Statistical Analysis of Time Delays in USB Type Sensor Interfaces on Windows-based Low Cost Controllers

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Lalitha Ramadoss

Certificate of Approval:

Thaddeus A. Roppel
Associate Professor
Electrical and Computer Engineering

John Y. Hung, Chair
Professor
Electrical and Computer Engineering

Robert N. Dean, Jr.
Assistant Professor
Electrical and Computer Engineering

George T. Flowers
Dean
Graduate School

# Statistical Analysis of Time Delays in USB Type Sensor Interfaces on Windows-based Low Cost Controllers

Lalitha Ramadoss

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 19, 2008

Statistical Analysis of Time Delays in USB Type Sensor Interfaces on Windows-based Low Cost Controllers

Lalitha Ramadoss



Signature of Author

Date of Graduation

## Thesis Abstract

## Statistical Analysis of Time Delays in USB Type Sensor Interfaces on Windows-based Low Cost Controllers

Lalitha Ramadoss

Master of Science, December 19, 2008
(B.E., Bharathidasan University, 2000)

97 Typed Pages

Directed by John Y. Hung

Real-time control systems need sensor data at periodic and predictable time intervals-this is one aspect of what is commonly called "determinism". Common forces that present challenges to determinism in Windows-based control systems are computer hardware, software, operating systems and external hardware interfaces. Popular, low cost interfaces that are deployed for attaching peripheral devices and sensors to computing platforms are Universal Serial Bus (USB) and Peripheral Component Interconnect (PCI).

This work examines the latency of such buses in Windows-based systems from a statistical perspective. Experimental evaluation reveals that PCI and USB offer different levels of performance. With the addition of more and more devices to any of these buses, overhead increases resulting in deterioration of determinism. This study looks at the issue of achieving predictability with a broader perspective of why and to what extent a real-time system is unpredictable.

## Acknowledgments

I would like to thank my committee members for their suggestions and contributions to this work. I would especially like to thank my advisor, Dr. John Y. Hung, for his patience and invaluable assistance. I would like to thank all of my friends for their help and support.

I would also like to thank my family, particularly my parents, Bhavani and Ramadoss. Without their love and support, I would have been unable to complete this.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

## List of Figures

# Chapter 1

## Introduction

Real-Time Systems are those in which the correctness of the system depends not only on the logical results of the computation, but also on the time at which the results are produced. Meeting timing constraints is a critical issue in real-time control systems. There are certain obstacles a real-time engineer has to be concerned about while designing real-time systems. These problems cannot be avoided completely, but it is better to be aware of pros and cons for implementing better real-time systems in the future. This study provides a set of recommendations for real-time developers and users to consider while building and using low-cost, Windows-based, real-time applications.

Most literature refers to two classes of real-time systems: hard and soft [4] [13]. A hard real-time system is one in which missing deadlines is not tolerated and leads to system failure. A soft real-time system is one in which deadline misses may lead to performance degradation but not system failure.

For example, shown in Fig. 1.1 is an autonomous robot-trailer system designed for geophysical mapping of the ground. The system is designed to operate in both autonomous and remote control modes. In the latter case, the system sends video information to a remote computer, and also responds to keyboard/mouse/joystick commands sent from the remote location. The control computer also can save data to a file for later analysis. A suite of onboard sensors is connected to the main control computer via standard USB interfaces. During system development, the latency of sensor information became an issue.

The 3 major objectives of this research are:


Figure 1.1: An autonomous robot-trailer system conducting geophysical mapping task

1. Explore the domains that cause latency in low-cost Windows-based real-time systems. Some of the issues that affect determinism in real time systems are computer hardware, computer software, operating system and external hardware interfaces. A real-time system designer strives for predictability with an appropriate mix of these that is cost effective.

2. Evaluate the suitability of USB for real-time applications. The behavior of different peripheral devices like keyboard, mouse, USB drive and webcamera with respect to USB and USB hub is studied from a latency perspective. The latencies caused by USB and USB hub were measured and then compared to determine whether USB hub adds overhead to USB latency.

3. Determine whether PCI is better than USB for achieving predictability in real-time systems. Another question that is studied while working on this research is, "When multiple devices are connected to the system, does operation of one device affect the latency of another device?"

Experimental results summarize the latencies caused by USB, USB hub and PCI interfaces. It can be inferred from data that they offer different levels of performance in meeting timing constraints. Statistical analysis of sample data was done in two distinct ways: one to fit the data with an appropriate distribution function which would give some insight into modeling the system and the other to compare the means of data with T-tests.

Chapter 2 discusses the domains that affect determinism in low-cost Windows-based real-time control systems. Chapter 3 describes the analysis made on latencies measured with USB and USB hub on different peripheral devices. A study on PCI and comparison of USB and PCI latencies with two devices communicating to the system at the same time

is explained in Chapter 4. Chapter 5 concludes the thesis with inferences and a sketch of relevant future work.

# Chapter 2

## Challenges to determinism in low-cost real-time control systems

An extensive analysis in the areas of computer hardware, software and operating system helps a real-time designer to gain deeper understanding of limitations imposed by them on real-time applications. An overview of these challenges is presented in this chapter; an experimental example helps illustrate the deadline miss that can result.

### 2.1 Introduction

New powerful processor architectures optimize the overall performance but an analysis of several hardware features like Instruction Set Architecture, system bus speed, caching, parallelism and pipelining is necessary to determine its suitability for real-time performance. Microsoft Windows series of operating systems are becoming more common in real-time applications because many developmental tools and techniques are widely available.

Real-Time Operating Systems (RTOS) better meet the needs of real-time applications than general purpose operating systems like Windows. RTOS when used with appropriate hardware configuration and systematically developed software might produce deterministic behavior. For programming real-time applications, real-time languages like ADA and PEARL may be used. But many of such languages are not commercially available [4].

Because of the market forces, real-time control engineers expect to get maximum performance with available commercial off-the-shelf products. For example, the common personal computer with Intel processor, Windows operating system, and standard peripheral

interfaces (USB, serial, PCI) is often considered a low-cost option to achieve real-time performance.

## 2.2 Computer Hardware

Technological advances and new ideas employed in processor architectural design give the illusion of meeting timing constraints for real-time applications. A real-time system is not expected to process data in micro or nanoseconds but meet its deadlines. It is significant to note that there is a clear boundary between real-time performance and high performance. For example, a massive supercomputer running scientific simulations may give impressive performance yet not real-time performance.

For time-critical systems, predictability but not speed is the foremost concern. Processors with high clock speed and more memory might not contribute to determinism. There are many hardware issues that cause latency in real-time systems. The first flight of the space shuttle was delayed, at considerable cost, because of a subtle timing bug that arose from a transient CPU overload during system initialization [20]. Hence, the architecture and design of processor should be closely tailored to the needs of real-time applications to meet the timing constraints.

To exploit high performance, processors are employed with advanced architectural considerations like pipelining, caching, Direct Memory Access, exception handling and interrupt handling. These techniques present hazards to determinism since they complicate the prediction of execution times of processors [4]. In contrast, determinism may be more easily achieved with less complicated structures. For this reason, some engineers prefer to implement real-time controllers on microcontrollers rather than general purpose microprocessors.

## 2.3 Computer Software

The choice of particular type of programming language is an important decision factor in real-time system design. Many programming languages are not suitable for developing real-time applications because they lack deterministic features for expressing timing constraints [8]. For example, high level implementation languages like C and C++ have built in garbage collection techniques which may occur at any time during the execution of program, thereby affecting real-time performance.

Garbage collection is a memory management technique that reclaims the memory used by objects in the program that is no longer needed by the application. This technique eases the programmer of not doing the task manually but affects timely execution of events. Also, C doesn't have strong structured options for exception handling which may cause immediate termination of programs due to memory overflow, divide by zero exceptions thereby affecting predictability.

Some of the real-time programming languages like ADA, PEARL [4] and extensions of C++, Java to RTC++ [12] and RTJAVA enable real-time programs to meet timing constraints. Real-time programming languages and extensions support features like special data types, strong type checking, exception handling, modular programming, task synchronization and temporal behavior. Though general purpose programming languages like C and C++ are not better for real-time applications, they are used because of their advantages like programmer's familiarity, good availability and Windows support for development.

## 2.4 Operating Systems

The Operating System(OS) is one of the factors that affect determinism in real-time systems. The OS determines allocation of resources and CPU scheduling for the set of cooperating tasks so that they can obtain the necessary resources at the right time. Much

work has been done to improve performance of existing operating systems with different scheduling approaches for meeting real-time performance requirements [10][16] [18].

### 2.4.1 Windows scheduling strategy

State-of-the-art literature reveals that Windows operating system has several features that limits its suitability for hard real-time systems [17]. Windows uses priority based preemptive scheduling; if a higher priority process becomes ready while lower priority process is running, the lower priority process is preempted by higher priority process giving preference for higher priority process.

The higher the priority, the more the processing time is attributed to the process. But for a real-time system, predictable performance is more crucial than overall system performance. Windows was designed as a general purpose operating system with the goal to maximize system performance.

### 2.4.2 Priority structure

Figure 2.1 shows the priority structure of the Windows operating system. Processes run in two modes: kernel and user. Interrupt Service Routines (ISR's) and Deferred Procedure Calls (DPC's) run in kernel mode and all other user level initiated processes run in user mode. In user mode, there are 32 priority levels; 16-31 for real-time processes and 1-15 for other processes.

When the number of real-time processes increases, time sharing approach is used between threads. The 32 priority levels are called thread base priorities and a particular thread base priority is obtained by combining process priority and thread level priority. Each process belongs to one of the following priority classes: IDLE, NORMAL, HIGH and REALTIME. REALTIME priority class is provided to support real-time applications.

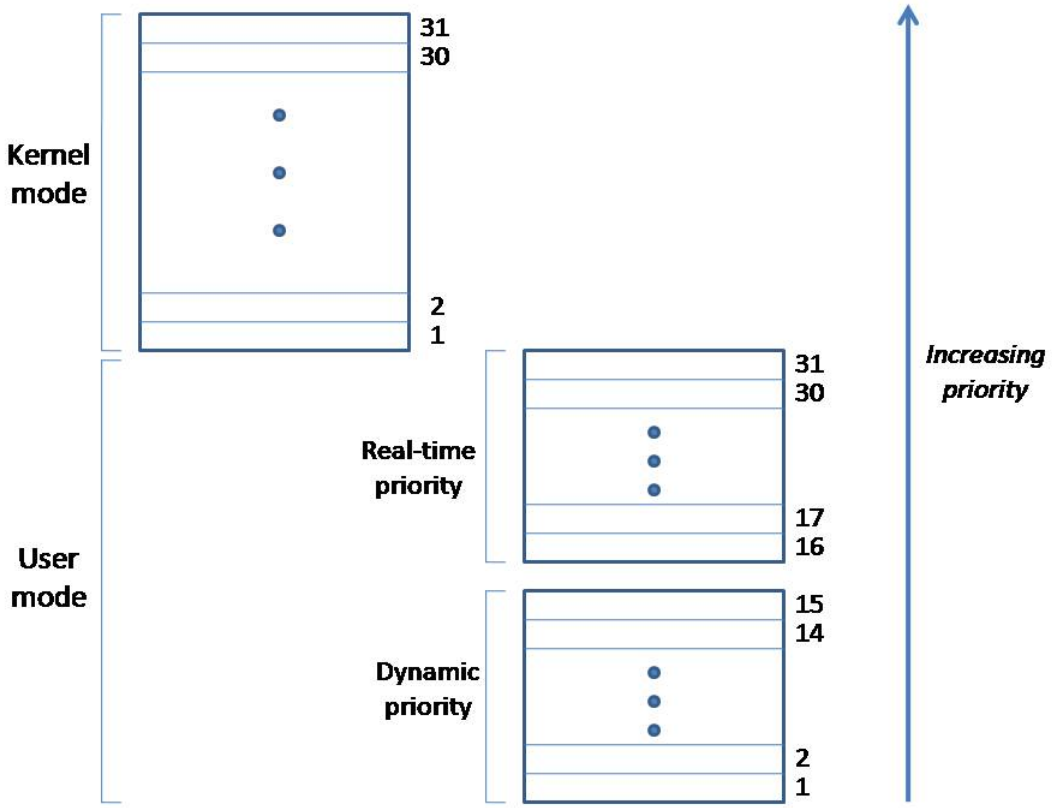


Figure 2.1: Windows priority structure

Each thread can have one of the following seven priority levels: IDLE, LOWEST, BELOW_NORMAL, NORMAL, ABOVE_NORMAL, HIGHEST and TIME_CRITICAL.

### 2.4.3 Context switching latency

When processes share the same priority level, context switching may cause delays in response times. When context switching occurs, the kernel switches from one running thread to another; the kernel saves context of the current running process in memory, retrieves the context of the process to be run next, restores it in CPU's registers, executes it, and then returns to the suspended process.

### 2.4.4 Latency with paging

Windows uses virtual memory and paging which allows the programs to reference more RAM than is available in the system. With virtual memory, the system looks for data that has been used least recently and copies it to the hard disk, creating free space in RAM to load new applications.

When a process needs data that is not in RAM, the OS determines its location in secondary memory, determines a location in RAM to load the data from secondary memory, and then loads it. Though it creates an illusion of unlimited RAM (RAM being more expensive), an insight into the paging mechanism reveals that, the OS is moving back and forth between hard disk and RAM. Page swapping can occur at any time during the execution of a thread, thereby jeopardizing predictability of a real-time system [2].

### 2.4.5 DPC latency

ISR is given the highest priority among kernel mode processes. When an interrupt occurs, critical processing is done by ISR and the rest by Deferred Procedure Calls (DPCs). DPCs are assigned the next priority level to ISR. With the arrival of multiple interrupts,

DPCs in FIFO queue cause significant indeterminism because the order in which DPCs are processed are not based on priority metrics. A higher priority DPC may have to wait for lower priority DPC to finish [16] [2]. Another noteworthy issue is that according to the priority hierarchy, real-time processes are at lower priority level than interrupts and a particular real-time running thread can be preempted by any interrupt.

### 2.4.6 RTOS and extensions

RTOS supports 256 levels of priority for real-time systems. The more the number of priority levels for real-time systems, the better the deterministic the system will be because this would overcome the overhead caused by context switching between threads.

Real-Time kernels often use the scheduling strategy that the task which needs quickest response is given the highest priority. The scheduler is the critical component of RTOS as it is responsible for providing deterministic time slices to real-time processes. It considers factors like criticality of the task, time deadline, precedence relations between tasks and resource constraints and map all of them into single figure called priority.

There are also extensions made to existing operating systems to support timing constraints like UNIX to RT-UNIX, LINUX to RTLINUX. Basically, this involves taking the kernel and adding necessary services like interprocess communication, scheduling, interrupt handling etc. Various problems arise when such extensions are made [18].

### 2.4.7 Windows I/O subsystem

Successful interfacing of peripheral devices is the responsibility of the OS. The I/O manager (see Figure 2.2) supports the communication between application software and peripheral device by providing a comprehensive set of system services. Every call to device driver from application is under the control of I/O manager. When I/O manager receives an

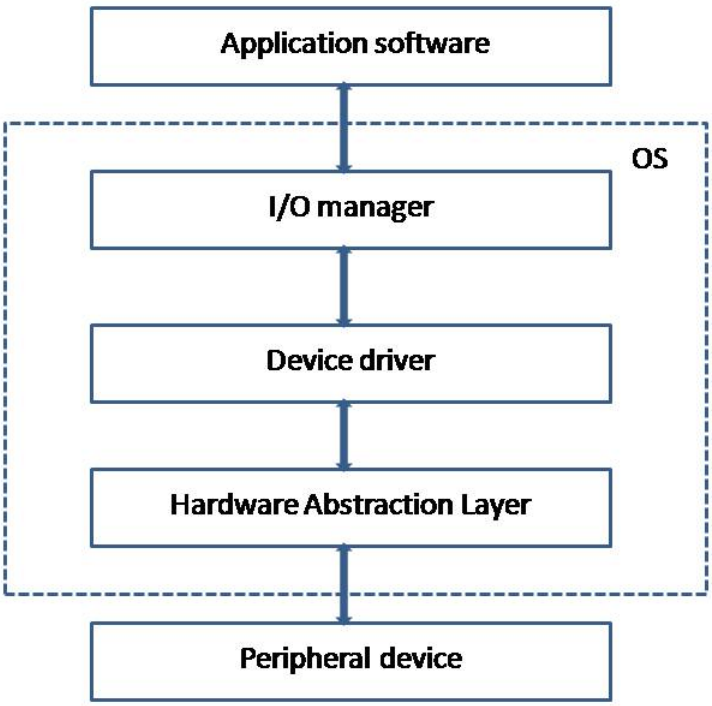


Figure 2.2: Windows I/O communication

I/O request, it constructs I/O Request Packets (IRPs) and directs them to the appropriate device driver which handles these IRPs. The application software does not have knowledge of which device driver services a particular device.

## 2.5 External Devices Interface

I/O devices are one of the major components of real-time control systems. Most real-time systems get input from outside world through standard interfaces like USB, SCSI etc. In a real-time operation, programs for processing of data arriving from external environment are permanently ready, so that the results are produced at the predetermined periods of

time. The arrival time of data is affected by communication protocol used, type of device, processor load etc.

### 2.5.1 Universal Serial Bus

USB presents an attractive option among other interfaces because of the features like high performance, capability to expand up to 127 devices, power conservation, and support for four different types of data transfers. This protocol makes the connection of peripheral devices to the computer easier and hence addresses simple I/O.

The four different types of data transfers are: interrupt, isochronous, bulk and control transfers [1].

- Interrupt transfers are used with interrupt driven devices like keyboard. A USB keyboard is polled periodically to determine if data is ready to be transferred.
- Isochronous transfers are continuous and periodic. This type of transfer is used in audio and video streaming applications. The device is guaranteed a slice of every frame.
- Bulk transfers are used for transferring large volumes of data that does not have any periodic requirement. A document to be printed transferred through USB is an example for bulk transfer.
- Control transfers are used during device configuration. The setup requests transferred during device enumeration is a typical control transfer.

USB is a shared bus that supports simultaneous attachment of many devices. Data is transferred in regular intervals of 1 ms called frames. Not every device will transfer data during each frame. Bandwidth allocation depends on throughput of the device and available bandwidth. A mix of transfer types is likely to be performed during each 1ms frame.

USB specification states that isochronous and interrupt transfers occupy 90% of USB bandwidth while control transfers have 10% reservation. Bulk transfers are allocated with the remaining bandwidth after all other transfers have been scheduled.

