Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-16-2004

Using Synchronised FireWire Cameras For Multiple Viewpoint Digital Video Capture

J. U and D. Suter

MONASH
UNIVERSITY

Using Synchronised FireWire Cameras For Multiple Viewpoint Digital Video Capture

James U
Electrical and Computer Systems Engineering
Monash University, Clayton
Email: james.u@eng.monash.edu.au

Assoc. Professor David Suter
Electrical and Computer Systems Engineering
Monash University, Clayton
Email: d.suter@eng.monash.edu.au

November 10, 2004

Contents

Chapter 1

Introduction

We are interested in developing a system to perform multiple viewpoint digital video capture. The scale of the system will be such that we are able to capture video of a human subject performing dynamic actions such as martial arts and dance movements. Applications of this system include three dimensional video reconstruction of the subject's actions for movement modelling, motion analysis, or simply entertainment purposes.

We propose a scalable system involving multiple synchronised cameras (at least six) capturing uncompressed video that is written to disk by multiple controlling host PCs. For this system we have chosen to use FireWire digital video cameras and standard PCs. This report details our investigation of the hardware and software needed to realise the synchronised multiple viewpoint video capture that we propose. Rai *et al.* describe the implementation of a similar system in [7], although they remain vague on the degree of synchronisation that they were able to achieve.

We begin in Chapter 2 by justifying our choice of FireWire digital video cameras for the system, including explanations of the relevant features and capabilities of FireWire. Next in Chapter 3 we provide a survey of the cameras that we tested and plan to use for our system. Chapter 4 follows with a study of the hardware issues that need to be addressed in order to ensure that the streamed video is written to disk without dropping any frames. Finally Chapters 5–7 detail the various inter-camera synchronisation methods that we investigated. We conclude with a summary of the proposed possible implementations of the synchronised multiple camera video capture system.

Chapter 2

Why FireWire?

IEEE-1394, or ‘FireWire’ as it is more commonly known, is a high speed serial bus implementation first developed by Apple Computers in the mid 1980s (the two terms will be used interchangeably throughout this report). It is a standardised architecture which allows for the interconnection of a wide range of devices, ranging from PCs to storage devices to digital video cameras. It was designed with many special features, including support for isochronous transfers which makes it ideally suited to digital video capture.

In this chapter we will discuss some key advantages of FireWire and justify its choice for multiple viewpoint digital video capture. For a comprehensive guide to the IEEE-1394 specification the reader is referred to [1]. Also of interest may be [3] which presents a case for FireWire as today’s technology of choice in machine vision applications.

2.1 Key Features

2.1.1 Speed

FireWire is a scalable high speed serial bus implementation. To clarify, the specific implementation of the FireWire/IEEE1394 standard we refer to in this report is *IEEE1394a*. This is the version currently employed by most consumer level FireWire devices, including digital video cameras. It is also known as ‘FireWire 400’ since the bus’ bandwidth is 400 Mbit/s (393.216 Mbit/s to be precise). FireWire 400 is backwards compatible with older implementations of FireWire running at 200 Mbit/s and 100 Mbit/s although the bandwidth of the bus is limited to that of the lowest speed implementation when they are mixed.

There exists a newer implementation of FireWire known as *IEEE1394b* or ‘FireWire 800’ which doubles the bandwidth to 800 Mbit/s.[1] It too is ‘bandwidth-limited’ backwards compatible. FireWire 800 is not yet widely supported at the consumer level — at present only storage devices are available and it is not clear as to when, if ever, digital video cameras will be made to this specification.

The high speed of the FireWire bus readily facilitates the streaming of uncompressed digital video. At 400 Mbit/s, a single FireWire bus is able to handle uncompressed digital

[1]Future developments in FireWire technology will see the bandwidth increasing further to 1600 Mbit/s and beyond to 3200 Mbit/s.

video capture from 2 cameras running at 1024x768 and 15 fps or from 3 cameras running at 640x480 and 30 fps (Section 2.3).

2.1.2 Plug and Play

The FireWire serial bus is designed for plug and play and all devices designed for it support automatic configuration. With FireWire, any device that is attached to the bus automatically participates in the configuration process — no manual intervention from the host system is necessary. Furthermore, each time a device is added or removed, the bus automatically reconfigures itself accordingly. These plug and play capabilities greatly enhance FireWire's ease of use.

2.1.3 Scalability

FireWire is a highly scalable technology. It allows for the attachment and support of a wide variety of peripheral devices to a given host system. Typically the IEEE-1394 serial bus is attached to the PCI bus of the host system via a 1394-to-PCI Open Host Controller Interface (or OHCI) compliant interface card. This PCI card acts as the 'root node' of the serial bus to which 'branch' or 'leaf' nodes (i.e. FireWire devices) are attached. These nodes may include storage and imaging devices, hubs and repeaters (which may be daisy-chained together to increase the amount of physically attachable nodes), or even another host system (when using FireWire as a networking technology).

A single IEEE-1394 serial bus implementation supports up to 64 node addresses. Of these, 63 may be physical nodes with the final one reserved as a broadcast address which all nodes recognise and may use to broadcast configuration messages and other information. Although theoretically this suggests the ability to attach 63 digital video cameras to a single host system for video streaming, in practice, due to FireWire bandwidth and channel restrictions (Section 2.1.4) and the underlying PCI bus and processing capabilities of the host system (Chapter 4), only 2–4 such cameras may realistically be attached to any one FireWire bus. However, considering the multiple camera system we propose, and the fact that a PC may support up to 3 FireWire buses (Section 4.2), this is degree of scalability provided by FireWire is clearly preferable to having one dedicated PC for each camera.

2.1.4 Asynchronous and Isochronous Transfer

The FireWire serial bus supports two distinct data transfer protocols — 'asynchronous' and 'isochronous'. The nature of the node application dictates which of these two protocols is to be used. The 'asynchronous' transfer mode is used for applications which require data delivery of guaranteed integrity over a non-specified period of time. An example of this is data storage where the integrity of the data must be ensured but the time in which the transfer is completed is not of critical importance. To facilitate this, the asynchronous transfer protocol employs data and acknowledgement packets, error checking and possible retransmission packets between the 'requester' (which is sending the data) and the 'responder' (which is receiving the data).

It is, however, the 'isochronous' data transfer capability of FireWire which is pertinent to video capture. Isochronous applications may be identified as those in which the *rate* of

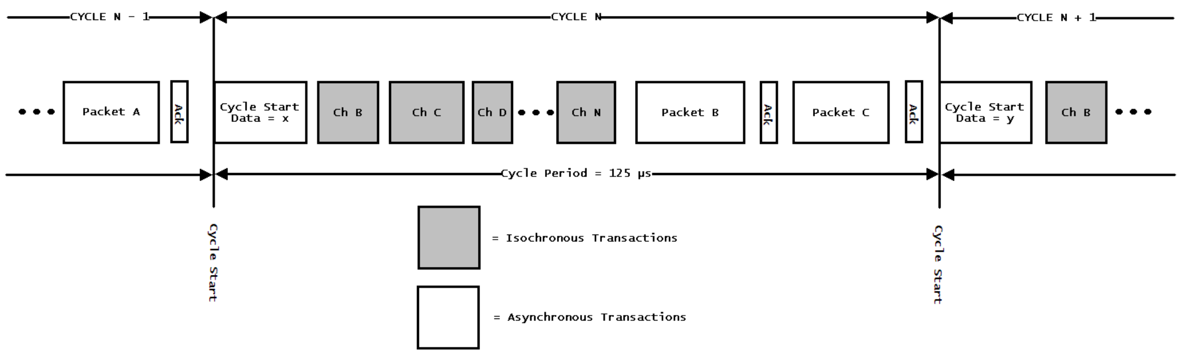


Figure 2.1: Timing of multiple isochronous and asynchronous data transfers. *Source:* [1], p. 155 (adapted)

data transfer rather than its integrity is of critical importance. As an example, consider a camera which captures 1024x768 images at a constant rate of 15 fps and streams the data across the IEEE-1394 bus to memory buffers on the host system. In this case we are not concerned if a few bytes of a particular image are corrupted. We will certainly not compromise the frame rate by requesting a retransmission of the corrupted bytes. Nor do we require explicit acknowledgement that the data has been received. Here the matter of critical importance is the guaranteed rate of transfer of the images across the bus. FireWire facilitates this requirement with the isochronous data transfer protocol which sends the data from the requester to the responder fragmented into packets at regular 125 μs intervals. The size of the payload data of these packets is determined such that the complete piece of data (say, one frame) is guaranteed to be transferred across the IEEE-1394 bus within the desired time interval (say, the requested frame rate).

FireWire is designed to handle *multiple* isochronous *and/or* asynchronous transaction requests by sharing its 400 Mbit/s bandwidth between the various requests. This allows, for example, multiple cameras attached to a single FireWire bus, to stream video concurrently. It is achieved by dividing the 125 μs periodic intervals into sub-intervals in which the entire bandwidth of the bus is owned by a particular isochronous/asynchronous transaction.

This bus sharing scheme is illustrated in Figure 2.1. The figure shows how each 125 μs cycle is divided into periodic 'channels' for the requested isochronous transactions, and packets/acknowledgements for the requested asynchronous transactions. The 'cycle start data' present at the beginning of each cycle period initialises the bus to handle these requests accordingly for the given period.[2]

> As an example isochronous transaction, consider an imaginary image 8000000 bytes in size that must be streamed from a camera at 1 frame per second. To ensure that this occurs, the FireWire bus allocates a channel in each 125 μs cycle period with a data payload size of 1000 bytes. Thus, 1000 bytes is guaranteed to be transferred every 125 μs during the slice of the cycle period owned by the allocated channel. Of course, `1000 bytes/125` μ`s = 80000000 bytes/s` as required (the

[2]Note that the FireWire 400 specification stipulates that the maximum data payload size per 125 μs cycle period is 4096 bytes for isochronous channels and 2048 bytes for asynchronous packets (data is fragmented into packets limited by these sizes if they are exceeded).

actual figures for isochronous transactions involving digital video cameras, along with real examples, are given in Section 2.3).

Isochronous and/or asynchronous transaction requests are handled by a root node (determined during the configuration process) which is capable of being the ‘bus manager’. This bus manager provides the necessary timing information and allocates the bus’ resources by arbitrating over the various requests. A complete explanation of the arbitration process is given in [1]. We will only note here that the IEEE-1394 specification states that a *maximum* of 80% of the bandwidth of the bus may be allocated to isochronous transactions with a *minimum* of 20% necessarily allocated to asynchronous transactions. This means that digital video cameras using isochronous transfer to stream video over a FireWire bus are limited to a bandwidth of 40 MB/s (320 Mbit/s) rather than the nominal 50 MB/s (400 Mbit/s).

2.2 Digital Camera Specification

Before the advent of FireWire digital video cameras, machine vision was largely the domain of analog video cameras and analog-to-digital frame-grabbers. Both the cameras and their respective frame-grabbers were invariably proprietary technologies, each with their own connection cables, command sets and control libraries. Without a standard to interface the cameras/frame-grabbers to host systems the user was burdened with having to learn how to operate each particular device as well as being restricted to a particular host operating system and unable to mix technologies.

Fortunately, along with the development of FireWire, the 1394 Trade Association introduced the ‘1394-based Digital Camera Specification’ (or DCAM spec for short). The DCAM spec defines a standardised set of functions and capabilities for FireWire digital video cameras along with a set of register based controls to interface cameras to host systems. The specification covers camera aspects such as video format, frame rate and external triggering as well as filter, shutter, balance, gain and brightness controls. The specification also allows for the implementation of manufacturer specific advanced camera capabilities. Currently the DCAM spec is at version 1.31 and is available for download (for a fee) from `http://www.1394ta.org`.

Writing code to control DCAM spec compliant cameras is made easy with the appropriate software libraries. One such open source C/C++ implementation of the DCAM spec is the 1394-based digital camera control library `libdc1394` for Linux based operating systems (`http://sourceforge.net/projects/libdc1394/`). With little difficulty, we interfaced a mixture of FireWire cameras from different manufacturers with a Linux PC using `libdc1394` and a GUI written for it called `coriander` (`http://sourceforge.net/projects/coriander/`).

Although it is useful to be able to write software to control a heterogeneous mix of FireWire cameras, ultimately we will be using a homogeneous set of cameras for our project. Moreover, we will have need to control manufacturer specific advanced camera capabilities. For this reason it was decided to write all camera software using the proprietary camera control library supplied by the camera manufacturer itself (which implements *both* the DCAM spec and provides access to model-specific advances features).

2.3 Isochronous Transfers and Video Streaming

An example of how an imaginary isochronous transaction is handled by the FireWire bus was given in Section 2.1.4. We now turn our attention to real examples of how the isochronous transactions (used by digital video cameras to stream images at a constant frame rate) are handled by the FireWire bus.

The 1394-based Digital Camera Specification indicates isochronous bandwidth requirements for streaming video at standard resolutions and frame rates which DCAM spec compliant cameras must adhere to. Figure 2.2 shows these requirements for the 'Format_0' ('low' resolution) video modes. Recall from Section 2.1.4 that the data payload per cycle period for an isochronous transaction cannot exceed 4096 bytes or 1024 'quadlets' (note that the DCAM spec prefers 'quadlets' to bytes, where 1 quadlet = 4 bytes).

Consider Mode_5 — 640x480 Y8. At 30 fps, the requirement is 320 quadlets/packet or 1280 bytes/packet. Since `640x480` $\times$ `8 bits/pixel = 307200 bytes/frame` is being transferred across the bus at 1280 bytes/packet at a rate of 1 packet/125 μs, the total time taken to transfer one whole frame is `307200 bytes/frame` $\div$ `1280 bytes/packet` $\times$ `125` μ`s/packet = 30 ms`. Since 30 fps = 1 frame every 33.3 ms, the whole frame is transferred within the required time period.

