The Driver's Design and Implementation of PCIe High Speed Image Acquisition Card Based on WDF

Qiang Wu [1,a,*], Liang Xu[1,b] and Xuwen Li[2,c]

[1] College of Information and Communication Engineering, Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China
[2] College of Life Science and Bioengineering, Beijing University of Technology Beijing, 100124, China
[a] email: wuqiang@bjut.edu.cn, [b] email:xuliang10521119@163.com, [c] email: lixuwen@bjut.edu.cn
*Corresponding author

Keywords: Windows Driver, PCIe Bus, WDF Model, DMA Transmission, Ring Buffer.

Abstract. As a common high-speed interface, PCIe interface is very extensive in the field of data acquisition[1]. As a development framework of Microsoft, WDF (Windows Driver Foundation) driver model can reduce the difficulty of the development. Based on the PCIe high-speed image acquisition card which is developed by ourselves, this paper designed a stable and reliable WDF driver for the high data transmission rate in the image acquisition process, and used the memory allocation method in WDF to share buffer with the application. In order to enhance the speed of the image acquisition, this paper designed a ring buffer mechanism to ensure the continuity and stability in the image acquisition and transmission process.

1. Introduction

Throughout the software development process, the driver communicated with the underlying hardware directly, and transferred the instructions of application and data requests to the hardware device. The driver played a very important role in the entire software process, and affected the efficiency of data transmission across the link and the integrity of the data. Since Windows 2000, the development of the driver must be based on WDM, but its development was difficult, we cannot expect the development to be easy as user mode application. To improve this situation, Microsoft introduced a new driver development environment WDF, which has the following characteristics.

(1)The development environment introduced the object-oriented technology. Microsoft had an initial concept of object-oriented in the WDM driver model. In a WDF development environment, Microsoft abstracted the concepts of drive queues, IO requests, memory, DMA adapters and so on. It also provided a good package, and defined the properties, methods, and events for each model.

(2) WDF provides a unified development environment for the kernel mode driver and user mode driver. Kernel mode and user mode objects are derived from the WDF. The kernel mode driver uses the KMD Framework; the user mode driver uses the UMD Framework.

(3) WDF provides some encapsulation for the driver's common behavior, such as plug and play, power management, and so on. For example, developers write plug-and-play and power management code at least thousands of lines, these processing will become the default behavior after using WDF. Compared with the implementation efficiency, it is not inferior to the original WDM driver.

By studying the basic framework of WDF, this paper designed the driver of PCIe high-speed image acquisition card based on the WDF development environment, and the driver system developed by the driver is Windows 7 32bit.

2. The Basic Analysis of Driver Development

2.1. The Hardware Structure of Acquisition Card

The hardware platform is mainly a dual LVDS image acquisition board which is studied by myself. The core of the board is Xilinx's Kintex7 series XC7K70T FPGA processor, which uses Xilinx's official PCIe hard core[2] to communicate with the driver. As a bridge between the application and the underlying hardware, the driver is mainly responsible for receiving the control commands from the application program and the image data from the FPGA, selecting the set image type by command, and using DMA to move the data transferred to the FPGA to the upper application according to the size of the image. The link block diagram of the Image acquisition card is shown in Figure 1.