Hence, when a real-time system is communicating to several devices that follow different data transfer modes, devices with bulk transfer have to wait for longer times causing latencies. Better bandwidth reservation algorithms have been proposed for USB to support real-time applications by modifying scheduling strategy in USB driver [9].

### 2.5.2 Peripheral Component Interconnect

PCI is a high performance IO bus that can be configured dynamically for devices that need high bandwidth requirements. Some of the interesting features of PCI are low-power consumption, low pin count, auto configuration and processor independence.

PCI uses a shared bus topology that allows different PCI devices like network card, sound card, or a RAID card to be attached to the same bus, through which they communicate with the CPU. PCI makes use of DMA (Direct Memory Access), the ability to access system memory without consulting the CPU which significantly improves system performance.

When several devices are attached to the PCI bus, a bus arbitration scheme is used to decide the allocation of the bus for devices. Devices are assigned priority based on the value of device's configuration register, which is maximum latency, the time it has to wait for getting bus access.

Once a device has control of the bus, it becomes the bus master and starts communicating with the processor and memory. Other devices have to wait until bus master relinquishes the bus, this affects the timeliness of critical activities. Most PCI devices are capable of initiating a data transfer as bus masters without the need for CPU.

## 2.6 An Experimental Example: Videocapture Latency

This study measures the latency of videocapture by attaching a webcamera to USB and then to PCI through a PCI/USB interface card. PCI interface could only be tested on Machine2 because it did not exist in Machine1 which is a laptop computer. As discussed in Section 2.1, latency comes from various sources. The experiment was performed on the following two different Machines.

1. Machine1: Windows Vista, Intel Core Duo processor with 1GB RAM, 120GB Hard Disk and 1.73GHz clock speed

2. Machine2: Windows XP, Intel Celeron processor with 512MB RAM, 60GB Hard Disk and 2.4GHz clock speed.

The software is written in VB.NET and timing is measured using QueryPerformanceCounter function. Since the results are expected to be in millisecond precision, events are timed using highest resolution timer offered by the system. Windows NT 3.1 or later versions and Windows 95 or later versions support this function. It accesses the CPU's high performance counter values. QueryPerformanceFrequency function retrieves the frequency of high-resolution performance counter. The resolution is system-dependent and is given by

$$R = 1/\text{QueryPerformanceFrequency}$$

The frequency of the high performance counter retrieved using QueryPerformanceFrequency with Machine1's Hardware is 14318180 Hz and for Machine2 is 3579545 Hz.

While timing an event, if $T_1$ is the start count value of the counter and $T_2$ is the end count value, then seconds elapsed $T_e$ is calculated by

$$T_e = (T_2 - T_1)/\text{QueryPerformanceFrequency}$$

Table 2.1: Statistical analysis for videocapture

| | Min (ms) | Max (ms) | Avg (ms) | std |
|---|---|---|---|---|
| Machine1 (USB) | 99.7660 | 139.8132 | 109.2908 | 3.1746 |
| Machine2 (USB) | 75.2070 | 268.4690 | 125.8854 | 35.5179 |
| Machine2 (PCI) | 84.7834 | 146.2633 | 111.3936 | 8.9181 |

The tests were conducted on two different machines with different operating systems, hardware configurations and different technical specifications to validate latency with videocapture. The preview rate for videocapture is set to 100ms.

The system is expected to capture one frame every 100ms. But due to limitations with hardware, software, operating system and peripheral intereface, frame capture rate of 100 ms was not observed. The experiment was run until 1000 samples were collected and statistical values were analyzed. Figures 2.3 and 2.4 show the results obtained from experimental runs and Table 2.1 shows the statistical values calculated for both machines. The test cases analyzed were webcam connected to

1. Machine1's USB port
2. Machine2's USB port
3. Machine2's PCI bus using a USB/PCI interface card

It can be noted that the minimum value is actually less than 100ms for both Machine1 and Machine2. A real-time application that depends on input from webcam gets data earlier for some frames and also delayed for some frames.

On Machine1, the average value is 9 ms greater than expected average value. This delay could significantly affect determinism in real-time systems. Figure 2.3 illustrates the frequency distribution of data with respect to time. More data appears to be in the range from 108 to 109 ms.

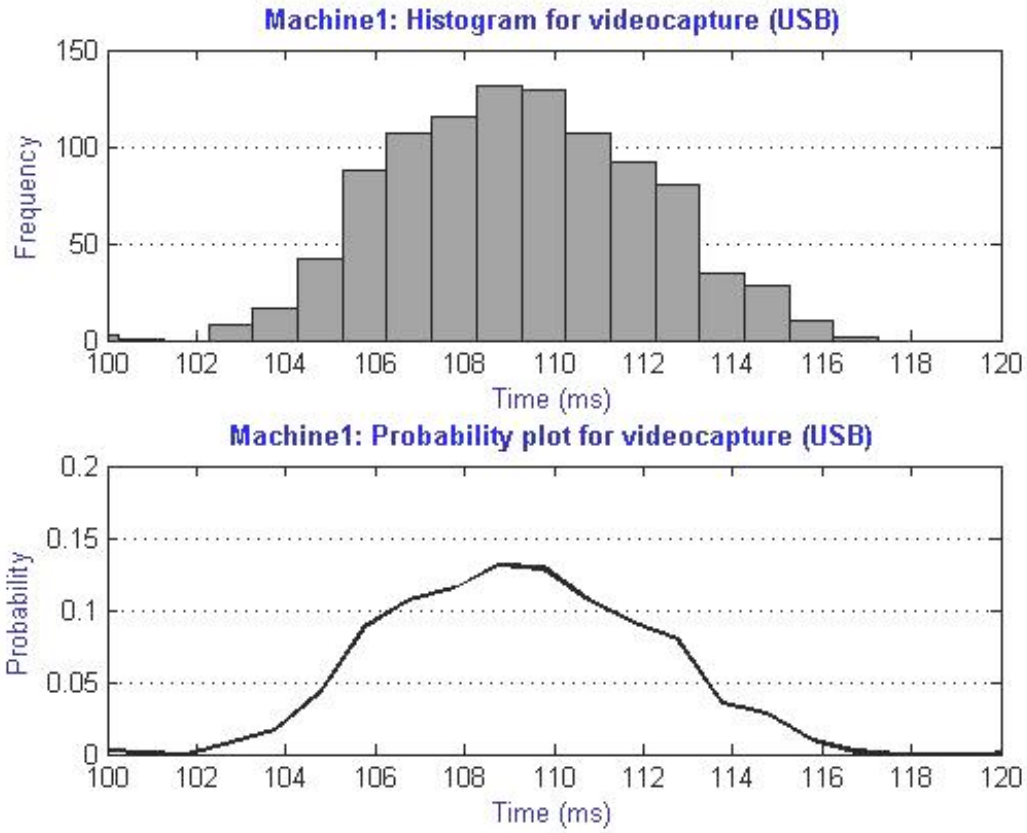


Figure 2.3: Machine1: Videocapture latency (USB)

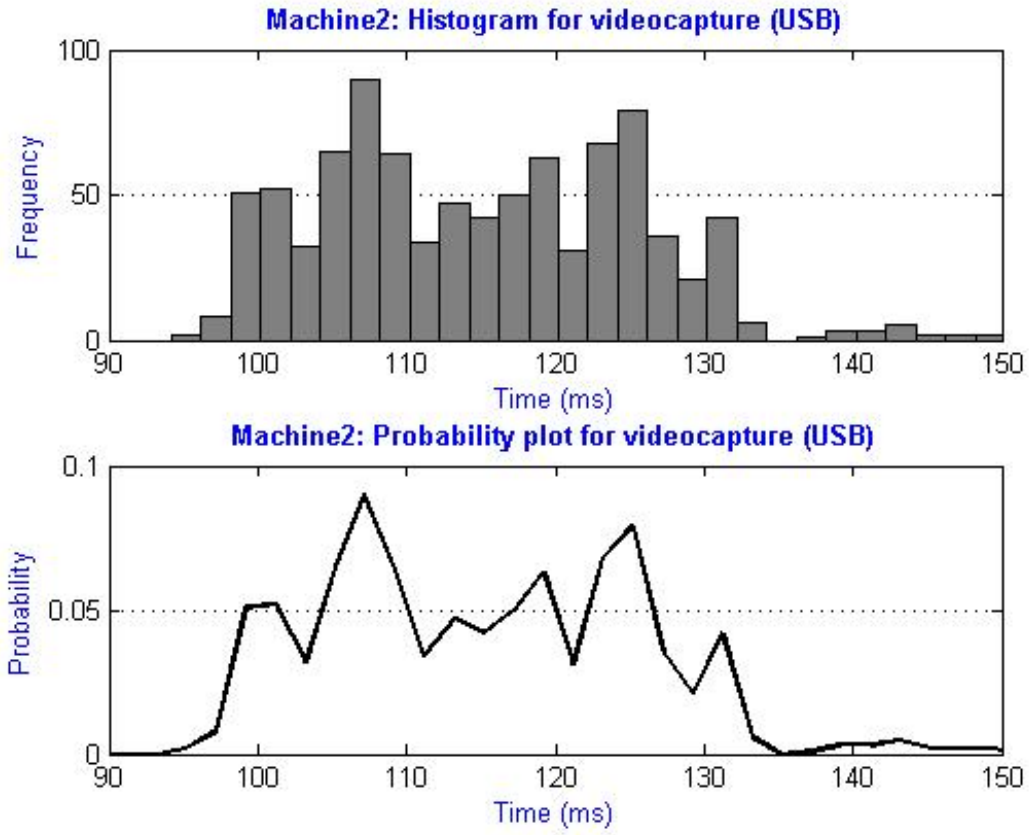


Figure 2.4: Machine2: Videocapture latency (USB)

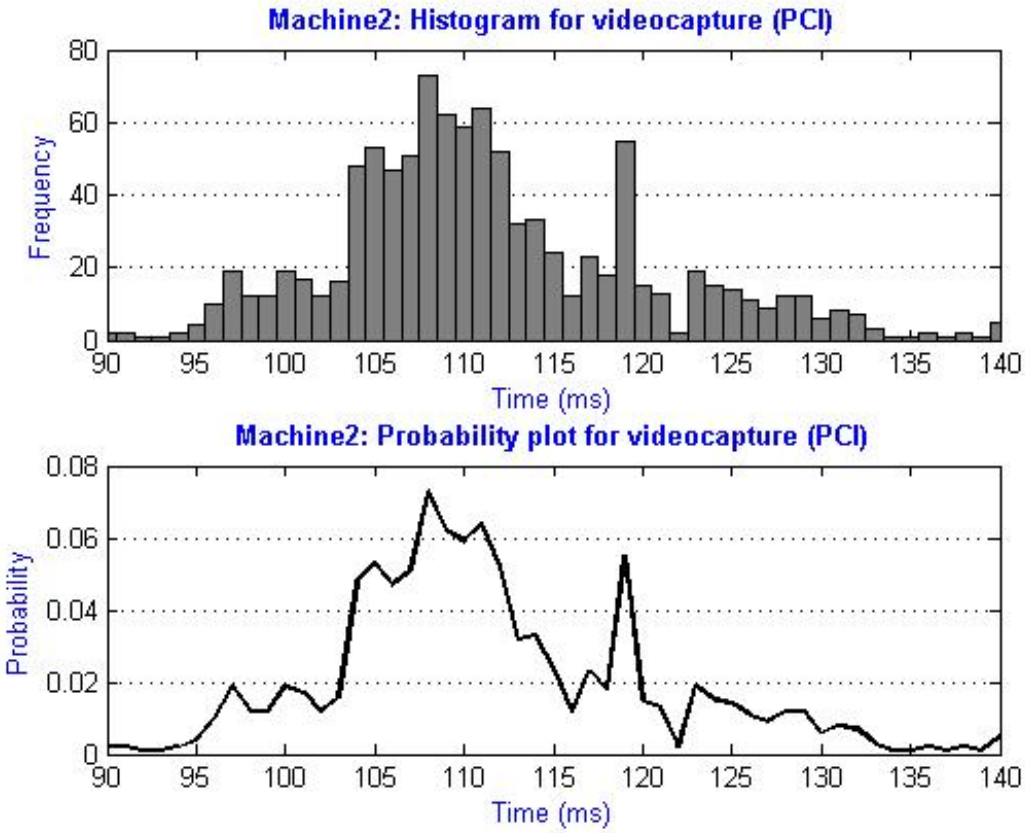


Figure 2.5: Machine2: Videocapture latency (PCI)

For Machine2 with USB, the minimum value is 25 ms lower and maximum value is 168 ms higher than expected time. This early occurrence of an event and delayed event indicates the extent to which determinism is affected. High value of standard deviation complicates the prediction of timely execution of events. This is evident from the probability plot of Figure 2.4 as the curve is more spread apart.

Changing the I/O interface on Machine2 from USB to PCI greatly improves predictability. Figure 2.5 illustrates the latency of videocapture with PCI bus on Machine2. The mean of 1000 samples is expected to be 100 ms but Table 2.1 shows the average to be close to 111 ms. Although functionally correct, frames captured beyond or before predetermined time are considered invalid from a real-time perspective.

## 2.7 Conclusion

Deadline miss was evident with USB webcamera where the combination of factors discussed in Section 2.1 comes into play. Also, statistical analysis of experimental data and comparison with two machines give us some idea about the spread and probability of latencies.

The next chapter demonstrates the effects of latency with USB when different devices like keyboard, mouse, USB drive and webcam are communicating to the real-time system. Another interesting related issue that is explored is, "Will a USB hub introduce latency with peripheral devices?"

# Chapter 3

## Latency analysis and Comparison - USB and USB hub

In this chapter, latency values for getting input from different peripheral devices like keyboard, mouse, USB drive and webcam were measured by attaching them to USB and then through hub. A statistical analysis was done to compare the delays and find whether hub causes more latency.

### 3.1 Introduction

In real-time operation of a computer system, the programs for the processing of input data are constantly operational so that the results will be available within predetermined periods of time. For example, an aircraft must process accelerometer data within specified time that depends on the specifications of the aircraft (for example every 5 ms) [13].

Universal Serial Bus has become an industry standard for attaching peripheral devices to the computer. This chapter presents an overview of USB, measures the latencies caused by USB and USB hub and compares their latencies to determine whether USB hub adds overhead to USB. Experimental evaluation shows that USB is a good interface for soft real-time systems that can tolerate delays in the order of few milliseconds. It also provides insight needed for building real-time architectures that depend on USB to collect sensor data.

USB's success is due to some of its characteristics like simplicity, plug and play, low power consumption, expandability, high performance and error detection and recovery. In recent years, sensors with USB connectivity have been developed for industrial applications;

therefore temperature, humidity, pressure and other sensors are utilizable to accomplish specific tasks [6].

One of the objectives of this research is to evaluate the suitability of USB for real-time applications. The behavior of different peripheral devices like keyboard, mouse, USB drive and webcam with respect to USB and hub is studied from a latency perspective. Section 3.2 gives an overview of USB's technical specifications like device detection, data communication mechanism, data transfer modes and packet formats. Since this research also involves testing with USB hub, a brief explanation about USB hub's functionality is also presented. Experimental results in Section 3.3 summarize the latencies caused by USB and USB hub.

## 3.2 USB Overview

USB makes the connection of multiple peripheral devices to the computer easier and more efficient and hence addresses simple I/O.

### 3.2.1 USB device detection

When a USB device is attached to the PC, the host software (operating system) detects and configures the device. The USB driver detects the device characteristics through a device descriptor. A descriptor is a data structure which contains information about the device and its properties. Then, the USB driver locates the appropriate device driver and loads it. Once the device driver is loaded, it establishes a logical connection called pipe between device driver and endpoint. Data transfer takes place through this communication pipe. Endpoint is a logical entity (register) where data enters or leaves USB device.

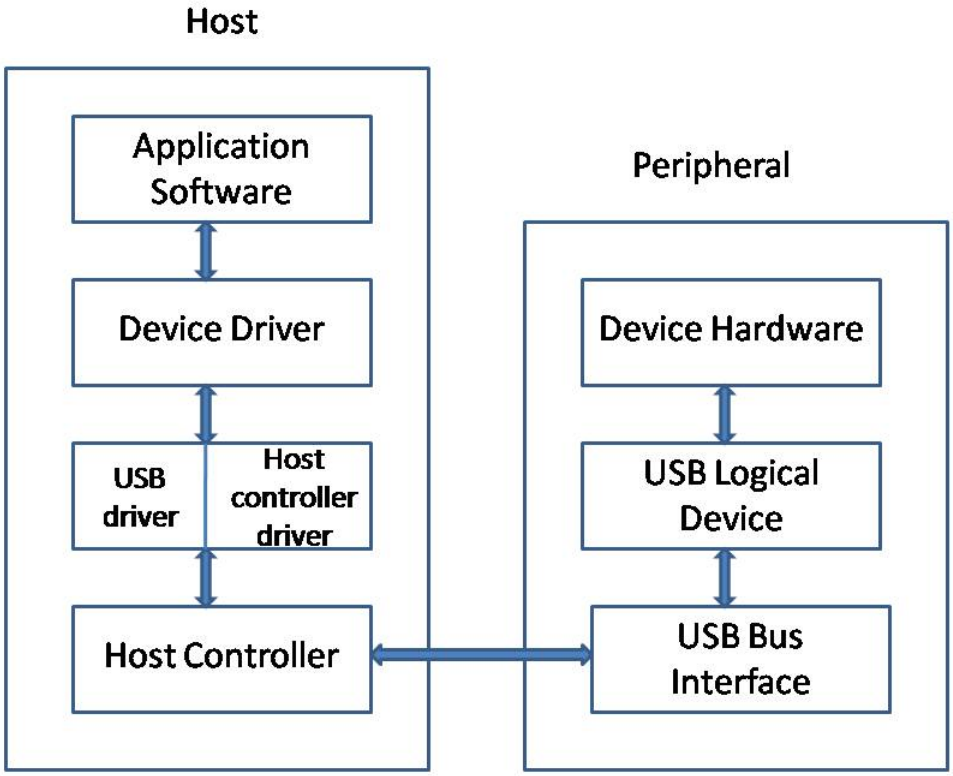


Figure 3.1: USB communication

### 3.2.2 USB communication

USB communication is based on master slave relationship between host computer and device (see Fig. 3.1). Slave responds to master's requests and there can be only one master in the system, i.e., host computer. Also, USB devices cannot interact with each other without host [15]. USB system software consists of USB driver and host controller driver which are explained below. USB communication could be explained as a series of the following steps:

1. When a USB device needs to transfer data, the device driver sends a transfer request to USB driver. These packets are termed as IO Request packets (IRP's).

