A resource and timing optimized PCIe DMA architecture using FPGA internal data buffer

Yingxiao Zhao\textsuperscript{a), Xin Liu, and Jiong Yang

National Key Laboratory of Science and Technology on ATR, National University of Defense Technology, 109 Deya Road, Changsha 410073, China

\textsuperscript{a)} zhaoyingxiao123@126.com

Abstract: In this paper, we propose a resource and timing optimized PCIe DMA architecture using FPGA internal data buffer memory. Firstly, an optimized PCIe DMA control process is proposed, focusing on reducing the capacity of required data buffer memory in PCIe DMA, which is realized by fastening DMA completion response, optimizing DMA register configuration, and avoiding conflicts between multiple threads. Required data buffer memory capacity is reduced by 97.6\% from 24.10 MB to 0.56 MB, making FPGA internal memory resource enough for DMA transmission. Secondly, timing failure problems in FPGA caused by large internal memory utilization as PCIe DMA data buffer are solved, realized by a timing-optimized FIFO structure and a low-delay FIFO control mechanism. FPGA memory resource utilization rate without timing failure is increased from 12.4\% to 100\%, ensuring the reliability of PCIe DMA data transmission. This paper expands the application of PCIe data transmission, reducing the cost and complexity of relevant circuit design.

Keywords: PCI Express, DMA architecture, FPGA, timing optimization, on-chip memory resource

Classification: Integrated circuits

References

[1] P. Durante, \textit{et al.}: “100 Gbps PCI-Express readout for the LHCb upgrade,” IEEE Trans. Nucl. Sci. \textbf{62} (2015) 1752 (DOI: 10.1109/TNS.2015.2441633).
[2] K.-H. Cheng, \textit{et al.}: “A 5-Gb/s inductorless CMOS adaptive equalizer for PCI Express generation II applications,” IEEE Trans. Circuits Syst. II, Exp. Briefs \textbf{57} (2010) 324 (DOI: 10.1109/TCSII.2010.2047311).
[3] S. Otani, \textit{et al.}: “Peach: A multicore communication system on chip with PCI Express,” IEEE Micro \textbf{31} (2011) 39 (DOI: 10.1109/MM.2011.93).
[4] J. Ajanovic: “PCI Express 3.0 overview,” IEEE Hot Chips 21 Symposium (HCS) (2009) 1 (DOI: 10.1109/HOTCHIPS.2009.7478337).
[5] J. Gong, \textit{et al.}: “An efficient and flexible host-FPGA PCIe communication library,” 24th International Conference on Field Programmable Logic and Applications (FPL) (2014) 1 (DOI: 10.1109/FPL.2014.6927459).
[6] H. Kavianipour, \textit{et al.}: “High performance FPGA-based DMA interface for
1 Introduction

PCI Express (PCIe) is a high-speed serial communication protocol widely used in data acquisition and data communication systems [1, 2, 3], with increasing link rate from 2.5 Gb/s per lane in Generation 1 (Gen 1), 5.0 Gb/s per lane in Gen 2, to 8.0 Gb/s per lane in Gen 3 [4]. Large-capacity PCIe data transmission between the endpoint and the root complex is usually realized by Direct Memory Access (DMA). With flexible programmable structure and high data throughput, Field Programmable Gate Array (FPGA) is often used as a PCIe endpoint, in which users can design their own DMA engine according to actual requirement.

Massive work has been done in PCIe DMA based on FPGA [5, 6, 7, 8]. [5] developed an efficient and flexible host-FPGA PCIe communication library. [6] presented a PCIe data communication suite, and made performance test and comparison on different FPGAs. [7] developed a new DMA structure using scatter/gather host memory acquisition scheme and FPGA DMA address ring buffer to get higher data throughput. [8] proposed a PCIe port multiplexing method for parallel acquisition of multi-channel data stream.