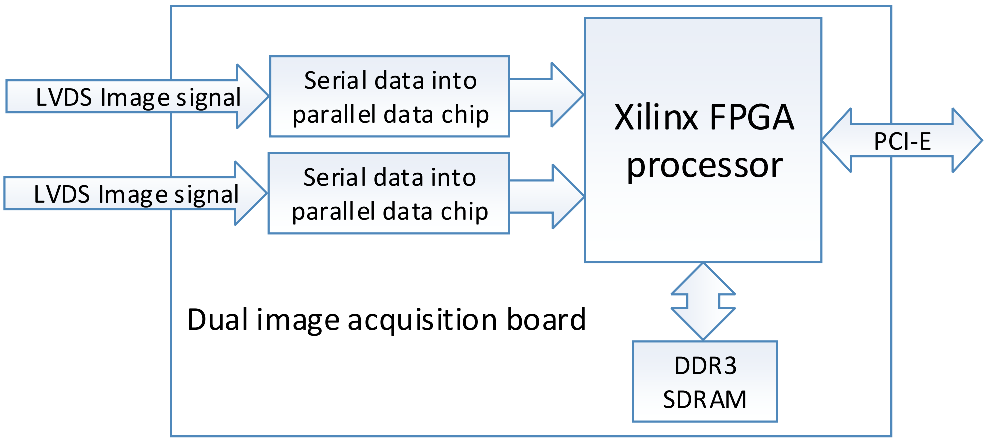
LVDS Image signal
Serial data into parallel data chip
LVDS Image signal
Serial data into parallel data chip
Xilinx FPGA processor
PCI-E
Dual image acquisition board
DDR3 SDRAM

Figure 1 The link block diagram of the Image acquisition card.

2.2. The Basic Analysis of Driver Software

Before we develop the driver, we need to analyze the basic functions needed to implement the driver and the key points to analyze.

Basic functional analysis:

(1)Hardware interface of the driver: PCIe bus interface was used in hardware design, and the driver only need to modify the official standard PCIe function driver routines.

(2) Instruction interaction between driver and hardware device: The core of the hardware board is the Xilinx Kintex7 series XC7K70T FPGA processor. PCIe hard core was been used to configure the PCIe configuration space directly, information can be read by the driver; and the definition register added in the FPGA can be mapped to the PCIe memory space, and the Instruction interaction between driver and hardware device was implemented by a custom register.

(3)Instruction interaction between driver and the upper software: The driver needs to receive and respond the control commands when the application set to start, stop, data size, and so on, WDF has packaged these functions into callback functions, see Table 1

Among those, CreateFile and EvtDeviceFileCreate can open the device. CloseHandle, EvtFileCleanup, and EvtFileClose can close the equipment. DeviceIoControl and EvtIoDeviceControl can achieve a small amount of data in two-way transmission after opening the device.

Table 1 Functions to interact with the upper software instructions.

Win32 function	KMDF Callback function
CreateFile	EvtDeviceFileCreate
DeviceIoControl	EvtIoDeviceControl
CloseHandle	EvtFileCleanup EvtFileClose

(4) Data transmission between the driver and hardware equipment: For the acquisition card, the underlying hardware data is transmitted to the driver layer software by using the DMA. Because of uncertainty of the acquisition time and the short interval between the data, we need to use interrupt processing , and move the data which is been transferred from DMA to the drive layer into the computer memory in the DPC [3](Deferred Procedure Call).

(5) Data buffering during transmission: During the data transfer process, data transmission is very frequent and is little possible to be interrupted. For non-real-time operating systems such as Windows, it is not practical to move the data which is collected by driver layer in real time to application layer, so we need to use the data cache mechanism in this driver software part. Data collected to the driver layer would be cached, and the cache data need to achieve the design quantity of the data before uploading the data.

(6) Data transmission between the driver and the upper software: Using the shared memory mechanism to read and write data directly can reduce the number of data copies during the data transmission process. The driver creates and locks the buffer of the required space, and maps the physical memory to the application software so that the buffer for the application operation and the buffer for the driver operation are the same address. In this way, the process involved is more complex compared to a simple memory copy, but more efficient, and it is suitable for large amounts of data.

3. The Driver Design of Acquisition Card

In the driver development process of the acquisition card, we need to consider the access of hardware interface and application, data transmission efficiency and so on. Among those, it involves many issues, includes the initialization of the driver entry function, the creation of DMA adapter interrupt and other resources, the release of DMA controller[4] to release memory and other resources, the use of shared memory area, data buffer in the data transmission process and so on. This paper designed the acquisition card driver software, which was characterized by the use of shared memory to transfer data to the upper layer of the software, using the ring buffer in the driver layer increased the driver's data acquisition capabilities, and facilitated the use of application software by packaging the interface ultimately.