2. The device driver supplies a memory buffer which is used to store data when transferring data to or from the device.

3. USB driver splits the IRP into individual transactions that will be executed during series of 1 ms time frames. It organizes the requests based on device requirements, needs of client driver and capabilities of USB.

4. The host controller driver schedules these requests according to the algorithm based on USB transfer capabilities.

5. The host controller then executes the transactions and passes data through USB. Each transaction results in data being transferred from USB device to client buffer or from client buffer to USB device.

### 3.2.3 USB data transfer format

USB transfers data in series of 1 ms time frames. Fig. 3.2 illustrates the packet transfer mechanism of USB. Each frame consists of 1 or more transactions. Transactions consist of three packets: token, data, and handshake [11].

- Each transaction begins with a token packet which indicates the type of transaction that follows: IN for data transfer from USB device to system, OUT for data transfer from system to USB device and SETUP for control transfer.

- Data packet carries the payload associated with transaction. There are two types of data packets: DATA0 and DATA1 each can carry 1024 bytes.

- Handshake packet contains information about transaction status. There are three types: ACK indicates error-free receipt of data, NAK signifies host that target is unable to accept data and STALL is used by target to report that it cannot complete transfer and requires intervention by host.

A packet is the mechanism for performing all transactions. Each packet consists of following fields:

- SYNC: It indicates the speed of data transfer.
- PID: Packet Identifier identifies the type of packet sent (token, data and handshake).
- ADDR: It is the address field that specifies the device to which the packet is destined for.
- ENDP: Endpoint number where data enters or leaves the system.
- CRC: Cyclic Redundancy Check is used to check for errors in data payload.
- EOP: signals the End of Packet.

### 3.2.4 USB hubs

USB hubs permit extension of USB system by providing additional ports for attaching USB devices. Hubs may be bus-powered (derive power from USB bus for itself and attached devices) or self powered. As Fig. 3.3 shows hubs consists of two major functional components; hub controller and repeater [1].

- Hub controller: It gathers hub and port status information that is read by USB system software to detect attachment and removal of device. It receives commands from USB system software to control various aspects of hub's operation such as powering and enabling ports.
- Repeater: It transfers the data between host controller and device based on control signal from hub controller.

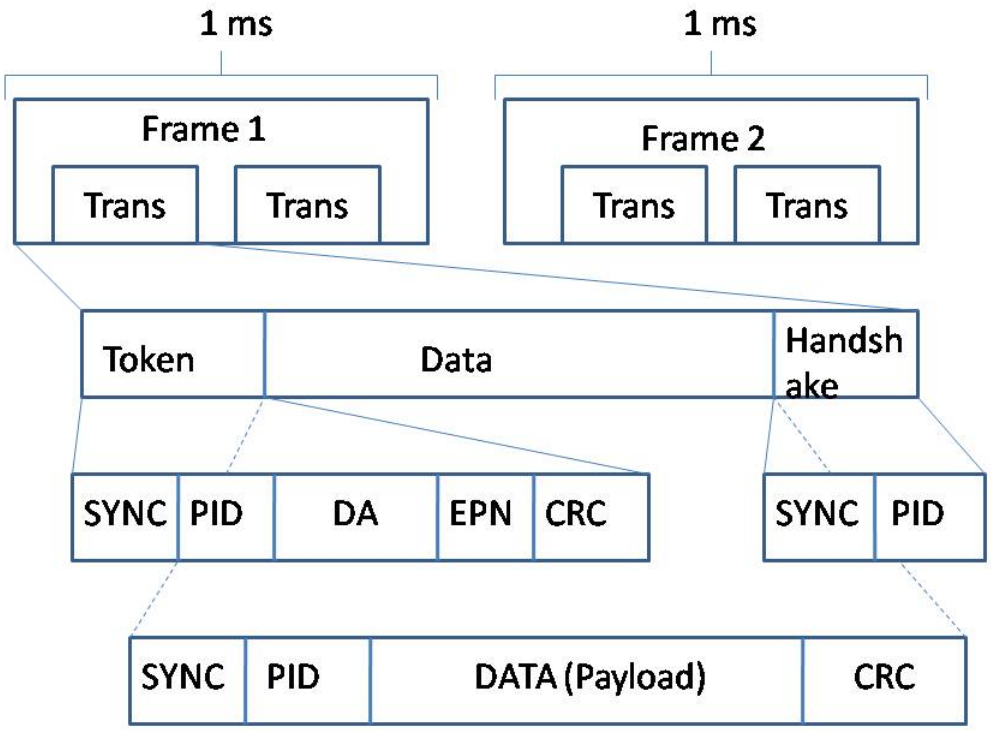


Figure 3.2: USB data transfer format

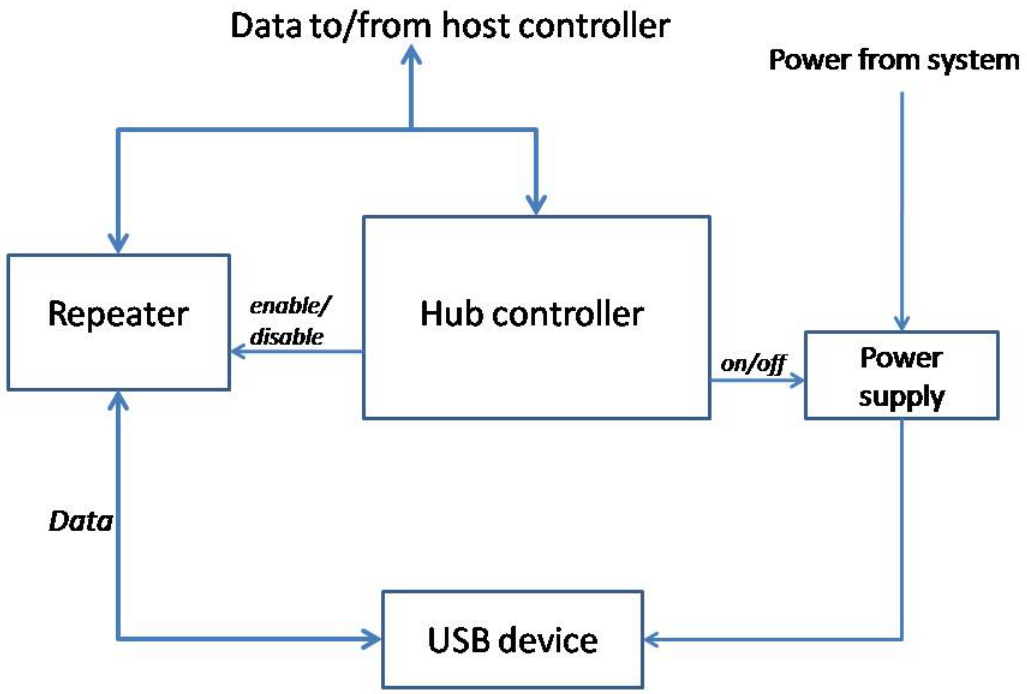


Figure 3.3: USB hub

### 3.3 Experimental Evaluation

A study was conducted to determine the latency with USB and USB hub using peripheral devices like keyboard, mouse, USB drive and webcamera. The experiment was performed on the following two different machines.

1. Machine1: Windows Vista, Intel Core Duo processor with 1GB RAM, 120GB Hard Disk and 1.73GHz clock speed

2. Machine2: Windows XP, Intel Celeron processor with 512MB RAM, 60GB Hard Disk and 2.4GHz clock speed.

The software is written in VB.NET and timing is done using QueryPerformanceCounter function. A series of experiments enable us to analyze and compare the results from different classes of USB devices that have different data transfer modes. The experimental data shows a considerable latency in the order of milliseconds with USB and USB hub for certain devices.

Since the results were expected to be in millisecond precision, events are timed using highest resolution timer offered by the system. Counter resolution is system dependent, but in all experiments the resolution is much less than one microsecond.

This section consists of four subsections, one for each tested device. Each subsection presents data in two tables; one table lists the values of minimum, maximum, average, and standard deviation (in milliseconds), and the other table summarizes the values of Student's T-tests.

The T-test compares the means of two sample sets based on the null hypothesis that both means are likely to be equal with 95 percent confidence (significance level $\alpha = 0.05$). If h=0, the null hypothesis is accepted and means are statistically equal. If h=1, the null

Table 3.1: Statistical analysis of keypress

| | Machine1 | | | | Machine2 | | | |
|---|---|---|---|---|---|---|---|---|
| Connection type | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| System | 21.986 | 42.054 | 31.931 | 0.789 | 11.036 | 60.38 | 33.222 | 4.261 |
| USB | 21.651 | 61.123 | 33.014 | 1.552 | 11.027 | 69.056 | 33.246 | 4.188 |
| USB hub | 17.146 | 41.515 | 33.017 | 0.813 | 13.624 | 68.194 | 33.520 | 5.479 |

hypothesis is rejected; the confidence interval for the difference between the two means is displayed between brackets.

### 3.3.1 Keypress latency

This experimental part determines the time taken for reading characters from the keyboard. The test is conducted for three cases:

- System keyboard connected through the PS/2 interface
- Keyboard connected through a USB port
- Keyboard connected through a USB hub

In each case, 1000 samples were collected. Shown in Figure 3.4 are that data histogram and candidate probability curves. Probability curves are plotted in both linear and logarithmic vertical axis. The plots show the sample data does not fit a normal density curve.

From Table 3.1, it can be inferred that latency is the highest for a USB keyboard, compared to the other two cases. On Machine1, it can be observed from Table 3.2 that the USB and USB hub introduce latency when compared to system keyboard. But, the USB hub does not introduce significant latency when compared against USB on either Machine1 or Machine2. The graphical representation of statistical data in tables is shown in Figures 3.5 and 3.6.

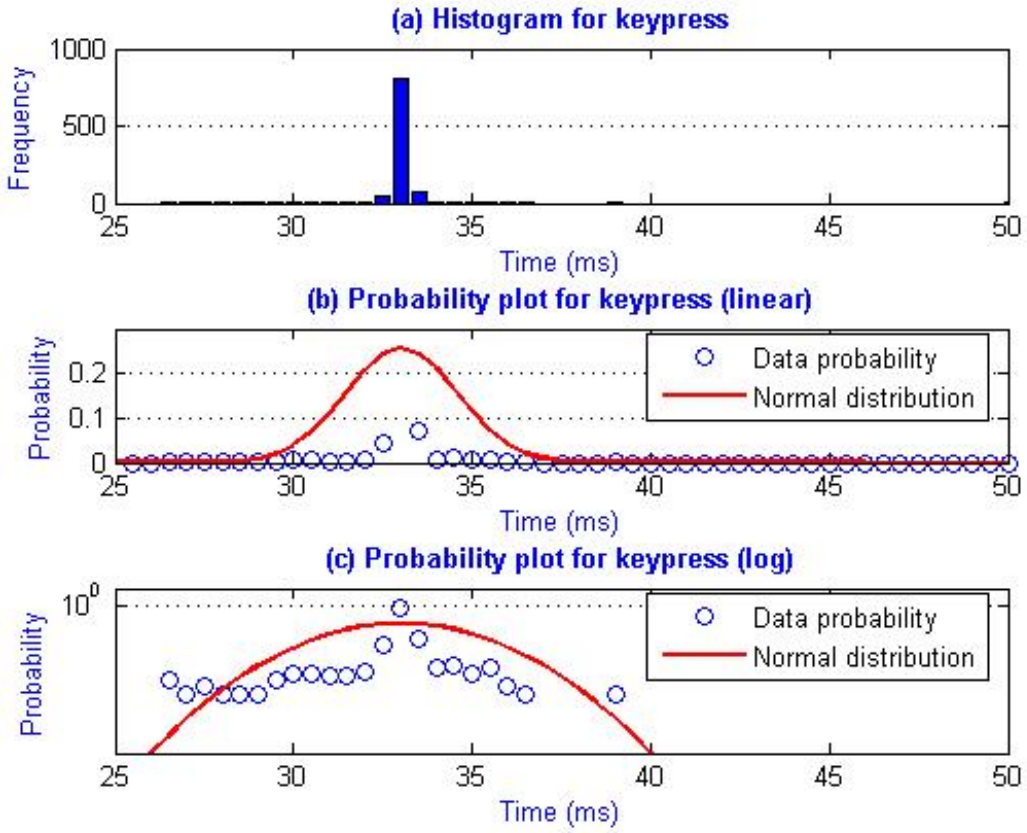


Figure 3.4: Machine1: Statistical analysis for USB keyboard

Table 3.2: Comparison of means (T-test $\alpha = 0.05$) for keypress

| Comparison | Machine1 | Machine2 |
|---|---|---|
| System vs. USB | h=1 [-1.1910,-0.9751] | h=0 (no difference) |
| System vs. USB hub | h=1 [-1.1565, -1.0160] | h=0 (no difference) |
| USB vs. USB hub | h=0 (no difference) | h=0 (no difference) |

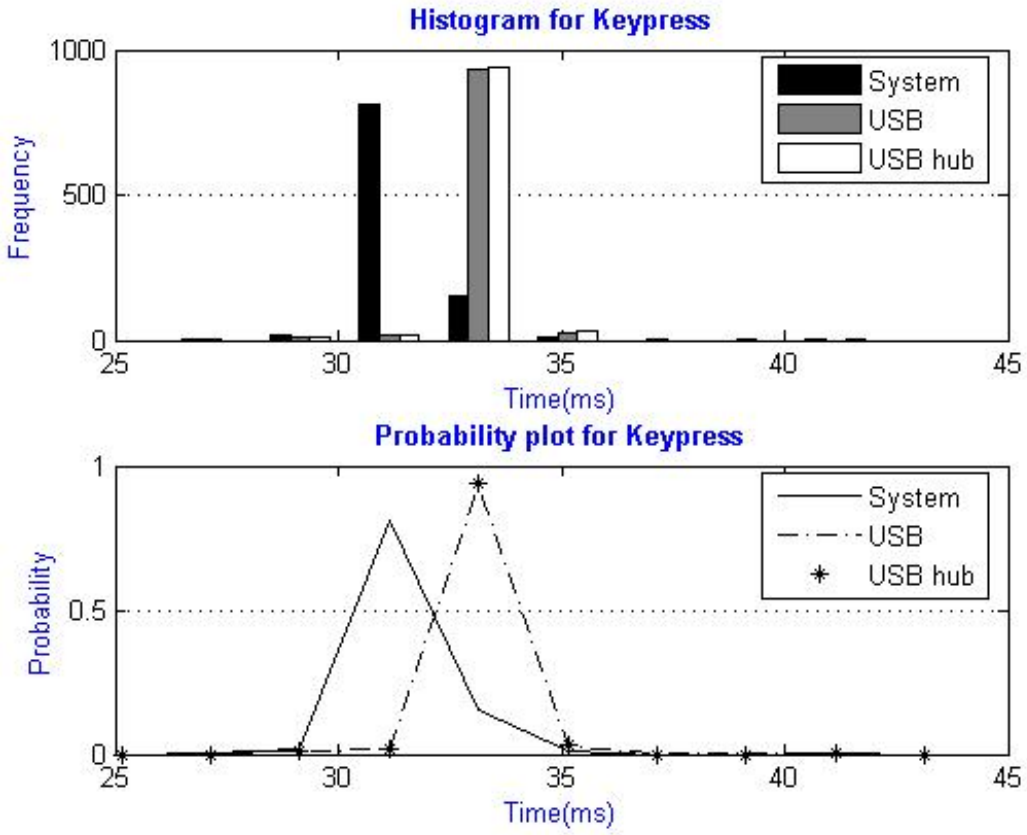


Figure 3.5: Machine1: Comparison of keypress latencies for three interfaces: system keyboard, USB keyboard and keyboard connected via USB hub

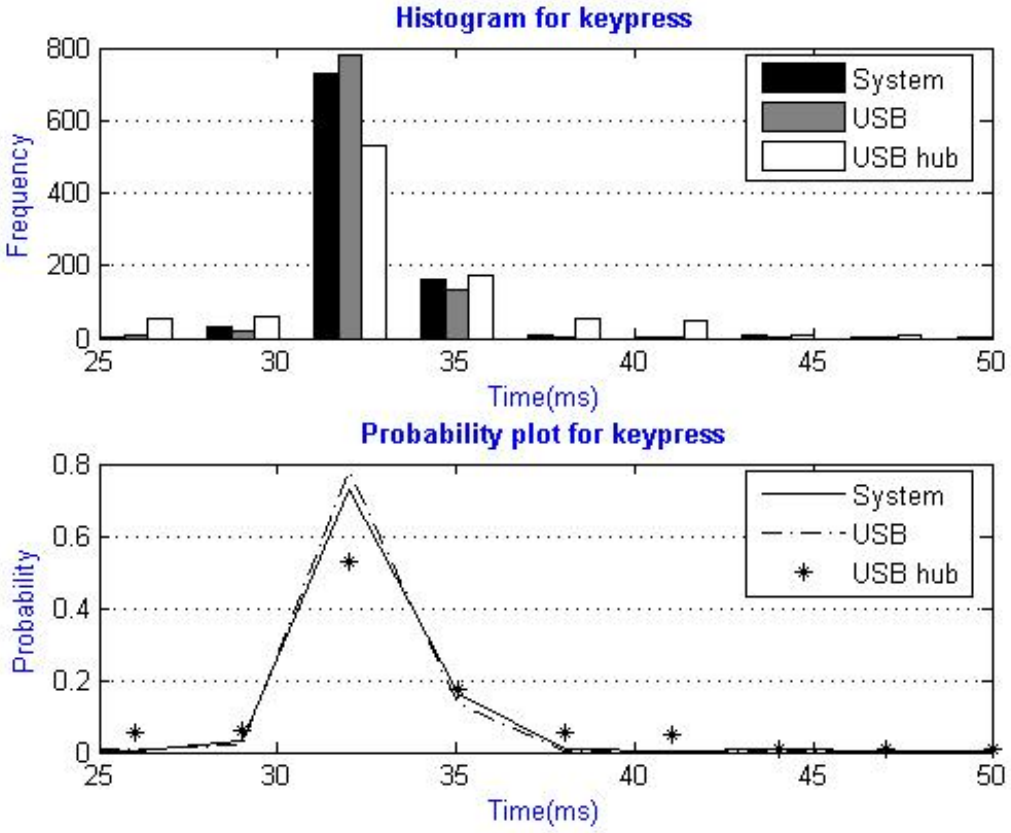


Figure 3.6: Machine2: Comparison of keypress latencies for three interfaces: system keyboard, USB keyboard and keyboard connected via USB hub

Table 3.3: Statistical analysis for fileread

| Device | Machine1 | | | | Machine2 | | | |
|---|---|---|---|---|---|---|---|---|
| | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| Hard disk | 0.136 | 17.798 | 0.185 | 0.583 | 0.693 | 38.801 | 1.005 | 1.974 |
| USB drive | 0.376 | 7.798 | 0.599 | 0.305 | 1.103 | 25.934 | 1.510 | 0.953 |
| USB drive hub | 0.371 | 5.402 | 0.629 | 0.188 | 1.124 | 46.345 | 1.582 | 1.746 |

Table 3.4: Comparison of means (T-test $\alpha = 0.05$) for fileread

| Comparison | Machine1 | Machine2 |
|---|---|---|
| Hard disk vs. USB drive | h=1 [-0.4547, -0.3731] | h=1 [-0.6406, -0.3687] |
| Hard disk vs. USB hub drive | h=1 [-0.4822, -0.4061] | h=1 [-0.7395, -0.4126 |
| USB drive vs. USB hub drive | h=1 [-0.0524, -0.0080] | h=0 (no difference) |

### 3.3.2 Fileread latency

To evaluate the latency of data transfer from a USB drive, time taken to read a text file is measured in three cases, from:

- Hard Disk
- USB drive
- USB drive connected through hub

The text file size is constant throughout the experiment and Table 3.3 summarizes the statistical values. The file is read 1000 times and the experiment is repeated for all the three cases with two Machines. Figure 3.7 illustrates that distribution of sample data for USB drive is different from rayleigh distribution; this is because sample data is more spread and most of the probability values are close to 0. It can be inferred from Table 3.4 that accessing data from USB drive is slower than hard disk. Although USB hub causes latency, its effect is not very significant, based on results from T-tests. The graphical representation of the statistical data in tables for both machines is shown in Figures 3.8 and 3.9.

