Synchronization of Video in Distributed Computing Systems

E.G.T. Jaspers[a], B.S. Visser[b] and P.H.N. de With[c]

[a]Philips Research, Eindhoven, The Netherlands
[b]University of Twente, Enschede, The Netherlands
[c]University of Mannheim, Mannheim, Germany

ABSTRACT

A distributed multimedia computing system consists of sources, processing units and presentation devices in which each component operates autonomously (see Figure 1). This independency can be exploited for optimization of individual component performance. Flexibility in performance improvement is enhanced by using independent clock domains. This paper presents a Video I/O model for such a multimedia system. This model provides an asynchronous communication interface for independent clock domains with the ability to synchronize a video display to one of the video sources. The communication interface has been used for the design of I/O modules in a multimedia system, which are briefly outlined.

Keywords: multimedia computing, robust synchronization, data-driven processing, video interfacing, asynchronous communication

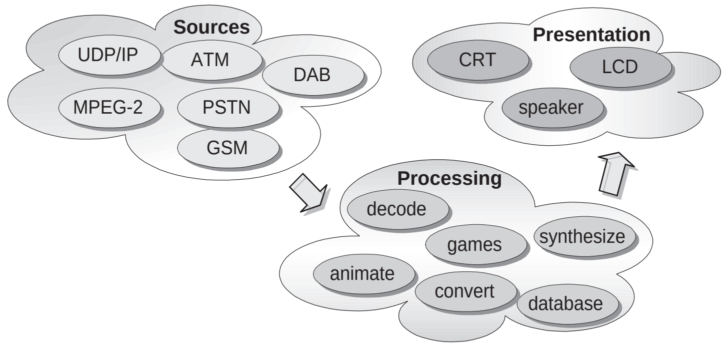

Figure 1. Independent domains in a distributed system.

1. INTRODUCTION

The MPEG standard has been developed to process multimedia information from various systems (TV, computer, telecom) in a more uniform way. In new extensions, such as MPEG-4 and MPEG-7, scenes are composed from a plurality of video objects. These new standards increasingly enable multiple independent source which are transmitted, composed and presented simultaneously. The presentation can be synchronized with one of the available sources.

The advantage of autonomous subsystems is readily understood from the broadcasting communication model, where a broadcaster (source) optimizes its performance, whereas the receiver is interested in low power and costs. Another advantage of independency is that it allows performance improvements of subsystems over time (scalability, e.g. more processing power due to a higher local clock frequency), without changing the overall system. Let us now consider the consequences of the described approach.

If the video display (presentation device) is independent from the source signals, the display is completely asynchronous. Similarly, the processing of the multimedia information is preferably independent of both the source(s) and the presentation device(s), and has its own separate processing and clock domains.

Email correspondence:
E.G.T. Jaspers: E-mail: egbert.jaspers@philips.com;
B.S. Visser: E-mail: bvisser@natlab.research.philips.com;
P.H.N de With: E-mail: dewith@ti.uni-mannheim.de

2. PROBLEM STATEMENT

The scheme in Figure 1 suggests that at least three different independent domains are used with communication between them. Video communication between the domains is traditionally stream-oriented, but video signals are intrinsically divided into frames and video lines, which can be regarded as data packets. These packets can be used to synchronize video communication between the various domains. The level of synchronization depends on the granularity of these data packets. For example, synchronization on video sample level requires accurate information about the start of a video sample, whereas for synchronization on video-line level (or frame), accurate time information about start of a video line (or frame) is required by means of Horizontal (or Vertical) synchronization signals. From now on, the rising edges of these H- and V-signals are referred to as pulses. Let us consider the following requirements for synchronization of video on a display. These requirements are constrained by the display, hence we have assumed that the video signals are adapted to the used display.

- On average, the frames of a source should be displayed at the correct time, although some jitter in the frame rate is allowed.
- The n-th pixel of each video line should form one straight vertical column on the screen.
- The number of pixels and lines of the video images should match with the display window which depends on the display type, e.g. for a CRT, the line rate and scan speed has to be known, for an LCD the resolution.

The aforementioned system aspects give direct requirements for the processing subsystem. Firstly, since independent clock domains are used also for communication, synchronization between the display and (one of) the sources should at least be provided at frame level. Secondly, a requirement for the processing subsystem is that the remaining processing bandwidth (if there is any) should become available for other tasks, or the system should switch in an idle mode. This enables scalability of resources and minimizes the power consumption per processing cycle. This goal is accomplished using data-driven communication which is discussed in the next subsection.

Our objective in this paper is to define an input and output module for video communication that fits in the distributed computing system and complies with the aforementioned requirements.

3. ASYNCHRONOUS COMMUNICATION

In this section, we discuss in more detail how communication between processing subsystems, having different clock domains, is established. For this purpose, we apply the Kahn[1] network model and explain rules for data-driven communication. For our distributed multimedia system, the following aspects of real-time video processing applications have to be considered.

- Video applications require high-speed computation on vast amounts of data. Thus, high parallelism in computing and communication is essential.
- The rate of making control decisions is much lower than the pixel processing rate. This means that a large amount of data is communicated and processed before the next control decision is made.
- The individual components in Figure 1, can have their own clock domain and their own latency. As a result, parallel operations cannot be scheduled at compile-time, because the timing relation is *a priori* unknown.