However, problems exist in current research. The usage of FPGA data buffer memory in PCIe DMA, which is necessary in the interval between each DMA cycle, has not been fully analyzed or discussed. Actually, because of large memory requirement, adding external memory chips for FPGA data buffer, such as DDR3 or QDRII, is a simple and effective measure which has been taken in [8]. However, it may be not proper considering factors such as resource occupation, design complexity, multi-channel parallel data acquisition, and FPGA function expansion. Firstly, it will occupy large numbers of FPGA pins and increase the complexity in circuit board design. Secondly, with the upgrading of FPGA products as shown in...
Table I [9, 10, 11], the amount of high speed serial transceivers is far more than external memory controllers, which will be not enough when making multi-channel parallel data acquisition in a single FPGA. Lastly, using external data buffer memory in PCIe DMA may limit the function expansion of FPGA, because data retransmission, real-time signal processing realized in FPGA may also require external memory.

So, using memory resource inside FPGA as data buffer, usually instantiated as a FIFO [6, 7], can be a better solution, which avoids the above problems. However, simply using a FIFO inside FPGA may cause data overflow because of low internal memory capacity, and data errors because of serious timing problems in FPGA with high internal memory utilization rate.

To solve these problems, this paper proposes a resource and timing optimized PCIe DMA architecture using FPGA internal data buffer. In Section 2, a basic PCIe DMA structure with FPGA internal buffer for data stream transmission has been introduced. In Section 3, DMA control optimization is proposed to decrease the required capacity of data buffer, making FPGA internal memory resource enough for PCIe DMA. In Section 4, FPGA timing optimization with a new FIFO structure is proposed to overcome the timing failure problems caused by large utilization of FPGA internal memory resource. In Section 5, performance test and comparison is made and the engineering application is introduced. Little Part of our work has been published in [12], based on which this paper makes full analysis and gives lots of adjustment and expansion to improve the stability and generality of the system.

2 Basic PCIe DMA structure for data stream transmission with FPGA internal data buffer

In PCIe DMA transmission, we use a disk array server as the host PC, and use an FPGA circuit card as the PCIe endpoint device. The device number of FPGA is Xilinx Kintex-7 xc7k480t, communicating with the disk array server in PCIe Gen2 8x mode.

A standard PCIe DMA process proposed by Xilinx Corporation is shown in Fig. 1 [13]. To finish the DMA operation, several registers in FPGA PCIe BAR memory have been set, which can be configured by the host PC.

The realization of PCIe DMA process needs the co-design of software project and FPGA project. Drivers and open source software project for host PC is provided by Windriver Corporation, and the open source demo project for FPGA with an IP core is provided by Xilinx Corporation.
In FPGA demo project provided by Xilinx, data with constant value is transferred, which should be replaced by data stream in actual PCIe DMA application. This paper designed an interface connection between FIFO and PCIe IP core to realize DMA write data stream transmission, which is shown in Fig. 2. (Implementation and optimization of DMA read data stream transmission is similar, which is omitted in this paper)

As Fig. 2 shows, the FIFO is used for temporary data store, data width conversion and clock domain conversion. Data stream is output to PCIe IP core, which has AXI stream user interface. Axi\_tx\_tdata bus has a valid assignment only when both of axis\_tx\_tready and axi\_tx\_tvalid signal are valid, at which moment, the output data of FIFO, which is connected to axi\_tx\_tdata bus, should be updated in the next clock cycle. The basic design in Fig. 1 and Fig. 2 can ensure correct PCIe data stream transmission in low speed. While for high PCIe throughput occupation rate with only FPGA internal data buffer memory used, it needs to be further optimized, which will be discussed in the Section 3 and 4.

3 DMA control optimization for lower capacity of required data buffer

As Fig. 1 shows, every DMA cycle needs initialization such as register configuration by the host PC, meanwhile PCIe transmission halts and data should be temporarily stored in FPGA data buffer memory. Let $T_{INT}$ be the interval time between each DMA transmission, $B$ the data throughput of PCIe port, $\alpha$ the throughput occupation rate in actual transmission, and $V$ the required data buffer memory capacity. It can be calculated in Eq. (1) that,

![Fig. 1. Xilinx standard PCIe DMA control process](image1)

![Fig. 2. FPGA design for DMA write data stream transmission](image2)
\[ V = a \cdot B \cdot \max(T_{\text{INT}}) \]  

In this platform, the theoretical value of \( B \) is 3459 MB/s with 128B payload size, and 32 bit PCIe memory addressing [7]. FPGA has only about 4 MB internal memory resource capacity, so even \( T_{\text{INT}} \) at level of milliseconds may cause data overflow in PCIe DMA, which can easily happen when multiple threads running on the host operating system. The composition of \( T_{\text{INT}} \) is derived in Eq. 2, in which \( T_R \) means the DMA completion response time, \( T_{\text{INS}} \) the configuration instruction execution time, \( T_O \) the occupation time by other threads. 

\[ T_{\text{INT}} = T_R + T_{\text{INS}} + T_O \]  

Next, we’ll make optimization based on \( T_R \), \( T_{\text{INS}} \) and \( T_O \) respectively to reduce \( V \). 