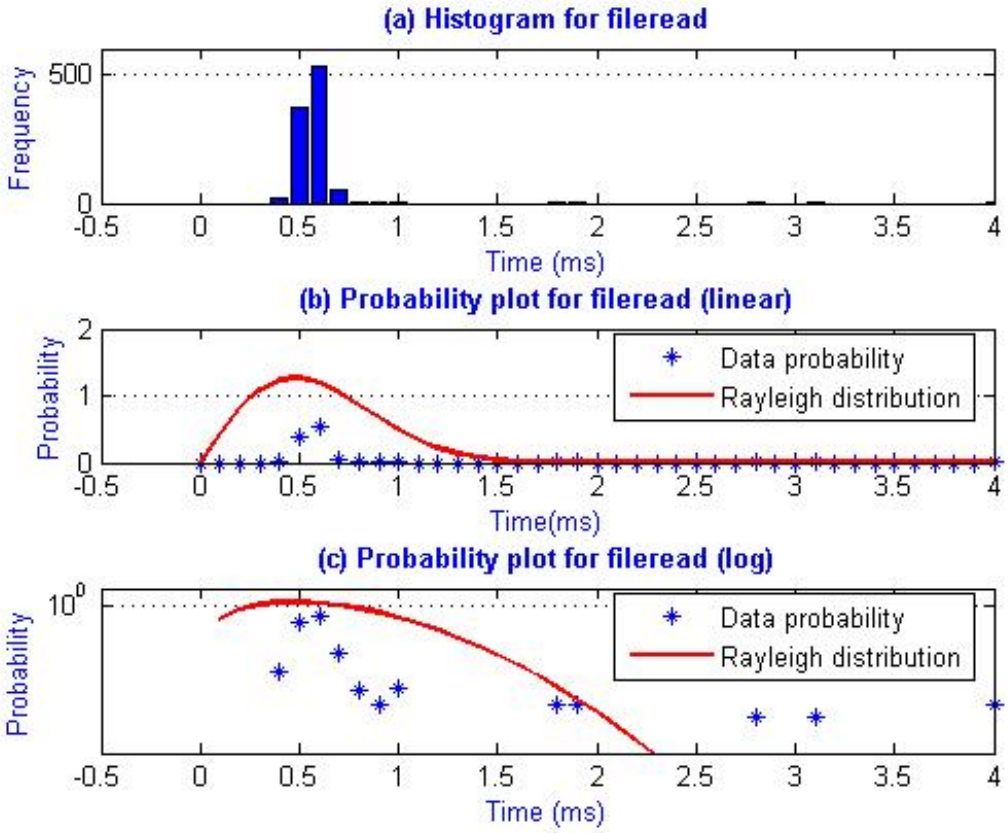


Figure 3.7: Machine1: Statistical analysis for fileread from USB drive

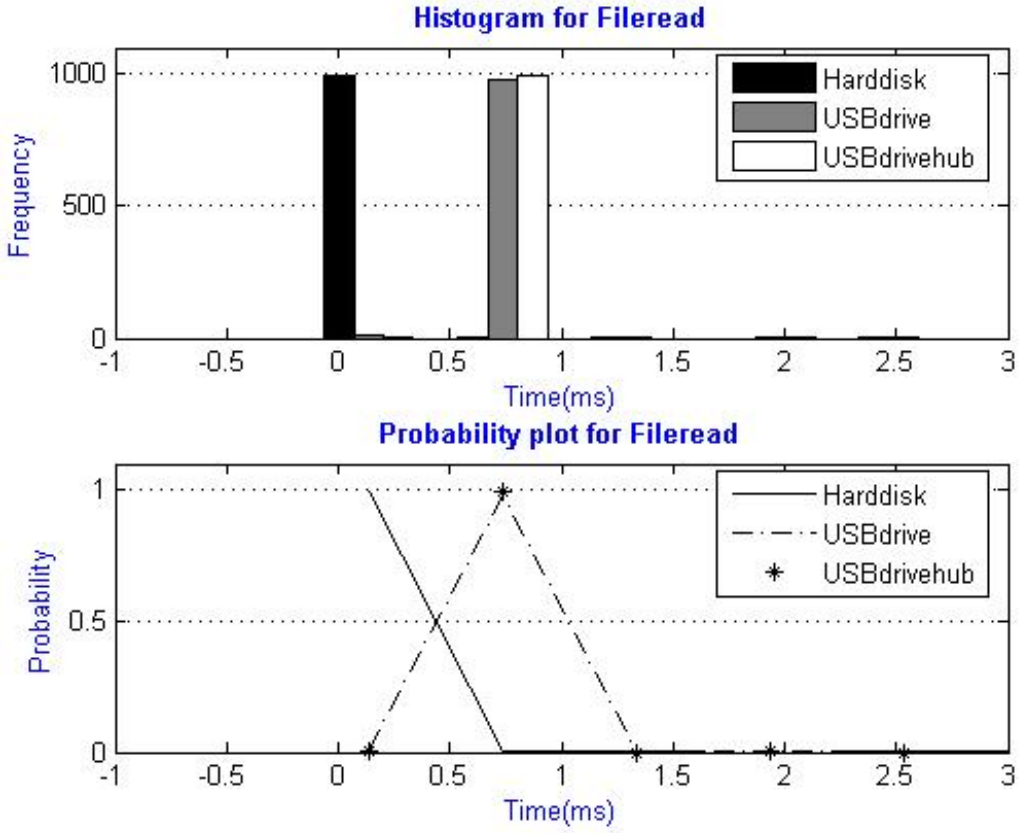


Figure 3.8: Machine1: Comparison of fileread latencies for three interfaces: system hard disk, USB drive and USB drive connected via USB hub

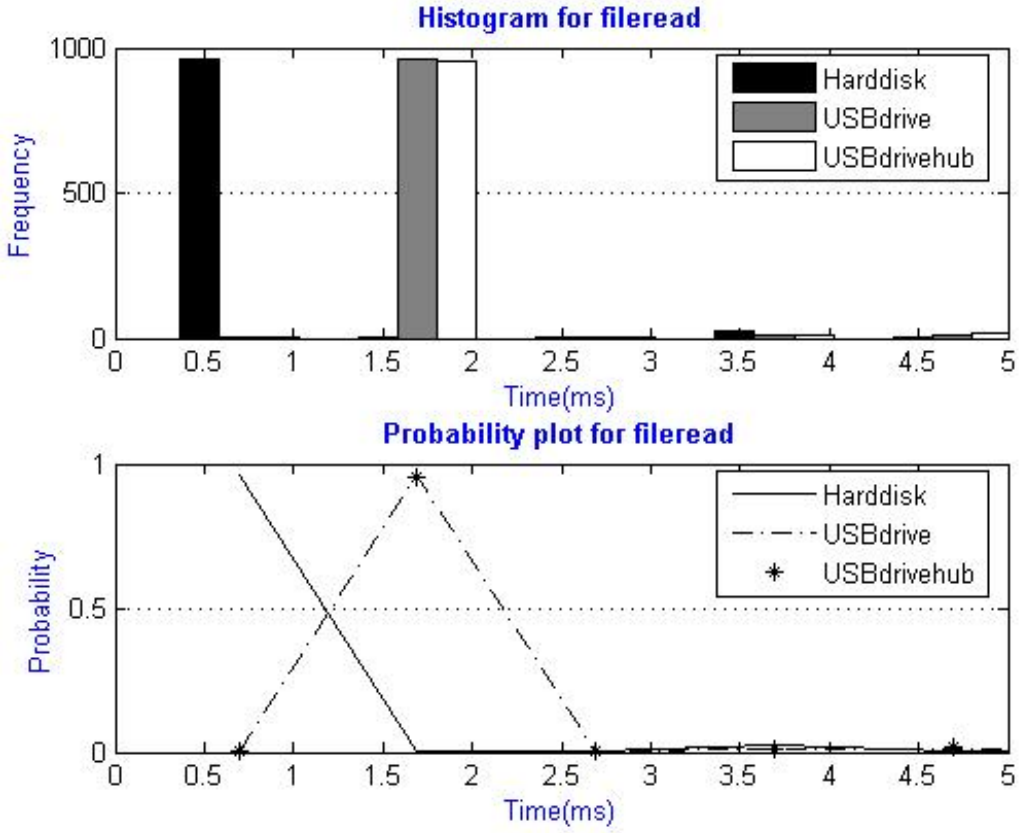


Figure 3.9: Machine2: Comparison of fileread latencies for three interfaces: system hard disk, USB drive and USB drive connected via USB hub

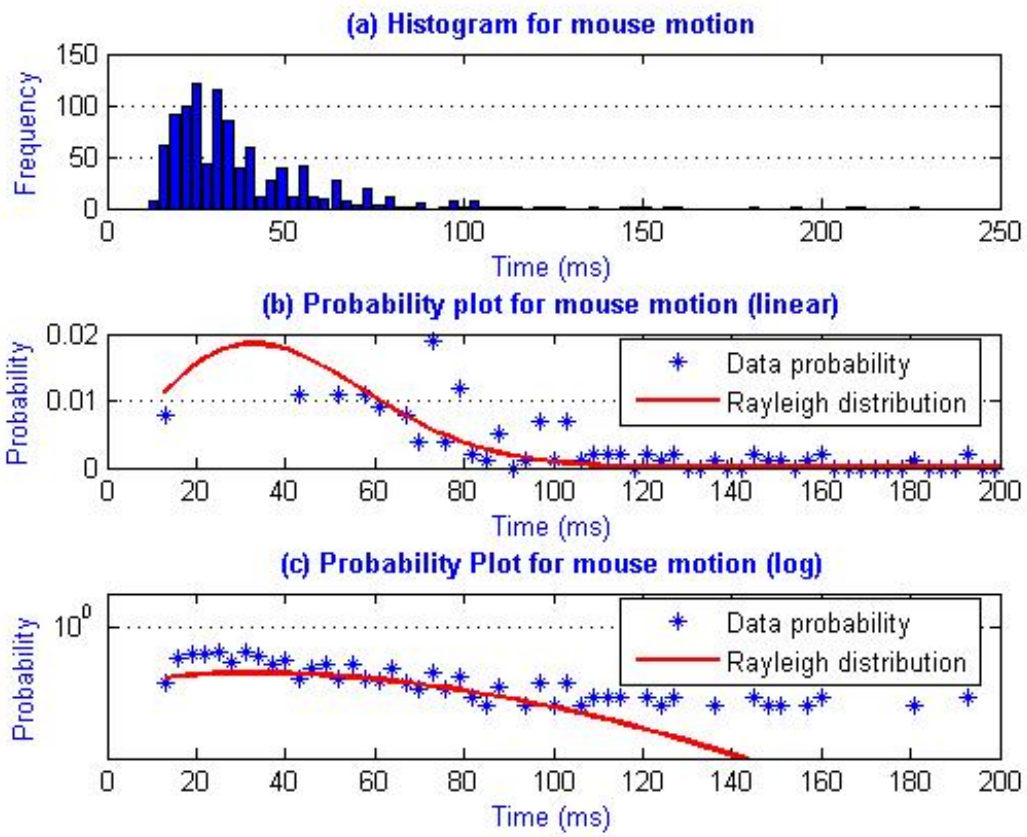


Figure 3.10: Machine2: Statistical analysis for mousemotion from USB mouse

### 3.3.3 Mousemotion latency

The third experiment consists of the implementation of the code that detects mouse position and timer value for every mousemove event. Over 1000 samples were collected on two different machines, each over three cases:

- Mouse conected through PS/2
- Mouse connected through USB
- Mouse connected through USB hub

Table 3.5: Statistical analysis for mousemotion

| | Machine1 | | | | Machine2 | | | |
|---|---|---|---|---|---|---|---|---|
| Connection type | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| PS/2 | 2.440 | 144.849 | 19.824 | 14.309 | 5.678 | 456.893 | 26.640 | 26.034 |
| USB | 7.482 | 128.145 | 22.513 | 10.950 | 13.159 | 225.027 | 38.572 | 25.530 |
| USB hub | 6.983 | 251.055 | 30.648 | 20.758 | 10.948 | 256.720 | 40.706 | 26.243 |

Table 3.6: Comparison of means (T-test $\alpha = 0.05$) for mousemotion

| Comparison | Machine1 | Machine2 |
|---|---|---|
| PS/2 vs. USB | h=1 [-3.8064, -1.5715] | h=1 [-14.1935, -9.6709] |
| PS/2 vs. USB hub | h=1 [-12.3876, -9.2603] | h=1 [-16.3587, -11.7737] |
| USB vs. USB hub | h=1 [-9.5908 -6.6793] | h=0 (no difference) |

The histogram, probability distribution and Rayleigh distribution of sample data in Figure 3.10 reveal the fact that sample data roughly fits the Rayleigh density function. Table 3.5 shows that the average values for sample data increases for mouse connected from PS/2 to USB to USB hub. A statistical comparison of means summarized in Table 3.6 is useful in validating the latencies with three different cases. Mouse connected through PS/2 is faster than USB mouse which is faster than mouse through USB hub. Hence, real-time applications that depend on input from mouse connected through USB hub are affected by latency issues. The graphical representation for the above statistical data in tables for both machines are shown in Figures 3.11 and 3.12.

### 3.3.4 Videocapture latency

To observe the effects of latency with videocapture, software was developed to determine the time taken for capturing every frame from an inexpensive Webcam. The frame capture rate was set to be 100 ms, i.e the system should ideally capture a single frame 10 times per second.

The QueryPerformanceCounter was used to measure the actual time taken for capturing every frame. Time taken for capturing 1000 frames were collected and analyzed. The experiment was conducted for two cases on 2 different machines.

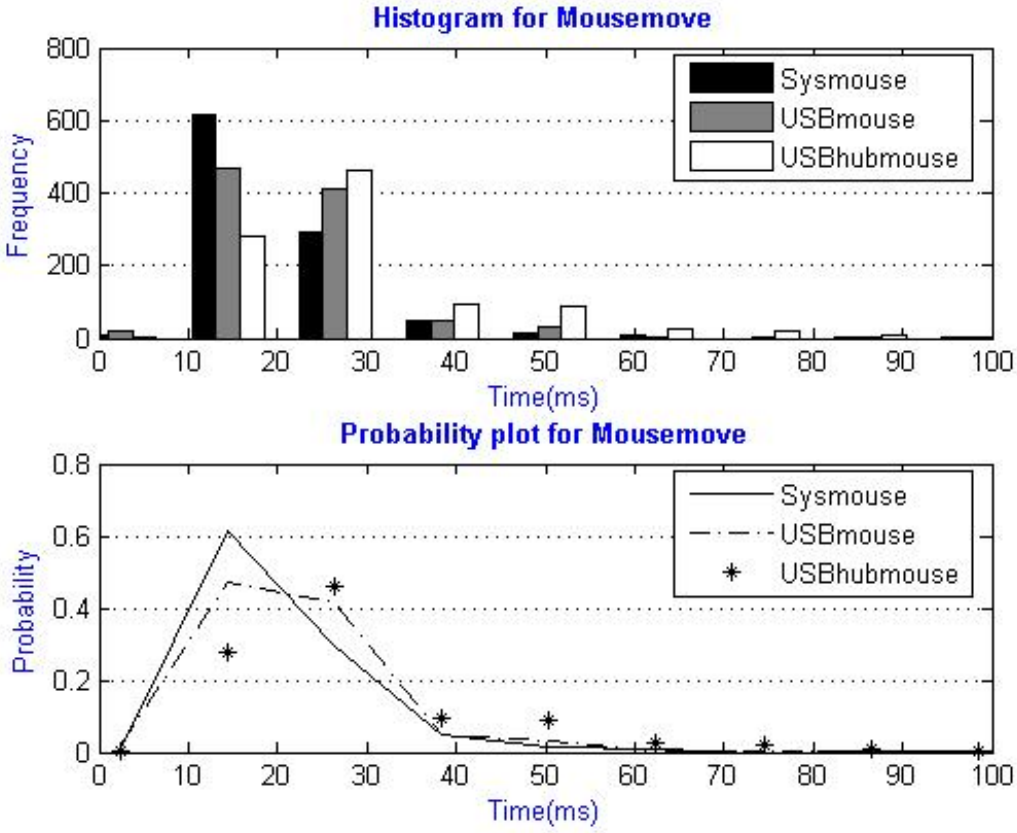


Figure 3.11: Machine1: Comparison of mousemove latencies for three interfaces: system mouse, USB mouse and mouse connected via USB hub