A data-driven communication model proves to be suitable for these aspects. An example of a video processing application using the data-driven model for asynchronous communication between separate clock domains is described by de With *et al.*[2].

The Kahn process network model[1] is an attractive dynamic model for data-driven communication. The model describes a unidirectional asynchronous communication network that is modeled as a directed graph. Each process is concurrent and is modeled as a node in the directed graph. These concurrent processes communicate asynchronously through unidirectional *edges*, for example queues of unlimited capacity (writes are never blocked). However, reads on input queues can block the corresponding process if no data is available.

A video application can be easily mapped onto the Kahn communication model. The data-flow graph, consisting of a set of independent video processing tasks is mapped onto the nodes of a directed graph. Video communication is asynchronous and is based on packets. These video packets are queued on the edges and are received in the same order as they were sent. In hardware, this queuing scheme can be implemented with FIFO buffers (see Figure 2). Alternative models for data-flow communication have been reported[3], but they do not support the system aspects as discussed earlier.

Implementation of the Kahn model in a real video system requires some adaptation of the communication rules. Read operations, which empty a FIFO, are no problem. However, unconstrained write operations (called non-blocking), which fill a FIFO, must be restricted, because FIFOs of unlimited capacity do not exist. Therefore, in hardware only so-called blocking write operations on a FIFO of limited capacity can be implemented. Blocking write operations can be implemented by suspending the internal clock of the video task. This modification does not violate the validity of the Kahn model. More details are explained by Leijten[4].

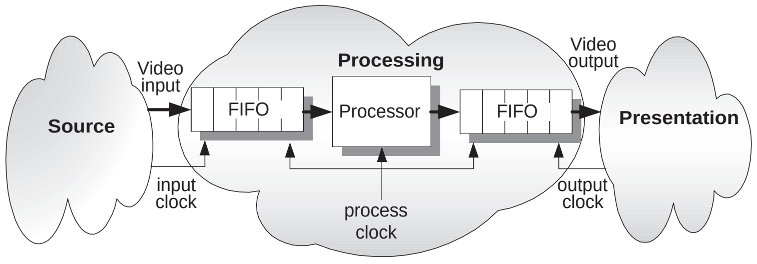

Figure 2. FIFO's for asynchronous communication.

In the sequel, we define a Video Input module (VIM) and Output Module (VOM) that can be applied in the modified Kahn model. Both VIM and VOM provide the interface between the data-driven processing and the real-time environment (e.g. camera or a display). When going from one domain to the other, timing information can be lost when no special precautions are taken, so that there is no relation between the pixel position and time. If the picture format is fixed (pixels $\times$ lines) and no pixels are lost during processing, the pixel position can be recovered by simply counting of the pixels, provided that the start of a frame is known (or indicated). This will be addressed in the next section. A picture format of fixed size is also referred to as "standard" signals.

Summarizing, a processing component in the distributed system based on asynchronous communication, has a fixed frame size and synchronization is established at frame level. On a local time basis, pixels are communicated asynchronously.

Due to the real-time behaviour of practical I/O devices, a second modification of the Kahn communication model for the VIM and the VOM is required. This modification is further supported by the following constraints and system considerations.

- A real-time video source will continuously produce samples and a video display will continuously consume samples.
- The timing information of a real-time source has to be recovered at the presentation side (for display).
- The sources and presentation devices can be unreliable. For example, the timing between successive frames in a video sequence can vary due to signal deterioration.
- The sources and presentation devices require different communication protocols (e.g. CCIR-656).

We have adopted the following measures to make the VIM and VOM robust for the presented constraints. Let us first address the design of the VIM.

The tasks successive to the VIM expect a fixed frame size. An overflow at the input FIFO of the VIM (due to too high input rate or a too low processing rate), is simply cut-off so that information loss cannot be avoided. The VIM will complete the video frame with dummy (e.g. value zero) samples (see next section). For the VOM similar measures are taken. If the display consumes samples too fast (the display expects video samples and no samples are provided), the VOM generates dummy video samples (e.g. black pixels) for its display. Finally, a separate synchronization V-pulse from the VIM to the VOM is communicated to recover the exact start time of the frames. (see Figure 7) Thus, synchronization between the input and output is provided at frame level, whereas the pixel and line rates are derived from an externally supplied clock.

4. SYNCHRONIZATION

This section explains how synchronization signals are used to divide a video stream into video frames and video lines. The timing relation between synchronization signals and a video stream needs to be maintained carefully to guarantee a stable video display without introducing artifacts. As we already explained in Section 3, timing information can be easily lost when a data stream crosses different communication domains. We propose an efficient method that maintains the timing information, when data streams cross from a real-time environment to a data-driven environment and back.