3.1 Optimization based on \( T_R \) 

Two methods can be used in DMA completion response. One is interrupt mode, in which the FPGA generates an interrupt request by sending an MSI TLP, so \( T_R \) mainly comprises a MSI memory write transaction and the interrupt response time of CPU. The other is polling mode, in which the host keeps reading the DMA done flag in FPGA by PCIe until it turns to be 1, so \( T_R \) mainly comprises a PCIe memory read transaction and a completion transaction. 

[13] adopted the interrupt mode which can save more CPU resource. However, when using FPGA internal data buffer memory, the real time performance of software operation should be firstly considered. On the whole, PCIe port has higher throughput and lower latency compared with the data link of interrupt message. So \( T_R \) value in polling mode will be smaller than in interrupt mode. Next, we will prove the conclusion by experiments on the platform.

We set a counter in FPGA PCIe register driven by 250 MHz PCIe user clock to measure \( T_R \), result of which is shown in Table II. \( T_R \) in polling mode is much lower, which should be selected, although it will occupy more CPU resource.

| DMA completion response mode | Samples | Mean value of \( T_R \) | Max value of \( T_R \) |
|-----------------------------|---------|------------------------|----------------------|
| [13] (Interrupt mode)       | 10000   | 84632 ns               | 250944 ns            |
| This paper (polling mode)   | 10000   | 2232 ns                | 14712 ns             |

3.2 Optimization based on \( T_{\text{INS}} \) 

In Fig. 1, each DMA cycle needs configuring 6 PCIe DMA registers [13]. If we simplify the DMA control process and cut the number of registers to be configured, the configuration instruction execution time \( T_{\text{INS}} \) will be also reduced.

In successive data transmission, the TLP count and TLP size can be set as a fixed value, and DMA address turns in Ping-Pong mode (for DMA thread and data store/process thread running simultaneously). Accordingly, we propose an optimized PCIe DMA process shown in Fig. 3.

As shown in Fig. 3, the configuration of TLP size, TLP count, DMA Ping/Pong address, and assert PCIe FIFO reset will be carried out only once after data transmission has started. In each DMA cycle, the host PC makes Ping-Pong address
selection, cancel PCIe FIFO reset, asserts DMA initiator reset, and sets time delay for FPGA operation, all of which will be finished in one register configuration, whose new definition is shown in Table III. Then the FPGA will carry on canceling DMA initiator reset and starting DMA accordingly, time of which can be neglected compared to $T_{INS}$.

By comparing Fig. 1 and Fig. 3, we can find that the number of registers to be configured has been reduced from 6 to 1, which decreases $T_{INS}$ by about 80%.

### 3.3 Optimization based on $T_D$

It has a bad effect on the real time performance of software operation when multiple threads running on the operating system like Window or Linux, which adds a time delay $T_D$ to $T_{INT}$. So we need to take extra measures to coordinate CPU resource utilization and decrease the occupation time of other threads in the interval of each DMA transmission.

(1) Overcome the conflict of PCIe port occupation

Usually, the host PC monitors the Built-in Test (BIT) information of FPGA board by reading relevant PCIe registers in a fixed frequency, which may lead to port confliction with PCIe DMA register configuration. To solve this problem, we use BIT interrupt alarm mechanism, in which the FPGA will send an MSI interrupt TLP alarm to the host PC once detecting abnormal BIT information. Because PCIe protocol supports full duplex transmission, this method will avoid port confliction with DMA register configuration and reduce FPGA buffer memory required.

(2) Coordinate the threads relevant to PCIe DMA.

Usually, data will be processed or stored after DMA transmission to the host PC memory. To ensure the parallel running of process/store thread and DMA thread, we add a copy thread, and use Ping-Pong mechanism and memory pool technique to ensure the parallel running of each thread.