3.1. The Design of Shared Memory Works

In order to improve the efficiency of data transmission and reduce the memory copy, we used the shared memory work way to communicate image data between the driver and the upper software. Once there was a lot of data needed to be interacted between the application and the driver, we should avoid the direct copy of large amounts of data as much as possible, we'd better use the memory mapping method, so that the application and the driver shared the same memory. Figure 2 shows the distribution relationship between the shared memory in the driver and the mapping memory.

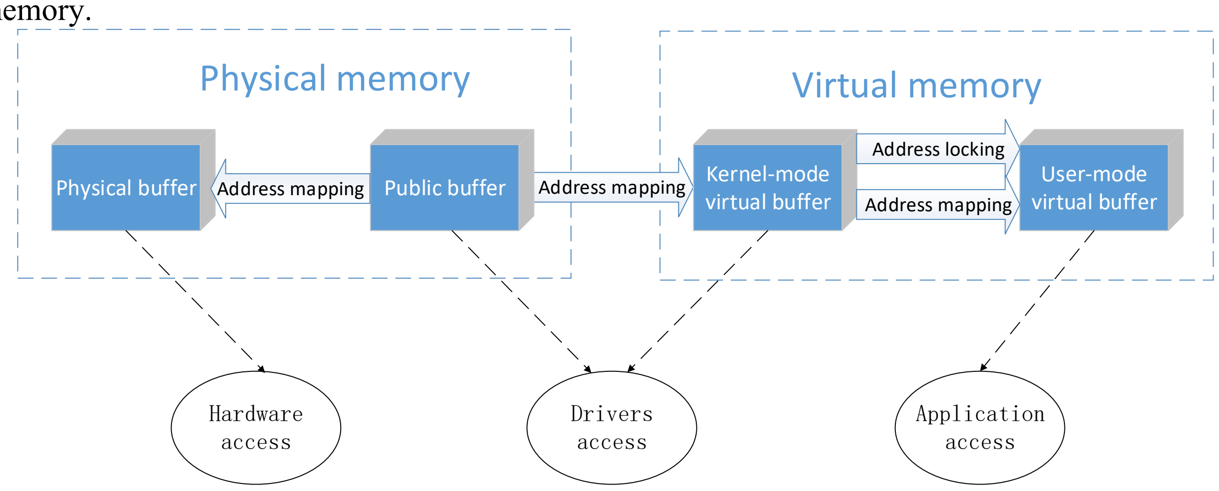

Figure 2 storage space map.

In order to ensure that the data address would not change, the project opened up memory in the driver, and passed the memory pointer to the application. Among this project, we received 1024 lines image data, and each line had 1024 bytes. In order to increase the effect of buffer, we opened up a public buffer as 10 times the amount of data.

The public buffer is a non-paged physical memory in Windows memory. We need to call the MmBuildMdlForNonPagedPool to manipulate the memory descriptor, and its physical address points to the hardware peripheral directly, and using WRITE_REGISTER_ULONG function can pass it to the peripheral for the access of Hardware directly; and the public buffer can be accessed through the kernel address in the kernel. In the kernel, through the memory descriptor Mdl and mapping function MmMapLockedPagesSpecifyCache, the virtual address of the kernel can be mapped to user mode Virtual address. User mode virtual address is directly for the application, by using the EvtIoDeviceControl, the address can be passed to the application for direct access to the application. After use, the driver needs to use MmUnmapLockedPages to release shared memory. If not released, once again to access these buffers, it may lead to system crashes.