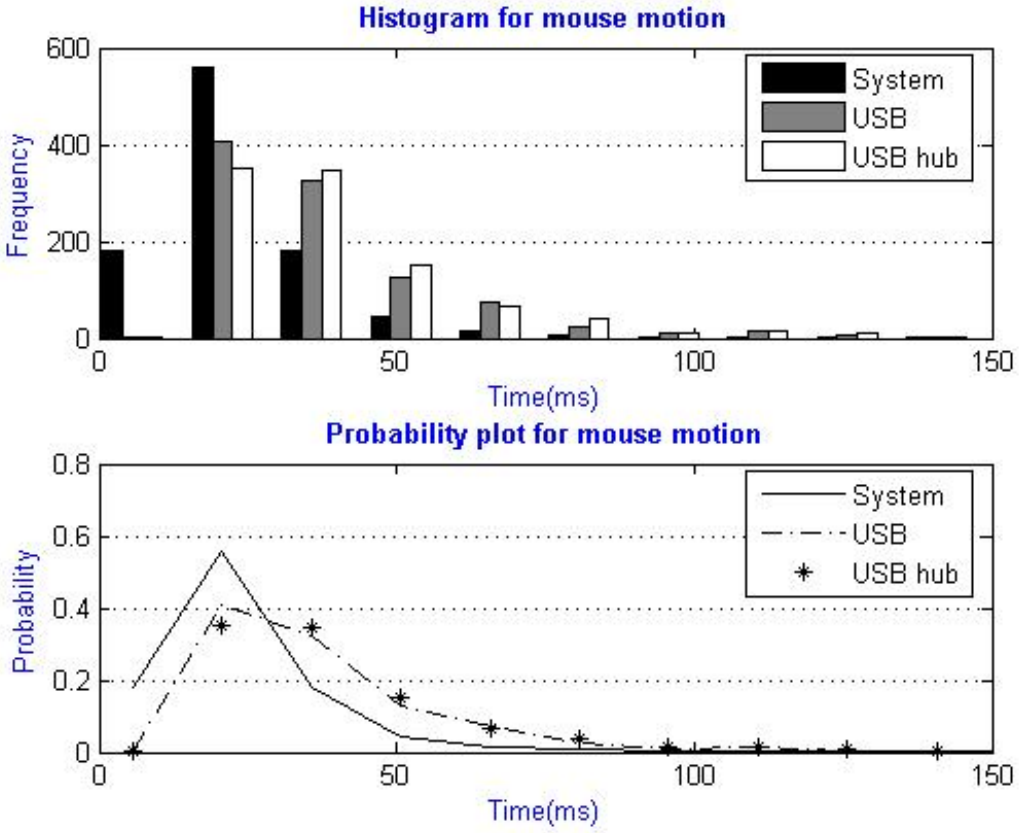


Figure 3.12: Machine2: Comparison of mousemove latencies for three interfaces: system mouse, USB mouse and mouse connected via USB hub

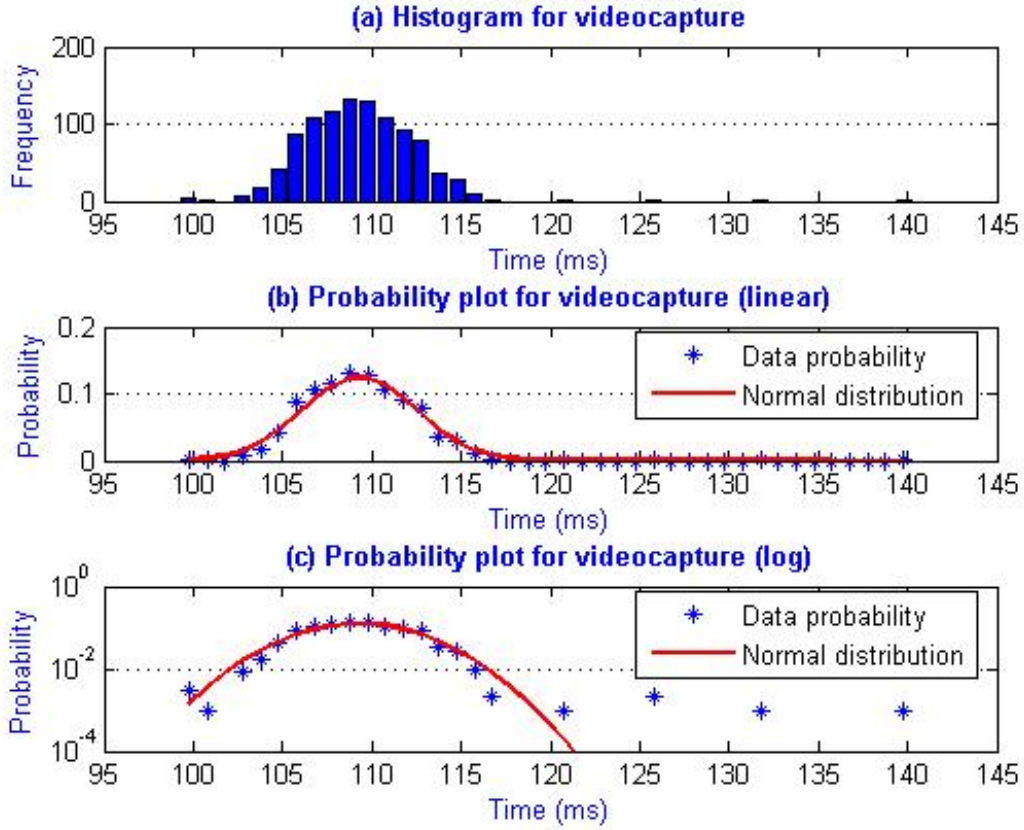


Figure 3.13: Machine1: Statistical analysis for videocapture from USB webcamera

- Webcam connected through USB
- Webcam connected through USB hub

Average values calculated from sample data described in Table 3.7 give us some idea about latency caused in capturing frames. The average value is supposed to be close to 100 ms but for both machines and for both cases, average values are greater than 100ms. From Table 3.8, one concludes that the USB hub does not introduce latency with videocapture.

In fact, on Machine2, the USB hub actually reduces the average latency and deviation compared to a direct USB connection. It can be observed that the behavior of the USB hub with webcamera is different from other devices. Probability curve of sample data closely follows the normal density curve which is evident from Figure 3.13. Hence, for real-time

Table 3.7: Statistical analysis for videocapture

| | Machine1 | | | | Machine2 | | | |
|---|---|---|---|---|---|---|---|---|
| Connection type | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| USB | 99.766 | 139.813 | 109.291 | 3.175 | 75.207 | 268.469 | 125.885 | 35.518 |
| USB hub | 95.687 | 136.102 | 109.231 | 3.059 | 76.330 | 287.400 | 119.787 | 26.448 |

Table 3.8: Comparison of means (T-test $\alpha = 0.05$) for videocapture

| Comparison | Machine1 | Machine2 |
|---|---|---|
| USB vs USB hub | h=0 (no difference) | h=1 [3.3515, 8.8445] |

applications both USB and USB hub have a greater effect on videocapture in meeting timing constraints. The graphical representation for the above statistical data in tables for both machines are shown in Figures 3.14 and 3.15.

## 3.4 Conclusion

This work measures the latency involved in using USB devices that one must consider when trying to weigh the real-time requirements of the system versus cost and reliability. A comparison of latency values of USB and USB hub is made to determine whether USB hub cause more latency. There are also interfaces other than USB that can be used in real-time applications. Statistical analysis of experimental data and comparison with models illustrates the spread and probability of time values for every event.

The concept developed from this research forms a baseline for studying the behavior of multiple devices that are communicating through USB and PCI to a real-time system at the same time. In the next chapter, a comparison of latencies of PCI and USB buses was done to decide the suitability for meeting timing constraints.

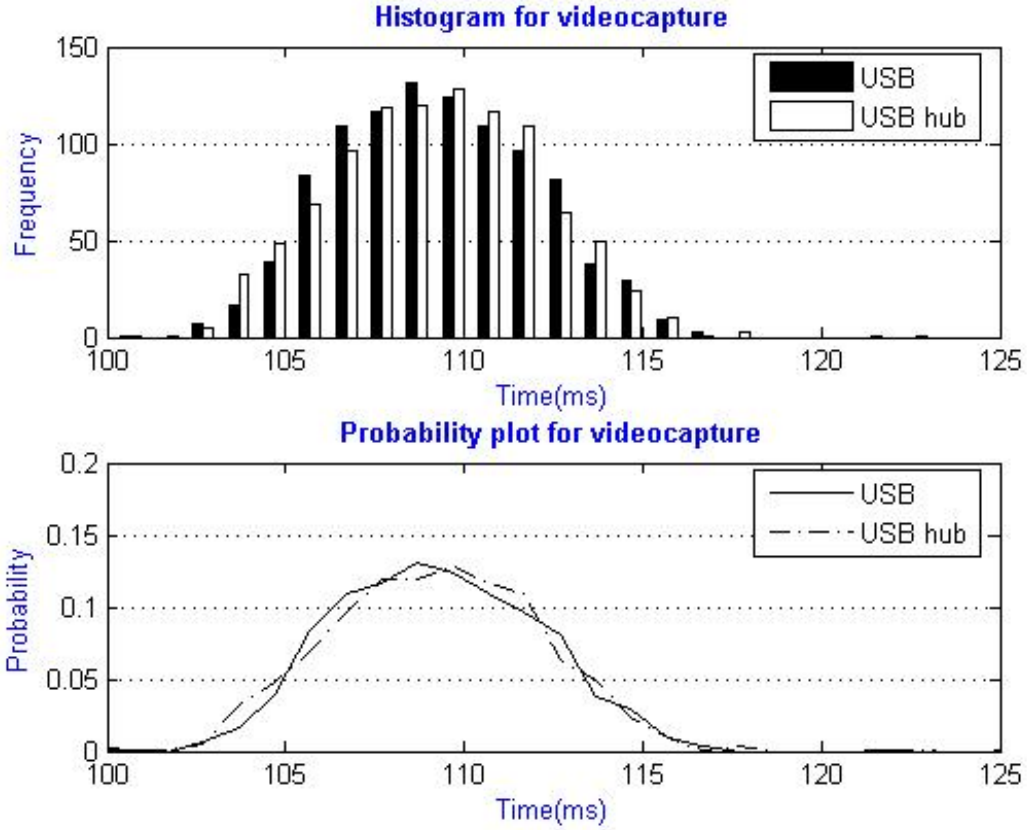


Figure 3.14: Machine1: Comparison of videocapture latencies for two interfaces: USB webcamera and webcamera connected via USB hub

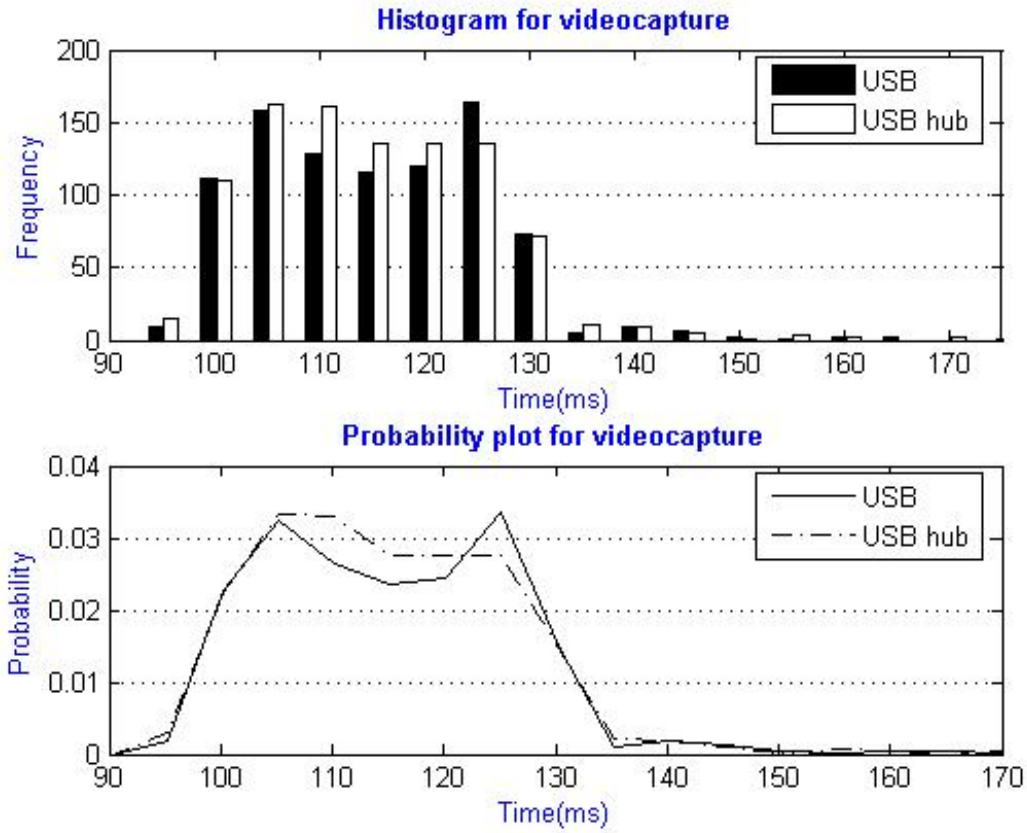


Figure 3.15: Machine2: Comparison of videocapture latencies for two interfaces: USB webcamera and webcamera connected via USB hub

# Chapter 4

## Multiple Devices - Comparison of USB and PCI Interfaces

Meeting timing constraints in a real-time system depends on the computational power of the processor, complexity of the control algorithm, operating system capabilities, and types of communication protocol used to communicate information. The impact of external device interfaces like PCI and USB in meeting timing constraints with real-time applications is studied in this chapter.

### 4.1 Introduction

PCI has become the standard in all high performance systems that require quicker IO data transfer. PCI has its widespread use not only in PC market but also in embedded systems, laptops and mobile systems. Universal Serial Bus is another popular standard for personal computer peripheral devices because of its versatile interconnection specifications. With the prominence of USB and PCI, it is interesting to explore whether PCI is better than USB for achieving predictability in real-time systems. To analyze latency with PCI, USB devices are connected to the computer via PCI/USB interface card.

### 4.2 Peripheral Component Interconnect

Peripheral Component Interconnect is an industry specification for connecting peripheral devices to CPU. Designed by Intel in 1992, it is high performance IO bus that can be configured dynamically for devices that have high bandwidth requirements. PCI bus is 64 bits wide, operates at 133 MHz speed with transfer rate of 1Gbps. PCI compliant

programmable logic devices such as FPGAs (Field Programmable Gate Arrays) are used to create PCI bus interfaces facilitating the benefits of both FPGA and PCI. Such a design requires adherence to PCI standard specification to guarantee compliance [5] [7].

PCI is an attempt to avoid data bottlenecks between processor and peripherals. Applications such as data acquisition and waveform generation require sufficient bandwidth to ensure that data can be transferred to memory quickly without being lost or overwritten. PCI Express uses high performance, point-to-point serial interface enabling faster data streaming [19].

### 4.2.1 PCI communication

PCI uses a shared bus topology that allows different PCI devices like network card, sound card, or a RAID card to be attached to the same bus, through which they communicate with the CPU. In a typical system, operating system queries all PCI buses at startup time to determine what devices are attached to PCI bus and what system resources (memory space, IO space etc) they need. It then allocates necessary resources they need and notifies the devices about its allotted resources. Devices connected to PCI have executable code in their ROM which can be read by processor and then operating system loads the corresponding device driver.

Because several devices can be attached to PCI bus, bus arbitration scheme is used to decide which device can access the bus when. Once a device has control of the bus, it becomes the bus master and can start communicating with the processor and memory. Other devices have to wait and this demand for bus may degrade performance of system [3].

All PCI devices are capable of initiating a data transfer as bus masters without the need for CPU. Hence CPU is not intervened which significantly improves system's performance.

When two or more bus masters want to transfer data at the same time, bus arbitration handles all possible scenarios and ensures fair bus access to all devices. PCI uses hidden bus arbitration mechanism in which bus arbitration will take place when bus data transfer is going on. This will reduce latency because the device is ready to transfer data once when bus will become available.

Priority is assigned to PCI devices based on the value specified by device's configuration register which is maximum latency, the time it has to wait for getting bus access. If a high priority device has been assigned bus access and if another device which is of higher priority than one that is granted bus access arrives in, then the the older device is preempted by the new device.

Once a device has started using the bus, it can begin its transaction and cannot be interrupted by other devices until it has completed the transfer. Each device has latency timer whose value indicates the guaranteed period of bus access. This latency timer is a configuration register, it is decremented for every bus cycle and when its value reaches zero, the device has to relinquish the bus for other device.

### 4.2.2 Data transfer mode

PCI uses burst mode of data transfer which offers superior performance when compared to other buses. Burst mode is faster than other modes because data is transferred in blocks and it seizes control over the bus. Also, PCI makes use of DMA (Direct Memory Access), the ability to access system memory without consulting the CPU. DMA is the natural mode of data transfer in PCI which uses burst operation [21]. This feature may specially be useful in real-time computing because current running processes are not stalled thereby avoiding the CPU overhead.

### 4.2.3 Device detection and configuration

When a new device is attached to PCI slot, configuration steps that are followed to identify the device are:

1. BIOS scans the PCI bus and tries to identify the device by sending a signal to the device.
2. The device sends back the device ID.
3. BIOS configures the device by assigning IRQ (Interrupt Request), DMA, Memory address and IO settings.
4. Operating System on boot up, checks the PCI bus and loads the corresponding device driver.
5. Once the device driver is installed, the device is ready for use.

### 4.2.4 PCI address space

When a PCI-enabled computer starts up, it initializes the PCI subsystem by allocating blocks of memory dedicated for each PCI device so that they will be accessible to the CPU. Once the devices know about their address spaces, they start listening to the bus for any commands and data in their way. The PCI target can implement three types of address spaces, configuration space, memory space and IO space.

Configuration space consists of 256 bytes of configuration registers and stores basic information about the device such as vendor and class of device. This space can be accessible at any time by configuration, initialization and error handling software. IO space is where basic peripheral devices are mapped and memory space is used as a general-purpose space.

