The Method of Isochronous Cycle Duration Measurement for Serial Interface IEEE 1394A

Michał Sawicki(✉) and Andrzej Kwiecień

Institute of Informatics, Silesian University of Technology, Gliwice, Poland
{msawicki,akwiecien}@polsl.pl

Abstract. On one bus IEEE 1394A may be a lot of protocols (eg. IIDC and SBP-2) that interact. On this bus cycle jitter may occur, which is not desired in A/V systems. This paper presents a method for measuring isochronous cycle duration. This method allows detection of cycle jitter. It is based on dedicated IEEE 1394 Device Driver and do not require reorganization of a topology of communication system. This article presents the results of measurement of cycle duration in communication system under test.

Keywords: Serial interface · IEEE 1394A · FireWire
Isochronous transfer · Cycle jitter

1 Introduction

A computer system can contain many different devices that communicate via the same interface. Therefore, the protocol designer must take into account the interaction between different communication protocols existing on the same bus. An example of this situation is a vision system based on the IEEE 1394 bus (FireWire), which includes recording devices and external mass storage. Figure 1 shows a system consisting of a FireWire camera, a computer and an external mass storage (hard drive) equipped with a FireWire port. There are two independent transfers executed in this case:

- an isochronous image transfer from camera to computer workstation supervised by IIDC protocol [2],
- an asynchronous data transfer between computer workstation and external mass storage under SBP-2 protocol [3].

It is the communication system with the single FireWire bus over which two communication protocols (IIDC and SPB-2) exist and operate with different types of data transfers (isochronous and asynchronous). The asynchronous transfer may influence the isochronous transfer and disturb the regularity of

P. Gaj et al. (Eds.): CN 2018, CCIS 860, pp. 206–215, 2018.
https://doi.org/10.1007/978-3-319-92459-5_17

isochronous data supply, that is basic requirement imposed on transfer of images. An example of this effect is the oscillation of isochronous cycle duration (called cycle jitter).

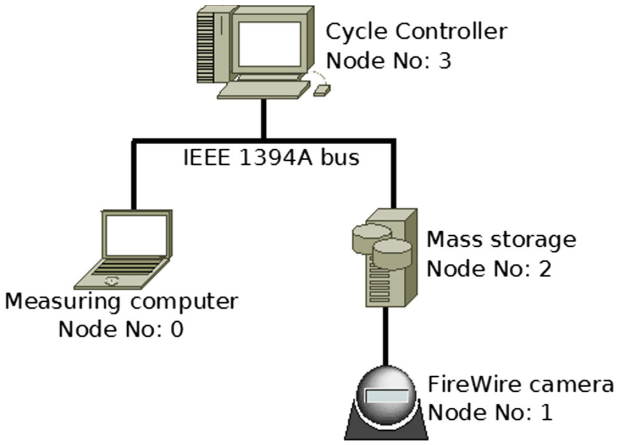

Fig. 1. Scheme of the FireWire communication system

This article presents a method of the cycle jitter detection. It allows to estimate the isochronous cycle duration and detect its oscillation. The first part of this paper contain characterization of isochronous cycle and explanation of cycle jitter. The second part presents a method of measuring the isochronous cycle duration and describes the tools used for this purpose.

Some related works, which presents the analysis of FireWire protocol, can be found in [7].

2 Isochronous Transfer

The IEEE 1394A standard defines two types of data transfer [6]: asynchronous and isochronous. Asynchronous data transfer is used to communicate with mass storage [12] and it is not subject of this article.

Isochronous transfer provides the regularity of data supply (data delivery to a receiver at regular intervals called isochronous cycles). In addition, isochronous transfer provides correctness control of received data, however corrupted data are not retransmitted.

Isochronous transfer is used to deliver short-life data, like video data in the system in Fig. 1. FireWire port allows only isochronous write operation. Operations are assigned to isochronous channels (time slots), in which they are executed [4]. These operations allow to write the same data to one or multiply devices simultaneously.