Using a fixed-sized picture format (pixels × lines), is the most straightforward approach to recover a unique pixel position. Now, counting pixels is sufficient to determine the exact position of a pixel. It is possible to extract a fixed-sized picture format from a video stream, because only part of a video signal is used for visible video display. This part is also called the active video part. Almost 20% of the pixels on a video line and 10% of the video lines in a video frame is not displayed and is called the blanking. For example, a CRT display uses this blanking time to reposition the electron beam to the start of the next video line. Of course, only the active part of a video stream is of interest for the processing units. It is even possible to select a smaller part of the active video for processing when a programmable capture window is used, e.g. to zoom out part of the video. Figure 3 shows a part of the active video signal that is captured for processing. The left-hand side of the figure shows the capturing in case of progressive video, whereas the right-hand side shows that for interlaced video. Section 4.2.2 explains in more detail the position of the H- and V-pulse for interlaced video. In the remainder of this paper, we assume that the complete active video

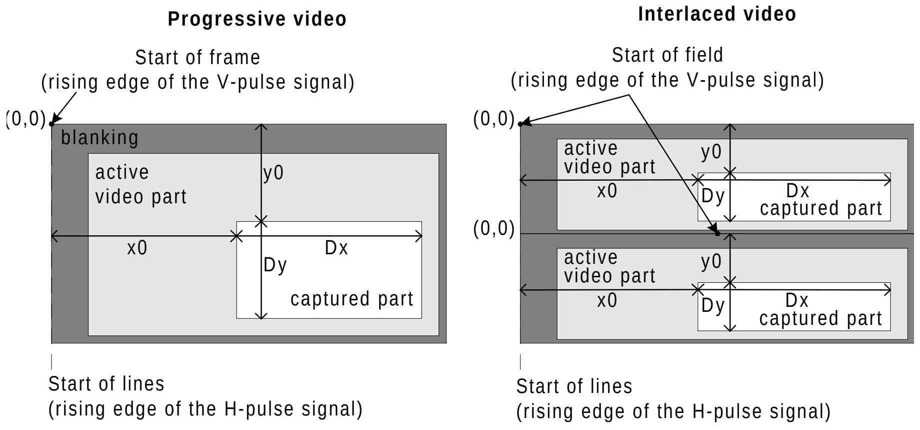

Figure 3. The active and captured part of a video signal.

part is captured. If in the sequel the active video part is indicated, it represents the captured window part. The blanking is referred to as the remaining part. Therefore, the active part clarified in Figure 4, has the dimension of $Dx \times Dy$. In the horizontal direction, the active part of a video line starts $x0$ pixels after an H-pulse. In vertical direction, the active part starts $y0$ lines after a V-pulse. Because a fixed-sized picture format is required, $x0 + Dx$ and $y0 + Dy$ (see figure) are constant. The remaining pixels of the video lines and the remaining video lines of the frame may vary. Only the fixed-sized active part is conveyed through the processing systems. The format may be regarded as the format of a standard signal (for a standard signal also time-stability constraints are involved).

Obviously, the Video Input Module (VIM) has to retrieve the fixed-sized active part ($Dx \times Dy$) from the video

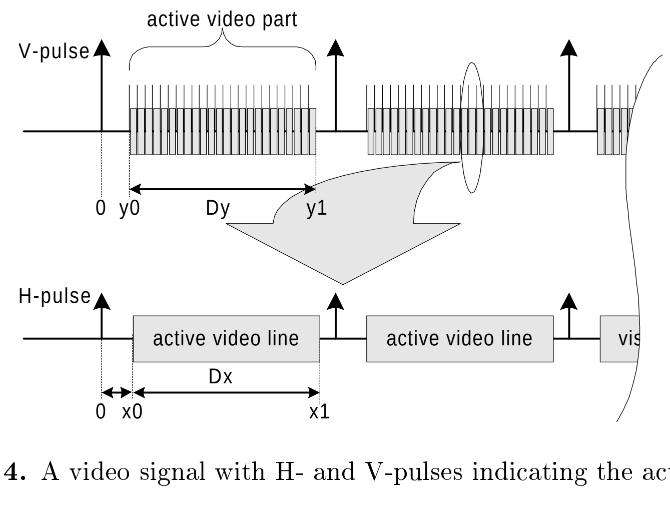

Figure 4. A video signal with H- and V-pulses indicating the active part.

stream in all circumstances. Section 4.1 explains how this is accomplished. The task of the Video Output Module (VOM) is to extend the active part with the blanking part again, such that synchronization between the VIM and the VOM can be maintained. This is elaborated in Section 4.2. Section 4.3 explains how a composition of video coming from multiple sources can be displayed on the same presentation device.

4.1. The Video Input Module

As mentioned before, the Video Input Module has to provide a fixed-size video signal to the processing units, which can be regarded as standard signal. In practice, non-standard video signals, which have a variable number of pixels per line and variable number of lines per frame, occur regularly. Possible causes for the variability are:

- change of source by the broadcaster or the processing system, e.g. selecting another channel in a TV receiver;
- noise in a distribution channel, e.g. due to bad-weather conditions, H- and V-pulse detection may be distorted;
- sources with a "non-standard" signal, for example a video recorder in trick-play mode.

Once the VIM is programmed for a certain video format, it supplies the processing units with a fixed-size video signal under all conditions. Consequently, the synchronization signals of the input signals may have to be adapted to achieve this goal while maintaining optimal video quality. To do this, the VIM uses a H- and V-pulse regenerator. The next two sections discuss these topics.

4.1.1. Frame synchronization

V-pulses indicate the start of a frame and may vary during the vertical blanking time. Normally a V-pulse is accepted if it appears during the blanking at a constant rate. However, input V-pulses may occur during the active part or at varying rate. The VIM will try to correct such irregularities. Figure 5 visualizes this process. The following list summarizes the irregularities.