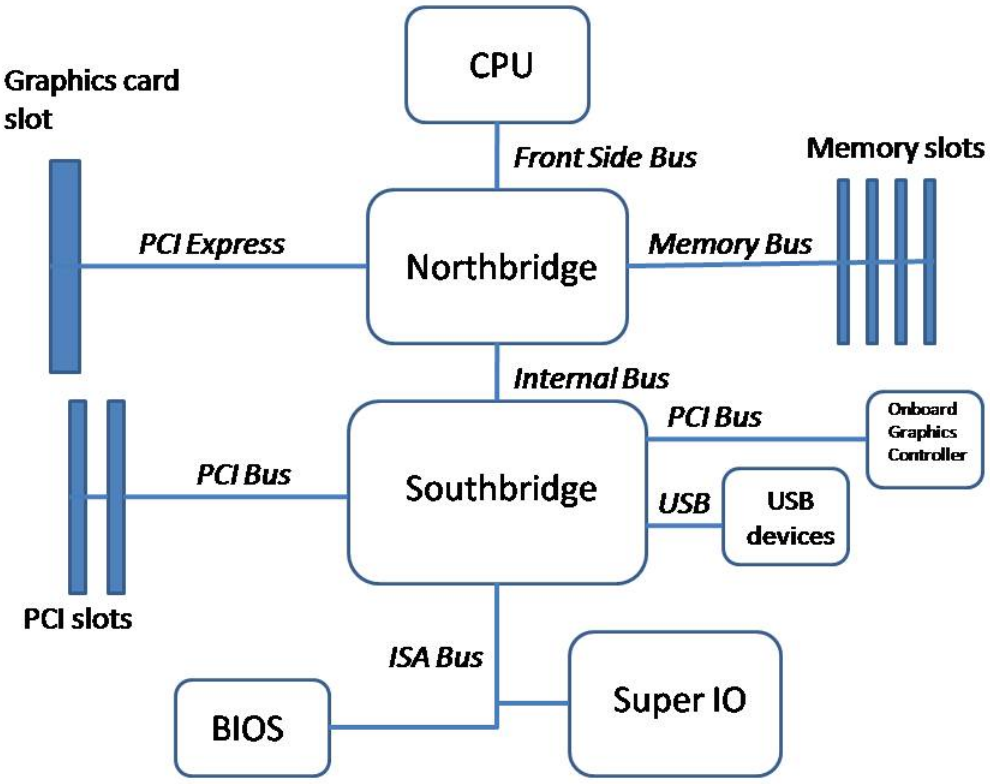


Figure 4.1: Motherboard layout

### 4.2.5 Motherboard architecture

Figure 4.1 shows the typical motherboard layout with buses, processor and chipset. The chipset consists of two chips: northbridge and southbridge. Northbridge also known as Memory controller hub controls communication between RAM, CPU and southbridge. Southbridge also known as I/O controller hub is not directly connected to CPU and handles communication between northbridge and IO devices. It is moved farther away from CPU and is responsible for slower devices.

Front Side Bus carries information between northbridge and CPU. It's frequency depends on processor speed and can operate upto 800MHz. ISA bus connects peripheral devices like mouse, keyboard, serial port to southbridge. It is 32 bits wide and operates at 8MHz with a transfer rate of 32Mbps. The memory bus is a set of wires that carry address and data information between system RAM and chipset. PCI bus has direct access to system memory but uses bridge to connect with Front Side Bus and CPU.

To support modern 3D applications, Intel established PCI Express for attaching video cards to motherboard. PCI Express based on PCI provides a scalable, high-bandwidth, low-latency interconnect. It is software compatible with PCI since no changes are necessary for device drivers at the operating systems level [14].

## 4.3 Experimental Evaluation

The experiment is performed on Windows XP, Intel Celeron processor with 512MB RAM, 2.4GHz processor speed, 60GB hard disk. Software is written in VB.NET and timing is done using QueryPerformanceCounter. This function retrieves the values of the highest resolution timer offered by Windows XP. The resolution of the timer with this hardware configuration is 2.79365114840015E-07 seconds. The experiments were run for the following two cases:

Table 4.1: Statistical analysis of keypress

| Event(s) | USB | | | | PCI | | | |
|---|---|---|---|---|---|---|---|---|
| | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| Kp only | 6.9601 | 68.3048 | 33.2412 | 3.8134 | 15.8383 | 50.6257 | 33.2081 | 2.0826 |
| Kp+Mm | 8.2052 | 144.1387 | 40.7836 | 12.4219 | 14.7418 | 84.0459 | 42.2598 | 11.7989 |
| Kp+Vc | 4.6394 | 112.4615 | 35.0038 | 14.3682 | 8.3642 | 72.1486 | 33.6172 | 11.0430 |

Table 4.2: Comparison of means (T-test $\alpha = 0.05$) for keypress

| Events | USB | PCI |
|---|---|---|
| Kp vs. kp+Mm | h=1 [-8.3486, -6.7363] | h=1 [-9.7951, -8.3082] |
| Kp vs. Kp+Vc | h=1 [-2.6849, -0.8402] | h=0 (no difference) |
| Kp+Mm vs. Kp+Vc | h=1 [4.6019, 6.9578] | h=1 [7.6403, 9.6448] |

- analyzing the behavior and measuring the latency effects of one device when another device is communicating to the system through USB bus
- analyzing the behavior and measuring the latency effects of one device when another device is communicating to the system through PCI bus with USB/PCI interface card.

### 4.3.1 Keypress latency

Time taken to read characters from keyboard is measured using QPC. This part focuses on evaluating the behavior of keyboard with the addition of webcam and mouse when connected to USB and PCI buses. The test was conducted to determine time of keypress event with

- only keypress (Kp)
- keypress and mousemove, both done simultaneously (Kp+Mm)
- keypress and videocapture, both done simultaneously (Kp+Vc)

These three test cases were analyzed for USB and PCI. Figures 4.2 and 4.3 indicate the fact that time of keypress event is affected by the addition of another event like mousemove and videocapture since the probability curves are more spread apart for keypress with mousemove and keypress with videocapture. Also, statistical data (see Table 4.1) illustrates

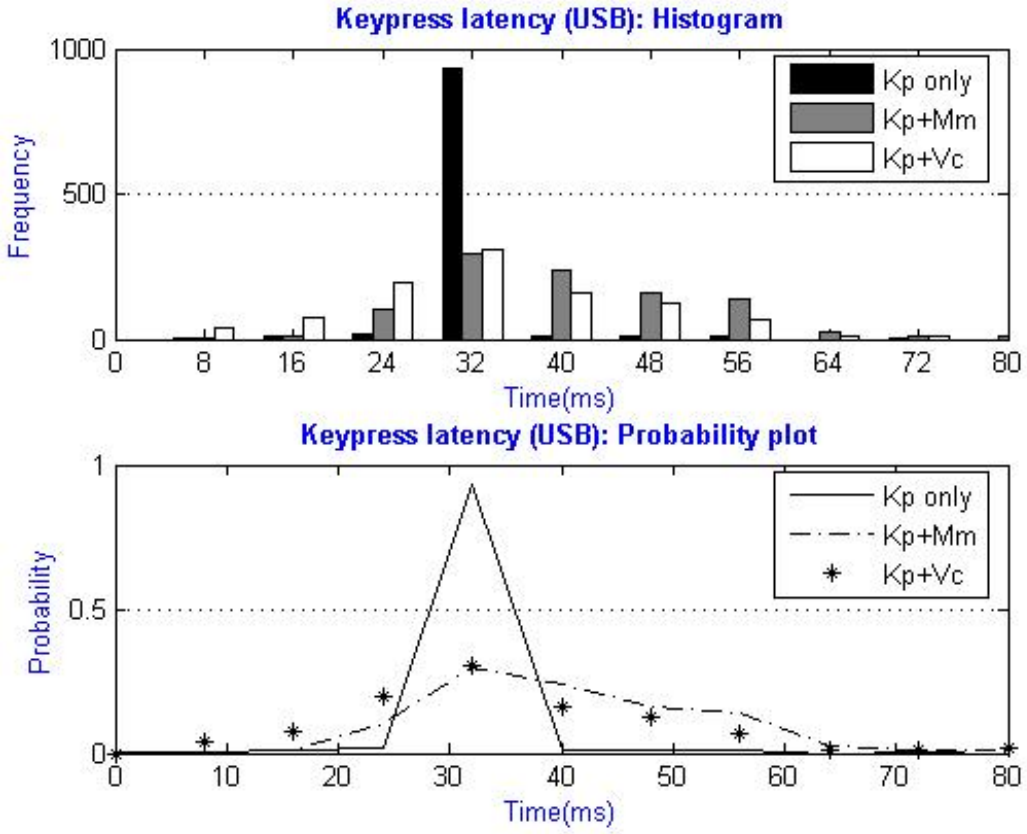


Figure 4.2: USB: Comparison of latencies for three cases: keypress only, keypress with mousemove and keypress with videocapture

Table 4.3: Comparison of means (T-test $\alpha = 0.05$) for keypress

| Events | USB vs. PCI |
|---|---|
| Kp only | h=0 (no difference) |
| Kp+Mm | h=1 [-2.5387, -0.4137] |
| Kp+Vc | h=1 [0.2626, 2.5104] |

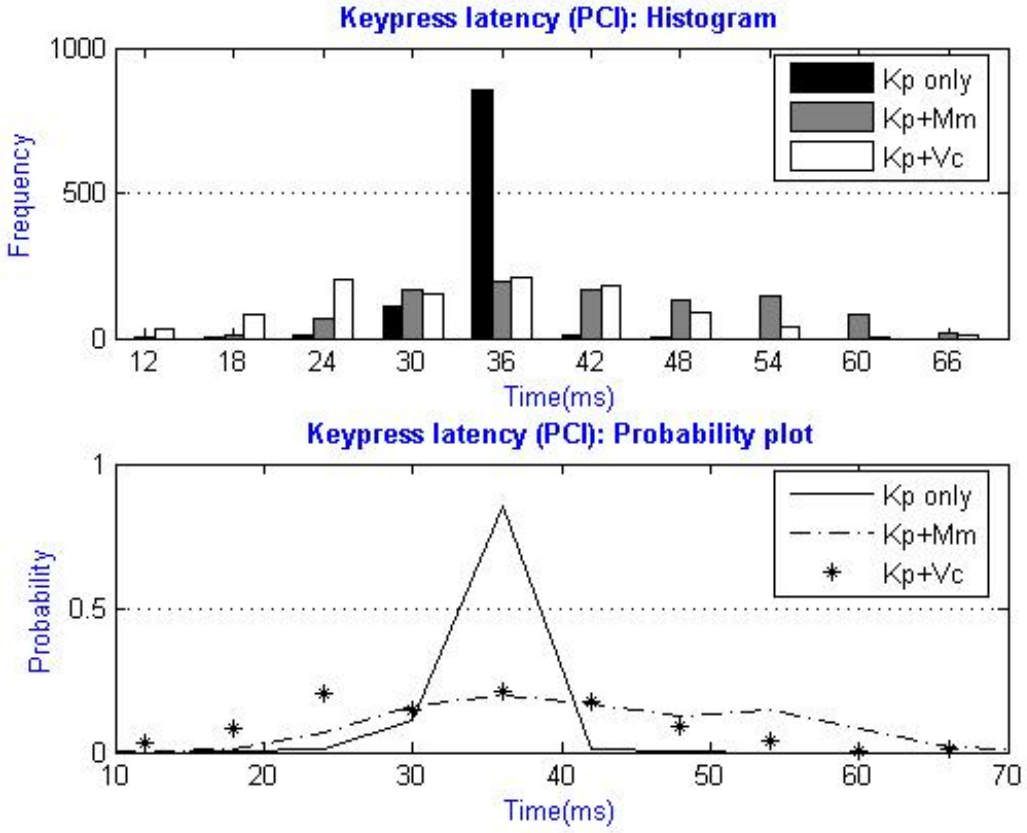


Figure 4.3: PCI: Comparison of latencies for three cases: keypress only, keypress with mousemove and keypress with videocapture

Table 4.4: Statistical analysis of fileread

| | USB | | | | PCI | | | |
|---|---|---|---|---|---|---|---|---|
| Event(s) | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| Fr only | 1.0216 | 23.9824 | 1.4948 | 1.3150 | 1.0236 | 19.7858 | 1.4196 | 0.7664 |
| Fr+Vc | 1.2015 | 48.2502 | 2.5407 | 4.4324 | 1.1507 | 43.8056 | 2.9127 | 5.2222 |

Table 4.5: Comparison of means (T-test $\alpha = 0.05$) for fileread

| Events | USB | PCI |
|---|---|---|
| Fr vs. Fr+Vc | h=1 [-1.3328, -0.7591] | h=1 [-1.8207, -1.1656] |

the fact that mouse has greater effect on keypress than webcam because mouse and keyboard uses same type of data transfer, interrupt transfer.

Based on T-tests results from Table 4.2 and 4.3, it can be stated that when a real-time system needs to get data from webcam and keyboard simultaneously, PCI would be a better protocol. But when keyboard and mouse are connected, USB would be a better option than PCI. For both PCI and USB, with addition of either mouse or webcam, the standard deviation has increased as high as 5 times. Hence, data transfer from keyboard in both buses is significantly affected by the addition of either webcam or mouse though the extent of effects on both cases are different.

### 4.3.2 Fileread latency

To experiment the effects of webcam with USB memory, time taken to read a text file from USB disk drive was measured when webcam is capturing and displaying video. The file was read 1000 times and time taken for every file read (Fr) is noted. This experimental run was done in two stages:

- only USB disk drive connected to USB port (Fr)
- USB disk drive and webcam connected to USB ports, webcam was in function when file read was done (Fr+Vc)

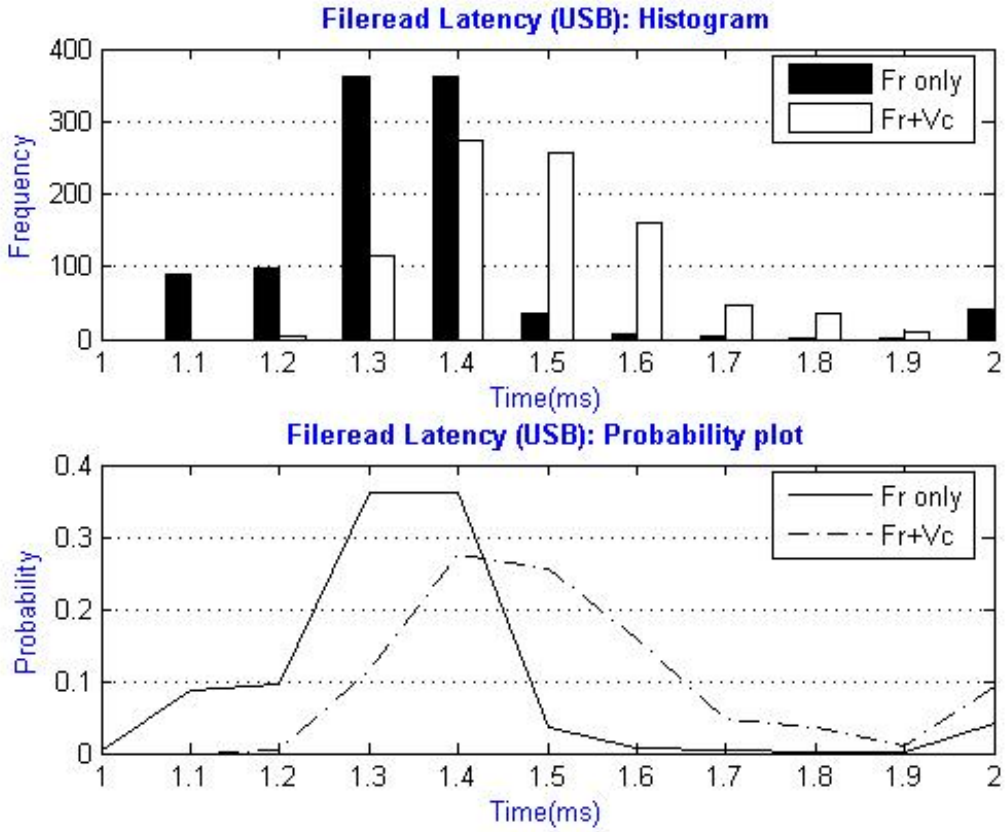


Figure 4.4: USB: Comparison of latencies for two cases: fileread only and fileread with videocapture

Table 4.6: Comparison of means (T-test $\alpha = 0.05$) for fileread

| Events | USB vs. PCI |
|---|---|
| Fr only | h=0 (no difference) |
| Fr+Vc | h=0 (no difference) |

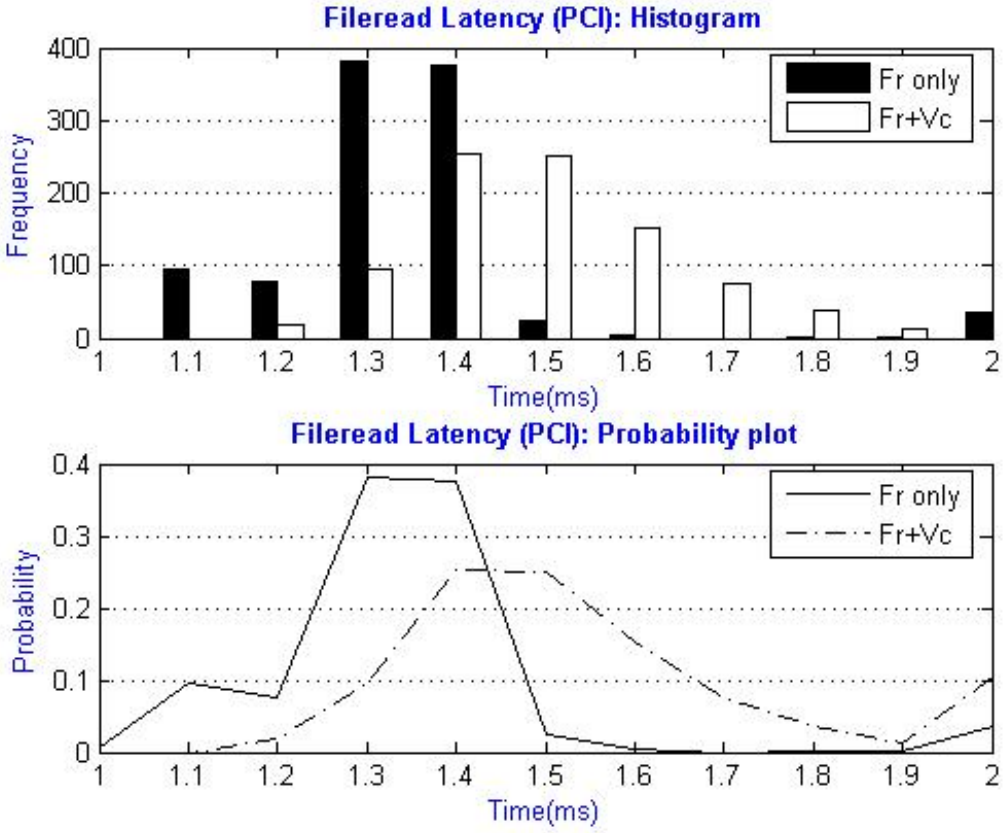


Figure 4.5: PCI: Comparison of latencies for two cases: fileread only and fileread with videocapture

Table 4.7: Statistical analysis of mousemove

| Event(s) | USB | | | | PCI | | | |
|---|---|---|---|---|---|---|---|---|
| | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| Mm only | 5.4630 | 135.9530 | 18.1194 | 9.7388 | 5.3786 | 106.3942 | 16.9752 | 10.1648 |
| Mm+Kp | 5.1102 | 80.7929 | 25.7740 | 13.8153 | 4.2827 | 51.8130 | 18.4522 | 8.2534 |

Table 4.8: Comparison of means (T-test $\alpha = 0.05$) for mousemove

| Events | USB | PCI |
|---|---|---|
| Mm vs. Mm+Kp | h=1 [-8.7029, -6.6062] | h=1 [-2.2891, -0.6650] |

These test cases were run with PCI and USB. Figures 4.4 and 4.5 show that probability curve for fileread with webcam is shifted from that of curve for fileread only and also it is wider. There is an increase in the minimum and maximum values for time taken for fileread during videocapture (see Table 4.4).