3.2. The Design of Ring Buffer

In the design of the acquisition drive, we need to acquit the continuous image, and stored on the hard disk. Because the computer memory was very small compared with the long image data, it's necessary to do the caching operation. At the same time, Windows isn't real-time, and the speed of writing the hard disk isn't stable, it cannot guarantee that the collected data can be written to the hard disk immediately. So we designed the driver layer by using the buffer mechanism to cache the data, and stored a piece of data in the application layer, and then stored the next piece of data to the memory.

In most acquisition systems, double buffering is widely used currently[5]. The implementation process are as follows: the driver allocates a shared memory for acquisition, and then divides the shared memory into two equal shares. In the data acquisition process, the data goes into the front area buffer firstly, filling the back area after filling full the front area. This process will always judge the two buffers, the area filled full with data is copied to the transmission buffer of the application, which is double buffering mechanism. But in this copy process, if the data has been full filled in another area, and the copy of the district is still copying, it will cause conflict access, data corruption, and it will difficult to ensure integrity and correctness of the data.

Because the transmission rate of the PCIe data was very fast, and the image was 1024 * 1024 bytes, so we applied a number of common buffer in the drive design, by using the ring buffer, it can prevent the memory block conflict. Ring buffer was an improved scheme of double buffering. The two key points of this design are as follows:

First of all, real-time data acquisition and data transmission need to work in parallel. The data acquisition processed in the interrupt service and the delay call can respond quickly to the hardware transmission interruption. The data transmission was processed in the IRP_MJ_DEVICE_CONTROL dispatch routine. The virtual address and data length can be transmitted directly to the application.

The second point is that the mode of operation is still used double-buffered memory, and divide it into public buffer of the drive layer and storage buffer of the application layer. The difference was that the transmission buffer of the drive part was divided into multiple partitions, the size of each partition was the same as storage buffer of the application layer.

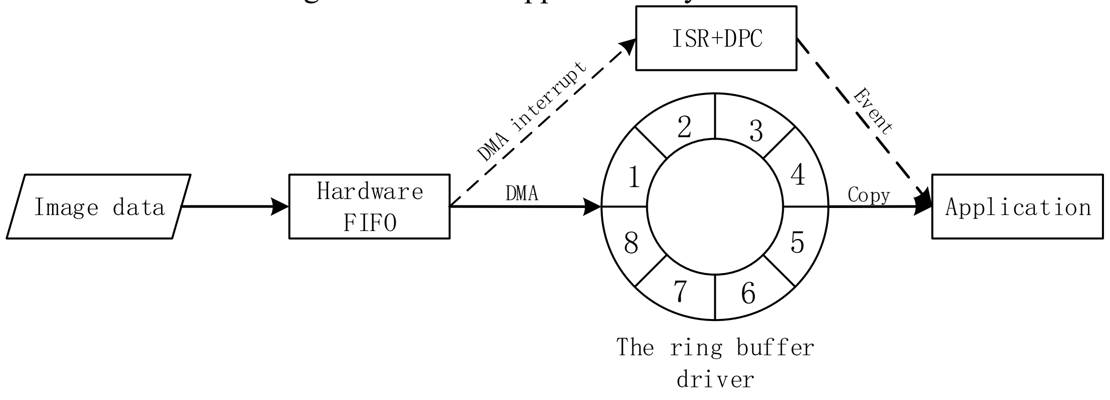

Figure 3 Schematic diagram of ring buffer operation.

In this design, we opened up eight buffer areas, and the size of each buffer was 10 times as large as the image data. In the driver part, the PCIe DMA moved the data to ring buffer 1 of the drive, using the interrupt_count signal as a Count flag of image collection. Each time the data in the driver's interrupt (ISR) and interrupt latency service function (DPC) is accumulated, when the data in buffer 1 is full, the ISR and DPC would inform the DMA controller to move data to the buffer 2. At the same time, by sharing the notice of the events, the application can do the data copy, data storage, and other work from the buffer zone 1. Here, we can allow a certain time instability of the application. And so on, we completed the data collection. Each sub-buffer of the ring buffer and the size of the buffer needed to be considered based on the image transmission speed and size. It's necessary to ensure the integrity of the data but also to save the non-paging memory of drive layer.

