. 65 (2016) 1116 (DOI: 10.1109/TC.2014.2375179).
[9] K. Han, et al.: “WAL-SSD address remapping-based write-ahead-logging solid-state disks,” IEEE Trans. Comput. 69 (2020) 260 (DOI: 10.1109/TBC.2019.2947897).
[10] Y. Cai, et al.: “Error characterization, mitigation, and recovery in flash memory based solid-state drives,” Proc. IEEE 105 (2017) 1589 (DOI: 10.1109/JPROC.2017.2713127).
[11] J.-U. Kang, et al.: “A multi-channel architecture for high performance NAND flash-based storage system,” Journal of Systems Architecture 53 (2007) 644 (DOI: 10.1016/j.sysarc.2007.01.010).
[12] J. Luo, et al.: “A solid state drive architecture with memory card modules,” IEEE Trans. Consum. Electron. 62 (2016) 17 (DOI: 10.1109/TCE.2016.7448558).
[13] L. Zuolo, et al.: “Solid state drives memory driven design methodologies for optimal performance,” Proc. IEEE 105 (2017) 9 (DOI: 10.1109/JSSC.2015.2486406).
[14] M.-L. Chiang, et al.: “Managing flash memory in personal communication devices,” Proc. International Symposium on Consumer Elec-

The SATA port is a 64-bit data bus, each flash port is an 8-bit data bus, and the M-buffer SRAM is 128 bits wide; thus, the bandwidth of the M-buffer SRAM is equal to the SATA bandwidth + flash bandwidth $\times$ 8. As shown in Fig. 8, the SATA port can write, while eight flash ports read stripes concurrently, and the flash port can achieve the full bandwidth. The gap that is shown on the waveform is caused by flash IP, not by the M-buffer limit.

The proposed controller with eight channel flash chips on the evaluation boards achieved good performance and sufficient reliability; Table III highlights performance that is sufficient for most embedded or mobile applications. There is no marked difference between the read speed of the proposed and DRAM methods. The sequential read speed is approximately 539 MB/s, which is near the upper limit of the SATA-III’s 6.0 Gb/s transfer rate when considering overhead. As expected, the DRAM method provides higher random read and write speeds.

5. Conclusion

In this paper, a new M-Buffer architecture that combines all required memory into one large buffer with several inter-
[15] M. Gupta: *Storage Area Network Fundamentals* (Cisco Press, Indianapolis, 2002) 1st ed. 270.

[16] S.J. Im and D.K. Shin: “Flash-aware RAID techniques for dependable and high-Performance flash memory SSD,” IEEE Trans. Comput. 60 (2011) 80 (DOI: 10.1109/TC.2010.197).

[17] K. Smith: “LSI corporation, understanding SSD over-provisioning,” EDN (2013).

[18] E.H. Nam, et al.: “An out-of-order flash memory controller architecture,” IEEE Trans. Comput. 60 (2011) 653 (DOI: 10.1109/TC.2010.209).

[19] Y. Cai, et al.: “Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation,” IEEE ICCD (2013) 123 (DOI: 10.1109/iccd.2013.6657034).

[20] Y. Cai, et al.: “Data retention in MLC NAND flash memory: Characterization, optimization, and recovery,” IEEE HPCA (2015) 551 (DOI: 10.1109/hpca.2015.7056062).

[21] F. Masuoka, et al.: “A new flash EEPROM cell using triple polysilicon technology,” IEEE Electron Devices Meeting (1984) 464 (DOI: 10.1109/iedm.1984.190752).

[22] Y. Lee, et al.: “The challenges and limitations on triple level cell geometry and process beyond 20 nm NAND Flash technology,” 2010 IEEE International Memory Workshop (2010) 1 (DOI: 10.1109/imw.2010.5488388).

[23] Y. Cai, et al.: “Error patterns in MLC NAND flash memory: measurement, characterization, and analysis,” IEEE DATE (2012) 521 (DOI: 10.1109/date.2012.6176524).

[24] Q. Xiong, et al.: “Characterizing 3D floating gate NAND flash: observations, analyses, and implications,” ACM Transactions on Storage 14 (2018) 1 (DOI: 10.1145/3162616).

[25] Y. Zhu, et al.: “ALARM: a location-aware redistribution method to improve 3D FG NAND flash reliability,” IEEE NAS (2017) (DOI: 10.1109/nas.2017.8026841).

[26] R. Bez, et al.: “Introduction to flash memory,” Proc. IEEE 91 (2003) 489 (DOI: 10.1109/NAS.2017.8026841).

[27] R. Micheloni, et al.: *Inside NAND Flash Memories* (Springer, 2010).

[28] M. Assar: U.S. Patent 5388083 (1995).

[29] C.-C. Chung, et al.: “A high-performance wear-leveling algorithm for flash memory system,” IEICE Electron. Express 9 (2012) 1874 (DOI: 10.1587/elex.9.1874).

[30] I. Shin and J. Kim: “Performance analysis of buffer management policy considering internal parallelism of solid state drives,” IEICE Electron. Express 15 (2018) 20180419 (DOI: 10.1587/elex.15.20180419).