The average time for fileread with webcam is 1 ms more than that of fileread without webcam and standard deviation is also higher. Hence, when both webcam and USB disk drive are connected to the system, webcam affects the behavior of USB memory by introducing significant overhead (See Table 4.5). However, when T-tests were performed for comparing the means of fileread with USB and PCI, Table 4.6 illustrates that both offer the same level of performance for USB memory.

### 4.3.3 Mousemotion latency

The experimental approach in this section explains the effects of keyboard over mouse. Time was measured for every mousemove event and 1000 samples were collected for statistical analysis. The tests were conducted for the following possible cases with PCI and USB buses.

- only mousemove (Mm)
- when mouse is moved, keyboard is pressed simultaneously (Mm+Kp)

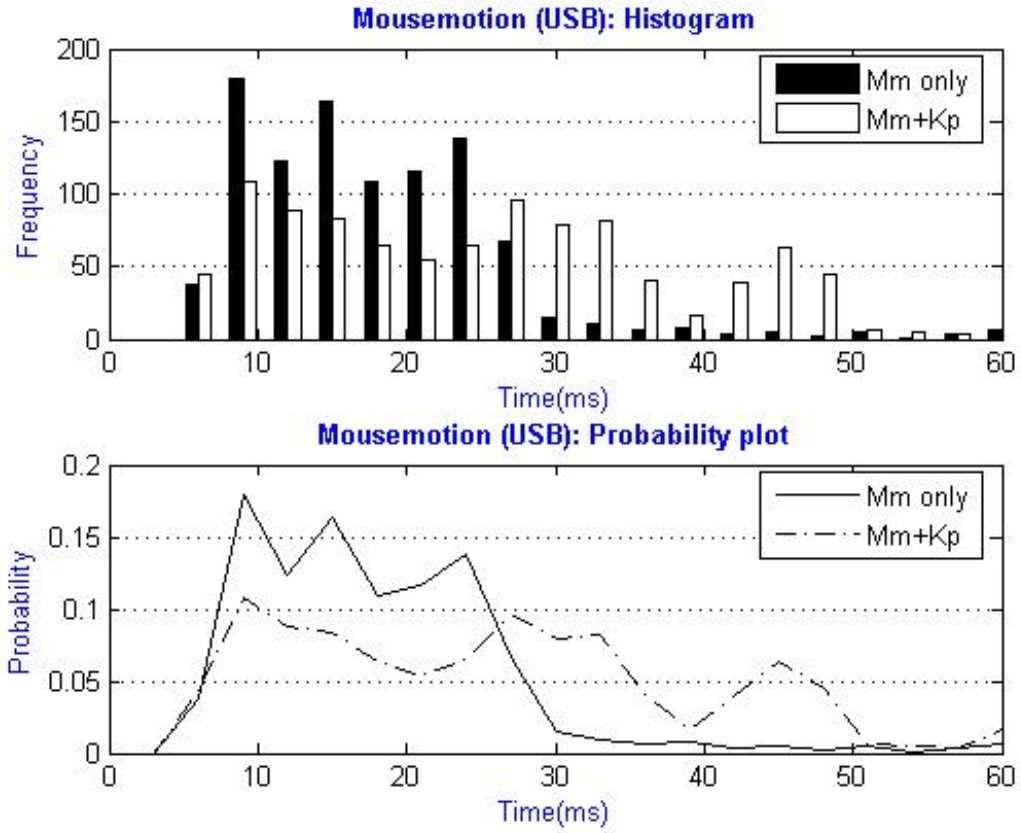


Figure 4.6: USB: Comparison of latencies for two cases: mousemove only and mousemove with keypress

Table 4.9: Comparison of means (T-test $\alpha = 0.05$) for mousemove

| Events | USB vs. PCI |
|---|---|
| Mm only | h=1 [0.2712, 2.0172] |
| Mm+Kp | h=1 [6.3236, 8.3199] |

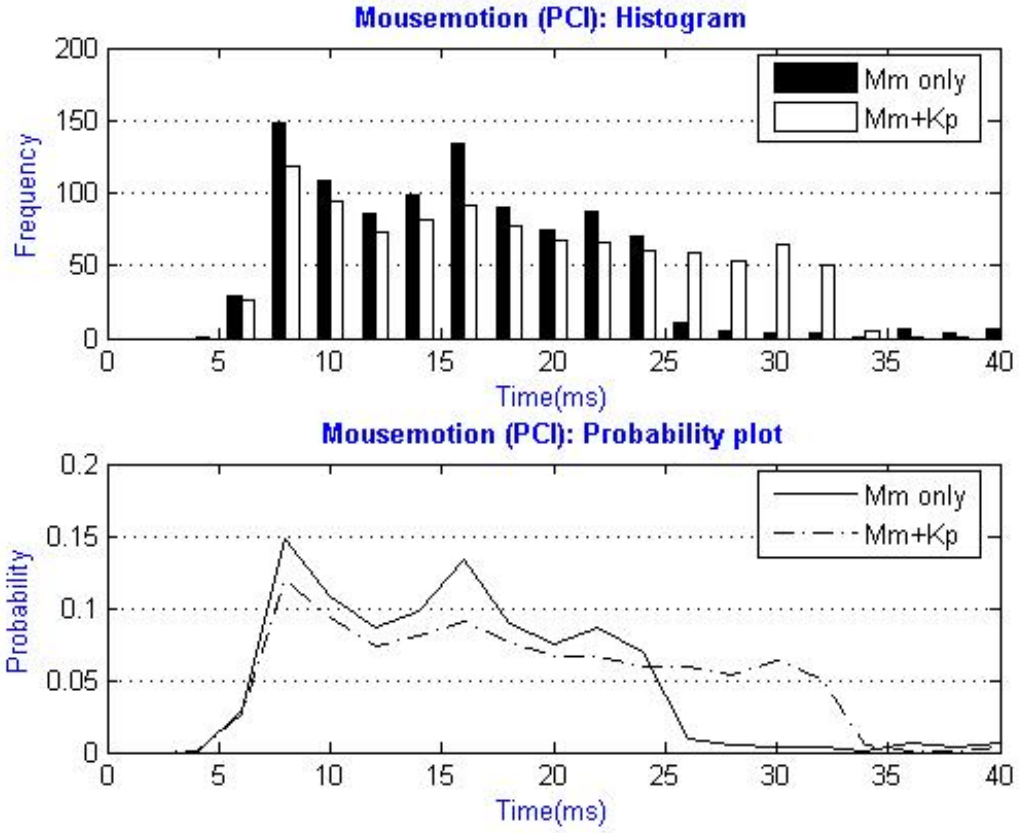


Figure 4.7: PCI: Comparison of latencies for two cases: mousemove only and mousemove with keypress

Table 4.10: Statistical analysis of videocapture

| | USB | | | | PCI | | | |
|---|---|---|---|---|---|---|---|---|
| Event(s) | Min(ms) | Max(ms) | Avg(ms) | std | Min(ms) | Max(ms) | Avg(ms) | std |
| Vc only | 83.7190 | 355.6731 | 112.5570 | 13.36184 | 84.7834 | 146.2632 | 111.3936 | 8.9181 |
| Vc+Kp | 39.2033 | 260.1895 | 113.3746 | 21.5718 | 60.2842 | 263.8042 | 113.2646 | 21.2167 |

Table 4.11: Comparison of means (T-test $\alpha = 0.05$) for videocapture

| Events | USB | PCI |
|---|---|---|
| Vc vs. Vc+Kp | h=0 (no difference) | h=1 [-3.2987, -0.4432] |

From Table 4.7 it can be observed that average value for mousemove with keypress is higher than just mousemove for both USB and PCI. An uniformity in mousemove latency between PCI and USB is that curves for mousemove with keypress is more spread than only mousemove which is evident from Figure 4.6 and Figure 4.7.

The effect of keypress on mousemove is less with PCI than USB (see Table 4.8). Comparing the means of USB and PCI with T-tests (see Table 4.9), USB takes more time than PCI for both mousemove event and mousemove with keypress. Hence, PCI offers quicker response and better level of performance than USB for real-time applications that communicate to mouse and keyboard at the same time.

### 4.3.4 Videocapture latency

This experimental part investigates whether keyboard affects the timing of videocapture with webcam. The preview rate for videocapture is set to 100 ms. Ideally, the system is expected to capture one frame for every 100 ms. But frame capture rate of 100 ms was not observed with both USB and PCI. The actual time taken for capturing 1000 frames is measured for the following possibilities with USB and PCI.

- only videocapture (Vc)
- videocapture and keypress are done simultaneously (Vc+Kp)

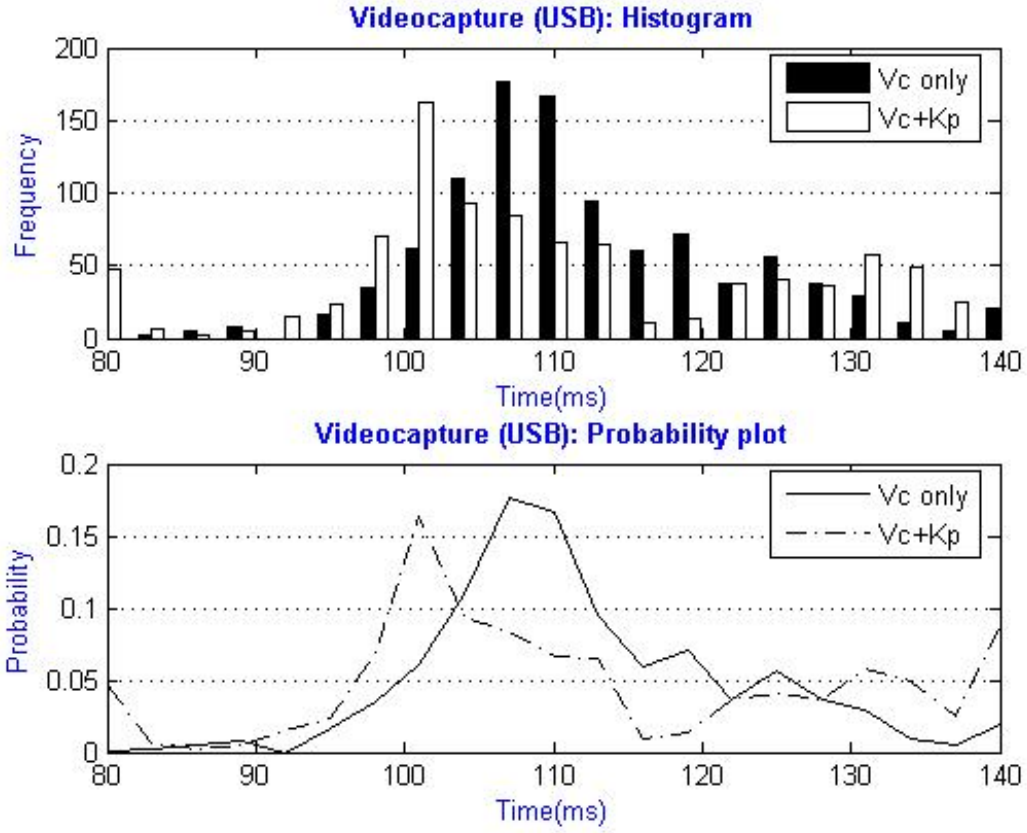


Figure 4.8: USB: Comparison of latencies for two cases: videocapture only and videocapture with keypress

Table 4.12: Comparison of means (T-test $\alpha = 0.05$) for videocapture

| Events | USB vs. PCI |
|---|---|
| Vc only | h=1 [0.1670 2.1597] |
| Vc+Kp | h=0 (no difference) |

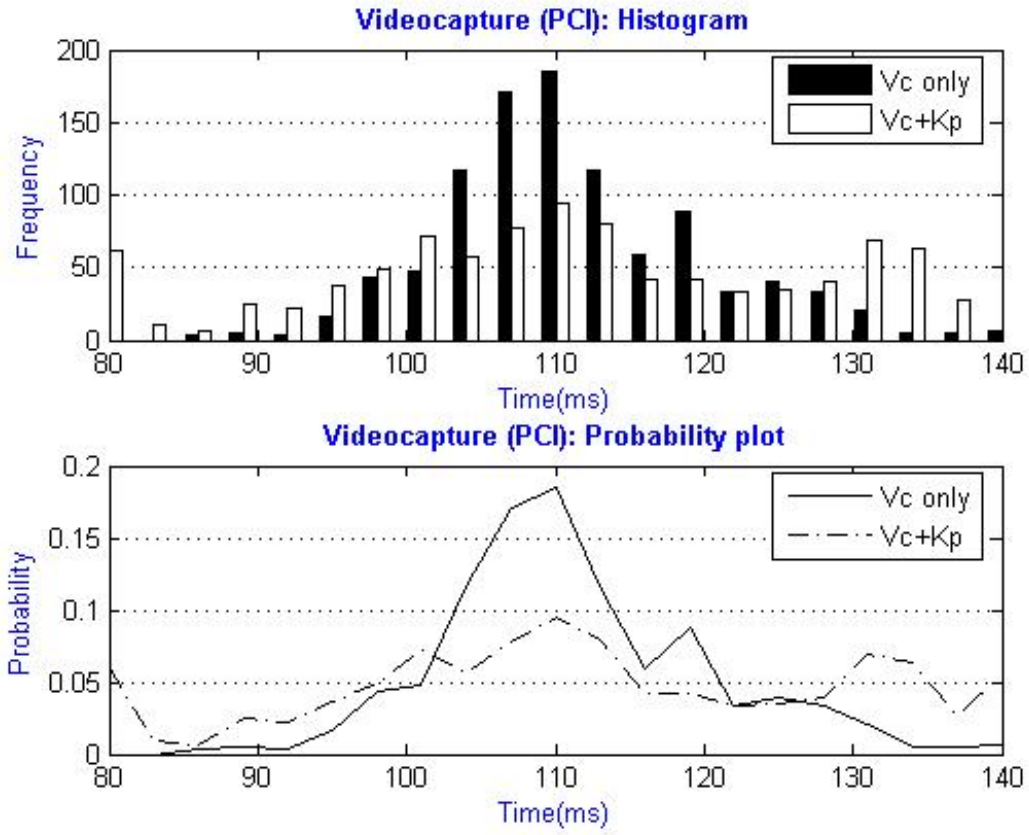


Figure 4.9: PCI: Comparison of latencies for two cases: videocapture only and videocapture with keypress

Behavior of webcam is slightly different from other devices because it could be inferred from Table 4.10 that the delay introduced by keyboard on videocapture is not significant (less than a millisecond for USB and less than 2 ms for PCI). The overhead caused by keypress on videocapture is illustrated in Figures 4.8 and 4.9.

Standard deviation of videocapture with keypress is higher than with only videocapture. This indicates that though videocapture's delay with keypress is less, it becomes more unpredictable when kepress event occurs. With USB, addition of keypress has almost null effect on timing of videocapture (See Table 4.11). Comparing the T-tests results of USB and PCI from Table 4.12, it can be inferred that PCI and USB offer the same level of performance for videocapture with keypress and PCI incurs less overhead than USB with only videocapture.

## 4.4 Conclusion

PCI caters to the increasing bandwidth requirements of high-speed applications where a faster system interconnect bus is required. USB is popular for its expandability and simplified hardware connectivity. The choice of particular communication protocol is one of the fundamental design decisions for a real-time control engineer. It can be inferred from this study that USB and PCI offer different levels of real-time performance and the choice of particular type of standard depends on the type and number of devices used.

# Chapter 5

# Conclusions, Recommendations and Future Research

## 5.1 Conclusions

Determinism in real-time systems is affected by computer hardware, software, operating system and external device interfaces. Real-time control systems that operate on inputs from USB peripheral devices have become more common because the USB interface eases the task of installation with plug and play.

A key element of this study has been to determine to what extent a Windows-based real-time system is unpredictable when it uses USB for data input. To better evaluate the performance, time for events with devices in the system were measured and then compared with time for events with devices connected through USB and USB hub. Statistical analysis shows that latency values of USB and USB hub are higher than system devices that affect predictability.

Most real-time applications get input from the environment through peripheral devices attached to them. With both USB and PCI, there can be significant impact on the latency of one device when another device is added to the system that is sending input simultaneously. Hence, as the number of I/O devices communicating to the real-time system increases, determinism is jeopardized.

## 5.2 Recommendations

It can inferred from this work that low-cost Windows based systems are suitable for building "soft" real-time applications that can tolerate delays in the order of few milliseconds. RTOS is a better option for hard real-time systems. To get input from keyboard and mouse, it is better to use the ones that are attached to the system through PS/2 interface than attaching them to the USB because the latency values are lower for devices connected through PS/2. Also, accessing data from hard disk is faster than accessing data from USB memory.

USB hub does not introduce significant latency with peripheral devices. Hence, deployment of USB hub to attach multiple devices to the real-time system simplifies the connection and will not add overhead to USB. When a real-time system needs to communicate with multiple devices at the same time, PCI is a better option than USB because the latency values for peripheral devices connected through PCI/USB interface card to the PCI is lower than latency values for peripheral devices connected through USB.