---

**Table III.** Register definition for optimized PCIe DMA control process (000H,R/W)*

| FPGA Family | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0 |
|-------------|--------|--------|--------|--------|
|             | 31–24  | 23–20  | 19–16  | 15–8   | 7–4   | 3    | 2    | 1    | 0    |
| Time delay count for FPGA to start a new DMA cycle | Data path width | Version number | Time delay count for FPGA to cancel initiator reset | Ping/Pong select | DMA write/read select | PCIe FIFO reset | Initiator reset |

*The underline parts in the table are new added for optimization based on [13]
(3) Overcome the CPU resource conflict with other threads.

Usually, there are hundreds of irrelevant threads running on the operating system. Because of strict requirement for the real time performance of DMA thread, we should set its priority to the highest level and make the DMA thread monopolize a certain CPU core. In Linux system, Function `sched.setaffinity()` can be used. And in Window system, function `SetThreadAffinityMask()` and Function `SetProcessAffinityMask()` can be used.

3.4 Test results

In this section, we optimize the DMA control process by decreasing the time interval between each DMA transmission. We set a counter in FPGA PCIe register driven by 250 MHz PCIe user clock to measure $T_{INT}$, and get the maximum value in 2 hours’ test. The test results are shown in Table IV. It can be found that, $\max(T_{INT})$ has been obviously reduced from 6.965 ms to 0.161 ms after optimization, with the required FPGA data buffer capacity $V$ reduced by 97.6% from 24.10 MB to 0.56 MB, which makes memory resource in FPGA (about 4 MB) enough for PCIe DMA transmission.

| PCIe DMA control process          | $\max(T_{INT})$ | $\alpha$ | Required data buffer capacity |
|-----------------------------------|-----------------|----------|-------------------------------|
| Standard [13]                     | 6.965 ms        | 100%     | 24.10 MB                      |
| Optimization in this paper        | 0.161 ms        | 100%     | 0.56 MB                       |

4 FPGA timing optimization for higher utilization rate of internal memory resource

Using optimized PCIe DMA control process in Section 3, FPGA internal data buffer memory has reduced to 0.56 MB when $\alpha = 100\%$. Because the available internal memory capacity of FPGA xc7k480t is about 4 MB, it can support data stream with instantaneous data rate 7 times of PCIe data throughput. While, timing failure risks exist in the implementation of FPGA project in Fig. 2, because of large FPGA internal memory resource used. And it can cause data error in transmission. By means of Vivado Suite, we make static timing analysis on FPGA projects with different FIFO capacity, the results of which are shown in Table V. Negative worst slack value means timing failure paths exist. Next, we will make timing optimization according to the reflected timing failure problems.

4.1 A novel timing-optimized FIFO structure for PCIe DMA

As shown in Table V, timing error appears when the FIFO capacity exceeds 1 MB. Combining the failing paths of implementation strategy 1 and 2, we conclude three main factors leading to timing failure.

1) Large FIFO capacity with scattered block RAM resource and complicated addressing logic.

2) FIFO internal delay path crossing clock domain (from data source clock to PCIe user clock).
3) FIFO rd.en delay path driven by PCIe IP core output signal axis.tx.tready with too much fan out and too long route delay.

The above three factors leading to timing failure problems are coupled with each other, and a single implementation strategy can’t overcome them altogether. So, we can design a three-part FIFO cascade structure to decouple the three factors and make targeted timing optimization respectively.

Xilinx FIFO IP core provides four basic types of FIFOs [14], the performance comparison of which is shown in Table VI.

| Table V. | Timing analysis results in basic PCIe DMA design |
|---|---|
| FIFO capacity | Place and route strategy | Worst slack (ns) | Failing path | Clock domain | Timing delay proportion | Max fan out |
| 524288 B | strategy 1.2 | 0 | none | none | none | none |
| 1048576 B | strategy 1 | −0.29 | delay path inside FIFO | data source clock to PCIe user clock | 50% logic | 50% route | 1 |
| 1048576 B | strategy 2 | −0.19 | FIFO rd.en signal delay path | PCIe user clock | 22% logic | 78% route | 209 |

| Table VI. | Four basic types of FIFO provided by Xilinx IP core |
|---|---|
| FIFO Type | Read/Write Clock Domain | Data Width Conversion | Memory Resource | Read/Write Control Logic | Read Mode |
| Block RAM | Common /Independent Clock | Support | Block RAM | Configurable Logic Block (CLB) | Standard /First-Word First-Through |
| Distributed RAM | Common /Independent Clock | Not Support | CLB | CLB | Standard /First-Word First-Through |
| Shift Register | Common Clock | Not Support | CLB | CLB | Standard |
| Built-in FIFO | Common /Independent Clock | Not Support | Block RAM | Dedicated Logic | Standard /First-Word First-Through |