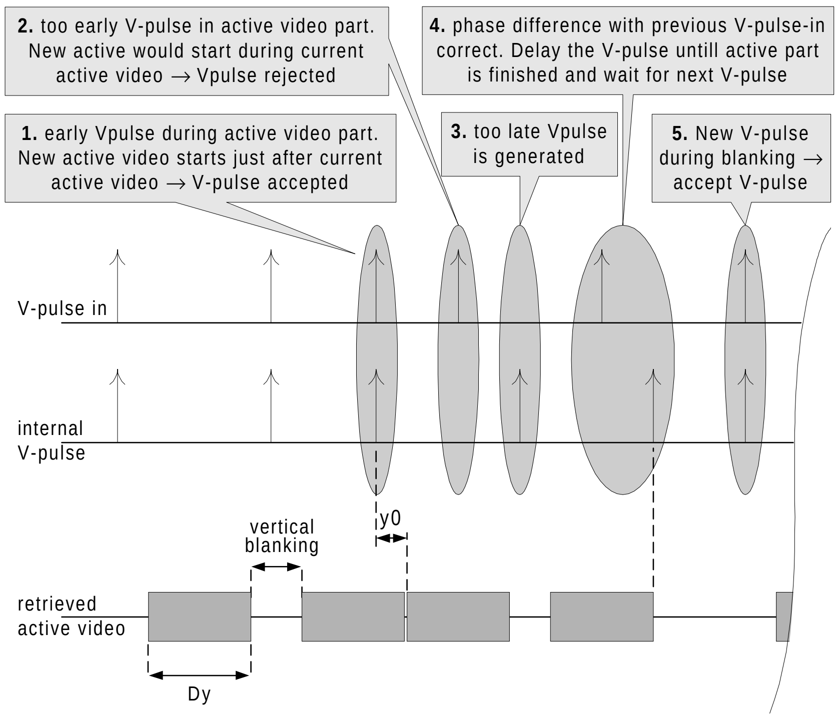
2. too early V-pulse in active video part. New active would start during current active video → Vpulse rejected
4. phase difference with previous V-pulse-in correct. Delay the V-pulse untill active part is finished and wait for next V-pulse
1. early Vpulse during active video part. New active video starts just after current active video → V-pulse accepted
3. too late Vpulse is generated
5. New V-pulse during blanking → accept V-pulse
V-pulse in
internal V-pulse
vertical blanking
y0
retrieved active video
Dy

Figure 5. V-pulse regeneration by the VIM.

1. **Early pulses**. Pulses that would interrupt an active video part cannot be accepted, because retrieval of the fixed-sized video part has to be finished. The active video part starts *y0* number of lines after a V-pulse. Thus if this V-pulse occurs less than *y0* number of lines earlier than the end of the preceding active video frame, the V-pulse can still be accepted (see case 1 in Figure 5). Ultimately, the active video starts immediately after finishing the preceding active video frame.

2. **Too early pulses**. A V-pulse that occurs even more than *y0* number of lines earlier in the active video part cannot be accepted and is called a too early pulse.
3. **Missing or too late pulses**. When a V-pulse is not received, a predetermined time after it is expected, a V-pulse is generated (see case 3). When the input of the VIM does not receive any video signal, the V-pulse signal is regenerated and the display can still be used to present some information (for example a warning message).
4. **Too large phase shifts**. An incoming V-pulse is accepted when it occurs during the blanking period and when the phase with respect to the previous input V-pulse is correct. Suppose that a V-pulse is deleted because of a too early V-pulse (case 2) and the next V-pulse occurs at the correct phase. As shown in case 4 of the figure, this might occur during the active video part. Apparently, the too early V-pulse that was deleted in case 2, was caused by a phase shift in the input signal. Since the current active video part has to be finished, the regenerated internal V-pulse starts directly after the active video. Immediately, conveying a new active video part would result in a too early V-pulse for the next V-pulse, because the remaining time is too small. Therefore, the VIM will not convey an active video part, but will go idle until a new input pulse occurs (case 5). Consequently, one field or frame is skipped at the input and has to be regenerated at the end of the processing chain (the VOM). Due to the fixed-size constraint, this is the only solution to correct the phase shift as fast as possible.
5. **Correct V-pulse**. This pulse occurs during the vertical blanking and at the correct phase.

4.1.2. Line synchronization

H-pulses indicate the start of a video line. Similarly to the V-pulse, variations in the H-pulse signal cause problems for maintaining a fixed-sized picture format. Figure 6 shows the simplified algorithm for line synchronization. Normally, the active part of a video line starts *x0* pixels after the occurrence of an H-pulse. If an H-pulse occurs during the active part of a video line, the fixed number of pixels *Dx* of this video line are captured first. This too early H-pulse is ignored and line synchronization is lost. Furthermore, if H-pulse are missing or occur to late, line synchronization is also lost. In both cases, the VIM switches to an asynchronous mode in which it generates its own H-pulses. Since fast recovery of the line synchronization is less critical than frame synchronization, the algorithm for H-pulse regeneration is less complex. All the H-pulses occurring during the visible part of a video line are rejected. The VIM only recovers line synchronization if an H-pulse occurs during the line blanking, but before the VIM regenerates the H-pulse itself.