2.1 Isochronous Cycle

Isochronous data transfer is divided into smaller units (called transactions) executed within the 125 μs isochronous cycles (Fig. 2). Each transaction delivers basic amount of data specified by buffer's BytesPerFrame parameter [8]. At the beginning of each isochronous cycle, the Cycle Master (device located on the top of FireWire tree topology) generates a cycle start packet (CSP). Then, by the isochronous arbitration process nodes are selected to transmit data in successive isochronous channels. After completion of the isochronous transactions, if in the cycle still is free bandwidth [7], the asynchronous data transactions are executed (asynchronous transfer is also divided into transactions).

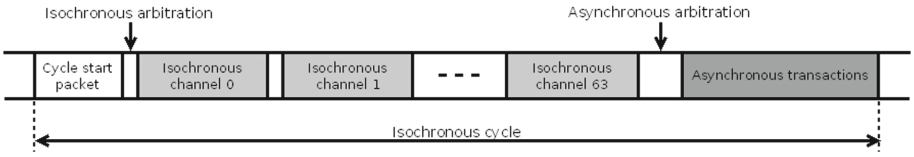

Fig. 2. Isochronous cycle in the FireWire port

2.2 Isochronous Cycle Jitter

In the FireWire interface may occur oscillations of the interval between successive transactions of the same isochronous transfer (Fig. 3). These oscillations are due to the shift of the beginning of the next cycle caused by the last asynchronous transaction extension in the previous cycle.[1] The cycle jitter is undesirable in audio-visual (A/V) systems.

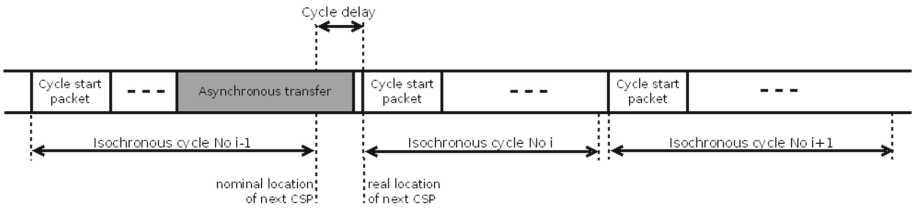

Fig. 3. Isochronous cycle jitter

[1] The requirement on the FireWire bus is not to break but to complete an asychronous transaction, which started before the end of nominal ischronous cycle.

2.3 Isochronous Arbitration

Isochronous arbitration is a FireWire bus access method [9] applyed by nodes that want to send isochronous data and is executed during gaps in the transmission (when bus is at idle). After the CSP broadcast, isochronous nodes, which initiate write isochronous requests, take part in the isochronous arbitration. After completion of the arbitration the node is selected, which can access the bus to execute one isochronous transaction (one isochronous channel).

After completion of the isochronous transaction next nodes compete for bus access. During isochronous arbitration, competing nodes indicate the desire to transmit (generate interface state TX_REQUEST). This signal is passed through successive layers of the communication system topology, up to the root (node located in the first layer). The root grants access the bus for node which TX_REQUEST comes first to one of its ports.

It is obvious that the nodes located closer to the root have precedence in access to the bus. This feature was used in the method described in Sect. 4.1.

3 Software Access to the FireWire Port

Communication between user's application and a device requires a specialized driver, which is installed on operating system. In Microsoft Windows, device driver works in kernel-mode, so the user application (which works in user-mode) does not have programmable access to the device. Therefore, applications need a special diver to communicate with the device. This driver provides an API for the programmer to manage the device.

VHPD1394 device driver (Versatile High Performance IEEE 1394 Device Driver) ensures communication between user application and device via FireWire bus, allowing multiple FireWire devices and multi-user application at the same time.

3.1 Architecture of VHPD1394 Device Driver

VHPD1394 device driver is one of many modules in the FireWire stack on Microsoft Windows (Fig. 4). It cooperates with FireWire bus driver, which provides an interface for FireWire device drivers. On the other hand VHPD1394 provides software interface for user's application.

This software interface allows to perform isochronous transfers (sending data in isochronous channel) as well as receiving data in the indicated isochronous channel. If packets of the same transfer are transmitted in the following cycles, then data is received with frequency 8 kHz. Microsoft Windows is not Real-Time operating system, so VHPD1394 groups received isochronous packets in the streaming buffers. User's application does not process single packet but processes single buffer. This buffer does not contain packets, which have been damaged.