From the above comparison, we can conclude that, Distributed RAM FIFO has the highest placement flexibility, built-in FIFO is the most suitable for simple data temporary storage, and only block RAM FIFO can make conversion of both clock domain and data width. So, we propose a novel timing-optimized FIFO structure as shown in Fig. 4. The FIFO structure adopts cascade connection of three basic types of FIFOs. The front part is a built-in FIFO, which makes temporary storage of input data. Considering the maximum capacity of a single FIFO in this design is about 2 MB, more required buffer memory needs cascade connection of several built-in FIFOs. The middle part is a block RAM FIFO, which makes clock domain and data width conversion. And the final part is a distributed RAM FIFO with small capacity, which makes data acquisition by PCIe IP core. Nearby FIFOs are connected by an 1-stage pipeline, and the prog.full port of the rear FIFO connects.
the rd_en port of the front FIFO through a timing false path, which can effectively decrease timing failure risks of delay paths crossing FIFOs.

![Timing Optimized FIFO Structure](image)

**Fig. 4.** A novel timing-optimized FIFO structure for PCIe DMA

Using the above structure, the timing results can be improved obviously. Fig. 5 shows the implementation design view of FPGA projects before and after optimization. We can find that much more memory resource (block RAM) in FPGA is utilized with the new FIFO structure, while no timing failing path exists. Table VII shows static timing analysis results of FPGA projects derived from Fig. 2 with different FIFO structures. By comparison, the FIFO structure in this paper has better timing performance than the others, and timing failure can be avoided even with 4 MB FIFO capacity, 100% block RAM utilization in FPGA.

![Implementation design view of FPGA project](image)

**Fig. 5.** Implementation design view of FPGA project (left: before optimization with 1 MB basic FIFO structure; right: after optimization with 4 MB novel FIFO structure)

| Table VII. Timing results comparison among different FIFO structures |
|---------------------------------------------------------------|
| FIFO Structure | Basic FIFO Used in | Preliminary Optimization Structure Used in | This Paper |
| Component      | [6, 7]             | [12]                                      |            |
| Capacity       | 524288B            | 1048576B                                  | 2097152B   |
|                | 1048576B           | 4153344B                                  |            |
| Block RAM utilization | 12.4%      | 25.7%                                     | 49.5%      |
|                | 25.7%             | 100%                                      |            |
| WNS            | 0                 | −0.29 ns                                  | −0.346 ns  |
| Timing result  | Succeed           | Fail                                      | Succeed    |

### 4.2 An optimized FIFO read control mechanism

Strategy 2 in Table V shows that timing failure exists in FIFO rd_en delay path, which is from axis.tx.tready port of PCIe IP core to internal FIFO logic. As shown
In Table V, there are large numbers of units driven by port axis.tx.tready (209 totally), which easily leads to high timing delay in the placement and routing of FPGA projects.

If axis.tx.tready signal can be predicted by user, the combinational logic (shown in Fig. 2) between rd.en and axis.tx.tready can be replaced by sequential logic which can remove axis.tx.tready signal from FIFO rd.en delay path and solve the relevant timing failure problems.

Accordingly, we propose a transfer waiting mechanism to make timing optimization, as shown in Fig. 6. When finishing a TLP transmission, the FPGA will wait for $N$ user clock cycle before a new TLP transmission. If $N$ is large enough, axis.tx.tready can be always valid in write request TLP transmission. So rd.en signal can be assigned by a register and doesn’t need connect with axis.tx.tready signal by combinational logic.