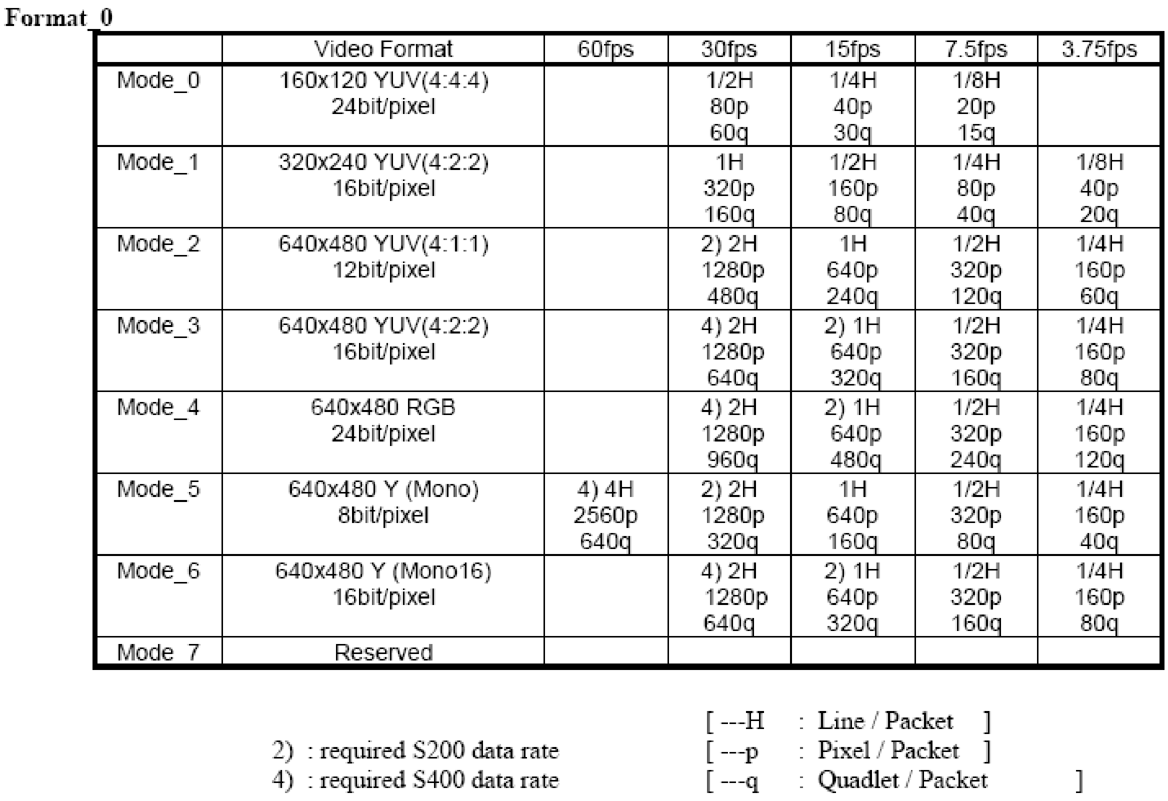
Clearly the DCAM spec's isochronous bandwidth requirements have been calculated to satisfy the video format/mode and frame rate requested. As another example, consider Mode_5 at 60 fps. Figure 2.2 indicates that 640 quadlets/packet or 2560 bytes/packet is required. This gives `307200 bytes/frame` $\div$ `2560 bytes/packet` $\times$ `125` μ`s/packet = 15 ms` to transfer on whole frame across the bus. Since 60 fps = 1 frame every 16.7 ms, the whole frame is again transferred within the required time period.

Notice in the first example the isochronous bandwidth required for 640x480 Y8 at 30 fps is 320 quadlets out of a maximum of 1024 quadlets per period. That is, `320` $\div$ `1024` $\times$ `100 = 31.25%` of the bus' bandwidth is used. This means that 3 such isochronous channels may exist simultaneously on the one FireWire bus, making it possible to stream uncompressed 640x480 Y8 images at 30 fps concurrently from 3 FireWire cameras. Note however at 60 fps, since 62.5% of the bus' bandwidth is used, only one such camera may be handled by a single FireWire bus.

Using the required bandwidth values specified by the DCAM spec, Table 2.1 shows the percentage of the $\approx$ 40 MB/s total isochronous bandwidth that is required by various standard video resolutions and frame rates. The maximum possible number of cameras attached to a single bus and streaming simultaneously may be inferred from this table.

2.4 FireWire Alternative — Camera Link

The combination of IEEE-1394a OHCI compliant PCI cards, DCAM spec compliant digital video cameras and standard FireWire cables has supplanted the older technology of analog video camera/frame-grabber and device specific cables for most machine vision applications. The FireWire solution provides the many advantages outlined in this chapter and is comparatively inexpensive (see [3] for a comparison of FireWire and analog video camera/frame-grabber technologies).



Format_0

	Video Format	60fps	30fps	15fps	7.5fps	3.75fps
Mode_0	160x120 YUV(4:4:4) 24bit/pixel		1/2H 80p 60q	1/4H 40p 30q	1/8H 20p 15q	
Mode_1	320x240 YUV(4:2:2) 16bit/pixel		1H 320p 160q	1/2H 160p 80q	1/4H 80p 40q	1/8H 40p 20q
Mode_2	640x480 YUV(4:1:1) 12bit/pixel		2) 2H 1280p 480q	1H 640p 240q	1/2H 320p 120q	1/4H 160p 60q
Mode_3	640x480 YUV(4:2:2) 16bit/pixel		4) 2H 1280p 640q	2) 1H 640p 320q	1/2H 320p 160q	1/4H 160p 80q
Mode_4	640x480 RGB 24bit/pixel		4) 2H 1280p 960q	2) 1H 640p 480q	1/2H 320p 240q	1/4H 160p 120q
Mode_5	640x480 Y (Mono) 8bit/pixel	4) 4H 2560p 640q	2) 2H 1280p 320q	1H 640p 160q	1/2H 320p 80q	1/4H 160p 40q
Mode_6	640x480 Y (Mono16) 16bit/pixel		4) 2H 1280p 640q	2) 1H 640p 320q	1/2H 320p 160q	1/4H 160p 80q
Mode_7	Reserved					

	[---H : Line / Packet]
2) : required S200 data rate	[---p : Pixel / Packet]
4) : required S400 data rate	[---q : Quadlet / Packet]

Figure 2.2: Isochronous bandwidth requirements for Format_0 images. *Source:* [5], pp. 69–70

Video Resolution and Frame Rate	Bandwidth Used
640x480 Y8 @ 30 fps	31.25 %
640x480 Y8 @ 60 fps	62.5 %
1024x768 Y8 @ 15 fps	37.5 %
1024x768 Y8 @ 30 fps	75.0 %
1280x960 Y8 @ 7.5 fps	31.25 %
1280x960 Y8 @ 15 fps	62.5 %

Table 2.1: Isochronous bandwidth required for standard video resolutions/frame rates.

Despite being a particularly suitable bus architecture, the IEEE-1394 serial bus was not designed specifically for machine vision applications. The usable bus bandwidth limitation of $\approx$ 40 MB/s (for IEEE-1394a) means that the absolute maximum video resolutions and frame rates achievable by FireWire cameras on the bus range from 640x480 at 60 fps to 1024x768 at 30 fps to 1280x960 and 1600x1200 at 15 fps. Higher frame rates may be achieved at the expense of resolution — the DCAM spec includes a user-definable video format which accepts a selectable 'area of interest', allowing for frame rates of up to 250+ fps when a smaller area of interest has been selected.[3]

FireWire 400's present $\approx$ 40 MB/s bandwidth limitation will be overcome with the newer implementations of FireWire 800, 1600, and 3200. However, digital cameras have yet to be designed for these versions of FireWire and it is not clear as to whether or not this will eventuate in the near future. But for machine vision applications which require video resolutions and frame rates in excess of those achievable by FireWire, there already exists an industry standard named Camera Link.

The specifications of the Camera Link interface standard for digital cameras and frame-grabbers is available on-line from `http://www.machinevisiononline.org`. In brief, Camera Link is a standard developed by a group of industrial camera and frame-grabber manufacturers which defines:

- A standard connector that will be used on both the camera and the frame-grabber and a standard cable to connect the two
- Formats for transmitting image data from the camera to the grabber
- Standard camera control inputs
- A standard method for transmitting serial communication data between the camera and the grabber
- A standard chip set that will be used in the camera and the grabber for image data transfer

Thus the Camera Link camera/frame-grabber solution parallels that of the FireWire one in terms of features with the exception of a couple of key differences:

- Camera Link is a point-to-point implementation as opposed to a bus one such as FireWire — rather than allowing multiple cameras to share one bus, it requires one frame-grabber for each camera used.
- Camera Link uses a chip set that employs low voltage differential signalling (LVDS) technology which enables 2.38 Gbit/s data transfer rates between camera and frame-grabber.

[3]It is important to note though, that the maximum possible integration (or shutter) time of the camera is subject to constraint by the frame rate. For example, at 100 fps an image is taken every 10 ms, so the camera is allowed, at most, only 10 ms to integrate the light energy from the scene (i.e. the scene being imaged needs to be *really* well lit).

At 2.38 Gbit/s Camera Link cameras can capture, for example, 1280x1024 images at *500* fps — well beyond the upper limits of FireWire's capabilities. The trade-off, not surprisingly, is cost of implementation (see `http://www.opsci.com` for an extensive list of Camera Link products and prices). At present, base prices for Camera Link cameras are typically 2–3 times more than that of FireWire ones, plus there is the cost of the frame-grabbers and cables required which are around 4–5 times that of their FireWire equivalents (not to mention needing a dedicated frame-grabber for *each* camera). Furthermore, the standard PCI-32/33 (133 MB/s) buses which FireWire connects to cannot handle the 250+ MB/s bandwidth that Camera Link outputs, making it necessary to use host PCs with PCI-64/66 (533 MB/s) buses instead, again adding further to the cost of implementation. And then finally, even with the ability to handle this bandwidth for video capture purposes, the host PCs will still need to be able to write the data out to disk somehow (Chapter 4). At these transfer rates, only a large striped RAID array of high-speed hard drives will be able to manage. Overall, it makes for a prohibitively expensive solution, especially when looking to implement a multiple camera system.

As a final note, we have decided that video at 1024x768 and 30 or even 15 fps, which the FireWire implementation is capable of, is quite sufficient for our needs of capturing human movement. The Camera Link alternative is clearly geared towards much higher speed/resolution machine vision applications where cost of implementation is not an issue.

Chapter 3

Cameras Tested

We tested a few different FireWire cameras for the multiple view video capture project (all of which comply with the 1394-based Digital Camera Specification version 1.31) — the Point Grey Scorpion (models SCOR-13SM and SCOR-13FF) and the Marlin F-131C from Allied Vision Technologies. To this list we add the Point Grey Flea which, subject to testing when available to us, we ultimately intend to use for the project. We now detail the key features and capabilities of these cameras.

3.1 Point Grey Scorpion

The Point Grey Scorpion is available in a range of models which are distinguished primarily by the imaging sensors used. The range includes cameras using CMOS and CCD imaging sensors which operate from 0.3 mega-pixels to 2.0 mega-pixels and are available in both mono and colour versions.

3.1.1 SCOR-13SM

The SCOR-13SM is a 1.3 mega-pixel model of the Scorpion which uses a Symagery 2/3" *CMOS* sensor (model VCA1281). We tested a set of 4 such cameras, 2 colour and 2 mono. The SCOR-13SM supports the DCAM spec standard video formats/modes and frame rates

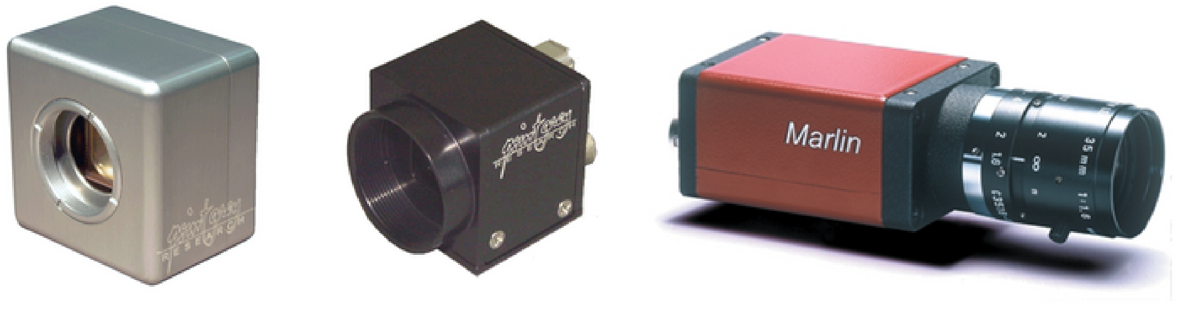


Figure 3.1: (Left to right) Point Grey Scorpion and Flea, AVT Marlin.

Format	**Mode**	**Mode Description**	**Frame Rate**						
			1.875	**3.75**	**7.5**	**15**	**30**	**60**	**120**
0	5	640x480 Y8		●	●	●	●	●	
1	2	800x600 Y8			●	●	●		
1	5	1024x768 Y8	●	●	●	●	●		
2	2	1280x960 Y8	●	●	●	●			

Table 3.1: SCOR-13SM supported video formats/modes and frame rates. *Source:* [6], p. 19

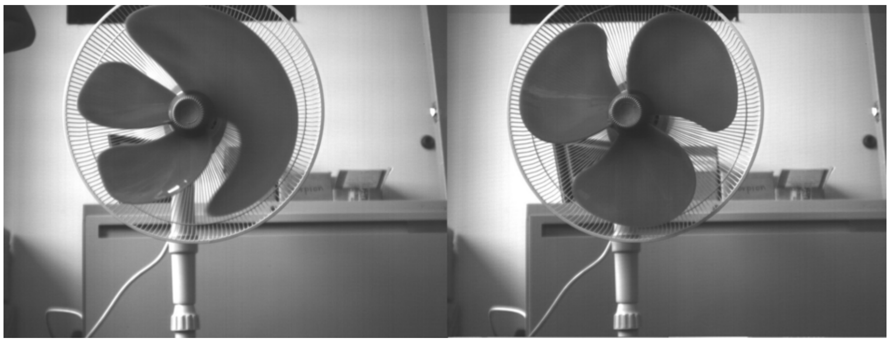


Figure 3.2: Left: Motion blur exhibited by CMOS rolling shutter. Right: Same scene imaged by CCD global shutter. *Source:* Point Grey Knowledge Base Article 115.

listed in Table 3.1. It also implements the DCAM spec Format_7 custom video format/mode[1], enabling up to 1280x1024 video capture at 15 fps. The SCOR-13SM is also capable of 640x480 at 100 fps and 320x240 at 275 fps using Format_7.

As mentioned, the SCOR-13SM uses a CMOS sensor for imaging. Almost all CMOS image sensors use a 'rolling shutter' (an exception is the SCOR-13FF's sensor — Section 3.1.2) wherein all the photodiodes (i.e. pixels) *do not* start collecting light at the same time. Instead, rows of pixels begin their light integration at the same time, but these integration periods are offset for each row, beginning at the top. Although this does not pose a problem for still images, image distortion may be created with moving objects since the upper image parts are scanned earlier than the lower ones as the 'integration by row' sweeps down the image. The resultant motion blur effect is shown in Figure 3.2. Note that this distortion does not occur with CCD sensors since they use 'global shutters' where *all* pixels start collecting light at the same time.

We note however that when the integration time is small enough, tests taken of a subject performing reasonably fast movements did not exhibit any clear evidence of such image distortion. We suspect that the motion blur seen in Figure 3.2 only becomes an issue with very high speed moving objects such as the blades of a fan.