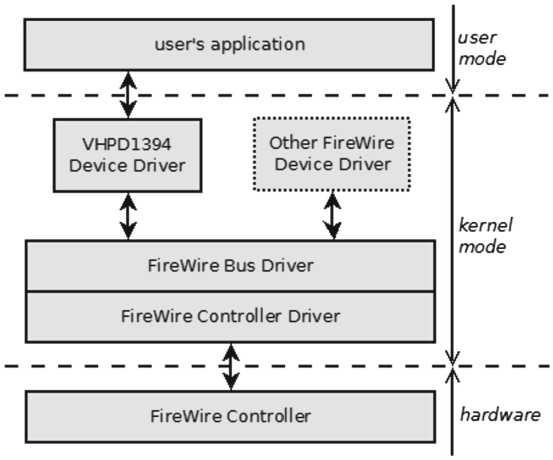

Fig. 4. IEEE 1394 driver stack in Microsoft Windows with VHPD1394

3.2 Data Transmission Using Buffer Queue Streaming Mode

Isochronous data exchange between user's application and VHPD1394 may be performed in the Buffer Queue Streaming Mode. In this mode, the application allocates a buffer pool (Fig. 5), which contains at least two buffers. Received isochronous packets are sent to the currently active buffer. When buffer is full of packets VHPD1394 switches (rotates) buffers: filled buffer is passed to the user's application and another empty buffer from pool is passed to the VHPD1394 driver. The application is notified of rotation of buffers in the pool. Buffer processed by the application is returned to the buffer pool and waits to fill by VHPD1394 driver. In this way it is possible to maintain a continuous flow of isochronous data. In a similar way data may be transmitted in isochronous channel.

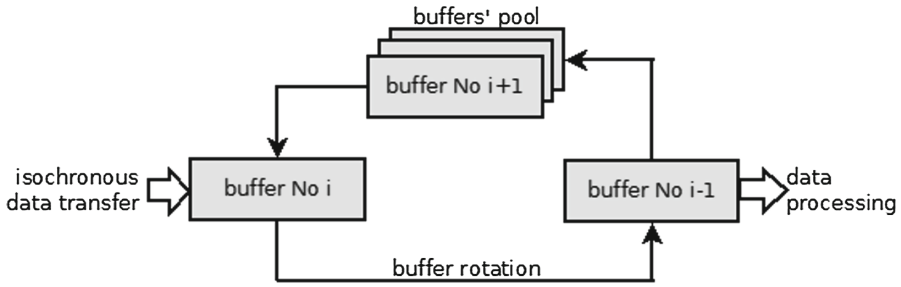

Fig. 5. Buffer Queue Streaming Mode

4 Isochronous Cycle Duration Measurement

To detect cycle jitter, cycle duration must be measured. If the measured time differs from the nominal value (125 μs) then cycle jitter occurred on the FireWire bus.

4.1 The Method of Isochronous Cycle Duration Measurement

Each node has a 32-bit counter Cycle_Time_Register, which is in the address space of the node [1] and is incremented at frequency of 25 MHz. At the beginning of the isochronous cycle Cycle Controller broadcasts CSP packet using asynchronous write broadcast. After receiving CSP packet each node synchronizes its local cycle time (value of Cycle_Time_Register) with time passed as CSP data [5]. Thus nominally every 125 μs each node performs synchronization cycle time with Cycle Controller. Therefore it is not possible to directly read cycle duration on the local node only on the basis of the value of its Cycle_Time_Register.