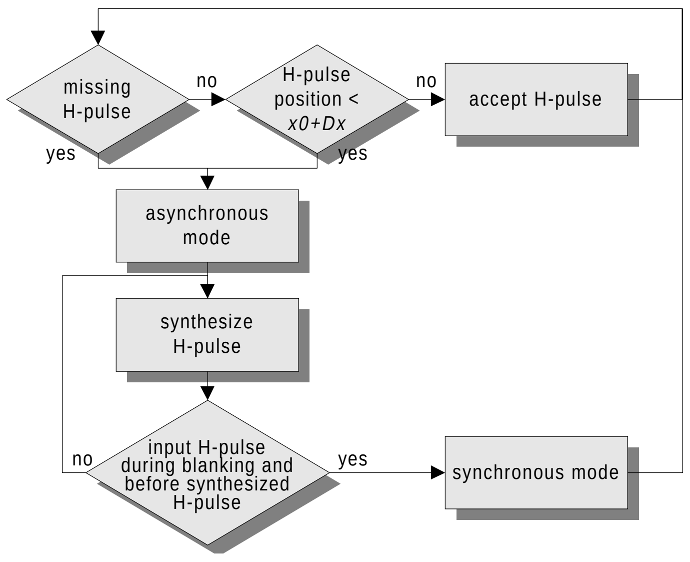

Figure 6. Flow chart of the line synchronization.

4.2. The Video Output Module

The main task of the VIM is to convert a video signal from the real-time environment to a signal that is suitable for a data-driven communication domain. Similarly, the VOM is responsible for making a domain transition in the opposite direction. Timing information and synchronization signals for the presentation device have to be recovered and synchronization with an input video signal of a VIM should possibly be maintained if appropriate. The following subsections describe the solution for this process.

4.2.1. Frame synchronization

Frame synchronization for the Video Output Module (VOM) is different from the frame synchronization techniques used in the Video Input Module (VIM), because the VOM has to take additional constraints into account. Due to data-driven communication, the VOM has to read all active pixels from the processing units. No pixels may be skipped, for example, to start the next frame before the active pixels of the current frame are completely read.

A frame memory which contains the conflicting frame, can be flushed to solve this problem. A V-pulse can never be ignored as the VIM would do in case of ”too early V-pulses”. Normally, the frame memory only acts as a FIFO where read and write pointers are synchronized and cannot have collisions. In case of a ”too early V-pulse”, the read and write pointers in the memory are decoupled and can have collisions. Now, the frame of the unacceptable V-pulse can be flushed, at the cost of limited visual artifacts. After receiving the next V-pulse, the FIFO mode of the frame memory is restored.

The VIM can also send a V-pulse to the VOM without capturing any video in case of ”too large phase shifts”. Consequently, the VOM cannot consume pixels from the processing units to reconstruct the frame; it would cause a buffer underflow. Instead, the VOM generates dummy pixels (e.g. black pixels) to fill the missing active part. The VIM informs the VOM about such an event through a separate synchronization channel. The V-pulse together with a valid signal for the V-pulse, is communicated over this channel. Figure 7 shows a high-level block diagram of the VIM, VOM, processing units and their synchronization signals.

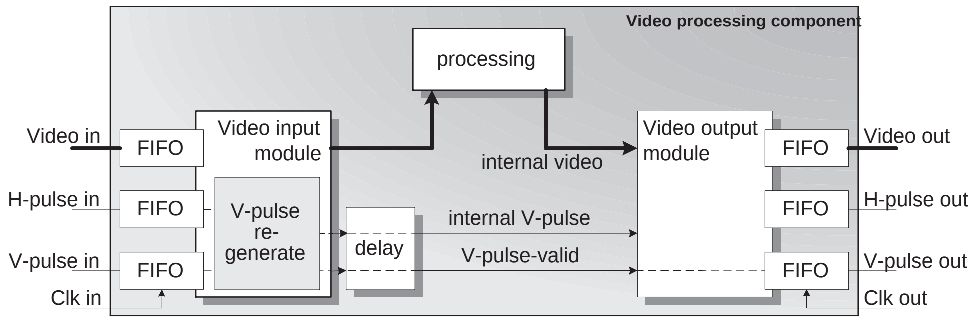

Figure 7. Block diagram of processing component.

4.2.2. Interlaced video

With interlaced video, first the odd lines are displayed and subsequently the even lines. These odd and even lines should be nicely interleaved on the display, such that the distance between all lines are equal. This is accomplished by a correct position of the V-pulse with respect to the H-pulse.

The top of Figure 8 shows that interleaving the V-pulse position at 0 (for the odd lines) and 0.5 (for the even lines) times the line time, results in a perfectly interlaced picture display. In the bottom part of the figure, the V-pulse for the even lines starts at 0.25 times the line time, which results in a non-uniform distribution of the video lines on the display.

Since the V-pulse is synchronized with the input signal and the H-pulse with the external display, the timing position can be incorrect. When the VOM receives a V-pulse from the VIM, the current frame is processed until the correct position of the output V-pulse is reached. At that position, the pulse is regenerated, thereby starting the new field. The determination of the correct position is measured at the VIM.