SCOR-13SMs, indeed all Scorpion cameras, provide a set of general-purpose I/O pins accessible via an external interface. Through the I/O pins the camera can be configured to trigger the start of image integration on an external electrical signal or produce a similar signal that allows devices external to the camera (say, a flash) to be synchronised to (and

[1]With Format_7 specific regions of interest may be selected, allowing for higher frame rates by reducing the amount of data being sent along the IEEE-1394 bus per frame.

possibly offset from) its start of integration. This external triggering ability is explored as an option for synchronising multiple cameras in Chapter 5.

A final very useful feature implemented in all Point Grey cameras is their hardware-level automatic synchronisation. Same model cameras operating on the same FireWire bus and capturing at the same frame rate will automatically synchronise their image acquisition to within 125 μs of one another. This automatic intra-bus camera synchronisation is detailed in Chapter 6.

Unfortunately, the SCOR-13SM cameras *do not* work with the Point Grey Sync Unit (Section 7.1), ruling out the latter's use as a means of synchronising multiple such cameras across different buses.

3.1.2 SCOR-13FF

Like the SCOR-13SM, the SCOR-13FF is a 1.3 mega-pixel model of the Scorpion. It is a newer model which is intended to replace the older SCOR-13SM. The sensor now used is a FloodFill 2/3" CMOS sensor (model IBIS5A). Unlike the Symagery sensor of the SCOR-13SM, the FloodFill sensor uses a *global* shutter rather than a rolling shutter, avoiding the possibility of rolling shutter image distortion as described above.

In terms of features and capabilities the SCOR-13FF is otherwise almost identical to the SCOR-13SM, again supporting up to 1280x1024 Y8 capture at 15 fps in custom image mode as well as the latter's standard formats/modes and frame rates. However the key advantage in choosing this model over the older SCOR-13SM (other than the global shutter functionality) is that this newer model Scorpion *does* work with the Point Grey Sync Unit.

We tested one colour version of the SCOR-13FF. Point Grey advised that there have been some problems with the FloodFill IBIS5A imaging sensor in terms of 'burned' pixels, resulting in fixed noise patterns in the images. We found this to indeed be the case (Figure 3.3). Given the degree of noise we observed, coupled with problems obtaining colour images at certain resolutions, it was decided that the benefits offered by this newer Scorpion model do not justify it's use. With neither model of Scorpion completely satisfying our requirements, we investigated the Point Grey 'Flea' range of cameras.

3.2 Point Grey Flea

The Point Grey Flea cameras are almost identical, in terms of features and capabilities, to the Scorpions. Of particular interest, the Fleas also (a) automatically synchronise their image acquisition to one another when connected on the same bus and, (b) support external triggering. And, like the SCOR-13FF, they are able to be synchronised across different buses using the Point Grey Sync Unit.

The key difference with the Fleas is that they use smaller imaging sensors, resulting in lower maximum resolutions. The Flea uses Sony 1/3" (global shutter) *CCD* imaging sensors, models ICX424 and ICX204, whose maximum resolutions and frame rates are 640x480 at 60 fps and 1024x768 at 30 fps respectively (as opposed to the Scorpions we tested, which are capable of 1280x1024 at 15 fps).

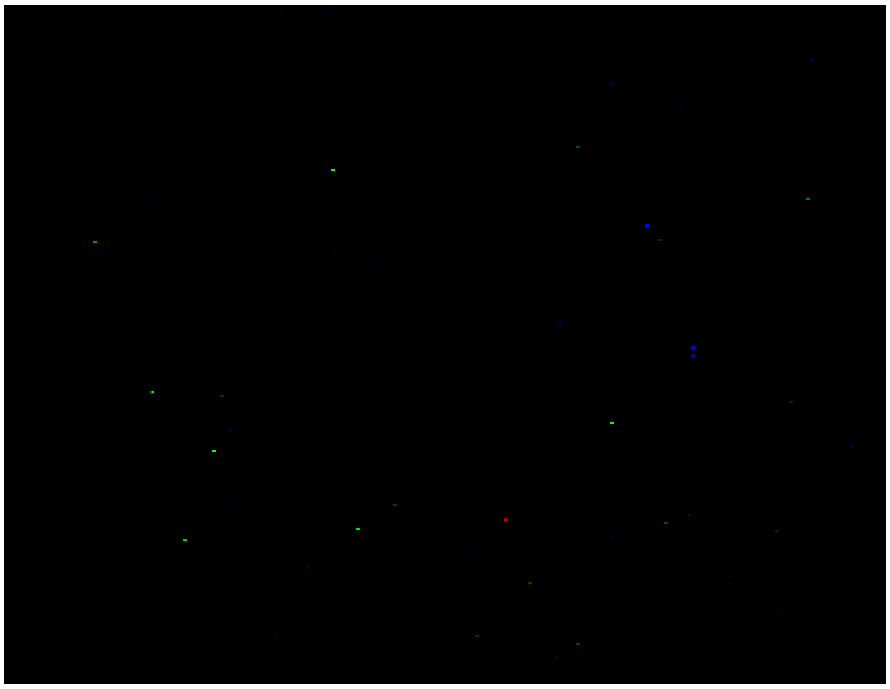


Figure 3.3: Example of burned pixels on dark image exhibited by IBIS5A sensor. *Source:* Point Grey Technical Application Note 2004005.

As shown in Table 2.1, bandwidth limitations mean that 640x480 at 30 fps, 1024x768 at 15 fps and 1280x960 at 7.5 fps are the maximum possible standard video resolutions and frame rates when looking to use multiple cameras on a single FireWire bus. Considering that 7.5 fps is possibly too low a frame rate for capturing human movement, it was decided that the Fleas lack of support for 1280x960 resolution was of little importance since they otherwise presented us with all the advantages of the SCOR-13SM and SCOR-13FF *without* the negative drawbacks of either.

Thus, subject to testing when the cameras are made available to us, we intend to use the Point Grey Flea (1024x768 high-resolution model) in our multiple camera system. Interestingly this model of the Flea costs the same as the higher resolution Scorpions, which suggests that one is likely paying the equivalent amount for the more expensive/higher quality CCD sensor at a lower resolution.

3.3 Allied Vision Technologies Marlin F-131C

The Marlin F-131C from Allied Vision Technologies was tested as an alternative to the Point Grey range of FireWire cameras. Like the SCOR-13FF, this camera uses the FloodFill 2/3" global shutter CMOS sensor (model IBIS5A). Interestingly, unlike the SCOR-13FF however, we observed no signs of the fixed pattern noise due to burned pixels that affected the SCOR-13FF.

The Marlin provides a host of features and capabilities not found in the Point Grey cameras. Whilst the camera is capable of all the standard video formats/modes and frame rates provided by the Scorpions, the Marlin further adds to this list the optional output of video data in (colour) YUV 4:2:2, and YUV 4:1:1 modes as well as standard (mono) Y8. Also, in Format_7 mode at the maximum resolution of 1280x1024, the camera is able to capture at 25 fps.

The Marlin includes a range of on-board functions including:

- Real-time shading and colour correction
- Real-time bayer demosaicing and RGB to YUV conversion
- User programmable look up table

These advanced features come at a cost though — the Marlin is approximately twice the price of the Point Grey cameras. It was decided that we do not require these extra (real-time) features since we wish to post-process the video capture data in any case. The AVT Marlin is left as an alternative for applications where immediate colour video output is required.

Chapter 4

Streaming Issues

4.1 Storage Capacity

Writing the video streamed from the cameras to disk presents a few issues that must be addressed. The most obvious issue which comes to mind is storage capacity. Consider for example, streaming to disk uncompressed video from 2 cameras capturing 8 bits/pixel images at a resolution of 1024x768 and a frame rate of 15 fps. This produces about 23.6 MB/s of data or 1.4 GB/min. Fortunately, such large amounts of data are no longer a major concern as today's hard drive capacities readily meet these demands.

4.2 Bandwidth and Throughput Limitations

Of greater concern than storage capacity is the actual ability of the host system to *handle* these necessarily high levels of data throughput. Figure 4.1 illustrates the flow of data from the cameras to disk. The cameras are connected to the host system's PCI bus via a 1394-to-PCI controller. As noted earlier, the maximum bandwidth of the 'effective' FireWire bus (consisting of the cameras and the controller card) is $\approx$ 40 MB/s. The PCI bus to which this FireWire bus is attached is a 32-bit PCI-32/33 bus with a maximum bandwidth of 133 MB/s. Considering these bus bandwidths, 3 FireWire buses may (theoretically) be handled by the one PCI bus, providing for up to 6–9 cameras per host PC depending on the capture resolution and frame rate.[1]

4.3 Writing to Disk

The ability to handle video throughput up to the maximum bandwidth of the host system's PCI bus (be it 32-bit or 64-bit) is nevertheless of no use to us if we are unable to write the video to disk in a timely manner — that is to say, without having to drop any frames. Figure

[1]A (relatively expensive) higher bandwidth alternative to the standard 32-bit PCI-32/33 buses are the newer 64-bit PCI-64/66 buses which increase the maximum bandwidth to 533 MB/s. These 64-bit PCI buses interface with the newer FireWire 800 cards. However, if used to connect FireWire cameras, it must noted that half the bandwidth of these FireWire 800 buses will be wasted as the speed drops from 800 Mbit/s to the 400 Mbit/s operating speed of the cameras (Section 2.1.1). A system with a PCI-64/66 bus might be useful if the user wished to implement a bank of FireWire 800 cards to attach a large amount of cameras to the host system, since the 64-bit PCI bus would be able to handle the combined throughput of these FireWire buses.

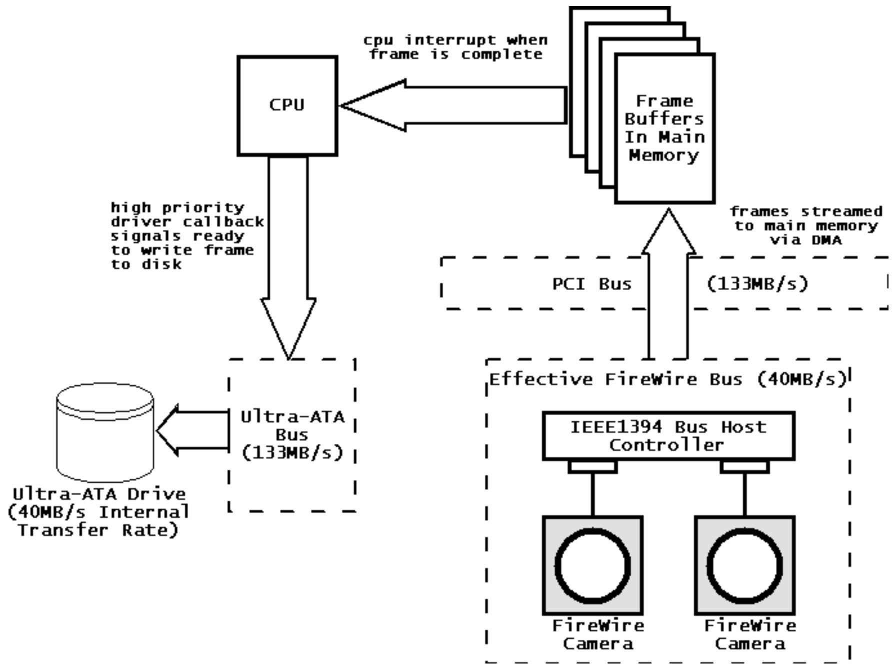

cpu interrupt when frame is complete
CPU
Frame Buffers In Main Memory
high priority driver callback signals ready to write frame to disk
frames streamed to main memory via DMA
PCI Bus
(133MB/s)
Effective FireWire Bus (40MB/s)
IEEE1394 Bus Host Controller
Ultra-ATA Bus (133MB/s)
Ultra-ATA Drive (40MB/s Internal Transfer Rate)
FireWire Camera
FireWire Camera

Figure 4.1: Flow of video data from camera to disk.

Model	Interface	Rate (MB/s)
Seagate Barracuda 7200 rpm	Serial ATA-150	32–58
Western Digital Raptor 10000 rpm	Serial ATA-150	46–72
Seagate Cheetah 15000 rpm	Ultra-320 SCSI	49–75

Table 4.1: Select hard drives and their average–maximum sustained transfer rates.

4.1 shows how the cameras stream the captured frames via direct memory access (DMA) over the PCI bus to buffers in main memory. These frame buffers are overwritten as newer images arrive from the cameras and there is an upper limit of 128 frame buffers per camera. This means that if we wished to allocate buffers for each frame to be captured and write the series of images out to disk afterwards, we would be limited to image sequences of a few seconds at most, even with gigabytes of available main memory.

Returning to Figure 4.1, we see that a CPU interrupt is generated whenever a whole frame has been transferred from a camera to a frame buffer. Next a high priority driver callback signals to the user-space camera capture process that the frame in memory is complete and ready to be written out to disk. If no other operations on the frame are required, the process can immediately write the raw image data out to disk. The hard drive controller is connected directly to the southbridge on most modern motherboards, so the full bandwidth of the hard drive bus can be used for this writeout task.

The common Ultra-ATA bus depicted has a maximum bandwidth of 133 MB/s, matching that of the PCI-32/33 bus. But it is important to note that this value is the maximum bandwidth of the hard drive *interface* provided by the bus — the actual rate at which data may be written to disk is determined by the *sustained transfer rate* of the hard drive itself. Table 4.1 shows typical sustained transfer rates for some common hard drives. The drives listed are actually higher-end Serial ATA and SCSI drives, but the average Ultra-ATA drive performs comparably, generally being capable of a 40 MB/s sustained transfer rate — which incidentally matches the maximum bandwidth of a single FireWire bus. That is to say, subject to latencies incurred by the host PC's operating system and the user-space capture process, one decent quality hard drive should be capable of writing to disk, in a timely manner, the video data streaming across a given FireWire bus operating at maximum throughput.

To summarise, it is possible, using a PC with a single decent quality hard drive, to stream up to the maximum 40 MB/s of video across a single FireWire bus directly to disk without dropping any frames. Indeed, we have achieved such results, streaming uncompressed 1024x768 images at 15 fps simultaneously from 2 cameras and 640x480 images at 30 fps simultaneously from 3 cameras for sustained periods without dropping any frames.

As mentioned earlier, it is also possible to have up to 3 FireWire buses attached to the one standard PCI-32/33 bus streaming concurrently. However, in order to write the video to disk at such high throughput levels *without dropping frames* we require a striped RAID volume consisting of drives whose combined sustained transfer rate at least matches that the PCI bus' bandwidth.

Chapter 5

Synchronisation Using an External Trigger

For our multiple viewpoint video capture we require that corresponding images captured by each camera be accurately synchronised to one another. The highest frame rate at which we intend to capture images is 30 fps, i.e. an image every $33.\dot{3}$ ms. Given this time interval, we will consider images from different cameras to be out of sync if they are captured at instances more than 2 ms apart. This allows for at most 6 % error for frame rates up to 30 fps.