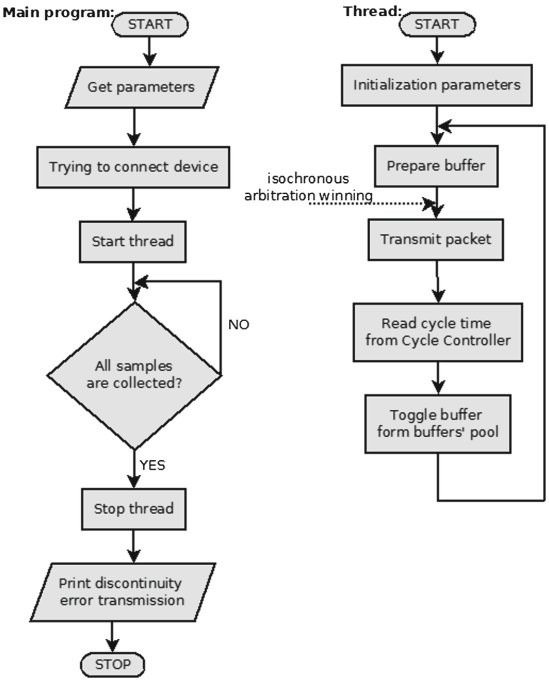

Fig. 6. Block diagram of described method

This problem can be solved by adding the node responsible for measuring the cycle duration to the existing communication system. This node generates

isochronous packets with smallest data field (4 bytes) using Buffer Queue Streaming Mode and simultaneously measures the duration between successive rotations of buffers in the pool. Buffers contain only one isochronous packet. Figure 6 shows described method (as block diagram). This method has been implemented in the sample FireWire communication system (Fig. 1) using multithreading and VHDP1394 Device Driver.

This node should be connected close to the system root (Cycle Controller), because immediately after CSP packet broadcast it must win isochronous arbitration process and get bus access. Then it performs one isochronous transaction and measures a moment of the beginning of cycle (moment of the buffers rotation in the pool). After each two cycles we get cycle duration by subtracting two measured values of cycle time.

4.2 Cycle Time Measurement in the Communication System Under Test

The communication system under test (Fig. 1) consisted of four nodes: one asynchronous (external hard drive), two isochronous (Cycle Controller and FireWire camera) and one node responsible for cycle duration measurement (Measuring Computer).

In the first part of the study cycle duration without asynchronous data transfer between external hard drive and computer was measured. Figure 7 shows the distribution of measured isochronous cycle duration (the first graph). The measured values are focused on the nominal cycle duration (125 μs), what could be expected. This distribution is unimode, that confirms the absence of the cycle jitter.

Microsoft Windows XP Professional was installed on measuring computer. This operating system is not Real-Time system. If user's application does not prepare a buffer before the rotation of buffers in the pool, then transmission discontinuity errors may occur. Therefore, for a sufficiently small number of buffers in the pool these errors may occur. Before the cycle duration measurement, the suitable (minimum) number of buffers in the pool was set, to protect against discontinuity errors. The minimum number of buffers depends on a computer, and may be different for various computers.

In the second part of the study cycle duration with asynchronous data transfer between external hard drive and computer was measured. In this case, cycle jitter may occur. Figure 7 shows the distribution of measured isochronous cycle duration (second graph). This distribution is multimode which confirms the existence of the cycle jitter.

The Fig. 8 shows the cycle duration for two sizes of data field in isochronous packet generated by measuring computer. Increasing size of data field reduces available bandwidth for asynchronous data transfer. Therefore, for larger data packet cycle duration extension is greater than for smaller one. The buffer size was set to 4 bytes to protect FireWire bus against cycle jitter caused by too large buffer in measuring computer (computer generating isochronous packets).