4.3. Multiple video sources

The previous sections explained that the VOM synchronizes to a video signal retrieved by the VIM, but some applications require an asynchronous display of the video signal. Video PC cards displaying a 50 Hz PAL or 60 Hz NTSC signal onto a 72 Hz monitor are an example of such an application. Furthermore, when an output signal consists of several input sources, the output can only be synchronized with one of its sources. All other sources have

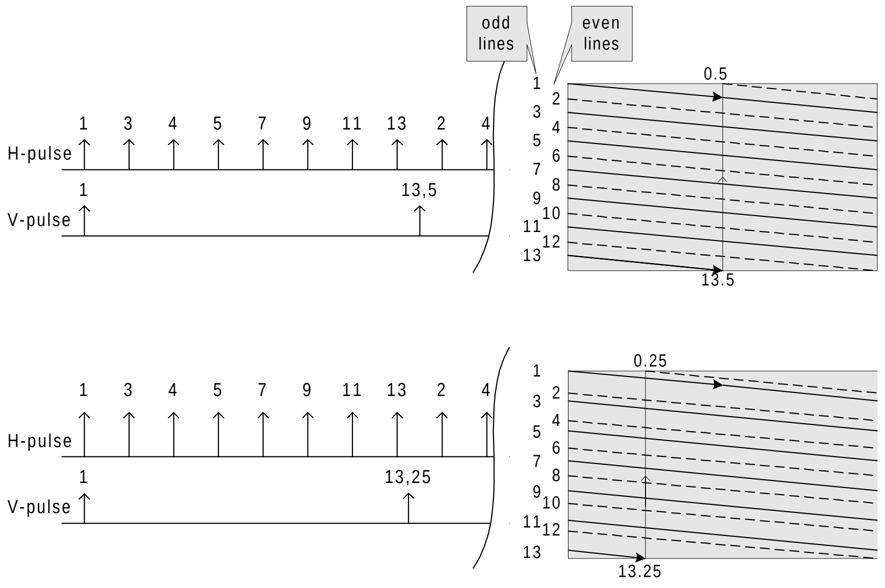

Figure 8. V-pulse position with respect to the H-pulse.

to be displayed asynchronously. Figure 12 shows an application that consists of two video broadcasts, an internet connection and a graphics-based wallpaper background. The memory that is used to compose the video output signal can also be used to provide the asynchronous communication. Each source uses one write pointer which addresses independently the shared memory together with one pointer for reading the composed output. These pointers can collide and overtake each other. Figure 9 shows how the pointers overtake each other for asynchronous display. When the pointers collide within the active part of a video stream, the resulting output picture consists of frames originating from different temporal moments. A horizontal border separating these frames can become visible, particularly when a video stream contains fast motion. Because pointers collide due to their difference in speed, the border scrolls over the screen at a rate equal to this speed difference. If the output video signal is synchronized with one of the sources, the corresponding read and write pointer have the same speed and overtake each other only during the blanking period.

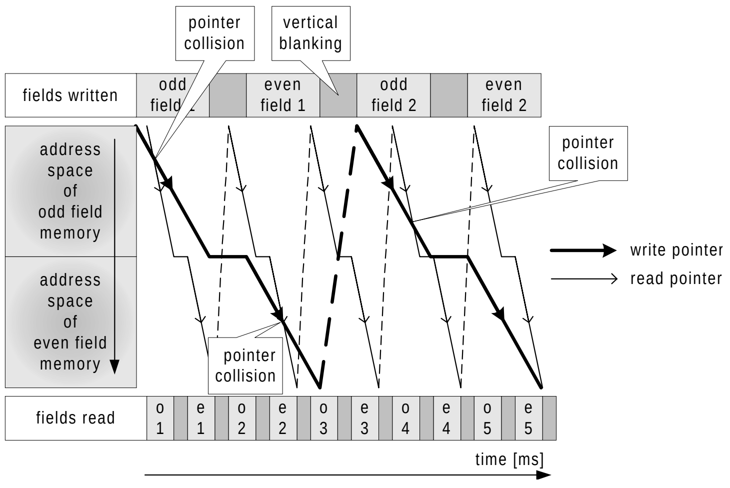

Figure 9. The read/write pointer behaviour for asynchronous output.

5. IMPLEMENTATION DETAILS

This section reveals aspects about the hardware implementation of the Video Input Module (VIM) and Video Output Module (VOM). Section 5.1 explains the architecture in which the VIM and VOM are implemented and how both interact with the processing elements and the external environment. Section 5.2 discusses the independent clock domains and the way asynchronous communication with the external environment is realized. Finally, Section 5.3 clarifies the need for a delay unit in the synchronization channel between the VIM and VOM. This delay is needed to provide a constant latency of processing system, even when the applications change dynamically.

5.1. Hardware Architecture

As illustrated in Figure 12, a video output stream can consist of multiple independently operating video input streams. However, the output stream can only be synchronized with a single input stream (see section 4.3). The processing system proposed by Jaspers *et al.*[5] supports the input of three independently operating sources and up to two presentation devices.

Parallelism for multiple video streams can be obtained by higher processing speeds or by parallel hardware. We have adopted a solution in which the video stream of each source and each presentation device is processed by a repeated implementation of the VIM and VOM units. The units operate independently of each other, have their own memory space and work at a 64 MHz clock speed. Consequently, video streams up to 64 MHz can be processed, although 54 MHz is generally sufficient for resolutions up to VGA resolution.