Determining quantitatively the level of synchronisation between corresponding images taken from different cameras with millisecond resolution requires very accurate timing information that may not be available for certain methods of synchronisation. For this reason we have also chosen to *qualitatively* estimate the level of synchronisation of each method by visually comparing the corresponding images — such a technique does not rely on the availability of timing information.

> Our standard qualitative test — which we shall present for all synchronisation methods, including the current external triggering — involves our two colour Scorpion (SCOR-13SM) cameras being used to capture video of the same scene at a resolution of 640x480 and a frame rate of 30 fps. The specific location of the cameras (in terms of the FireWire bus/host PC to which they are attached) will be determined by the synchronisation method in question.
>
> The actual scene we have chosen to capture is a simple setup of a marker pen being waved in front of a ruler. This pen-waving experiment will facilitate the visual comparision of corresponding images as the marker pen's position can be determined directly in reference to the ruler's markings. For the purposes of this experiment we will assume that the hand waving the pen is moving at a rate which would sweep out a 1 m arc in 0.5 s; that is, the pen is moving at *2 mm/ms*. It is important to note that this is purely an assumption based on a reasonable estimate of the speed of the moving hand which we have made for the purposes of visually estimating the degree of synchronisation as reported by the test.

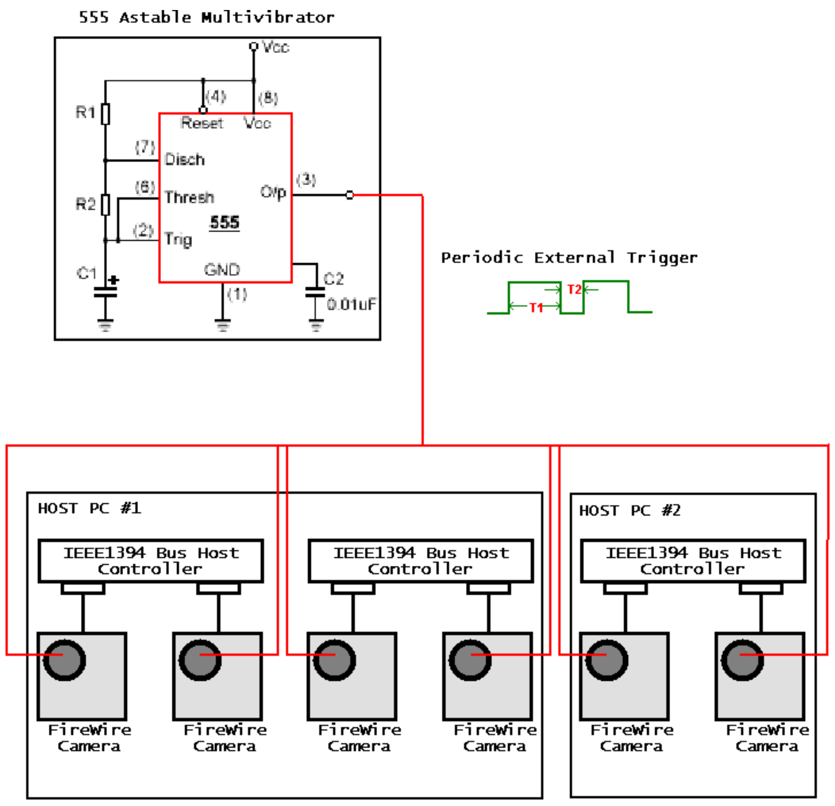


Figure 5.1: Externally triggering 6 cameras located on 3 FireWire buses across 2 PCs.

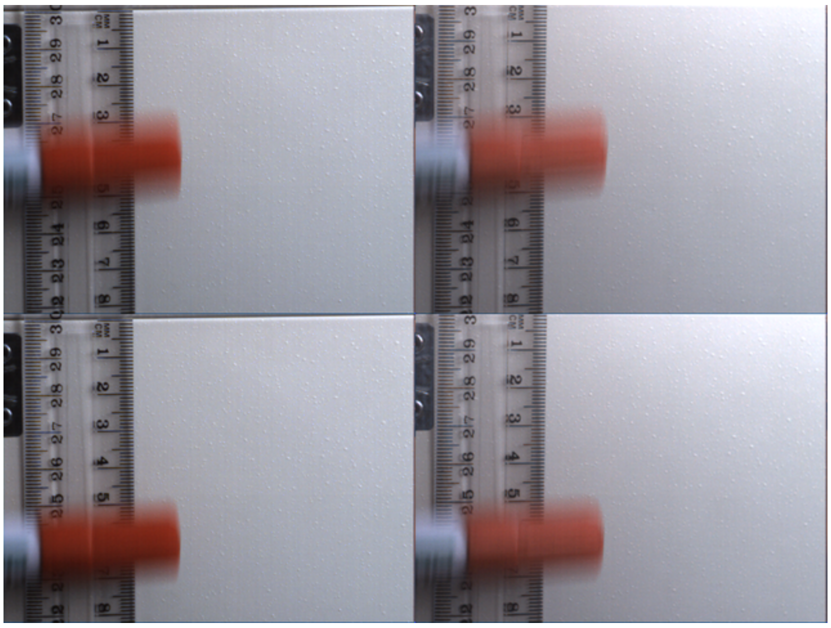


Figure 5.2: Consecutive frames of a pen being waved in front of two cameras being triggered externally.

5.1 Using an External Trigger

The Point Grey cameras we are using have general purpose I/O pins which can be configured to trigger the start of image integration on a voltage transition. As illustrated in Figure 5.1, using a TTL 5V periodic signal output from a 555 timer-based astable multivibrator circuit, we are able to externally trigger the image acquisition of any number of cameras located on different FireWire buses *synchronously* at a rate equal to the frequency of the trigger voltage.

5.1.1 External trigger results

Figure 5.2 shows 2 sequential frames of our pen-waving experiment. The cameras were attached to *different* FireWire buses on *different* PCs. Despite the blur (which may be ameliorated by reducing the integration time), both the top and the bottom pairs of corresponding images clearly show the marker in precisely the same position for left and right images, confirming that the cameras are synchronised. Indeed, aside from viewpoint differences between the cameras, there is no visible disparity between the left and right images, suggesting that the cameras are perfectly synchronised to one another.

Whilst we are able to visually confirm the synchronisation between the two cameras, it is useful to be able to quantify the discrepancy (if any) between the times the images are taken by the different cameras, and hence verify that the synchronisation satisfies our requirement of no more than 2 ms difference. For cameras located on the same FireWire bus this task is made easy by the Point Grey cameras with their `FRAME_TIMESTAMP` functionality.

5.1.2 FRAME_TIMESTAMP

Point Grey cameras such as the Scorpions we tested implement a time-stamping function which allows captured images to be stamped with the value of the FireWire bus' 32-bit `CYCLE_TIME` register at the instant the camera's shutter is closed and integration for that image is complete.[1] This `FRAME_TIMESTAMP` is embedded into the first 4 bytes of each image captured.

With the 2 cameras attached to the *same* FireWire bus, we enabled time-stamping and tested the synchronisation of externally triggered video capture by examining the corresponding timestamps for each image for each camera. The exact synchronisation of the externally triggered cameras was confirmed by the fact that *each corresponding time-stamp was identical* for all video sequences tested at various frame rates.

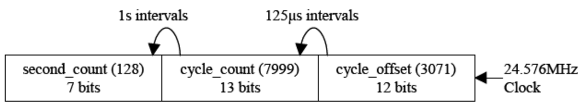


Figure 5.3: CYCLE_TIME register format. *Source:* [5], p. 63

Note however that we are unable to use this time-stamp comparison as a basis for synchronisation tests when the cameras are attached to different FireWire buses. This is because the `CYCLE_TIME` value that the `FRAME_TIMESTAMP` duplicates derives from a different physical clock for each bus and, whilst they operate at the same frequency, their counters will have been started at different times. The synchronisation of cameras located on different FireWire buses *can* however be quantified by comparing relative `FRAME_TIMESTAMP` values (given an initial offset) *if the different buses' initial values are assumed to correspond precisely.*

5.2 External Trigger Limitations

The synchronisation of multiple cameras spread across separate FireWire buses and PCs is ideally performed using this external triggering technique *except for one key limitation*, which we will now detail. The problem with using this method of camera synchronisation is that externally triggering the cameras necessarily disables the usual isochronous data transfer mode employed to stream video at constant rates. Instead, the cameras are forced into transferring data *asynchronously* across the FireWire bus.

We hypothesize that when the cameras operate in their default isochronous mode they use their on-board frame memory to buffer the current image whilst the previous image is being transferred across the FireWire bus. This buffering allows the camera the full period between frames to transfer the captured image. So if the cameras are capturing in their normal 'free-running' (i.e. isochronous) mode at 30 fps, they have $33.\dot{3}$ ms between frames to transfer the entire image across the bus. Returning to the example given in Section 2.3, this is done by transferring 320 quadlets/packet per 125 μs cycle period as specified by the DCAM spec.

[1]Note that the `CYCLE_TIME` register is simply an incremental counter of the FireWire bus' 24.576 MHz clock timing signal that all FireWire devices must implement.

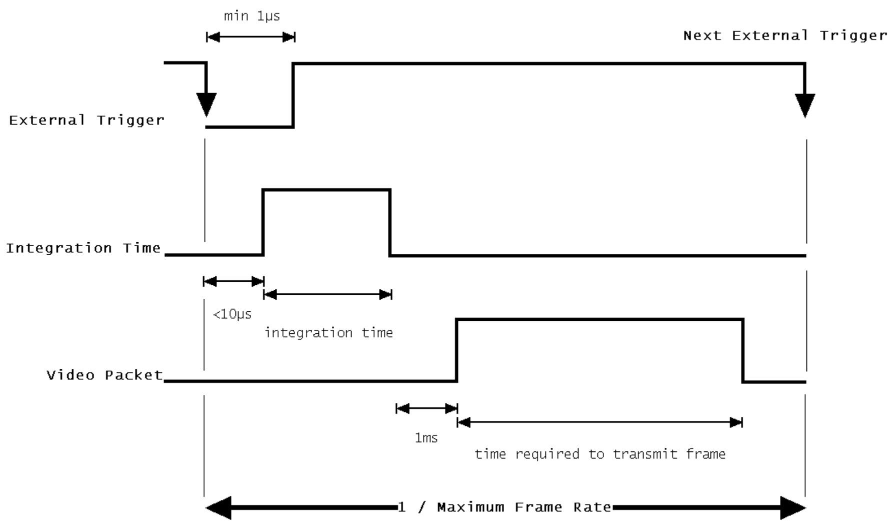


Figure 5.4: Timing of image acquisition when using an asynchronous external trigger. *Source:* [6], p. 24 (adapted)

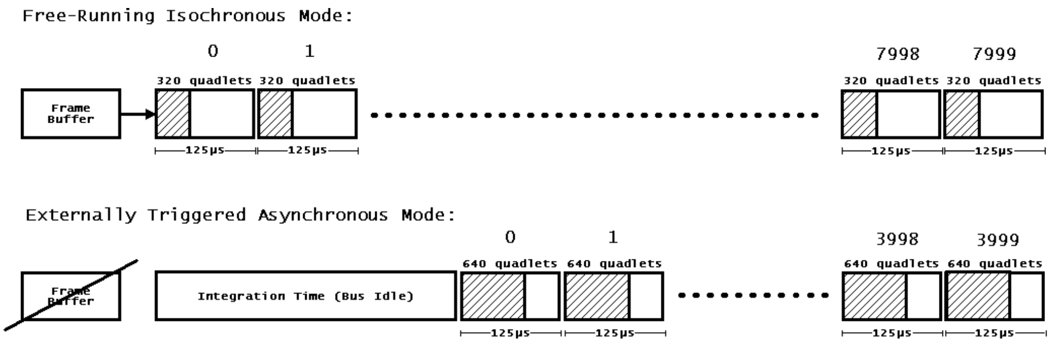

Free-Running Isochronous Mode:
0
1
7998
7999
320 quadlets
Frame Buffer
125µs
Externally Triggered Asynchronous Mode:
3998
3999
640 quadlets
Frame Buffer
Integration Time (Bus Idle)

Figure 5.5: Difference in timing and bandwidth usage for normal 'free-running' isochronous and externally triggered asynchronous capture modes.

If however, the cameras are being externally triggered, it appears that the on-board frame-buffer is disabled and subsequently the advantage of having the full period between frames to transfer the captured image is lost. Instead, this period is shared between the integration time and the time required to transfer the captured image across the FireWire bus.

Figure 5.4 illustrates the timing of a capture event between external trigger signals. Notice that the next external trigger may only be fired *after* the current frame has been entirely transferred across the bus. This is a necessary restriction to ensure that the current image is completely transferred before the next one is taken.

In order to satisfy the constraint imposed by the reduced image transmission period, the camera must operate at a higher frame rate than its actual image acquisition rate. For example, in order to perform 30 fps video capture, an externally triggered camera must be running at *60 fps*. In effect, externally triggered video capture requires the cameras to be operating at twice the normal speed.

External triggering requires cameras to operate at twice the nominal

speed to achieve a given frame rate

> Consider the example given in Section 2.3. It was shown that for 640x480 Y8 video at 30 fps it takes 30 ms of the possible $33.\dot{3}$ ms frame interval to transfer an entire frame across the FireWire bus. But, if the cameras are externally triggered, the on-board frame buffering is disabled and the integration time must be added to the frame transfer time. Even if we are using a very small integration time, say 12.8 ms, this means that it will take `30 ms + 12.8 ms = 42.8 ms > 33.3 ms` to transfer each frame. If the next external trigger signal is given before this 42.8 ms period (i.e. at $33.\dot{3}$ ms to achieve 30 fps), it will simply be dropped. Therefore every second trigger signal is dropped and as a result the camera actually captures at *15* fps and not the requested 30 fps.
>
> The solution is to force the camera to capture at a higher frame rate than the actual image acquisition rate. Referring again to Section 2.3, it was shown that for 640x480 Y8 video at *60* fps it takes 15 ms to transfer an entire frame across the FireWire bus. Now, if we add this time to the integration time we obtain `15 ms + 12.8 ms = 27.8 ms < 33.3 ms` to transfer each frame. Now, if the next external trigger signal is received at $33.\dot{3}$ ms the camera will be ready to receive it as the previous frame will have already been transferred across the bus. Of course, at 60 fps, *twice* the bandwidth of the FireWire bus is being used as well. In this case, it means that only *one* camera per FireWire bus is possible (as opposed to *three* cameras when not using an external trigger).
>
> Figure 5.5 illustrates the differences in timing and bandwidth usage between normal ‘free-running’ and externally triggered video capture for the given example of 30 fps capture. The comparision highlights in particular the necessary bus idle time caused by external triggering during which image integration is performed, as well as the subsequent need for double the normal bandwidth due to the now shortened time window in which the frame must be transferred.