## 5.3 Directions for future work

The behavior of a real-time system with multiple heterogeneous USB devices would be an interesting area to explore. It is a theoretical issue that as the number of USB of devices connected to the system increases, latency increases. This work would be a baseline for further study on latency with low-cost Windows-based real-time systems with multiple devices communicating to them at the same time. There are many USB devices used in real-time control applications. Research on this domain could be done to observe the effects of latency with different types of devices.

Firewire, also known as IEEE 1394, is a serial bus interface standard used for high-speed communications and real-time data transfers. Some of the interesting features of

Firewire are plug and play performance, expandability up to 63 devices, peer-to-peer device communication, and high data transfer rates of up to 800 Mbps. The key difference between USB and Firewire is that Firewire is used where bulk data is to be transferred like camcorders, DVD players and digital audio equipment. Future research can examine the latency of Firewire. The topology is different from USB and it has not gained the same acceptance as USB, primarily because the hardware cost is higher.

# Bibliography


[1] Don Anderson. *Universal Serial Bus System Architecture*. Addison Wesley Developers Press, 1997.

[2] Allan Baril. Using Windows NT in real-time systems. pages 132–141, 1999.

[3] Al Chame. PCI bus in high speed I/O systems applications. In *Proceedings of IEEE Aerospace Conference*, 1998.

[4] Matjaz Colnaric. State of the art review paper: Advances in embedded hard real-time systems design. In *Industrial Electronics*, 1999.

[5] Pablo Costi-Kowolik and Roberto Gutierrez-Mazon. A first approach towards a high-speed PCI-based data acquisition card for industrial applications. In *IEEE 24th Convention of Electrical and Electronics Engineers*, 2006.

[6] A. Depari, A. Flammini, D. Marioli, and A. Taroni. USB sensor network for industrial applications. In *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference*, 2004.

[7] E. Finkelstein and S. Weiss. Implementation of PCI-based systems using programmable logic. In *IEE proceedings of Circuits, Devices and Systems*, 2000.

[8] Alceu Heinke Frigeri, Carlos E. Pereira, and Wolf. An object-oriented extension to PEARL90. In *Object-Oriented Real-Time Distributed Computing*, 1998.

[9] Chih-Yuan Huang, Li-Pin Chang, and Tei-Wei Kuo. A cyclic-based QoS guarantee over USB. In *Real-Time and Embedded Technology and Applications Symposium*, 2003.

[10] Shyh-In Hwang, Chia Mei Chen, and Ashok K. Agrawala. Scheduling an overloaded real-time system. In *Computers and Communications. Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference*, 1996.

[11] John Hyde. *USB Design by Example: A practical guide to building IO devices*. John Wiley & Sons, 1999.

[12] Yutaka Ishikawa, Hideyuki Tokuda, and Clifford W. Mercer. An object-oriented real-time programming language. *IEEE Computer Magazine*, 25:66–73, 1992.

[13] Philip A. Laplante. Design issues in real-time. *IEEE Potentials*, 12:24–26, 1993.

[14] Jiuxing Liu, Amith Mamidala, Abhinav Vishnu, and Dhabaleswar K Panda. Performance evaluation of infiniband with PCI Express. In *Proceedings of 12th Annual IEEE Symposium High Performance Interconnects*, 2004.

[15] Ulrich Muehlmam, Miguel Ribo, Peter Lag, and Axel Pinz. A new high speed CMOS camera for real-time tracking applications. In *Proceedings of ICRA'04 IEEE International Conference on Robotics and Automation*, 2004.

[16] Kevin M. Obenland, Tiffany Frazier, Jin S. Kim, and John Kowalik. Comparing the real-time performance of Windows NT to an real-time extension. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, 1999.

[17] Krithi Ramamritham, Chia Shen, Oscar Gonzale, and Shreedhar Shirgurkar. Using Windows NT for real-time applications: Experimental observations and recommendations. In *Proceedings of Fourth IEEE Real-Time Technology and Applications Symposium*, pages 102–111, 1998.

[18] Krithi Ramamritham and John A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82:55–67, 1994.

[19] Murali Ravindran. Cabled PCI Express-A standard high-speed instrument interconnect. In *IEEE Autotestcon Magazine*, 2007.

[20] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer Magazine*, 21:10–19, 1988.

[21] J. Toledo, H. Muller, J. Buytaert, F. Bal, A. David, A. Guirao, and F. J. Mora. A plug and play approach to data acquisition. *IEEE transactions on Nuclear Science*, 49:1190–1194, 2002.

# Appendices

# Appendix A

## VB.NET code

```
Imports System

Imports System.Globalization

Imports System.Threading

Imports System.IO

Imports System.Runtime.InteropServices

Public Class Form1

Inherits System.Windows.Forms.Form


'declare all parameters for capturing

and displaying video


Const WM_CAP As Short = &H400S

Const WM_CAP_DRIVER_CONNECT As

Integer=WM_CAP + 10

Const WM_CAP_DRIVER_DISCONNECT

As Integer =WM_CAP + 11

Const WM_CAP_EDIT_COPY

As Integer = WM_CAP + 30

Const WM_CAP_SET_CALLBACK_FRAME

As Integer = WM_CAP + 5
```

```
Const WM_CAP_SET_PREVIEW
As Integer = WM_CAP + 50
Const WM_CAP_SET_PREVIEWRATE
As Integer = WM_CAP + 52
Const WM_CAP_SET_SCALE
As Integer = WM_CAP + 53
Const WS_CHILD As Integer = &H40000000
Const WS_VISIBLE As Integer = &H10000000
Const SWP_NOMOVE As Short = &H2S
Const SWP_NOSIZE As Short = 1
Const SWP_NOZORDER As Short = &H4S
Const HWND_BOTTOM As Short = 1

' Current device ID
Dim iDevice As Integer = 0

'Handle to preview window
Dim hHwnd As Integer

'function for sending messages to window
Declare Function SendMessage Lib "user32"
Alias "SendMessageA" _(ByVal hwnd As Integer,
ByVal wMsg As Integer, ByVal wParam As Integer, _
<MarshalAs(UnmanagedType.AsAny)> ByVal lParam
As Object) As Integer
```

```
'function to set position and size of the window
Declare Function SetWindowPos Lib "user32" Alias
"SetWindowPos" (ByVal hwnd As Integer, _ByVal
hWndInsertAfter As Integer, ByVal x As Integer,
ByVal y As Integer, _ByVal cx As Integer,
ByVal cy As Integer, ByVal wFlags As Integer)
As Integer

'function to destroy window
Declare Function DestroyWindow Lib
"user32" (ByVal hndw As Integer) As Boolean

'function to create capture window
Declare Function capCreateCaptureWindowA Lib
"avicap32.dll" _(ByVal lpszWindowName As
String, ByVal dwStyle As Integer, _ByVal x As
Integer, ByVal y As Integer, ByVal nWidth As Integer, _
ByVal nHeight As Short, ByVal hWndParent As
Integer, _ByVal nID As Integer) As Integer

'function to be called every time when
frame is captured
Declare Function capSetCallbackOnFrame Lib
"vfw32.lib" (ByVal hpWnd As Long,
```

```
ByVal lpProc As Long) As Long ' Boolean


'function to execute the code in another domain
Delegate Function CallBackDelegate(ByVal
hwnd As IntPtr, ByRef lpVHdr As VIDEOHDR) As IntPtr
Public delCallBack As CallBackDelegate = New
CallBackDelegate(AddressOf FrameCallbackTarget)
Public Declare Function SendMessage2 Lib
"user32.dll" Alias "SendMessageA"
(ByVal hWnd As IntPtr, ByVal msg As Integer,
ByVal wParam As IntPtr, ByVal lParam As
CallBackDelegate) As IntPtr


Structure VIDEOHDR
'address of video buffer
Dim lpData As Integer
'size, in bytes, of the Data buffer
Dim dwBufferLength As Integer
Dim dwBytesUsed As Integer
Dim dwTimeCaptured As Integer
Dim dwUser As Integer
Dim dwFlags As Integer
<VBFixedArray(3)> Dim dwReserved() As Integer
End Structure
```

```
Declare Function capGetDriverDescriptionA
Lib "avicap32.dll" (ByVal wDriver As Short,
_ ByVal lpszName As String, ByVal cbName As Integer,
ByVal lpszVer As String,
_ ByVal cbVer As Integer) As Boolean

'function for retrieving QueryPerformanceCounter
and frequency values
Public Class PerfCount
<System.Runtime.InteropServices.DllImport
("Kernel32.dll")> _
Public Shared Function QueryPerformanceCounter
(ByRef perfcount As Int64) As Boolean
End Function
<System.Runtime.InteropServices.DllImport
("Kernel32.dll")> _
Public Shared Function QueryPerformanceFrequency
ByRef freq As Int64) As Boolean
End Function

'timer values
Public Shared Function QueryPerformanceCounter()
As Int64
Dim perfcount As Int64
QueryPerformanceCounter(perfcount)
```

```
Return perfcount
End Function

'frequency value
Public Shared Function QueryPerformanceFrequency()
As Int64
Dim freq As Int64
QueryPerformanceFrequency(freq)
Return freq
End Function
End Class
Dim objStreamWriter As StreamWriter

' procedure for writing frequency and
resolution of Queryperformancecounter into text file
Private Sub Button9_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button9.Click
Dim freq As Int64 = PerfCount.QueryPerformanceFrequency()
Dim Filename As String = "c:\Results\Freq&Res.txt"
objStreamWriter = New StreamWriter(Filename)
objStreamWriter.WriteLine("Frequency of the
counter is: " & freq)
objStreamWriter.WriteLine("Resolution of the
counter is: " & (1 / freq))
objStreamWriter.Close()
```

```
End Sub

'procedure for creating text file with date and
time stamp to write timer values of keypress events

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
Dim str As String = "c:\Results\kp " &
Format(Now, "MMddyy_HHmmss") & ".txt"
objStreamWriter = New StreamWriter(str, True)
TextBox2.Text = str
objStreamWriter.Close()
End Sub

'procedure for timing keypress events
Private Sub TextBox3_KeyPress(ByVal sender As Object,
ByVal e As System.Windows.Forms.KeyPressEventArgs)
Handles TextBox3.KeyPress
TextBox3.Text += e.KeyChar
Dim startcount = PerfCount.QueryPerformanceCounter()
objStreamWriter = New StreamWriter(TextBox2.Text, True)
objStreamWriter.WriteLine(startcount)
objStreamWriter.Close()
End Sub
```

```
'procedure for timing mousemove events

Private Sub Form1_MouseMove(ByVal sender As Object,
ByVal e As System.Windows.Forms.MouseEventArgs)
Handles Me.MouseMove
Label1.Text = "X: " + CStr(e.X)
Label2.Text = "Y:  " + CStr(e.Y)
Dim startcount1 = PerfCount.QueryPerformanceCounter()
TextBox5.Text += CStr(startcount1) & vbCrLf
End Sub

'procedure for creating new text file for writing
timer values of mousemove events into text file

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
Dim str1 As String = "c:\Results\mm " &
Format(Now, "MMddyy_HHmmss") & ".txt"
objStreamWriter = New StreamWriter(str1, True)
TextBox4.Text = str1
objStreamWriter.WriteLine(TextBox5.Text)
objStreamWriter.Close()
End Sub

'procedure that writes timer values of keypress
```

```
event in textbox in form
Private Sub Button10_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button10.Click
Dim startcount = PerfCount.QueryPerformanceCounter()
TextBox1.Text += CStr(startcount) & vbCrLf
End Sub

'procedure for creating a new text file for writing timer
values of mouseclick event
Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
Dim str1 As String = "c:\Results\mc " &
Format(Now, "MMddyy_HHmmss") & ".txt"
objStreamWriter = New StreamWriter(str1, True)
TextBox6.Text = str1
objStreamWriter.WriteLine(TextBox1.Text)
objStreamWriter.Close()
End Sub

'procedure for creating new text file
to write timer values
into it and read a file 1100 times
Private Sub Button4_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button4.Click
Dim str1 As String = "c:\Results\fr " &
```

```
Format(Now, "MMddyy_HHmmss") & ".txt"
objStreamWriter = New StreamWriter(str1, True)
TextBox8.Text = str1
Dim myStream As Stream = Nothing
Dim myReader As System.IO.StreamReader
Dim I As Integer
Dim startCount As Int64 = PerfCount.
QueryPerformanceCounter()

'select text file to be read
OpenFileDialog1.InitialDirectory = "c:\"
OpenFileDialog1.Filter = "txt files (*.txt)|*.txt"
OpenFileDialog1.FilterIndex = 2
OpenFileDialog1.RestoreDirectory = True
If OpenFileDialog1.ShowDialog()=
System.Windows.Forms.DialogResult.OK Then
Try
TextBox7.Text = OpenFileDialog1.FileName

'open the selected file for reading and read
myStream = OpenFileDialog1.OpenFile()
For I = 1 To 1100
startCount = PerfCount.QueryPerformanceCounter()
objStreamWriter.WriteLine(startCount)
myReader = New StreamReader(TextBox7.Text)
```

```
If Not myStream Is Nothing Then


'write timer values into textbox in form
TextBox10.Text += myReader.ReadToEnd & vbCrLf
End If
Next I
Catch Ex As Exception
MessageBox.Show("Cannot read file from disk.
Original error: " & Ex.Message)
Finally
If Not myReader Is Nothing Then
myReader.Close()
End If


'Check this again, since we need to make sure
we didn't throw an exception on open.
If Not myStream Is Nothing Then
myStream.Close()
End If
End Try
End If
objStreamWriter.Close()
End Sub
```

# Appendix B

## VB.NET code for videocapture

```
Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs)Handles MyBase.Load
LoadDeviceList()
If lstDevices.Items.Count > 0 Then
btnStart.Enabled = True
lstDevices.SelectedIndex = 0
btnStart.Enabled = Then
Else
lstDevices.Items.Add("No Capture Device")
btnStart.Enabled = False
End If
btnStop.Enabled = False
picCapture.SizeMode = PictureBoxSizeMode.
StretchImage
End Sub
Private Sub LoadDeviceList()
Dim strName As String = Space(100)
Dim strVer As String = Space(100)
Dim bReturn As Boolean
Dim x As Integer = 0
```

```
' Load name of all avialable devices
into the lstDevices
Do
' Get Driver name and version
bReturn=capGetDriverDescriptionA
(x, strName,100,strVer, 100)

' If there was a device add device name to the list
If bReturn Then lstDevices.Items.Add(strName.Trim)
x += 1
Loop Until bReturn = False
End Sub

Private Sub OpenPreviewWindow()
Dim iHeight As Integer = picCapture.Height
Dim iWidth As Integer = picCapture.Width

' Open Preview window in picturebox
hHwnd = capCreateCaptureWindowA
(iDevice, WS_VISIBLE Or WS_CHILD, 0, 0,
640, _480, picCapture.Handle.ToInt32, 0)

' Connect to device
If SendMessage(hHwnd, WM_CAP_DRIVER_CONNECT,
```

```
iDevice, 0) Then

'Set the preview scale
SendMessage(hHwnd, WM_CAP_SET_SCALE, True, 0)

'Set the preview rate in milliseconds
SendMessage(hHwnd, WM_CAP_SET_PREVIEWRATE, 100, 0)

'Start previewing the image from the camera
SendMessage(hHwnd, WM_CAP_SET_PREVIEW, True, 0)

' Resize window to fit in picturebox
SetWindowPos(hHwnd, HWND_BOTTOM,
0, 0, picCapture.Width,
piCapture.Height,_SWP_NOMOVE Or SWP_NOZORDER)
SendMessage2(New IntPtr(hHwnd),
WM_CAP_SET_CALLBACK_FRAME,
New IntPtr(0), delCallBack)
btnStop.Enabled = True
btnStart.Enabled = False
Else

' Error connecting to device close window
DestroyWindow(hHwnd)
End If
```

```
End Sub


'procedure to open display window
Private Sub btnStart_Click
(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnStart.Click
iDevice = lstDevices.SelectedIndex
OpenPreviewWindow()
End Sub


'procedure to close display window
Private Sub ClosePreviewWindow()

' Disconnect from device
SendMessage(hHwnd, WM_CAP_DRIVER_DISCONNECT,
iDevice, 0)

' close window
DestroyWindow(hHwnd)
End Sub


'procedure to enable user to stop display
Private Sub btnStop_Click(ByVal sender
As System.Object, ByVal e As System.EventArgs)
Handles btnStop.Click
```

```
ClosePreviewWindow()
btnStart.Enabled = True
btnStop.Enabled = False
End Sub

Private Sub Form1_Closing(ByVal sender As Object,
ByVal e As System.ComponentModel.CancelEventArgs)
Handles MyBase.Closing
If btnStop.Enabled Then
ClosePreviewWindow()
End If
End Sub

'procedure to create new text file
for writing timer values
Private Sub Button6_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button6.Click
Dim str As String = "c:\Results\wc " &
Format(Now, "MMddyy_HHmmss") & ".txt"
objStreamWriter = New StreamWriter(str, True)
TextBox9.Text = str
objStreamWriter.Close()
End Sub

'function called every time when a frame is
```

```
captured by the driver
Public Function FrameCallbackTarget
(ByVal hwnd As IntPtr,
ByRef lpVHdr As VIDEOHDR) As IntPtr
Dim startcount = PerfCount.QueryPerformanceCounter()
Try
objStreamWriter = New StreamWriter(TextBox9.Text, True)
objStreamWriter.WriteLine(startcount)
TextBox11.Text += CStr(startcount) & vbCrLf
objStreamWriter.Close()
Catch Ex As Exception
MessageBox.Show(Ex.Message)
End Try
End Function

'procedure to start reading text file
Private Sub Button7_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button7.Click
Dim myStream As Stream = Nothing
Dim ofile As System.IO.File
buttonstop.Enabled = False
Dim myReader As System.IO.StreamReader
Try
myReader = ofile.OpenText("c:\TestFolder\Testtext.txt")
TextBox12.Text += myReader.ReadToEnd & vbCrLf
```

```
Catch Ex As Exception
MessageBox.Show("Cannot read file from disk.
Original error: " & Ex.Message)
Finally
If Not myReader Is Nothing Then
myReader.Close()
End If

' Check this again, since we need to make sure we
didn't throw an exception on open.
If Not myStream Is Nothing Then
myStream.Close()
End If
End Try
End Sub

Private Sub buttonstop_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs)Handles buttonstop.Click
buttonstop.Enabled = True
End Sub
```