The motivation for using separate implementations of VIM and VOM is as follows. Let us suppose that a single implementation is used that can handle multiple streams. Such a unit would require multi-tasking capabilities and an additional scheduler for dynamically scheduling of real-time processing tasks (i.e. video streams). Moreover, local memory is required to store the state when a task switch is performed. However, the implementation of such unit would have to run at a multiple of the 64 MHz clock, because in the worst case, three input streams and two output streams at high pixel rate have to be processed simultaneously. This high clock speed results in a high power consumption and would require additional pipelining delays for high speed processing. Separate implementations of the VIM and the VOM give more flexibility in handling different processing capabilities per video stream and it allows efficient power consumption management.

To separate the functionality of the VIM and VOM units from the remaining processing units in the data-driven system, they are contained in a shell; one shell for all VIM units and one shell for the VOM units. The shell takes care of fetching parameters to program the functionality of the VIM and VOM and, in case of the VIM, also of sending features of the received video streams to the remaining processing system and the VOM. For example, the VIM measures the picture resolution and the position of a V-pulse relative to a H-pulse. Subsequently, the shell sends an interrupt to a global controller which forwards these parameters to the appropriate tasks.

Figure 10 depicts a high-level schematic of the VIMs, the VOMs, their shells and the interconnections. As can be noticed, both shells communicate through a communication network. The network enables connections of the VIMs and VOMs to the other processing units. Connections via this network are established via small FIFO buffers. The VIM, VOM and processing units communicate asynchronously through these buffers (see Section 3). The signal flow-graph through the VIMs, VOMs and processing units are defined by the application and are programmed by means of a task graph. Finally, Figure 10 shows that one of the VOM units has a blender. This blender mixes video streams and graphics, such as on-screen-display, into a single RGB video stream. The figure also shows multiplexers in the VIM shell which are used for selecting the input signal to which the output stream will be synchronized. They convey a frame synchronization signal via the synchronization channel mentioned in Section 4.2.1.

5.2. Independent Clock Domains

Each VIM and VOM unit operates on an independent 64 MHz clock and a separate video clock signal. The internal 64 MHz clock signal can be switched off to disable a VIM or VOM unit and to save power. The video clock signal is offered by the external real-time environment. The VIM uses this signal to determine the rate by which a video stream is produced. The VOM uses this signal to determine the rate by which a video stream is consumed. The parameters of both VIM and VOM are adapted to these clock rates to ensure that the VIM produces a fixed-sized picture at regular intervals for the processing units and that the VOM consumes enough data to prevent a buffer overflow.

Problems may arise when the video clock signal is interrupted. A missing video clock for the VIM hampers the reception of the video and the synchronization signals. When the VOM misses the external video clock, no video and

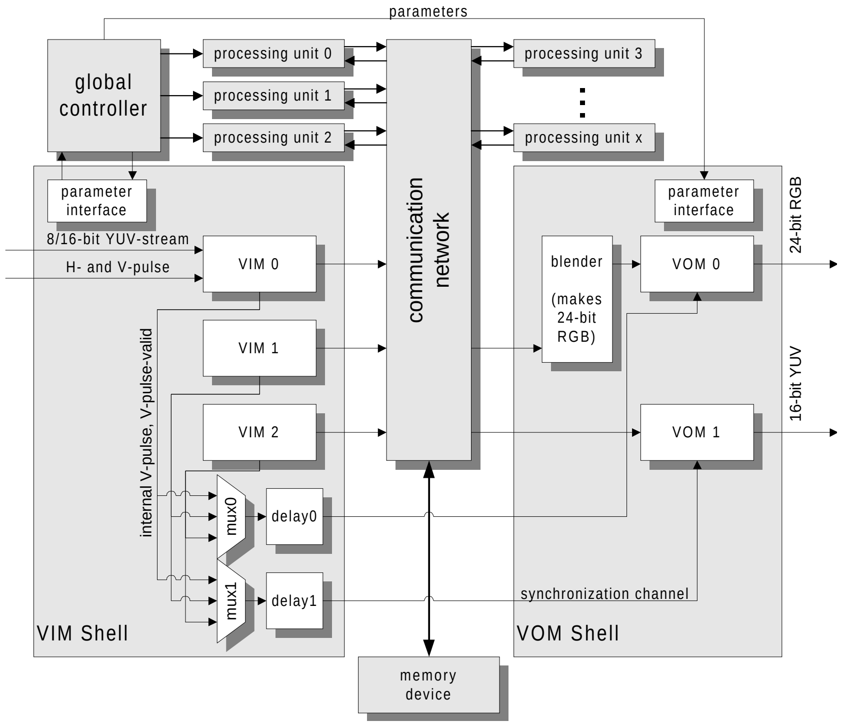

Figure 10. High-level schematic of the hardware architecture.

synchronization signals are available for the display. Therefore, we introduce a watchdog for the video clocks that ensures insertion of a synthesized clock signal when the external video clock is not present. When a predetermined number of 64 MHz clock cycles occur between two successive video clock cycles, insertion of the synthesized clock is enabled.