The camera capture processes for both normal ‘free-running’ isochronous and the externally triggered asynchronous modes of image acquisition are summarised in Figure 5.6. The reason externally triggered video capture must simulate the usual isochronous data transfer mode using this ‘rushed’ asynchronous technique is due to the nature of the trigger signal itself. Although we know that we are supplying a continuous periodic external trigger signal, it remains nonetheless an *asynchronous* signal as far as the camera is concerned — the camera only ever sees the current asynchronous trigger and is unable to preempt the arrival of a possible future trigger. Therefore, it must not be allowed to buffer the captured image, but rather it must ‘rush’ it across the bus at twice the usual rate in order to be ready for the next trigger.

To summarise, we found it possible to externally trigger any number of cameras located on different FireWire buses both within and across host PCs to capture perfectly synchronised images. The drawbacks of using external triggering that we have observed are that (a) it requires that the cameras be able to operate at twice the normal speed for any given frame rate, and (b) subsequently double the usual FireWire bandwidth is required per camera.

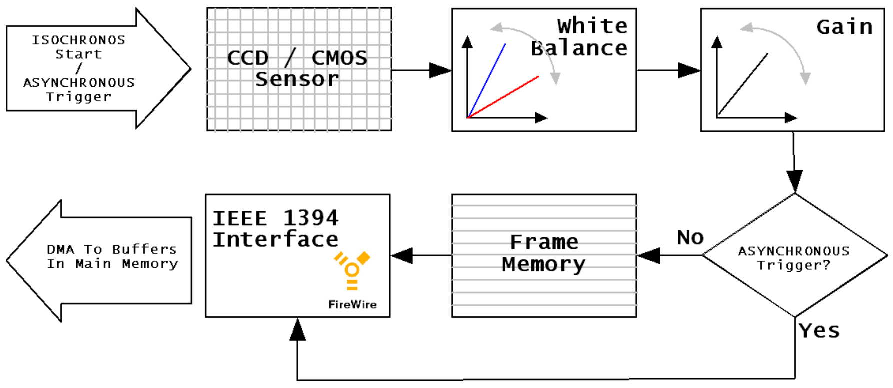


Figure 5.6: Flow diagram for normal isochronous and externally triggered asynchronous modes.

Since the cameras we have chosen are capable of 640x480 at 60 fps and 1024x768 at 30 fps, we are still able to achieve rates of 30 fps and 15 fps for the two video modes respectively. However, instead of being able to have 3 cameras attached to the one FireWire bus at 640x480 and 2 cameras at 1024x768, the double bandwidth demands of an externally triggered system requires *one FireWire bus per camera.*

Chapter 6

Synchronisation of Cameras on the Same FireWire Bus

A feature of the Point Grey cameras we tested is that they synchronise to within 125 μs of each other when they are connected to the same FireWire bus. This synchronisation is done at a hardware level and is only available when using the Point Grey proprietary (Windows XP) drivers and the API libraries included in the C/C++ `PGR FlyCapture SDK`. According to Point Grey,

> . . . the grabbing of images will automatically be synchronized at the hardware level using timing information provided by the 1394 bus.
>
> There is no master camera to which the other cameras are synced. The way synchronization is achieved is that each camera is constantly receiving the 1394 bus cycle timer information, and a register on the camera contains this cycle time information. . . The camera firmware is designed to grab at selected intervals of the 1394 cycle time, so each camera on the bus has the same cycle time info and can grab at the same cycle time interval[4].

This explanation given by Point Grey remains vague on the details of implementation. By testing the cameras and writing our own `capture` program using the `PGR FlyCapture SDK` we have arrived at a probable explanation of how the synchronisation actually works; which we now present in detail.

6.1 How Automatic Synchronisation Works

To follow our explanation of how the automatic intra-bus camera synchronisation is performed, it is instructive to study the pseudo-code of our `capture` program, given in Figure 6.1. The code is fairly self-explanatory; the `flycapture*` API functions are unambiguously named.

After the initial setup for the cameras is complete, a call to `flycaptureStartLockNext()` is made for each camera on the FireWire bus. Each of these calls starts the isochronous DMA streaming of images from the camera to the frame buffers in main memory (Figure

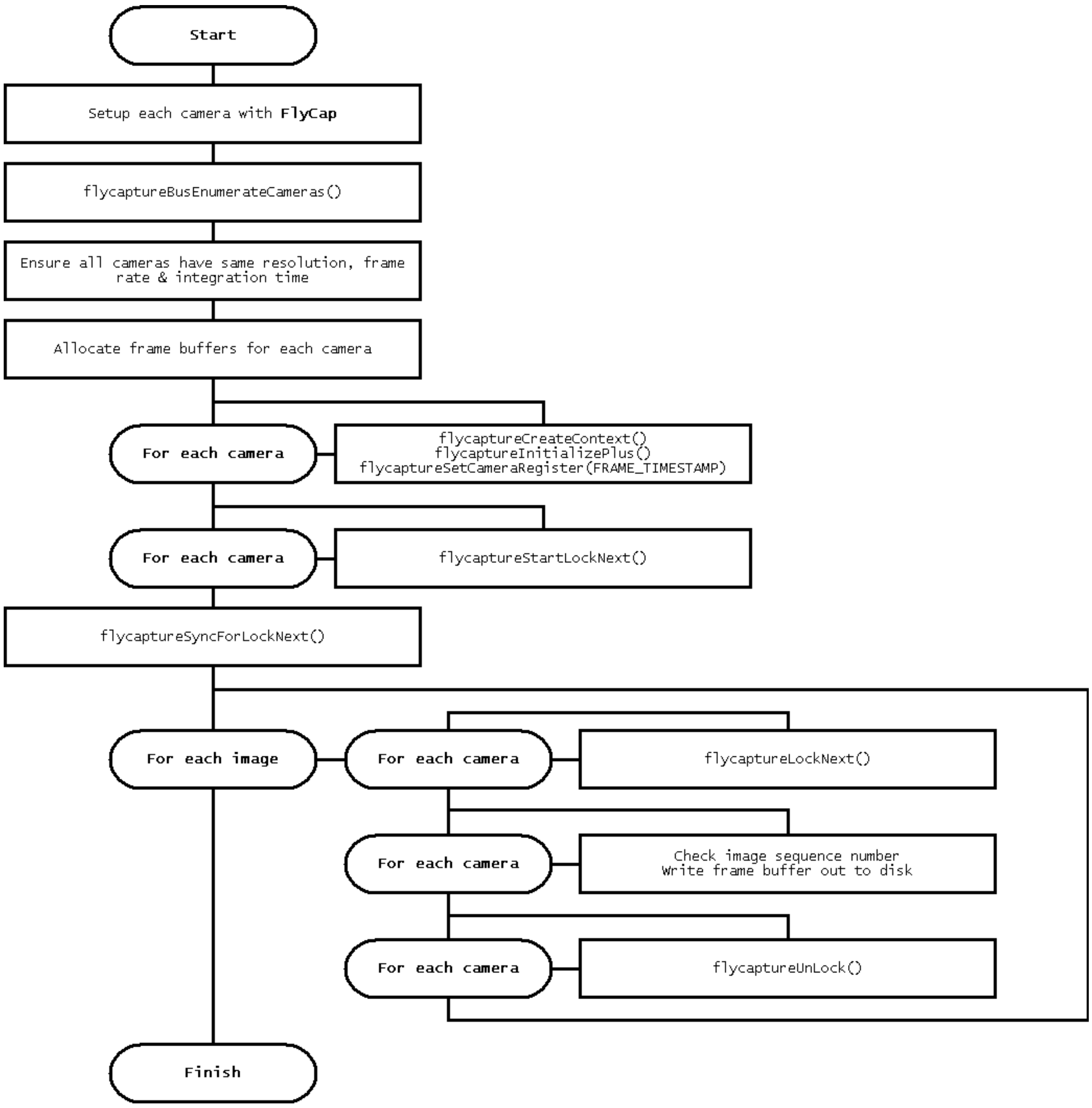

Start
Setup each camera with FlyCap
flycaptureBusEnumerateCameras()
Ensure all cameras have same resolution, frame rate & integration time
Allocate frame buffers for each camera
For each camera
flycaptureCreateContext()
flycaptureInitializePlus()
flycaptureSetCameraRegister(FRAME_TIMESTAMP)
For each camera
flycaptureStartLockNext()
flycaptureSyncForLockNext()
For each image
For each camera
flycaptureLockNext()
For each camera
Check image sequence number
Write frame buffer out to disk
For each camera
flycaptureUnLock()
Finish

Figure 6.1: Pseudo-code for `capture`.

Frame Rate	**Frame Period**	**Equivalent** `cycle_count`
7.5 fps	$133.\dot{3}$ ms	1066 or 1067
15 fps	$66.\dot{6}$ ms	533 or 534
30 fps	$33.\dot{3}$ ms	266 or 267
60 fps	$16.\dot{6}$ ms	133 or 134

Table 6.1: Standard frame rates and their equivalent `cycle_count` values.

4.1). The 'lock next' mode ensures that the next full frame buffer (i.e. complete image in memory) is locked and not able to be overwritten until the user explicitly unlocks it with `flycaptureUnLock()`.

Notice that the `flycaptureStartLockNext()` calls for each camera are made sequentially, *not concurrently.* As a result, if the cameras actually started integrating images immediately after their own `flycaptureStartLockNext()` call, operating system and user-space process latencies between calls would mean that they would *not* be synchronised to one another. If, however, the `CYCLE_TIME` register value of the IEEE-1394 bus at the instant that the first camera began integrating images was made available to the subsequent cameras, these cameras could potentially use this information to synchronise themselves to the first camera.

Figure 5.3 shows that the 32-bit `CYCLE_TIME` register is divided into 3 fields containing wrapping counters: `second_count`, `cycle_count` and `cycle_offset`. Camera synchronisation is quoted accurate to within 125 μs which indicates that the `cycle_count` value is used for synchronisation purposes. Now imagine that `flycaptureStartLockNext()` is called for the first camera on the FireWire bus and it decides to start integrating images when `CYCLE_TIME, cycle_count` reaches `0042`. The camera's firmware instructs it to store this `initial_cycle_count` value.

When `flycaptureStartLockNext()` is called for any subsequent cameras, these cameras first query the bus for any instance of `initial_cycle_count`. When `initial_cycle_count = 0042` is returned to these cameras, their firmware will instruct them begin their image integration at some time `initial_cycle_count + n * frame_rate_cycle_count`, where `frame_rate_cycle_count` is the equivalent `cycle_count` value for the given frame rate. For example, at 30 fps, `frame_rate_cycle_count = 266 or 267` since `266` $\times$ `125` μ`s = 33.25 ms` and `267` $\times$ `125` μ`s = 33.375 ms`. A list of frame rates and their equivalent `cycle_count` values is given in Table 6.1.

Starting integration at `initial_cycle_count + n * frame_rate_cycle_count` ensures that these subsequent cameras are synchronised to the first camera to within 125 μs, *but possibly offset by an integral multiple of a frame period.* Referring back to the pseudo-code, we notice that, immediately following the `flycaptureStartLockNext()` calls and directly before the image grab loops, a call to `flycaptureSyncForLockNext()` is made. This function accounts for the possible frame offsets introduced by `n * frame_rate_cycle_count` by skipping the required number of frame buffers for each camera started *before* the *last* camera. This effectively aligns the frames being captured by each camera on the FireWire bus, thereby synchronising the image acquisition of all the cameras to one another.

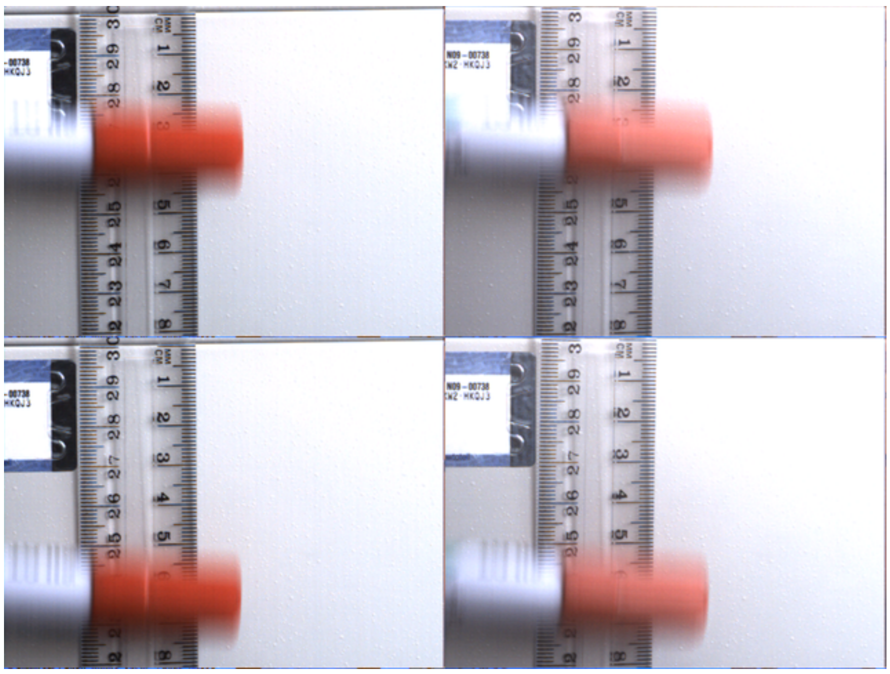


Figure 6.2: Consecutive frames of a pen being waved in front of two cameras located on the same FireWire bus.

6.2 Automatic Synchronisation Results

To visually test this hardware-level intra-bus synchronisation we again performed our standard pen-waving experiment. For this method the two cameras were attached to the same FireWire bus and configured to stream in normal isochronous mode. Notice again that the corresponding images from left and right cameras appear perfectly synchronised to one another for each frame.

Since the cameras were located on the same FireWire bus, we were able to quantify the synchronisation as explained in Section 5.1.2 with the image time-stamp values. As opposed to the external triggering method, we found the Point Grey cameras' automatic intra-bus camera synchronisation to not always be precise. The image time-stamps often reported the cameras to be one `cycle_count` out of sync. But this was to be expected since the guarantee given was that automatic synchronisation was accurate to within 125 μs (i.e. one `cycle_count`). Table 6.2 shows the average percentage of frames that were out of sync by one `cycle_count` for 500, 1000 and 1500 frame capture sequences at various resolutions and frame rates.

Note that the cameras *never* reported themselves to be out of sync by more than one `cycle_count`. Given that 125 μs is only 1/16th of our 2 ms 'out of sync' definition, we consider these synchronisation results to be quite acceptable.

Resolution	**640 x 480**			**1024 x 768**			**1280 x 960**		
No. Frames	**500**	**1000**	**1500**	**500**	**1000**	**1500**	**500**	**1000**	**1500**
1.875 fps							16	15	13
3.75 fps	14	13	12	14	12	12	14	13	14
7.5 fps	13	12	12	12	12	12	11	12	13
15 fps	10	11	10	11	10	11			
30 fps	12	11	11						

Table 6.2: Percentage of images out of sync by 125 μs for various resolutions and frame rates for video capture sequences of 500, 1000 and 1500 images.