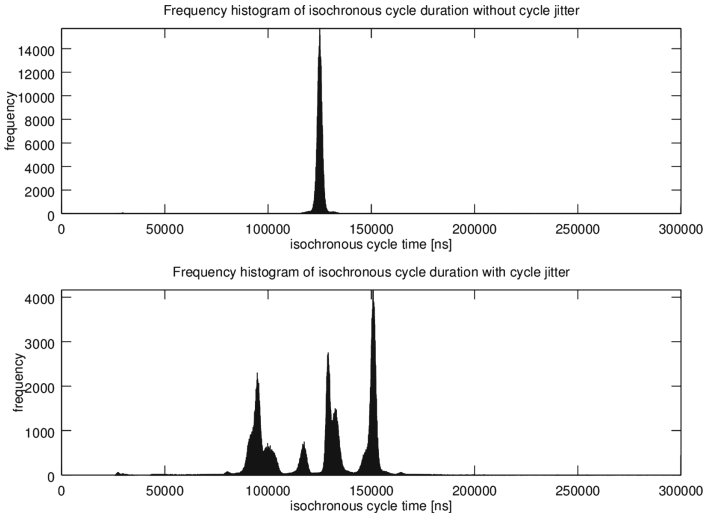

Fig. 7. The histogram of isochronous cycle time

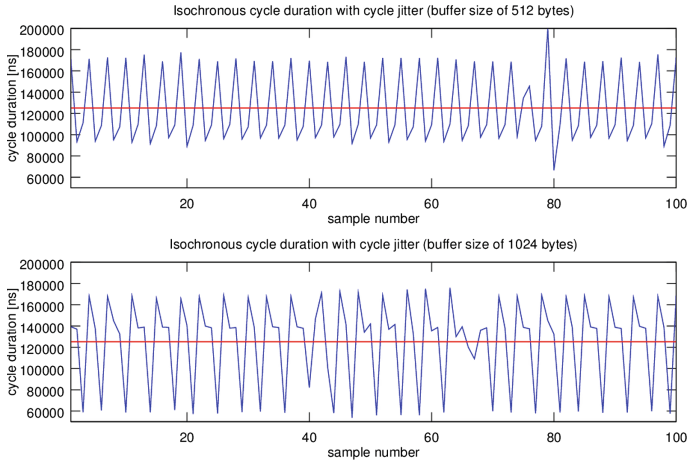

Fig. 8. Isochronous cycle duration with cycle jitter

5 Conclusions

The presented method allows to measure isochronous cycle duration and detect the cycle jitter on the FireWire bus. The cycle jitter is the effect of asynchronous transaction that was not completed within nominal isochronous cycle. Usually it happens, when different communication protocols are active at the same time on the FireWire bus. Cycle jitter is undesirable in communication systems [10] (e.g. A/V system), which use isochronous transfer.

This method uses dedicated IEEE 1394 Device Driver and do not require reorganization of a topology of FireWire bus. This is the advantage of this method, because cycle duration measurement and cycle detection do not require expensive protocol analyzers [7,11]. This method uses the Buffer Queue Streaming Mode, which allows detection of the start of isochronous cycle and register cycle time.

Another way to measure the isochronous cycle (and cycle jitter) is to use an expensive protocol analyzer. However, the method presented in the paper does not require the purchase of FireWire hardware analyzer.

The proposed method has been implemented and tested in a real communication system (Fig. 1). Cycle duration measurement was performed for two cases: without asynchronous transfer and with asynchronous data transfer between a computer and external hard drive, which allowed for cycle jitter detection in communication system under test (Fig. 7).

References

1. IEEE Std 1394A–2000: IEEE Standard for High Performance Serial Bus
2. IIDC2 Digital Camera Control Specification Ver. 1.0.0 (2012)
3. Serial Bus Protocol 2 Specification
4. Anderson, D.: FireWire System Architecture, Mindshare. Inc. Addison-Wesley Developers Press, Boston (2000)
5. Park, S., Jang, I., Lee, S., Choi, S., Cho, K., Lee, J.: Improved cycle time synchronization method for isochronous data transfer on wireless 1394 network. In: The 2008 IEEE International Conference on Ultra-Wideband, Hannover (2008)
6. Zahariadis, T., Pramataris, K.: Multimedia home networks: standards and interfaces. Comput. Stand. Interfaces **24**(5), 425–435 (2002)
7. Steinberg, D., Birk, Y.: An empirical analysis of the IEEE-1394 serial bus protocol. IEEE Micro **20**(1), 58–65 (2000)
8. VHPD1394 Versatile High Performance IEEE 1394 Device Driver for Windows Reference Manual, Ilmenau (2010)
9. Mielczarek, W.: Digital serial bus FireWire. Silesian University of Technology, Gliwice (2010)
10. Domański, A., Domańska, J., Czachórski, T., Klamka, J.: The use of a non-integer order PI controller with an active queue management mechanism. Int. J. Appl. Math. Comput. Sci. **26**(4), 777–789 (2016)

11. Maćkowski, M.: The influence of electromagnetic disturbances on data transmission in USB standard. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2009. CCIS, vol. 39, pp. 95–102. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02671-3_11
12. Sawicki, M.: Analysis of asynchronous data transfer in communication system models for USB and FireWire. In: Scientific Conference Computer Networks 2012, Studia Informatica, vol. 33, no 1A (107), Szczyrk (2012)