5.3. Latency of the processing system

The moment that the VOM receives a V-pulse from the VIM, determines when active video pixels from the processing system are used to reconstruct the frames. As a result, the amount of data that is buffered into the processing system depends on the time a V-pulse is received by the VOM. A small latency of the processing units or a large delay of the V-pulse from the VIM to the VOM will require a large data buffer, whereas a large latency or a small delay of the V-pulse requires little buffering. Obviously, the latency of the total processing system should be constant to provide a regular video stream for the display. Because the processing may vary over time, also the latency of the processing changes. For example, the user may activate a vertical sampling-rate converter to do aspect-ratio conversion in a widescreen TV set. This will increase the latency of the processing chain depending on the scaling factor, the order of the filter and the position of the video window to be scaled. The variation in the latency has to be compensated by buffering. With a delay unit for the internal V-pulse signal (see Figure 7), the exact time the VOM receives the V-pulse can be set such that for a maximum latency application, the buffer requirement is minimal. The application with the minimum latency will then determine the maximum buffering for compensation that should be available. This is illustrated in Figure 11

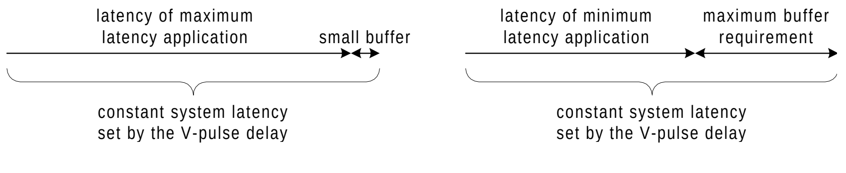
latency of maximum latency application
small buffer
constant system latency set by the V-pulse delay
latency of minimum latency application
maximum buffer requirement
constant system latency set by the V-pulse delay

Figure 11. Relation between the processing latency and the V-pulse delay.

6. FEATURES

The combination of video input module (VIM) and video output module (VOM) has a powerful robust synchronization mechanism that copes with signal irregularities of the external environment. Furthermore, the modules support several formats and are highly parameterizable. This is illustrated by the following features.

- **Format.** The VIM accepts YUV (chrominance / luminance) video streams and enables communication of both 8-bit multiplexed (U-Y-V-Y) and 16-bit (UY-VY) video streams. The output formats of the VOM are similar to the VIM formats, but it supports also RGB-based video streams.
- **Synchronization.** The VIM accepts 3 different synchronization mechanisms: according to CCIR-656, synchronization on H-/V-pulses and synchronization on blanking identifier signals.
- **H-, V-pulse regeneration.** See Section 4.
- **Window capture.** Defines the video area to be processed. The VIM and VOM are able to handle any image resolution up to High Definition (HD). Sizes of the input and output format can be modified by e.g. a scaler.
- **Resolution Recognition.** The VIM measures the picture size of the incoming video stream and the interlace factor and communicates it to the rest of the processing unit. This can be exploited for automatic adaptation to the input signal by means of a controller that can re-program the processing system when a new format is detected.

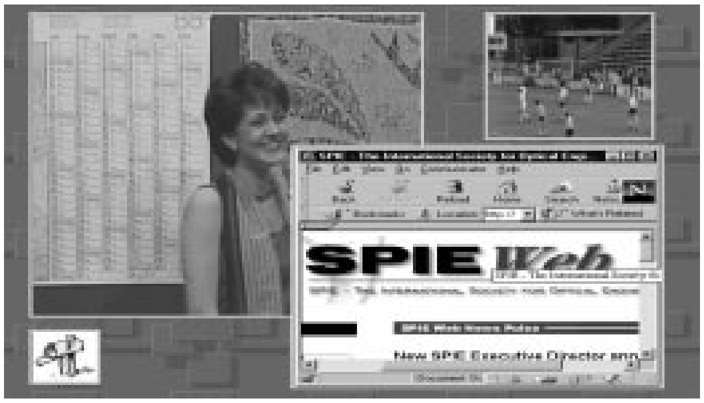

Figure 12. A video composition from four input sources.

7. CONCLUSION

We have proposed a Video I/O module featuring asynchronous communication of video, which enables synchronization of the display to one of multiple sources. The I/O modules split the internal and external clock domains and convert "non-standard" signals to frame-synchronous "standard" signals. The presented I/O model allows processing units to be individually optimized for speed, complexity, memory resources, etc. The concept of such a Video I/O module has been evaluated in an experimental chip-set[2,5], where it was generalized with interface protocols such as CCIR-656. Figure 12 shows an application produced with the chip-set. Several distributed sources such as two video broadcasts, internet and a graphics-based wallpaper background are processed, composed and subsequently presented on a display device.

REFERENCES

1. G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing 74* , pp. 471–475, Aug. 1974.
2. P.H.N. de With, E.G.T. Jaspers *et al.*, "A video display processing platform for future TV concepts," *IEEE Trans. Cons. Electr.* **45**, pp. 1230–1240, Nov. 1999.
3. D.D. Gajski *et al.*, "A second opinion on data flow machines and languages," *IEEE Computer* **15**, pp. 58–69, Feb. 1982.
4. J.A.J. Leijten, J.L. van Meerbergen, *et al.*, "Stream communication between real-time tasks in a high-performance multiprocessor," in *Proceedings of DATE '98*, pp. 125–131, February 1998.
5. E.G.T. Jaspers, P.H.N. de With and J.G.W.M. Janssen, "A flexible heterogeneous video processor system for TV applications," *IEEE Trans. Cons. Electr.* **45**, pp. 1–12, Febr. 1999.