Chapter 7

Synchronisation of Cameras on Different FireWire Buses

In Chapter 5 we explored the possibility of using an external trigger to synchronise multiple cameras located on different FireWire buses both within and across host PCs. It was shown that such a synchronisation method functions ideally except for the bandwidth limitations imposed by the asynchronous mode required by externally triggered cameras.

Next, in Chapter 6, we showed how Point Grey cameras are able to overcome these bandwidth limitations by synchronising to one another at the hardware level in normal isochronous mode. Unfortunately this method of synchronisation is restricted to cameras located on the *same* FireWire bus, which limits us to only 2–3 synchronised cameras.

With the multiple view digital video capture system that we are developing we require a scalable means of synchronising any number of cameras attached to *different* FireWire buses located on different host PCs. Whilst the external triggering method functions perfectly, we would prefer to avoid the bandwidth limitations that it necessarily imposes. To achieve this, we now explore methods of synchronising separate FireWire buses which are each hosting automatically intra-bus synchronised Point Grey cameras.

7.1 Point Grey Sync Unit

For the purpose of synchronising cameras across FireWire buses, Point Grey have developed a ‘Sync Unit’ which it claims is able to effectively synchronise the image acquisition of cameras attached to different IEEE-1394 buses. It is important to note here that this Sync Unit is

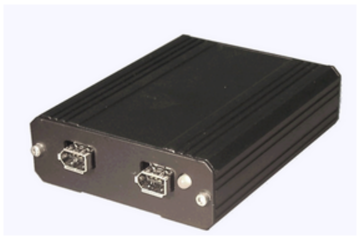


Figure 7.1: Point Grey Sync Unit.

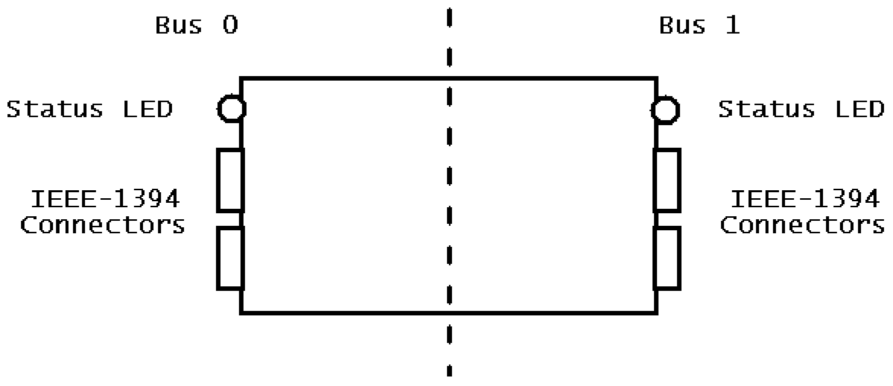


Figure 7.2: The Sync Unit bridges timing information across 2 buses.

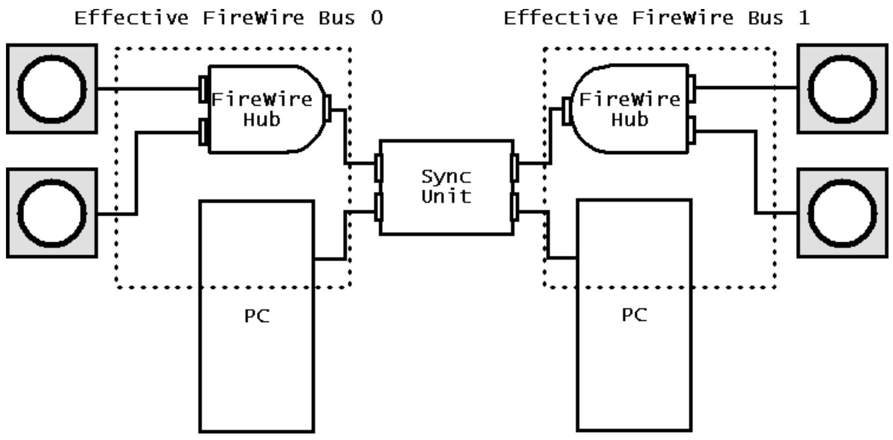


Figure 7.3: Synchronising 4 cameras across 2 buses using the Sync Unit.

not a generic IEEE-1394 camera synchronisation device and has been designed specifically to work with particular Point Grey cameras only (including the Fleas and newer model Scorpion SCOR-13FF but not the old SCOR-13SM). Information on the Sync Unit may be found at `http://www.ptgrey.com/products/sync/index.html`.

According to Point Grey, it essentially works by bridging the timing information between the 2 buses to which it is connected. An example of this functionality is given in Figure 7.3. In this configuration, the unit is being used to synchronise 4 cameras across 2 PCs (2 cameras per FireWire bus per PC). Note that each side of the Sync Unit only 'sees' the effective FireWire bus to which it is attached.

If more than 2 IEEE-1394 buses are required to be synchronised, multiple Sync Units may be daisy-chained together to form an independent 'synchronisation bus'. Figure 7.4 shows an example of such a configuration. Notice that *for more than 2 buses, the number of Sync Units required is equal to the number of IEEE-1394 buses being synchronised.*

7.1.1 How the Sync Unit Works

Point Grey does not provide any specific details on how these Sync Units work. However, given our explanation of how the Point Grey cameras achieve *intra*-bus synchronisation (Chapter 6),

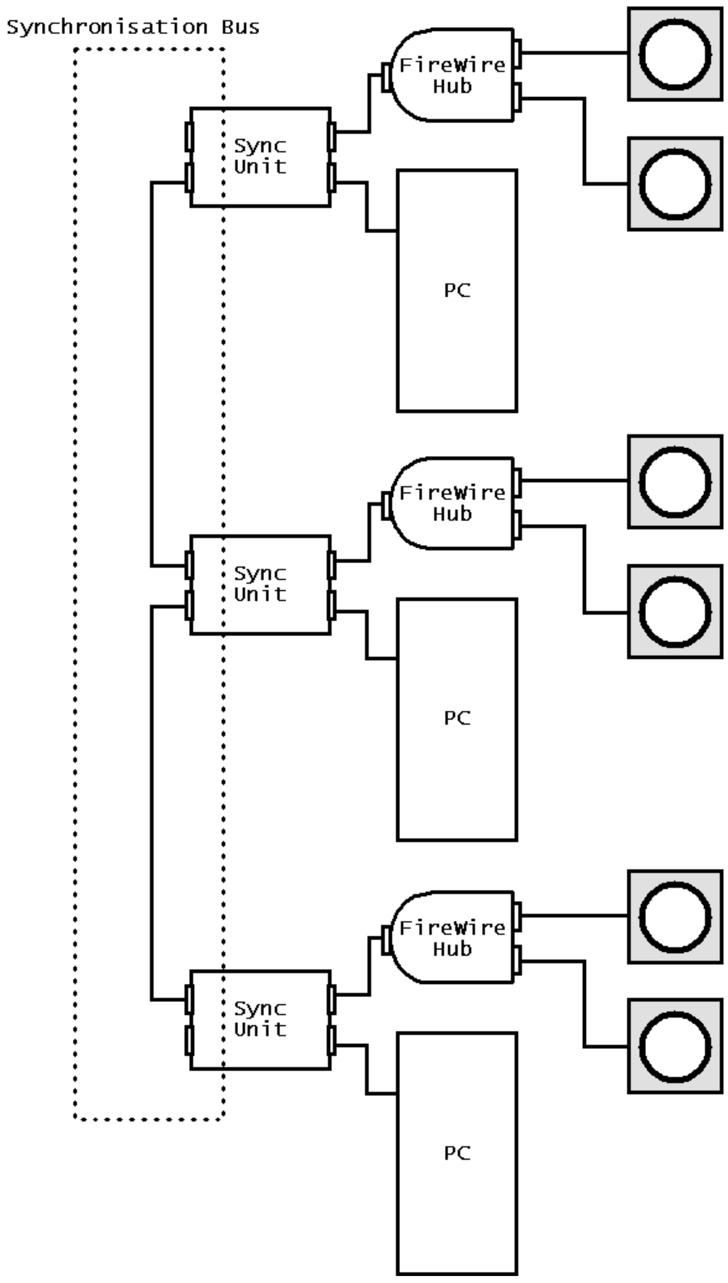


Figure 7.4: Synchronising 6 cameras across 3 buses by daisy-chaining 3 Sync Units.

we argue that the *inter*-bus synchronisation functionality of the Sync Unit may be explained as a logical extension. The following is our hypothesis of how we believe the Sync Unit functions in light of our experiments and observations.

It appears that the Sync Unit operates by passing on the `CYCLE_TIME` register information from the first camera seen by it to any subsequent cameras. These cameras then base the timing of their image acquisition on this value, effectively synchronising themselves to the first camera.

As has been previously explained, the timing of any IEEE-1394 bus is governed by it's own 24.576 MHz clock which, for Point Grey cameras, is incrementally counted by their 32 bit `CYCLE_TIME` register. Obviously the clocks of different buses are going to be inherently out of sync as they are running off separate physical clocks which will most likely have been started at different times.

The first task of the Sync Unit then must be to determine the offset between the clock values of the 2 separate buses attached to it. To achieve this, the device must take a 'snapshot' of the 2 buses' `CYCLE_TIME` registers at a given moment. Say, for example, that at time $t_{snapshot}$ the values are `bus0_cycle_count = 0021` and `bus1_cycle_count = 0347`. This gives `bus_offset = 326`.

Having determined the clock offset between the 2 buses attached to the Sync Unit, now imagine that in Figure 7.3 the first camera to be attached to either of the PCs is the top one on the left hand side. Say `flycaptureStartLockNext()` is called for this camera and it starts streaming at `initial_cycle_count = 0042`. Now when the bottom camera is attached and `flycaptureStartLockNext()` is called, its firmware instructs it to start streaming only at some time `initial_cycle_count + n * frame_rate_cycle_count` (Section 6.1). Thus the streaming of the 2 cameras on the left side bus is effectively synchronised.

This `initial_cycle_count` value is propagated across the device to the bus on the other side. Given this value and the `bus_offset`, when `flycaptureStartLockNext()` is subsequently called for cameras attached to the right side bus, their firmware will instruct them to start streaming only at some time `initial_cycle_count + (n * frame_rate_cycle_count) + bus_offset`. Thus their image acquisition will be effectively synchronised to that of the cameras on the left side bus. This process extends readily to the configuration of daisy-chained Sync Units presented in Figure 7.4. Here the `initial_cycle_count` value is propagated along the length of the left side 'synchronisation bus' so that cameras attached to the right side FireWire buses may all be synchronised to one another.

7.1.2 Sync Unit Issues

These Sync Units developed by Point Grey are clearly ideal for the synchronisation of multiple cameras across multiple FireWire buses that we require, except for a couple of key factors. Firstly, the Point Grey Scorpion SCOR-13SM cameras which we have are not actually compatible with the Sync Unit. Furthermore Point Grey have not indicated that they intend to implement such functionality in the near future. Fortunately, the Point Grey Flea cameras which we intend to use for all additional cameras to our system are indeed compatible with the device and thus able to be synchronised using the Sync Unit. However this still leaves us with the problem of how to synchronise a heterogeneous mix of Sync Unit compatible and non-compatible cameras.

Secondly, synchronising cameras with Point Grey Sync Units is far from an inexpensive exercise. As explained above, for any more than 2 FireWire buses, the number of (rather expensive) Sync Units must equal the number of buses required to be synchronised. Note finally that, as of yet, we have not been able to test the functionality of the Sync Unit because we only have multiple SCOR-13SM cameras, which are not supported by the device.

7.2 Software Synchronisation

To replicate the Sync Unit's inter-bus synchronisation functionality in software as we have explained it, we require the ability to control camera capture timing at a low level. More specifically, we require that `flycaptureStartLockNext()` be able to accept a user-defined `cycle_time` value as an argument. Unfortunately this functionality is not provided by the `PGR FlyCapture SDK`, nor is it found in other 1394-based digital camera control libraries such as `libdc1394`.

The alternative method of inter-bus synchronisation we devised involves networking the separate host PCs and using a software-based trigger to start all the cameras capturing at the same time. We assume that the cameras located on the different FireWire buses are automatically synchronised with one another at the hardware level (Chapter 6). Thus rather than triggering each *individual camera* at the same time (as is done with the external triggering), we simply need to ensure that each *FireWire bus* is triggered to begin streaming at the same time.

7.3 Using an Ethernet trigger

The host PCs we are using for video capture are interconnected via 10 Mbit/s Ethernet using a network switch. To implement the software-level synchronisation trigger, we used the `Winsock` networking API to modify our `capture` program (Figure 6.1) and write a corresponding `trigger` program.

Each host PC is running an instance of the 'client' `capture` program specifically configured for the cameras attached to its FireWire bus. Once `capture` has completed the initial setup for the cameras, and immediately before any of the `flycaptureStartLockNext()` calls are made, it blocks and waits for the 'trigger signal' from the 'server' `trigger` program which is also running on one of the host PCs.

The 'trigger signal' itself is simply a UDP broadcast message sent by `trigger` to the network's directed broadcast address on a specified port. Whilst they are blocking, the client `capture` programs are listening to the same port for the broadcast message from the server `trigger`. The instant the broadcast message is received, the `capture` programs continue execution with the calls to `flycaptureStartLockNext()` to start the cameras streaming isochronously. Thus, if all the instances of `capture` are blocking before the trigger signal is sent, `flycaptureStartLockNext()` should be executed synchronously for all host PCs upon the broadcast message's arrival.

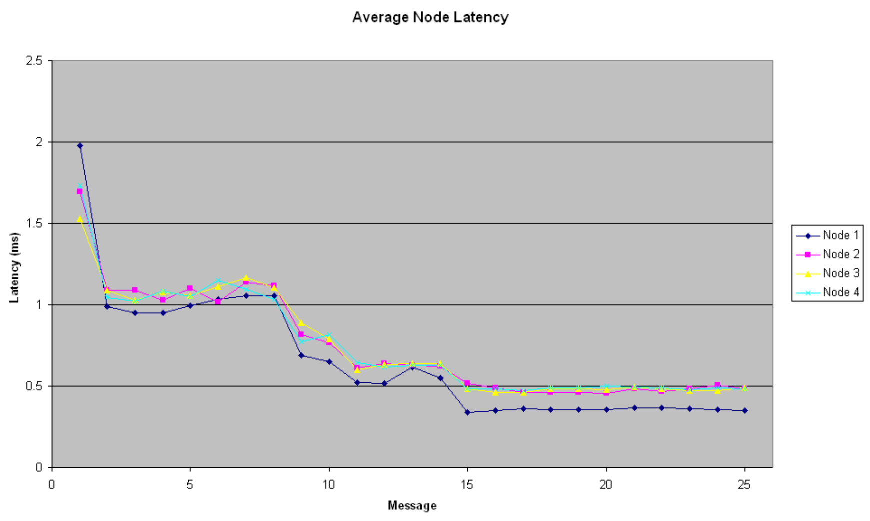


Figure 7.5: Average one-way latency times observed for UDP messages between server and client.