3.3. The Design of Driver Interface Function

In order to facilitate the access to the baseband device of upper application, we used visual Studio to develop interface functions of driver. We also encapsulated many functions, such as device creation, register read, register write, DMA read shared buffer, receive parameter configuration, close the device and so on. The driver interface functions are as follows:

(1) **The name of function:** HANDLE hOpen_RevBoard (BYTE CardId);

Functions description: Find the board and allocate the board resources;

Parameters description: CardId, the value ranges from 0 to 7 (if the same 8 cards are inserted into the IPC at the same time, according to the distance from the board slot to CPU, the number of the board is numbered 0, 1, ..., 7; if only one board is inserted into the PC, the board number is 0).

Return Value: Returns a HANDLE type handle for subsequent operation of the device. If NULL is returned, the board fails to open.

(2) **The name of function:** HANDLE hSetRevEvent (HANDLE hDriver);

Functions description: Sets the receive completion event

Parameters description: hDriver, Enter a board handle

Return Value: Returns a HANDLE type handle that is used to configure the receive function. If the return is NULL, the setup completion event fails.

(3) **The name of function:** bool hStartRev_RevBoard (HANDLE hDriver, int ImageSize);

Functions description: Starts the receive function

Parameters description: hDriver, enter a board handle, ImageSize is the receiving image data size that is inputted before (in bytes).

Return Value: true if set true, false after setting failed.

(4) **The name of function:** DWORD hReadData_RevBoard (HANDLE hDriver, LPVOID pCommonBuffer, int DMATransferSize, HANDLE m_hDMAEvent, int timeout);

Functions description: Starts the receive function

Parameters description: hDriver,Enter a board handle. PCommonBuffer, Store the data buffer pointer address. DMATransferSize, The amount of data read from the buffer at a time. m_hDMAEvent, indicates the previously received event (when received 10 frames data will trigger this event). Timeout is overtime, that is how long does it not trigger 10 frames receiving event would automatically implement this function. If less than 10 frames, it would fetch out of the dissatisfaction data and return the quantity of data. If there is no data, the function will return 0, it indicates that no data has been received.

Return Value: Returns a DWORD type of data, representing the byte size of the received data.

(5) **The name of function:** bool hClose_RevBoard (HANDLE hDriver, HANDLE m_hDMAEvent);

Functions Description: Close the board function, release the board resources, and release the shared event resources.

Parameters description: hDriver, Enter a board handle. m_hDMAEvent, Sets the receiving event

Return Value: Returns true for shutdown success, false for shutdown failed.

4. Conclusions

Aiming at the requirement of high performance of drive transmission in the process of image acquisition card, this paper has designed and implemented a PCIe interface driver based on WDF driver framework. The driver, as the middle part of the whole system data link, using the shared memory and ring buffer method, provided a stable and reliable transmission link for the image data acquisition and ensured the correctness of the data by a large number of tests. By packaging DLL interface and developing the call for the driver interface of the application routines, we ensured the high-speed data transmission and the stability of entire data link.

References

[1] Bu K, Xu H, Sun Z L, et al. A High Speed and Compact Data Acquisition and Storage System[J]. Advanced Materials Research, 2013, 753-755:3125-3128.

[2] Mauch S, Reinlein C, Beckert E. FPGA-accelerated adaptive optics wavefront control part II[C]. SPIE Photonics West. 2014:121-134.

[3] Wei Y I, Xin X U, Sun Z L. Development of Driver for PXIe Port based on WDF[J]. Microprocessors, 2011.

[4] Jin Y J, Xiong J P, Shuai S I. Design of real-time data acquisition and processing system based on DMA way[J]. International Electronic Elements, 2008.

[5] Meng S, Lu J. Design of a PCIe Interface Card Control Software Based on WDF[C]. International Conference on Instrumentation & Measurement. 2015:767-770.