![Fig. 6. Process of transfer waiting mechanism for timing optimization](image)

Through PCIe DMA data stream test, if $N > 2$, axis.tx.tready will be always valid when rd.en is high. This method can improve FPGA timing performance, whose static timing analysis results are shown in Table VIII.

| Table VIII. Timing results comparison between different rd.en assignment way |
|---------------------------------------------------------------|
| **Rd.en control logic** | **Basic combinational logic for AXI port** | **Optimized sequential logic in this paper** |
| FIFO structure       | Block RAM FIFO                          | Block RAM FIFO                          |
| FIFO capacity        | 1048576B                                 | 1048576B                                 |
| WNS (ns)             | $-0.29$ ns                              | $0.07$ ns                                |
| Timing result        | Fail                                    | Succeed                                 |

4.3 Summary

In this section, we propose two FPGA timing optimization methods from the view of FIFO structure and FIFO read control logic respectively. The two methods can be adopted simultaneously in a single FPGA project without any conflict. In actual application, we prefer the optimized FIFO structure as the main measure to eliminate timing errors, and can choose to use optimized read control logic as an auxiliary measure when there is enough allowance in PCIe data throughput.

5 Performance test and target application of the novel PCIe DMA architecture

In this section, we make performance comparison on the transmission throughput, transmission latency, and required data buffer capacity before and after optimization. The test platform has been introduced in Section 2. The latency is measured by setting a counter driven by PCIe user clock in FPGA PCIe register. It starts to increase from zero at the beginning of each DMA cycle, and is read and recorded...
when the host polls the DMA done flag or enters the DMA done interrupt. Throughput and buffer capacity are measured in similar ways. The test results are shown in Fig. 7–Fig. 9.

![Fig. 7. Measured latency vs. data size per DMA cycle](image1)

![Fig. 8. Measured throughput vs. data size per DMA cycle](image2)

![Fig. 9. Required data buffer capacity vs. maximum instantaneous data rate](image3)

From Fig. 7 and Fig. 8, we can find that lower latency and higher throughput of data transmission is acquired after optimization. According to the measured maximum interval time between each DMA cycle, we get the required data buffer capacity curve shown in Fig. 9. Before optimization, using FPGA internal data buffer, the DMA structure can support transmission with only 80 MB/s maximum instantaneous data rate (the intersection point of the pink and green curve in Fig. 9). After optimization, it can support data transmission with more than 8 GB/s maximum instantaneous data rate.

The new architecture has realized FPGA-based high speed and large capacity PCIe data transmission using internal data buffer. On one hand, only using on-chip...
memory without adding off-chip memory can decrease the cost and design complexity of circuit board, on the other hand, on-chip memory can be configured and placed more flexibly, which is better suited to multi-channel parallel data transmission.

It has been applied in a wideband digital phased array radar system for multi-channel parallel data recording, which is shown in Fig. 10. The recording system can receive narrowband DBF data in 16 channels and wideband de-chirp data in 1 channel. Multi-channel data from different radar beams arrived in the same time range with different data rate, performing different functions like detection, tracking, anti-interference, and imaging. According to Table I, it will use at least 3 FPGA with 17 external memories, considering that the single-port external memory (QDR, DDR, etc.) controller can’t be shared by simultaneous multi-channel data with different rate. While using internal FPGA memory to flexibly build FIFO structure, only 1 FPGA can realize it, obviously reducing the cost and design complexity of circuit boards.

![Fig. 10. Application in a wideband phased array system](image)

6 Conclusion

This paper proposed a resource and timing optimized PCIe DMA architecture using FPGA internal data buffer. With higher data throughput and lower latency, the required data buffer capacity has decreases by 97.6% from 24.10 MB to 0.56 MB. Besides, the FPGA FIFO capacity without timing failure has increased from 524288B to 415344B with 100% FPGA internal memory utilization.

The novel PCIe DMA architecture realized high-speed data transmission using only FPGA internal data buffer. Without adding external memory chips (DDR3, QDR, etc.) for FPGA, it can reduce the cost and design complexity of circuit boards. Besides, it expands the application of PCIe DMA transmission based on FPGA, especially in multi-channel parallel data transmission when the external memory is not enough.

To facilitate readers’ validation and migration, this paper makes optimization based on a standard DMA process proposed by Xilinx with open source project. And the optimization method can be also applied in most PCIe DMA transmission situation, including PCIe gen1 to gen3, block and scatter/gather DMA mode, Virtex-5, Virtex-6, Virtex-7 and Ultrascale FPGA device.