7.3.1 Latency Under Windows XP

Of course, the timing of the ideal situation described above is, in practice, subject to latencies introduced by the Ethernet network and by the Windows XP operating system on which the `capture` and `trigger` programs are running. To ensure that we achieve synchronisation results as close to the ideal as possible, our system must account for these introduced latencies.

Windows XP is not a real-time operating system — the user it not guaranteed instantaneous execution of a process on demand. Instead, it is a multitasking operating system based on a preemptive multitasking kernel. The kernel handles the simultaneous execution of multiple processes/threads by scheduling their execution according to their 'priority levels'. It is also able to 'preempt' when a certain process/thread should be executed.

Given that our `trigger` and `capture` programs operate within a multitasking environment, we face the possibility that, contrary to our expectations, `capture` may not respond instantaneously to the trigger signal sent by `trigger` because (a) the operating system may have scheduled some other process/thread with a higher priority to execute at that instant or (b) the operating system may have put the `capture` process to 'sleep', in which case it must be 'awoken' before it can begin executing. Due to the time-critical nature of our task, the latencies produced by either situation may be large enough to invalidate the synchronisation. In particular, waking up a thread in Windows XP takes tens of milliseconds [8], whereas we require synchronisation to be within 2 ms.

We attempt to minimise the potential for such latencies by using the Windows API function `SetThreadPriority()` to set the priority levels to `THREAD_PRIORITY_TIME_CRITICAL` for `capture` and `trigger` — the highest possible user-space priority level (only the few kernel-space threads using `REALTIME_PRIORITY_CLASS` have higher priority).

Also, to ensure that the `capture` processes are 'awake' at the time the trigger signal is sent, `trigger` first sends a series of identical preamble messages to each host PC before broadcasting the actual trigger signal. We tested a system setup comprising of 4 nodes — 3 client PCs running `capture` and 1 client/server PC running both `capture` and `trigger`. Using the Windows API function `QueryPerformanceCounter()` to obtain high precision time measurements, we were able to measure the average round-trip latency and (hence estimate the average one-way latency) from `trigger` to each `capture` node over a series of 25 preamble messages, the last one of which being the actual broadcast trigger signal. The results are displayed in Figure 7.5 (note that Node 1 is the client/server PC).

The graph clearly shows that the latency between server and client reduces steadily up to the 15th preamble message. The regularity observed may be explained as the preemptive kernel 'learning' to be ready for the next message. The lack of variation in latency times after the 15th preamble message suggests that the `capture` programs are now guaranteed to be 'awake' and ready to receive the final broadcast trigger signal. Notice that there is negligible difference in one-way latency time from the server to the client PCs by the 25th message, meaning that they should all receive the trigger signal at almost exactly the same time. The PC which is running both the client and server processes naturally has a lower latency (since it is effectively sending messages to itself) and will thus receive the trigger signal before the other client PCs. But, as seen in the graph, it will only do so approximately 250 μs before the others, meaning that all 4 nodes are still synchronised to within a small fraction of 2 ms as we require.

We extensively tested our `trigger/capture` with an actual experimental setup consisting of two cameras, each attached to the FireWire buses on different host PCs, being software-triggered to stream video synchronously. Over a sample of 25 tests, the reported time difference between receiving the trigger signal for the two PCs (and hence the degree of synchronisation between the FireWire buses) was on average ≈ 150 μs, with a minimum of ≈ 100 μs and maximum of ≈ 250 μs. Again, these values are but a small fraction of our 2 ms allowable discrepancy, and thus the `trigger/capture` Ethernet synchronisation method satisfies our synchronisation requirements.

7.3.2 Ethernet Synchronisation Results

To visually test the accuracy of our Ethernet synchronisation we once again performed our standard pen-waving experiment. For this method the two cameras were attached to different host PCs. They were triggered to stream in normal isochronous mode by the Ethernet trigger. Observing Figure 7.6 closely, it can be seen that the left and right images for the consecutive frames shown are clearly out of sync (an observation that was not able to be made for the previous external triggering and automatic synchronisation tests).

The noticeable difference in position of the pen between left and right images seen in Figure 7.6 suggests that the ≈ 150 μs average time discrepancy for the Ethernet synchronisation stated above cannot fully account for the actual discrepancy observed in the image acquisition times of the different cameras. The difference in position of the pens appears to be about 4 mm. From the assumptions about the speed at which the hand holding the pen was being waved (Chapter 5), at 2 mm/ms this 4 mm discrepency equates to a 2 ms difference in image acquisition times — our maximum allowable value to still be considered synchronised.

Whilst we still consider this degree of synchronisation to be acceptable, it highlights a fundamental flaw in the premise upon which we based the Ethernet synchronisation method. Our

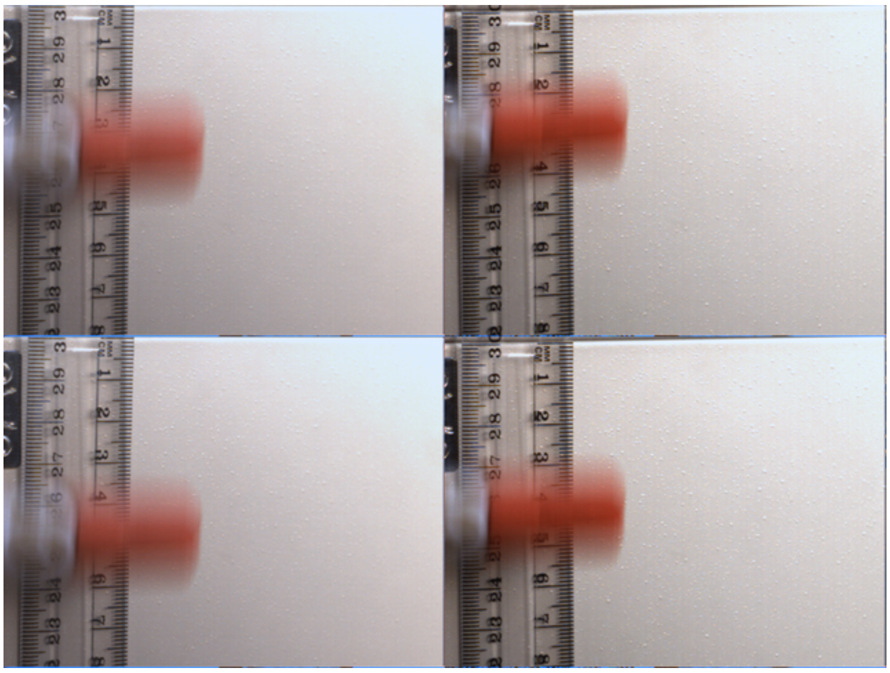


Figure 7.6: Consecutive frames of a pen being waved in front of two cameras attached to different host PCs and being triggered to capture isochronously over Ethernet.

assumption was that the call to `flycaptureStartLockNext()` started the camera streaming *immediately*. On the contrary however, it appears that `flycaptureStartLockNext()` takes a non-deterministic period of time to initiate the video streaming. This means that whilst `trigger` reports that there is only a ≈ 150 μs discrepancy between the times that the trigger signal is received by the various instances of `capture`, any given camera may take up to a couple of milliseconds to actually start its streaming in response to the broadcast trigger signal.

7.3.3 Further Synchronisations Tests

Digital Clock

We replicated the digital clock synchronisation test that Rai *et al.* [7] performed to test the synchronisation of their similarly software-triggered multiple FireWire camera system. In our experiment, we tested our `trigger/capture` Ethernet synchronisation method using all four of our Scorpion cameras (two colour and two mono SCOR-13SMs). We tested a two host PC system, attaching the two colour cameras to the FireWire bus on one PC and the two mono cameras to the FireWire bus on the other. All the cameras were configured to capture video at a resolution of 640x480 and a frame rate of 30 fps. They were then pointed at a CRT computer screen (85 Hz refresh rate) which was displaying a running digital stopwatch program (with millisecond resolution). The two PCs were then software-triggered to synchronously capture 20 s of video (isochronous mode). Figures 7.7 and 7.8 show images from all four cameras for frames near the start and end of the capture period respectively. Notwithstanding ghosting effects of the digits due to the refresh rate of the screen, clearly all four cameras are displaying

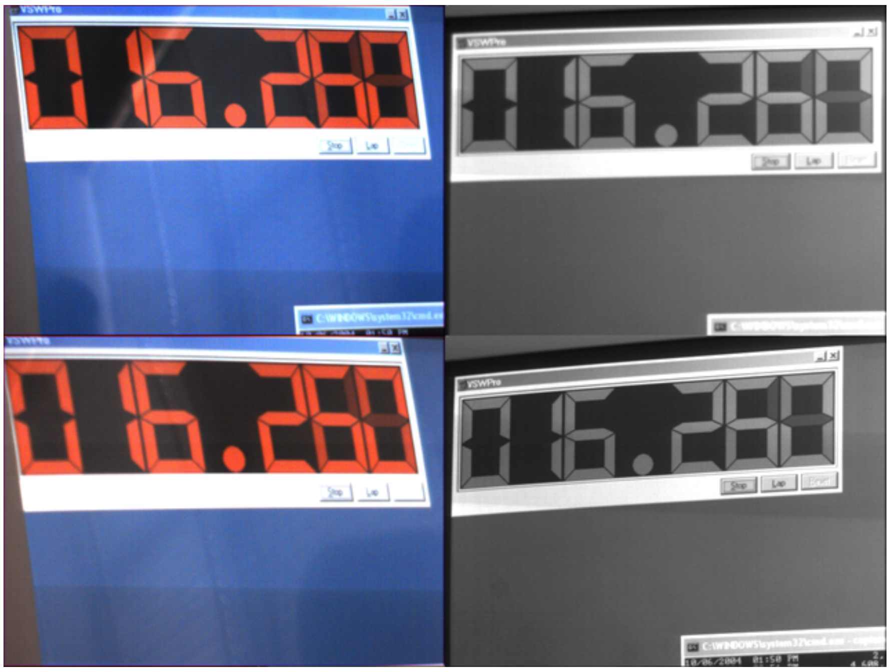


Figure 7.7: Stopwatch images from all 4 cameras for a frame near the start of capture. The top-left and bottom-left colour cameras are both located on the same FireWire bus on one host PC. The top-right and bottom-right mono cameras are both located on the same FireWire bus on a different host PC.

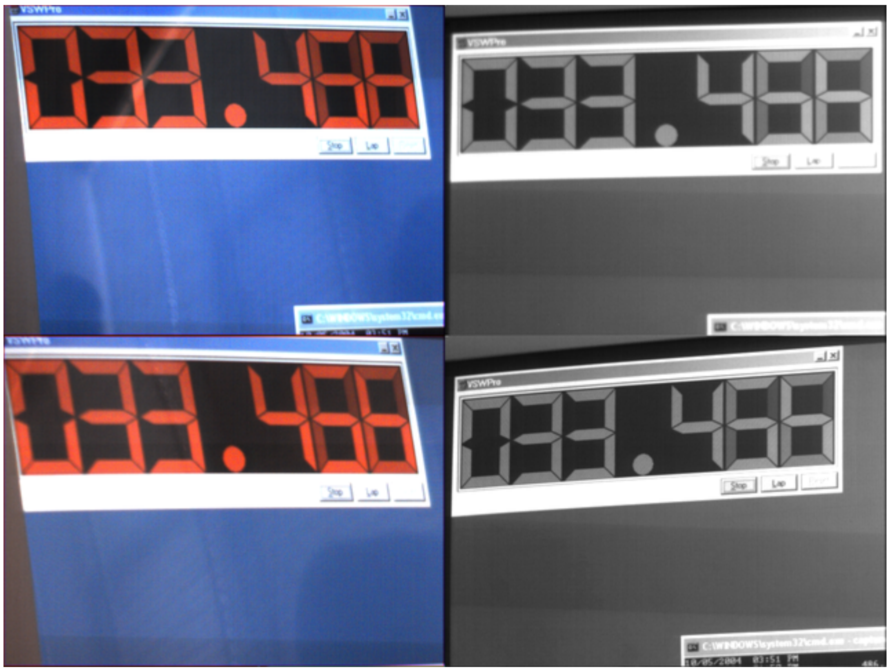


Figure 7.8: Stopwatch images from all 4 cameras for a frame near the end of capture.

the same number in both frames, visually indicating that they are synchronised to within the millisecond accuracy of the digital stopwatch.

But exactly how synchronised were the cameras? Since each pair of cameras attached to a particular FireWire bus was operating in isochronous mode, their automatic hardware-level synchronisation was readily verified by observing the image time-stamp values (Section 6.2). As expected, for each FireWire bus, the attached cameras were synchronised to one another to within 125 μs. However we cannot be as sure of the inter-PC synchronisation accuracy. As seen in Section 7.3.2, our Ethernet based inter-bus synchronisation method was only accurate to within an *estimated* 2 ms — such a synchronisation disparity could very well be lost in the refresh rate of the displayed digital clock values. Ultimately, the accuracy of testing synchronisation using a computer-based digital clock is limited in comparison to the pen-waving experiment we peformed earlier.

Milk Drops

As another, more aesthetically appealing test of the degree of synchronisation achievable using the Ethernet trigger, we performed the 'milk drop' experiment shown in Figures 7.9 and 7.10. Again, we used 2 colour cameras attached to different host PCs and triggered them to stream in normal isochronous mode with the Ethernet trigger. The image resolution was 640x480 and the frame rate 60 fps. The cameras were placed next to one another, pointing at a bowl of milk. We then captured video of a drop of milk falling into the bowl and the resultant splashing effect. From top to bottom, the figures show 10 sequential frames of the milk splashing as seen by both left and right cameras. Observing closely the shapes of the splashes we see clearly that the corresponding images from each camera are *nearly* but not perfectly synchronised. As an estimate of the level of synchronisation, let us assume that the milk drop hits the surface at anywhere between 1–2 mm/ms (a reasonable estimate considering that it was dropped from about 30 cm above the bowl and assuming that it's acceleration rate is soley due to gravity). Under this assumption, the shape difference may be explained as one camera capturing the scene at 1–2 ms before the other one — a result consistent with our expectations of our achievable software-based synchronisation levels.

7.4 Remarks on the Ethernet Synchronisation Results

Although we did not use it for our Ethernet synchronisation tests, the discrepancy between capture times could be accurately measured using the following technique. With the cameras located on different FireWire buses, the `bus_offset` value could be determined by externally triggering all the cameras to take a 'snapshot' at a given time and then examining the resultant image time-stamp values to calculate the timing offset between buses. Now when the cameras are synchronised by the Ethernet trigger, the image time-stamp values along with the determined `bus_offset` value may be compared to calculate the synchronisation discrepancy between FireWire buses.

Finally, it must be noted that although the pen-waving and milk drop tests revealed that our Ethernet synchronisation was significantly less accurate than both the external trigger method and the automatic synchronisation, these tests were specifically chosen to highlight synchronisation accuracy — they are not representative of the video capture application that

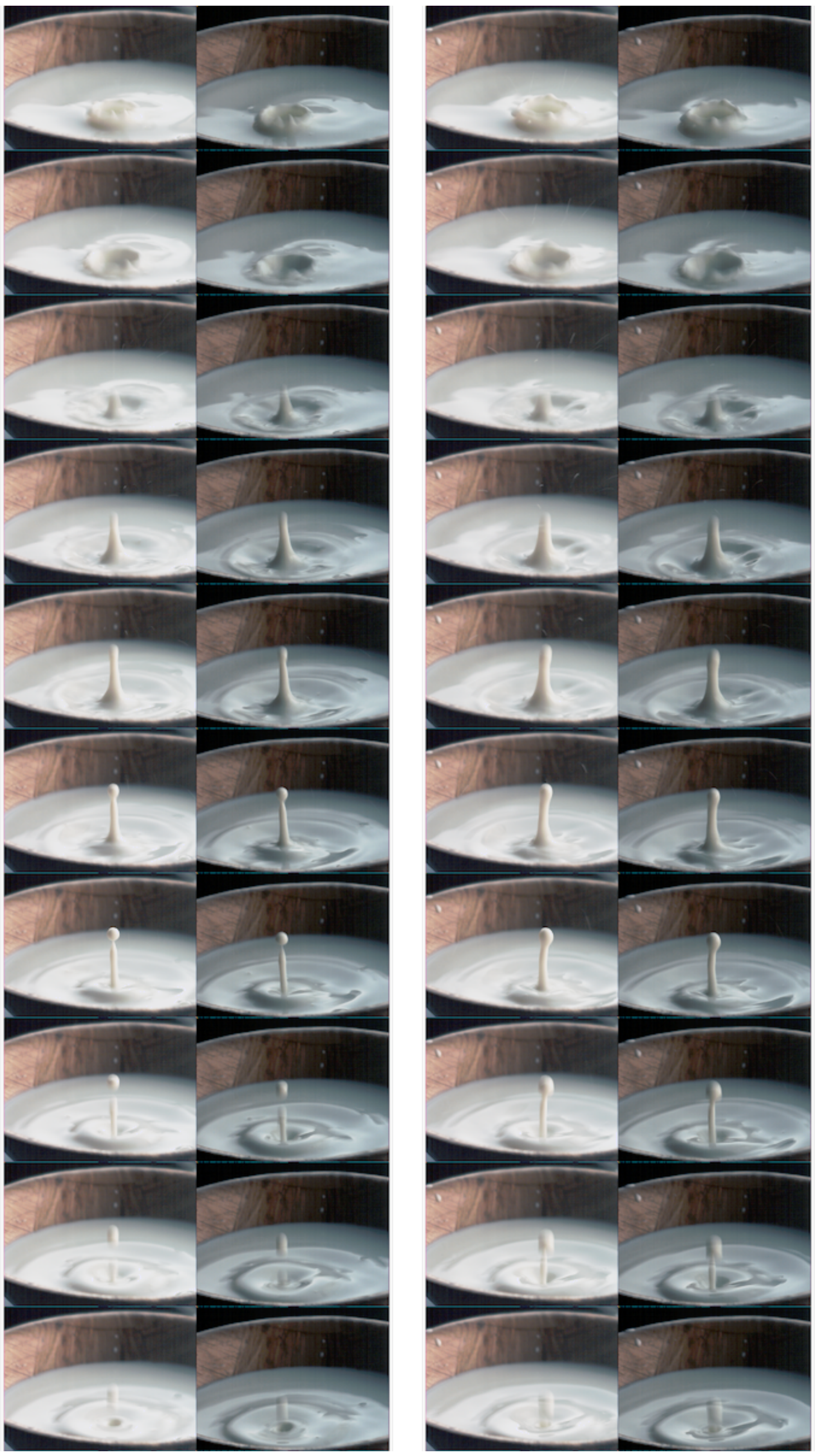


Figure 7.9: Milk drop tests 1 and 2. Left and Right: (Top to bottom) 10 consecutive frames of the result of a milk drop hitting a bowl of milk as captured by two colour cameras located on different host PCs and being triggered to capture isochronously over Ethernet.

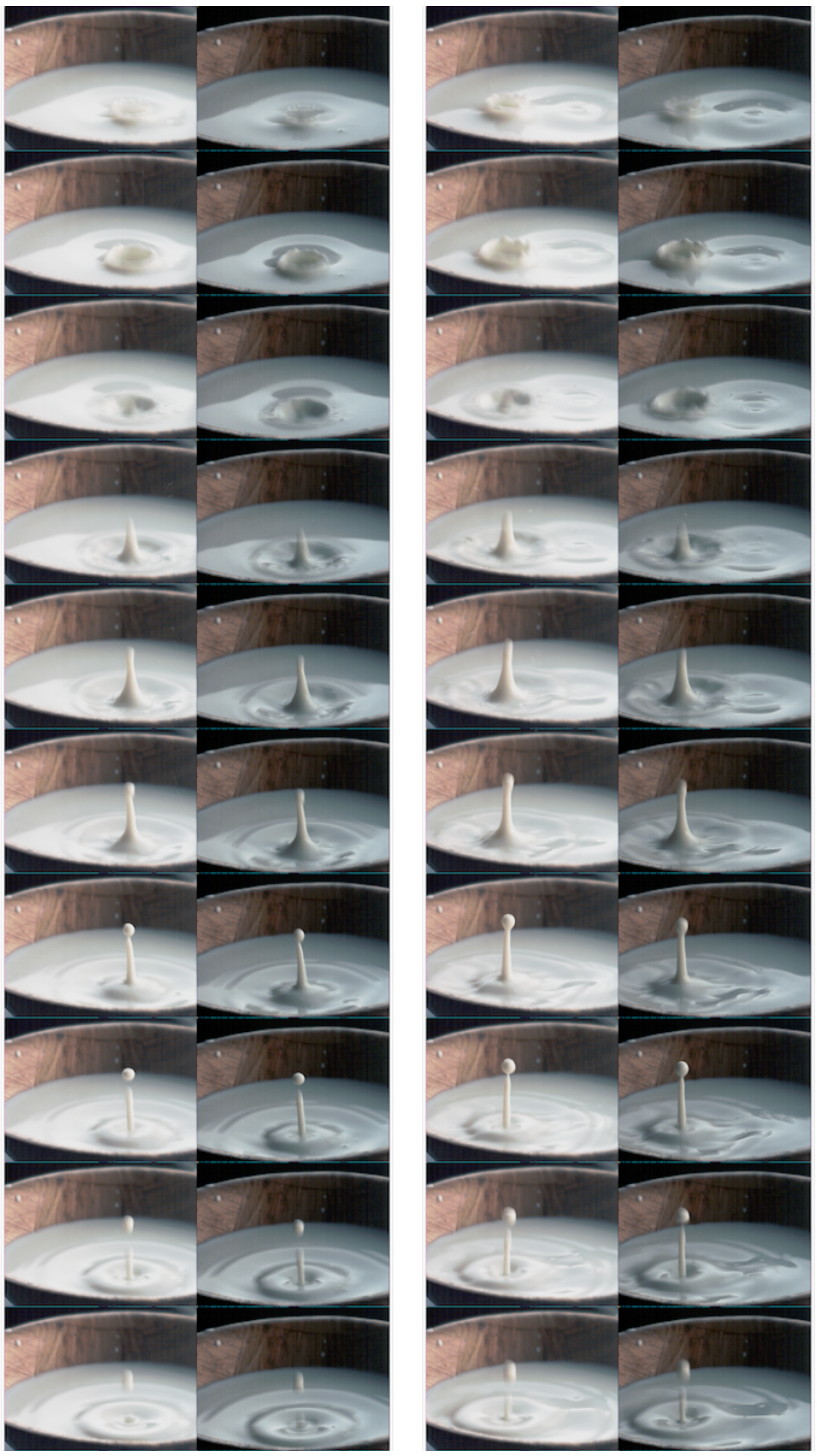


Figure 7.10: Milk drop tests 3 and 4.

we intend to perform with our multiple viewpoint system. Ultimately we will be recording the movements of a human subject in a large volume of space whose scale will most likely diminish the effect of these observed synchronisation discrepencies, meaning that the ≈ 2 ms synchronisation accuracy our method achieves is actually quite acceptable for our purposes.

7.5 Software Synchronisation Alternative — NTP

We conclude this chapter with a few remarks about an alternative method we explored to synchronise the capture from cameras attached to different host PCs. The idea here was simply to synchronise the actual system clocks of the networked host PCs using NTP and then schedule them to call `flycaptureStartLockNext()` (and hence start the cameras capturing) at a given time. The synchronised system clocks would thus effectively replace the Ethernet trigger signal.

NTP (Network Time Protocol) is a TCP/IP protocol used to synchronise the clocks of computer systems over packet-switched, variable-latency data networks such as LANs and the Internet. NTP clients run a daemon which operates by exchanging time-stamp packets with its configured NTP server(s), adjusting the local clock according to an algorithm based on the observed variations in latencies of the time-stamp packets. The NTP algorithm is particularly robust with the current version (NTPv4) being able to maintain clients' times synchronised to within 10 ms of their server(s) over the Internet, with the possibility of achieving accuracies of 200 μs or better under ideal conditions in LANs [9]. For a concise introduction to NTP refer to [2].

Tests running the NTP client on our network indicated that we were only able to synchronise the system clocks of the host PCs to within ≈ 10 ms at best, rather than the quoted 200 μs or less accuracy. Note that this was most likely due to the fact that we were synchronising to an external NTP server as we were unable to set up our own internal NTP server. In any case, the ≈ 150 μs average time difference achieved by our `trigger/capture` method betters these figures. Furthermore, as shown in Section 7.3.2, the accuracy of the the Ethernet/NTP synchronisation is overshadowed by the non-deterministic time in which the cameras actually begin their image acquisition once `flycaptureStartLockNext()` is called.

Chapter 8

Conclusion

In this report we have explored at length the issues involved in designing a system capable of synchronised multiple viewpoint digital video capture using FireWire cameras and standard PCs. In particular we have explained how the FireWire system architecture facilitates the streaming of uncompressed digital video at constant frame rates. We provided a survey of the features and capabilities of FireWire digital cameras that we used and considered for our multiple camera system. We also examined the various hardware issues that arise when interfacing FireWire cameras to host PCs for the purposes of streaming to disk. Finally we explored and tested various methods of performing the inter-camera synchronisation that we require for our system, including using an external triggering circuit to synchronise the cameras directly, as well as networking the host PCs of the cameras and using an software-based trigger to synchronise the cameras indirectly.

Based on the findings of this report, we now conclude by summarising the three possible design options we will consider for implementing our multiple viewpoint digital video capture system. Regardless of the specific design option, the cameras used for the system will be capturing video at either a ‘low resolution’ of 640x480 and a frame rate of 30 fps or a ‘high resolution’ of 1024x768 and a frame rate of 15 fps. Furthermore the system will guarantee that no video frames are dropped by any of the capturing cameras or their host PCs.

Combining our 4 existing Point Grey Scorpion SCOR-13SM cameras with the Point Grey Flea cameras we have ordered, our synchronised multiple camera system will be designed according to one of the following options:

1. **External Trigger** — The inter-camera synchronisation will be performed to exact accuracy using an external trigger circuit we have built. Each individual camera we use will require it’s own FireWire bus. A maximum of 3 FireWire buses and hence 3 cameras will be able to be controlled by any one host PC.

2. **Point Grey Sync Unit** — The inter-camera synchronisation will be performed to an accuracy of 125 μs using Point Grey Sync Units. Each FireWire bus will require its own Sync Unit and will be able to handle 2 cameras capturing at high resolution or 3 cameras capturing at low resolution. Once again, a maximum of 3 FireWire buses will be able to be controlled by any one host PC, allowing for up to 9 cameras to be attached to a given host PC.

3. **Ethernet Trigger** — The host PCs will be interconnected by an Ethernet network. No special hardware will be required since the inter-camera synchronisation will be performed by a software-based trigger broadcast to all the host PCs over the network. However, the accuracy of synchronisation for this method will only be guaranteed in the order of 1–2 ms. As with the previous method, each FireWire bus will be able to handle 2 cameras capturing at high resolution and 3 cameras at low resolution, and again each host PC will be capable of controlling 3 FireWire buses and hence up to 9 cameras.

Note finally that for all options, if any more than one FireWire bus per host PC is to be implemented, a striped RAID volume is the recommended storage system for that PC. This is to ensure it's ability to write out the video streams it is receiving to disk in a timely manner without dropping any frames.

Each of these options has its advantages and disadvantages in terms of synchronisation accuracy, implementation cost, and ease of use. We will further consider each of the options in light of budget constraints and hardware availability in the near future as we progress towards completing the multiple viewpoint video capture system.

Bibliography

[1] D. Anderson, *FireWire Systems Architecture: IEEE 1394a*, 2nd ed. Reading, Mass.: Harlow : Addison-Wesley, 1999.

[2] D. Deeths and G. Brunette, "Using NTP to Control and Synchronize System Clocks - Part 1: Introduction to NTP," July 2001. [Online]. Available: http://www.sun.com/blueprints/0701/NTP.pdf

[3] F. Dierks, "Analog Goes Digital," 2002. [Online]. Available: http://www.baslerweb.com/popups/888/Analog_Goes_Digital.pdf

[4] *PGR FlyCapture Single-Lens Digital Video Camera System User Manual and API Reference*, Point Grey Research, Oct. 2004.

[5] *PGR IEEE-1394 Digital Camera Register Reference*, Point Grey Research, Sept. 2004.

[6] *PGR Scorpion Technical Reference Manual*, Point Grey Research, Sept. 2004.

[7] P. K. Rai, K. Tiwari, P. Guha, and A. Mukerjee, "A Cost-Effective Multiple Camera Vision System using FireWire Cameras and Software Synchronization," in *10th International Conference on High Performance Computing*, Hyderabad, India, Dec. 2003.

[8] G. Wideman, "Precision Timing Under Windows Operating Systems," Mar. 2000. [Online]. Available: http://www.wideman-one.com/gw/tech/dataacq/wintiming.htm

[9] Wikipedia, the free encyclopaedia, "Network Time Protocol," 2004. [Online]. Available: http://en.wikipedia.org/wiki/Network_Time_Protocol